van Basic tot Machinetaal

Opbouwwerk voor
Commodore 64
gebruikers met Basic
Machinetaal, Graphics,
Geluid en alle hulptalen.

8

Andere talen

Inhoud

8/1 Simons'Basic[1)]
8/2 Logo[1)]
8/3 Pascal[1)]
8/4 C[1)]
8/5 FORTH[1)]
8/6 COMAL[1)]
8/7 Superbase[1)]
8/8 PILOT[1)]
8/9 BASIC 7.0[1)]

1) Dit hoofdstuk heeft een eigen inhoudsopgave.

De Commodore 64 is in vele opzichten een uitstekende computer. Het geluid is onovertroffen en de grafische mogelijkheden kennen hun gelijke niet, althans in deze prijsklasse.
Maar de ingebouwde BASIC 2.0 is een aanfluiting. Dat fraaie geluid en dat prima beeld laten zich slechts met vele PEEKs en POKEs aansturen, zodatvelen er maar van afzien. Om op de 64 echt iets leuk te doen met audio en video is het eigenlijk noodzakelijk om in machinetaal te gaan programmeren. al die superspelen voor de 64 zijn er niet voor niets in geschreven.

Vandaar dan ook dat ML programmeren uitgebreid aan de orde komt in dit boek, hoofdstuk 9 is er helemaal aan gewjd. Toch zijn er gelukkig ook andere mogelijkheden te bedenken. Er zijn meer keuzes in programmeertalen dan alleen BASIC 2.0 of ML.

Zo kan men voor SIMONS' BASIC kiezen, een BASIC die allerlei extra mogelijkheden toevoegt aan BASIC 2.0. Vooral voor geluid en graphics biedt SIMONS' BASIC veel mogelijkheden. Maar wat ook kan is een heel andere taal kiezen. Basic is niet de meest geschikte taal voor sommige toepassingen en zeker niet de beste taal om goed te leren programmeren. Daarvoor kan beter een gestructureerde taal als PASCAL of zelfs ADA gekozen worden.

Voor kinderen zijn er weer andere mogelijkheden. LOGO bijvoorbeeld, een taal die door kinderen heel snel geleerd wordt, niet in de laatste plaats door de eenvoudig te begrijpen tekencomando's die deze taal kenmerken. Dat maakt programmeren heel aanschouwelijk.

Dan bestaat er nog een taal als FORTH, niet echt makkelijk om te leren, maar ijzersterk. FORTH staat heel dicht bij de logica die de computer intern hanteert en kan eventueel als overstap naar ML gebruikt worden. Daardoor is het ook een heel snelle taal, waarin spelletjes geschreven kunnen worden die niet echt onderdoen voor in ML geschreven spellen.

Of om er nog een te noemen, COMAL, een taal die sterk op BASIC lijkt maar allerlei voordelen biedt die eigenlijk in de gestructureerde talen thuishoren.
COMAL combineert de vrijheid van BASIC met het gemak van PASCAL.

Inderdaad, mogelijkheden genoeg. De Commodore 64 heeft van huis uit een tamelijk beperkte BASIC meegekregen, maar de machine is zo opgebouwd dat we zonder veel problemen die BASIC kunnen uitschakelen en door een andere, sterkere taal kunnen vervangen.

In dit boek beginnen we met SIMONS' BASIC, de meest verbreide tweede taal op de 64, maar daar zal het zeer zeker niet bij blijven.

8/1.1

Inleiding

De Commodore 64 heeft aantrekkelijke eigenschappen, o.a. de grafische mogelijkheden en de drie toongeneratoren waarmee driestemmige melodieën uitgevoerd kunnen worden.

Een nadeel is, dat het BASIC 2.0, waarover de Commodore 64 beschikt, geen mogelijkheden biedt voor het maken van tekeningen of het samenstellen van melodieën, tenzij met een grote hoeveelheid PEEK en POKE commando's. POKE 53281,15 is een commando, dat de beginner niets zegt: een lijst met adressen - een zogenaamde memory map - alleen brengt uitkomst. Hier ligt het grote struikelblok voor veel Commodore 64 eigenaars.

Een jonge Engelsman, David Simons, vond dat blijkbaar ook. Op 15-jarige leeftijd stelde hij een hulpprogramma samen, dat het maken van tekeningen, sprites, muziek en nog andere zaken vereenvoudigt. Naar de samensteller wordt dit programma: SIMONS' BASIC genoemd. Hoewel, programma, het is in feite een nieuwe programmeertaal.

Het wordt geleverd als cartridge, het is dus onmiddellijk klaar voor gebruik. Ook wordt er een handleiding in boekvorm bij geleverd. Wie de Engelse taal niet beheerst staat voor een probleem, dat we in dit boek zullen oplossen. Ook zullen we allerlei fouten in deze handleiding aangeven, vooral de eerste drukken waren alles behalve foutloos.

Soms denkt men dat Simons' BASIC in de plaats komt van het gewone BASIC van de Commodore 64. Dit is niet zo: alle bestaande BASIC commando's blijven gehandhaafd, de Simons' BASIC commando's komen daar bij. Alleen wordt er 8K geheugen door Simons' BASIC in beslag genomen, maar de resterende 30 Kbytes zijn ruim voldoende, zeker met alle extra commando's die Simons' BASIC ons biedt.

Voor de Commodore 64 wordt ingeschakeld moet eerst de cartridge in de gleuf achterin worden gestoken, als we dit doen terwijl de computer aanstaat kunnen we de computer beschadigen.
Op het scherm komt dan na inschakelen:
EXPANDED CBM V2 BASIC
30719 BYTES FREE

Dit staat op een witte achtergrond met blauwe rand.
De 114 commando's van Simons' BASIC kunnen als volgt worden ingedeeld.
1) Commando's voor het maken van tekeningen;
2) Commando's voor het maken van muziek;
3) Commando's voor het maken van spri-

1.1 Inleiding

tes en speciale letter- of grafische tekens;
4) Commando's om het programmeren te vereenvoudigen;
5) Commando's om een goede schermindeling te maken, het scherm te kleuren, teksten te plaatsen;
6) Commando's om een goed gestructureerd programma te maken;
7) Commando's om joysticks, paddles, een printer of een disk-drive te gebruiken;
8) Commando's met extra reken mogelijkheden, zoals het omrekenen van hexadecimale naar decimale getallen, binaire getallen naar decimale;
9) Commando's om fouten op te sporen, regels te nummeren of opnieuw te nummeren.

De grootste moeilijkheden bij de Commodore 64 ondervindt de gebruiker bij het maken van tekeningen. Daarom worden allereerst de commando's besproken die het maken van tekeningen vergemakkelijken. De gewone BASIC commando's zijn voorlopig voldoende om een programma voor het maken van zulke tekeningen samen te stellen.

Maar voor alles moeten we eens kijken naar de handleiding, zoals die bij Simon's BASIC wordt geleverd. De fouten hierin maken het gebruik ervan vooral voor de beginner wel erg lastig.

8/1.2

Fouten in de handleiding bij Simons' BASIC

Helaas moet geconstateerd worden dat er in deze handleiding - zoals in veel boeken en tijdschriften op computergebied - nog al eens fouten te vinden zijn. Vooral de oudere drukken vertonen fouten, zodat enkele programma's niet goed lopen. Als U daarmee geconfronteerd wordt, kan deze foutenlijst U helpen. De programma's op blz.13-3 en volgende geven ook vaak problemen.

Jammer genoeg zit in de oudere cartridges een 'bug': LOW COL werkt niet met REC en BLOCK en ook dat heeft invloed op programma's met deze commando's.

Blz. 1-6 CONVENTIONS, punt 6:
Daar staat: Met uitzondering van het commando FIND (zie hoofdstuk 2.10) moeten alle commando's van Simons' BASIC van de daarachter komende gegevens worden gescheiden door een spatie. Dit blijkt niet zo te zijn: bij de volgende commando's blijkt de spatie overbodig:
ARC ANGL BLOCK CIRCLE COLOUR HIRES LINE LOW COL PAINT PLOT TEST.
Er zullen nog wel meer voorbeelden te vinden zijn.

Blz. 2-7 Het programma RESET.
U kunt bij een paar namen vreemde tekens op het scherm krijgen.
Als U TERRIER vervangt door POODLE
STARLING door FALCON
en TROUT door SHARK
dan gaat alles goed.

Blz. 2-11 Het getal 30 achter DISPLAY moet weg. (In de nieuwste drukken is dit al verbeterd.)

Blz. 2-12 De twee regels met ACTION doorstrepen. Na de tweede regel met ACTION toevoegen: TRACE 0
(In de nieuwste drukken verbeterd.)

Blz. 2-13
Achter regel 30 toevoegen: <RETURN>
(In de nieuwste drukken verbeterd.)

Blz. 3-4 Achter het woord IS in regel 75 een spatie toevoegen. (In de laatste drukken is dit al gedaan).

Blz. 3-6 Achter AT geen spatie gebruiken, anders volgt de foutmelding:
?BAD SUBSCRIPT ERROR IN ... of er wordt een 0 voor de string gedrukt.

Blz. 3-7 De woorden 'two' en 'three' in de tweede regel van boven omwisselen: three lines below and two characters... (In laatste drukken OK.)

Blz. 3-8 5 regels van onder: CURSOR LEFT moet zijn: CURSOR RIGHT. In

1.2 Fouten in de handleiding bij Simons' BASIC

dezelfde regel: 'Numeric' veranderen in 'Alphanumeric'.

Blz. 4-3 Het programma PRINT%.
Het grootste binaire getal dat kan worden omgerekend naar decimaal is: 11111111 (acht enen). Als een groter getal wordt ingetypt worden alleen de eerste 8 bits omgerekend.

Blz. 4-3 Het programma PRINT$.
Achter PRINT$ moeten altijd 4 (vier) hexadecimale getallen staan. Kleinere getallen aanvullen tot 4 cijfers met nullen er voor.
Dus: PRINT$00C4 PRINT$000F PRINT$0A4C
Anders verschijnt de foutmelding: ?NOT HEX CHARACTER.

Blz. 4-4 Programma EXOR.
In regel 5 de ' ' vervangen door „ „ (In laatste drukken OK.)

Blz. 6-2 COMMODORE 64 COLOURS.
Vervang 9 door BROWN
10 door LIGHT RED
11 door GRAY 1
12 door GRAY 2
15 door GRAY 3
(In laatste drukken OK.)

Blz. 6-3 Programma COLOUR.
Achter FORMAT: COLOUR sc,bo moet zijn: COLOUR bo,sc

Blz. 6-4 9 regels van boven.
Vervang COLOUR 3,6 door COLOUR 6,3

Blz. 6-5 Programma MULTI.
7 regels van onder: vervang de woorden 'red' en 'cyan' door resp. 'black' en 'red'. (Laatste drukken OK.)

Blz. 6-7 en 6-8
Met oudere cartridges geven de programma's met LOW COL niet het gewenste resultaat. De 'bug' is niet te herstellen. Om toch na LOW COL andere kleuren te zien, zouden de volgende programma's kunnen worden gebruikt:

voor blz. 6-7 boven:

```
10 HIRES 0,1
20 REC 20,20,60,60,1
30 LOW COL 0,7,0
40 PAINT 25,50,2
50 PAUSE 5
60 NRM
```

Voor blz. 6-7 midden:

```
10 HIRES 0,1:MULTI 2,3,6:Z=10
20 FOR X = 1 TO 3
30 REC 10,Z,30,30,X
40 PAINT 20,Z+10,X
50 Z = Z +40:NEXT
60 LOW COL 4,5,7:Z=10
70 FOR X = 1 TO 3
80 REC 50,Z,30,30,X
90 PAINT 60,Z + 10,X
100 Z = Z + 40:NEXT
110 PAUSE 5
120 NRM
```

1.2 Fouten in de handleiding bij Simons' BASIC

```
10 HIRES 0,1:MULTI 2,3,6:Z=10
20 FOR X = 1 TO 3
30 REC 10,Z,30,30,X
40 PAINT 20,Z+10,X
50 Z = Z +40:NEXT
60 LOW COL 4,5,7:Z=10
70 FOR X = 1 TO 3
80 REC 50,Z,30,30,X
90 PAINT 60,Z + 10,X
100 Z = Z + 40:NEXT
110 PAUSE 5
120 NRM
```

Dit programma geeft zes rechthoeken, elk met een andere kleur.
Voor blz. 6-8 boven:

```
10   HIRES 0,1:MULTI 2,3,6:Z=10
20   FOR Y=10 TO 50 STEP 40
30   FOR X = 1 TO 3
40   REC Y,Z,30,30,X
50   PAINT Y+10,Z+10,X
60   Z=Z+40:NEXTX:Z=10:LOW COL
     4,5,7:NEXT Y
70   HI COL
80   FOR X = 1 TO 3
90   REC Y,Z,30,30,X
100 PAINT Y+10,Z+10,X
110 Z=Z+40:NEXT
120 PAUSE 5
130 NRM
```

Negen rechthoeken, 3 in de eerste kleuren, 3 in de tweede en 3 weer in de eerste kleuren.

Blz. 6-8 Programma PLOT
Verander in regel 20 het getal 160 in 80. (In nieuwe drukken al verbeterd).

blz. 6-9 boven.
Vergeet de . voor de 5 niet in regel 20. Zonder die punt loopt het programma wel, maar de curve bestaat uit veel minder punten.

Blz. 6-11 Het programma ARC.
De parameters sa (start arc) en ea (end arc) geven de beginhoek en de eindhoek aan van de boog. Ligt het begin van de boog bovenaan, dan is sa 0 en ea b.v. 60. Ligt het eind van de boog bovenaan, dan is sa b.v. 315 en ea = 360 (en niet 0).

Blz. 6-12 Programma ANGLE onderaan. Vergeet in regel 50 de - niet voor het getal 5. Die - staat onduidelijk gedrukt. Dat in FOR X = 170 TO 0 STEP -5 een - moet worden gebruikt spreekt eigenlijk vanzelf.

Blz. 6-17 Regel 25 van het programma moet zijn:

```
25 C = INT(90 * RND (1))
      + 2 :......
```

Voeg de volgende regel toe aan het programma:

```
37 NEXT I
```

(Beide fouten in de nieuwste drukken al verbeterd.)

Blz. 6-18 In het programma veranderen:

```
50 CHAR I * 8,A,I + J *40,1,2
60 NEXT : A=A+15:NEXT
```

Als in regel 30 wordt gezet:

```
30 FOR J = 0 TO 12
```

dan begint de reeks tekens op het scherm bij het begin.

Blz. 7-2 Programma BCKGNDS.

1.2 Fouten in de handleiding bij Simons' BASIC

Neem in regel 20:

```
20 BCKGNDS 1,3,5,7  OF
```

of,

```
20 BCKGNDS 1,3,5,15
```

De letters zijn dan wat beter zichtbaar.

Blz. 7-5 Programma FCHR.
In sommige handleidingen is regel 15 weggevallen. Die regel alsnog intypen:

```
15 FCOL 0,0,10,10,0
```

Blz. 7-6 programma FCOL.
Ook hier is regel 15 uitgevallen. Typ in:

```
15 FCOL 12,15,10,10,0
```

Blz. 7-7 programma MOVE.
Vervang regel 50 door:

```
50 PRINT "<CTRL 5> OF THE SCREEN
```

De letters zijn dan beter zichtbaar.
In regel 60 moet tussen de " " een spatie worden getypt.
Blz. 7-8 programma INV.
regel 50 tussen de " " een spatie typen.
regel 60 moet zijn:

```
60 INV 0,0,19,4
```

Blz. 7-9 achter FORMAT:
Daar staat direction W,sr, sc.....
en direction B,sr,sc.....
De komma achter W en B moet weg.

Blz. 8-4, MOBS
Onder aan blz. 8-2 staat: Als het MEM commando al gebruikt is, dan zijn alleen de blokken 192 t/m 255 beschikbaar voor MOB data.
Het is gebleken, dat de ze bloknummers ALTIJD moeten worden gebruikt. Als regel 90 wordt getypt als aangegeven (DESIGN 0,2048) dan werkt het program niet. De computer kan zelfs op hol slaan zodat LIST, RUN enz. niet meer werken. Dit zelfde geldt voor regel 320 (DESIGN 1,2112). Dat komt doordat in regels 700 en 710 op blz. 8-6 de bloknummers 32 en 33 worden gebruikt.

```
700 MOB SET 0,192,0,1,0
710 MOB SET 1,193,2,0,1
```

Het beginadres voor de MOB data wordt berekend door het bloknummer te vemenigvuldigen met 64.
Beginadres blok 192 is: 192 ∗ 64 = 12288
Beginadres blok 193 is: 193 ∗ 64 = 12352
Deze adressen moeten in regels 90 en 320 worden gebruikt. Breng dus de volgende veranderingen aan:
Blz. 8-4

```
90 DESIGN 0,12288
```

Blz. 8-5

```
320 DESIGN 1,12352
```

Blz. 8-6

```
700 MOB SET 0,192,0,1,0
710 MOB SET 1,193,2,0,1
```

Na deze veranderingen werkt het programma wel.
Blz. 8-4, voor de regels 100 t/m 300 moet een @ teken staan:
ASCIs-code= CHR$(64)
Blz. 8-5, voor de regels 400 t/m 600 moet eveneens een @ teken worden gezet.

1.2 Fouten in de handleiding bij Simons' BASIC

Blz. 8-6, 7 regels van onder; wijzig de tekst in:
... the MOB is coloured black and passes under all screen data.
Blz. 8-7, 6 regels van boven:
... The parameters x1 and y1 are ...
Blz. 8-13, in de regels 30 t/m 90 een @ teken zetten voor de eerste . of B.
Blz. 9-2, programma onder aan de bladzijde.
In regel 20 moet achter REPEAT een : staan. (Nieuwe drukken OK)
Om de letters A t/m G af te drukken moet aan het eind van regel 20 staan:

```
20 .........:UNTIL A>71
```

Blz. 9-3, achter het woord STRANGER in regel 30 moet een spatie komen.
Blz. 9-4, in oudere drukken staat in vette letters ... MORE THAN NINE NESTED LOOPS.
NINE veranderen in FIVE.
Blz. 9-4 programma LOOP .. EXIT IF .. END LOOP
Bij het intikken van de letters NIET op RETURN drukken. Dus achter C, B en S moet ‹RETURN› vervallen.
Blz. 9-5, verander in de WARNING het woord NINE door FIVE.
Blz. 9-6, programma CALL
In oudere drukken moeten de volgende regels als aangegeven worden verbeterd of toegevoegd:

```
45 IF X>10 AND X<16 THEN DIM A$(X)
50 IF X<16 THEN CALL ENTER NAMES
55 GOTO 10
```

Blz. 9-7, programma EXEC
Verander – indien nodig – de volgende regels:
110 PRINT"‹SHIFT/CLR HOME› ‹CRSR DOWN›"
120 FOR I = 1 to X:PRINT TAB(20)A$(I):NEXT
1040 T=0:FOR I = 1 to X – M

Blz. 9-9 GLOBAL
Het volgende programma laat de werking van GLOBAL en LOCAL duidelijker zien:

```
10 REM *** VOORBEELD VAN
   LOCAL/GLOBAL ***
20 PRINT"<SHIFT/CLR HOME>"
30 A$="BEGINWAARDEN":A%=123:
   A=456.7
40 PRINT A$,A%,A
50 PAUSE 5
60 LOCAL A$,A%,A
70 A$="NIEUWE WAARDEN!":
   A%=789:A=321.4
80 PRINT"<CRSR DOWN><CRSR DOWN>"
90 PRINT A$,A%,A
100 PAUSE 5
110 GLOBAL
120 PRINT"<CRSR DOWN><CRSR DOWN>"
130 PRINT A$,A%,A
```

Blz. 12-6, JOY STICK PROGRAMMA
Voeg aan regel 180 toe:

```
180 IF JOY=128 THEN TEXT27,170,
    "WELL DONE PICASSO",1,3,16:
    PAUSE 2:GOTO 340
```

Regel 190 moet vervallen.
Denk om de spatie tussen " " in regel 380.

8/1.3

Tekenen

Simons' BASIC voorziet in het maken van tekeningen op twee manieren:
1) HIGH RESOLUTION tekeningen, waarbij een scherm van 320 bij 200 punten (pixels) kan worden gebruikt. Het aantal kleuren en het gebruik daarvan is beperkt.
2) MULTI COLOUR tekeningen, waarbij een scherm van 160 bij 200 punten (pixels) kan worden gebruikt, met een groter aantal kleuren en minder beperkingen in het gebruik daarvan. Om een scherm van 320 bij 200 punten te vullen moet elk van die punten programmeerbaar zijn. Voor die 64000 punten zijn 64000 bits nodig, 8 Kb.

Het gewone schermgeheugen, dat zich bevindt op de adressen 1024 t/m 2023 en dus 1Kbyte adresruimte omvat, is daarvoor veel te klein. Daarom wordt het 'puntengeheugen' elders ondergebracht, het oude schermgeheugen dient nu als kleurgeheugen.
Voor 8 Kbytes punten is dus maar 1 Kbyte kleur beschikbaar, dat kan problemen opleveren. Daarover later meer.

Het maken van een tekening op een scherm met 64000 punten is een onderneming, die enige zorg vereist.
We maken eerst een ontwerp van de gewenste tekening op een werkblad. Daarna begint het programmeren van zo'n tekening. Een model werkblad is bijgevoegd achteraan dit hoofdstuk.
Het wordt gemaakt op ruitjespapier met ruitjes van 5 bij 5 mm, multo map formaat is daarvoor zeer geschikt: de tekeningen kunnen dan goed bewaard blijven.

Het scherm wordt voorgesteld door een rechthoek van 20 cm breed en 12.5 cm hoog, we krijgen zo 1000 ruitjes. (Dit heeft inderdaad te maken met het 1Kbyte grote kleurgeheugen).
Bij de bovenrand staan de nummers 0, 16, 32, 48, 64 .... 288, 304, 319 en bij de zijrand de nummers 16, 32, 48, 64 .... 176, 192, 199.

Bij computers telt de 0 heel vaak mee en dat is bij de nummers van het werkblad ook het geval. De nummers lopen dus niet van 1-320 (1-200) maar van 0-319 (0-199). Deze getallen dienen als x en y coördinaten van elk punt. De x en de y alsmede het woord 'coördinaten' zijn afkomstig uit de wiskunde: x is de horizontale as, y de vertikale en coördinaten zijn getallen die aangeven waar een punt precies op zo'n as ligt. Met twee getallen, resp. voor x en voor y is elk punt op het werkblad vastgelegd.
De linkerbovenhoek heeft als coördinaten 0,0, de rechter benedenhoek ligt op 319,199. Het midden van het scherm wordt vaak gesteld op 160,100.

1.3 Tekenen

Het is de gewoonte om eerst de horizontale (x) coördinaat te vermelden en dan pas de vertikale (y).
Nog een praktische wenk: maak de ontwerptekening met een potlood. Fouten kunnen dan gemakkelijk met een gum worden verbeterd.

Dan begint nu het bespreken van de tekencommando's van Simons' BASIC.

Overzicht van de gebruikte termen:
I. TEKST SCHERM - hierop komen teksten en tekeningen met de bekende grafische tekens (zie elders). U ziet dit scherm zodra U de Commodore 64 inschakelt;
TEKEN SCHERM - speciaal voor de tekeningen met Simons' BASIC commando's. Om dit scherm te krijgen is het commando HIRES vereist (zie verder).
II. DIRECT gebruik van commando's - een commando wordt zonder regelnummer op het scherm gezet. Na het indrukken van de RETURN-toets wordt dat commando uitgevoerd.
Voorbeeld: PRINT"Commodore" + <RETURN>
HIRES0,3:PAUSE2 + <RETURN>
INDIRECT gebruik van commando's - het gebruik van commando's in een programma.
Er staat dan altijd een regelnummer voor.
Voorbeeld:

```
10 HIRES 5,13
20 PAUSE 3
30 NRM
```

III. PLOT TYPE - Bij veel commando's speelt 'plot type' een belangrijke rol. Vandaar dat dit hier uitgebreid wordt besproken zodat later naar dit deel verwezen kan worden om niet steeds in herhalingen te vallen.

We moeten onderscheid maken tussen PLOT TYPE bij HIGH RESOLUTION en bij MULTI COLOUR toepassingen.

A) De plot types bij HIGH RESOLUTION (320 bij 200 pixels):
Plot type 0 - een figuur wordt uitgewist.
Plot type 1 - een figuur wordt getekend.
Plot type 2 - een figuur dat al op het scherm staat wordt uitgewist. Is het scherm leeg, dan wordt de figuur daar getekend.

B) De plot types bij MULTI COLOUR (160 bij 200 pixels)
Plot type 0 - een figuur wordt uitgewist;
Plot type 1 - een figuur wordt getekend in de 1e kleur van het commando MULTI of LOW COL;
Plot type 2 - een figuur wordt getekend in de 2e kleur van het commando MULTI of LOW COL;
Plot type 3 - een figuur wordt getekend in de 3e kleur van het commando MULTI of LOW COL;
Plot type 4 - kleuren worden verwisseld:
een figuur met plot type 0 krijgt plot type 3;
een figuur met plot type 1 krijgt plot type 2;
een figuur met plot type 2 krijgt plot type 1;
een figuur met plot type 3 krijgt plot type 0.

Om dit alles te verduidelijken dient het

1.3 Tekenen

volgende programma ingetikt te worden:

```
10 HIRES 0,3: MULTI 2,4,5
20 LINE 0,0,150,180,0
30 PAUSE 2
40 LINE 0,0,150,180,4
50 PAUSE 2
60 NRM
```

Run dit programma en de functie van de 4 in regel 40 (4=plot type) is op het scherm te zien.
Verander dan de laatste 0 van regel 20 achtereenvolgens in 1, 2 en 3 en laat telkens het programma lopen. U ziet dan wat plot type 4 inhoudt.

IV. Verdere commando's - Van de commando's die in het vervolg worden besproken wordt steeds een overzicht gegeven, dat er als volgt uitziet:

COMMANDO - geeft aan uit welke letters het commando bestaat.
OMSCHRIJVING - hier wordt aangegeven wat het commando tot gevolg heeft.
GEBRUIK - hier vindt U welke getallen of letters er achter een commando dienen te komen. Zulke getallen en letters heten met een vreemd woord: parameters.
Ook worden de parameters vermeld zoals ze in de handleiding van Simons' BASIC staan, meestal volgt een cijfervoorbeeld.
OPMERKINGEN - wat U van elk commando dient te weten.
BIJZONDERHEDEN - worden indien nodig hier vermeld.

DE EERSTE COMMANDO'S.
We beginnen met een commando dat duidelijk maakt, hoe goed Simons' BASIC werkt. Om te tekenen moet het tekstscherm worden veranderd in een tekenscherm. Wie de Programmer's Reference Guide voor de Commodore 64 heeft doorgewerkt, zal ongetwijfeld het programma, dat deze verandering mogelijk maakt, zijn tegengekomen en gemerkt hebben, dat het een langdurige geschiedenis is.

Gelukkig beschikken we nu over het
COMMANDO : HIRES
OMSCHRIJVING : maakt het scherm gereed voor tekeningen van 320 bij 200 pixels. Bovendien wordt de kleur van de tekening en van de achtergrond bepaald.
GEBRUIK : HIRES tekenkleur, achtergrondkleur
HIRES pc,sb
HIRES 0,3
OPMERKINGEN: De cijfers achter HIRES kunnen liggen tussen 0 en 15.
Niet alle kleurcombinaties geven een goed resultaat. Zie het overzicht van te gebruiken kleuren voor tekst en achtergrond elders in dit boek.
BIJZONDERHEDEN: LOW COL werkt ook na HIRES, het derde cijfer moet wel getypt worden, doch telt niet mee.
HIRES kan direct en indirect worden gebruikt.
Een spatie tussen HIRES en het volgende getal is niet absoluut vereist, hoewel in enkele handleidingen het tegendeel wordt beweerd.

COMMANDO : PAUSE
OMSCHRIJVING: stopt het programma gedurende een bepaalde tijd.
GEBRUIK : PAUSE aantal seconden
PAUSE „string", aantal seconden
PAUSE „message", number of seconds
OPMERKINGEN: De string hierboven genoemd, kan een mededeling zijn,b.v.:
380 PAUSE „Druk op spatiebalk",6
120 PAUSE 5 (de string kan worden weggelaten).
BIJZONDERHEDEN: PAUSE vervangt

1.3 Tekenen

de tijdlus FOR T = 1 TO 3000 : NEXT
PAUSE kan zowel direct als indirect worden gebruikt.

Een heel kort programma:

```
10 HIRES 0,3
20 PAUSE 5
```

Na RUN en <RETURN> krijgt het scherm de kleur cyaan, die 5 seconden blijft. Dan wordt teruggeschakeld naar het tekstscherm.

COMMANDO: NRM
OMSCHRIJVING: het tekenscherm wordt uitgeschakeld en het tekstscherm is weer beschikbaar
GEBRUIK: NRM (geen parameters)
OPMERKINGEN: NRM wordt gebruikt aan het eind van een tekenprogramma.
COMMANDO: PLOT
OMSCHRIJVING: zet een punt op het tekenscherm of wist een punt uit.
GEBRUIK: PLOT x coordinaat, y coordinaat, plot type
PLOT x, y, plot type
PLOT 125,100,1
OPMERKINGEN: In high resolution geldt: x tussen 0 en 319, y tussen 0 en 199. In multicolour geldt: x tussen 0 en 159, y tussen 0 en 199.
BIJZONDERHEDEN: PLOT kan ook direct worden gebruikt. Typ: HIRES0,3:PLOT160,100,1:PAUSE2:NRM

Enkele toepassingen.
STERRENBEELDEN.
Zie werkblad 1. (Bijgevoegd achteraan dit hoofdstuk)

```
10 HIRES 1,6
20 PLOT160,32,1:PLOT144,32,1:
   PLOT128,40,1:PLOT128,56,1
30 PLOT115,56,1:PLOT128,72,1:
   PLOT116,72,1:PLOT168,96,1
40 PLOT96,98,1:PLOT112,102,1:
   PLOT128,105,1:PLOT80,112,1
50 PLOT174,128,1:PLOT136,128,1
60 PAUSE 5
70 NRM
```

Na RUN en <RETURN> komen er twee sterrenbeelden op het scherm. Als je een lijn trekt door de achterste twee sterren van de 'Grote Beer' vind je de Poolster. Dat kan het programma laten zien, typ achter het programma 'Sterrenbeelden':

```
70  X=174
80  FOR Y=128 TO 32 STEP-3
90  PLOT X Y,1
100 X=X - 0.4
110 NEXT Y
120 PAUSE 5
130 NRM
```

EEN LIJN.

```
10 HIRES 0,3
20 FOR X = 0 TO 319
30 PLOT X,100,1
40 NEXT
50 PAUSE 5
60 NRM
```

Commentaar: Dit programma zet 319 punten zo dicht achter elkaar dat ze een lijn vormen.
Om een stippellijn te krijgen moet regel 20 van het programma veranderd worden in:

```
20 FOR X = 0 TO 319 STEP 5
```

STERRENHEMEL.

```
10 HIRES 1,0
20 TL = 0
30 X=INT(RND(.)*320):
   Y=INT(RND(.)*200)
40 PLOT X,Y,1
50 TL=TL + 1:IF TL > 100 THEN 70
60 GOTO 30
70 PAUSE 5
80 NRM
```

1.3 Tekenen

Commentaar:
TL in regels 20 en 50 is een teller, die er voor zorgt, dat het aantal sterren tot 100 beperkt blijft.
In regel 30 wordt RND(.) gebruikt, dat volgens een artikel in Compute's Gazette van dec. 1983 sneller werkt dan RND(1) of RND(0).

HEEN EN WEER.

```
10 HIRES 0,7
20 FOR X = 0 TO 319
30 PLOT X,100,1
40 NEXT
50 PAUSE 2
60 FOR X = 319 TO 0 STEP -1
70 PLOT X,100,0
80 NEXT
90 PAUSE 2
100 NRM
```

Commentaar:
Regels 20 t/m 40 trekken een lijn over het scherm. Na 2 seconden verdwijnt deze lijn weer.
Regel 70 zorgt daarvoor, omdat als plot type 0 wordt gebruikt.
Volgt nog een demonstratie van PLOT TYPE 2.

LIJNEN.

```
10 HIRES 0,3
20 FOR X = 0 TO 80:PLOT X,100,1:
   NEXT
30 FOR X = 160 TO 240:PLOTX,100,
   1:NEXT
40 FOR X = 0 TO 319:PLOTX,100,2:
   NEXT
50 PAUSE 2
60 GOTO 40
```

Commentaar:
Dit doorlopend programma kan onderbroken worden door op de toets RUN STOP te drukken.
In de regels 20 en 30 worden twee lijnstukken op het scherm gezet. Door PLOT TYPE 2 in regel 40 verdwijnen de lijnstukken die op het scherm staan, waar niets stond, komen nu lijnen.
KLEUREN.
De kleur van de rand van het scherm blijft blauw, maar dat kan worden veranderd met het
COMMANDO: COLOUR
OMSCHRIJVING: Dit commando bepaalt de kleuren van de rand en de achtergrond.
GEBRUIK: COLOUR kleur rand, kleur achtergrond
COLOUR bo,sc
COLOUR 3,4 (rand: cyaan en scherm: paars)
OPMERKINGEN: De cijfers achter COLOUR liggen tussen 0 en 15.
Deze cijfers corresponderen met de kleuren in de kleurentabel.
BIJZONDERHEDEN: Vergeet de U in COLOUR niet (Gebruik niet de Amerikaanse spelling COLOR.)
In oudere drukken van de handleiding zijn bo en sc achter COLOUR abusievelijk verwisseld.
COLOUR kan direct en indirect worden gebruikt.

KLEUREN VAN RAND EN BEELD 1.

```
10 PRINT"♥"
20 FOR X = 0 TO 15
30 FOR Y = 0 TO 15
40 COLOUR X,Y
50 PAUSE 1
60 NEXT Y
70 NEXT X
80 NRM
```

Commentaar:
regel 10 Alles wat op het scherm stond wordt verwijderd.
Alle mogelijke combinaties van rand- en schermkleuren worden getoond.

1.3 Tekenen

KLEUREN RAND EN BEELD 2.

```
10 HIRES 0,3
20 PAUSE 3
30 COLOUR 4,5
40 PAUSE 3
50 COLOUR 8,9
60 PAUSE 3
70 NRM
```

Commentaar:
Na RUN <RETURN> blijft het scherm cyaankleurig, alleen de rand verandert van blauw naar paars en dan naar oranje. Na HIRES kan door COLOUR alleen de kleur van de rand worden gewijzigd.
Typ nu RUN 30 <RETURN>.
Ook de kleur van het beeldscherm verandert: groen wordt bruin. COLOUR werkt volledig, omdat HIRES geen deel van het programma meer uitmaakt.

In de programma's zijn commando's uit het gewone BASIC en uit Simons' BASIC door elkaar gebruikt.
Het is mogelijk om na het commando LIST, dat het programma op het scherm brengt, duidelijk de Simons' BASIC commando's van de overige te onderscheiden. Dat gebeurt met het
COMMANDO: OPTION

Als er met hoeken wordt gewerkt, dan spelen de cosinus en sinus een rol.

Hoeken worden gemeten in graden. De boven- en zijkant van deze pagina vormen een rechte hoek. Zo'n hoek heeft 90 graden. Wie zich omdraait maakt een hoek van 180 graden en een hele draai is 360 graden.

Een andere hoekmaat is de 'radiaal'.
Kijk nu eens op het toetsenbord van de Commodore 64. Op de voorkant van de ↑-toets (naast RESTORE) staat een Griekse letter, de 'pi'.
Dit teken stelt een getal voor dat iets groter is dan 3. Wie wil weten hoe groot dat getal precies is, typt in: PRINT <SHIFT>↑ en op het scherm komt 3.14159265...
Dit beroemde getal speelt een grote rol bij cirkels. De omtrek van een cirkel is namelijk 2 ★ PI maal de straal van de cirkel.

Een radiaal is de middelpuntshoek, waarvan de boog gelijk is aan de straal van de cirkel. En omdat een cirkel 360 graden telt is 1 radiaal 360/2★PI = 180/PI graden. Dat is iets minder dan 60 graden. De Commodore 64 rekent dat nauwkeuriger uit.

Nu kan de sinus of de cosinus heel goed in een tekenprogramma worden gebruikt.
Typ het volgende programma in:
SINUS GRAFIEK.

```
10 HIRES 0,3
20 FOR X = 0 TO 319
30 PLOT X,100+SIN(X),1
40 NEXT
50 PAUSE 5
60 NRM
```

OMSCHRIJVING: Alle Simons' BASIC commando's worden met een gekleurde achtergrond getoond. (Engels: highlighted).
GEBRUIK: OPTION getal
OPTION n
OPMERKINGEN: Het getal n ligt tussen 0 en 255.
Als n = 10 dan treedt het commando OPTION in werking.
Is n <> 10 dan wordt de werking van OPTION uitgeschakeld.
BIJZONDERHEDEN: Het is aan te raden om OPTION niet te gebruiken wanneer een listing met een printer wordt

gemaakt. Het lint van de printer heeft dan te veel te lijden.
OPTION is uiteraard zeer geschikt voor direct gebruik.
Als er een programma in het geheugen van de Commodore 64 zit, typ dan - zonder regelnummer -:
OPTION 10 : LIST en druk op <RETURN>
De Simons' BASIC commando's zijn duidelijk zichtbaar in de listing.
Typ dan OPTION 7: LIST en <RETURN>.
De listing is weer gewoon.

IETS UIT DE WISKUNDE.
Hopelijk klinkt dat niet al te afschrikwekkend, maar een paar wiskundige begrippen komen hier goed van pas, met name sinus (SIN) en cosinus (COS).
Dat zijn getallen tussen 0 en 1. De Commodore 64 heeft deze getallen in de ROM opgeborgen.

Typ: PRINT SIN(1) of PRINT COS(.5) en <RETURN> en de betreffende waarden van de sinus of cosinus komen op het scherm.

Commentaar:
regel 30 100+SIN(X) wordt de Y coordinaat. Het getal 100 zorgt er voor, dat de tekening midden op het scherm komt. Het resultaat is pover, een golflijntje.
Dat komt omdat de sinus een getal is, dat hoogstens 1 kan zijn, de Y varieert tussen 99 en 101.
Daarom moet die waarde vergroot worden.
Verander regel 30 in:

```
30 PLOT X,100+(60*SIN(X)),1
```

Na RUN verschijnt er een sterrenhemel: de hoek X wordt te vaak herhaald, want die wordt in radialen gemeten.
Verander regel 30 nu in:

```
30 PLOT X,100+(60*SIN(X/20)),1
```

Op het scherm staat een mooie sinusgrafiek.
Dit programma kan ook gebruikt worden om een cosinus grafiek te krijgen. Het enige dat veranderd moet worden is SIN in regel 30, dat wordt COS.
De COS en SIN kunnen worden gebruikt om cirkels of ellipsen te tekenen.

CIRKEL.

```
10 HIRES 0,3
20 X=60:Y=60
30 FOR H=0 TO 2*PI
40 PLOT 160+X*COS(H),
   100+Y*SIN(H),1
50 NEXT
60 PAUSE 5
70 NRM
```

Commentaar:
regel 30: PI wordt verkregen door <SHIFT> ↑.
regel 40: De getallen 160 en 100 zorgen er voor, dat het middelpunt van de cirkel in het midden van het scherm ligt.
De 'cirkel' bestaat uit slechts zeven punten. Want 2★PI - dat is iets meer dan 6 - geeft, omdat het tellen bij 0 begint, slechts 7 punten.
Verander regel 30 in

```
30 FOR H = 0 TO 2*PI STEP .1
```

Commentaar:
Vergeet de punt (.) voor de 1 niet.
Na RUN <RETURN> blijkt de tekening iets beter te zijn geworden. Neem nu

```
30 FOR H = 0 TO 2*PI STEP .01
```

Dit geeft een goed resultaat.
Verander regel 20 in

```
20 X=80 : Y=50
```

1.3 Tekenen

en RUN het programma opnieuw.
Nu wordt de tekening een ellips.

Dit cirkel/ellips programma laat duidelijk zien welke mogelijkheden het PLOT commando kan geven. Bovendien komt het later, bij het bewegen van sprites over het scherm, weer van pas.

Voor het maken van tekeningen is het trekken en van lijnen wel een eerste vereiste en dat wordt door Simons' BASIC heel gemakkelijk gemaakt.
COMMANDO: LINE
OMSCHRIJVING: Het trekken van een rechte lijn tussen twee punten op het scherm.
GEBRUIK: LINE x-coordinaat beginpunt, y-coordinaat beginpunt, x-coordinaat eindpunt, y-coordinaat eindpunt, plot type.
LINE x,y,x1,y1,plot type
LINE 8,16,220,180,1
OPMERKINGEN: Achter LINE komen vijf parameters. Als er een ontbreekt komt tijdens het uitvoeren van een programma de foutmelding: ?SYNTAX ERROR IN ..... op het scherm.
Voor PLOT TYPE zie Overzicht plottypes.
Het maakt niet uit of X, of Y, groter of kleiner is dan X1 of Y1.
Typ dit korte programma in:

```
10 HIRES 0,3
20 LINE 0,30,300,182,1
30 PAUSE 3
40 NRM
```

en RUN <RETURN>
Vervang dan regel 30 door:

```
30 LINE 300,182,0,30,1
```

Na RUN blijkt het resultaat hetzelfde te zijn.

Als X1=X en Y1 niet gelijk aan Y dan loopt de lijn vertikaal.
Als Y1=Y en X1 niet gelijk aan X dan loopt de lijn horizontaal.

Zolang er met high resolution wordt gewerkt, kan X(X1) liggen tussen 0 en 319 en Y(Y1) tussen 0 en 199.

Het volgende programma zet verschillende lijnen op het scherm.
Op de vraag: „Waar begint de lijn?" moet eerst de X worden ingetoetst en <RETURN>worden ingedrukt, daarna gebeurt hetzelfde voor Y.
Dit nog eens herhalen voor het eindpunt.
X(X1) mag niet groter zijn dan 319 en Y(Y1)niet groter dan 199

LIJNEN 1.

```
10 INPUT"WAAR BEGINT DE LIJN";
   X,Y
20 INPUT"WAAR EINDIGT DE LIJN";
   X1,Y1
30 HIRES 1,0
40 LINE X,Y,X1,Y1,1
50 PAUSE 3
60 NRM
70 PRINT"♥"
80 GOTO 10
```

Commentaar: regels 10 en 20 vragen elk twee getallen.
regel 70 Het scherm wordt schoon geveegd.
Als een te groot getal voor een van de parameters wordt gegeven, verschijnt de foutmelding: ?ILLEGAL QUANTITY IN .. als het programma wordt ge-RUNd.

LIJNEN 2.

```
10 X = INT(RND(.)*320)
20 Y = INT(RND(.)*200)
30 X1 = INT(RND(.)*320)
40 Y1 = INT(RND(.)*200)
```

1.3 Tekenen

```
50 IF X = X1 AND Y = Y1 THEN 10
60 HIRES 6,7
70 LINE X,Y,X1,Y1,1
80 PAUSE 2
90 GOTO 10
```

Commentaar:
Dit steeds doorlopende programma kan onderbroken worden door op RUN STOP te drukken.
Er wordt gebruik gemaakt van de formule:
INT(RND(.)★(B-A+1))+A
waarin A het laagste gewenste getal voorstelt en B het hoogste.
Voor de X coördinaat geldt A=0 en B=319.
De formule wordt nu:
INT(RND(.)★(319-0+1))+0 ,uitgewerkt
INT(RND(.)★320)

Sommige schuine lijnen lijken eerder een soort trap dan een rechte lijn. Dat is niet te verhelpen, het aantal 'traptreden' en de lengte daarvan hangt af van de helling van de lijn.

```
10 HIRES 6,14
20 X1=10
30 LINE10,180,X1,80,1
40 X1=X1+40:IFX1>290 THEN 60
50 GOTO 30
60 Y1 = 90
70 LINE 10,180,310,Y1,1
80 Y1=Y1+30:IF Y1 >190 THEN 100
90 GOTO 70
100 PAUSE 10
110 NRM
```

Zo'n 'traplijn' ontstaat vooral bij lange lijnen, waar X en X1 ver uit elkaar liggen en Y en Y1 dicht bij elkaar, of omgekeerd.

```
10 HIRES 0,3
20 LINE 0,100,319,110,1
30 LINE 150,0,160,199,1
40 PAUSE 5
50 NRM
```

DE EERSTE TEKENING.
Zie werkblad 2. (Bijgevoegd achteraan dit hoofdstuk.) Van een tekening wordt altijd eerst een ontwerp op papier gezet.
Probeer zo overzichtelijk mogelijk te werken. Maak bij gebroken lijnen het beginpunt van de tweede lijn gelijk aan het eindpunt van de eerste.

B.v. LINE 0,72,88,16,1: LINE 88,16,128,64,1,: LINE 128,64,...... enz.

```
10 HIRES 0,3
20 LINE 0,72,88,16,1:LINE88,16,128,64,1:LINE128,64,200,56,1
30 LINE200,56,240,24,1:LINE240,24,288,40,1:LINE288,40,319,16,1
40 LINE108,40,136,48,1:LINE136,48,160,8,1:LINE160,8,176,40,1
50 LINE176,40,184,32,1:LINE184,32,208,48,1
60 LINE88,136,0,152,1:LINE88,136,0,168,1
70 LINE120,136,144,199,1:LINE120,136,200,199,1
80 LINE202,128,184,168,1:LINE202,128,216,168,1:LINE184,168,216,168,1
90 LINE200,168,200,192,1:LINE204,168,204,192,1
100 LINE250,120,236,160,1:LINE250,120,264,160,1:LINE236,160,264,160,1
110 LINE248,160,248,176,1:LINE252,160,252,176,1
120 PAINT200,160,1:PAINT248,152,1
130 PAINT 160,48,1:PAINT16,156,1
140 LINE 0,120,319,96,1:LINE0,132,319,100,1
150 LOW COL 5,0,0
160 LINE 0,136,319,136,1
170 PAUSE 10
180 NRM
```

1.3 Tekenen

Commentaar:
In de regels 120 en 130 wordt het commando PAINT gebruikt.
Een bespreking volgt na dit programma.
regels 20-50 : omtrek van de bergen
regels 60-70 : wegen
regels 80-110 : bomen
regel 140 : weg omhoog
Het commando LOW COL wordt later besproken.

Na 10 seconden verdwijnt de tekening van het scherm. Om de tekening nog eens te zien kan RUN <RETURN> worden ingetoetst. Het kan bij grote tekeningen enige tijd duren voordat het programma klaar is.

Er bestaat een commando om een tekening snel weer op het scherm te brengen. Typ zonder regelnummer in:
CSET 2 : PAUSE 10 en druk op <RETURN>.
De tekening is dan onmiddellijk weer te zien.
COMMANDO: CSET
OMSCHRIJVING: Dient om teksten te maken in hoofdletters, in hoofd- en kleine letters of om het laatste tekenscherm (grafische scherm) weer te voorschijn te roepen.
GEBRUIK: CSET getal (0,1 of 2)
CSET n
OPMERKINGEN: Als het getal 0 is, dan bestaat de tekst uit hoofdletters.
Als het getal 1 is, dan bestaat de tekst uit hoofd- en kleine letters.
Als het getal 2 is, dan wordt het laatste tekenscherm weer vertoond.

```
10 PRINT"♥"
20 PRINT"VAN BASIC TOT
   MACHINETAAL"
30 PRINT"OP DE COMMODORE 64."
40 CSET 1 : PAUSE 2: CSET 0 :
   PAUSE 2
50 GOTO 10
```

typ RUN <RETURN>
COMMANDO: PAINT
OMSCHRIJVING: Een geheel door lijnen omgeven figuur wordt gekleurd.
GEBRUIK: PAINT X-coordinaat, Y-coordinaat, plot type
PAINT 100,130,1
OPMERKINGEN: Het is absoluut vereist, dat de te kleuren figuur geheel is ingesloten door lijnen. Anders komt de kleur ook buiten de figuur en worden delen van het scherm gekleurd waar dat niet gewenst is. De randen van het scherm dienen wel als afsluiting van een figuur.
De X en Y coordinaten moeten binnen de figuur liggen, het maakt niet uit waar.
De achterste bergen in de tekening worden gekleurd met PAINT 160,48,1 maar PAINT 128,56,1 of PAINT 200,50,1 werken even goed.
Met LOW COL 5,0,0 is de horizon 8 pixels breed met groen gekleurd. Dat ziet er nog wel acceptabel uit.
Maar wordt een figuur tussen schuine lijnen gekleurd, dan komen er problemen.

Breid het teken-programma uit met:

```
170 LOW COL 6,0,0
180 PAINT 96,116,1
190 PAUSE 5
200 NRM
```

Nu wordt het omhoog gaande pad gekleurd, maar er verschijnen ook vreemde kleurblokken op het scherm. Dit komt doordat het kleurgeheugen slechts 1 Kbyte groot is. De 64000 pixels kunnen dus niet elk apart gekleurd worden.

Met het kleuren van tekeningen in high resolution moet dus enige voorzichtigheid

1.3 Tekenen

in acht worden genomen.
COMMANDO: LOW COL
OMSCHRIJVING: dient om de kleuren waarmee getekend of gekleurd wordt te wijzigen.
GEBRUIK: LOW COL getal, getal, getal
LOW COL c1,c2,c3
LOW COL 0,4,5
OPMERKINGEN: De getallen voor de kleuren lopen van 0 t/m 15. Zie tabel.
BIJZONDERHEDEN: Als LOW COL in high resolution wordt gebruikt, dan heeft het derde getal geen effect, maar het moet wel worden ingetypt.

VIERKANT.

```
10 HIRES 0,3
20 LINE 111,31,160,31,1:LINE160,
   31,160,80,1
30 LINE160,80,111,80,1:LINE111,
   80,111,31,1
40 LOW COL 7,0,0
50 PAINT 125,60,1
60 PAUSE5:NRM
```

Commentaar:
– het vierkant wordt eerst in banen van 8 pixels zwart gekleurd, daarna geel.
– de coördinaten van de linkerbovenhoek zijn (veelvouden van 8)-1: X = 14★8-1 en Y = 4★8-1.
– De coördinaten van de rechterbenedenhoek zijn veelvouden van 8: X = 20★8 en Y = 10 ★ 8. Andere coördinaten geven dikkere randen of geen rand. Dit is weer een gevolg van de indeling van het kleurgeheugen in blokken van 8 bij 8 pixels. Het tekenen van een rechthoek wordt vereenvoudigd met het
COMMANDO: REC
OMSCHRIJVING: Hiermee wordt een rechthoek getekend.
GEBRUIK: REC X-coördinaat linker bovenhoek, Y-coördinaat linker bovenhoek, breedte, hoogte, plot type
REC x,y,A,B,1
REC 63,47,73,33,1
OPMERKINGEN: A en B zijn dus geen coördinaten.
REC komt van het Engelse RECtangle = Nederlands REChthoek. De rechthoek uit het vorige programma kan dus zo worden getekend:

```
10 HIRES 0,3
20 REC 111,31,49,49,1
30 LOW COL 7,0,0
40 PAINT 125,60,1
50 PAUSE 5:NRM
```

Commentaar:
– de X en Y coördinaten van de linker bovenhoek zijn weer (8 vouden)-1.
– de breedte en de lengte zijn zo genomen, dat de coördinaten van de rechter benedenhoek een veelvoud van 8 zijn:
X + breedte = 111 + 49 = 160
Y + hoogte = 31 + 49 = 80
Typ ook in:

```
10 HIRES 4,7
20 REC 37,41,113,93,1
30 PAINT 80,60,1
40 PAUSE 5: NRM
```

Op het scherm komt een paars vierkant op gele achtergrond: de kleuren die HIRES aangeeft.
Voeg aan het programma toe:

```
25 LOW COL 1,0,0
```

en RUN <RETURN>
Het vierkant heet geen regelmatige zwarte rand.

Verander regel 20 in

```
20 REC 39,15,113,89,1
```

en RUN

1.3 Tekenen

Commentaar: De randen zijn gelijk omdat de rechthoek nu coördinaten heeft die in het 8 bij 8 kleurenpatroon passen.

RECHTHOEKEN.

```
10 HIRES 6,15
20 REC 79,63,57,25,1
30 REC 119,79,81,41,1
40 REC 175,103,97,89,1
50 LOW COL 7,0,0:PAINT 85,65,1
60 LOW COL 2,0,0:PAINT 180,82,1
70 LOW COL 5,0,0:PAINT 260,150,1
80 LOW COL 4,0,0:PAINT 122,82,1
90 LOW COL 1,0,0:PAINT 196,105,1
100 PAUSE 5:NRM
```

Commentaar:
– Met CSET 2:PAUSE 10 en <RETURN> komt het Mondriaan schilderij onmiddellijk weer op het scherm.
– let op de getallen voor X, Y en A,B; X+A is een 8-voud en eveneens Y+B.
– duidelijk is te zien dat een geheel door lijnen ingesloten figuur wordt gekleurd. Allereerst blijven er twee vierkantjes open.

DRIEHOEK.

```
10 HIRES 0,3
20 LINE 160,24,200,144,1:LINE200,
   144,120,144,1
30 LINE 120,144,160,24,1
40 PAINT 160,100,1
50 PAUSE 5:NRM
```

Commentaar:
– als in high resolution met de twee kleuren achter HIRES vermeld wordt gewerkt, dan ontstaan er geen problemen. Maar typ in:

```
35 LOW COL 7,0,0
```

en RUN het programma
Er komen nu zwarte blokken op de opstaande zijden van de driehoek, wat wel niet de bedoeling was. Dit is het gevolg van de beperkte ruimte van het kleurgeheugen.
Vervang regel 35 door

```
35 LOW COL 7,3,0
```

Dit geeft een beter resultaat omdat nu de achtergrondkleur dezelfde is als achter HIRES. Toch zijn er nog zwarte punten zichtbaar.

VIJFHOEK.

```
10 HIRES 4,7
20 LINE40,20,250,25,1:LINE250,
   25,312,85,1
30 LINE312,85,170,130,1:LINE170,
   130,25,63,1:LINE25,63,40,20,1
40 PAINT100,60,1
50 PAUSE 5:NRM
```

Commentaar: de cöordinaten van de lijnen sluiten op elkaar aan en de eindcöordinaten van de laatste lijn zijn gelijk aan de begincöordinaten van de eerste. De figuur is dus gesloten. Verander in regel 30 het getal 40 in 38 en RUN het programma.

Het gevolg van het kleuren van een niet gesloten figuur is duidelijk zichtbaar.
Verander het getal 38 in 39.

Feitelijk is de figuur niet gesloten maar dat heeft nu geen ongewenste gevolgen.
Voor wat langere programma's heeft Simons' Basic een commando, dat automatisch de regelnummers verzorgt. Het heet dan ook:
COMMANDO: AUTO
OMSCHRIJVING: Regelnummers voor een programma worden automatisch na <RETURN> gegeven. De tussenruimte tussen de nummers kan naar wens worden vastgesteld.
GEBRUIK: AUTO nummer beginregel,

1.3 Tekenen

tussenruimte
AUTO 10,10
OPMERKINGEN: Om de werking van AUTO uit te schakelen op <RETURN> drukken nadat een regelnummer op het scherm is gezet.

AUTO wordt vanzelfsprekend direct gebruikt.
Typ in:
AUTO 100,10 en druk op <RETURN>
Op het scherm komt:
READY
100 (met de cursor)
Typ nu:
PRINT"VAN BASIC „ <RETURN>
Op het scherm komt:
110 (met de cursor)
Typ:
PRINT"TOT MACHINETAAL" <RETURN>
Op het scherm komt:
120 (met de cursor)
Als nu op <RETURN> wordt gedrukt wordt AUTO uitgeschakeld.
Regel 120 is dus niet mee opgenomen.
Men kan later weer verder gaan met AUTO 120,10 <RETURN>.Let er dan wel op of regel 120 al in het voorgaande deel van het programma staat.
AUTO 120,10 wist n.l. een bestaande regel 120 uit.

TEKENING.
Zie werkblad 3. (Bijgevoegd achteraaan dit hoofdstuk.)
Typ in:
AUTO 100,10 <RETURN>
De AUTO functie is nu ingeschakeld, typ dus geen regelnummers in. Dat geeft kans op fouten. De enkele regelnummers tussen haakjes zijn slechts ter oriëntatie gegeven.

Commentaar:
– regels 340-360 Als een zelfde figuur enkele malen moet worden getekend, dan kan een herhalingslus worden gebruikt.
– In regel 360 staat het
COMMANDO: IF ... THEN ... :ELSE: ...
OMSCHRIJVING: Na het controleren van een voorwaarde zijn er twee mogelijkheden.

```
(100) HIRES 0,14
 ---  LOW COL 5,14,0
 ---  LINE 0,144,319,144,1
 ---  PAINT 160,160,1
 ---  LOW COL 1,14,0
(150) REC 32,128,8,16,1:REC48,128,8,8,1
 ---  PAINT34,130,1:PAINT50,130,1
 ---  LOW COL 10,14,0
 ---  REC24,120,48,24,1
 ---  PAINT40,122,1
(200) LOW COL 2,14,0
 ---  LINE80,119,16,119,1:LINE16,119,24,96,1
 ---  LINE24,96,72,96,1:LINE72,96,80,119,1
 ---  PAINT 48,104,1
 ---  LOW COL 9,14,1
(250) REC104,120,4,24,1:PAINT106,128,1
 ---  REC 144,120,4,24,1:PAINT146,128,1
 ---  LOW COL13,14,1
 ---  LINE106,64,120,119,1:LINE120,119,94,119,1
 ---  LINE94,119,106,64,1:PAINT104,96,1
```

1.3 Tekenen

```
(300) LINE 146,64,160,119,1:LINE160,119,134,119,1
 ---  LINE134,119,146,64,1:PAINT148,96,1


 ---  LOW COL15,14,1
 ---  REC200,128,8,15,1
 ---  X = 216
(350) REC X,120,8,16,1:PAINT X+4,128,1
 ---  X=X+16:IF X>248 THEN 370 :ELSE: GOTO 350
 ---  REC190,80,20,7,1:PAINT204,136,1:PAINT200,84,1
 ---  LOW COL1,14,0
 ---  LINE192,88,192,143,1:LINE208,88,208,112,1
(400) LINE208,112,264,112,1:LINE264,112,264,143,1
 ---  PAINT200,112,1
 ---  LINE104,16,104,32,1:LINE104,32,112,40,1
 ---  LINE112,40,88,56,1:LINE88,56,64,56,1
 ---  LINE64,56,48,48,1:LINE48,48,48,32,1
(450) LINE48,32,56,24,1:LINE56,24,104,16,1
 ---  PAINT80,40,1
 ---  LOW COL2,14,0
 ---  LINE200,40,208,79,1:LINE208,79,192,79,1
 ---  LINE192,79,200,40,1:PAINT200,72,1
(500) LINE209,96,256,96,1:LINE256,96,264,111,1:LINE264,111,209,111,1
 ---  LINE209,111,209,96,1:PAINT232,104,1
 ---  LOW COL 13,14,0
 ---  LINE272,143,280,128,1:LINE280,128,296,124,1
 ---  LINE296,124,304,112,1:LINE304,112,319,108,1
(550) PAINT304,128,1
 ---  PAUSE10:NRM
```

GEBRUIK: IF X=320 THEN X=0 :ELSE: X=Y
OPMERKINGEN:Dit is een uitbreiding van IF .. THEN in Basic 2.0

Denk er om dat :ELSE: tussen dubbele punten moet staan.
Als na: ELSE: naar een regel wordt verwezen, dan moet GOTO worden ingevoegd:
IF X>248 THEN 370 :ELSE: GOTO 350

1.3 Tekenen

HET GEBRUIK VAN LUSSEN
(Zie ook deel 7/5.1 blz. 1)
De gebruiker van Simons' BASIC kan zich bij het maken van tekeningen heel wat programmawerk besparen door lussen in het programma op te nemen. Er volgen een paar voorbeelden om dit duidelijk te maken.
RAILS MET DWARSLIGGERS

```
10 HIRES0,3
20 LINE0,150,319,150,1
30 LINE0,152,319,152,1
40 X=4
50 RECX,153,8,3,1
60 X=X+16:IFX<310THEN50
70 PAUSE5:NRM
```

Sommige gebouwen: scholen, kantoorflats, e.d. hebben rijen ramen. Ook bij het tekenen van zulke gebouwen brengen lussen uitkomst:
GEBOUW

```
10 HIRES1,14
20 LINE0,160,319,160,1
30 LINE20,80,30,60,1:LINE30,60,
   268,60,1:LINE268,60,278,80,1
40 REC20,80,258,79,1
50 X=24
60 RECX,100,6,6,1:LINEX,103,X+6,
   103,1:LINEX+3,100,X+3,106,1
70 RECX,130,8,16,1:LINEX+4,130,
   X+4,146,1:LINEX,134,X+8,134,1
80 X=X+12:IFX<270THEN60
90 PAUSE10:NRM
```

De tekencommando's van Simons' BASIC kunnen dus ook in een of meer lussen worden opgenomen. Wat ermee bereikt kan worden hangt van Uw vindingrijkheid af.

HIRES SAMEN MET LOW COL.
(Lees eerst 8/1.3 blz. 2 en blz. 11.)
Waar vermeld wordt, dat in beide commando's sprake is van een tekenkleur en een achtergrondkleur.

In het volgende programma gaan we het verband na tussen die twee kleuren. De tekenkleur in HIRES en LOW COL wordt aangeduid met resp. TH en TL, de achtergrondkleur met AH en AL.

```
10 TH=1:AH=0:HIRESTH,AH:TE=1
20 X1=INT(RND(.)*320)
30 Y1=INT(RND(.)*200)
40 TH=INT(RND(.)*16)
50 TL=TH:AL=AH
60 LINE160,100,X1,Y1,1
70 LOW COLTL,AL,0
80 TE=TE+1:IFTE<100THEN20
90 PAUSE10
100 NRM
```

Met dit programma kunnen vier mogelijkheden bestudeerd worden, door in regel 50 die tekens en achtergrondkleuren in te stellen.

1) TL=TH : AL=AH
Op het scherm komen allerlei gekleurde lijnen. Daarvoor zorgen de RND-opdrachten in de regels 20 t/m 40. De lijnen staan op een egaal zwarte achtergrond, want AL=AH=0

2) TL<>TH : AL=AH
Verander eerst regel 50 in:
50 TL=TH+1:AL=AH en RUN het programma.
Het resultaat is praktisch gelijk aan dat van programma 1)

3) TL=TH : AL<>AH
Verander eerst regel 50 in:
50 TL=TH:AL=AH+1
Op het scherm komen gekleurde lijnen, die als achtergrond blokken van 8 bij 8 pixels hebben in de kleur AL. De rest van de achtergrond is nog steeds zwart, kleur AH.

4) TL<>TH : AL=AH+1

1.3 Tekenen

Verander regel 50:
50 TL=TH+1:AL=AH+1
Het resultaat is gelijk aan dat van het derde experiment.

CONCLUSIE:
Zolang de achtergrondkleur van HIRES en LOW COL dezelfde is, geeft het kleuren van een tekening weinig of geen problemen.

DE COMMANDO'S LOW COL EN REC.
(zie 8/1.3 blz. 11)
In het programma hierna wordt een lus gebruikt, maar bovendien kunt U iets meer te weten komen over uw Simons' BASIC cartridge.

```
10 HIRES0,15:X=5:X1=200:A=195:
   B=10:KL=1
20 FORY=5TO180STEP25
30 LINEX,Y,X1,Y,1
40 RECX,Y+5,A,B,1
50 LOW COLKL,15,15
60 KL=KL+1:NEXT
70 PAUSE10
80 NRM
```

Het scherm vertoont een aantal lijnen en rechthoeken in verschillende kleuren.
Als U een oude cartridge hebt, dan zijn de lijnen wel verschillend gekleurd maar de rechthoeken niet!
In de oude cartridges zit namelijk een 'bug': LOW COL werkt niet samen met REC.
Deze bug geldt helaas ook voor de combinatie BLOCK en LOW COL.

COMMANDO: BLOCK
OMSCHRIJVING: tekent een ingekleurde rechthoek.
GEBRUIK: BLOCK X-coördinaat linkerbovenhoek, Y-coördinaat linkerbovenhoek, X-coördinaat rechterbenedenhoek, Y-coördinaat rechterbenedenhoek, plot type
BLOCK X,Y,X1, Y1, plot type
OPMERKINGEN: Dit commando is een combinatie van REC en PRINT.
Maar:
1) bij REC komen achter X en Y de breedte A en de hoogte B (zie 8/1.3 blz. 11); bij BLOCK de coördinaten van de rechterbenedenhoek;
2) bij REC kan met PRINT een andere kleur worden ingevuld, bij BLOCK is die kleur gelijk aan die van de omtrek;
3) BLOCK werkt veel sneller dan REC + PRINT.

```
10 HIRES0,15
20 BLOCK30,20,180,90,1
30 REC31,103,153,73,1
40 LOW COL4,15,0
50 PAINT150,150,1
60 PAUSE5:NRM
```

COMMENTAAR:
Let op het verschil in tekensnelheid.
regel 30 – coördinaten X en Y zijn een 8-voud −1
– coördinaten A en B zijn een 8-voud + 1 (zie 8/1.3 blz. 11)
– als U een gekleurde rechthoek hebt getekend met BLOCK, kunt U er met REC b.v. een zwarte rand omheen tekenen.

BLOCK EN KLEUREN
Oudere insteekprogramma's geven niet de gewenste kleuren:

1.3 Tekenen

```
10 HIRES0,15:X=0:X1=319:KL=1
20 FORY=0TO199STEP8
30 BLOCKX,Y,X1,Y+7,1
40 LOW COL KL,15,0
50 KL=KL+1:NEXT
60 PAUSE10:NRM
```

regel 20 t/m 50 toont het gebruik van een lus.
regel 20 – de 8 in STEPS zorgt er voor, dat elk blok steeds op een 8-voud voor Y begint. We hebben hier weer te maken met het kleurgeheugen, dat slechts 1 Kbyte groot is.
Met ...STEP 7 of minder komt er geen goede kleurindeling: sommige kleuren ontbreken. Met ...STEP 9 of hoger zijn er geen problemen. Ga dat na, door in regel 20 het getal na STEP te wijzigen.

HET TEKENEN VAN CIRKELS EN ELLIPSEN.
In 8/1.3 blz. 7 is al besproken, hoe met COS en SIN cirkels en ellipsen getekend kunnen worden.
Simons' BASIC maakt het gemakkelijker met het COMMANDO: CIRCLE
OMSCHRIJVING: met dit commando kunnen cirkels en ellipsen worden getekend.
GEBRUIK: CIRCLE x-coördinaat van het middelpunt, Y-coördinaat van het middelpunt, lengte van de straal horizontaal gemeten, lengte van de straal verticaal gemeten, plot type.
CIRCKE X,Y,XR,YR,plot type
CIRCLE 160,100,50,30,1
OPMERKING: Als voor XR en YR gelijke waarden worden genomen, dan ontstaat een cirkel. Bij verschillende waarden voor XR en YR ontstaan ellipsen.
BIJZONDERHEDEN: In de handleiding bij Simons' BASIC (blz. 6-10, NOTE) wordt vermeld, dat in HIRES-mode XR 1.4 maal YR moet zijn om een exacte cirkel te verkrijgen. Alleen als de cirkel met een printer op papier wordt afgedrukt zou XR gelijk moeten zijn aan YR.
Dit blijkt niet zo te zijn. Op de meeste beeldschermen geeft:
CIRCLE 160,100,50,50,1 een cirkel en
CIRCLE 160,100,70,50,1 een ellips.

```
10 HIRES0,15:KL=2
20 FORR=90TO10STEP-10
30 LOW COLKL,15,0
40 CIRCLE160,100,R,R,1
50 KL=KL+1:NEXT
60 PAUSE5:NRM
```

COMMENTAAR:
regel 40 – beide stralen zijn gelijk, op het scherm komen cirkels. Verander regel 40 in:
40 CIRCLE 160,100,1.4*R,R,1
en RUN het programma.
Nu worden er ellipsen getekend.

```
10 HIRES1,0
20 X=40:Y=40:TE=1
30 R1=INT(RND(.)*30)+5
40 R2=INT(RND(.)*30)+5
50 KL=INT(RND(.)*16):IFKL=
   0THEN50
60 LOW COL KL,0,0
70 CIRCLEX,Y,R1,R2,1:PAINTX,Y,1
80 TE=TE+1:IFTE<5THENX=X+80:
   GOTO30
90 IFTE=5THENY=Y+90:X=40:GOTO30
100 IFTE>5ANDTE<9THENX=X+80:
    GOTO30
110 PAUSE5:NRM
```

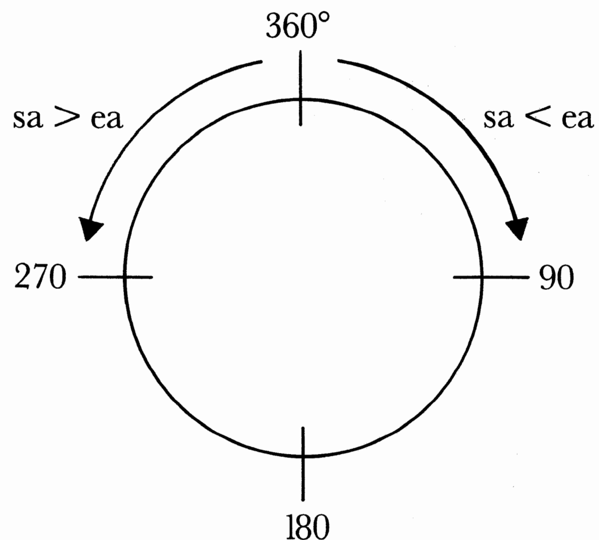
COMMENTAAR:
30 en 40 – bij de random-waarden wordt 5 opgeteld om te voorkomen dat de straal te klein wordt;
50 – IF KL=0 THEN 50 voorkomt, dat er een onzichtbare (zwarte) cirkel wordt getekend.
60 – merk op, dat de achtergrondkleuren

1.3 Tekenen

van HIRES en LOW COL gelijk zijn. Zo niet, dan komen de bekende kleurblokken te voorschijn. Probeer dat, door regel 60 te veranderen in:
60 LOW COL KL,1,0
70 – in 8/1.3 blz. 10 is er al op gewezen, dat de coördinaten voor PAINT binnen de figuur moeten liggen. Daarom zijn voor PAINT dezelfde coördinaten gekozen als voor het middelpunt van de cirkel.
80 t/m 100 zorgen voor een goede verdeling van de figuren over het scherm.
Voor het trekken van cirkelbogen – gedeelten van een cirkelomtrek – dient het volgende commando:

COMMANDO: ARC
OMSCHRIJVING: het tekenen van cirkelbogen, cirkels en veelhoeken.
GEBRUIK: ARC, X-coördinaat middelpunt, Y-coördinaat middelpunt, beginhoek van de boog, eindhoek van de boog, toename factor, lengte horizontale straal, lengte verticale straal, plot type.
ARC X,Y,SA,EA,I,XR,YR, plot type
ARC 160,100,30,210,1,80,80,1
OPMERKINGEN: de beginhoek (SA = start arc) en de eindhoek (EA = end arc) worden gemeten in graden. Voor de verhouding XR en YR zie CIRKEL.
BIJZONDERHEDEN: I is de beginletter van 'increment', dit betekent 'toename'.

Bij het bepalen van begin- en eindhoek moet rekening gehouden worden met het volgende:
1) 0 graden ligt bovenaan;
2) de cirkelboog wordt rechtsom getekend als SA‹EA en linksom als SA›EA



Figuur 8/1.3-1: Tekenrichting ARC commando

```
10 HIRES0,1
20 ARC220,100,330,20,1,40,40,1
30 ARC110,100,20,330,1,40,40,1
40 PAUSE5:NRM
```

COMMENTAAR:
De twee bogen samen vormen een complete cirkel. In beide gevallen begint het tekenen bij 20 graden, bij de eerste boog linksom naar 330 graden, bij de tweede rechtsom naar 330 graden.
De toenamefactor i geeft het aantal booggraden aan, waarmee de door ARC getekende stap voor stap toeneemt.
Voor I=1 is dat dus 1 booggraad. Daardoor duurt het tekenen van een boog tamelijk lang. Deze tijd kan bekort worden door voor I een hogere waarde te nemen.

```
10 HIRES0,1
20 ARC220,100,20,270,1,40,40,1
30 ARC110,100,20,270,8,40,40,1
40 PAUSE5:NRM
```

Vergelijk de tijden benodigd voor het tekenen van de twee bogen. Qua vorm is het verschil te verwaarlozen.

1.3 Tekenen

GEBRUIK BIJ UW TEKENINGEN EEN HOGERE WAARDE VOOR I DAN 1!

De I – en daarmee de tekensnelheid – kan verder verhoogd worden. Voor het tekenen van bogen (en cirkels) is I=15 wel het maximum. Bij gebruik van een nog hogere I gaat de cirkelvorm verloren, maar daarover straks meer.
Met het ARC commando kunnen ook cirkels worden getekend, SA is dan 0 en EA is 360.

```
10 HIRES0,3
20 CIRCLE240,100,60,60,1
30 ARC100,100,0,360,15,60,60,1
40 PAUSE5:NRM
```

COMMENTAAR:
- De twee cirkels verschillen qua vorm en tijd van tekenen maar weinig.
- De cirkel met ARC wordt getekend in stappen van 15 booggraden.
- Wijzig in regel 30 het getal 15 in 20 en RUN het programma opnieuw. De linkse figuur is nog maar net een cirkel.
- Maak van de 20 in regel 30 het getal 24 en RUN het programma. De linkse figuur wordt in stappen van 24 booggraden getekend en is feitelijk een regelmatige 15-hoek.
- Wijzig het getal 24 in regel 30 en maak er 30 van. Nu hebben we duidelijk te maken met een regelmatige 12-hoek. Algemeen geldt:
  I = 360/N waarin n het aantal hoekpunten van de veelhoek voorstelt.

Tussenliggende waarden voor I geven onregelmatige veelhoeken (de zijden zijn niet even lang). Bijvoorbeeld I=130 een driehoek.

I-waarde	Getekende figuur
24	regelmatige 15-hoek
30	regelmatige 12-hoek
36	regelmatige 10-hoek
40	regelmatige 9-hoek
45	regelmatige 8-hoek
60	regelmatige 6-hoek
72	regelmatige 5-hoek
90	vierkant
120	gelijkzijdige driehoek
180	verticale lijn

Maak tekeningen van deze veelhoeken met:
ARC 160,100,0,360,I,60,60,1
door voor i allerlei waarden in te vullen.

Een welkome aanvulling op Circle en ARC is het volgende commando.

COMMANDO: ANGL
OMSCHRIJVING: dient voor het tekenen van
1) stralen in een cirkelomtrek;
2) stervormige lijnenbundels.
GEBRUIK: ANGL X-coördinaat middelpunt, Y-coördinaat middelpunt, hoek tussen opeenvolgende stralen, lengte horizontale straal, lengte verticale straal, plot type.
ANGL X,Y,angle,XR,YR,plot type
ANGL 160,100,45,60,60,1
OPMERKINGEN: Om een aantal stralen te tekenen is een lus vereist.

```
10 HIRES0,3
20 ANGL160,100,0,60,60,1
30 PAUSE5:NRM
```

1.3 Tekenen

COMMENTAAR:
- er wordt slechts een lijn getekend;
- neem voor de 0 in regel 20 andere getallen en let op de stand van de lijn. Getallen groter dan 360 worden wel verwerkt, maar hebben weinig nut.

WAAIER

```
10 HIRES0,3
20 BLOCK157,20,160,100,1
30 PAUSE2
40 LOW COL4,3,0
50 FORH=0TO180STEP3
60 ANGL160,100,H,80,80,1
70 NEXT
80 PAUSE3
90 FORH=180TO0STEP-3
100 ANGL160,100,H,80,80,0
110 NEXT
120 PAUSE3:NRM
```

COMMENTAAR:
- regels 10 en 30: de achtergrondkleur in HIRES en LOW COL moet dezelfde zijn;
- regel 100: plot-type 0, de lijnen worden uitgewist.

RADARSCHERM

```
10 HIRES0,3
20 REC100,40,120,120,1
22 LINE100,40,130,20,1:LINE130,
   20,250,20,1:LINE250,20,220,
   40,1
24 LINE250,20,250,140,1:LINE250,
   140,220,160,1
30 CIRCLE160,100,52,52,1
40 PAINT105,42,1
50 LOW COL 7,3,0
60 H=0
70 ANGL160,100,H,43,43,1
80 ANGL160,100,H,43,43,0
90 H=H+10:IFH>360THENH=0
100 GOTO70
110 PAUSE2:NRM
```

COMMENTAAR:
Maak de ronddraaiende lichtstraal niet te lang, anders komen de beruchte kleurblokken te voorschijn.

De beweging wordt gerealiseerd door eerst een lijn te tekenen met plot-type 1 en die lijn dan met plot-type 0 uit te wissen. Dit proces wordt herhaald, maar nu staat de lijn een stapje verder door 'angle' te vergroten.

SPAKEN

```
10 HIRES0,3
20 CIRCLE160,100,60,60,1
30 CIRCLE160,100,5,5,1:PAINT160,
   100,1
40 FORH=0TO360STEP120
50 ANGL160,100,H,60,60,1
60 NEXT:PAUSE5:NRM
```

COMMENTAAR:
Het wiel heeft drie spaken. Het aantal spaken hangt af van het getal achter STEP in regel 40. Eerst moet het aantal spaken worden vastgesteld, noem dit aantal N.
Het getal achter STEP wordt gevonden met de berekening: 360/N.

Getal achter step	Aantal spaken
120	3
90	4
72	4
60	6
45	8
36	10
30	12
22.5	16
11.25	32

Probeer een paar van deze getallen in regel 30 van het laatste programma. Neem ook eens STEP12, STEP6, STEP1. (de spaken staan niet meer op onderling gelijke afstanden)
Nog een paar programma's met toepassing:

ZON

```
10 HIRES13,14
20 BLOCK0,160,319,199,1
```

1.3 Tekenen

```
30 LOW COL7,14,0
40 CIRCLE250,100,15,15,1:
   PAINT250,100,1
50 FORX=249TO340
60 ARCX,100,180,360,20,15,15,0
70 ARCX,100,0,180,20,15,15,1
80 NEXT:NRM
```

Maak bij het intikken van het volgende program gebruik van AUTO10,10 (zie 8/1.3 blz. 13). Zie ook werkblad 4.

```
10 HIRES0,14
20 LINE56,128,48,96,1:LINE48,96,
   296,88,1
30 LINE296,88,280,128,1
40 LINE96,95,96,80,1:LINE96,80,
   208,80,1
50 LINE208,80,208,91,1
60 LINE144,80,136,32,1:LINE136,
   32,160,32,1
70 LINE160,32,168,80,1:REC184,64,
   16,16,1
80 LINE137,40,162,40,1:LINE138,

90 PAINT150,44,1
100 REC192,66,6,6,1
110 LINE240,4,240,90,1
120 CIRCLE280,96,4,4,1
130 FORX=104TO200STEP16
140 CIRCLEX,88,2,2,1:NEXT
150 ARC184,128,270,90,15,32,32,1
160 ARC184,128,270,90,15,24,24,1
170 FORH=270TO360STEP22.5
180 ANGL184,128,H,35,35,1:NEXT
190 FORH=0TO90STEP22.5
200 ANGL184,128,H,35,35,1:NEXT
210 LOW COL 5,14,0
220 BLOCK0,104,40,112,1:BLOCK296,
    104,319,112,1
230 LOW COL1,14,0
240 LINE56,128,144,128,1:LINE224,
    128,280,128,1
250 LINE152,128,0,132,1:LINE216,
    128,32,136,1
260 LINE280,128,136,136,1
270 LOW COL 6,14,0
280 TE=1:X=8:Y=160
290 LINEX,Y,X+180,Y,1
300 X=X+15:Y=Y+4
310 IFY<199THEN290
320 LOW COL7,14,0
330 CIRCLE48,40,16,16,1:PAINT48,
    40,1
340 FORH=0TO360STEP20
350 ANGL48,40,H,25,25,1
360 NEXT
370 PAUSE10:NRM
```

– om de tekening nog eens te zien wordt in DIRECT MODE getypt:
CSET: PAUSE 10 en druk op RETURN
– Typ ook eens:
OPTION 10: LIST en druk op RETURN.
Alle gebruikte SIMONS' BASIC commando's zijn nu duidelijk te zien.

We kunnen vaststellen, of op een punt van het scherm iets getekend staat, of niet. Daarmee kan al dan niet een programma-onderdeel worden gestart.
Dat gebeurt met het
COMMANDO: TEST
OMSCHRIJVING: Er wordt vastgesteld of er op een bepaalde plaats van het beeldscherm iets getekend staat of niet.
GEBRUIK: TEST (X-coördinaat van het te controleren punt, Y-coördinaat van dat punt)
TEST (X,Y)
P=TEST(X,Y)
OPMERKINGEN: Als op (X,Y) een punt getekend staat, dan heeft TEST(X,Y) de waarde van het plot-type van dat punt. Dat zal dus vaak 1 zijn. De onder GEBRUIK vermelde P krijgt dan de waarde van het plot-type, meestal P=1. Zo'n punt kan deel uitmaken van een tekening. Staat er op (X,Y) NIETS, dan heeft TEST(X,Y) de waarde 0, en P=0

De volgende programma's laten zien wat er zoal met TEST kan worden gedaan.

1.3 Tekenen

TEST

```
10 HIRES7,4
20 LINE319,0,0,199,1
30 Y=4
40 FORX=0TO319
50 IFTEST(X,Y)=1THENY=Y+8:GOTO40
60 PLOTX,Y,1:NEXT
70 GOTO40
80 PAUSE5:NRM
```

ARCEREN

```
10 HIRES7,4
20 CIRCLE160,100,50,50,1
30 Y=50
40 Y=Y+4:FORX=110TO210
50 IFTEST(X,Y)=1THEN90
60 PLOTX,Y,0
70 NEXT:IFY>150THEN130
80 GOTO40
90 FORX=X+3TO210
100 IFTEST(X,Y)=1THEN40
110 PLOTX,Y,1
120 NEXT
130 PAUSE5:NRM
```

COMMENTAAR:
De cirkel ligt tussen X=110 en X=210
tussen Y= 50 en Y=150
Daarom wordt in regel 30: Y=50
regel 40: FOR X=110 TO 210
regel 70: IF Y>150
regel 90: FOR X=X+3 TO 210

Natuurlijk kunnen de getallen gebruikt worden, die de afmetingen van het beeldscherm aangeven:
regel 30: Y=0
regel 40: FOR X=0 TO 319
regel 70: IF Y > 199
regel 90: FOR X=X+3 TO 319
Dan duurt het programma echter ontzettend lang. Regel daarom de waarden van X en Y zo mogelijk naar de afmetingen van de te arceren figuur.
Als in regel 90 X+1 of X+2 wordt genomen in plaats van X+3 dan vindt geen volledige arcering plaats.
Met X+25 in plaats van X+3 ontstaat er een gedeeltelijk verlichte maansikkel.
Als dit alles bekeken is, zorg er dan voor dat in regel 90 weer FOR X=X+3 TO 210 staat en verander regel 20 in:
20 ARC 160,100,0,360,72,50,50,1
en RUN het program.
Ook in veelhoeken vindt arcering plaats.
Met IF TEST(X,Y)=... THEN...
kunnen allerlei programmaonderdelen worden ingeschakeld of overgeslagen. Bij het ontwerpen van spellen kan TEST een belangrijke rol spelen.

ONREGELMATIGE VORMEN TEKENEN.
Voor het tekenen van figuren die geen rechthoek of cirkelvorm hebben moet gebruik worden gemaakt van LINE al dan niet gecombineerd met ARC. Maar voor het tekenen van onregelmatige figuren zoals de omtrek van een boom, een hoop zand is er een speciaal commando.

COMMANDO: DRAW
OMSCHRIJVING: hiermee kunnen allerlei (onregelmatige) figuren worden getekend.
GEBRUIK: DRAW "string", X-coördinaat van het beginpunt van de tekening, Y-coördinaat van dat beginpunt, plot type
DRAW "nnnnn.....9",X,Y,plot type
DRAW "5757878686569", 160,100,1
OPMERKINGEN: De string bestaat uit getallen van 0-9, die aangeven dat de lijn waarmee wordt getekend, 1 pixel in een horizontale of verticale richting wordt verplaatst. Deze string mag hoogstens 74 getallen bevatten. Voor grotere figuren moeten verschillende strings samengekoppeld worden: T$ = T$ + "5788687845679"

1.3 Tekenen

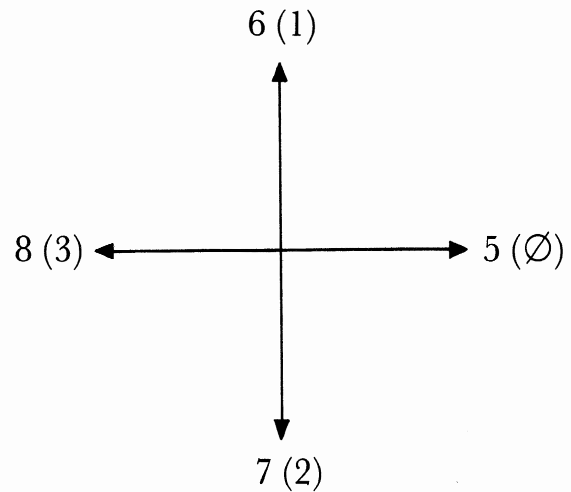
BIJZONDERHEDEN: Het commando DRAW bepaalt de vorm van de figuur, maar om de tekening op het scherm te zetten, moet het commando ROT voor het commando DRAW in het programma zijn opgenomen.

```
10 HIRES0,3:H=0
20 FORG=1TO25
30 ROTH,G
40 DRAW"5757878686569",160,100,1
50 NEXT
60 HIRES0,3:H=H+1:IFH<8THEN20
70 PAUSE2:NRM
```

COMMANDO: ROT
OMSCHRIJVING: Dit commando zet een met DRAW gevormde figuur op het beeldscherm in een bepaalde stand en een bepaalde grootte.
GEBRUIK: ROT rotatiehoek, grootte van de tekening.
ROT R,S
ROT 5,20

DE GETALLEN GEBRUIKT IN DE STRING VAN 'DRAW'

Waarde	DRAW-actie
0	een pixel naar rechts
5	een pixel naar rechts en tekent een punt
1	een pixel omhoog
6	een pixel omhoog en tekent een punt
2	een pixel omlaag
7	een pixel omlaag en tekent een punt
3	een pixel naar links
8	een pixel naar links en tekent een punt
9	moet aan het eind van de string staan



Figuur 8/1.3-2:DRAW-richtingen

OPMERKINGEN: Het getal r loopt van 0 t/m 7. Het getal s loopt van 1-255. s=1 geeft de normale grootte in pixels.

De getallen 5 t/m 8 maken dus de tekening, de getallen 0 t/m 3 verschuiven een pixel maar tekenen niet.

Getal	Rotatiehoek in graden
0	0
1	45
2	90
3	135
4	180
5	225
6	270
7	315

Dit alles lijkt misschien wat ingewikkeld. De volgende programma's laten zien hoe DRAW en ROT in de praktijk werken:

```
10 HIRES0,3
20 ROT0,20
30 DRAW"5757878686569",160,100,1
40 PAUSE5:NRM
```

1.3 Tekenen

In veel gevallen is het gemakkelijker om de string achter DRAW ook werkelijk als string samen te stellen. Dat tonen de volgende programma's:

```
10 HIRES0,3
20 T$="5757878686569"
30 ROT0,20
40 DRAWT$,160,100,1
50 DRAWT$,80,50,1
60 PAUSE3:NRM
```

Zo'n string kan natuurlijk in een lus worden gebruikt.

```
10 HIRES0,3
20 T$="5757878686569"
30 ROT1,5
40 TE=1
50 X=INT(RND(.)*300):Y=INT
   (RND(.)*180)
60 DRAWT$,X,Y,1
70 TE=TE+1:IFTE>15THEN80:ELSE:
   GOTO50
80 PAUSE3:NRM
```

Het volgende programma laat zien wat het gevolg is van het veranderen van de rotatiehoek.

```
10 HIRES0,3:H=0
20 T$="5757878686569"
30 FORG=1TO25
40 ROTH,G
50 DRAWT$,160,100,1
60 NEXT
70 HIRES0,3:H=H+1:IFH<8THEN30
80 PAUSE2:NRM
```

COMMENTAAR: Als in regel 70 HIRES 0,3 weggelaten wordt blijven alle tekeningen op het scherm.

Het samenstellen van de voor DRAW benodigde string lijkt misschien een moeilijke opgave.
Het volgende programma is bedoeld als hulpmiddel bij het samenstellen van zo'n string.

STRINGMAKER

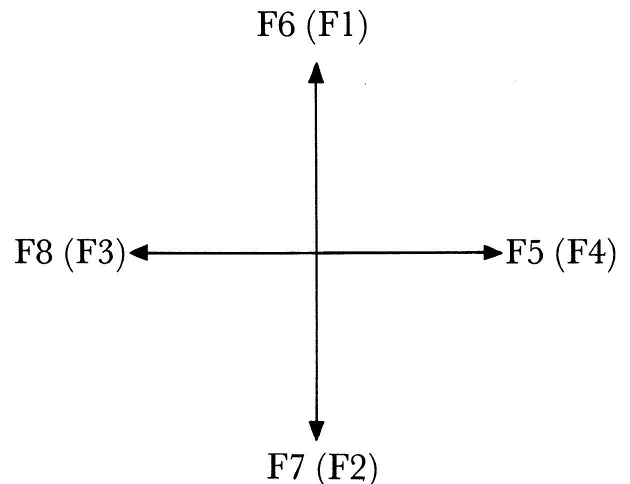
```
10 PRINT"♥":INPUT"X=COORDINAAT
   BEGINPUNT";X
20 INPUT"Y-COORDINAAT BEGINPUNT";Y
30 HIRES0,3:T$=""
40 A=INKEY
50 ONAGOSUB100,150,200,250,300,
   350,400,450,500
60 GOTO40
100 Y=Y-8:PAUSE1:T$=T$+"1":RETURN
150 Y=Y+8:PAUSE1:T$=T$+"2":RETURN
200 X=X-8:PAUSE1:T$=T$+"3":RETURN
250 X=X+8:PAUSE1:T$=T$+"0":RETURN
300 LINEX,Y,X+8,Y,1:X=X+8:PAUSE1:
    T$=T$+"5":RETURN
350 LINEX,Y,X,Y-8,1:Y=Y-8:PAUSE1:
    T$=T$+"6":RETURN
400 LINEX,Y,X,Y+8,1:Y=Y+8:PAUSE1:
    T$=T$+"7":RETURN
450 LINEX,Y,X-8,Y,1:X=X-8:PAUSE1:
    T$=T$+"8":RETURN
500 IFLEN(T$)>74THEN600
510 T$=T$+"9":NRM
520 PRINT"♥":PRINT:PRINT"T$=
    "CHR$(34)T$CHR$(34)
530 END
600 NRM:PRINT"♥":PRINT"T$ IS TE
    LANG"
610 END
1000 HIRES0,3
1020 ROT0,1
1030 DRAWT$,160,100,1
1040 PAUSE5:NRM

READY.
```

Typ de listing zorgvuldig in en SAVE het program eerst op disk of tape.
Tik dan pas RUN (RETURN)
Eerst wordt gevraagd naar de coördinaten van het beginpunt, die moet U dus typen.
Dan komt het lege beeldscherm, waarop U de gewenste tekening kunt gaan maken. Dat gebeurt met de functietoetsen F1 t/m F8. (F2 = SHIFT/F1, F4 = SHIFT/F3 enz.) Met F1 t/m F4 wordt

1.3 Tekenen

het tekenpunt een pixel verschoven, maar er wordt NIETS getekend. Met F5 t/m F8 wordt het tekenpunt een pixel verschoven en wel getekend.
Ter wille van de duidelijkheid komen er op het beeldscherm lijnen van 8 pixels voor elke druk op een functietoets. Anders zou de tekening te klein zijn om goed te bekijken.
Het einde van de string wordt aangegeven door het getal 9, daarvoor dient u op F9 te drukken. Dit gebeurt door eerst de C=toets links onder op het toetsenbord in te drukken en dan F1. De richtingen waarin met de functietoetsen wordt getekend ziet U in deze figuur.

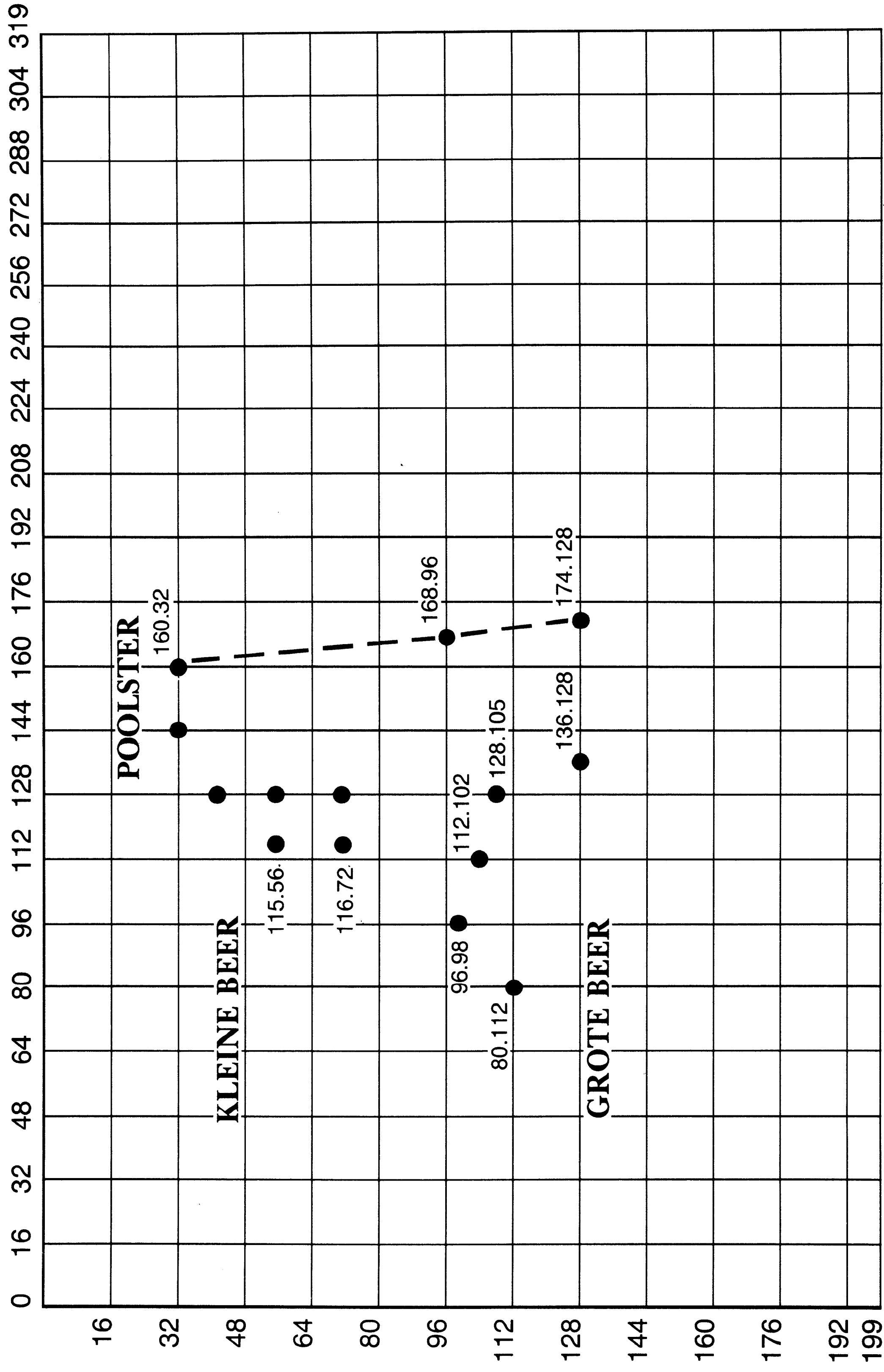
Is de tekening met F1 t/m F8 gemaakt, dan beëindigt F9 het tekenprogramma. Op het beeldscherm komt dan de string T$ te staan, die de zojuist gemaakte tekening voorstelt.
Breng dan de cursor op de letter T met de cursor omhoog toets. Verschuif vervolgens de string door SHIFT ingedrukt te houden en dan 4 maal op INST/DEL te



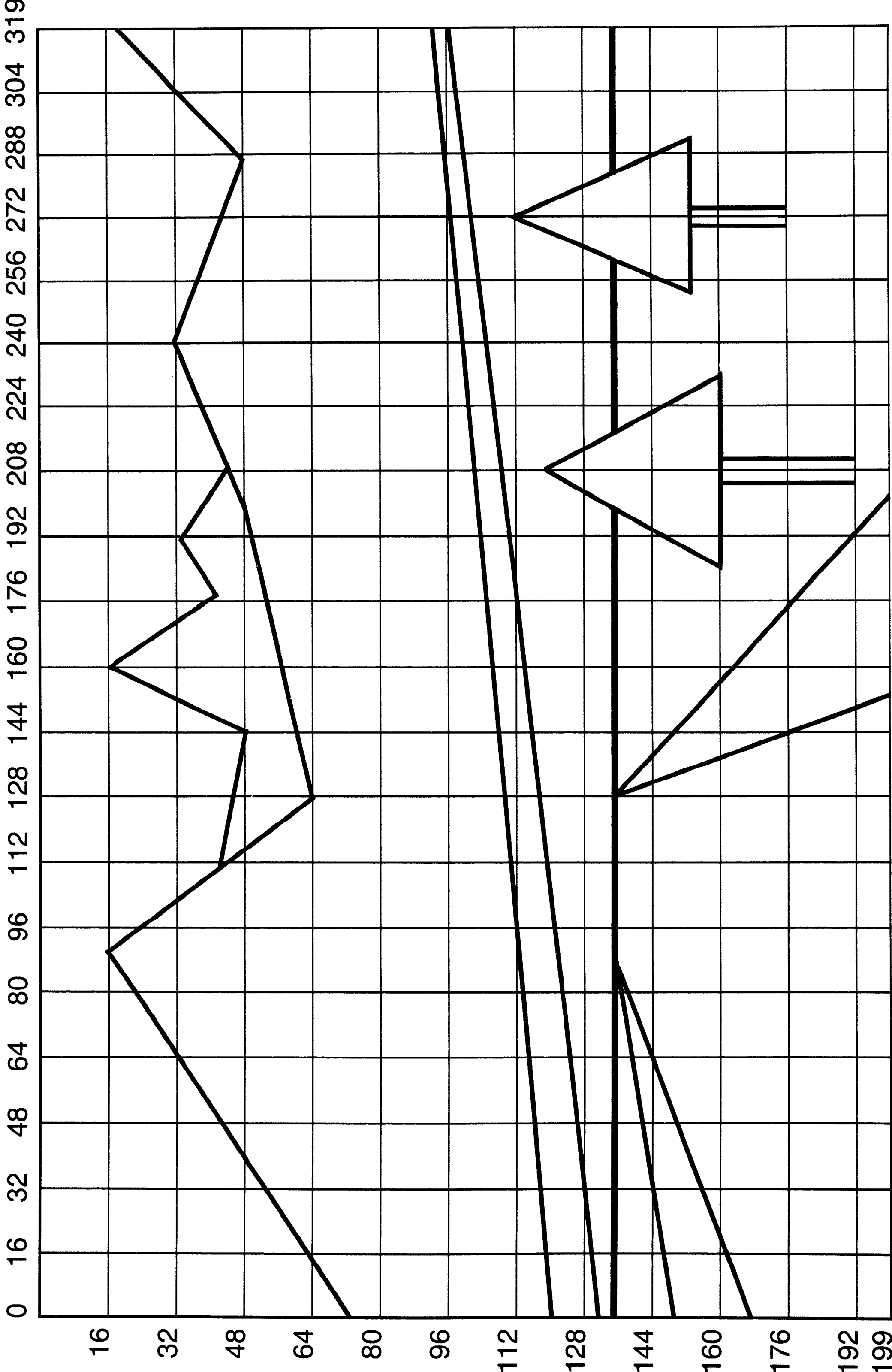
Figuur 8/1.3-3: Richtingen tekenhulpje

drukken. Zet in de vrijgekomen ruimte 1010
Typ nu: RUN 1000 en druk op RETURN
De tekening voorgesteld door T$ komt dan op het scherm.
Door de getallen achter ROT in regel 1020 te wijzigen kunnen rotatiehoek en grootte naar wens worden ingesteld.

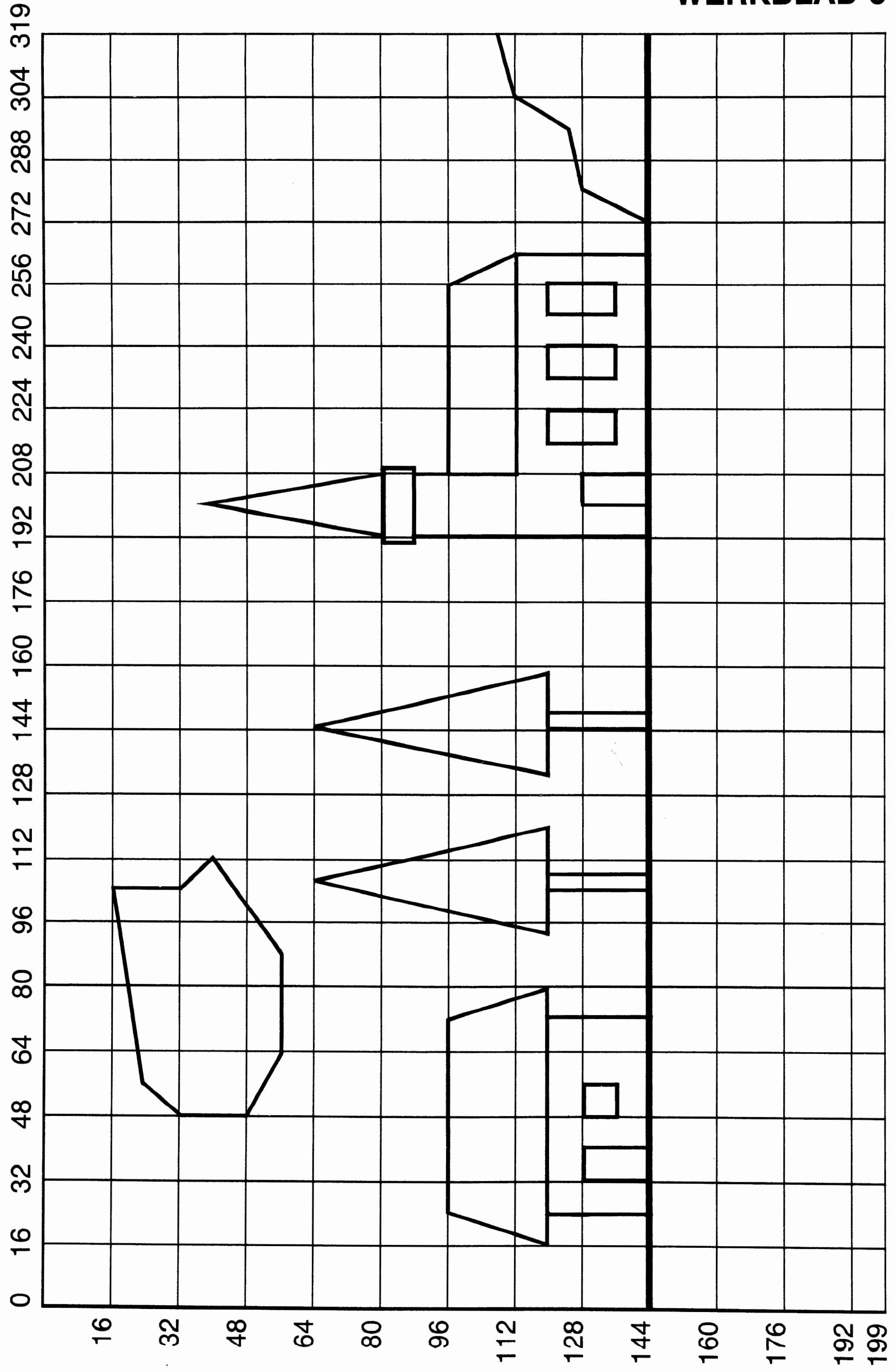
WERKBLAD 1



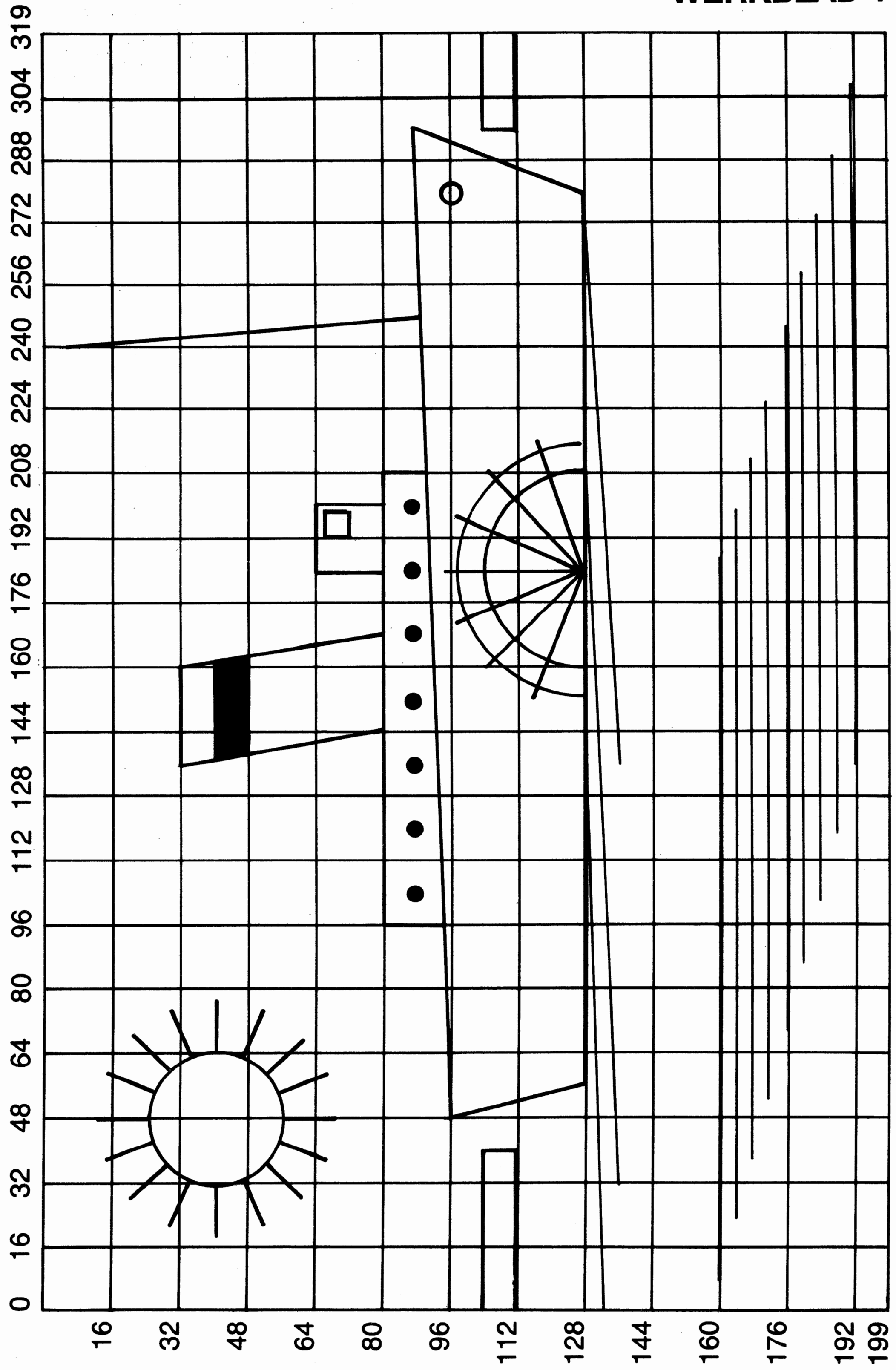
WERKBLAD 2



WERKBLAD 3



WERKBLAD 4



WERKBLAD MODEL

319
304
288
272
256
240
224
208
192
176
160
144
128
112
96
80
64
48
32
16
0

16 32 48 64 80 96 112 128 144 160 176 192 199

8/1.4

Sprites (MOBs)

De mogelijkheid om met de Commodore 64/128 'sprites' te gebruiken kan het schermbeeld echt aantrekkelijk maken. Het woord sprite betekent geest of kabouter; soms spreekt men ook wel van 'MOB' (movable object block, oftewel beweegbaar beeldstuk). Een sprite is een figuur die in alle richtingen over het beeldscherm kan worden bewogen en bovendien onafhankelijk is van letters of tekeningen die al op het scherm staan.
Een sprite beweegt over die tekens heen zonder ze uit te wissen en kan, desgewenst, ook achter de tekens langs gaan. Als op het beeldscherm tekst staat en ook een sprite, dan verdwijnt de tekst na het indrukken van SHIFT/CLR, maar de sprite blijft staan.

Het werken met sprites op de Commodore 64 vereist weer de nodige POKEs en DATA, maar Simons' BASIC maakt alles veel eenvoudiger.
Zo'n sprite is een bewegend beeldje, dat bestaat uit 21 rijen van 24 pixels. Een byte kan de gegevens van 8 pixels bevatten. Voor elke rij moeten dus 3 bytes geheugenruimte worden gebruikt en dat is voor een sprite 21 maal 3 = 63 bytes. In de praktijk neemt men altijd 64 bytes.
Dit is de HIRES-sprite, die maar één enkele kleur kan bezitten.
Daarnaast bestaan er MULTICOLOUR sprites, die bestaan uit 21 rijen van 12 pixels. Deze sprite kan hoogstens 3 verschillende kleuren bevatten, maar de tekening is wat grover dan die van de hires-sprite. Elke punt van dit beeldje is n.l. twee pixels breed. Ook voor deze sprite moet 63 (64) bytes worden uitgetrokken.

Voor beide soorten geldt dat er hoogstens 8 sprites tegelijkertijd op het beeldscherm mogen gebruikt worden. Wel kunnen er gegevens voor meer dan 8 sprites in het geheugen worden gezet.

Het gebruik van sprites kan in de volgende delen gesplitst worden:

1) het vastleggen van een stukje geheugenruimte voor elke sprite.
2) het ontwerpen van de sprite.
3) de sprite op het beeldscherm zetten.
4) de sprite over het beeldscherm laten bewegen.
5) – indien gewenst – botsingen van sprites signaleren en een reactie op zo'n botsing programmeren.

1) GEHEUGENRUIMTE VOOR SPRITES.

Sprites kunnen zowel op een tekstscherm als op een tekenscherm gebruikt worden. Nu is er, behalve voor sprites, ook geheugenruimte nodig voor het BASIC programma, variabelen, strings enz.

1.4 Sprites (MOBs)

Het gevaar, dat het gebruik van sprites geheugenruimte voor andere programmadelen overschrijft is lang niet denkbeeldig. (Zie hoofdstuk 8/1.2).

Geheugenruimte voor een sprite wordt gereserveerd door het

COMMANDO: DESIGN

OMSCHRIJVING: legt geheugenruimte vast voor een sprite.
GEBRUIK: DESIGN, aanduiding hires- of multicolour sprite, adres van de eerste byte van de sprite

DESIGN c,ad
DESIGN c,ad + 49152
DESIGN 0, 12288
DESIGN 1, 61440

OPMERKINGEN: De 0 achter DESIGN geeft aan, dat we met een HIRES-sprite te maken hebben, een 1 duidt een MULTICOLOUR sprite aan;

ad = het adres van het eerste byte van de sprite;
Het getal 49152 moet bij het adres worden opgesteld, als de sprite op het tekenscherm gebruikt zal worden.

Hoe komen we nu aan de adressen voor de sprite?
Elke sprite beslaat een 'blok' geheugenruimte van 64 bytes. Het zal bekend zijn, dat het getal 64 (2 tot de zesde macht) beter hanteerbaar is in het computersysteem dan 63.
Daarom krijgt elke sprite zijn eigen bloknummer. In de handleiding bij Simons' BASIC wordt vermeld, dat als bloknummers gebruikt kunnen worden: 32 t/m 63 en 128 t/m 255.

U kunt nu het bij een blok behorende adres vinden, door het bloknummer met 64 te vermenigvuldigen.

BLOK-NUMMER	ADRES EERSTE BYTE	
	TEKST-SCHERM	TEKEN-SCHERM
32	2048	51200
33	2112	51264
34	2176	51328
...	.....	.....
128	8192	57344
129	8256	57408
130	8320	57472
...	.....	.....
192	12288	61440
193	12352	61504
194	12416	61568

enzovoorts.

De eerste rij adressen moet worden gebruikt voor sprites op het tekstscherm, de tweede rij voor sprites op het tekenscherm.

Zoals gezegd, er bestaat gevaar dat een bepaald geheugenblok al in gebruik is genomen door het gewone programma. Wordt in datzelfde blok dan een sprite geplaatst dan 'crasht' het programma. De listing kan (gedeeltelijk) verdwijnen en de computer kan zelfs vast komen te staan, zodat er niet anders opzit, dan de computer te resetten. Het programma is daarna wel verdwenen.

SAVE daarom altijd eerst het programma op tape of disk alvorens RUN te geven. Dat kan veel programmeerwerk

1.4 Sprites (MOBs)

besparen en het is dan altijd mogelijk het bloknummer met het bijbehorende adres te wijzigen.
Over het algemeen geven bloknummers van 192 en hoger de minste last.

2) HET ONTWERPEN VAN EEN SPRITE

Dit wordt door Simons' BASIC al heel gemakkelijk gemaakt: we kunnen de sprite ontwerpen in het programma zelf. Daarvoor moet een 'rooster' worden opgezet van 21 programmaregels, elk beginnend met het @ teken en daarachter 24 punten voor een hires-sprite (12 voor een multicolour sprite, maar daarover later).

Elk van die 21 regels ziet er zo uit:
REGELNUMMER @ 24 PUNTEN
1000 @..........

Het is belangrijk, dat alle regelnummers uit een zelfde aantal cijfers bestaan. Anders komen er verschuivingen in het rooster en dat maakt het ontwerpen onnodig moeilijk. Neem dus b.v. regels 300 t/m 500 of 2700 t/m 2900, niet een reeks van 900 tot en met 1100.

Als in het werkrooster in plaats van een punt de letter B wordt gezet, dan stelt elke B een punt van de sprite voor.
We tekenen de sprite met letters B.

Een voorlopig ontwerp zou op papier gemaakt kunnen worden, maar hier volgt een programma, dat het ontwerpen van een sprite op het scherm mogelijk maakt.

SPRITEMAKER

```
100 REM SPRITEMAKER
110 PRINT"■":COLOUR3,15
120 PRINT"■■■■■■■■■■*** SPRITE MAKER ***
130 PRINT"■■■■■■■■■■        DOOR
140 PRINT"■■■■■■■■■■     G.N.M.MERZ
150 PRINT"■■■■■■■■■■   COPYRIGHT 1985
160 TE=0
200 PAUSE5
210 PRINT"■":COLOUR3,3
220 PRINT"■■■"
390 PRINT"■":PRINT"■■■■■U KRIJGT EEN RASTER TE ZIEN VAN
400 PRINT"      21 REGELS MET ELK 24 PUNTEN.
410 PRINT"DIT IS DE GROOTTE VAN EEN HIRES-SPRITE.
420 PAUSE10
430 PRINT"■ ■MET DE CURSOR TOETSEN KUNT U DE CURSOR
440 PRINT" OP ELKE GEWENSTE PUNT IN EEN  REGEL
450 PRINT" ZETTEN.ALS U DAN OP DE TOETS 'B' DRUKT
460 PRINT" KOMT ER IN DE PLAATS VAN DE . EEN B"
470 PAUSE10
480 PRINT"■■  ALS U EEN ANDERE REGEL WILT INVULLEN
490 PRINT"    DRUK DAN EERST OP ■■ RETURN ■"
500 PAUSE10
510 PRINT"DE DOOR U GEMAAKTE REGEL WORDT DAN IN
520 PRINT"HET GEHEUGEN VAN DE C=64 GEZET.
530 PAUSE5
540 PRINT"■■ALS U ALLE REGELS HEBT BEWERKT,TYP DAN:
550 PRINT"■ ■■ RUN 600 ■ EN DRUK OP RETURN.
```

1.4 Sprites (MOBs)

```
560 PRINT"U ZIET DAN DE SPRITE DIE U GEMAAKT HEBT.
570 PRINT" DRUK OP EEN TOETS "
580 GETA$:IFA$=""THEN580
590 GOTO1230
600 GOTO960
960 PRINT""
970 IFTE=1THEN1300
980 TE=1
990 DESIGN0,16320
1000 @........................
1010 @........................
1020 @........................
1030 @........................
1040 @........................
1050 @........................
1060 @........................
1070 @........................
1080 @........................
1090 @........................
1100 @........................
1110 @........................
1120 @........................
1130 @........................
1140 @........................
1150 @........................
1160 @........................
1170 @........................
1180 @........................
1190 @........................
1200 @........................
1210 PRINT""
1220 IFTE=1THEN1300
1230 PRINT"":LIST1000-1200
1300 PRINT""
1310 MOB SET0,255,0,0,0
1320 MMOB0,30,180,30,180,0,255
1330 PRINT" F1 -SPRITE BEWEEGT "
1340 PRINT" F3 -SPRITE 2 MAAL BREDER"
1350 PRINT" F5 -SPRITE 2 MAAL HOGER"
1360 PRINT" F7 -SPRITE 2 MAAL HOGER EN BREDER."
1370 PRINT" F2  (SHIFT/F1) - EINDE.
1380 A=INKEY
1390 ONAGOSUB1500,1600,1700,1800,1900,2000,2100,2200
1400 GOTO1380
1410 END
1500 REM SPRITE BEWEEGT NAAR RECHTS
1510 FORP=60TO340
1520 RLOCMOB0,P,180,0,100
```

1.4 Sprites (MOBs)

```
1530 NEXTP
1540 RLOCMOB0,60,180,0,50
1550 RETURN
1600 REM EINDE PROGRAM
1610 MOB OFF 0
1620 PRINT"[illegible]WILT U DE GEGEVENS VAN DE ZOJUIST
1630 PRINT"GEMAAKTE SPRITE NOG EENS BEKIJKEN OF
1640 INPUT"MISSCHIEN OP PAPIER ZETTEN? JA/NEE";A$
1650 IF LEFT$(A$,1)="J"THENLIST1000-1200
1660 IF LEFT$(A$,1)="N"THENPRINT"[illegible]"
1670 PRINT"[illegible]ALS U EEN NIEUWE SPRITE WILT MAKEN
1680 PRINT"DAN MOET U ALLE B'S IN HET ROOSTER WEER
1690 PRINT"VERANDEREN IN ......"
     :PRINT"[illegible]V E E L   S U C C E S !":TE=0:END
1700 REM SPRITE 2 MAAL BREDER
1710 MMOB0,60,180,60,180,1,255
1720 PAUSE5
1730 MMOB0,60,180,60,180,0,255
1740 RETURN
1900 REM SPRITE 2 MAAL HOGER
1910 MMOB0,60,180,60,180,2,255
1920 PAUSE5
1930 MMOB0,60,180,60,180,0,255
1940 RETURN
2100 REM SPRITE 2 MAAL BREDER EN HOGER
2110 MMOB0,60,180,60,180,3,255
2120 PAUSE5
2130 MMOB0,60,180,60,180,0,255
2140 RETURN

READY.
```

COMMENTAAR:
– SAVE eerst het programma alvorens RUN te geven.
– Lees de tekst op het scherm rustig door; druk daarna op een willekeurige toets.
– Op het scherm komt een werkrooster waarin met letters B de sprite kan worden getekend. Als een regel klaar is, druk op RETURN alvorens een andere regel te bewerken.
– Is de sprite 'geheel getekend, toets dan in RUN600 en druk op RETURN. Op het scherm verschijnt nu een menu. Met de aangegeven functietoetsen kan de sprite van alle kanten worden bekeken.
Na het indrukken van F3, F5 of F7 moet u enige ogenblikken wachten, daarna krijgt de sprite zijn oorspronkelijke grootte weer.
Het werkrooster kan (met een printer) worden gekopieerd.

3) en 4) DE SPRITE OP HET BEELDSCHERM BRENGEN EN VOORTBEWEGEN

Aangenomen wordt, dat het commando DESIGN en de 21 regels met de sprite al

1.4 Sprites (MOBs)

in het programma zijn opgenomen. Denk er om, dat het getal achter de 0 of de 1 van DESIGN moet liggen tussen 2048 en 16320 als de sprite op een tekstscherm wordt gebruikt. Bij gebruik van een tekenscherm ligt dat getal tussen 51200 en 65472.
De sprite – of MOB zo u wilt – wordt in het leven geroepen met het

COMMANDO: MOB SET

OMSCHRIJVING: Met dit commando wordt een sprite gedefinieerd.
GEBRUIK: MOB SET sprite nummer, blok nummer van de sprite, kleur, prioriteit, soort sprite

MOB SET mb,blk,col,pr,res
MOB SET 1,192,0,0,0

OPMERKINGEN: het nummer van de sprite kunt u zelf bepalen, meestal worden de nummers 1 t/m 8 gebruikt.
BELANGRIJK: als twee of meer sprites over het scherm bewegen, dan gaat de sprite met het laagste nummer VOOR de andere sprite(s) langs. Het spritenummer is dus tevens een soort volgnummer: eerst sprite 1, dan sprite 2 enz.
: met het bloknummer is het beginadres van de sprite al berekend;
: alle beschikbare kleuren (0 t/m 15) kunnen gebruikt worden.
: wordt er voor pr(ioriteit) een 0 ingevuld, dan gaat de sprite voor de tekst of de tekening op het beeldscherm langs. Bij pr=1 beweegt de sprite achter de tekst of tekening op het beeldscherm.
: res = 0 duidt een hires sprite, res = 1 een multicolour sprite aan.

Om de sprite op het scherm te zetten dient het

COMMANDO: MMOB

OMSCHRIJVING: zet de sprite op het beeldscherm en/of beweegt de sprite over het scherm.

GEBRUIK: MMOB nummer van de sprite, X-coordinaat linkerbovenhoek sprite, Y-coordinaat linkerbovenhoek, X-coordinaat linkerbovenhoek na verplaatsing, Y-coordinaat linkerbovenhoek na verplaatsing, vergroting, snelheid

MMOB mn,x1,y1,x2,y2,expansion, speed
MMOB 1,0,190,350,190,0,100

OPMERKINGEN: het spritenummer mn is gelijk aan het spritenummer mb in MOB SET.
: de x-coordinaten lopen van 0 t/m 346 en de y-coordinaten van 0 t/m 255. Dit is verder dan de normale scherm-coordinaten: zo kan de sprite aan alle kanten van het scherm opkomen of verdwijnen.
: voor grootte (expansion) kunnen de cijfers 0 t/m 3 worden gebruikt. Ze hebben de volgende betekenis:
0 - de sprite heeft zijn normale grootte;
1 - de sprite wordt 2 x breder;
2 - de sprite wordt 2 x hoger;
3 - de sprite wordt 2 x breder en hoger.
Dit alles kan worden bekeken met het hierboven gebruikte programma SPRITEMAKER.
: de snelheid waarmee de sprite over het scherm beweegt, wordt geregeld met de getallen 1 (snel) t/m 255 (langzaam).

In het volgende programma worden de besproken commando's en het werkrooster gebruikt.

1.4 Sprites (MOBs)

LOCOMOTIEF

```
20 HIRES 0,15:COLOUR15,15
30 LINE0,160,319,160,1
40 LOW COL 12,15,0
50 BLOCK 40,120,120,159,1
55 LOW COL 5,15,0
60 LINE180,159,260,159,1:LINE260,159,230,110,
   1:LINE230,110,200,125,1
70 LINE200,125,180,159,1:PAINT200,156,1
80 DESIGN 0,51776
100 @........................
110 @..BBBBBBB...............
120 @..BBBBBBB...............
130 @..........BBBBBB........
140 @..........BBBBBB........
150 @.................BBB....
160 @.................BBB....
170 @........................
180 @....................BB..
190 @BBBBBB..............BB..
200 @.BBBBB.....BBB......BB..
210 @.B..BB.....BBB......BB..
220 @.B..BBBBB..BBB..BB..BB..
230 @.B..BBBBBBBBBBBBBBBBBBB.
240 @.BBBBBBBBBBBBBBBBBBBBBBB
250 @.BBBBBBBBBBBBBBBBBBBBBBB
260 @.BBBBBBBBBBBBBBBBBBBBBB.
270 @BBBBBBBBBBBBBBBBBBBBBBBB
280 @.BBBB..BBBB..BBBB..BBBB.
290 @..BB...BBBB..BBBB...BB..
300 @..BB....BB....BB....BB..
310 MOB SET1,41,0,0,0
320 MMOB 1,0,190,350,190,0,100
330 MOB OFF 1
340 PAUSE3:NRM
```

COMMENTAAR:
– Regel 80 – de sprite is een hires-sprite, gebruikt op een hires scherm.
– Regel 310 – sprite nummer 1, bloknummer 41.
– Regel 320 – sprite nummer 1, let op X2 = 350: de sprite verdwijnt aan de rechterkant.

Regel 330 bevat het

COMMANDO: MOB OFF

OMSCHRIJVING: verwijdert een sprite geheel van het scherm.
GEBRUIK: MOB OFF nummer van de sprite

MOB OFF n
MOB OFF 1

OPMERKING: Met dit commando wordt voorkomen, dat een (deel van een) sprite op het scherm blijft staan.

1.4 Sprites (MOBs)

Als in het werkrooster van bovenstaand programma de gegevens voor de bekende ballon uit de handleiding bij de Commodore 64 worden gezet, dan kunt u daarmee allerlei bewegingen mee laten uitvoeren, door andere waarden voor x, y, x1 en y1 in regel 320 te plaatsen.

Door regel 310 als volgt te wijzigen:
310 MOB SET 1,41,6,1,0
gaat een blauwe locomotief achter de obstakels langs.
Probeer zelf eens de loc voor het bruine blok maar achter de groene berg te laten bewegen.

Er is nog een commando om een sprite voor te bewegen:

> COMMANDO: RLOCMOB
>
> OMSCHRIJVING: beweegt een sprite, die al ergens op (of buiten) het scherm staat, naar een andere plaats.
> GEBRUIK: RLOCMOB spritenummer, x-coordinaat van het eindpunt, y-coordinaat van het eindpunt, grootte, snelheid
>
> RLOCMOB mn,x,y,expansion,speed
> RLOCMOB 1,350,158,0,100
>
> OPMERKINGEN: de parameters hebben dezelfde betekenis als die bij MMOB. Het verschil tussen beide commando's is, dat bij RLOCMOB slechts twee coördinaten worden gebruikt. Daarom kan RLOCMOB met succes in een lus worden opgenomen.

AUTO

```
10 PRINT"[illegible]":COLOUR15,15
20 PRINT"[illegible]*****      S  P  R  I  T  E  S     *****"
25 PRINT"[illegible]"
30 PRINTSPC(15)"[illegible]               "
40 PRINTSPC(15)"[illegible]               "
50 PRINTSPC(15)"[illegible]               "
60 PRINT"[illegible]
90 DESIGN0,12288
100 @........................
110 @........................
120 @........................
130 @........................
140 @........................
150 @........................
160 @........................
170 @........................
180 @........................
190 @....BBBBBBBBBBB.........
200 @...BB....BB....B........
210 @..BB.....BB.....B.......
220 @.BBB.....BB......BBB..B.
230 @BBBBBBBBBBBBBBBBBBBBBBB.
240 @BBBBBBBBBBBBBBBBBBBBBBB.
250 @BBBBBBBBBBBBBBBBBBBBBBB.
260 @.BBBBBBBBBBBBBBBBBBBBBBB
270 @..BBBB............BBBB..
280 @..BBBB............BBBB..
290 @...BB..............BB...
```

1.4 Sprites (MOBs)

```
300 C........................
310 MOB SET 1,192,0,0,0
320 MMOB 1,40,158,40,158,0,100
330 PAUSE3
340 RLOCMOB1,350,158,0,100
350 PAUSE3:NRM
```

COMMENTAAR:
- SAVE het programma eerst op tape of disk.
- Regels 30-60 tekenen een weg met een obstakel.
- Regel 90 – het getal 12288 laat zien, dat de sprite op een tekstscherm komt.
- De sprite staat eerst even stil (regel 330) en gaat dan verder het scherm af door RLOCMOB in regel 340.

Door regel 310 te veranderen in:

310 MOB SET 1,192,0,1,0

gaat de auto achter het rode blok om.

Wijzig de volgende regels:

320 MMOB 1,40,138,40,138,3,100
340 RLOCMOB 1,350,138,3,100

Nu gaat een veel grotere auto over het scherm. Daartoe moet de y-coordinaat van de sprite in de regels 320 en 340 veranderd worden, en uiteraard ook het getal 0 in een 3.

5) BOTSINGEN TUSSEN SPRITES EN/OF TEKST

Om het constateren van botsingen mogelijk te maken moet eerst in het programma worden opgenomen:

'COMMANDO: DETECT

OMSCHRIJVING: maakt de computer gereed voor het signaleren van botsingen.
GEBRUIK: DETECT getal

DETECT 1

OPMERKINGEN: als achter DETECT een 0 wordt gezet, dan is de computer gereed om botsingen tussen sprites onderling te signaleren.
: het getal 1 achter DETECT moet worden gebruikt als botsingen tussen één of meer sprites en tekst of tekeningen op het scherm moeten worden geconstateerd
BIJZONDERHEDEN: het commando DETECT moet TWEE MAAL worden gebruikt in een programma:
de eerste keer om het 'botsingsregister' van de Commodore 64 (adres 53278 of 53279) leeg te maken en vervolgens om het registreren van een botsing mogelijk te maken.

De feitelijke botsing wordt geconstateerd met:

COMMANDO: CHECK

OMSCHRIJVING: controleert of een botsing heeft plaatsgevonden.
GEBRUIK: IF CHECK (spritenummer 1, spritenummer 2) = 0 THEN actie

: IF CHECK (1) = 0 THEN actie

OPMERKINGEN: Als er tussen de

1.4 Sprites (MOBs)

haakjes achter CHECK een 1 staat, dan wordt nagegaan of er een botsing tussen sprite en tekst/tekening heeft plaatsgevonden. (In de handleiding wordt abusievelijk een 0 vermeld).
BIJZONDERHEDEN: Het CHECK commando moet in een lus worden opgenomen. In deze lus wordt de beweging van de sprite(s) gerealiseerd door het commando RLOCMOB (zie aldaar).

De mogelijkheden van DETECT met CHECK ziet u in de volgende voorbeelden:
BOTSING SPRITE – SPRITE

```
10 PRINT"♥":COLOUR15,15
20 PRINT"■*****     T W E E   S P R I T E S   *****"
30 PRINT"QQQQQQQQQQQQQQQQQ"
60 PRINT"■■――――――――――――――――――――――――――――――――――――――
90 DESIGN 0,12288
100 @........................
110 @........................
120 @........................
130 @........................
140 @........................
150 @........................
160 @........................
170 @........................
180 @........................
190 @....BBBBBBBBBBB.........
200 @...BB....BB....B........
210 @..BB.....BB.....B.......
220 @.BBB.....BB......BBBBBB.
230 @BBBBBBBBBBBBBBBBBBBBBBB.
240 @BBBBBBBBBBBBBBBBBBBBBBB.
250 @BBBBBBBBBBBBBBBBBBBBBBBB
260 @.BBBBBBBBBBBBBBBBBBBBBBB
270 @..BBBB............BBBB..
280 @..BBBB............BBBB..
290 @...BB..............BB...
300 @........................
310 MOB SET 1,192,0,0,0
390 DESIGN 0,12352
400 @........................
410 @........................
420 @........................
430 @........................
440 @........................
450 @........................
460 @........................
470 @........................
480 @........................
490 @........................
500 @.....BBBBBBBBBBB........
510 @.....BBBBBBBBBBB........
520 @.....BBBBBBBBBBB........
```

1.4 Sprites (MOBs)

```
530 @.....BBBBBBBBBBB........
540 @.....BBBBBBBBBBB........
550 @.....BBBBBBBBBBB........
560 @.....BBBBBBBBBBB........
570 @.....BBBBBBBBBBB........
580 @.....BBBBBBBBBBB........
590 @.....BBBBBBBBBBB........
600 @.....BBBBBBBBBBB........
610 DETECT 0
620 MMOB 1,0,158,0,158,0,0
630 MOB SET 2,193,2,0,0
640 MMOB 2,250,158,250,158,0,0
650 FORT=0TO350
660 RLOCMOB1,T,158,0,100
670 DETECT 0:IFCHECK(2,1)=0THEN1000
680 NEXT
690 PAUSE2:NRM:END
1000 REM ACTIE
1010 RLOCMOB1,0,0,0,100:MOB OFF 1
1020 PAUSE2:NRM:END
```

COMMENTAAR:
- SAVE dit programma op tape of disk.
- Regel 610: eerste maal DETECT ledigt het botsingsregister.
- Regel 310/630 – spritenummers 1 en 2 in blok 192 en 193.
- Regel 640 – sprite 2 wordt vast opgesteld, de x (X1) en de y (Y1) coordinaten zijn dezelfde.
- Regels 650/680: de lus met RLOCMOB, DETECT en CHECK. Telkens als de sprite zich een pixel heeft verplaatst wordt gecontroleerd of er een botsing heeft plaatsgevonden. Is dat het geval, dan springt het programma naar een subroutine in regel 1000 en verder.

BOTSING SPRITE – TEKST/TEKENING

```
10 PRINT"▚":COLOUR15,15
20 PRINT"▒▒▒▒▒▒****     S P R I T E S    ****"
30 PRINT"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
40 PRINTSPC(25)"■A"
50 PRINT"▒▒────────────────────────────────────────
90 DESIGN0,12288
100 @........................
110 @........................
120 @........................
130 @........................
140 @........................
150 @........................
160 @........................
170 @........................
180 @........................
190 @....BBBBBBBBBBB.........
200 @...BB....BB....B........
```

1.4 Sprites (MOBs)

```
210 @..BB.....BB.....B.......
220 @.BBB.....BB......BBB..B.
230 @BBBBBBBBBBBBBBBBBBBBBBB.
240 @BBBBBBBBBBBBBBBBBBBBBBB.
250 @BBBBBBBBBBBBBBBBBBBBBBB.
260 @.BBBBBBBBBBBBBBBBBBBBBBB
270 @..BBBB............BBBB..
280 @..BBBB............BBBB..
290 @...BB..............BB...
300 @........................
310 MOB SET 1,192,0,0,0
320 MMOB 1,0,138,0,138,3,0
330 DETECT 1
340 FORA=0TO350
350 DETECT 1:IFCHECK(1)=0THEN1000
360 RLOCMOB1,A,138,3,100
370 NEXT
380 NRM:"■GEEN ACTIE!":END
1000 REM ACTIE
1010 MOB SET1,192,2,0,0
1020 RLOCMOB1,0,158,0,100
1030 PAUSE2:NRM:END
```

COMMENTAAR:
– Regel 40 – de letter A wordt als obstakel op de weg geplaatst.
– Regel 330 eerste maal DETECT, en wel met het getal 1. Het gaat hier om een botsing tussen sprite en tekst.
– Regels 340/370 de lus met DETECT 1, CHECK en RLOCMOB.
Wordt de botsing geconstateerd, dan volgt de subroutine op regel 1000.
– Regel 1010 – de kleur van de sprite is nu rood door de 2 achter MOB SET. Ook krijgt de sprite zijn normale grootte.
– Regel 1020 – de sprite staat al op het scherm bij de letter A, RLOCMOB is nu het juiste commando voor een verdere beweging.

Het gebruik van DETECT en CHECK geeft allerlei mogelijkheden bij het samenstellen van spellen. U hebt nu de gereedschappen om zoiets te realiseren.

MULTICOLOUR SPRITES

Alles wat voor de hires sprite geldt

1.4 Sprites (MOBs)

(adressen, beweging, constateren van botsingen e.d.) gaat ook voor de multicolour sprite op.
Er zijn natuurlijk wel een paar verschillen:

MULTICOLOUR SPRITE

21 rijen van 12 pixels
DESIGN 1, adres
3 kleuren
MOB SETmb,blk,col,pr,1

HIRES SPRITE

21 rijen van 24 pixels
DESIGN 0, adres
1 kleur
MOB SETmb,blk,col,pr,0

Omdat er bij de multicolour sprite drie kleuren gebruikt kunnen worden, heeft Simons' BASIC het speciale

COMMANDO: CMOB

OMSCHRIJVING: het vastleggen van twee extra kleuren voor een multicolour sprite.
GEBRUIK: CMOB kleur1, kleur2

CMOB c1,c2
CMOB 5,7

OPMERKINGEN: alle kleuren (0-15) kunnen worden gebruikt.
c1 is de kleur voor de letter B.
c2 is de kleur voor de letter D.
Deze letters en ook nog de C (deze kleur wordt gekozen met de derde parameter 'col' achter MOB SET), worden gebruikt bij het ontwerpen van een multicolour sprite. Zie de volgende programma's.

MULTICOLOUR SPRITEMAKER

```
100 REM MULTI COLOUR SPRITEMAKER
110 PRINT"":COLOUR3,15
120 PRINT"* MULTI COLOUR SPRITE MAKER *
130 PRINT"           DOOR
140 PRINT"         G.N.M.MERZ
150 PRINT"           1985
160 TE=0
200 PAUSE3
210 PRINT"":COLOUR3,3
220 PRINT""
390 PRINT"":PRINT"U KRIJGT EEN RASTER TE ZIEN VAN
400 PRINT"21 REGELS MET ELK 12 PUNTEN.
410 PRINT"DIT IS DE GROOTTE VAN EEN
415 PRINT"MULTI COLOUR SPRITE.
420 PAUSE10
430 PRINT"MET DE CURSOR TOETSEN KUNT U DE CURSOR
440 PRINT"OP ELKE GEWENSTE PUNT IN EEN  REGEL
450 PRINT"ZETTEN.ALS U DAN OP TOETS 'B','C' OF
460 PRINT"'D' DRUKT KOMT ER IN DE PLAATS VAN DE .
470 PRINT"EEN B, C OF D. C STELT EEN PUNT VOOR IN
480 PRINT"DE HOOFDKLEUR, B EEN PUNT IN DE EERSTE
490 PRINT"EN D EEN PUNT IN DE TWEEDE EXTRA
495 PRINT"KLEUR VOOR.
500 PAUSE10
```

1.4 Sprites (MOBs)

```
510 PRINT"   ALS U EEN ANDERE REGEL WILT INVULLEN
520 PRINT"    DRUK DAN EERST OP  RETURN "
530 PAUSE5
540 PRINT"ALS U ALLE REGELS HEBT BEWERKT,TYP DAN:
550 PRINT"  RUN 600  EN DRUK OP RETURN.
560 PRINT"U ZIET DAN DE SPRITE DIE U GEMAAKT HEBT.
570 PRINT" DRUK OP EEN TOETS "
580 GETA$:IFA$=""THEN580
590 GOTO1230
600 GOTO960
960 PRINT""
970 IFTE=1THEN1300
980 TE=1
990 DESIGN1,16320
1000 @............
1010 @............
1020 @............
1030 @............
1040 @............
1050 @............
1060 @............
1070 @............
1080 @............
1090 @............
1100 @............
1110 @............
1120 @............
1130 @............
1140 @............
1150 @............
1160 @............
1170 @............
1180 @............
1190 @............
1200 @............
1210 PRINT""
1220 IFTE=1THEN1300
1230 PRINT"":LIST1000-1200
1300 PRINT""
1305 CMOB5,7
1310 MOB SET0,255,0,0,1
1320 MMOB0,30,180,30,180,0,255
1330 PRINT" F1 -SPRITE BEWEEGT "
1340 PRINT" F3 -SPRITE 2 MAAL BREDER"
1350 PRINT" F5 -SPRITE 2 MAAL HOGER"
1360 PRINT" F7 -SPRITE 2 MAAL HOGER EN BREDER."
1370 PRINT" F2  (SHIFT/F1) - EINDE.
1380 A=INKEY
1390 ONAGOSUB1500,1600,1700,1800,1900,2000,2100,2200
1400 GOTO1380
1410 END
1500 REM SPRITE BEWEEGT NAAR RECHTS
```

1.4 Sprites (MOBs)

```
1510 FORP=60TO340
1520 RLOCMOB0,P,180,0,100
1530 NEXTP
1540 RLOCMOB0,60,180,0,50
1550 RETURN
1600 REM EINDE PROGRAM
1610 MOB OFF 0
1620 PRINT"{CLR}{DOWN}WILT U DE GEGEVENS VAN DE ZOJUIST
1630 PRINT"GEMAAKTE SPRITE NOG EENS BEKIJKEN OF
1640 INPUT"MISSCHIEN OP PAPIER ZETTEN? JA/NEE";A$
1650 IF LEFT$(A$,1)="J"THENLIST1000-1200
1660 IF LEFT$(A$,1)="N"THENPRINT"{CLR}"
1670 PRINT"■{DOWN}{DOWN}{DOWN}ALS U EEN NIEUWE SPRITE WILT MAKEN
1680 PRINT"DAN MOET U ALLE B´S,C´S EN D´S IN HET
1690 PRINT"ROOSTER WEER VERANDEREN IN ......
1695 PRINT"{DOWN}{DOWN}{DOWN}{DOWN}V E E L   S U C C E S !":TE=0:END
1700 REM SPRITE 2 MAAL BREDER
1710 MMOB0,60,180,60,180,1,255
1720 PAUSE5
1730 MMOB0,60,180,60,180,0,255
1740 RETURN
1900 REM SPRITE 2 MAAL HOGER
1910 MMOB0,60,180,60,180,2,255
1920 PAUSE5
1930 MMOB0,60,180,60,180,0,255
1940 RETURN
2100 REM SPRITE 2 MAAL BREDER EN HOGER
2110 MMOB0,60,180,60,180,3,255
2120 PAUSE5
2130 MMOB0,60,180,60,180,0,255
2140 RETURN
```

COMMENTAAR:

- SAVE het programma na intypen eerst op cassette of disk.
- Op het scherm komt na RUN een beschrijving van het programma. Het komt in grote trekken overeen met het programma SPRITEMAKER.
- De multicolour sprite kan nu in het werkrooster worden ontworpen met de letters B, C of D.
- B is een punt van de sprite in de eerste kleur (c1) na het commando CMOB.
- C is een punt van de sprite in de kleur vermeld achter MOB SET.
- D is een punt van de sprite in de tweede kleur (c2) na het commando CMOB.
- Geef na het ontwerpen van de multicolour sprite RUN 600 en druk op RETURN. Dan komt op het scherm een menu en daaronder de zojuist gemaakte sprite. Met de functietoetsen kunnen beweging en grootte worden bekeken. Wacht na het gebruik van F3, F5 of F7 even, de sprite krijgt zijn oorspronkelijke grootte weer terug.

De data voor de sprite kan op papier worden overgenomen of met een printer worden afgedrukt.

Het verdient aanbeveling de listing van dit programma te bestuderen; het ge-

1.4 Sprites (MOBs)

bruik van de verschillende commando's zal daardoor worden verduidelijkt. Met OPTION 10 worden de Simons' BASIC commando's duidelijk zichtbaar (zie 8/1.3 blz. 6).

GEANIMEERDE SPRITES

In voorgaande programma's werd een sprite over het scherm bewogen, maar de sprite zelf bleef onveranderd.
Het is ook mogelijk de sprite zelf te animeren. Daarvoor zijn minstens twee sprites nodig, die iets van elkaar verschillen. De beweging is gebaseerd op het principe van de (teken)film: er wordt een plaatje getoond, het verdwijnt en er wordt een iets verschillend plaatje getoond, enz. als dit verwisselen snel genoeg gebeurt ontstaat voor het oog een vloeiende beweging.

Vertaald in sprites gaat dat zo:
Op het beeldscherm wordt een sprite getoond, hij verdwijnt, iets later komt een tweede sprite op het scherm, even verder dan waar de eerste stond. Deze verdwijnt ook weer waarna het proces wordt herhaald. Met twee sprites is de bewegingsillusie nog wel wat houterig, maar dat kan met drie of meer sprites worden verbeterd.

EEN BEWEGENDE SPRITE

```
10 REM BEWEGENDE MULTICOLOUR SPRITES
20 HIRES13,12:COLOUR12,12
30 LINE0,120,319,120,1
40 LOW COL2,12,0:BLOCK40,88,100,119,1
50 LOW COL10,12,0:LINE40,87,50,70,1:LINE50,
   70,90,70,1:LINE90,70,100,87,1
60 LINE100,87,50,87,1:PAINT80,78,1
70 LOW COL 9,12,0
80 FORX=120TO300STEP40:BLOCKX,88,X+3,112,1:NEXT
90 LOW COL 5,12,0:FORX=120TO300STEP40:
   CIRCLEX,72,12,15,1:PAINTX,72,1:NEXT
100 DESIGN1,65472
110 @....CCC.....
120 @...CBBBC....
130 @...B.BBB....
140 @...BBBB.....
150 @....BBB.....
160 @....BBB.....
170 @BB..CCCC....
180 @BB..CCCCC...
190 .CCCCCCC.CC..
200 @..CCCCCCC...
210 @...CCCCC....
220 @...DDDD.....
230 @...DDDD.....
240 @.DDDDDDD....
250 @DDD...DD..CC
260 @DDD...DD.CCC
270 @DD....DDDD..
280 @DD.....DD...
290 @CCC.........
```

1.4 Sprites (MOBs)

```
300 @............
310 .............
320 DESIGN 1,65408
330 @....CCC.....
340 @...CBBBC....
350 @...B.BBB....
360 @...BBBB.....
370 @....BBB.....
380 @....BBB.....
390 @....CCCC....
400 @B...CCCCC...
410 @BC.CCCC.CC..
420 @.CCCCCC..CC.
430 @...CCCC.BB..
440 @...CCCC.B...
450 @...DDDD.....
460 @...DDDD.....
470 @..DD.DD.....
480 @..DD.DD.....
490 @..DD.DD.....
500 @..DD.DD.....
510 @...DD.DD....
520 @..CCC.CC....
530 @............
540 CMOB10,7
550 MOB SET1,255,0,0,1
560 MOB SET2,254,0,0,1
570 X=350
580 MMOB1,X,152,X,152,0,100
590 MMOB2,X,152,X,152,0,100
600 MOB SET1,255,0,0,1:RLOCMOB1,X-2,152,0,255:MOB OFF 1
610 MOB SET2,254,0,0,1:RLOCMOB2,X-4,152,0,255:MOB OFF 2
620 X=X-6:IF X>6THEN600:MOB OFF 1:MOB OFF 2
630 PAUSE4
```

COMMENTAAR:
– SAVE het programma na intypen eerst op cassette of disk.
– Regels 30/90 tekenen een achtergrond.
– De sprite bevat de kleuren lichtrood (10), zwart (0) en geel (7).
– Regels 600/620 – de bewegingslus. Het op het scherm komen en weer verdwijnen van de twee sprites wordt gerealiseerd door: MOB SET en MOB OFF.

8/1.5

Letters en tekens.

ZELF ONTWORPEN LETTERS EN TEKENS.
De letters, cijfers en tekens die we met het toetsenbord op het scherm kunnen brengen zijn als blokken van 8 bij 8 pixels opgeslagen in het ROM. Dit is een deel van het Commodore 64 geheugen, dat alleen kan worden uitgelezen, maar niet kan worden gewijzigd. Voor elke letter of ander teken is dus een geheugenruimte van 8 bytes vereist. Als we met het toetsenbord andere lettervormen of speciale tekens willen produceren, dan zullen we de gegevens daarvoor ergens anders vandaan moeten halen. En wel uit het RAM, want dat deel van het geheugen kan wel worden gewijzigd.
Ons eerste werk is: de hele letterset overbrengen van ROM naar RAM. Dat doet het

COMMANDO: MEM

OMSCHRIJVING: brengt de letterset over van ROM naar RAM.
GEBRUIK: MEM
OPMERKINGEN: Dit commando heeft geen parameters.
BIJZONDERHEDEN: Als het commando MEM in een programma wordt gebruikt, dan heeft dat invloed op de bruikbare geheugenruimte voor eventuele sprites in dat programma. Nu mogen alleen bloknummers 192 of hoger voor sprites worden gebruikt.

Het gebruik van zelfgemaakte letters en tekens blijft beperkt tot het tekstscherm, op een tekenscherm kunnen ze niet worden toegepast.

Om terug te keren tot de normale letterset moet de RUN/STOP-RESTORE toets-combinatie gebruikt worden.

Alvorens we een letter gaan ontwerpen moeten we er eerst een plaats in het geheugen voor inruimen.

Daartoe dient het

COMMANDO: DESIGN

OMSCHRIJVING: geeft aan, welke letter of welk teken wordt vervangen door een zelf gemaakte figuur.

GEBRUIK: DESIGN 2, beginadres geheugendeel + scherm-code van de te vervangen letter * 8

DESIGN 2,57344 + ch * 8

OPMERKINGEN: het beginadres van het lettergeheugen is $E000 hexadecimaal.

ch is de scherm-code van de letters: A=1, B=2, C=3 enz. (zie appendix 10/1).

1.5 Letters en tekens.

Het getal 8 wordt gebruikt omdat voor elk teken 8 bytes geheugenruimte vereist is.
BIJZONDERHEDEN: Het commando DESIGN is ook in gebruik bij de sprites.

DESIGN 0 = hires sprite
DESIGN 1 = multi colour sprite
DESIGN 2 = zelf gemaakte letter.

Het ontwerpen van letters of tekens gebeurt, zoals bij de sprites in een werkrooster.
Zo'n rooster bestaat uit 8 regels van 8 punten, elke regel begint met het @ teken. Door op de plaats van een punt een B te zetten kan elk gewenst teken worden gemaakt.

ANDERE LETTERS.

```
10 REM ANDERE KARAKTERS
20 MEM
30 DESIGN2,57344+1*8
140 @....B...
150 @...B..B.
160 @..B...B.
170 @.B....B.
180 @.B....B.
190 @..B..BB.
200 @...BB..B
210 @........
220 DESIGN2,57344+2*8
230 @..BBB...
240 @.B...B..
250 @.B..B...
260 @.B.B....
270 @.B..BB..
280 @.B....B.
290 @.B...B..
300 @....B...
310 DESIGN2,57344+3*8
320 @....BB..
330 @...B..B.
340 @..B....B
350 @.B....B.
360 @.B......
370 @..B.....
380 @...B....
390 @....BB..
400 DESIGN2,57344+4*8
410 @BBBB....
420 @..B.B...
430 @..B..B..
440 @..B...B.
450 @..B...B.
460 @..B...B.
470 @.BBBBB..
480 @........
```

COMMENTAAR:
- In dit programma worden de letters A, B, C, en D (POKE codes 1, 2, 3 en 4) door andere vervangen. Deze vervangende letters komen op het scherm steeds als er een van de toetsen A, B, C of D wordt gebruikt. De andere toetsen geven de normale letters en tekens.

Er moet dus veel werk worden verricht om een compleet nieuw alfabet te realiseren. Als slechts enkele nieuwe tekens gebruikt moeten worden, dan is het raadzaam daarvoor weinig gebruikte toetsen als de vierkanten haken of het pond-teken (POKE codes resp. 27, 29 en 28) te reserveren. U kunt elke gewenste tekening ontwerpen: de enige beperking is, dat die tekening in een rooster van 8 bij 8 punten moet passen. Voor grotere tekeningen kunnnen letters gecombineerd worden.

TEKST OP HET TEKENSCHERM.

De zelf gemaakte letters of tekens kunnen niet op het tekenscherm worden gebruikt, maar de standaard letters en tekens wel!
Om een enkel teken of cijfer op een hiresscherm te plaatsen gebruikt men het

COMMANDO: CHAR

OMSCHRIJVING: zet EEN letter of

1.5 Letters en tekens.

teken op een hires-scherm.
GEBRUIK: CHAR x-coördinaat, y-coördinaat, scherm code (POKE code), plot type, grootte

CHAR x,y,POKE code,plot type, size
CHAR 40,100,24,1,4

OPMERKINGEN: De coördinaten zijn die van de linkerbovenhoek van het 8 bij 8 rooster. De schermcode (A=1, B=2, enz) wordt gegeven in appendix 10/1. Voor plot type: zie 8/1.3 blz. 2.
De grootte kan variëren van 1 t/m 8. De 1 geeft normale hoogte van het teken, d.w.z. 8 pixels. Een grootte van 3 betekent een hoogte van 3 maal 8 = 24 pixels. Helaas kan de breedte van de letter niet worden gewijzigd. Grootte 8 geeft daarom een lang gerekte letter.

LETTERS OP HET TEKEN-SCHERM.

```
10 REM TEXT3
20 HIRES0,3
30 TEXT60,50,"[A]V[B]AN [A]B[B]ASIC TOT
   [A]M[B]ACHINETAAL",1,4,8
40 TEXT0,100,"[A]E[B]R GAAN OP EEN
   REGEL WEL VEERTIG LETTERS"
   ,1,1,8
50 PAUSE5:NRM
```

Waarschijnlijk zal van dit commando CHAR niet zoveel gebruik worden gemaakt. Meestal wil men enige woorden of een korte zin op het tekenscherm zetten, als verklaring van een tekening.
Dat kan gebeuren met het

COMMANDO: TEXT

OMSCHRIJVING: zet een string, bestaande uit letters, cijfers en/of tekens op het tekenscherm.
GEBRUIK: TEXT X-coördinaat eerste letter van de string, Y-coördinaat van die letter, '(controle letter) string', plot type, hoogte van de letters, afstand tussen letters onderling.

TEXT x,y,"(CTRL A/B) character string", plot type,s,i
TEXT 10,160,"(CTRL A)W*E*K-*A",1,2,8

OPMERKINGEN: zoals bij CHAR zijn X en Y de coördinaten van de linkerbovenhoek van het 64-punt rooster. Als de tekst alleen uit hoofdletters bestaat, dan moet na de aanhalingstekens eerst CTRL A worden getypt, d.w.z. de toets CTRL ingedrukt houden en een A typen. Wil men alleen kleine letters in de tekst hebben, dan dient met CTRL B te typen. Bestaat een tekts uit hoofd- en kleine letters, dan voor elke (groep) hoofd letter(s) CTRL A typen en voor elke (groep) kleine letter (s) CTRL B. Voor de betekenis van 'plot type', zie 8/1.3 blz. 2.
De hoogte van de letters in de tekst kan variëren van 1 t/m 8. De breedte van de letters kan niet worden veranderd.

De normale onderlinge afstand tussen de letters wordt verkregen door het getal 8. Met een getal grotere dan 8 komen de letters verder uiteen te staan. Gebruik geen kleiner getal dan 8, anders komt de ene letter op de andere te staan.

De volgende programma's geven een indruk van de mogelijkheden.

VERSCHILLENDE MOGELIJKHEDEN MET TEKST OP EEN TEKENSCHERM.

```
10 REM TEXT2
15 FORX=8TO48STEP8
20 HIRES0,3
30 TEXT0,100,"■W*E*K*A",1,8,X
40 PAUSE2
50 NEXTX
60 PAUSE3:NRM
```

```
10 REM TEXT1
15 FORX=1TO8
20 HIRES0,3
30 TEXT140,100,"■W*E*K*A",1,X,8
40 PAUSE2
50 NEXTX
60 PAUSE3:NRM
```

```
10 REM CHARACTERS
20 FORX=0TO5:FORY=1TO8
30 HIRES0,3
40 PLOT40,100,1
50 CHAR 40,100,X,1,Y
60 PAUSE1
70 NEXTY:NEXTX
```

8/1.6

Multi colour graphics mode

De titel zegt het al: Tekenen met veel kleuren. Dat gebeurt normaal ook in de HIRES mode, maar daar is het gebruik van kleuren beperkt, zeker als verschillende kleuren aaneengrenzen. De kleurenblokken van 8 bij 8 pixels (zie hoofdstuk 10/3.1) vormen dan een belemmering.
In MULTI COLOUR MODE is dat bezwaar in mindere mate aanwezig, hoewel er in uiterste gevallen toch rekening mee gehouden moet worden. Vier kleuren kunnen nu dicht op elkaar worden gebruikt.

KLEUREN IN HIRES EN MULTI

```
20 LOW COL 0,7,0
30 BLOCK10,10,50,50,1
40 LINE10,10,160,160,1
50 PAUSE5
60 HIRES0,3:MULTI0,5,7:
   COLOUR3,3
70 BLOCK10,10,50,50,1
80 LINE10,10,140,140,3
90 LINE10,20,140,150,2
100 PAUSE5:NRM
```

COMMENTAAR: Het eerste deel van het programma is in HIRES mode, onjuist gebruik van kleuren levert 'kleurblokken' op. Het tweede deel is in multi colour mode, hier geven de kleurblokken geen problemen.
Helaas moet voor de winst in kleuren iets worden opgeofferd: elke punt in multi colour mode bestaat uit 2 pixels naast elkaar. In HIRES mode is een punt slechts 1 pixel groot. Daar is een tekening dus wat fijner gedetailleerd. Voor meer details hieromtrent wordt verwezen naar hoofdstuk 10/2.2 blz. 1.

MULTI COLOUR MODE wordt ingeschakeld door na HIRES het
COMMANDO: MULTI
te gebruiken.
OMSCHRIJVING: schakelt multi colour mode in en kiest drie tekenkleuren.
GEBRUIK: MULTI kleur 1, kleur 2, kleur 3
MULTI c1, c2, c3
MULTI 0,4,7
OPMERKINGEN: Eerst moet het commando HIRES worden gebruikt, daarna (vaak in dezelfde regel) MULTI
BIJZONDERHEDEN: PLOT TYPE krijgt na MULTI een andere betekenis. Zie 8/1.3 blz. 2.
In LOW COL hebben nu alle drie parameters een betekenis. Met LOW COL kunnen drie andere kleuren worden gekozen, zie 8/1.3 blz. 11.
Om na LOW COL terug te keren tot de oorspronkelijke kleuren achter MULTI wordt het
COMMANDO: HI-COL
gebruikt.
OMSCHRIJVING: brengt het pro-

1.6 Multi colour graphics mode

gramma terug naar de kleuren achter HIRES of MULTI vermeld, na gebruik van het commando LOW COL.
GEBRUIK: HI COL
OPMERKINGEN: HI COL heeft geen parameters.
Hoofdzaak in multi colour mode is dat elke punt nu uit twee pixels naast elkaar bestaat en dat heeft nogal wat gevolgen:
1) vooral de vertikale lijnen van een tekening worden breder;
LIJNEN IN HIRES EN MULTI

```
10 COLOUR3,3
20 HIRES0,3
30 LINE2,25,319,25,1
40 PAUSE3
50 HIRES0,3:MULTI0,4,6
60 LINE2,25,159,25,1
70 PAUSE3
80 HIRES0,3
90 LINE160,0,160,199,1
100 PAUSE3
110 HIRES0,3:MULTI0,4,6
120 LINE80,0,80,199,1
130 PAUSE3
```

COMMENTAAR: Eerst komt in hires een vertikale, daarna een horizontale lijn op het scherm. Dan gebeurt het zelfde in multi colour mode. Let op het verschil in dikte van de vertikale lijnen.
2) cirkels en bogen worden ook dikker getekend. Maar waar vooral op gelet moet worden: voor een cirkel is de verhouding tussen de X-straal en de Y-straal niet langer 1:1 maar 1:2.
CIRKELS IN HIRES EN MULTI

```
10 COLOUR3,3
20 HIRES0,3
30 CIRCLE80,100,40,40,1:
   CIRCLE240,100,40,40,1
40 TEXT10,160,"HI-RES",1,1,8
50 PAUSE3
60 HIRES0,3:MULTI0,4,6
70 CIRCLE40,100,40,40,1:
   CIRCLE120,100,20,40,1
80 TEXT10,160,"MULTI",1,1,8
90 PAUSE3:NRM
```

COMMENTAAR: regel 70 – als de stralen gelijk zijn ontstaat er in multi colour mode een ellips. Bij een verhouding X-straal : Y-straal = 1:2 ontstaat er een cirkel.
3) tekens en teksten met CHAR of TEXT op het scherm gezet worden breder; hieraan wordt nog apart aandacht besteed.
4) het werkblad, waarop tekeningen worden ontworpen, krijgt een andere indeling. Inplaats van 320 bij 200 pixels wordt de grootte van het tekenscherm nu 160 bij 200 pixels. De getallen op het werkblad lopen horizontaal van 0-159 en vertikaal van 0-199; een voorbeeld wordt bijgevoegd.
5) er kunnen nu ten hoogste 4 kleuren (achtergrondkleur inbegrepen) vlak bij elkaar worden gebruikt;
VERSCHILLENDE KLEUREN BIJ ELKAAR.

```
10 COLOUR3,3
20 HIRES0,3:MULTI0,7,4
30 BLOCK 40,40,120,120,1
50 LINE120,40,40,120,2:LINE40,
   120,120,120,2:LINE120,120,
   120,40,2
60 PAINT110,110,3
70 PAUSE3
80 LOW COL 0,1,2
90 LINE40,40,40,110,2:LINE40,
   110,110,40,2:LINE110,40,40,
   40,2
100 PAINT45,45,3
110 PAUSE3
```

6) de tot nu toe besproken teken-commando's blijven in multi colour mode van kracht, alleen moeten enkele para-

1.6 Multi colour graphics mode

meters worden aangepast. Dit geldt in het bijzonder voor PLOT TYPE parameters en X- en Y-parameters.
7) PLOT TYPE achters commando's als PLOT, LINE, REC enz. krijgt in multi colour mode een andere betekenis. Zie deel 8/1.3 blz. 2.
8) bij LOW COL telt in multi mode ook de derde parameter mee;
9) CSET2 kan nog steeds worden gebruikt om een tekening onmiddellijk op het scherm terug te krijgen. Wel moet daarna het commando MULTI worden geplaatst met de drie kleuren daarachter. Bijv.:
CSET 2:MULTI 0,4,7: PAUSE 10 (RETURN)
Zijn er in de tekening meer dan die drie kleuren gebruikt, dan komen alleen de eerste drie kleuren op het scherm na CSET2.
De twee volgende programma's zijn zeer geschikt om in kort bestek de mogelijkheden van de multi colour mode te laten zien.

De X-coordinaten kunnen niet hoger zijn dan 159. Let vooral op het gebruik van de plot type parameters.

VLAGGEN1

```
10 HIRES1,3:MULTI1,2,6:
   COLOUR15,15
20 BLOCK0,0,60,60,1
30 BLOCK0,0,60,20,2
40 BLOCK0,40,60,60,3
50 TEXT0,62,"NEDERLAND",1,1,8
60 LOW COL 0,2,7
70 BLOCK 90,0,150,60,1
80 BLOCK 90,0,110,60,2
90 BLOCK 130,0,150,60,3
100 TEXT90,62,"BELGIE",1,1,8
110 LOW COL1,2,6
120 BLOCK0,80,60,140,3
130 BLOCK15,80,25,140,1:BLOCK0,
    105,60,115,1
140 BLOCK17,80,23,140,2:BLOCK0,
    107,60,113,2
150 TEXT0,150,"IJSLAND",1,1,8
160 LOW COL 0,5,7
170 BLOCK90,80,150,140,1
180 LINE90,135,140,80,3:
    PAINT92,82,3
190 LINE90,131,136,80,2:
    PAINT92,82,2
200 LINE100,140,150,85,3:
    PAINT102,139,3
210 LOW COL 0,6,7
220 LINE104,140,150,89,2:
    PAINT148,138,2
230 TEXT90,150,"TANZANIA",1,1,8
240 PAUSE5:NRM
```

VLAGGEN 2.

```
10 HIRES0,15:MULTI1,5,7:COLOUR15,15
20 BLOCK0,0,60,60,2
30 LINE0,0,60,30,1:LINE60,30,0,60,1:PAINT5,30,1
40 LINE0,4,52,30,3:LINE52,30,0,56,3:PAINT5,30,3
50 LOW COL 0,2,7
60 LINE0,8,30,30,1:LINE30,30,0,52,1:PAINT5,30,1
70 LINE0,10,26,30,2:LINE26,30,0,50,2:PAINT5,30,2
80 TEXT0,64,"GUYANA",1,1,8
90 LOW COL 1,2,6
100 BLOCK 80,0,140,58,1
110 FORY=0TO60STEP9:BLOCK80,Y,140,Y+5,2:NEXT
120 BLOCK80,0,106,30,3
130 X=83
140 FORY=3TO28STEP6:PLOTX,Y,1:NEXT
```

1.6 Multi colour graphics mode

```
150 X=X+4:IFX<106THEN140
160 X=85
170 FORY=6TO24STEP6:PLOTX,Y,1:NEXT
180 X=X+4:IFX<104THEN170
190 TEXT80,64,"U.S.A.",1,1,8
200 LOW COL 0,5,7
210 BLOCK0,80,60,140,3
220 LINE0,82,25,110,1:LINE25,110,0,138,1:PAINT5,110,1
230 LINE60,82,35,110,1:LINE35,110,60,138,1:PAINT55,110,1
240 LINE6,80,30,105,2:LINE30,105,54,80,2:PAINT30,82,2
250 LINE6,140,30,115,2:LINE30,115,54,140,2:PAINT30,138,2
260 TEXT0,152,"JAMAICA",1,1,8
270 LOW COL0,1,2
280 BLOCK 80,80,140,140,3
290 LINE80,80,120,140,2:LINE100,80,140,140,2:PAINT90,90,2
300 LINE84,80,124,140,1:LINE96,80,136,140,1:PAINT88,82,1
310 TEXT80,152,"TRINIDAD",1,1,8
320 PAUSE10:NRM
```

Nog zo'n programma.
De vlag van Guyana vertoont een paar ongewenste zwarte vlekken. Ook in multi colour mode moet men blijkbaar voorzichtig zijn met kleuren te dicht op elkaar.
De sterren en strepen in de vlag van de USA worden met lussen getekend.

Dan nog even aandacht besteed aan het gebruik van de commando's CHAR en TEXT in multi colour mode.
Het breder worden van de punten maakt het mogelijk om zelf gemaakte programma's van kleurige titels te voorzien. Met wat fantasie kunnen prima resultaten worden bereikt.

TEKST IN MULTI.

```
10 REM LETTERS
20 HIRES0,3:MULTI1,4,7
30 BLOCK0,0,159,140,2
40 BLOCK0,141,159,199,3
50 TEXT24,40,"W.E.K.A.",1,8,16
60 TEXT32,148,"[A]A[B]MSTERDAM.",
   1,5,10
65 PAUSE3:LOW COL1,2,5
70 BLOCK0,0,159,140,2:BLOCK0,
   141,159,199,3
80 TEXT36,10,"[A]V[B]AN [A]B[B]ASIC",
   1,6,10
90 TEXT68,80,"[B]TOT",1,4,10
100 TEXT16,148,"MACHINETAAL.",
    1,6,12
110 PAUSE5:NRM
```

COMMENTAAR: regel 60 – (CONTROL A) geeft hoofdletters, (CONTROL B) geeft kleine letters. Wordt dit niet gebruikt – zie regel 70 – ('default'-waarde), dan komen hoofdletters op het scherm.

TEKST IN EEN LUS

```
10 REM STEEDS GROTER
20 HIRES0,3:MULTI1,3,7:COLOUR3,3
30 U=8
40 FORX=1TO8
50 TEXT 4,100,"W.E.K.A.",1,X,U
55 TEXT4,130,"[A]A[B]MSTERDAM",1,X,U
60 U=U+4:PAUSE2:IFU<24THENBLOCK0,
   100,159,150,2:NEXTX
70 PAUSE5:NRM
```

COMMENTAAR: In dit programma zijn de parameters voor lettergrootte

1.6 Multi colour graphics mode

en letterafstand in lussen opgenomen. Hiermee zijn allerlei effecten te realiseren.
Er doet zich echter een vreemd verschijnsel voor bij het gebruik van lussen. Wanneer de string uit meer dan 8 letters en/of spaties bestaat dan worden er niet meer dan 8 letters op het scherm gezet. De string in regel 50 wordt in zijn geheel afgebeeld, maar van de string in regel 55 komen 8 letters op het scherm. De geïnverteerde A en B tellen als letter.

DE SPRINTER

```
10 REM SPRINTER
20 HIRES0,15:MULTI0,14,7
30 LINE159,136,50,136,1:LINE50,136,48,152,1:LINE48,152,32,152,1
40 LINE32,152,20,120,1:LINE20,120,20,80,1:LINE20,80,32,40,1:
   LINE32,40,40,32,1
50 LINE40,32,159,32,1:LINE30,48,159,48,1:LINE28,56,36,58,1:
   LINE36,58,32,88,1
60 LINE32,88,20,80,1
70 REC40,72,4,24,1:REC54,66,4,32,1:REC62,66,4,32,1:REC52,64,16,72,1
80 FORX=72TO128STEP28:RECX,64,20,24,1:LINEX,72,X+20,72,1:
   LINEX+10,64,X+10,72,1
90 NEXTX
100 REC156,64,3,24,1:LINE156,72,159,72,1:BLOCK24,104,27,112,1
110 PAINT112,112,3:PAINT60,112,3:LINE60,64,60,136,1
120 LINE36,96,36,136,1:LINE48,96,48,136,1
130 LINE76,112,74,108,1:LINE74,108,76,104,1:LINE76,104,80,104,1
140 LINE80,104,82,108,1:LINE82,108,86,108,1:LINE86,108,84,104,1
150 LINE86,108,84,112,1:LINE84,112,80,112,1:LINE80,112,78,108,1
160 LINE78,108,74,108,1
170 CIRCLE58,146,6,10,1:PAINT58,146,1:CIRCLE74,146,6,10,1:
    PAINT74,146,1
180 LINE84,136,92,152,1:LINE92,152,159,152,1:BLOCK52,144,80,152,1
190 LINE0,156,159,156,1:LINE0,159,159,159,1:PAINT104,157,1
200 LINE100,48,105,64,2:LINE114,88,132,136,2:LINE120,48,128,70,2
210 LINE134,88,152,136,2:LINE128,48,134,64,2:LINE144,88,159,128,2
220 LINE148,48,156,68,2:PAINT124,96,2:PAINT156,96,2
230 LOW COL 5,9,12
240 X=4
250 BLOCKX,160,X+6,163,2
260 X=X+16:IFX<156THEN250
270 PAINT104,40,3:PAINT104,144,3
280 LINE0,108,4,98,1:LINE4,98,12,120,1:LINE12,120,16,124,1:
    LINE16,124,20,136,1
290 LINE20,136,0,136,1:PAINT8,120,1
300 LOW COL 0,1,13
310 TEXT12,176,"SPRINTER - NS ",1,1,8
320 PAUSE10:NRM
```

1.6 Multi colour graphics mode

COMMENTAAR:
Het programma geeft een indruk van wat er in multi colour mode zoal te tekenen valt.
Een werkblad is voor zulke tekeningen absoluut onmisbaar. Zie werkblad SPRINTER.
BELANGRIJK. Let op het ontstaan van de tekening. De omtrek en de ramen worden met zwarte lijnen getekend. Daarna wordt het ingesloten vlak geheel gekleurd.
Het is met Simons' BASIC dus mogelijk om de omtrek van een figuur te tekenen met een kleur en die figuur dan met PAINT op te vullen met een ANDERE KLEUR. Dit kan bij sommige andere computers niet.

8/1.7

Muziek

De Commodore 64 kent veel mogelijkheden om allerlei geluidseffecten te realiseren, om de klank van diverse muziekinstrumenten na te bootsen en zelfs om drie-stemmige muziek te laten horen.
Helaas dient dit weer te geschieden met allerlei POKEs op een aantal adressen en dit maakt het programmeren van muziek nogal bewerkelijk.
Bovendien is enige kennis van muziek, met name wat betreft notennamen, duur van noten, maatindelingen enz. onontbeerlijk om met enig succes muziek te kunnen produceren. Vooral dit laatste zal wel de reden zijn, waarom velen nog niet gekomen zijn tot het gebruiken van de muzikale capaciteiten van de computer.
Enkele begrippen die gebruikt worden bij het programmeren van muziek op de Commodore 64 zijn: volume, frequentie, driehoek, zaagtand, puls, ruis. Daar komen dan nog namen als attack, decay, sustain en release bij, samen afgekort als ADSR. Deze begrippen spelen in het vervolg van dit verhaal een belangrijke rol en daarom is het bestuderen van deel 11 van dit boek, in het bijzonder hoofdstuk 1 daarvan, absoluut vereist.
Simons' BASIC kent een aantal commando's die het programmeren van muziek aanzienlijk eenvoudiger maken. Wel wordt in de handleiding van Simons' BASIC uitdrukkelijk vermeld, dat deze commando's bedoeld zijn als inleiding in het programmeren van muziek. Wie alle mogelijkheden van de Commodore 64 op muziekgebied wil beheersen dient speciale software te gebruiken, welke in de handel verkrijgbaar is.
Om een reeks van geluiden te produceren dienen de volgende zaken te worden ingesteld:
1) de geluidssterkte warmee de klanken worden wergegeven. Dit heet VOLUME;
2) welk van de drie stemmen zal worden gebruikt: VOICE;
3) welke golfvorm het geluid heeft: DRIEHOEK, ZAAGTAND, PULS of RUIS. Deze golfvorm zou op een oscilloscoop zichtbaar gemaakt kunnen worden. Zie fig. 11/1-1: Golfvormen;
4) het ADSR-verloop, zie hoofdstuk 11;
5) de notennamen en hun tijdsduur. Voor dit onderdeel is kennis van muziek(schrift) vereist;
6) de commando's voor het hoorbaar maken van de geluiden.

1) De geluidssterkte wordt bepaald door het
COMMANDO: VOL
OMSCHRIJVING: bepaalt de geluidssterkte.

1.7 Muziek

GEBRUIK: VOL sterkte
VOL n
VOL 12
OPMERKINGEN: De sterkte kan worden aangegeven met de getallen 0 (geen geluid) tot en met 15 (sterkste geluid). Deze sterkte blijft gehandhaafd tot een nieuw VOL commando anders aangeeft.
2) Welk van de drie stemmen wordt gebruikt moet worden vermeld achter de commando's besproken in 3) en 4).
3) De golfvorm wordt gekozen met het
COMMANDO: WAVE
OMSCHRIJVING: kiest een van de stemmen, een golfvorm en bepaalt synchronisatie of ringmodulatie.
GEBRUIK: WAVE nummer van de stem, getal in het tweetallig stelsel van 8 cijfers 0 of 1.
WAVE voice number, binary number
WAVE 3,00010010
OPMERKINGEN: het nummer van de stem kan zijn: 1, 2 of 3. Het binaire getal bestaat uit 8 bits, die van RECHTS naar LINKS worden genummerd 0 t/m 7. Dit is dezelfde opbouw als die van het processor status register, zie hoofdstuk 9/2 blz. 2.
Zo'n bit kan alleen de waarden 0 of 1 hebben. Wil men de functie kiezen, die door een bit wordt bepaald, dan moet dat bit de waarde 1 hebben. Een niet gezet bit heeft de waarde 0. BIT 0 – wordt door het (nog te bespreken) commandop PLAY automatisch gezet. De waarde moet dus altijd 0 zijn;
BIT 1 – synchronisatie. Met een stem wordt een constante toon gemaakt en met een andere stem verschillende tonen, b.v. een melodie. Zo kunnen bepaalde effecten worden verkregen. De constante toon moet LAGER zijn dan de laagste van de melodie in de andere stem, de stem met de constante toon is dan afhankelijk van de melodiestem.
Dit zijn de mogelijkheden:

MELODIE IN	CONSTANTE TOON IN
Stem 1	Stem 3
Stem 2	Stem 1
Stem 3	Stem 2

Dit stemnummer is de eerste parameter achter het commando WAVE.
BIT 2 – ring modulatie. Geeft iets andere effecten dan synchronisatie. De melodie echter moet gespeeld worden met een driehoek golfvorm. Ook hier kunnen alleen de combinaties van stemmen gebruikt worden die hierboven zijn aangegeven.
BIT 3 – wordt in Simons' BASIC niet gebruikt en moet altijd 0 zijn.
BIT 4 – als dit 1 is wordt de driehoek golfvorm ingeschakeld.
BIT 5 – als dit 1 is wordt de zaagtand golfvorm ingeschakeld.
BIT 6 – als dit 1 is wordt de puls golfvorm ingeschakeld.
BIT 7 – als dit 1 is wordt de ruisgenerator ingeschakeld.
LET OP: als een van de bits 4 t/m 7 op 1 staat moeten de drie anderen 0 zijn.
4) Het ADSR-verloop wordt bepaald door het
COMMANDO: ENVELOPE
OMSCHRIJVING: bepaalt de omhullende van een geluid
GEBRUIK: ENVELOPE, nummer van de stem, attack, decay, sustain, release
ENVELOPE vn,a,d,s,r
ENVELOPE1,2,9,1,15
OPMERKINGEN: vn = 1 t/m 3
a = 1 t/m 15
d = 1 t/m 15
s = 1 t/m 15
r = 1 t/m 15

1.7 Muziek

Voor verdere uitleg van het besprokene zie hoofdstuk 11/2.1 blz. 1 en 2.
5) De melodie of de geluiden worden geprogrammeerd met het
COMMANDO: MUSIC
OMSCHRIJVING: bepaalt tempo en de te spelen muziek
GEBRUIK: MUSIC tempo, 'string met muziek'
MUSIC n, variabele + variabele . . .
MUSIC 10, A$ + B$ + B1$
OPMERKINGEN: Het getal n, dat het tempo aangeeft, loopt van 1 (zeer langzaam) t/m 255 (zeer snel).
De string van noten kan ten hoogste 255 tekens lang zijn. Voor langere melodieën dient het commando MUSIC herhaald te worden. Een voorbeeld volgt hierna.
6) Tot slot het
COMMANDO: PLAY
OMSCHRIJVING: maakt de melodie hoorbaar.
GEBRUIK: PLAY getal
PLAY n
PLAY 1
OPMERKINGEN: Het getal n kan zijn:
0 – muziek uit;
1 – muziek wordt gespeeld en als dat afgelopen is wordt de rest van het programma – indien aanwezig – afgewerkt;
2 – muziek wordt gespeeld en tijdens het spelen wordt de rest van het programma afgewerkt.

HET SAMENSTELLEN VAN DE MUZIEKSTRING
De string staat tusen aanhalingstekens ("") en als eerste wordt een geïnverteerd hartje in de string opgenomen door de SHIFT- en de CLR-HOME toets in te drukken. Daarachter wordt het nummer van de stem gezet, waarmee gewerkt wordt. Dan volgen de notennamen, elk met bijbehorende octaaf en tijdsduur en aan het eind van de string wordt opnieuw en geïnverteerd hartje met een druk op SHIFT/CLR HOME gezet, gevolgd door de letter G. Deze zorgt voor een juiste release triggering van elke noot.

DE NOTENNAMEN

Simons' BASIC gebruikt de gewone notennamen C, D, E, F, G, A en B. Een verhoogde noot wordt verkregen door de SHIFT toets ingedrukt te houden en de betrefende letter te typen. Op het scherm ziet men dan het grafische teken, dat rechts op de toets van die letter staat. Een verlaagde noot moet worden genoteerd als de verhoogde voorgaande noot: As = Gis, Des = Cis, enz.
Ook moet achter elke noot de octaaf waarin die noot zit worden vermeld. Dus: C4, E5, B2, enz.
Een rust wordt gemaakt met de letter Z.

DE TIJDSDUUR.

Die wordt zoals gebruikelijk uitgedrukt in zestienden, achtsten, kwarten, enz.
De tijdsduur wordt gekozen met de functietoetsen.

FUNCTIETOETS	TIJDSDUUR
F1	1/16
F2	1/8
F3	1/4
F4	1/2
F5	1
F6	2
F7	4
F8	8

Op het scherm ziet U de grafische tekens die bij de respectieve funtietoetsen behoren. Die staan afgedrukt in deel 1:

1.7 Muziek

Voorwoord bij de programmalistings.
Voor elke toon moeten dus drie gegevens worden getypt:
de naam van de noot
de octraaf van de noot
de duur van de noot.

Degenen die muziek kunnen lezen zullen niet te veel moeite hebben om met deze gegevens een melodie te programmeren.

KLANKMAKER

```
10 PRINT""
20 FORI=54272TO54296:POKEI,0:NEXT
30 VOL15
40 PRINT"--------------------------------------------------------"
50 PRINT"|A WAS";A;:INPUT"      ATTACK    ";A:PRINTSPC(38)"|"
60 PRINT"|D WAS";B;:INPUT"      DECAY     ";B:PRINTSPC(38)"|"
70 PRINT"|S WAS";C;:INPUT"      SUSTAIN   ";C:PRINTSPC(38)"|"
80 PRINT"|R WAS";D;:INPUT"      RELEASE   ";D:PRINTSPC(38)"|"
90 PRINT"|G WAS ";G$;:INPUT"      GOLFVORM";G$:PRINTSPC(38)"|"
100 PRINT"--------------------------------------------------------"
110 IFG$="D"THENWAVE1,00010000
120 IFG$="Z"THENWAVE1,00100000
130 IFG$="P"THENWAVE1,01000000:POKE54275,8:POKE54274,0
140 IFG$="N"THENWAVE1,10000000
150 ENVELOPE 1,A,B,C,D
160 A$="1C4D4E4F4G4A4B4C5G"
170 MUSIC 10,A$
180 PLAY 1
190 PRINT"":PLAY 0:GOTO20
```

SAVE het programma na het intypen.
COMMENTAAR: Met dit programma kunnen de getallen voor het ADSR-verloop worden bepaald.
Voor ATTACK, DECAY, SUSTAIN en RELEASE moet een getal van 0-15 worden ingetypt. Voor golfvorm geldt:
D = driehoek, Z = zaagtand, P = puls en N = ruis.
U hoort dan welk geluid de gekozen combinatie voortbrengt: er wordt een korte melodie gespeeld. De getallen kunnen desgewenst veranderd worden; gemakshalve wordt de eerder gekozen waarde/golfvorm op het scherm aangegeven. Maak een lijst van gewenste combinaties.
Regel 20 – reset alle SID-registers.
Regel 130 – registers 54274 en 54275 regelen de breedte van de puls
Regel 160 – een voorbeeld van een muziekstring.

PLAY 1 OF PLAY 2

```
10 PRINT""
20 INPUT"WILT U PLAY 1 OF PLAY 2 HOREN? (1/2)";U
30 IFU<1ORU>2THENPRINT"":GOTO20
40 FORI=54272TO54296:POKEI,0:NEXT
50 VOL15
60 WAVE1,00010000
70 ENVELOPE1,2,9,0,15
80 A$="1C4E4G4C5B4A4G4F4E4D4C4B3C4C5G"
```

1.7 Muziek

```
90 MUSIC10,A$
100 PLAYV:IFV=1THEN150
110 PRINT"HET";:PAUSE1:PRINT"PROGRAMMA";:PAUSE1:PRINT"GAAT";:PAUSE1
120 PRINT"VERDER";:PAUSE1:PRINT"TERWIJL";:PAUSE1:PRINT"HET":PAUSE1
130 PRINT"LIED";:PAUSE1:PRINT"SPEELT."
140 PLAY0:END
150 PRINT"NU PAS GAAT HET PROGRAMMA VERDER!"
160 PLAY0
```

Dit programma is een voorbeeld van het gebruik van PLAY1 en PLAY 2.
Als antwoord op de vraag moet het getal 1 of het getal 2 worden getypt. Regel 30 zorgt er voor, dat andere getallen geen rol spelen. Voor de besturingstekens raadpleeg het Voorwoord bij de Programmalistings in deel 1.

MODULATIE/SYNCHRONISATIE.

```
10 FORT=54272TO54296:POKET,0:NEXT
20 VOL15
30 WAVE1,00100100:WAVE3,00010000
40 ENVELOPE1,2,9,1,15:ENVELOPE3,4,8,8,2
50 A$="1C5C5D5E5C5E5G5C6C6C6C6G5E5C5"
60 A1$="1F5A5F5E5G5E5D5C5D5C5G"
70 B$="3G3"
80 PLAY1
90 MUSIC10,B$+A$+A1$
100 VOL0
```

COMMENTAAR:
Dit programma geeft een soort doedelzak muziek met de gegevens voor WAVE1 in regel 30. Wanneer na RUN meteen READY op het scherm komt dan nog eens RUN geven.
Verander in de regel 30 WAVE1 in WAVE1,00010100 en U krijgt een indruk van synchronisatie.
WAVE1,00010010 laat ringmodulaite horen.
Stem 1 melodie, stem 3 continu toon.

EEN MELODIE.

```
10 VOL15
20 FORT=54272TO54296:POKET,9:NEXT:POKE54275,8
30 PAUSE1
40 WAVE2,00100000:WAVE3,01000000
50 ENVELOPE2,2,10,0,15:ENVELOPE3,5,5,9,6
60 A$="2D4-4E4C5E4C5E4C5"
70 B$="3C5D5-5E5C5D5E5B4D5C5G"
80 C$="2A4G4-4A4C5E5D5C5A4B4C5D5G"
90 D$="3C5D5E5C5D5E5C5D5C5E5C5D5E5G"
100 E$="2C5D5C5E5C5D5E5B4D5C5C6G"
110 MUSIC12,A$+B$+A$+C$+A$+B$
120 PLAY1
130 MUSIC12,D$+E$
140 PLAY1
150 VOL0
```

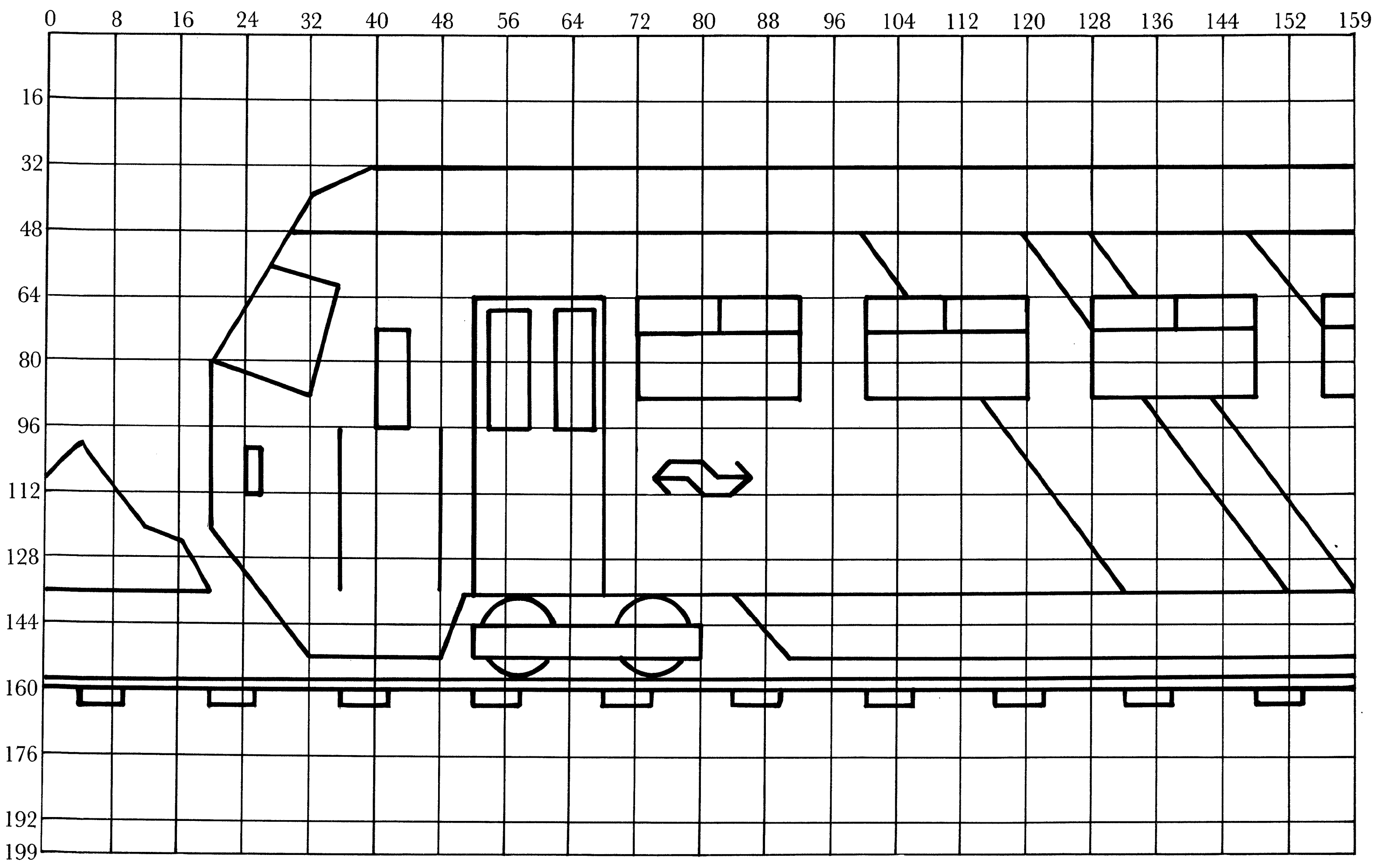
1.7 Muziek

Dit programma speelt een wat langere melodie. Afwisselend worden de stemmen 2 en 3 gebruikt.
De muziekstrings zijn samen te lang, vandaar dat twee keer MUSIC12 gebruikt wordt met de te spelen strings.

RUIS.

```
10 FORT=54272TO54296:POKET,0:NEXT
20 VOL15
30 WAVE1,10000000
40 ENVELOPE1,3,8,2,3
50 MUSIC8,"▙1C5|▌C5|▌C5|▌C5|▌"
60 FORA=1TO3
70 PLAY1
80 NEXTA
```

Een voorbeeld van ruis: het geluid van een stoomlokomotief.



0
8
16
24
32
40
48
56
64
72
80
88
96
104
112
120
128
136
144
152
159
16
32
48
64
80
96
112
128
144
160
176
192
199

8/1.8

Hulp bij het programmeren

Er zijn van die opdrachten die bij het gebruik van de computer herhaaldelijk voorkomen. Neem b.v.: LIST met RETURN, RUN met RETURN. In dit geval kunnen de functietoetsen geprogrammeerd worden om werk uit handen te nemen. Dit werd al genoemd in deel 7/1.1 blz. 2. Daar wordt gesproken over de functietoetsen 1 t/m 8, maar we kunnen beschikken over de functietoetsen 1 t/m 16.
Hoe gaat dat in zijn werk?

NUMMER FUNCTIETOETS	HOE TE TYPEN?
F1-F3-F5-F7	F1-F3-F5-F7 intikken
F2-F4-F6-F8	SHIFT ingedrukt houden, en dan F1-F3-F5-F7 intikken
F9-F11-F13-F15	Commodore toets ingedrukt houden en dan F1-F3-F5-F7 intikken
F10-F12-F14-F16	SHIFT en Commodore toets ingedrukt houden en F1-F3-F5-F7 intikken.

(De handleiding die bij Simons' BASIC wordt geleverd geeft foutieve informatie over de toetsen F9 t/m F16.)
Bijna alle BASIC commando's kunnen weliswaar al in verkorte vorm worden ingetypt, zie voor nadere bijzonderheden de BASIC cheatsheet.
Echter ook in dit geval kunnen de functietoetsen worden gebruikt; dit wordt gedaan met het
COMMANDO: KEY
OMSCHRIJVING: verbindt aan een genoemde functietoets een bepaalde opdracht.
GEBRUIK: KEY nummer functietoets, "opdracht"
KEY 3,"LIST"+CHR$(13)
OPMERKINGEN: De getallen achter KEY lopen van 1 t/m 16. De opdracht kan uit meer delen bestaan: "LIST" +CHR$(13).
KEY kan zowel direct als indirect (in een programma) worden gebruikt.

Enkele voorbeelden:
KEY1,"HIRES0,3"
KEY8,"MULTI0,3,5"
KEY11,"RUN"+CHR$(13)
KEY13,"LIST"+CHR$(13)
KEY16,"DISPLAY"+CHR$(13)

CHR$(13) is gelijk aan RETURN.
Door het indrukken van bijvoorbeeld functietoets 13 wordt als het ware LIST getypt en op RETURN gedrukt. Een goed gebruik maken van deze mogelijkheid kan veel werk besparen.
Verbind aan een bepaalde functietoets wel altijd dezelfde opdracht. Dus F1 is

1.8 Hulp bij het programmeren

bijv. altijd "LIST"+CHR$(13). Maak een lijst van de door U gebruikte functietoetsen en de bijbehorende opdracht, eventueel op een cheatsheet. Want als de computer uitgeschakeld wordt zijn de combinaties verdwenen.
Tijdens het programmeren kan altijd worden nagegaan, welke opdracht aan welke functietoetsen is verbonden. Dit gebeurt met het
COMMANDO: DISPLAY
OMSCHRIJVING: Zet alle functietoetsen met de daarbij behorende opdrachten op het tekstscherm.
GEBRUIK: DISPLAY
OPMERKINGEN: DISPLAY heeft geen parameters. Als aan een functietoets geen opdracht is verbonden dan staan er achter het nummer twee aanhalingstekens: " "
DISPLAY wordt direct gebruikt.
Als KEY15,"DISPLAY"+CHR$(13) bij de aanvang van het programmeren wordt getypt, dan kunt U door op F15 (= Commodore toets met F7) te drukken een overzicht van alle functietoetsen krijgen. Dat kan uiteraard ook met DISPLAY gevolgd door RETURN.

Ook het hierna te bespreken commando INKEY werkt met de functietoetsen. Daardoor zouden problemen kunnen ontstaan.
COMMANDO: INKEY
OMSCHRIJVING: stelt vast welke functietoets werd ingedrukt
GEBRUIK: INKEY
OPMERKINGEN: Zie voor het juiste gebruik van INKEY het volgende programma.
In de handleiding op blz. 3-9 staat, dat alleen F1 t/m F8 kunnen worden gebruikt. Dit is niet juist: ook F9 t/m F16 geven het gewenste resultaat.

GEBRUIK VAN INKEY.

```
10 PRINT"[illegible]"
20 PRINT"[illegible]DRUK OP EEN VAN DE FUNCTIETOETSEN!"
30 Z=INKEY
40 ONZGOSUB100,200,300,400,500,600,700,800,900,1000,1100,,,,,1600
50 GOTO30
100 PRINT"F1 INGEDRUKT":PAUSE1:RETURN
200 PRINT"F2 INGEDRUKT":PAUSE1:RETURN
300 PRINT"F3 INGEDRUKT":PAUSE1:RETURN
400 PRINT"F4 INGEDRUKT":PAUSE1:RETURN
500 PRINT"F5 INGEDRUKT":PAUSE1:RETURN
600 PRINT"F6 INGEDRUKT":PAUSE1:RETURN
700 PRINT"F7 INGEDRUKT":PAUSE1:RETURN
800 PRINT"F8 INGEDRUKT":PAUSE1:RETURN
900 PRINT"F9 INGEDRUKT":PAUSE1:RETURN
1000 PRINT"F10 INGEDRUKT":PAUSE1:RETURN
1100 PRINT"F11 INGEDRUKT":PAUSE1:RETURN
1600 PRINT"F16 INGEDRUKT":PAUSE1:RETURN
```

Commentaar:
Regel 30 – hier staan evenveel adressen van subroutines als er functietoetsen gebruikt worden, n.l. F1 t/m F11. F12 t/m F15 worden niet gebruikt, maar er wordt wel een plaats voor deze toetsen opengehouden met komma's. Tenslotte staat er nog een adres voor F16.

1.8 Hulp bij het programmeren

Regel 40 – zorgt dat het programma in een lus blijft lopen tot er op een functietoets wordt gedrukt.
Regel 100 – PAUSE 1 zorgt er voor, dat er niet te veel tekst op het scherm komt. Laat PAUSE 1 maar eens weg.
Dit is een demonstratieprogramma, de gebruiker zal zelf praktische toepassingen moeten bedenken. Er volgt hierna nog een voorbeeld.
Als dus één van de betreffende functietoetsen wordt ingedrukt, dan komt op het scherm b.v.: F3 INGEDRUKT.
Intoetsen van F12 t/m F15, die niet in gebruik zijn, geeft de foutmelding:

UNDEF'D STATEMENT ERROR IN 30.

Wie zowel KEY als INKEY in een programma wil gebruiken zal een indeling moeten maken. Bijv. F1 t/m F8 voor het INKEY-commando en F9 t/m F16 voor het KEY-commando.
INKEY kan heel goed worden toegepast in menu's, waar een keuze uit een aantal mogelijkheden kan worden gemaakt. Door het indrukken van zo'n toets komt de gewenste mogelijkheid tot uw beschikking.

TOEPASSING VAN INKEY

```
10 PRINT"{CLR}"
20 PRINT"{DOWN}{DOWN}DIT PROGRAMMA GEEFT U ENIGE KLEUR-"
30 PRINT"VARIATIES VOOR SCHERM EN RAND."
40 PRINT"{DOWN}F1: SCHERM GEEL  - RAND PAARS"
50 PRINT"F3: SCHERM GROEN - RAND ZWART"
60 PRINT"F5: SCHERM CYAAN - RAND CYAAN"
70 PRINT"{DOWN}U KUNT NU OP DE GEWENSTE TOETS DRUKKEN."
80 Z=INKEY
90 ON Z GOSUB 500,,600,,700
100 GOTO80
500 COLOUR4,7:RETURN
600 COLOUR0,5:RETURN
700 COLOUR3,3:RETURN
```

Commentaar: regel 500 – COLOUR is besproken in hoofdstuk 8/1.3 blz. 5.

Elke programmeur houdt er rekening mee dat er later nieuwe regels tussen de reeds geprogrammeerde regels moeten worden gevoegd. Vandaar de regelnummering 10,20,30, 40 enz., meestal met 10 opklimmend.
Later toegevoegde regels verstoren de regelmatige nummering. Die wordt dan bijv. 10, 20, 21, 23, 27, 30, 32, 34 . . .
Dit levert bij het RUN-nen van het programma geen enkel bezwaar op. Maar wie prijs stelt op een regelmatige nummering zou gebruik kunnen maken van het

COMMANDO: RENUMBER
OMSCHRIJVING: Geeft alle regels van een programma een nieuw nummer. De regelafstand kan men zelf bepalen.
GEBRUIK: RENUMBER nummer van de nieuwe beginregel, regelafstand
RENUMBER 10,10
RENUMBER 100,50
RENUMBER 1,1
OPMERKING: Met RENUMBER wordt het gehele programma hernummerd. RENUMBER werkt niet

1.8 Hulp bij het programmeren

voor gedeelten van een programma. RENUMBER moet bijna altijd worden gebruikt als men het nog te bespreken commando MERGE wil toepassen. RENUMBER wordt direct gebruikt.

```
1 PRINT"♥"
2 FORI=1TO10
3 PRINT"RENUMBER"
4 NEXTI
5 IFI>15THENGOSUB7
6 GOTO6
7 PRINT"EINDE"
8 END
```

Commentaar: Tik dit korte programma in. Typ vervolgens: RENUMBER10,10 en RETURN.
Dan: LIST en RETURN.
Het nieuwe programma begint nu met regel 10 en de regelnummers klimmen met 10 op.
Bekijk nu regel 50. Daar staat: GOSUB 7. Die regel is er echter niet meer. Zo ook regel 60: GOTO 6. Ook deze regel ontbreekt.
BELANGRIJK. Het commando RENUMBER in Simons' BASIC hopudt geen rekening met GOTO's en GOSUBs in een programma. De gebruiker zal die voor stuk moeten opzoeken en aan de nieuwe nummering moeten aanpassen.
Nu is dat in een kort programma niet zo bezwaarlijk, maar in een lange listing kan gemakkelijk een GOTO of GOSUB over het hoofd worden gezien. Nu zal dat bij het RUNnen van een programma geen ramp zijn: er komt alleen een foutmelding en de vergeten GOTO of GOSUB kan alsnog worden aangepast.

Echter... GOSUBs, GOTO's en nog veel meer woorden of strings in een listing kunnen op eenvoudige manier worden opgespoord met het

COMMANDO: FIND
OMSCHRIJVING: zoekt in een listing na of er een bepaald commando of een bepaalde string in staat. De regelnummers, waarin het gezochte commando of string zich bevindt worden op het scherm afgedrukt.
GEBRUIK: FINDcommando
FIND"string"
OPMERKINGEN: FIND wordt in directe mode gebruikt.
Achter FIND geen spatie typen!
Ter demonstratie wordt het programma RENUMBER gebruikt. Typ dat – indien nodig – in.
Typ vervolgens: Op het scherm komt:

FINDI [RETURN]	2 4 5
FIND> [RETURN]	5
FINDNEXT [RET]	4
FINDGOTO	6
FINDGOSUB	5
FINDPRINT	1 3 7
FINDRENUMBER	niets
FIND"RENUMBER"	3
FIND"EINDE"	7

CGOTO
Het standaard BASIC commando GOTO doorbreekt de gang van een programma – van één regel naar de daarop volgende – en dirigeert het programma naar een ver(der) verwijderde regel. Zie hoofdstuk 7/4.2.
We werken dus met GOTO 510, GOTO 32050 enz.
Achter GOTO kan geen berekening van het gewenste sprongadres worden geplaatst. Dat kan echter wel met het
COMMANDO: CGOTO
OMSCHRIJVING: vindt met een berekening het regelnummer waar het programma vervolgd moet worden.
GEBRUIK: CGOTO berekening of formule.

1.8 Hulp bij het programmeren

Om het gebruik nader toe te lichten dient het volgende programma:

LOTTO GETALLEN

```
10 PRINT""
20 TE=1
30 X=INT(RND(.)*41)+1
40 IFX=AORX=BORX=CORX=DORX=EORX=FTHEN30
50 CGOTOTE*10+100
110 PRINT"1E GETAL =";X:A=X:GOTO180
120 PRINT"2E GETAL =";X:B=X:GOTO180
130 PRINT"3E GETAL =";X:C=X:GOTO180
140 PRINT"4E GETAL =";X:D=X:GOTO180
150 PRINT"5E GETAL =";X:E=X:GOTO180
160 PRINT"6E GETAL =";X:F=X:GOTO180
170 PRINT"RESERVE GETAL =";X:END
180 PAUSE2:TE=TE+1:GOTO40
```

COMMENTAAR:
30 – maakt random getallen van 1 t/m 41 (zie verder hoofdstuk 7/7.3).
40 – zorgt er voor dat alle vermelde getallen verschillend zijn.
De auteur kan deelnemers aan de wekelijkse lotto geen succes garanderen.

In deel 7, hoofdstuk 8/2, worden de BASIC commando's READ, DATA en RESTORE besproken. Van RESTORE wordt uitdrukkelijk vermeld dat het een 'alles of niets'-instructie is: we gaan terug naar de EERSTE gegevens van de EERSTE DATA-regel. Een andere mogelijkheid is er niet en dat kan wel eens lastig zijn.
Simons' BASIC heeft een instructie die ons in staat stelt om DATA te lezen uit een (of meer) door ons zelf gekozen DATA-regels in het programma. Dat is het
COMMANDO: RESET
OMSCHRIJVING: werkt als RESTORE, maar kan verwijzen naar elke gewenste dataregel.
GEBRUIK: RESET regelnummer
RESET 200
RESET wordt gebruikt in het volgende programma:

WERELDSTEDEN

```
10 PRINT""
20 PRINT"DIT PROGRAMMA GEEFT NAMEN VAN VIER"
30 PRINT"GROTE STEDEN. U KUNT KIEZEN UIT:"
40 PRINT"1 - EUROPA"
50 PRINT"2 - AMERIKA"
60 PRINT"3 - AFRIKA"
70 PRINT"4 - AZIE"
80 PRINT"5 - AUSTRALIE"
90 INPUT"TOETS EEN GETAL IN VAN 1 T/M 5";G
100 IFG=1THENRESET200
110 IFG=2THENRESET210
120 IFG=3THENRESET220
130 IFG=4THENRESET230
140 IFG=5THENRESET240
```

1.8 Hulp bij het programmeren

```
150 FORI=1TO4:READA$:PRINT:PRINTA$:NEXT
160 PAUSE8:GOTO10
200 DATA AMSTERDAM,LONDEN,PARIJS,BERLIJN
210 DATA NEW YORK,LOS ANGELES,CHICAGO,RIO DE JANEIRO
220 DATA CAIRO,ALEXANDRIE,TUNIS,KAAPSTAD
230 DATA TOKIO,BANGKOK,SINGAPORE,PEKING
240 DATA MELBOURNE,SYDNEY,BRISBANE,PERTH
```

Het programma kan worden beëindigd door op RUN STOP/RESTORE te drukken.
De grote geheugencapaciteit van de Commodore 64 maakt het mogelijk om lange programma's samen te stellen. Vaak bestaat zo'n lang programma uit een reeks van korte, min of meer zelfstandige onderdelen. Simons' BASIC biedt de mogelijheid om twee programma's of programmadelen tot één geheel te maken en wel met het
COMMANDO: MERGE
OMSCHRIJVING: verbindt twee programma's tot een geheel.
GEBRUIK: MERGE "naam van het toe te voegen programma",n
Bij gebruik cassette-recorder: n=1.
Bij gebruik diskdrive: n=8.

GANG VAN ZAKEN
1) Zorg er voor dat het toe te voegen programma BEGINT met een regelnummer dat HOGER is dan het LAATSTE REGELNUMMER van het eerste programma. Is dat niet het geval dan kan het toe te voegen programma met RENUMBER alsnog worden aangepast.
2) Controleer of programma 1 en 2 nu in orde zijn. Vooral programma 2 na hernummering: zijn er nog GOTO's of GOSUBs naar niet-bestaande regels?
3) Laad (of typ) programma 1. Meestal staat dit al op tape of disk.
4) Typ dan: MERGE "naam programma 2"(,8) en RETURN. De twee programma's worden nu tot één programma geMERGEd.
5) RUN het nieuwe programma ter controle. Als programma 1 eindigde met bijv.
150 GOTO 150 of
150 END
dan moet zo'n regel uiteraard worden verwijderd.
6) SAVE het nieuwe programma op tape/disk.
Nu kan een volgende aanvulling op het programma worden ontworpen en zo kan het 'oude' programma steeds worden uitgebreid, tot het geheugen vol is. Daar kan op gecontroleerd worden middels het PRINT FRE(0)-commando.

PROGRAMMA1

```
10 PRINT"♥"
20 PRINT"▒▒DIT IS PROGRAMMA 1."
30 PRINT"▒▒AAN DIT BESTAANDE PROGRAMMA,DAT"
40 PRINT"▒HET TEKENEN VAN EEN CIRKEL BEHANDELT,"
50 PRINT"▒MOET PROGRAMMA 2 WORDEN GEKOPPELD."
60 PRINT"▒▒DAT MOET WEL OP TAPE OF DISK WORDEN."
70 PRINT"▒OPGENOMEN"
80 END
```

1.8 Hulp bij het programmeren

In dit programma staat alleen tekst.

PROGRAMMA 2

```
100 HIRES0,3
110 CIRCLE160,100,50,50,1
120 TEXT120,164,"EEN CIRKEL",
    1,4,8
130 PAUSE5
```

Dit programma tekent een cirkel op het tekenscherm.
1) SAVE programma 2
2) Laad of typ programma 1.
Verander hierin regel 80 in
80 PAUSE 4
Anders zou het eerste programma toch bij regel 40 stoppen.
3) Typ: MERGE "PROGRAMMA2"(,8), gevolgd door RETURN.
De programma's worden nu geMERGEd.
4) Typ: RUN <RETURN>. Eerst verschijnt de tekst op het scherm, na enige tijd de cirkel.
LET OP: MERGE kan alleen met succes worden toegepast als er al een programma (programma 1 bijvoorbeeld) in het geheugen zit. Is dit niet het geval dan werkt MERGE gewoon als LOAD.
Probeer eens eerst programma 2 te laden en dan met MERGE programma 1 daar aan vast te koppelen. Dit lukt wel, maar als de listing goed bekeken wordt blijkt deze vreemd genummerd te zijn. Wat u ook doet, dit blijft zo, zelfs na RENUMBER. Het programma kan wel geRUNd worden.
Tot slot wordt nog verwezen naar het commando AUTO dat het nummeren van programmaregels heel wat vereenvoudigt. AUTO werd al besrpoken in dit deel, hoofdstuk 1.3 blz. 12-13.

1.8 Hulp bij het programeren

Het listen van een programma.

Een van de meest voorkomende werkzaamheden van de programmeur is het bestuderen van de list van een programma.
Standaard BASIC kent daarvoor het commando LIST. Na RETURN verschijnen de regels van een programma op het scherm, de snelheid waarmee dat gebeurt is hoog. Bij een lange listing bewegen de regels zo vlug naar boven, dat het lezen ervan, laat staan het bestuderen, haast ondoenlijk is.
Er zijn twee manieren om dat tempo te vertragen:
1) houd de CONTROL-toets ingedrukt. De snelheid waarmee de regels naar boven schuiven ('scroll' heet dat in het Engels) wordt beduidend kleiner.
2) druk de Commodore logo-toets in, de toets links beneden op het toetsenbord. De listing stopt en dat is een groot gemak. Er is alle tijd om elke regel te bekijken. Na het loslaten van de Commodore-toets scrollt de listing verder, totdat deze weer wordt ingedrukt.
Dit blijkt alleen te werken met de SIMONS' BASIC cartridge in de computer, en zelfs dan niet altijd. Het is een kleine moeite om dit na te gaan.

Simons' BASIC heeft een aantal commando's, die speciaal bedoeld zijn om het bestuderen van een listing te vergemakkelijken.
1) COMMANDO: DELAY
OMSCHRIJVING: vertraagt het tempo van de listing zolang één van de SHIFT toetsen wordt ingedrukt.
GEBRUIK: DELAY n
DELAY 100
OPMERKINGEN: de parameter n loopt van 1 (snel) t/m 255 (langzaam), n=0 herstelt het oorspronkelijke tempo.
Vergeet niet een van de SHIFT toetsen ingedrukt te houden!
2) COMMANDO: PAGE
OMSCHRIJVING: verdeelt de listing van een programma in delen, die uit een te kiezen aantal regels bestaan.
GEBRUIK: PAGE n
PAGE 10
OPMERKINGEN: het getal n loopt van 1 t/m 23. n=0 herstelt de normale listing.
Bij n=10 komen er op het tekstscherm tien regels, dat wil zeggen schermregels; geen programmaregels. Een vol scherm heeft 25 schermregels.

De toepassing van het commando PAGE.
LOAD eerst een wat langere listing, bijvoorbeeld die van het programma SPRINTER. Als dat gebeurd is, typ dan:
PAGE 10 <RET>
a) op het scherm komt alleen het nummer van de eerste regel van het programma.
b) druk op RETURN. Er verschijnen dan 10 regels op het scherm, aan het eind komt alleen het nummer van de volgende programmaregel. Deze regels kunnen nu rustig worden bekeken.
c) druk weer op RETURN om verder te gaan. De tien volgende regels komen op het scherm. Zo kan men doorgaan, tot het einde van de listing wordt bereikt, het woord READY besluit de listing.
d) als men de listing nog eens wil zien, typ dan LIST en het proces herhaalt zich.
e) PAGE 0 <RET> herstelt de normale listing.
f) de commando's PAGE en DELAY

1.8 Hulp bij het programeren

kunnen desgewenst samen worden gebruikt.
Typ dan:
PAGE 10:DELAY 200
3) Het commando OPTION haalt in de listing duidelijk de speciale Simons' BASIC commando's naar voren.
OPTION 10 zorgt daarvoor.
OPTION 0 (of een ander getal) herstelt de normale listing.
Voor verdere bijzonderheden zie dit deel, hoofdstuk 1.3 blz. 6.
4) FIND zoekt in een listing een bepaald teken, woord of string. FIND is reeds besproken in dit hoofdstuk.

8/1.9

Het opsporen van fouten

Het komt nogal eens voor dat de gang van een programma wordt verstoord door een fout ergens in de listing. Er is dan geen fout gemaakt in commando's of anderszins, want dan was die fout wel op het scherm vermeld: de Commodore 64 kent zo'n 30 foutmeldingen. Toch wil het programma om de een of andere reden niet lopen.
Met Simons' BASIC hebben we de mogelijkheid de loop van een programma regel na regel te volgen. Dat gebeurt met het
COMMANDO: TRACE
OMSCHRIJVING: laat de regelnummers zien, die een programma achtereenvolgens gebruikt.
GEBRUIK: TRACE n
TRACE 10
OPMERKINGEN: Met n=10 wordt TRACE ingeschakeld;
Met n=0 wordt TRACE uitgeschakeld.
BIJZONDERHEDEN: De regelnummers, met een # er voor, verschijnen in een wit blok van 6 letters breed en 6 regels hoog in de rechter bovenhoek van het scherm. Door dit blok wordt een tekst, die daar al stond, overschreven. Houd daarmee rekening.
TRACE kan NIET worden gebruikt op een tekenscherm en ook niet wanneer in het programma het commando MEM voorkomt.
TRACE wordt in directe mode gebruikt.
TRACE wordt genoemd in deel 7/15.2 blz. 1.

VOORBEELD:
1) Laad het bovenstaande programma in;
2) Typ: TRACE 10 <RET>
3) Typ: RUN <RET>
In de rechter bovenhoek komen de nummers van de regels, zoals die in het programma worden doorlopen. Door de vertragingslussen in regel 40 en 60 kunnen die nummers goed bekeken worden.
Als er in een door u gemaakt pro-

```
20 FORT=1TO3
30 PRINT"♥TRACE LAAT ELKE REGEL ZIEN,"
40 FORD=1TO1000:NEXT
50 PRINT"DIE HET PROGRAM DOORLOOPT."
60 FORD=1TO1000:NEXT:NEXT
70 LIST
```

1.9 Het opsporen van de fouten

gramma iets fout gaat en die fout wordt niet gemeld, typ dan TRACE 10 gevolgd door RUN. In het witte blok kunt u het nummer van de regel vinden waar het programma stopt.
Gaat het te snel, druk dan op de logotoets (C-64 toets), links onder het toetsenbord. Dit vertraagt de loop van het programma enigszins.
Wanneer RUN STOP/RESTORE wordt ingedrukt is het scherm weer leeg en ook het witte blok met de regelnummers verdwijnt. Zonder het programma opnieuw te laten lopen kunt u die regelnummers weer te voorschijn halen met het
COMMANDO: RETRACE
OMSCHRIJVING: toont de regelnummers van een programma na indrukken van RUN STOP/RESTORE of na het verbeteren van een programmaregel.
GEBRUIK: RETRACE
OPMERKINGEN: RETRACE heeft geen parameters
RETRACE wordt direct gebruikt.
BIJZONDERHEDEN: Als na RETRACE het commando RUN gegeven moet worden dient de cursor – indien nodig – met de toets CRSR DOWN onder het witte blok gebracht te worden. Dit omdat de regelnummers als parameters voor het commando RUN gezien kunnen worden.

Bij het opsporen van fouten kan het van belang zijn om te weten, welke waarden de variabelen in het programma hebben gekregen. Als variabele A steeds de waarde 0 houdt, terwijl A volgens het programma een waarde van b.v. 42 moet hebben, dan is er kennelijk iets mis met de programmering van A. Natuurlijk kan men intypen: PRINT A waarop de waarde van A op het scherm verschijnt.
Als er in het programma 20 variabelen gecontroleerd dienen te worden, dan geeft Simons' BASIC een betere oplossing met het
COMMANDO: DUMP
OMSCHRIJVING: geeft de waarden van alle variabelen in een programma, met uitzondering van waarden van variabelen in arrays.
GEBRUIK: DUMP
OPMERKINGEN: DUMP heeft geen parameters.
DUMP wordt direct gebruikt.
BIJZONDERHEDEN: Met meer dan 25 variabelen scrollt de tekst naar boven. Druk in dat geval op de CONTROL toets, men heeft dan de tijd om de eerste variabelen te bekijken.
In deel 7/15.2 wordt naar DUMP verwezen.

1.9 Het opsporen van de fouten

VOORBEELD:

```
10 PRINT"[CLR]"
20 COLOUR0,0
30 INPUT"[CRSR]UW NAAM?";A$
40 INPUT"[CRSR]GEBOORTEJAAR";G
50 INPUT"[CRSR]HUIDIG JAARTAL";H
60 K%=RND(.)*8+149:L=H-G
70 PRINT:PRINTCHR$(K%),A$;" IS NU ";L;" JAAR OUD[CRSR]"
80 PAUSE4
```

COMMENTAAR:
regel 60 – levert getallen op tussen 149 en 156. Zie dit deel hoofdstuk 1.3 blz. 9
– LOAD (typ) dit programma en geef RUN.
Na het berekenen van de leeftijd komt er READY op het scherm.
Typ nu: DUMP <RET>
De variabelen uit het programma staan op het scherm met de bijbehorende waarden. De waarde van K% kan variëren, omdat die via RND wordt vastgesteld.

In APPENDIX K van de Programmer's Reference Guide staan een 30-tal foutmeldingen, zoals die door de Commodore 64 worden gebruikt. De gebruiker zal door ervaring wel op de hoogte zijn van de meest voorkomende foutmeldingen.

Simon's BASIC speelt – ten overvloede? – op deze meldingen in met het
COMMANDO: ON ERROR: GOTO
OMSCHRIJVING: spoort een regel op, waarin een fout zit.
GEBRUIK: ON ERROR: GOTO n
ON ERROR: GOTO 150
OPMERKINGEN: Elke fout heeft een nummer, voorgesteld door ERRN. Zo heeft b.v. SYNTAX ERROR het nummer 11 en UNDEF'D STATEMENT het nummer 17.
ERRLN bevat het nummer van de regel, waarin de fout zit.
Met ON ERROR: GOTO kan de gewone foutmelding worden voorzien van een omschrijving in het nederlands, zoals uit het volgende voorbeeld zal blijken.

VOORBEELD:

```
10 ON ERROR:GOTO 60
20 PRIN"[CLR]"
30 GOSUB200
60 IFERRN=11 THEN PRINT" WOORD VERKEERD GETYPT IN REGEL";ERRLN
70 IFERRN=17THEN PRINT" DE REGEL GENOEMD IN";ERRLN;"BESTAAT NIET."
```

1.9 Het opsporen van de fouten

Commentaar:
10 ON ERROR: GOTO moet in de eerste regel staan
20 – het woord PRIN is opzettelijk fout getypt;
30 – regel 200 bestaat niet in dit programma;
60 en 70 – hier zijn de nummers van de fouten opgenomen met een omschrijving van de fout in het Nederlands.
Na RUN komt de betreffende foutmelding op het scherm. De kans bestaat, dat de cursor wegblijft. Druk dan op RUN STOP. Verbeter nu de fout in regel 20, en geef RUN. Nu komt de melding over de niet bestaande regel.

Het nut van het commando ON ERROR: GOTO is discutabel. Bovendien blijken sommige C-64 computers hiermee van slag te raken. Mochten er moeilijkheden worden ondervonden, dan is er maar één advies: gebruik ON ERROR: GOTO niet! De gewone foutmeldingen zijn duidelijk genoeg.

8/1.10

Bijzondere commando's

Als men een programma, dat in het geheugen zit, compleet wil uitwissen, dan doet men dat met NEW. Dit werkt natuurlijk zowel met als zonder Simons' BASIC.
Met Simons' BASIC beschikken we voor zoiets ook nog over het
COMMANDO: COLD
OMSCHRIJVING: verwijdert een programma uit het geheugen en start Simons' BASIC opnieuw.
GEBRUIK: COLD
OPMERKINGEN: COLD wordt direct gebruikt.
COLD heeft geen parameters.
BIJZONDERHEDEN: Overtuig u ervan, alvorens u COLD gebruikt, dat het programma wel degelijk uit het geheugen verwijderd kan worden. Is het al op cassette of disk gezet?

Het gebeurt toch nog wel eens, dat NEW of COLD te haastig gebruikt worden. Dat heeft tot gevolg, dat een programma verloren dreigt te gaan.
Gelukkig is er een middel om na gebruik van NEW of COLD het programma van de ondergang te redden, en wel met het
COMMANDO: OLD
OMSCHRIJVING: maakt een NEW of een COLD aangedaan.
GEBRUIK: OLD
OPMERKINGEN: OLD heeft geen parameters.
OLD wordt uiteraard in directe mode gebruikt.
BIJZONDERHEDEN: Als na het gebruik van COLD een of meer regels van een nieuw programma zijn ingetypt, dan kan OLD het vorige programma niet meer terug halen.

Laad een willekeurig programma in
1) Typ: COLD <RET>.
2) Typ: LIST <RET>. De listing zal verdwenen zijn.
3) Typ: OLD <RET>.
4) Typ: LIST <RET>. De listing komt weer op het scherm en het programma kan weer geRUNd worden.

PRINT AT
Het gewone BASIC kent enkele instructies om de plaats van de cursor te bepalen.
Dat zijn: SPC en TAB. Deze worden besproken in deel 7, hoofdstuk 15.1
Helaas kunnen we niet op een simpele wijze de cursor op een willekeurige plek van het scherm zetten, tenzij we onze toevlucht zoeken tot een tweetal POKE-instructies, zie deel 7, hoofdstuk 14.1 blz.1
Simons' BASIC heeft hiervoor het
COMMANDO: PRINT AT
OMSCHRIJVING: maakt het mogelijk een string van één of meer tekens op

1.10 Bijzondere commando's

een willekeurige plek van het scherm zetten, tenzij we onze toevlucht zoeken tot een tweetal POKE-instructies, zie deel 7, hoofdstuk 14.1 blz. 1
Simons' BASIC heeft hiervoor het
COMMANDO: PRINT AT
OMSCHRIJVING: maakt het mogelijk een string van één of meer tekens op elke gewenste plek van het tekstscherm te zetten.
GEBRUIK: PRINT AT(c,r)"string"
PRINT"string"AT(c,r)"string"
PRINT AT(20,10)"WEKA"
OPMERKINGEN: c= kolom: een getal van 0 t/m 39
r= regel: een getal van 0 t/m 24
VOORBEELD:

```
10 PRINT"♥";
20 PRINTAT(4,4)"WEKA":PRINT
30 PRINT"POSTBUS 61196"AT(4,8)"AMSTERDAM"
40 PAUSE3:PRINT"♥";
50 PRINTAT(0,0)"WEKA"AT(35,0)"WEKA"AT(0,23)"WEKA"AT(35,23)"WEKA"
60 PAUSE5
70 END
```

Dit korte programma spreek voor zich zelf.
Ook kunnen met PRINT AT bewegingen op het tekstscherm vrij eenvoudig worden geprogrammeerd.
Een bewegende bal kan worden gerealiseerd door:
1) de bal ergens op het scherm te zetten;
2) even daarna deze plek leeg te maken;
3) daarna de bal op een volgende plek te zetten;
4) de handelingen 2) en 3) te herhalen.
Dit ziet u in het programma:

KAATSENDE BAL.

```
10 PRINT"♥"
20 X=0:Y=0:DX=1:DY=1
30 PRINTAT(X,Y)"●"
40 FORD=1TO25:NEXT
50 X=X+DX:Y=Y+DY
60 PRINTAT(X-DX,Y-DY)" "
70 IFX=38ORX=0THENDX=-DX
80 IFY=23ORY=0THENDY=-DY
90 GOTO30
```

Commentaar:
regel 20 – DX en DY stellen een kleine toename van resp. X en Y voor
regel 40 – deze vertragingslus geeft een wat beter beeld
regel 60 – de 'oude' bal verdwijnt van het scherm
regels 70 en 80 – zorgen voor het omkeren van de beweging als de bal een rand raakt.
In hoofdstuk 1.3 blz. 7 van dit deel is er op gewzen, dat met SIN en COS desgewenst cirkels of ellipsen kunnen worden getekend. Maar bovendien kunnen met SIN en COS cirkelbewegingen worden geprogrammeerd: het bewegen van een satelliet om een planeet, of iets dergelijks.
In het volgende programma draait een bal over het tekstscherm rond.

1.10 Bijzondere commando's

CIRKELBEWEGING.

```
10 REM CIRKELBEWEGING
20 PRINT"♥"
30 X=10:Y=10:TE=1
40 FORH=0TO2*πSTEP.1
50 PRINT AT(20+X*COS(H),12+Y*SIN(H))"●"
60 PRINT AT(20+X*COS(H),12+Y*SIN(H))" "
70 NEXT
80 TE=TE+1:IFTE<4 THEN40
90 END
```

Commentaar:
regel 30 – de straal van de cirkelbeweging. Als X=10 wordt gewijzigd in X=20, dan doorloopt de bal een ellipsvormige baan. Teller TE maakt het mogelijk om de beweging enkele malen te herhalen.
– π wordt getypt met: SHIFT ↑

Programmabeveiliging
Stel, dat u zelf met veel moeite het programma CIRKELBEWEGING hebt samgensteld en dat u de belangrijkste regels van het programma geheim wilt houden voor anderen. Dan kunt u gebruik maken van het
COMMANDO: DISAPA
OMSCHRIJVING: geeft aan, dat de inhoud van een programmaregel verborgen moet worden. (Engels 'to disappear' betekent: verdwijnen).
GEBRUIK: DISAPA:
OPMERKINGEN: Achter DISAPA moet een dubbele put staan.
DISAPA: moet worden opgenomen direct achter het regelnummer. Met SHIFT/INS kunt u zeven spaties achter het regelnummer invoegen en daar dan DISAPA: zetten.
BIJZONDERHEDEN: Om de inhoud van een programmaregel werkelijk te laten verdwijnen moet SECURE gebruikt worden, waarover straks.
Als de inhoud van de regels 40 t/m 70 van het programma CIRKELBEWE-

1.10 Bijzondere commando's

GING onzichtbaar moet worden zet dan eerst achter die regelnummers DISAPA:
Typ daarna LIST <RET>
Dan ziet u op het tekstscherm:

```
10 REM CIRKELBEWEGING
20 PRINT"♥"
30 DISAPA::::X=10:Y=10:TE=1
40 DISAPA::::FORH=0TO2*πSTEP.1
50 DISAPA::::PRINT AT(20+X*COS(H),12+Y*SIN(H))"●"
60 DISAPA::::PRINT AT(20+X*COS(H),12+Y*SIN(H))" "
70 NEXT
80 TE=TE+1:IFTE<4 THEN40
90 END
```

Merk op, dat er nu achter DISAPA vijf maal een dubbele punt staat. Deze worden automatisch ingevoegd. Houd er rekening mee, dat in zo'n regel niet meer dan 30 tekens kunnen staan, DISAPA en de dubbele punten niet meegerekend.
Nu moet de inhoud van de regels 40 t/m 70 onzichtbaar worden. Dat wordt gedaan met het
COMMANDO: SECURE.
OMSCHRIJVING: De inhoud van alle regels die met DISAPA: beginnen wordt onzichtbaar gemaakt. Alleen de regelnummers blijven over.
GEBRUIK: SECURE 0
OPMERKINGEN: SECURE heeft één parameter, te weten het getal 0
SECURE wordt in directe mode gebruikt.
BIJZONDERHEDEN: De verdwenen inhoud van de regel(s) kan niet meer worden terruggeroepen. Bewaar dus altijd een kopie van het betreffende programma, waarin de regels nog compleet zijn, op tape of disk.

Natuurlijk blijft het programma normaal werken ondanks die lege regels. U kunt het dus aan iemand anders geven zonder de geheimen van programma openbaar te maken. Want als er nu LIST <RET> wordt gegeven, is er slechts dit te zien:

```
10 REM CIRKELBEWEGING
20 PRINT"♥"
30 X=10:Y=10:TE=1
40
50
60
70
80 TE=TE+1:IFTE<4 THEN40
90 END
```

En daar wordt men niet veel wijzer van.

8/1.11

Strings

Wat er zoal met strings kan worden gedaan staat uitvoerig beschreven in deel 7, hoofdstuk 6.
Simon's BASIC geeft nog een paar extra mogelijkheden met strings.

COMMANDO: INSERT
OMSCHRIJVING: zet een string in een reeds bestaande string.
GEBRUIK: INSERT ('nieuwe string', 'oude string', p)
INSERT (N$, O$, 12)

OPMERKINGEN: p geeft de plaats aan in de oude string, waarachter de nieuwe string moet komen. Als p=12 dan wil dat zeggen dat er in die string eerst 12 tekens van de oude string komen te staan, op de 13e plaats volgt dan de nieuwe string. De overige tekens van de oude string worden hier dan nog achter geplakt.
Hoe het commando INSERT dit op twee manieren kan realiseren blijkt uit het volgende voorbeeld:

```
10 PRINT"♥"
20 O$="DIT BOEK IS ONMISBAAR.":PRINTAT(4,2)O$:PAUSE5
30 N$="PRAKTISCH "
40 C$=INSERT(N$,O$,12)
50 PRINTAT(4,4)C$
60 PAUSE5
70 N$="PRAKTISCH "
80 C$=INSERT(N$,"DIT BOEK IS ONMISBAAR",12)
90 PRINTAT(4,10)C$
100 END
```

Commentaar:
Als voor p in regel 40 het getal 24 wordt genomen, dan volgt de foutmelding:
INSERT PARAMETER TOO LARGE IN 40
Strings O$ in regel 20 heeft maar 22 plaatsen, de spaties uiteraard meegerekend. De INSERT parameter p=24 is dus te groot.
Zou de nieuwe string meer dan 233 tekens lang zijn, dan komt op het scherm:
?CREATED STRING TOO LONG
Want 22 + 234 = 256, meer dan de maximaal toegestane lengte van 255 tekens.

1.11 Strings

COMMANDO: INST
OMSCHRIJVING: vervangt een deel van een bestaande string door een nieuwe string.
GEBRUIK: INST ('nieuwe string', 'oude string', p)
OPMERKINGEN: p is weer de plaats in de oude string, waarachter de nieuwe string komt. Zie de opmerkingen bij INSERT.
Voor toepassing van INST zie het volgende programma:

Dan is er het
COMMANDO: CENTRE
OMSCHRIJVING: plaatst een string in het midden van een schermregel.
GEBRUIK: CENTRE 'string'
OPMERKINGEN: de computer houdt automatisch rekening met de lengte van de string: er hoeft niets berekend te worden. In het volgende voorbeeld worden strings van verschillende lengte in het midden van een regel gezet.

```
10 PRINT"♥"
20 O$="DIT BOEK IS ONMISBAAR.":PRINTAT(4,2)O$:PAUSE3
30 N$="GEWELDIG GOED."
40 C$=INST(N$,O$,12)
50 PRINTAT(4,5)C$
60 PAUSE5
70 N$="GEWELDIG GOED."
80 C$=INST(N$,"DIT BOEK IS ONMISBAAR",12)
90 PRINTAT(4,10)C$
100 END
```

```
10 PRINT"♥":PAUSE1
20 CENTRE"A":GOSUB120
30 CENTRE" AM":GOSUB120
40 CENTRE"A AMS":GOSUB120
50 CENTRE".A AMST":GOSUB120
60 CENTRE"K.A AMSTE":GOSUB120
70 CENTRE".K.A AMSTER":GOSUB120
80 CENTRE"E.K.A AMSTERD":GOSUB120
90 CENTRE".E.K.A AMSTERDA":GOSUB120
100 CENTRE"W.E.K.A AMSTERDAM"
110 PAUSE2:END
120 PRINT"□":FORD=1TO75:NEXT:RETURN
```

1.11 Strings

Zo kan een bijzonder effect worden verkregen bij het printen op het scherm van b.v. een titel van een programma.
Om te weten te komen, wáár precies een woord in een string begint (het getal p) dan kan dat gebeuren met het
COMMANDO: PRINT PLACE
OMSCHRIJVING: bepaalt de plaats waar een woord in een string begint en geeft de plaats van de eerste letter van dat woord aan met een getal.
GEBRUIK: PRINT PLACE ('woord', 'string')
PRINT PLACE ('woord', O$)
OPMERKINGEN: PRINT in bovenstaand commando zorgt er voor, dat het getal ook werkelijk op het scherm komt. PLACE berekent het getal, meer niet.
Typ dit programma in:

90 t/m 140 – Het met PLACE berekende getal wordt bij INSERT en INST gebruikt als G-1. Want de nieuwe string wordt geplaatst ACHTER de plaats aangegeven door G!
120 – de string met ZIN, wordt qua lengte gelijk gemaakt aan het woord REGEL. door twee extra spaties. Anders eindigt de regel op: ZIN.L.
RUN het programma. Op het scherm komt:
1) twee keer het getal 23 door PRINT in regel 40 en 50;
2) een keer het getal 23 door PRINT in regel 80
3) een verlengde string O$ door PRINT A$ in regel 110;
4) een gewijzigde string O$ door PRINT B$ in regel 140.

```
10 PRINT"♥"
20 O$="DIT IS EEN VRIJ LANGE REGEL."
30 N$="REGEL"
40 PRINTPLACE(N$,O$)
50 PRINTPLACE("REGEL",O$)
60 PAUSE3
70 G=PLACE(N$,O$)
80 PRINT:PRINTG
90 A$=INSERT ("BASIC ",O$,G-1)
100 PAUSE3
110 PRINT:PRINTA$
120 B$=INST("ZIN.  ",O$,G-1)
130 PAUSE3
140 PRINT:PRINTB$
```

Commentaar:
20 – O$ is de 'oude string'
30 – N$ is de 'nieuwe' string
40/50 – zowel de hele string als de variabele kan worden gebruikt
70 – hier wordt alleen PLACE gebruikt: het getal moet worden berekend.

Een string kan ook enige malen worden herhaald met het
COMMANDO: DUP
OMSCHRIJVING: herhaalt een string 'n' maal.
GEBRUIK: DUP('string',n)
DUP(N$,n)
OPMERKINGEN: De totale lengte

1.11 Strings

van de nieuwe string mag niet groter dan 255 zijn.
Voorbeeld:

```
10 PRINT"♥"
20 A$=DUP("HIEP-",3)
30 B$="HOERA"
40 C$=A$+B$:PRINTC$
50 PAUSE3
60 D$="HIEP-"
70 E$=DUP(D$,3)
80 F$=E$+B$:PRINT:PRINTF$
```

Commentaar:
Het programma laat twee manieren zien waarop DUP kan werken. Omdat B$ en D$ elk 5 letters lang zijn, kan D$ hoogstens 50 maal worden herhaald.
Doe dit, door in regel 70 het getal 3 te vervangen door 50. Er komt dan op het scherm een string van 255 tekens. Wordt de string langer dan 255 tekens, dan kan:

1) slechts een deel van die string worden afgedrukt;
2) de foutmelding ?CREATED STRING TOO LONG (de gemaakte string is te lang) op het scherm verschijnen.

Tot slot van de string-commando's het
COMMANDO: USE
OMSCHRIJVING: zet getallen op het scherm in een bepaald formaat.
GEBRUIK: USE'##.###',vs:PRINT
GEBRUIK: USE'##tekst.###tekst', vs:PRINT
OPMERKINGEN: Het te bewerken getal wordt ingetypt als een string, tussen aanhalingstekens, vandaar de parameter vs.
Als 'tekst' kan b.v. gebruikt worden: het guldensteken f, het dollar- of het pondteken ($ of £), enz.
BIJZONDERHEDEN: USE breekt het gegeven getal af en rondt niet af volgens de gebruikelijke manier: 1 t/m 4 naar beneden, 5 en hoger naar boven.
Typ dit programma in:

```
10 PRINT"♥"
20 USE"#.#####","3.14159265789":PRINT:PRINT
30 USE"#.####",".142857142857":PRINT:PRINT
40 USE"F ####.##","123.4567":PRINT:PRINT
50 USE"###### GULDEN.## CENT","567.5678":PRINT
60 LIST20-50
```

1.11 Strings

Denk er om, dat de Commodore 64 de punt als decimaalteken gebruikt en niet – zoals wij dat doen – de komma!
Door een stringvariabele te gebruiken kan de programmering bekort worden en de notatie van de getallen tevens tot een vast formaat worden beperkt.
Zo kunnen geldbedragen, ongeacht hun grootte, op de juiste manier onder elkaar worden gezet, wat bij een optelling b.v. absoluut vereist is.

Een voorbeeld:

```
10 PRINT"♥"
20 F$="F #####.###"
30 USEF$,"1453.56":PRINT
40 USEF$,"36.17":PRINT
50 USEF$,"9.367":PRINT
60 USEF$,"237.198":PRINT
70 PRINT"------------------"
80 USEF$,"1736.28":PRINT
```

8/1.12

Invoer via het toetsenbord

Met INPUT kan de computer voorzien worden van gegevens die vereist zijn om een programma uit te voeren. Dit commando wordt besproken in deel 7, hoofdstuk 4.1 blz. 2 en verder. Simons' BASIC maakt het mogelijk om aan die input bepaalde voorwaarden te stellen, zodat niet elk willekeurig antwoord door de computer wordt geaccepteerd.

COMMANDO: FETCH
OMSCHRIJVING: werkt als INPUT, maar beperkt de soort en het aantal van de tekens, die worden ingetypt.
GEBRUIK: FETCH"controle teken",n,string variabele
OPMERKINGEN: Het controle teken bepaalt uit welke tekens het antwoord kan bestaan. Er zijn drie mogelijkheden voor dit teken:
1) CLR/HOME: alleen lettertoetsen zonder shift worden toegestaan;
2) CURSOR DOWN: alleen cijfertoetsen worden toegestaan;
3) CURSOR RIGHT: lettertoetsen met of zonder shift worden toegestaan.

n = het aantal tekens dat maximaal kan worden ingetypt. Er kunnen wel minder letters worden ingetypt, maar niet meer!
Het voorbeeld laat zien hoe FETCH werkt.

RUN dit programma. Als antwoord op de eerste vraag kunt u maximaal vier letters typen; punten tussen de letters worden niet geaccepteerd. U kunt wel

```
10 PRINT"[CLR]DOOR WIE WORDT DIT BOEK UITGEGEVEN?
20 FETCH"[HOME]",4,A$
30 PRINT"[DOWN]INDERDAAD, DOOR "A$
40 PRINT"[DOWN]WAT IS HET POSTBUSNUMMER VAN WEKA?"
50 FETCH"[DOWN]",5,B$
60 PRINTB$" IS HOPELIJK HET JUISTE NUMMER"
70 PRINT"[DOWN]ZET NU ONDER 'AMSTERDAM' EEN STREEP."
80 PRINT"AMSTERDAM"
90 FETCH"[RIGHT]",9,C$
100 PRINTC$
110 PRINT"[DOWN]U KUNT HET RESULTAAT ZELF BEOORDELEN."
120 END
```

1.12 Invoer via het toetsenbord

een verkeerd antwoord geven.
Dit kan worden voorkomen door de volgende regel aan het program toe te voegen:
25 IF A$ < >"WEKA" THEN 10
De vraag aan het postbusnummer kan alleen met cijfers worden beantwoord, 5 cijfers of minder. Ziet u kans om de computer alleen het juiste nummer 61196 te laten accepteren?
Met SHIFT E kan het woord AMSTERDAM worden onderstreept. Het antwoord op INPUT kan ook beperkt worden tot een meer bepaalde letters. Hiervoor dient het
COMMANDO: ON KEY .... GOTO
OMSCHRIJVING: De computer gaat na, of de toets(en), in dit commando genoemd, ook werkelijk is (zijn) ingedrukt en gaat alleen dan verder met het programma. Aan letters niet in het commando genoemd wordt voorbijgegaan.
GEBRUIK: ON KEY"letter(s)",: GOTO n
ON KEY"ABCDE",: GOTO 60
K$="ABCDE":ON KEY K$,:GOTO 60
OPMERKINGEN: Tussen ON en KEY een spatie openlaten.
Denk om de ,: achter de string
de string, b.v. K$="ABCDE", mag worden gebruikt, maar moet worden opgenomen in het programma voor het commando ON KEY .... GOTO
Voorbeeld 1, de string met de bepalende letters IN het commando:

```
10 PRINT"[CLR]DRUK OP EEN TOETS (E=EINDE)"
20 ON KEY"ABCDE",:GOTO40
30 GOTO20
40 PRINT"PROGRAMMA GAAT VERDER"
```

Voorbeeld 2, de string apart bepaald:

```
10 PRINT"[CLR]DRUK OP EEN TOETS (E=EINDE)"
20 K$="ABCDE"
30 ON KEYK$,:GOTO50
40 GOTO30
50 PRINT"PROGRAMMA GAAT VERDER"
```

Voor alle zekerheid moet na gebruik van ON KEY het commando DISABLE in het programma worden opgenomen. Anders zou er steeds, bij intoetsen van een van de bij ON KEY vermelde letter(s), kunnen teruggesprongen worden naar de achter GOTO genoemde regel.

Wil men het commando ON KEY .... GOTO nogmaals gebruiken, dan moet men het commando RESUME in het programma opnemen. De combinatie ON KEY .... GOTO, DISABLE en RESUME wordt in het volgende programma gebruikt:

1.12 Invoer via het toetsenbord

```
10 PRINT"[CLR]U ZIET DE NAMEN VAN VIJF PROVINCIES."
20 PRINT"[DOWN]DE HOOFDSTEDEN VINDT U, DOOR DE EERSTE"
30 PRINT"[DOWN]LETTER VAN DIE PROVINCIE TE TYPEN."
40 PRINT"[DOWN]GRONINGEN ,FRIESLAND ,DRENTE ,OVERIJSSEL ,LIMBURG.":PRINT
50 ON KEY "GFDOL",:GOTO70
60 GOTO50
70 DISABLE
80 W$=CHR$(ST):Z=PLACE(W$,"GFDOL")
90 ONZGOTO100,200,300,400,500
100 PRINT"GRONINGEN":RESUME
200 PRINT"LEEUWARDEN":RESUME
300 PRINT"ASSEN":RESUME
400 PRINT"ZWOLLE":RESUME
500 PRINT"MAASTRICHT":RESUME
```

Commentaar:
regel 40 – In plaats van de string "GFDOL" kan ook een string-variabele, bijv. A$="GFDOL", worden opgenomen. Die moet dan wel van te voren worden gedefinieerd en in het programma vóór ON KEY worden opgenomen
60 – Let op de plaats van DISABLE: na ON KEY
70 – Het commando PLACE is een onderdeel van het reeds besproken commando PRINT PLACE. De ingetypte letter wordt vastgelegd in W$. De plaats van W$ in de string "GFDOL" wordt door PLACE berekend en vastgelegd in de variabele Z. De waarde van Z is bepalend voor het antwoord, zie regel 80.
100 t/m 500 – Deze regels eindigen met RESUME, waardoor het programma blijft doorlopen. U kunt het beëindigen door op RUN/STOP te drukken.

8/1.13

Rekencommando's

Rekenmoeilijkheden van de basisschool worden kinderspel met Simons' BASIC. Neem bijv. de zo beruchte staartdelingen:

DELER/DEELTAL/QUOTIENT (UITKOMST)
REST

De computer rekent het quotient en de rest uit. Hiervoor gebruikt men het
COMMANDO: DIV
OMSCHRIJVING: rekent bij een deelsom van twee hele getallen het quotient uit.

GEBRUIK: DIV(deeltal,deler)
DIV(x,y)
DIV(137,14)

OPMERKINGEN: deeltal en deler moeten kleiner zijn dan 65536; de deler mag nooit 0 zijn, anders verschijnt op het scherm de foutmelding: DIVISION BY ZERO ERROR;

DIV kan zowel direkt als in een programma worden gebruikt.

COMMANDO: MOD
OMSCHRIJVING: rekent bij een deelsom van twee hele getallen de rest uit.
GEBRUIK: MOD(deeltal,deler)
MOD(x,y)
MOD(137,14)
OPMERKINGEN: de deler en het deeltal moeten kleiner zijn dan 65536; de deler mag nooit 0 zijn, anders verschijnt de foutmelding: DIVISION BY ZERO ERROR;

MOD kan direkt en in een programma worden gebruikt.

De twee commando's DIV en MOD worden gebruikt in het volgende programma:

```
20 REM DEELSOMMEN
30 PRINT"□□□WAT IS HET DEELTAL? (< 65536)
40 FETCH"□",5,A
50 PRINT"□WAT IS DE DELER? (< 65536)"
60 FETCH"□",5,B
70 PRINT AT(2,8)"HET QUOTIENT IS";DIV(A,B)"EN DE REST IS";MOD(A,B)
```

1.13 Rekencommando's

Commentaar:
40 – door het gebruik van FETCH kan het aantal in te typen cijfers beperkt blijven.
Op een van de volgende bladzijden vindt u nog een toepassing van DIV en MOD: het omrekenen van een decimaal getal naar hexadecimaal.
Mocht het ooit nodig zijn, dan is er nog het
COMMANDO: FRAC
OMSCHRIJVING: geeft het deel van een tiendelige breuk dat achter het decimaalteken staat.
GEBRUIK: FRAC(n)
FRAC(11/7)
FRAC(3.141592)
OPMERKINGEN: FRAC is de tegenhanger van INT: INT geeft het deel van het getal dat voor het decimaalteken staat, FRAC het deel dat er achter staat.
FRAC geeft hoogstens 9 cijfers achter het decimaalteken;
FRAC kan direkt en in een programma worden gebruikt.

Bij computers geldt de punt als decimaalteken.
Typ in:
PRINT 11/7;INT (11/7);FRAC(11/7)
en druk op RETURN. Op het scherm komt:
1.57142857 1 .571428571

Voor degenen die met machinetaal werken, kunnen de volgende commando's van nut zijn.
COMMANDO: PRINT%
OMSCHRIJVING: zet een binair getal om in een decimaal getal: rekent om van het tweetallig naar het tientallig stelsel.
GEBRUIK: PRINT%binair getal
PRINT%00000101
PRINT%11001110
OPMERKINGEN: het binaire getal achter PRINT% moet uit 8 binaire cijfers (0 of 1) bestaan. Zo niet, dan niet u de foutmelding: ?NOT BINARY CHARACTER
Vul dus een getal van minder dan 8 cijfers aan tot 8 cijfers door er nullen voor te zetten:
PRINT%00000100 Antwoord: 4
PRINT%00100001 Antwoord: 33
Als u meer dan 8 cijfers achter PRINT% plaatst, dan worden de eerste 8 cijfers omgerekend; de overige worden onveranderd weergegeven:
PRINT%1100110011101 Antwoord: 204 11101
Het intypen van meer dan 8 cijfers achter PRINT% is dus zinloos.

COMMANDO: PRINT$
OMSCHRIJVING: zet een hexadecimaal getal om in een decimaal getal: rekent om van zestientallig naar tientallig stelsel.
GEBRUIK: PRINT$ hexadecimaal getal
PRINT$15EF
PRINT$ACDF
OPMERKINGEN: achter PRINT$ moeten vier hexadecimale cijfers worden geplaatst, anders volgt de foutmelding: ?NOT HEX CHARACTER.
kortere getallen aanvullen tot vier cijfers door er nullen voor te zetten:
PRINT$000F
PRINT$014E
gebruikt u meer dan vier cijfers achter PRINT$ plaatst, dan worden er slechts vier omgerekend; de overige worden onbewerkt weergegeven:
PRINT$EEFF12AB Antwoord: 61183 12 0
Het plaatsen van meer dan 4 cijfers achter PRINT$ is dus zinloos.

1.13 Rekencommando's

Helaas geeft Simons' BASIC geen commando voor het omrekenen van decimale getallen naar het hexadecimale stelsel. Daar zullen we dus zelf iets op moeten vinden. Met behulp van een paar specifieke Simons' BASIC commando's is dat niet al te moeilijk.

VAN DECIMAAL NAAR HEXADECIMAAL.

Commentaar:

– de specifieke Simons' BASIC commando's worden duidelijk zichtbaar na het intypen van:
OPTION10:LIST
(zie dit deel, hoofdstuk 1.3 blz.6)
– voor meer bijzonderheden over talstelsels zie deel 7, hoofdstuk 12.1 en deel 9, hoofdstuk 1.1 blz. 1
– getallen groter dan 65535 worden

```
10 PRINT"WELK DECIMAAL GETAL MOET OMGEZET"
20 PRINT"WORDEN IN HEX?"
30 INPUTG
40 H=DIV(G,4096):R=MOD(G,4096):IFH<10THENH$=STR$(H):ELSE:GOSUB200
50 PRINTAT(2,8)G"= HEXADECIMAAL $":PRINTAT(25,8)H$
60 H=DIV(R,256):S=MOD(R,256):IFH<10THENH$=STR$(H):ELSE:GOSUB200
70 PRINTAT(27,8)H$
80 H=DIV(S,16):T=MOD(S,16):IFH<10THENH$=STR$(H):ELSE:GOSUB200
90 PRINTAT(29,8)H$
100 H=T:IFH<10THENH$=STR$(H):ELSE:GOSUB200
110 PRINTAT(31,8)H$
120 END
200 IFH=10THENH$="A"
210 IFH=11THENH$="B"
220 IFH=12THENH$="C"
230 IFH=13THENH$="D"
240 IFH=14THENH$="E"
250 IFH=15THENH$="F"
260 RETURN
```

door dit programma niet verwerkt: DIV en MOD kunnen zulke getallen niet omzetten.

LOGISCHE BEWERKINGEN.
Over logische bewerkingen wordt uitvoerig gesproken in deel 7, hoofdstuk 12.2 en 12.3. Waarheidstabellen voor AND, OR en EXOR(EOR) vindt u in deel 9, hoofdstuk 2.1 blz. 1 t/m 3.
De bewerkingen AND en OR behoren tot de standaard commando's van BASIC 2.0
Exclusive OR wordt door Simons' BASIC mogelijk gemaakt met het
COMMANDO: EXOR
OMSCHRIJVING: voert een exclusive OR bewerking uit met twee getallen.
GEBRUIK: EXOR(n,n1)
EXOR(15,194)
Type in:

1.13 Rekencommando's

PRINT 15 AND 194;15 OR 194;EXOR(15,194) en druk op RETURN.	Als antwoord komt op het scherm: 2 207 205

8/1.14

Disk-commando's

Simons' BASIC kent twee commando's voor het gebruik van disk-drives:
COMMANDO: DISK
OMSCHRIJVING: opent een commando-kanaal en sluit dit na de bewerking weer af.
GEBRUIK: DISK"bewerking"
OPMERKINGEN: In de handleiding op blz. 5-1 staat achter DISK in DISK, "operation" een komma. Deze komma moet vervallen.

VOORBEELDEN:
Om een nieuwe schijf te formatteren moet de procedure gevolgd worden als beschreven in deel 6, hoofdstuk 1.2.2 blz. 1
Met Simons' BASIC kan die procedure worden bekort tot:
DISK"N0:disknaam,id"
Hierin staat DISK in de plaats van:
OPEN15,8,15:PRINT,#15,----------
:CLOSE 15

Ook met wissen van een file op een schijf (SCRATCH) wordt met dit commando flink bekort:
DISK"S0:filenaam"

COMMANDO: DIR
OMSCHRIJVING: geeft een listing van alle op de schijf voorkomende programma's (directory) of alleen van door de gebruiker gewenste programma's.
GEBRUIK: DIR"$

DIR"$:string*
DIR"$:?string*

VOORBEELDEN:
DIR"$ – geeft de complete listing;
DIR"$:MULTI* – geeft alleen de files, waarvan de naam MULTI is of met MULTI begint;
DIR"$:??Z* – geeft alleen de files, waarvan de derde letter in de naam een Z is;
DIR"$:?????K* – geeft alleen de files, waarvan de zesde letter in de naam een K is;
DIR"$:???IEK* – geeft alleen de files, waarvan de 4e, 5e en 6e letters in de naam IEK zijn.
De vraagtekens staan dus in de plaats van niet ter zake doende letters.
Het omhoog scrollen van de listing na het commando DIR"$ kan worden vertraagd door de CTRL-toets ingedrukt te houden.
BELANGRIJK: Als in het geheugen van de computer een programma aanwezig is, dan moet men voorzichtig zijn met het gebruik van:
LOAD"$",8 gevolgd door LIST. Hierdoor verdwijnt de listing van dat programma waardoor het dus ook niet meer ge-RUN-d kan worden.
Het gebruik van DIR met al zijn variaties wist zo'n programma niet uit het geheugen!

8/1.15

Commando's ter verfraaiing van het tekstscherm

In dit hoofdstuk worden commando's besproken, waarmee teksten op het scherm wat aantrekkelijker kunnen worden gemaakt dan de gewone zwarte letters op een witte of grijze achtergrond.
En – maar dat geldt wel voor al het computerwerk – met wat vindingrijkheid en fantasie kunnen verrassende effecten worden bereikt.
Allereerst het
COMMANDO: BCKGNDS
OMSCHRIJVING: BCKGNDS kan een letter, woord of regel een andere achtergrondkleur geven dan de op dat moment gebruikte schermkleur.
GEBRUIK: BCKGNDS sc,b1,b2,b3
BCKGNDS 1,3,10,7
OPMERKINGEN: sc – de kleur van het gehele scherm (1 = wit)
b1 – de kleur van de letters, die getypt worden met ingedrukte SHIFT of SHIFT LOCK toets (3 = cyaan)
b2 – de kleur van de letters, die getypt worden nadat u eerst op CTRL/RVS ON hebt gedrukt (10 – licht rood)
b3 – de kleur van de letters, die u met ingedrukte SHIFT of SHIFT LOCK toets hebt getypt, nadat u eerst op CTRL/RVS ON hebt gedrukt (7 = geel).
BIJZONDERHEDEN: BCKGNDS is een samentrekking van het engelse woord 'BACKGROUNDS', dat 'achtergronden' betekent;
CTRL/CVS ON – u houdt de CTRL-toets ingedrukt, en drukt dan op de toets '9' (RVS ON).
De kleuren met de bijbehorende getallen vindt u in de kleurentabel, deel 10 hoofdstuk 2, blz. 5

Het volgende programma geeft u een voorbeeld. De tekst van regel 30 is getypt met ingedrukte SHIFT-toets. Tussen de aanhalingstekens staat: VAN BASIC TOT MACHINETAAL. In regel 50 staat A M S T E R D A M (tussen elke letter een spatie), eveneens getypt met ingedrukte SHIFT-toets. En als u in 'quote mode' CTRL/RVS ON typt, dan ziet u op het scherm een geinverteerde R.

```
10 PRINT"▚"
20 BCKGNDS1,3,10,7
30 PRINTAT(8,4)"X♠/ │♠♥◥─ │┌│ ╲♠─│◥╱▔│ ♠♠L"
40 PRINTAT(8,6)"▚UITGEVERIJ WEKA"
50 PRINTAT(8,8)"▚♠ ╲ ♥ │ ▔ ▁ ─ ♠ ╲"
```

1.15 Commando's ter verfraaiing van het tekstscherm

Om op woorden in een tekst speciale aandacht te vestigen is er het
COMMANDO: FLASH
OMSCHRIJVING: schakelt in een gekozen tempo om tussen letters en geinverteerde letters.
GEBRUIK: FLASH kleur, tempo
FLASH 2,10
OPMERKINGEN: kleur – getal van 0 t/m 15, zie tabel deel 10, hoofdstuk 2 blz. 5;
tempo – van 1 (snel) t/m 225 (langzaam)
FLASH kan niet in hires of multi colour mode worden gebruikt.
BIJZONDERHEDEN: Als u b.v. FLASH 4,40 kiest, dan worden alleen letters in de kleur paars omgeschakeld. U dient er bij het intypen van de tekst voor te zorgen, dat de 'knipperende' woorden ook werkelijk in paarse kleur op het scherm komen. Dat gebeurt door gelijktijdig op CTRL en toets 5 (PUR) te drukken alvorens het bewuste woordt te typen. Letters in een andere kleur knipperen niet!

Ter verduidelijking het volgende programma. Als het programma BCKGNDS nog in het geheugen zit, dan moet u dit met NEW verwijderen, anders werkt FLASH niet goed.

```
10 PRINT"[CLR]"
20 PRINTAT(8,8)"VAN [PUR]BASIC[BLK] TOT [PUR]MACHINETAAL[BLK]"
30 FLASH4,40
40 PAUSE5
50 OFF
60 PRINTAT(10,10)"VOOR DE [GRN]COMMODORE 64[BLK]"
70 FLASH5,40
80 PAUSE5
90 OFF
100 PRINTAT(8,12)"UITGEVERIJ [RED]WEKA[BLK] AMSTERDAM"
110 FLASH2,40
120 PAUSE5
130 FLASH4,15:FLASH5,15:FLASH2,15
140 PAUSE5
150 OFF
160 END
```

FLASH kan ook gebruikt worden met een parameter, die voor kleur. Zo kunt u hele regels laten knipperen, als het getal achter FLASH en de kleur van de getypte letters maar overeenkomen.

1.15 Commando's ter verfraaiing van het tekstscherm

```
10 PRINT"DIT IS EEN REGEL MET WITTE LETTERS."
20 PRINT"DIT IS EEN REGEL MET PAARSE LETTERS."
30 PRINT"DIT IS EEN REGEL MET GROENE LETTERS."
40 PRINT"DIT IS EEN REGEL MET BLAUWE LETTERS."
50 PRINT"DIT IS EEN REGEL MET RODE LETTERS."
60 FLASH1,30:FLASH2:FLASH4:FLASH5:FLASH6
70 PAUSE10
80 OFF
90 PRINT"NU IS FLASH UITGESCHAKELD"
```

Commentaar:
80 – OFF wordt hierna besproken
90 – geeft de gewone tekstkleur terug en voorkomt, dat het hele beeldscherm gaat knipperen.
– als u na SHIFT/CLR HOME direkt intypt: FLASH 0, dan knippert het hele scherm. Typ dan OFF en het knipperen stopt.

COMMANDO: OFF
OMSCHRIJVING: maakt het commando FLASH ongedaan.
GEBRUIK: OFF
OPMERKINGEN: OFF heeft geen parameters.

Ook is er de mogelijkheid om de rand van het scherm afwisselend in twee kleuren te laten knipperen:

COMMANDO: BFLASH
OMSCHRIJVING: lat de rand van het scherm in twee kleuren knipperen.
GEBRUIK: BFLASH tempo, kleur1, kleur2
BFLASH 20,4,7
OPMERKINGEN: BFLASH 0 maakt het commando ongedaan;
tempo – van 1(snel) t/m 255 (langzaam);
kleur1 en kleur2 – van 0 t/m 15 volgens de kleurentabel deel 10, hoofdstuk 2 blz. 5;
B – de beginletter van Border = rand.

Het volgende programma is een voorbeeld van het gebruik van FLASH en BFLASH:

```
10 PRINT""
20 PRINT"WAT IS DE HOOFDSTAD VAN DE PROVINCIE"
30 PRINT"NOORDHOLLAND?":PRINT
40 FETCH"",9,A$
50 IFA$="HAARLEM"THEN100
60 PRINT""A$" IS FOUT!"
70 FLASH5,20:PAUSE5:OFF
80 PRINT"PROBEER HET NOG EENS.":PAUSE3
90 GOTO10
100 PRINT" PRIMA! "
110 FLASH2,20:BFLASH10,4,7:PAUSE2
120 BFLASH10,3,10:PAUSE2
130 BFLASH10,1,7:PAUSE2
140 BFLASH10,8,14:PAUSE2
150 OFF:BFLASH0:PRINT"":COLOUR6,15
```

1.15 Commando's ter verfraaiing van het tekstscherm

Commentaar:
Bekijk wat er gebeurt bij een goed en een foutief antwoord. Zo kunnen vraag- en antwoordspellen wat aantrekkelijker gemaakt worden.
regel 40 – het antwoord kan hoogstens 9 letters bevatten en de SHIFT toets indrukken heeft geen zin;
regel 150 – herstelt de oorspronkelijke toestand van het tekstscherm.

Dan volgen nu enkele commando's, waarmee het scherm – of delen daarvan – met letters, cijfers of grafische tekens kan worden gevuld.
COMMANDO: FCHR
OMSCHRIJVING: vult (een deel van) het scherm met een letter, cijfer, lees- of grafisch teken.
GEBRUIK: FCHR r,c,w,d,code
FCHR 0,0,10,12,1
OPMERKINGEN: r – het nummer van de REGEL, waar het te vullen schermdeel begint;
c – het nummer van de KOLOM, waar het te vullen schermdeel begint;
w – de breedte van het te vullen deel, gemeten in aantallen letters;
d – de hoogte van het te vullen deel, gemeten in aantallen regels;
code – de schermcode van de gewenste letter, het gewenste cijfer of teken. Deze vindt u in APPENDIX 10/1: Scherm POKE- waarden, onder SET 1.

```
10 PRINT"[CLR]"
20 FCHR0,0,10,12,23:PAUSE1
30 FCHR0,10,10,12,5:PAUSE1
40 FCHR0,20,10,12,11;
50 FCHR0,30,10,12,1:PAUSE1
60 FCHR12,2,4,13,1:PAUSE1
70 FCHR12,6,4,13,13:PAUSE1
80 FCHR12,10,4,13,19:PAUSE1
90 FCHR12,14,4,13,20:PAUSE1
100 FCHR12,18,4,13,5:PAUSE1
110 FCHR12,22,4,13,18:PAUSE1
120 FCHR12,26,4,13,4:PAUSE1
130 FCHR12,30,4,13,1:PAUSE1
140 FCHR12,34,4,13,13
150 GOTO150
```

Commentaar:
– het programma wordt beeindigd met een druk op RUN STOP;
– de code getallen achter FCHR kunnen door elk willekeurig getal van 0 t/m 255 worden vervangen. Dat kan aardige effecten opleveren.
Alle letters op het scherm zijn zwart. Om andere kleuren te krijgen gebruikt u het
COMMANDO: FCOL
OMSCHRIJVING: geeft de tekens in een deel van het scherm een bepaalde kleur.
GEBRUIK: FCOL r,c,w,d,kleur
FCOL 0,0,10,12,5
OPMERKINGEN: voor r,c,w, en d zie de OPMERKINGEN onder FCHR;
kleur: een getal van 0 t/m 15 volgens de bekende tabel in deel 10, hoofdstuk 2 blz. 5
FCHR bepaalt de tekens;
FCOL bepaalt de kleur van de tekens.
FCOL moet dus altijd samen met FCHR worden gebruikt in een programma.
Een voorbeeld van FCHR en FCOL (als het vorige FCHR programma nog in het geheugen staat hoeft u van het volgende programma alleen maar de regels te typen, waarvan het nummer op 5 eindigt). De programma's kunnen worden beeindigd met te drukken op RUN STOP.

1.15 Commando's ter verfraaiing van het tekstscherm

```
10 PRINT"♥"
20 FCHR0,0,10,12,23:PAUSE1
25 FCOL0,0,10,12,1
30 FCHR0,10,10,12,5:PAUSE1
35 FCOL0,10,10,12,2
40 FCHR0,20,10,12,11:PAUSE1
45 FCOL0,20,10,12,4
50 FCHR0,30,10,12,1:PAUSE1
55 FCOL0,30,10,12,5
60 FCHR12,2,4,13,1:PAUSE1
65 FCOL12,2,4,13,6
70 FCHR12,6,4,13,13:PAUSE1
75 FCOL12,6,4,13,7
80 FCHR12,10,4,13,19:PAUSE1
85 FCOL12,10,4,13,8
90 FCHR12,14,4,13,20:PAUSE1
95 FCOL12,14,4,13,9
100 FCHR12,18,4,13,5:PAUSE1
105 FCOL12,18,4,13,10
110 FCHR12,22,4,13,18:PAUSE1
115 FCOL12,22,4,13,11
120 FCHR12,26,4,13,4:PAUSE1
125 FCOL12,26,4,13,12
130 FCHR12,30,4,13,1:PAUSE1
135 FCOL12,30,4,13,13
140 FCHR12,34,4,13,13
145 FCOL12,34,4,13,14
150 GOTO150
```

Nog een voorbeeld:

```
10 PRINT"♥"
20 FILL4,12,4,4,23,1:PAUSE1
30 FILL4,16,4,4,5,2:.
40 FILL4,20,4,4,11,4:PAUSE1
50 FILL4,24,4,4,1,5:PAUSE1
60 FILL7,8,4,4,160,7:PAUSE1
70 FILL11,4,4,4,1,6:PAUSE1
80 FILL15,8,4,4,13,8:PAUSE1
90 FILL19,12,4,4,19,9:PAUSE1
100 FILL19,16,4,4,20,10:PAUSE1
110 FILL19,20,4,4,5,11:PAUSE1
120 FILL19,24,4,4,18,12:PAUSE1
130 FILL15,28,4,4,4,13:PAUSE1
```

1.15 Commando's ter verfraaiing van het tekstscherm

```
140 FILL11,32,4,4,1,14:PAUSE1
150 FILL7,28,4,4,13,0:PAUSE1
160 GOTO160
```

Blokken met verschillende kleuren ziet u in dit programma:

```
10 PRINT"▚"
20 FCHR2,2,36,20,81
30 FCOL2,2,36,20,1
40 FCOL4,4,32,16,2
50 FCOL6,6,28,12,4
60 FCOL8,8,24,8,5
70 FCOL10,10,20,4,6
80 FCOL11,11,18,2,7
90 GOTO90
```

Tenslsotte een bewegend abstract schilderij:

```
10 PRINT"▚"
20 C%=RND(.)*39
25 R%=RND(.)*24
30 B%=RND(.)*18+1
40 H%=RND(.)*12+1
50 K%=RND(.)*14
60 IFC%+B%>39ORR%+H%>24THEN20
70 FILL R%,C%,B%,H%,160,K%
80 GOTO20
```

Verplaatsen van een deel van het scherm – het maakt niet uit wat er staat – naar een andere plek gebeurt met het
COMMANDO: MOVE
OMSCHRIJVING: verplaatst een bepaald deel van het scherm naar een andere plaats.
GEBRUIK: MOVEr,c,w,d,dr,dc
OPMERKINGEN: De parameters, r,c,w en d zijn dezelfde als die voor FCHR;
dr en dc zijn de coordinaten van de linkerbovenhoek van de nieuwe plaats op het scherm. dr – nieuwe regel dc – nieuwe kolom:
BIJZONDERHEDEN: FCHR kan ook wel blokken verplaatsen, maar zo'n blok is gevuld met een en hetzelfde teken;
MOVE verplaatst blokken gevuld met willekeurige tekens.
Toepassingen:

```
10 PRINT"▚";
20 PRINT"▛╭─╮"
30 PRINT"╰─╯"
40 PRINT"▛ | "
50 PRINT" | "
60 PRINT" | "
70 PRINT"■───"
80 PAUSE2
90 MOVE0,0,3,6,0,5
100 MOVE0,0,3,6,0,10
110 MOVE0,0,3,6,0,15
120 MOVE0,0,3,6,0,20
130 MOVE0,0,3,6,0,25
140 MOVE0,0,3,6,0,30
150 MOVE0,0,3,6,0,35
```

Commentaar:
10 – vergeet aan het eind van deze regel de ; niet.

```
10 PRINT"▚";
20 PRINT"▛   WEKA   "
30 PRINT"▛UITGEVERIJ"
40 PRINT"▛ AMSTERDAM"
50 PAUSE2
60 MOVE0,0,10,3,0,30
70 PAUSE2
```

1.15 Commando's ter verfraaiing van het tekstscherm

```
80 MOVE0,0,10,3,22,0
90 PAUSE2
100 MOVE0,0,10,3,22,30
110 PAUSE2
120 MOVE0,0,10,3,12,15
130 GOTO130
```

Commentaar:
10 – vergeet aan het eind van deze regel de ; niet.
Met MOVE zouden bewegingen van tekeningen te realiseren zijn als de laatste twee parameters met een variabele, X of Y konden worden aangeduid.
Dit werkt jammer genoeg niet. ▶

Commentaar VARIABELE:
70 – de laatste variabele van MOVE is X; de bedoeling was een rij blokken te maken.
Als het programma wordt ge-RUN-d, dan komen de waarden voor X wel op het scherm, maar het blok letters wordt niet gereproduceerd.
– vervang nu de X in regel 70 door 9 en RUN het programma. Dan werkt move wel!
COMMANDO: INV
OMSCHRIJVING: inverteert een bepaald deel van het tekstscherm.
GEBRUIK: INVr,c,w,d
INV10,8,25,5
OPMERKINGEN: De parameters r,c,w en d zijn dezelfde als die van FCHR;

```
10 PRINT"■";
20 PRINT"■ABCD"
30 PRINT"■DEFG"
40 PRINT"■HIJK"
50 PRINT"■LMNO"
60 X=6
70 MOVE0,0,4,4,0,X:PRINTAT(X,10)X:PAUSE2
80 X=X+5:IFX<35THEN70
90 END
```

wanneer letters als geinverteerd zijn voordat INV zijn werk doet, dan worden die letters nu normaal. ▶
Voorbeeld:

```
10 PRINT"■■"
20 PRINTAT(8,10)"■VAN BASIC TOT MACHINETAAL"
30 PRINTAT(12,12)"■■UITGAVE WEKA BV"
40 PRINTAT(14,14)"■AMSTERDAM."
50 PAUSE3
60 FORX=1TO10:INV10,8,25,5:PAUSE2
70 NEXT
```

1.15 Commando's ter verfraaiing van het tekstscherm

Commentaar:
10 – de kleur lichtgrijs wordt hier ingevoerd. Anders zou de achtergrond van het hele tekstblok ook inverteren van grijs naar de laatst gebruikte tekstkleur en vice versa.
30 – deze regel wordt al geinverteerd geprogrammeerd.

Mooie effecten kunnen tenslotte bereikt worden met het 'scrollen' van teksten of tekeningen. Hiermee wordt bedoeld het verschuiven van een deel van het scherm naar links, rechts, boven of beneden. Lichtkranten, bewegende tekeningen enz. kunnen op een vrij eenvoudige manier worden ontworpen.
COMMANDO: RICHTING (LEFT, RIGHT, UP of DOWN)
OMSCHRIJVING: verschuift een deel van het tekstscherm in de aangegeven richting.
GEBRUIK: RICHTING W sr,sc,ec,er
RICHTING B sr,sc,ec,er
RICHTING W 0,0,20,25
RICHTING B 0,20,20,25
OPMERKINGEN: Er zijn vier richtingen mogelijk:
LEFT – naar links
RIGHT – naar rechts
UP – naar boven
DOWN – naar beneden.
Als commando moet u een van deze vier kiezen. De keus wordt bepaald door de richting waarin u het deel van het scherm wilt laten verschuiven.
W – 'wrap around': wat aan de ene kant van het scherm verdwijnt komt aan de andere kant weer te voorschijn.
B – 'blanking': wat aan de ene kant van het scherm verwijnt komt niet meer te voorschijn.
sr en sc – beginregel en beginkolom van verschuiven schermdeel (s= start)
ec – het aantal kolommen dat dit schermdeel breed is.
er – het aantal regels dat dit schermdeel hoog is (e= end)
Achter de RICHTING en W (of B) komt geen komma!
Verschillende schermdelen kunnen in verschillende richtingen verschoven worden.
Deze commando's kunnen helaas niet gebruikt worden in hires of multi colour mode.
Dat ziet er misschien wat ingewikkeld uit, maar de volgende programma's helpen u verder op weg.
Eerst een eenvoudige 'lichtkrant'.

```
10 A=TI
20 PRINT"▣"
30 PRINTAT(0,12)"▣WEKA  ▣UITGEVERIJ  ▣AMSTERDAM."
40 LEFTW0,0,40,25:FORT=1TO100:NEXT
50 IFTI-A<1000THEN40
```

Commentaar:
regel 10 en 50 – hier wordt gebruik gemaakt van de functie TI. TI = 0 als de computer wordt ingeschakeld en elke

1.15 Commando's ter verfraaiing van het tekstscherm

1/60 seconde wordt TI met 1 vermeerderd. A is de waarde van TI bij het begin van het programma; zodra TI 1000 meer is dan A (dat duurt ca. 16 seconden) stopt het programma.
RUN dit programma.
Breng dan de volgende veranderingen aan:
40 RIGHT W 0,0,40,25 en RUN
40 UP W 0,0,40,25 en RUN
40 DOWN W 0,0,40,25 en RUN
40 LEFT B 0,0,40,25 en RUN
Tegengestelde bewegingen van teksten krijgt u met:

```
10 A=TI
20 PRINT""
30 PRINTAT(0,12)"DIT IS EEN VOORBEELD"
40 PRINTAT(21,12)"VAN TEKST-SCROLLEN"
50 UPW0,0,20,25:DOWNW0,20,20,25
60 IFTI-A<600THEN50
```

Commentaar:
regel 10 en 60 – ook hier wordt TI gebruikt. Wilt u het programma laten doorlopen vervang regel 60 dan in
60 GOTO 50
Het programma wordt nu beeindigd door te drukken op RUN STOP.
Breng vervolgens deze veranderingen aan en RUN die:
50 DOWN W 0,0,20,25: UP W 0,20,20,25
50 LEFT W 0,0,20,25: DOWN W 0,20,20,25
50 UP B 0,0,20,25: DOWN B 0,20,20,25
Met bovenstaande programa's is het 'scrollen' van teksten vermoedelijk wel duidelijker geworden.
Een praktische toepassing: op deze manier kunnen gebruikers van uw programma's duidelijke aanwijzingen gegeven worden, bijvoorbeeld hoe ze een 'blad kunnen omslaag.

```
10 PRINT"DE GEBRUIKER KRIJGT EEN TEKST."
20 PRINT"NA DE TEKST GELEZEN TE HEBBEN,"
30 PRINT"DIENT DE LEZER HET 'BLAD' OM TE"
40 PRINT"SLAAN.
50 PAUSE4:REM DIENT OM DE LEZER TIJD TE GEVEN DIT BLAD TE LEZEN
60 GOSUB1000
70 PRINT"EN NU KOMT HET VOLGENDE BLAD.
80 PRINT"IS DE LEZER KLAAR,DAN KAN HIJ OPNIEUW"
90 PRINT"OMSLAAN EN VERDER LEZEN."
100 PAUSE4:REM WEER EVEN TIJD GEVEN
110 GOSUB1000
120 PRINT"VOLGEND BLAD."
130 PRINT"EN ZO KUNT U DOORGAAN!"
140 END
1000 REM PROCEDURE VOLGEND BLAD
```

1.15 Commando's ter verfraaiing van het tekstscherm

```
1010 PRINTAT(0,23)"■VOLGEND BLAD?  DRUK OP DE ▒ SPATIEBALK "
1020 LEFTW23,0,40,1
1030 FORD=1TO150:NEXT
1040 GETA$:IFA$=" "THENRETURN
1050 GOTO1020
```

Commentaar:
50 en 100 – hier krijgt de lezer de tijd om de tekst ongestoord te lezen. De lengte van PAUSE hangt af van de tijd die benodigd is om alles rustig te lezen.
1000 – de subroutine VOLGEND BLAD
1030 – maakt het lezen van de bewegende tekst wat aangenamer.
1040 – tussen de aanhalingstekens een spatie zetten. Bij deze spatie niet op SHIFT drukken, anders werkt het programma niet.
1050 – de aanwijzing blijft onder aan het scherm bewegen tot er op spatiebalk wordt gedrukt.

ANIMATIE VAN TEKENINGEN.
Een golfbeweging:

```
10 PRINT"▒"
20 FORX=0TO39
30 Y%=8*SIN(X/π)+12
40 PRINTAT(X,Y%)"●"
50 NEXT
60 LEFTW0,0,40,25
70 GOTO60
```

Commentaar:
30 krijgt u door op SHIFT en ^ te drukken.
40 krijgt u door op SHIFT en Q te drukken.
Breng ook volgende veranderingen aan en RUN het veranderde programma.
60 RIGHT W 0,0,40,25
60 RIGHT B 0,0,40,25
60 LEFT W 0,0,20,25: RIGHT W 0,20, 20,25
60 DOWN W 0,0,20,25: RIGHT W 0, 20,20,25

Verrassende effecten kunnen worden bereikt met stilstaande sprites op een bewegende achtergrond. Met de grafische tekens van de Commodore 64 wordt een achtergrond tekening gemaakt en die achtergrond wordt over het scherm bewogen. Het intypen van het programma moet zorgvuldig gebeuren. Raadpleeg vooral de tekens in het VOORWOORD BIJ DE PROGRAMMALISTINGS. Tel de spaties in de regels 30 t/m 90 nauwkeurig.

```
10 PRINT"▒"
20 PRINTAT(20,6)"▒▁"
30 PRINTAT(19,7)"▒/ \"
40 PRINTAT(18,8)"▒|    ‾\"
50 PRINTAT(17,9)"▒/‾        ‾‾\"
60 PRINTAT(13,10)"▒/‾‾‾‾                ‾‾\"
```

1.15 Commando's ter verfraaiing van het tekstscherm

```
70 PRINTAT(10,11)"
80 PRINTAT(9,12)"
90 PRINTAT(8,13)"
100 PRINTAT(0,14)"
110 DESIGN0,12288
120 @BBBBB...................
130 @BBBBB...................
140 @.....BBBBB..............
150 @.....BBBBB..............
160 @...........BBBB.........
170 @................BB......
180 @................BB......
190 @........................
200 @...................BB...
210 @..BBBBBB.......B...BB...
220 @...B...B.B....BBB..BB...
230 @...B...BBBB...BBB..BB...
240 @...B..BBBBBBBBBBBBBBBB..
250 @BBBB..BBBBBBBBBBBBBBBBB.
260 @BBBB..BBBBBBBBBBBBBBBBBB
270 @BBBB..BBBBBBBBBBBBBBBBB.
280 @BBBB..BBBBBBBBBBBBBBBB..
290 @BBBB..BB...........BB..B
300 @BBBBBBBBBBBBBBBBBBBBBBBB
310 @.BB....BBB..BBB...BBBBB.
320 @.BB....BBB..BBB....B.B..
330 MOB SET1,192,0,0,0
340 MMOB1,120,141,120,141,0,255
350 LEFTW6,0,40,8:FORD=0TO50:NEXT
360 GOTO350
```

Commentaar:
340 – de x en y coordinaten van de sprite zijn gelijk aan de z1 en y1 coordinaten. De sprite staat dus stil.
350 – de vertragingslus geeft een wat rustiger beeld.

Speciale schermcommando's
Mocht u een tekening zoals die in het vorige programma willen bewaren zonder het programma zelf te gebruiken dan kan dat met het
COMMANDO: SCRSV
OMSCHRIJVING: slaat de gegevens van een tekstscherm op.
GEBRUIK: SCRSV 2,8,2,"naam,S,W" voor disk
SCRSV 1,1,1,"naam" voor tape
OPMERKINGEN: de enige parameter die u moet invullen is de naam van de file.
SCRSV staat voor SCReen SaVe.
Hires of multicolour schermen kunnen niet door dit commando worden verwerkt.
Als u het vorige programma nog eens RUN-t en op RUN STOP drukt, dan heeft u een tekening op het scherm.
Typ dan op de onderste regel:

1.15 Commando's ter verfraaiing van het tekstscherm

SCRSV 2,8,2,"LOK,S,W" <RET> als u de tekening op diskette wilt bewaren.
SCRSV 1,1,1,<RET> om de tekening op tape te zette. U ziet dan de bekende medeling: PRESS RECORD & PLAY ON TAPE.
Als u dat doet wordt de tekening op tape opgenomen.
Sprites worden door SCRSV niet verwerkt.
De tekening kunt u later weer op het scherm krijgen met het
COMMANDO: SCRLD
OMSCHRIJVING: zet data van een tekening op disk of tape om in een tekening op het scherm.
GEBRUIK: SCRLD 2,8,2,"naam" voor disk
SCRLD 1,1,0,"naam" voor tape.
OPMERKINGEN: de enige parameter die u moet invullen is de naam, die bij SCRSV aan de file was gegeven. Die kunt u desgewenst met DIR"$ in de directory vinden.
Let op de 0 voor tape gebruik.
SCRLD = SCReen LoaD

Na SCRLD 1,1,0,"LOK" verschijnt op het scherm de bekende mededeling: PRESS PLAY ON TAPE.
Als u dat doet komt de tekening weer op het scherm. U zult dan ook zien, dat de sprite op de originele tekening niet wordt afgedrukt.

Wie over een printer beschikt kan met eenvoudige commando's teksten of tekeningen, die op het monitorscherm staan, op papier overbrengen. Dit is van belang als men grafieken of histogrammen wil afdrukken, maar artistieke prestaties kunnen ook vereeuwigd worden.

Voor het uitvoeren van de volgende programma's is voornamelijk gebruik gemaakt van de Commodore printer MPS-801.
Zorg er dus voor dat een printer is aangesloten en klaar staat voor gebruik.
COMMANDO: COPY
OMSCHRIJVING: produceert met de printer een afdruk van een tekenscherm (zie dit deel, hoofdstuk 1.3 blz. 2).
GEBRUIK: COPY
OPMERKINGEN: COPY heeft geen parameters
BIJZONDERHEDEN: COPY kan direkt zowel als indirekt worden gebruikt.

Bij het bespreken van het commando CIRKEL (deel 8, hoofdstuk 1.3 blz. 17) is al gewezen op de fout in de handleiding bij Simons' BASIC. De lengte van de stralen xr en yr is daar onjuist vermeld. Dezelfde fout komt weer ter sprake in diezelfde handleiding op blz. 7-11.

1) Het printen van een hi-res tekenscherm.
De waardes van de stralen xr en yr moeten dezelfde zijn. Op het tekenscherm en op het papier komt een cirkel. Ellipsen worden natuurlijk getekend en afgedrukt met verschillende waarden voor xr en yr.
Een voorbeeld:

```
10 HIRES0,1
20 CIRCLE160,100,70,70,1
30 ANGL160,100,120,70,70,1
40 ANGL160,100,160,70,70,1
50 ANGL160,100,220,70,70,1
60 ANGL160,100,330,70,70,1
70 COPY
80 END
```

1.15 Commando's ter verfraaiing van het tekstscherm

Als bovendien in MULTI COLOUR mode wordt getekend, dan moet er wel verschil zijn tussen xr en yr. Er zal wat met deze waardes voor xr en yr moeten worden geëxperimenteerd.
Richtlijn: yr=1.8 – 2 maal xr.
Toepassing:

```
10 HIRES0,1:MULTI0,1,2
20 CIRCLE80,100,39,78,1
30 CIRCLE80,100,45,90,1
40 ANGL80,100,60,39,78,1
50 ANGL80,100,120,39,78,1
60 ANGL80,100,180,39,78,1
70 ANGL80,100,240,39,78,1
80 ANGL80,100,300,39,78,1
90 ANGL80,100,360,39,78,1
100 COPY
110 END
```

Er is enig verschil tussen de cirkel op het scherm en die op het papier.
Met de kleine Citizen printer CDU1 geeft het commando COPY niet het gewenste resultaat.
Als u het programma SPRINTER uit een der vorige hoofdstukken nog op tape of disk hebt staan, LOAD dat dan. We zullen een afdruk van die tekening gaan maken.
Er moet wel aan gedacht worden, dat er nogal wat commando's PAINT in dat programma staan en het is sterk af te raden die bij het printen te handhaven. De printer zal dan alle vlakken geheel proberen te vullen, en dat is schadelijk voor het inktlint en de printkop.
PAINT moet dus uit het programma worden verwijderd. In welke regels staat dat commando? Dat is eenvoudig na te gaan:
Typ in: FINDPAINT <RET>
(Denk er om: geen spatie tussen FIND en PAINT: die spatie wordt n.l. beschouwd als deel van het te zoeken woord!)

Op het scherm komt:
110 170 190 220 270 290
Noteer die regelnummers.
LIST nu elke regel en verwijder het commando PAINT met bijbehorende parameters op de bekende manier. Voeg dan aan het programma deze regel toe:
315 COPY
RUN nu het programma. Op het scherm komt eerst de tekening van de sprinter – zonder kleuren natuurlijk – en dan wordt die tekening op papier afgedrukt.

2) Het printen van een 'low-res' tekstscherm.
Hiervoor dient het
COMMANDO: HRDCPY
OMSCHRIJVING: produceert met een printer een afdruk van het tekstscherm.
GEBRUIK: HRDCPY
OPMERKINGEN: HRDCPY heeft geen parameters.
BIJZONDERHEDEN: HRDCPY kan direkt en indirekt worden gebruikt.
HRDCPY werkt wel met een kleine printer als b.v. de Citizen CDU1. Zo kunnen met de grafische tekens grafieken etc. worden samengesteld, die dan ook met een kleine printer worden afgedrukt.
Als de listing van het laatste programma nog in het geheugen zit, zet dat dan op het scherm met LIST
Typ dan: HRDCPY <RET>
De listing wordt op het papier gedrukt.
De printer kan worden gestopt met RUN STOP/RESTORE.
Voorbeeld van indirekt gebruik:

```
10 PRINT"♥"
20 FORT=1TO20
30 PRINT"VAN BASIC TOT MACHINETAAL"
40 NEXT
50 HRDCPY
60 END

READY.
```

8/1.16

Gestruktureerd programmeren.

Als programma's langer gaan worden, dan is er een grote kans dat het geheel onoverzichtelijk wordt. Want meestal wordt het steeds moeilijker om de gang van zaken in het programma te volgen.
Dat kan komen door b.v. het veelvuldig gebruik van GOTO en GOSUB.
Wel is het mogelijk om 'REM'-regels hier en daar in het programma te zetten die duidelijk aangegeven, wat het daarachter staande programma-deel nu eigenlijk doet. Maar dat kost weer geheugenruimte.
Het is dus zaak om elk programma dat je maakt logisch op te bouwen, een duidelijke structuur te geven.
In deel 12 van dit boek boek wordt het hoe en waarom van programma-struktuur behandeld.

In elk programma staat wel een aantal regels, waar een beslissing moet worden genomen. In eerste instantie door de programmeur en in tweede instantie door de computer. Want de gebruiker van een programma zal moeten aangeven of hij wil doorgaan of stoppen, of zijn antwoord 'JA' of 'NEE' is, e.d. Vaak gebeurt dat met de test: IF .. THEN .. , besproken in deel 7 van dit boek, hoofdstuk 4.3.
Simons' BASIC maakt deze mogelijkheid iets uigebreider met het commando: IF .. THEN .. ELSE ..
We hebben het commando al besproken in hoofdstuk 1.3 blz. 13 van dit deel, en daar wordt het ook verder toegelicht.
Nog even een voorbeeld:

```
20 REM IF..THEN..ELSE
30 PRINT"■WELKE KLEUR MOETEN DE LETTERS HEBBEN,"
40 INPUT"■ROOD OF GROEN?";A$
50 IFLEFT$(A$,2)="RO"THEN70
60 IFLEFT$(A$,2)="GR"THEN90:ELSE:GOTO40
70 PRINT"■■■DE TEKST BESTAAT UIT ■ RODE  LETTERS."
80 PRINT"■":END
90 PRINT"■■■DE TEKST BESTAAT UIT ■ GROENE  LETTERS."
100 PRINT"■":END
```

1.16 Gestruktueerd programmeren

Commentaar:
In dit programma moet een keuze gemaakt worden: rode of groene letters.
Regel 60 – vergeet niet :ELSE: tussen dubbele punten te zetten.
Als ELSE naar een regelnummer verwijst, dan moet GOTO voor dat regelnummer staan, want :ELSE:30 geeft de foutmelding: ?SYNTAX ERROR.
Er is zelfs een commando dat ons in staat stelt om zo'n keuze een aantal keren te herhalen, zonder steeds opnieuw voorwaarden voor die keuze te vermelden.
COMMANDO: RCOMP..:ELSE:..
OMSCHRIJVING: Met RCOMP.. :ELSE:.. wordt de laatst gebruikte IF.. THEN.. :ELSE: .. test herhald.
GEBRUIK: RCOMP: 1e keuze :ELSE: 2e keuze
OPMERKINGEN: RCOMP is blijkbaar de afkorting van: Repeat COMParison = herhaal de vergelijking.
Het gebruik van RCOMP..:ELSE:.. wordt verduidelijkt in het volgende programma: 'Landen of Steden'.

```
10 PRINT"{CLR}"
20 INPUT"STEDEN OF LANDEN";A$
30 IF LEFT$(A$,2)="LA" OR LEFT$(A$,2)="ST"THEN 40:ELSE:GOTO 10
40 IF LEFT$(A$,2)="ST" THEN PRINT"{DOWN}AMSTERDAM":ELSE:PRINT"{DOWN}NEDERLAND"
50 RCOMP:PRINT"LONDEN":ELSE:PRINT"ENGELAND"
60 RCOMP:PRINT"BRUSSEL":ELSE:PRINT"BELGIE"
70 RCOMP:PRINT"PARIJS":ELSE:PRINT"FRANKRIJK"
80 RCOMP:PRINT"BERN":ELSE:PRINT"ZWITSERLAND"
90 PAUSE5:GOTO10
```

Om gedeelten van een programma te herhalen wordt de z.g. FOR..NEXT lus gebruikt, zie deel 7, hoofdstuk 7/5.1
Het aantal herhalingen wordt begrensd door de getallen achter FOR A = ...
FOR A = 1 TO 5 wil zeggen: het betreffende programma-deel wordt vijf maal herhaald.
Iets dergelijks doet het commando REPEAT .. UNTIL .., met dit verschil, dat de begrenzing van het herhalen wordt aangegeven NA het programmagedeelte.
(Repeat = herhaal, until = totdat)

COMMANDO: REPEAT .. UNTIL ..
OMSCHRIJVING: voert een lus in een programma uit totdat aan een bepaalde voorwaarde wordt voldaan.
GEBRUIK: REPEAT: programma deel : UNTIL voorwaarde
REPEAT: Print "OK" : UNTIL T=20
OPMERKINGEN: Als in zo'n programmadeel nog meer lussen voorkomen, 'lussen in een lus', dan mag het aantal lussen niet groter zijn dan vijf. (De handleiding vermeldt abusievelijk: negen).
Hier volgt een voorbeeld van het gebruik:

```
10 PRINT"{CLR}"
20 REM REPEAT .. UNTIL
30 T=33
```

1.16 Gestruktureerd programmeren.

```
40 REPEAT:PRINTCHR$(T);:T=T+1:UNTIL T=133
50 PRINT
60 PRINT"███DIT ZIJN DE EERSTE" T-33 "TEKENS."
70 PRINT"■":END
```

Commentaar: In regel 40 staat de begrenzing opgenomen:
UNTIL T = 133
Als u dit getal verhoogt, dan resulteert dat tenslotte in leeg maken van het scherm – CHR$(147) – of in kleurveranderingen – CHR$(159) = cyaan.
Als u nog meer voorwaarden voor een lus in het programma wilt opnemen dan gebruikt u het

COMMANDO: LOOP .. EXIT IF .. END LOOP
OMSCHRIJVING: Vormt een doorlopende lus totdat aan een of meer voorwaarden wordt voldaan.
GEBRUIK: zie voorbeeldprogramma hierna.
OPMERKINGEN: Ook hier geldt, dat het aantal lussen in een lus hoogstens vijf mag zijn.

```
10 PRINT"█"
20 REM LOOP..EXIT IF
30 LOOP
40 INPUT"GEEF EEN GETAL VAN 3 T/M 7";A
50 EXIT IF A>7
60 EXIT IF A<3
65 PRINT:PRINTA:PRINT
70 END LOOP
80 PRINT"██DIT GETAL IS NIET GEVRAAGD!"
90 PAUSE2
100 GOTO10
```

U zou zich kunnen afvragen of zo'n Simons' BASIC commando nu wel zo opzienbarend is. Om bij het laatste commando te blijven: dit kan ook in normaal BASIC uitstekend worden uitgevoerd, getuige het volgende programma:

```
10 PRINT"█"
20 INPUT"GEEF EEN GETAL VAN 3 T/M 7";A
30 IF A<3 OR A>7 THEN 50
40 PRINT:PRINT A:PRINT:GOTO 20
50 PRINT "DIT IS GEEN GOED GETAL!"
60 FOR T=1 TO 1500: NEXT:GOTO 20
```

1.16 Gestruktureerd programmeren

U ziet, het is niet eens zo ingewikkeld. Maar de nu volgende commando's hebben veel meer te maken met de opbouw van programma's. De struktuur van een programma zal door de programmeur zelf moeten worden bepaald: een 'struktuurplan', van te voren op papier gezet, is daarvoor bijna onmisbaar.
Stel, we willen een programma maken over cirkels en rechthoeken. In dat programma moet ook wat tekst staan om een en ander toe te lichten.
Een 'struktuurplan' zou er zo uit kunnen zien:

STRUKTUURPLAN.

1) INLEIDING;
2) TEKST – hoe wordt een cirkel getekend;
3) TEKENING van een cirkel (HIRES);
4) TEKST – hoe wordt een rechthoek getekend;
5) TEKENING van een rechthoek (HIRES);
6) TEKST – besluit;
7) TEKENING : eind (HIRES);

In dit plan staan drie delen tekst. Die delen verenigen we tot het 'hoofdprogramma'. Zodra een tekening gewenst is schakelen we over op een procedure (= subroutine). De procedures zijn: tekening van een cirkel, van een rechthoek, van een tekst met 'EINDE' en bovendien nog een procedure 'tijd'. Deze zorgt er voor dat tekst en tekeningen gedurende enige tijd op het scherm blijven staan.
We maken gebruik van vier speciale PROCEDURE-commando's (nogmaals: een subroutine wordt in Simons' BASIC een 'procedure' genoemd).

COMMANDO: PROC
OMSCHRIJVING: geeft aan een procedure een (toepasselijke) naam.
GEBRUIK: PROC naam procedure
PROC cirkel
OPMERKINGEN: In een programmaregel met PROC mag geen enkel ander commando worden gebruikt. Alle tekens in die regel worden n.l. beschouwd als de naam van de procedure. Dit betekent dus, dat een procedure naam een regel lang zal kunnen zijn, hoewel dat in de praktijk nooit nodig zal blijken.

COMMANDO: END PROC
OMSCHRIJVING: moet in de laatste regel van een procedure staan en geeft het einde van de procedure aan.
GEBRUIK: END PROC
OPMERKINGEN: Zoals een subroutine in BASIC veelal eindigt met RETURN, zo moet een 'gesloten' procedure eindigen met END PROC. Hierdoor wordt het programma gestuurd naar de regel, volgend op die waar de procedure werd aangeroepen.

COMMANDO: CALL
OMSCHRIJVING: laat het programma springen naar de beginregel van de met name genoemde procedure.
GEBRUIK: CALL naam procedure
CALL eind
OPMERKINGEN: CALL werkt als GOTO, maar verwijst naar een procedure en niet naar een regelnummer.
CALL moet alleen in een regel staan, er mogen geen andere commando's in die regel voorkomen.
Call verwijst naar een 'open' procedure: dit is een procedure die niet eindigt met END PROC.

COMMANDO: EXEC

1.16 Gestruktureerd programmeren.

OMSCHRIJVING: laat het programma springen naar de met name genoemde procedure.
GEBRUIK: EXEC naam procedure.
OPMERKINGEN: EXEC werkt als GOSUB, maar verwijst naar een procedure en niet naar een regelnummer; EXEC moet evenals CALL alleen in ▶ een regel staan;
EXEC roept alleen 'gesloten' procedures aan: deze hebben als laatste regel: END PROC.

Praktische toepassing van deze vier commando's vindt u in het volgende programma: CIRKELENRECHTHOEK.

```
10 REM CIRKEL EN RECHTHOEK PROGRAMMA
20 PRINT"♥"
30 PRINT"HET TEKENEN VAN CIRKELS EN RECHTHOEKEN."
40 PRINT"────────────────────────────────────────"
50 PRINT"▯DOOR GEBRUIK TE MAKEN VAN HET COMMANDO"
60 PRINT"'CIRCLE' KAN MEN MOEITELOOS EEN CIRKEL"
70 PRINT"TEKENEN.
80 PRINT"WE ZULLEN DAT LATEN ZIEN MET EEN"
90 PRINT"VOORBEELD:"
100 EXEC TIJD
110 EXEC CIRKEL
120 PRINT"♥"
130 PRINT"▯OOK RECHTHOEKEN KUNNEN ZONDER MOEITE"
140 PRINT"MET HET COMMANDO 'REC' OP HET SCHERM"
150 PRINT"WORDEN GEBRACHT."
160 EXEC TIJD
170 EXEC RECHTHOEK
180 PRINT"♥"
190 PRINT"▯SIMONS' BASIC IS VOOR TEKENWERK TOCH"
200 PRINT"WEL EEN UITSTEKEND HULPMIDDEL."
210 EXEC TIJD
220 CALL EIND
230 PROC TIJD
240 FORT=1TO6000:NEXT
250 END PROC
260 PROC CIRKEL
270 HIRES 0,3
280 CIRCLE160,100,50,50,1
290 EXEC TIJD
300 NRM
310 END PROC
320 PROC RECHTHOEK
330 HIRES 4,7
340 REC30,30,150,80,1
350 EXEC TIJD
360 NRM
370 END PROC
380 PROC EIND
```

1.16 Gestruktureerd programmeren

```
390 HIRES 7,14
400 TEXT120,80,"EINDE.",1,8,16
410 EXEC TIJD
420 NRM
430 PRINT"♥"
440 END
```

Comentaar:

regel	10-220	Hoofdprogramma
regel	230-250	PROCEDURE tijd (gesloten)
regel	290-310	PROCEDURE cirkel (gesloten)
regel	320-270	PROCEDURE rechthoek (gesloten)
regel	380-440	PROCEDURE eind (open)

De procedure TIJD is nodig om de gebruiker de kans te geven tekst en tekeningen te bekijken. Merk op, dat de procedure TIJD binnen een andere procedure (CIRKEL, RECHTHOEK en EIND) wordt gebruikt. Men zou dat een procedure binnen een procedure kunnen noemen. Er mogen ten hoogste vijf 'geneste' procedures worden gebruikt.

Een bijkomend voordeel van het gebruik van deze struktuur-commando's is, dat de regels zonder meer met het commando RENUMBER van andere nummers kunnen worden voorzien. Probeert u eens RENUMBER 20000,100 en laat dan het programma starten. Geen moeilijkheden! Daarna RENUMBER 1,1, weer het programma starten. Alles blijft feilloos werken. Als we GOTO's of GOSUB's in ons programma hadden, dan zouden we na RENUMBER alle regelnummers met GOTO of GOSUB er in moeten aanpassen!

In het voorbeeld zijn de procedures uiteraard erg kort gehouden, maar niets let u om ze zo lang te maken als u wilt.

Lokale en globale variabelen

In elk programma vindt men wel een aantal variabelen: combinaties van letters, cijfers of tekens, waaraan een bepaalde waarde wordt toegekend. A=5: B1=28: A%=13: AB$="WEKA". (Zie ook deel 7, hoofdstuk 3.2.)

Meestal houdt zo'n variabele zijn waarde tijdens het afwerken van een programma, maar men kan die waarde desgewenst eenvoudig wijzigen. In het begin van het programma wordt gedefinieerd: A=7 maar later in het program bepaalt men: A=135. Van dat ogenblik af geldt de laatst vermelde waarde. Men zou zich kunnen voorstellen, dat een programmeur het aantal aanduidingen voor variabelen in een programma tot een minimum wil beperken: alleen A, A% en A$. Wel moeten de bijbehorende waarden in een programma onderdeel gewijzigd kunnen worden, maar daarna dienen de oorspronkelijke waarden weer van kracht te worden.

Dat kan gebeuren met het

COMMANDO: LOCAL

OMSCHRIJVING: geeft aan variabelen in een programma onderdeel andere waarden.

GEBRUIK: LOCAL 1e variabele, 2e variabele, 3e variabele, ...

1.16 Gestruktureerd programmeren.

LOCAL A,A%,A$
A=15: A%= 134: A$="AMSTERDAM"
OPMERKINGEN: De variabelen achter LOCAL genoemd moeten al eerder in het programma in gebruik zijn genomen. Hun waarden worden immers door LOCAL (tijdelijk) gewijzigd! Heeft u dat verzuimd, dan kan het programma – nadat u RUN hebt gegeven – vastlopen. Met RUN STOP/RESTORE is dat te verhelpen.
Merk op, dat er na LOCAL een regel komt, waarin de gewijzigde waarden staan.
Is het programma onderdeel in kwestie afgewerkt, dan kunnen de oorspronkelijke waarden van de variabelen worden hersteld met het
COMMANDO: GLOBAL
OMSCHRIJVING: geeft variabelen die met LOCAL een andere waarde kregen, hun oorspronkelijke waarde terug.
GEBRUIK: GLOBAL
OPMERKINGEN: GLOBAL heeft geen parameters.
Het woord GLOBAL duidt op het gehele (globale) programma en LOCAL verwijst naar een bepaalde plek (locatie) in dat programma.
Het nu volgende programma toont u de werking van LOCAL en GLOBAL.

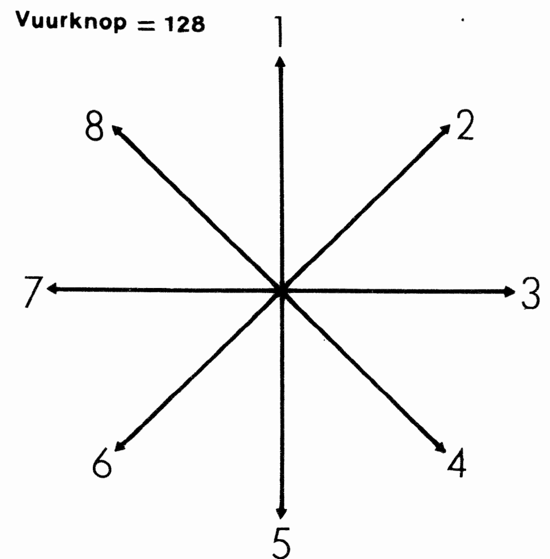
```
10 PRINT"♥"
20 A=2:A%=4:A$="3 KEER A$ "
30 FORT=ATOA%:PRINTA$:NEXT
40 PRINTA,A%,A$
50 PAUSE4
60 PRINT
70 LOCAL A,A%,A$
80 A=25:A%=31:A$="NU A$ 7 KEER"
90 FORT=ATOA%:PRINTA$:NEXT
100 PRINTA,A%,A$
110 PAUSE4
120 PRINT
130 GLOBAL
140 FORT=ATOA%:PRINTA$:NEXT
150 PRINTA,A%,A$
160 END

READY.
```

Commentaar:
20 – hier worden A, A% en A$ al in gebruik genoemen;
40 – deze regel zet A, A% en A$ met hun waarden op het scherm;
70 – LOCAL noemt de variabelen weer, die in regel
80 – een nieuwe waarde krijgen;
100 – drukt A, A% en A$ met hun nieuwe waarden af;
130 – GLOBAL herstelt de oorspronkelijke waarden, en die worden in regel
150 – weer afgedrukt.

8/1.17

Joystick, lightpen en paddle.

Met de variabele JOY kan men op eenvoudige manier vaststellen, in welke richting de joystick wijst.
In neutrale stand heeft JOY de waarde 0. Wordt de vuurknop ingedrukt, dan krijgt JOY de waarde 128 en voor de acht mogelijke richtingen van de joystick krijgt JOY waarden van 1-8.



De joystick werkt maar in een van de twee 'ports', en wel port 2.
Hoe JOY in een programma wordt gebruikt ziet u in 'DE BALLON':

```
10 HIRES0,15:MULTI0,5,13
20 LINE0,120,159,120,1
30 BLOCK0,121,159,199,3
50 LINE0,110,40,80,2:LINE40,80,70,20,2:LINE70,20,110,40,2
60 LINE110,40,140,100,2:LINE140,100,159,80,2
70 PAINT80,110,2
80 DESIGN 0,52992
100 @...........BB...........
110 @.........BBBBBB.........
120 @.......BBBBBBBBBB.......
130 @......BBBBBBBBBBBB......
140 @.....BBBBBBBBBBBBBB.....
150 @....BBBBBBBBBBBBBBBB....
160 @...BBBBBBBBBBBBBBBBBB...
170 @...BBBBBBBBBBBBBBBBBB...
180 @...BBBBBBBBBBBBBBBBBB...
190 @...BBBBBBBBBBBBBBBBBB...
200 @....BBBBBBBBBBBBBBBB....
210 @.....BBBBBBBBBBBBBB.....
220 @.....BBBBBBBBBBBBBB.....
230 @......BBBBBBBBBBBB......
240 @......BBBBBBBBBBBB......
250 @.....B.B.BBBBBB.B.B.....
260 @.....B.B..BBBB..B.B.....
270 @.....B..B..BB..B..B.....
280 @.....B...BBBBBB...B.....
290 @.....B....BBBB....B.....
300 @..........BBBB..........
310 MOB SET1,60,0,0,0:X=80:Y=100
320 RLOCMOB1,X,Y,3,1
330 IF JOY=1 THEN Y=Y-1:GOTO420
```

1.17 Joystick, lightpen en paddle

```
340 IF JOY=2 THEN Y=Y-1:X=X+1:GOTO420
350 IF JOY=3 THEN X=X+1:GOTO420
360 IF JOY=4 THEN X=X+1:Y=Y+1:GOTO420
370 IF JOY=5 THEN Y=Y+1:GOTO420
380 IF JOY=6 THEN Y=Y+1:X=X-1:GOTO420
390 IF JOY=7 THEN X=X-1:GOTO420
400 IF JOY=8 THEN X=X-1:Y=Y-1:GOTO420
410 IF JOY=128 THEN X=20:Y=50:GOTO320
420 IF X>300 THENX=300:GOTO320
430 IF X<20 THEN X=20:GOTO320
440 IF Y>220 THEN Y=220:GOTO320
450 IF Y<50 THEN Y=50:GOTO320
460 GOTO320
```

Op een dergelijke manier kan men gebruiken de commando's PENX en PENY.
COMMANDO: PENX
OMSCHRIJVING: leest de waarde van de X-coordinaat van een lichtpen.
GEBRUIK: PENX =
OPMERKINGEN: De waarden achter PENX lopen van 0 t/m 320.
Eerst moet altijd PENX worden afgelezen, pas daarna PENY.
COMMANDO: PENY
OMSCHRIJVING: leest de waarde van de Y-coordinaat van een lichtpen.
GEBRUIK: PENY =
OPMERKINGEN: De waarden achter PENY lopen van 0 t/m 200.
Altijd eerst PENX aflezen, dan pas PENY.

Een toepassing van beide commando's vindt u in de handleiding bij Simons' BASIC op blz. 12-2.

Om vast te stellen hoever een paddle is gedraaid gebruikt men het
COMMANDO: POT
OMSCHRIJVING: leest de weerstandswaarde van een paddle.
GEBRUIK: POT (n) =
OPMERKINGEN: n kan 0 of 1 zijn, en bepaalt welke paddle er moet worden afgelezen.
Het getal achter POT(n) loopt van 0 t/m 255.
Een toepassing van POT(n) vindt u in de handleiding op blz. 12-3 en 12-4.

8/1.18

Kort overzicht van enkele andere BASIC-uitbreidingen

We zijn aan het einde gekomen van het verhaal met zo'n 110 Simons' BASIC commando's. Of al die commando's wezenlijk bijdragen tot en beter gebruik van de Commodore 64 zal wel een kwestie van persoonlijke smaak zijn: iedere gebruiker zal die commando's paraat hebben die hem het meest interesseren.
De voorbeeldprogramma's zijn met opzet kort gehouden, ze dienen alleen om de mogelijkheden aan te geven. Wat het toepassen betreft: ook hier geldt 'oefening baart kunst'. Dat er behoefte aan dit soort hulpprogramma's bestaat is duidelijk. Er zijn nog enkele programma's op de markt, die in grote trekken dezelfde mogelijkheden bieden als Simons' BASIC. Maar qua aanschafprijs, uitvoering (cartridge) en mogelijkheden blijkt Simons' BASIC toch wel favoriet.
De komst van o.a. de Commodore 128 zal dit soort programma's wel overbobig maken: BASIC 7.0 beschikt over de meeste mogelijkheden die de hulpprogramma's bieden.
Voor wie in Simons' BASIC zijn draai niet kan vinden zijn er nog een aantal andere BASIC-uitbreidingen waarvan er hier twee kort besproken worden:

BREDEN'S BASIC
Auteur: Simon Breden
Uitgave: Visions Ltd.
1 Felgate Mews
Studland Street
London W6 9JT
England

Dit programma wordt geleverd op cassette, disk of insteekprogramma (cartridge)
De commando's bestrijken de volgende gebieden:

Sprites	11 commando's
Graphics	30 commando's
Geluid	22 commando's
Input/Output	11 commando's
Disk	4 commando's
Bewerkingen met getallen	21 commando's
Fouten signaleren	4 commando's
Programmahulpjes	11 commando's
Programma struktuur	11 commando's

Deze 125 commando's worden besproken in een fraai uitgevoerde handleiding in de engelse taal. Het opzoeken van een commando gaat snel door de handige indeling met gekleurde tabbladen.
Uitgewerkte programma's staan er vrijwel niet in, de disk geeft enkele demonstratie programma's. Men dient de listing daarvan goed te bestuderen.
Bij Graphics ontbreken commando's voor cirkel, boog en ellips. Die moeten met PLOT geconstrueerd worden door

1.18 Overzicht BASIC-uitbreidingen

gebruik te maken van de COS/SIN formule (zie dit deel, hoofdstuk 1.3 blz.7).

SUPER BASIC voor de Commodore 64
Auteur: A. Dijkhuizen
Uitgave: Stark-Texel

Het programma wordt geleverd op cassette.
De commando's bestrijken de volgende gebieden:

Programmeeromgeving	7 commando's
Programmeerhulpjes	11 commando's
Disk	16 commando's
Nieuwe funkties	18 commando's
Grafisch werken	15 commando's
Grafieken	6 commando's
Speciale commando's	8 commando's
Gestruktureerd programmeren	9 commando's

Dit 90-tal commando's wordt besproken in een – uiteraard nederlandstalige – handleiding.
Super BASIC heeft geen commando's voor multicolour graphics, sprites en geluid.
Slechts enkele commando's zijn voorzien van een voorbeeld met listing. De gebruiker moet zelf maar zien te ontdekken hoe een commando werkt, en dat is bij commando's als b.v. MIX en SORT niet zo eenvoudig.

8/2

LOGO

INHOUD

8/2.1 Inleiding
8/2.2 Graphic in LOGO
8/2.3 Opslaan, inladen en wijzigen
8/2.3 Rekenen en programmeren
8/2.5 Functies tekenen
8/2.6 Woorden en lijsten
8/2.7 Spelen met tekst
8/2.8 Foutmeldingen in LOGO
8/2.9 LOGO programma structuur
8/2.10 Recursie: perspectief-tekenen
8/2.11 De utility diskette
8/2.12 LOGO wetenswaardigheden
8/2.13 Commando's en primitieven
8/2.14 OPTION
8/2.15 Een behendigheidsspel in LOGO
8/2.16 SPRITES
8/2.17 De utility diskette (vervolg)
8/2.18 Het werken met de sprite editor SPRED
8/2.19 Demonstratie programma's
8/2.20 Verwarringen met LOGO
8/2.21 Muziek met LOGO
8/2.22 LOGO en Machinetaal
8/2.23 Files op de utility diskette
8/2.24 Het opslaan van routines
8/2.25 Belangrijke adressen

8/2.1

Inleiding.

Na BASIC, Simons' BASIC en machinetaal is LOGO de eerste niet tot de standaarduitrusting van uw Commodore 64 behorende computertaal waar dieper op ingegaan wordt. LOGO is eigenlijk pas mogelijk geworden door de voortschrijdende techniek. In de begintijd van de computer waren de beschikbare geheugenruimtes te klein van omvang en als gevolg hiervan moesten de talen waarin met de machine geconverseerd kon worden 'computer-vriendelijk' zijn.

Later, door de uitgebreidere mogelijkheden, kon er 'mens-vriendelijker' gewerkt worden in in 1980 kwam uit een onderwijssomgeving (onder leiding van Seymour Papert) met de publicatie van 'MINDSTORMS' een nieuwe visie op de computer als hulpmiddel in het onderwijs tot stand. Na ruim 15 jaar onderzoek was de computertaal LOGO een feit. Het meest bekend is deze taal geworden door de zogenaamde 'Turtle graphics' (grafische mogelijkheden met behulp van de schilpad). Met LOGO kan echter nog veel meer worden gedaan, er kunnen rekenkundige manipulaties worden verricht, er kan met woorden en lijsten worden gewerkt terwijl ook muziek en sprites tot de mogelijkheden behoren. In deze aanvulling kunt u kennis maken met de 'Turtle graphics', u gaat met getallen werken en rekenen en ook de woorden en lijsten zullen aan de orde komen. U zult zelf ontdekken wat de grote kracht van LOGO in het onderwijs is, namelijk het heel erg simpel (of primitief) beginnen en daarna uitbreiden op basis van de reeds aanwezige kennis. Deze dient dan weer als bouwsteen voor nog gecompliceerdere zaken, enzovoort, enzovoorts. Ook met een tweede krachtige eigenschap van LOGO zult u kennis maken, namelijk de recursieve mogelijkheden. Dit betekent dat in een proces bij elke nieuwe stap gebruik gemaakt wordt van de resultaten van de voorgaande stappen; dat wil zeggen het proces maakt van zichzelf gebruik.

8/2.2

Graphic in LOGO

Na het inladen van LOGO verschijnt op het scherm behalve een welkomstbeeld ook een vraagteken. Met dit teken zal LOGO zich altijd melden als er instructies nodig zijn. U wilt gaan werken met de grafische mogelijkheden, dat wil zeggen dat u gaat tekenen en dus geeft u het commando 'DRAW'. Het bovenste gedeelte van het scherm verandert in een tekenblad met daarin geplaatst een kleine driehoekige cursor (de TURTLE) die altijd zal bewegen in de richting van de punt. Op het onderste deel van het scherm verschijnt weer het vraagteken, de 'input-prompt'.

De simpelste figuur om te tekenen is een rechte lijn. Als u ingeeft FORWARD 100 zal de cursor zich 100 stappen vooruit begeven en een lijn op het scherm trekken. De spatie tussen FORWARD en 100 is absoluut noodzakelijk, in LOGO is de spatie een scheidingsteken dat volgens de syntax beslist niet mag worden weggelaten. Als u het commando ingeeft als FD 100 gebeurt precies hetzelfde, want deze afkorting heeft dezelfde betekenis als de volledige instructie. Men heeft deze elementaire instructie de naam primitive (primitief) gegeven. LOGO kent vele primitieven. Bovendien kunt u er zelfgemaakte primitieven aan toevoegen, hierop komen we later terug.

Een in gecompliceerdheid op de lijn volgende figuur is het vierkant. Als u naar de opbouw kijkt, dan is deze als volgt: een bepaalde afstand vooruit, een rechte hoek naar rechts of links (90 graden), dezelfde afstand als zoëven vooruit, weer 90 graden draaien enzovoorts, tot we weer terug zijn op ons uitgangspunt.
Een veel kortere formulering is natuurlijk: 'herhaal 4 maal [stap vooruit 90 graden draaien naar rechts]'. Beide mogelijkheden kent LOGO, de volgende reeksen instrukties tekenen allen een vierkant op het scherm.

1) FORWARD 100
   RIGHT 90
   FD 100 RT 90 ;AFKORTEN IS TOEGESTAAN
   FD 100 RT 90 FD 100 RT 90

In één van de regels is een puntkomma opgenomen, wat in LOGO achter de puntkomma staat is commentaar en wordt niet uitgevoerd (vergelijk met REM in BASIC). Als u na het intikken en uitvoeren van de derde regel op de ^ (pijljte omhoog) toets drukt wordt de bovenstaande regel herhaald. Dit kan u veel tikwerk besparen. U ziet dat u mag afkorten en dat u meer instructies op een regel mag plaatsen.

Een tweede mogelijkheid:

2.2 Graphic in LOGO

2) FD 100 RT 90 FD 100 RT 90 FD 100 RT 90 FD 100 RT 90

Let u vooral op de spaties, want deze vormen immers ht scheidingsteken in LOGO!

De derde manier tenslotte:

3) REPEAT 4[FD 100 RT 90]

U heeft nu kennis gemaakt met het begrip 'primitief'. Er is standaard een groot aantal primitieven aanwezig in LOGO, een aantal treft u hieronder aan, met hun (eventuele afkorting en een korte beschrijving.

BK BACK
De cursor beweegt het aangegeven aantal stappen in tegengestelde richting.

BG BACKGROUND
Bepaalt de kleur van de achtergrond met behulp van de cijfers 0 tot en met 15. De nummers komen overeen met de kleurnummers uit uw handleiding, verminderd met 1.

CS CLEARSCREEN
Het grafische scherm wordt gewist, maar de cursor blijft op de positie waar hij het laatste stond.

DOUBLECOLOR
Maakt het werken met twee kleuren in een 8 x 8 matrix mogelijk.

DRAW
Brengt LOGO in de tekenmode. Indien LOGO reeds in de tekenmode is wordt het scherm gewist en gaat de cursor naar de uitgangspositie.

DRAWSTATE
Geeft u informatie over 9 zaken betreffende de toestand van het scherm en de cursor.

FD FORWARD
Beweegt de cursor het aangegeven aantal stappen voorwaarts.

[F5] FULLSCREEN
Maakt het hele scherm tot grafisch scherm.

HEADING
Geeft de richting aan waarin de grafische cursor wijst. Omdat een volledige cirkel 360 graden omvat is het resultaat van HEADING altijd een getal tussen 0 en 360.

HT HINDETURTLE
Maakt de grafische cursor onzichtbaar. Een niet onbelangrijk commando want het tekenen gaat sneller met een verborgen cursor.

HOME
Brengt de cursor naar zijn uitgangspositie zonder het scherm te wissen.

LT LEFT
Draait de cursor het aangegeven aantal graden linksom. Ook gebroken en negatieve getallen alsmede 0 zijn toegestaan.

ND NODRAW
Geeft u een leeg tekstscherm over het gehele oppervlak. U kunt ook tijdelijk het volledige tekstscherm bekijken zonder uw tekening te wissen met behulp van [F1]; hetzelfde als het commando TEXTSCREEN.

NOWRAP

2.2 Graphic in LOGO

Als u tekent en over de rand van het scherm komt dan wordt de tekening aan de andere zijde van het scherm voortgezet.
NOWRAP schakelt deze eigenschap uit. WRAP of nogmaals NOWRAP schakelen de oude toestand weer in, evenals de commando's SINGLECOLOR en DOUBLECOLOR.

PC PENCOLOR
Bepaalt de kleur van de lijnen die getrokken worden.
Ook hier de cijfers 0 tot en met 15 in overeenkomst met de kleurentabel in uw handleiding verminderd met 1.

PD PENDOWN
Zet als het ware de punt van de tekenstift op het papier zodat er lijnen getrokken kunnen worden. De normale uitgangstoestand van LOGO.

PU PENUP
Haalt de tekenstift van het papier. De cursor maakt wel de opgedragen bewegingen maar laat geen sporen na.

RT RIGHT
Draait de grafische cursor het aangegeven aantal graden naar rechts.

SETH SETHEADING
Draait de cursor het aangegeven aantal graden ten opzichte van de uitgangspositie.
0 is recht naar boven en het tellen gebeurt in de richting van de wijzers van de klok.

SETSHAPE
Commando met betrekking tot SPRITES. Hierop wordt in de volgende aanvulling uitgebreid ingegaan.

SETX
Beweegt de cursor horizontaal naar de aangegeven positie.

SETXY
Beweegt de cursor naar de aangegeven X en Y coördinaten. Er zijn dus twee getallen nodig bij dit commando.
Denkt u om de spaties? Bovendien moet een negatieve waarde voor Y tussen haakjes worden geplaatst.

SETY
Beweegt de cursor naar de aangegeven verticale positie.

SHAPE
Ook een commando dat met SPRITES te maken heeft. Komt volgende keer aan de orde.

ST SHOWTURTLE
Maakt de grafische cursor zichtbaar. Uitgangspositie van LOGO.

SINGLECOLOR
Maakt het werken met slechts één kleur in een 8 x 8 matrix mogelijk.

[F3] SPLITSCREEN
Bovengedeelte van het scherm voor grafisch werk, onderste gedeelte voor tekst. Met de cijfers van 0 tot 24 kunt u de verhouding tussen tekenblad en tekstblad wijzigen.

Er moeten dan haakjes gebruikt worden: (SPLITSCREEN 5). Het cijfer geeft het aantal regels van het tekstgedeelte van het scherm aan.

STAMPCHAR
Plaatst een letter op de plaats van de cursor op het grafische scherm.

2.2 Graphic in LOGO

TELL
Met dit commando geeft u aan welke sprite alle grafische commando's ontvangt. Cijfers van 0-7.

TEXTBG
Bepaalt de kleur van het tekstscherm. Dezelfde kleurcodes als bij de overige kleurcommando's

TEXTCOLOR
Bepaalt de tekstkleur, werkt niet bij STAMPCHAR. Omdat hier de grafische cursor bij betrokken is moet u in dat geval PENCOLOR gebruiken. De normale codering voor de kleuren.

[F1] TEXTSCREEN
Maakt het gehele scherm tot tekstscherm.

TOWARDS TOWARDS :X :Y
laat de grafische cursor naar het punt met coördinaten X en Y gaan.

De dubbele punt ':' voor een variabele in LOGO betekent altijd 'de waarde van die variabele'.

WHO
Dit geeft aan welke turtle (sprite) door TELL het laatste werd geactiveerd.

WRAP
Zorgt ervoor dat een lijn die aan een kant van het scherm afloopt aan de tegenovergestelde kant wordt voortgezet.

XCOR
Geeft de X coördinaat van de grafische cursor.

YCOR
Geeft de Y coördinaat van de grafische cursor.

U kunt met al deze commando's naar hartelust experimenteren in de directe mode. Alle opdrachten worden direct uitgevoerd, maar er zijn problemen.

Na het uitvoeren van de opdracht is de taak van de computer volbracht en is het betreffende commando vergeten. Daardoor is het een erg tijdrovende manier om ingewikkelde tekeningen te maken.

Een methode om tijd te besparen is om aan de computer nieuwe woorden te leren. Dat wil zeggen dat u een aaneenschakeling van primitieven onder een nieuwe naam in het computergeheugen zet.

Als u dan die nieuwe naam aanroept dan wordt de gehele reeks commando's automatisch uitgevoerd. Men noemt deze pakketten instructies 'procedures' (in zekere zin zijn ze vergelijkbaar met subroutines in BASIC, het verschil is dat een procedure op zich zelf kan functioneren).

U deelt aan de computer mee dat er een procedure gedefinieërd gaat worden door het woord 'TO'.

Voorbeelden:

```
TO  VIERKANT
REPEAT 4 [FD 50 RT 90]
END
```

```
TO  ZESHOEK
REPEAT 6 [FD 50 RT 60]
END
```

2.2 Graphic in LOGO

Op deze manier kunt u doorgaan en de vocabulaire van uw computer 'ongelimiteerd' uitbreiden. De op deze wijze gecreëerde procedures zijn echter statisch. Dat wil zeggen dat ze altijd identiek zijn, het enige verschil qua effect bij de uitvoering kan zitten in de plaats van de grafische cursor als u de procedure aanroept. Die plaats van de grafische cursor bepaalt immers waar getekend wordt. Overigens, als u de lijst primitieven heeft bekeken heeft u gezien dat de plaats van de grafische cursor ook via een primitief te bepalen valt.
Het primitief FORWARD heeft als u het op zichzelf gebruikt een input nodig. Indien u FORWARD in een procedure hebt staan en u neemt na de procudurenaam een vermelding op dat er input nodig is, dan heeft u uw procedure veel flexibeler gemaakt. Door diezelfde naam namelijk achter de primitief te vermelden wordt deze waarde doorgegeven.

Voorbeeld:

```
TO ZESHOEK :AFMETING
REPEAT 6 [FD :AFMETING RT 60]
END
```

Nu kunt u ZESHOEK verder op dezelfde manier gebruiken als de primitieven, maar er is na het woord ZESHOEK een input nodig van een waarde 'AFMETING'. Dat dit nodig is wordt aangegeven door de dubbele punt. ZESHOEK 100 en ZESHOEK 60 tekenen zeshoeken met verschillende afmetingen.

Een van de sterke eigenschappen van LOGO is de mogelijkheid om primitieven en procedures door elkaar te gebruiken. Dit is eigenlijk heel logisch, een procedure is immers een woord dat UW LOGO begrijpt, in wezen dus een primitief in UW LOGO. 'UW' staat met opzet in hoofdletters, want het zal u duidelijk zijn dat uw versie van LOGO (standaard primitieven aangevuld met uw eigen procedures) een geheel andere LOGO zal zijn dan de LOGO versie van uw buurman.

Een voorbeeld van dit door elkaar gebruiken van primitief en procedure:

```
TO TEKENING
REPEAT 6 [FD 20 RT 80 ZESHOEK
50]
END
```

Probeert u nu maar het commando:

```
TEKENING
```

Een tweede sterke eigenschap van LOGO is de mogelijkheid om een procedure te gebruiken in die procedure zelf. Dat wil zeggen dat u de procedure als het ware zichzelf laat aanroepen. U ziet gelijk al dat u hier de mogelijkheid krijgt om in LOGO zaken te gaan herhalen. Als u weet hoe u het programma moet stoppen kunt u zelf bepalen hoeveel maal er herhaald wordt.
Door op (CTRL) en G te drukken kunt u het tekenen onderbreken, maar u begrijpt wel dat u in de toekomst een soort IF....THEN constructie zult tegenkomen.

Als voorbeeld van deze recursie:

```
TO ZES :MAAT
FD :MAAT
RT 60
ZES :MAAT
END
```

2.2 Graphic in LOGO

Deze procedure tekent ook een zeshoek op het scherm. Het verschil tussen ZES en ZESHOEK zit in het feit dat bij ZES het tekenen alsmaar doorgaat, terwijl het bij ZESHOEK slechts eenmaal gebeurt. Nu u de aanzet kent zult u niet veel moeite hebben met de constructie:

```
TO DRAAIVIER :MAAT :HOEK
VIERKANT :MAAT
RT :HOEK
DRAAIVIER :MAAT + 3 :HOEK
END
```

U begrijpt dat u hiervoor ook de procedure VIERKANT :MAAT nodig heeft.

```
TO VIERKANT :MAAT
REPEAT 4 [FD :MAAT RT 90]
END
```

Een ander leuk tekensysteem is:

```
TO VEELHOEK :MAAT :HOEK
FD :MAAT
RT :HOEK
VEELHOEK :MAAT :HOEK
END
```

Door te experimenteren met VEELHOEK 40 30 en DRAAIVIER 40 30 enzovoorts kunt u al leuke figuren maken. U moet natuurlijk wel zorgen dat de procedures die u aanroept ook in het geheugen van de computer bekend zijn.

U kunt het geheel nog veel interessanter maken als u weet dat LOGO niet alleen maar genoegen neemt met een INPUT bestaande uit getallen, maar dat u ook een expressie kunt gebruiken. LOGO evalueert eerst de expressie en gebruikt de uitkomst als INPUT.

Een leuk voorbeeld is:

```
TO VEELSPIR :MAAT :HOEK :STAP
FD :MAAT
RT :HOEK
MAKE "MAAT :MAAT + :STAP
VEELSPIR :MAAT :HOEK :STAP
END
```

Nu kunt u door simpel VEELSPIR 2 20 3 (of iets dergelijks) fraaie spiralen maken. U heeft hier ook al kennis gemaakt met het commando 'MAKE' dat een waarde toekent aan een variabele.
U ziet ook dat de naam van een variabele vooraf gegaan moet worden door een stel quotes, dubbele aanhalingstekens.
Een van de directe gevolgen van de recursieve eigenschappen van LOGO is dat u met simpele middelen ingewikkelde zaken kunt laten gebeuren.

Wat u tot nu toe heeft gedaan bent u allemaal kwijt als u de computer uitzet. Bovendien krijgt u de tekeningen op het scherm maar u bent nog niet zover dat u ze op papier kunt afdrukken of kunt bewaren op diskette. Voor de bezitters van een Commodore 64 die geen diskdrive bezitten is LOGO helaas niet bereikbaar, de taal is volledig disk-georiënteerd. Als u eenmaal de mogelijkheid heeft om door u ontwikkelde procedures op schijf op te slaan en deze naderhand naar believen weer op te roepen kunt u natuurlijk de vocabulaire van UW LOGO ongelimiteerd uitbreiden.

8/2.3

Opslaan, inladen en wijzigen

Als u het geheugengedeelte waarin de procedures zijn opgeslagen ziet als een werkblad, dan kunt u de volledige inhoud van het werkblad op diskette opslaan via het commando:

SAVE "TEKENINGEN

Dit slaat AL uw op dat moment aanwezige procedures op in een bestand 'TEKENINGEN'. U kunt echter ook selectief bepaalde procedures opslaan via:

SAVE "DEELTEK [VIERKANT ZESHOEK]
Deze vorm slaat alleen de procedures VIERKANT en ZESHOEK op diskette op onder de naam DEELTEK. Op deze wijze kunt u natuurlijk een bibliotheek opbouwen van door u ontwikkelde procedures. Ook hier weer een stel quotes voor de bestandsnaam, een tweede stel achter de naam hoeft u niet te gebruiken.

Indien u nieuwsgierig bent naar de inhoud van een diskette, via het commando:
CATALOG
verschijnt op uw scherm een lijst met de op de diskette voorkomende bestanden. U zult dan ook zien dat LOGO de bestanden duidelijk identificeert als LOGO bestanden, want elke naam wordt gevolgd door het achtervoegsel '.LOGO'.

Het weer inladen van bestanden van uw diskette gaat via het commando:
READ "BESTANDSNAAM
Als het inladen is voltooid zal LOGO dit melden met het antwoord:
BESTANDSNAAM DEFINED
U vraagt zich na verloop van tijd natuurlijk wel af wat voor procedures er in een bepaald bestand zitten.
Als u dit bestand heeft ingeladen in de computer en u geeft LOGO de instructie 'POTS' (een afkorting voor PRINTOUT TITLES) dan krijgt u een lijst van de in het werkblad aanwezige procedures.

Om alle commando's in een bepaalde procedure te zien volstaat het om te tikken
PO PROCEDURENAAM
(PO is een afkorting van PRINTOUT). Dit commando is vergelijkbaar met het commando LIST rn1-rn2 in BASIC. Let u er op dat u hier geen "" moet gebruiken!
U heeft natuurlijk ook de mogelijkheid om procedures en bestanden te verwijderen. Om procedures uit uw werkblad te verwijderen geeft u de instructie:

ERASE [PROCEDURE1 PROCEDURE2....]

Hierbij geen quotes gebruiken.

2.3 Opslaan, inladen en wijzigen

Om een bestand van de diskette te verwijderen is het commando:
ERASEFILE "BESTANDSNAAM
(wel "")
in LOGO aanwezig.

In BASIC kent u de instructie NEW, in LOGO heeft u daarvoor een equivalent, met het commando
GOODBYE
verwijdert u alle aanwezige procedures uit het computergeheugen.

Ook is het in LOGO mogelijk om de resultaten van uw werk rechtstreeks op diskette te bewaren. Gemaakte tekeningen kunt u opslaan en weer inlezen met behulp van de commando's:
SAVEPICT "PAARD
en
READPICT "PAARD

Met
ERASEPICT "PAARD
verwijdert u de tekening PAARD weer van de diskette. Als u een tekening op diskette heeft opgeslagen ziet u dat de informatie in twee gedeelten is opgesplitst, namelijk twee bestanden die eindigen met het achtervoegsel 'PIC1' en 'PIC2'. Het ene bestand bevat de informatie over de vorm van de tekening, het ander bestand heeft de kleurinformatie als inhoud. U kunt echter ook zien dat het bewaren van tekeningen op diskette veel plaats kost. Als u een simpel vierkantje op diskette opslaat ziet u dat de twee files samen 38 blokken in belag nemen.
Tenzij het noodzakelijk is om de complete tekening op te slaan kunt u veel beter de procedures om de tekening te maken opslaan.

Wanneer u echt gaat programmeren in LOGO heeft u nog een paar hulpmiddelen nodig. Een zeer belangrijk middel is het afdrukken van de inhoud van een bestand of procedure op papier, want het bewerken van een bestand alleen op het scherm is tamelijk lastig. Het toverwoord hiervoor is simpel:
PRINTER
Dit woord zorgt er voor dat de output die anders naar het scherm gaat nu op papier komt te staan.

PRINTER
PO [KOE KOP STAART POTEN]
NOPRINTER

zorgt ervoor dat u de inhoud van de procedures 'KOE', 'KOP', 'STAART' en 'POTEN' op papier krijgt te staan, waarna u weer terug gaat naar LOGO met alleen maar output naar het scherm.

PO PROCEDURES geeft u de inhoud van alle aanwezige procedures op papier.

LOGO werkt standaard met de 801 printer van Commodore, maar via een passende interface kunt u ook met andere printers werken.
Als u een procedure wilt veranderen gebeurt dit in de 'EDIT' mode. Via:
TO procedurenaam
komt u in de EDIT mode. Ook kunt u via EDIT in de EDIT mode komen, dan kunt u de laatste ingevoerde procedure wijzigen. Bent u in de EDIT mode, dan is er een serie nuttige toetsaanslagen voorhanden:

(CTRL) P - de cursor gaat naar de vorige regel.
(CTRL) N - de cursor gaat naar de volgende regel.
(CTRL) O - er wordt een regel tussenge-

2.3 Opslaan, inladen en wijzigen

voegd.
(CTRL) A - de cursor gaat naar het begin van de regel.
(CTRL) L - de cursor gaat naar het einde van de regel.
(CTRL) D - het teken onder de cursor wordt verwijderd, merk op dat dit iets anders is dan het gebruiken van de DELETE toets, want hiermee wordt het teen links van de cursor verwijderd.
(CTRL) K - verwijdert de tekens rechts van de cursor tot het einde van de regel.

Het is natuurlijk erg gevaarlijk om in een procedure te gaan veranderen zonder de naam van deze procedure ook te veranderen. Immers, als u de procedure waarover het gaat ook in andere procedures heeft gebruikt, verandert alles wat u heeft geproduceerd op een ongecontroleerde wijze. Als u in de EDIT mode bent om uw procedure te wijzigen gaat u met de cursor naar de procedurenaam en wijzigt deze. Daarna gaat u weer met de cursor naar beneden en u handelt de zaken af zoals u gewend bent. Uw gewijzigde procedure is dan met een nieuwe naam in het werkblad aanwezig of op diskette opgeslagen, al uw eerdere werk is niet voor niets geweest.

8/2.4

Rekenen en programmeren

Net zoals elke computertaal die zich zelf respecteert kent LOGO de begrippen INPUT en OUTPUT. Het begrip INPUT heeft u reeds leren kennen bij het maken van procedures die gegevens nodig hadden. OUTPUT gebruikt u om de resultaten van berekeningen zichtbaar te maken via PRINT of om het resultaat van een berekening in de ene procedure mee te geven als invoer bij een andere procedure.
Een voorbeeld:

```
TO GEMIDDELDE :X :Y
OUTPUT (:X + :Y) / 2
END
```

Na het intikken van GEMIDDELDE 4 6 wordt

```
RESULT: 5
```

afgedrukt.

PRINT heeft dezelfde functie als in BASIC, het maakt uitvoer zichtbaar op het scherm, u kunt bijvoorbeeld met

```
PRINT (GEMIDDELDE 4 5)
```

het resultaat zichtbaar maken. U kunt echter ook gelijk het resultaat gebruiken als invoer in een 'andere' procedure.
PRINT (GEMIDDELDE 1 2) + (GEMIDDELDE 4 5)

De uitkomst 6 wordt nu afgedrukt zonder de tekst RESULT:
U kunt zelfs nog verder gaan:

```
PRINT (GEMIDDELDE 1 2) 3)
```

De uitkomst 2.25 wordt keurig afgedrukt. De haakjes zijn in simpele LOGO regels niet beslist voorgeschreven, maar zij maken de tekst veel leesbaarder. Het is dan ook een goede gewoonte om u er aan te wennen haakjes te gebruiken wanneer dit nuttig is.
Diverse elementen en functies die in andere computertalen aanwezig zijn kent LOGO natuurlijk ook.
De constructie IF... THEN heeft in LOGO de vorm:

```
IF: ‹voorwaarde› THEN ‹opdracht›
```

Een voorbeeld bij het een bepaald aantal malen herhalen van een procedure:

```
TO TEKENING :MAAL
IF :MAAL = 0 THEN STOP
ZESHOEK 30
RT 40
TEKENING :MAAL −1
END
```

Als u voor de zeshoek nog geen procedure op de diskette heeft staan, dan maakt u hem als volgt:

2.4 Rekenen en programmeren

```
TO ZESHOEK :MAAT
REPEAT 6[FD :MAAT RT 60]
END
```

Het zal u duidelijk zijn wat er gebeurt als u intikt:

TEKENING 6
a) LOGO test of 6 gelijk is aan 0. Dit is niet waar, dus gaat LOGO verder met de volgende regel, tekent de ZESHOEK en draait 40 graden naar rechts. Vervolgens wordt de procedure TEKENING 5 aangeroepen. Hier wordt getest of 5 gelijk is aan 0 enzovoorts. Er komt een moment waarop getest wordt of 0 gelijk is aan 0, dit is WAAR en dan wordt de instructie na THEN uitgevoerd.

Foutzoeken

Om fouten op te sporen in LOGO procedures of bestanden heeft u diverse hulpmiddelen tot uw beschikking.

U kunt in de procedure die u aan het testen bent voor de regel met de IF....THEN conditie een PR (afkorting voor PRINT) instructie opnemen waarmee u de variabele waarom het gaat op het scherm laat afdrukken. In de procedure TEKENING betekent dit het tussenvoegen van een regel (juist voor IF....THEN) als volgt:
PR :MAAL

Ook kunt u de uitvoering op een zodanige wijze onderbreken dat u de variabelen kunt bekijken, om daarna het programma weer voort te laten zetten. U onderbreekt daartoe de uitvoering met behulp van ‹CTRL› Z en kunt dan met behulp van PR, PO, ST enzovoorts aan de weet komen wat u interesseert. Door het intikken van CO (of voluit CONTINUE) wordt de uivoering van de procedure voortgezet.

U heeft ook de beschikking over de functie TRACE. TRACE laat de instructieregel, die aan de beurt is om uitgevoerd te worden, zien en wacht op een toetsindruk alvorens deze instructie uit te voeren. Daarna wordt de volgende regel afgebeeld enzovoorts. Het zal u niet verbazen dat de andere foutzoetmethoden ongewijzigd van kracht blijven tijdens het ingeschakeld zijn van de TRACE functie, evenmin zal het niet vreemd voor u zijn dat NOTRACE de TRACE functie weer uitschakelt.

Cijfers en functies.

LOGO kent twee soorten getallen, namelijk 'integer' en 'real' getallen. Net als bij andere talen zijn integers gehele getallen; 'real numbers' hebben een decimaal gedeelte. Ook in LOGO zijn er grenzen aan de mogelijkheden, het kleinste getal dat voorgesteld kan worden is 1.999N38, in onze woorden vertaald 1.999✱10^−38. Het grootste getal dat in LOGO mogelijk is: 1.7E38. Dit zijn voor u min of meer bekende grenzen. Het bereik van gehele getallen is een stuk groter dan bijvoorbeeld bij BASIC, de grenzen zijn hier 2147483647 en −2147483647. U bent dus met uw berekeningen niet zo snel aan de grenzen der mogelijkheden. Rekenen in LOGO lijkt erg veel op ons rekenen, de normale rekenkundige operaties zijn mogelijk:

Optellen, aangegeven met het teken '+'.

Aftrekken, aangegeven met het teken '−'.

2.4 Rekenen en programmeren

Vermenigvuldigen, aangegeven met het teken '*'.

Delen, aangegeven met het teken '/'.

Machtsverheffen is in LOGO niet rechtstreeks mogelijk, hiervoor moet gebruik gemaakt worden van een procedure 'EXPONENT', die u daarvoor zelf moet ontwikkelen.
De volgorde van de bewerkingen wijkt niet van de gangbare regels af. Allereerst kijkt LOGO naar haakjes, wat daarbinnen staat wordt eerst uitgevoerd. Verder is de volgorde: eerst vermenigvuldigen en delen, daarna optellen en aftrekken, dat alles van links naar rechts. Belangrijk om te onthouden is dat LOGO eerst gaat rekenen voordat het andere dingen gaat doen.

Rekenkundige bewerkingen geven altijd een resultaat, de 'OUTPUT'. Als u op het toetsenbord intikt:
36/4 + 5 + 9/3 dan is het resultaat:
RESULT :17

LOGO kent evenals andere talen diverse functies namelijk:

RANDOM N
Dit geeft u een getal tussen 0 en de bovengrens N. Het getal is niet geheel willekeurig, voordat u RANDOM gebruikt is het verstandig om RANDOMIZE in te tikken. Hiermee vermijdt u dat u steeds dezelfde reeks willekeurige getallen krijgt.

ROUND
Rondt een getal met decimale cijfers af naar het dichtstbijzijnde gehele getal. Dit is echt afronden en iets anders dan de functie INT in BASIC.

INTEGER
Geeft u het getal zonder de in het getal voorkomende decimale cijfers. Ook hier een afwijking van het begrip INTEGER zoals u het kent:
INTEGER −4.9 geeft −4 en niet zoals u in BASIC gewend bent −5.

QUOTIENT
Geeft u het integergedeelte van de uitkomst van een deling met twee getallen.
QUOTIENT 10 3 geeft als OUTPUT 3.
QUOTIENT 4 5 geeft als OUTPUT 0.

REMAINDER
Geeft u de rest van een deling van twee gehele getallen.
REMAINDER 2 3 geeft als resultaat 2.
REMAINDER 5 −2 geeft een foutmelding, omdat LOGO dit ziet als een enkele INPUT. U kunt dit oplossen op twee manieren namelijk:
1) REMAINDER 5 (−2)
2) REMAINDER 5 0−2
Als u QUOTIENT en REMAINDER gebruikt met 'reals' dan worden deze eerst via ROUND afgerond tot INTEGERS.

SQRT
Geeft als resultaat de tweedemachtswortel van een positief getal.

SIN
Geeft de sinus van een hoek, gegeven in graden.

COS
Geeft de cosinus van een hoek, gegeven in graden.

U ziet dat u bij SIN en COS niet moeilijk hoeft te doen door hoeken eerst van graden om te gaan rekenen naar radialen.

2.4 Rekenen en programmeren

Er zijn nog een paar primitieven die belangrijk zijn bij getallen en cijfers namelijk:

>
Output is TRUE als de eerste input groter is dan de tweede, anders FALSE.

<
Output is TRUE als de eerste input kleiner is dan de tweede, anders FALSE.

ATAN
Vraagt twee inputs en geeft in graden de arctangens van het quotient.

BITAND
Vergelijkbaar met AND in BASIC. Heeft twee inputs nodig en geeft de booleaanse AND als resultaat. Dat wil zeggen dat corresponderende paren bits met elkaar worden vergeleken.
Alleen als ze beiden 1 zijn is het overeenkomstige bit in het resultaat ook 1. De inputs moeten beiden integers zijn.

BITOR
Twee integer inputs, het resultaat is de booleaanse OR. Hier worden overeenkomstige paren bits vergeleken. Het corresponderende bit in het resultaat wordt 1 als een van de twee vergeleken bits 1 is.

BITXOR
Twee inputs nodig, output booleaanse EXCLUSIVE OR. Deze functie is standaard niet aanwezig in de 64 (in het BASIC statement WAIT is hij wel verwerkt). Als de twee vergeleken bits gelijk zijn dan is het bit in het resultaat gelijk aan 0, indien ze ongelijk zijn dan is het resultaat 1. Inputs moeten integers zijn.

NUMBER?
Output is TRUE als de input een getal is, anders is de output FALSE.

In procedures moet u altijd iets doen met het resultaat van berekeningen, anders gaat LOGO klagen. U kunt het resultaat gebruiken als INPUT zoals in:
FD 360/8 – de cursor gaat 45 stappen vooruit.
U kunt het resultaat toekennen aan een variabele zoals in:
MAKE "A 360/8 – Hier wordt aan de variabele A de waarde 45 toegekend.
U kunt dan de waarde van A eventueel weer laten afdrukken door:
PRINT :A – Dit drukt 45 af.

De term variabele is reeds diverse malen gebruikt zonder duidelijk aan te geven waaraan de naam van een variabele in LOGO moet voldoen.
Namen van variabelen in LOGO mogen alle tekens bevatten behalve de rekenkundige operatoren, haakjes, de vierkante haken en een stel aanhalingstekens. Verder is alles toegestaan, ook de lengte is niet aan beperkingen onderhevig. Zelfs namen van primitiven mogen als variabele naam worden gebruikt. De naam van een variabele wordt altijd voorafgegaan door een stel aanhalingstekens (bijvoorbeeld "WISSELWACHTER), de waarde van een variabele wordt voorafgegaan door een dubbele punt.
Het LOGO primitief 'MAKE' kent een waarde toe aan een variabele. Deze waarde blijft behouden tot er met behulp van 'MAKE' een andere waarde wordt toegekend. De waarde van de variabele wordt in alle procedures waarin de variabele naam voorkomt gebruikt. Men spreekt in dit geval van een 'global' variabele.

2.4 Rekenen en programmeren

LOGO kent ook 'locale' variabelen. Deze komen voor in procedures. Tijdens de uitvoering van de procedure is de variabele met zijn waarde aanwezig, nadat de procedure afgelopen is bestaat de locale variabele niet meer. De INPUT voor een procedure gedraagt zich als een locale variabele.

Er is op deze regeling een zeer belangrijke uitzondering via het commando LOCAL. U maakt met dit commando een variabele die in een procedure via MAKE een waarde krijgt, dus global is, tot een locale variabele.
Een voorbeeld hiervan is de volgende procedure:

```
TO UITKOMST :GETAL
PRINT :GETAL* :GETAL* 5
END
```

U kunt dan bijvoorbeeld de volgende uitkomsten krijgen:
UITKOMST 5 drukt af 125
UITKOMST 20 drukt af 2000

Als de procedure UITKOMST afgelopen is dan bestaat :GETAL niet meer, want:
PRINT :GETAL werkt niet.

Indien u echter GETAL via MAKE "GETAL N een waarde had gegeven was GETAL een globale variabele geweest en had u hem wel kunnen laten afdrukken.

Als u nadenkt over het voorgaande zult u begrijpen dat er een methode moet zijn om de waarde van een variabele uit een procedure te halen om hem elders te kunnen gebruiken. De eerste oplossing heeft u reeds gezien, namelijk de variabele GLOBAL maken, dan geldt hij overal en is dus op te vragen.
De tweede mogelijkheid is via het primitief OUTPUT. Als u de vorige procedure verandert in:

```
TO UITKOMST1 :GETAL
OUTPUT :GETAL* : GETAL* 5
END
```

dan krijgt u als u intikt UITKOMST1 5 op het scherm afgedrukt:

RESULT : 125.

Hoewel LOGO geen functie TAN heeft, is het natuurlijk erg makkelijk om deze te introduceren, want er geldt TANX = (SINX)/(COSX).

8/2.5

Functies tekenen

Uiteraard is de grafische cursor in combinatie met rekenwerk bij uitstek geschikt om (wiskundige) functies te tekenen.
LOGO hanteert twee assen, namelijk de X-as en de Y-as.
Het algemene beeld van een functie is Y=f (X). Als u hier het LOGO primitief SETXY gebruikt, dan kunt u de twee INPUTS die SETXY nodig heeft met de functie die u in beeld wilt brengen berekenen. Drie zaken moeten daarbij in het oog gehouden worden, namelijk:

1) De grafiek moet op het scherm blijven.
2) De juiste plaats van de grafiek op het scherm.
3) De schaalverhouding van X en Y in verband met het duidelijk zichtbaar maken van de functie.

U gaat nu een procedure opbouwen om een sinuscurve, dus Y=SIN X, te tekenen in LOGO. U kunt aan de variabele Y een waarde toekennen met behulp van:

```
MAKE "Y SIN(:X) en daarna
SETXY :X :Y
```

De oorsprong van het denkbeeldige assenstelsel in LOGO ligt in het midden van het scherm, om de sinuscurve aan de linkerkant van het scherm te laten beginnen is het dus noodzakelijk om met de waarde van X gelijk aan −155 te starten. We moeten met tekenen stoppen als de waarde van X gelijk is aan 155. Omdat geldt Y = SIN X nemen we voor de positie op de X-as dus X−155.
Een sinus-waarde varieert tussen 0 en 1, als u daaraan niets doet zal uw sinuscurve heel slecht zichtbaar zijn. Het scherm heeft als verticale begrenzingen +130 en −130. als u de waardes van Y met 100 vermenigvuldigt krijgt u een redelijk duidelijke curve op het beeld. Het hart van de procedure wordt als volgt:

```
TO GRAFIEK.SIN :X
SETXY :X – 155 100 * SIN :X
END
```

Dit berekent een punt van de grafiek. Als u voor het einde van de procedure deze opnieuw aanroept, maar nu met een andere waarde voor X krijgt u weer een punt enzovoorts enzovoorts. Als volgt:

```
TO GRAFIEK.SIN :X
SETXY :X – 155 100 * SIN :X
GRAFIEK SIN :X + 5
END
```

Het enige dat u nu nog moet doen is de maximale waarde van het punt op de X-as begrenzen tot 155, dat wil zeggen:

2.5 Functies tekenen

```
TO GRAFIEK.SIN :X
IF :X – 155 › 155 STOP
SETXY :X – 155 100 * SIN :X
GRAFIEK SIN :X + 5
END
```

Het intikken van GRAFIEK.SIN 0 geeft u een prachtig uitgebalanceerde sinuscurve.
Er is natuurlijk ook een eenvoudige procedure te maken voor het tekenen van de X-as, waarna u in de procedure GRAFIEK.SIN :X de procedure AS kunt aanroepen en het geheel in één keer op het scherm verschijnt.

```
TO AS
DRAW
HT
SETXY 155 0
HOME
SETXY –155 0
END
```

Nu definiëren we nog een procedure:

```
TO SINUS
AS
GRAFIEK.SIN 0
END
```

Na het intikken van SINUS verschijnt uw grafiek.
Op deze wijze zijn er ellipsen, parabolen en allerlei andere functies af te beelden. Essentieel is dat u de juiste schrijfwijze van de functies tot uw beschikking heeft en de bovenvermeldde drie punten in gedachten houdt, namelijk:f waar begint mijn grafiek, hoe maak ik hem goed zichtbaar en waar stop ik met tekenen.

8/2.6

Woorden en lijsten

Met LOGO tekenen en rekenen is, door het geheel op te splitsen in procedures, die ook weer zichzelf kunnen aanroepen, relatief gemakkelijk. De manier waarop LOGO een reeks gegevens (data) behandelt is iets ingewikkelder. Dit gebeurt met lijsten (lists). Allereerst de definities:

WOORD
Een woord in LOGO mag bestaan uit een reeks afdrukbare tekens, uitgezonderd de vierkante haken. De waarde van het woord wordt bij de toekenning tussen [] – die vierkante haken – geplaatst. Het is zelfs mogelijk om ook vierkante haken in een woord op te nemen. Dit woord moet u dan wel tussen apostrofs plaatsen. Bijvoorbeeld:
'KLEINSTE GEDEELTE' is een toegestaan woord.
Kijkt u maar:
MAKE "WOORD [APPEL[BO]OM]
PRINT :WOORD
en
MAKE "WOORD ['APPEL[BO]OM]
PRINT :WOORD

LIJST
Een lijst is een geordende reeks gegevens. Uitgebreider geformuleerd een geordende reeks van cijfers, woorden en lijsten. (U ziet weer het recursieve element in LOGO, lijsten kunnen voorkomen in lijsten).

Voordat u met enige voorbeelden van woorden en lijsten gaat werken is het zinvol om alle primitieven die LOGO kent om woorden en lijsten te behandelen te leren kennen.

=
Vergelijking van twee cijfers, twee woorden of twee lijsten.
Als de beide zaken die vergeleken worden identiek aan elkaar zijn, is de uitkomst TRUE, anders is het resultaat FALSE.

BF BUTFIRST
Geeft – indien de input een woord is – het woord minus het eerste karakter als output. Als de input een lijst is dan wordt de output een nieuwe lijst gelijk aan de oude minus het eerste element. Omdat cijfers in wezen ook woorden zijn werkt BUTFIRST op dezelfde manier, het eerste getal wordt verwijderd.

BL BUTLAST
Soortgelijk primitief als BUTFIRST, alleen wordt hier het laatste element of teken verwijderd.

COUNT
Als de input een woord is, dan is het resultaat het aantal tekens waaruit het woord is samengesteld. Als de input een lijst is dan is het resultaat het aantal elementen in de lijst. Er wordt slechts op

2.6 Woorden en lijsten

één niveau geteld, als de lijst nog andere lijsten bevat dan telt elk van die lijsten als een element.

EMPTY?
Test een input, als de input een leeg woord "" of een lege lijst '[]' is, dan is het resultaat TRUE, in elk ander geval is het resultaat FALSE.

FIRST
Als de input een woord is, dan is de output het eerste teken van het woord, als de input een lijst is, dan is de output het eerste element van de lijst. U krijgt een foutmelding als u FIRST toepast met een leeg woord of een lege lijst.

FPUT
Dit primitief heeft twee inputs nodig, waarvan de tweede een lijst moet zijn. De output is een lijst bestaande uit de eerste input gevolgd door de tweede. Voorbeeld:
FPUT [A B] [C D] geeft [[A B] C D]

ITEM
De inputs hiervoor zijn een getal en een woord of lijst.
Stel, het getal is N, dan geeft ITEM als output het N-teken als de tweede input een woord is. Als de tweede input een lijst is, dan geeft ITEM het N – de element als resultaat.

LAST
Geeft het laatste teken of element al naar gelang de input een woord of een lijst is. Geeft een foutmelding indien toegepast met een leeg woord of een lege lijst.

LIST
Vraagt minimaal twee inputs. Het commando maakt van de inputs een nieuwe lijst.

LIST?
Geeft als output TRUE als de input een lijst is, zo niet dan is de output FALSE.

LPUT
Vraagt twee inputs. In tegenstelling met FPUT komt nu de tweede input vooraan te staan in het resultaat.

MEMBER?
Vraagt twee inputs. Dit kunnen zowel woorden als lijsten zijn. Als de eerste input aanwezig is in de tweede dan is het resultaat TRUE, zo niet dan is het resultaat FALSE.

SENTENCE
Vraagt minimaal twee inputs en koppelt deze aan elkaar tot één nieuwe lijst. Een woord wordt in dit verband beschouwd als een lijst met slechts een element.

WORD
Vraagt minimaal twee inputs en voegt deze samen tot één woord.

WORD?
Als de input een woord is dan is het resultaat TRUE, anders FALSE. Een cijfer als input geeft altijd TRUE als resultaat.

Als u de volgende bewerkingen met lijsten en woorden eens goed bekijkt dan zal dat het een en ander verduidelijken:

[1 2 AAP NOOT MIES] is een lijst van vijf elementen. Een lijst kan ook een element van een lijst zijn:

2.6 Woorden en lijsten

[[TEUN VUUR] AAP NOOT] is een lijst van drie elementen. Het resultaat van:
FIRST [1 2 AAP NOOT MIES] is 1, terwijl het in:
FIRST [[TEUN VUUR] AAP NOOT] geeft als resultaat [AAP NOOT].

FPUT [BOK] [2 AAP NOOT MIES] geeft als resultaat een lijst bestaande uit: [[BOK] 2 AAP NOOT MIES].

FPUT BOK [AAP NOOT MIES] geeft een foutmelding.

FPUT 2 [AAP NOOT MIES] geeft als resultaat [2 AAP NOOT MIES]

SENTENCE [LOGO IS] [EEN COMPUTERTAAL] geeft als output de lijst [LOGO IS EEN COMPUTERTAAL].

Het grote verschil met strings in BASIC is dat lijsten als waarde kunnen worden toegekend aan variabelen, dat lijsten als input kunnen dienen voor procedures en dat ze de output van procedures kunnen vormen. Bijvoorbeeld:

MAKE "X [TARA BOEM]
MAKE "Y [HOLA DIEE]

U kunt ook primitieven combineren bij het werken met lijsten:

FIRST FIRST [[TEUN VUUR] AAP NOOT] geeft TEUN als resultaat.

U kunt natuurlijk procedures schrijven die lijsten bewerken.
Bijvoorbeeld:

```
TO DUBBEL :L
OUTPUT SENTENCE :L :L
END
```

```
PRINT DUBBEL [TARA BOEM]
TARA BOEM TARA BOEM
PRINT DUBBEL DUBBEL [TARA BOEM]
TARA BOEM TARA BOEM TARA BOEM TARA BOEM
```

8/2.7

Spelen met teksten

Hoewel LOGO het meest bekend is geworden door de schildpad, de 'turtle graphics', is het een taal die erg geschikt is om met woorden en zinnen om te gaan. Doordat het eigenlijk familie is van de taal LISP, die gebruikt wordt in onderzoek naar kunstmatige intelligentie, kunt u wel begrijpen dat er met LOGO erg leuke dingen zijn te doen. In LOGO is de mogelijkheid aanwezig om gemakkelijk interactieve programma's te schrijven aangezien de mogelijkheden voor vraag en antwoord eenvoudig te hanteren zijn. Behalve de reeks primitieven waarmee u reeds heeft kennis gemaakt zijn er nog die INPUT en OUTPUT behandelen. Deze plus enkele die nog niet aan de orde zijn geweest volgen hier:

CTYO
Vraagt als input een getal tussen 0 en 255. Het teken met een ASCII code gelijk aan de input wordt afgedrukt (de afkorting betekent Character TYpe Out).

ASCII
Vraagt een teken als input en drukt de ASCII waarde daarvan af.

CHAR
Vraagt een integer als input en geeft het teken met het ASCII nummer gelijk aan de input als output.

CLEARTEXT
Maakt het tekstscherm schoon en brengt de cursor naar de linker bovenhoek van dit tekstscherm.

CLEARINPUT
Maakt de toetsenbordbuffer leeg, dat wil zeggen vooruit ingetikte tekens worden verwijderd.

CURSOR
Vraagt twee inputs, de eerste voor de kolom (0-39), de tweede voor de regel (0-24). De cursor gaat naar deze positie.

CURSORPOS
Geeft een lijst met de coördinaten van de huidige cursorpositie.

FPRINT
Werkt als PRINT, maar drukt ook de quotes en de buitense haken van een lijst af. Maakt de uitvoer iets leesbaarder.

JOYSTICK
Vraagt als input een 0 of een 1 die de joystickpoort aangeeft en geeft als uitvoer de toestand van de joystick.
0-7 voor de diverse richtingen, 1 indien geen richting wordt aangegeven.

JOYBUTTON
Vraagt 0 of 1 als input. Geeft TRUE als de vuurknop is ingedrukt, FALSE indien

2.7 Spelen met teksten

dit niet het geval is.

NOPRINTER
Beëindigt PRINTER mode.

PADDLE
Vraagt om invoer van 0 tot en met 3 die de paddle aangeeft en geeft als uitvoer een getal van 0 tot 255, de waarde van die paddle.

PADDLEBUTTON
Als JOYBUTTON, input 0 tot en met 3, output TRUE of FALSE.

PRINT
reeds bekend.

PRINT1
Als PRINT, maar zonder linefeed. De cursor blijft op dezelfde regel.

PRINTER
Verlegt de uitvoer van het scherm naar de printer.

RC READCHARACTER
De output van dit primitief is het laatste teken uit de toetsenbordbuffer. Als deze buffer leeg is wacht de computer op invoer.

RC?
TRUE als er een teken in de toetsenbordbuffer aanwezig is, anders FALSE als output.

RQ REQUEST
Wacht tot er een regel is ingevoerd die met RETURN beëindigd is.

SETDISK
Om met meer diskdrives te kunnen werken (9, 10, 11). Standaardwaarde is 8.

DOS
Commando's voor de diskdrive kunnen na DOS ingevoerd worden.

GO
Vraagt een woord als input. Het programma springt naar de regel met als label het ingevoerde woord. In LOGO krijgt een regel een label door de regel hiermee te laten beginnen, gevolgd door een dubbele punt. GO mag alleen binnen een procedure worden gebruikt.

RUN
Voert een lijst (niet meer dan 255 tekens lang) uit alsof hij in een commandoregel was ingevoerd.

REPEAT
Voert de ingegeven lijst het door de andere input bepaalde aantal malen uit. Reeds bekend.

IFFALSE
Voert de rest van de regel alleen uit als het resultaat van de voorafgaande TEST FALSE was.

IFTRUE
Rest van de regel wordt uitgevoerd indien resultaat van TEST TRUE was.

TEST
Test een conditie om het resultaat te gebruiken in IFTRUE of IFFALSE. TEST vraagt één input, TRUE of FALSE.

THING
Geeft de waarde van de input. THING "AAA is gelijk aan :AAA.

THING?
Output TRUE als THING een waarde heeft, anders FALSE.

2.7 Spelen met teksten

DEFINE
Hiermee kunt u buiten de EDITOR om procedures definiëren. Wordt praktisch nooit gebruikt.

De eigenschap om lijsten te kunnen manipuleren, ja zelfs woorden en lijsten samen te voegen tot nieuwe lijsten maakt deze taal ook erg geschikt om tijdens de uitvoering van een programma de aanwezige woordenschat uit te breiden. Als u met zo'n programma bezig bent is het verbazingwekkend hoe snel LOGO schijnt te 'leren'. Het is natuurlijk maar schijn, alles wat een computer kan en kent is immers door de programmeur mogelijk gemaakt. U zult in het gedeelte dat handelt over lijsten en woorden met enkele van de prettige karaktereigenschappen van LOGO kennismaken. Straks zult u ook zien als we een blik werpen op de utility diskette dat een redelijk stuk tekstverwerking voor LOGO in het geheel geen probleem vormt.

Een primitieve tekstgenerator maken lukt u met de aanwezige kennis best. Uit een reeks namen moet een willekeurige gekozen worden, deze moet worden afgedrukt. Hierna moet een willekeurige keuze uit een reeks werkwoordsvormen worden gekozen en afgedrukt, gevolgd door hetzelfde voor een beschrijving.
Eerst maakt u uw lijsten als volgt:

```
MAKE "NAMEN [JAN PIET KLAAS TRUUS NEL]

MAKE "WERKWOORDEN [IS LOOPT ZWEMT WANDELT DANST]

MAKE "OMSCHRIJVINGEN [GEK LIEF MOOI ENG LANG]
```

De procedure om een willekeurig element te kiezen:

```
TO KIES INPUT
MAKE "N COUNT :INPUT
OUPUT ITEM (1 + RANDOM :N) :INPUT
END
```

De procedure voor de tekst:

```
TO TEKST
PRINT1 NAAM
PRINT1 WERKWOORD
PRINT OMSCHRIJVING
TEKST
END
```

De procedures voor de naam, het werkwoord en de omschrijving:

```
TO NAAM
OUTPUT KIES :NAMEN
END

TO WERKWOORD
OUTPUT KIES :WERKWOORDEN
END

TO OMSCHRIJVING
OUTPUT KIES :OMSCHRIJVINGEN
END
```

Na het invoeren van TEKST krijgt u continu zinnetjes van verschillende samenstelling afgedrukt. Stoppen doet u met ‹CTRL› G, maar u kunt er ook een teller in bouwen zodat de procedure na een door u te bepalen aantal malen te zijn uitgevoerd zal stoppen.
Een nog niet eerder in een procedure gebruikt commando is PRINT1, dat afdrukt zonder naar een nieuwe regel te

2.7 Spelen met teksten

springen. Voor het afdrukken van scheidings-spaties moet u ook al een oplossing vinden. Het gaat goed als u de woorden in de invoerlijst plus een spatie tussen apostrofs plaats. Als volgt:

```
MAKE "NAMEN ['JAN' 'PIET' enzovoorts]
```

Een eenvoudige quiz in LOGO is ook niet zo moeilijk.
De vragen met de daarbij behorende antwoorden:

```
TO QUIZ
VA [WAT IS DE HOOFDSTAD VAN FRANKRIJK] [PARIJS]
VA [WAT IS DE HOOFDSTAD VAN ENGELAND] [LONDEN]
VA [WAT IS DE HOOFDSTAD VAN GRIEKENLAND] [ATHENE]
END
```

Het vraag en antwoordspel:

```
TO VA :VRAAG :ANTWOORD
TYPE :VRAAG
IF :ANTWOORD = VRAAG PR [GOEDZO] STOP
PRINT SENTENCE [NEE SUFFERD HET ANTWOORD IS ] :ANTWOORD
END
```

Zo'n programma vraagt in BASIC veel meer ruimte.

8/2.8

Foutmeldingen in LOGO

Er zijn in LOGO diverse foutmeldingen die u helpen bij het opsporen van fouten in de door u geschreven procedures. Er bestaan twee groepen foutmeldingen, de eerste groep heeft altijd dezelfde tekst, de tweede begint telkens met een ander woord, meestal de naam van een procedure.
De foutmeldingen die voor zichzelf spreken zullen u geen problemen bezorgen, een paar moeilijkere foutmeldingen worden hier uitgelegd:

END SHOULD BE USED ONLY IN THE EDITOR
U gebruikt END op een plaats waar dit niet is toegestaan.

ELSE IS OUT OF PLACE
LOGO kent de constructie IF...THEN... ELSE, als u ELSE gebruikt op een andere plaats dan toegestaan krijgt u de voorgaande foutmelding.

LINE GIVEN TO DEFINE TOO LONG
LINE GIVEN TO REPEAT TOO LONG en
LINE GIVEN TO RUN TOO LONG
De betreffende regel is langer dan de toegestane 256 tekens.

MISSING INPUTS INSIDE ()'S
De procedure of het primitief binnen de haakjes was gebruikt met te weinig inputs.

PROCEDURE NESTING TOO DEEP
De grens van procedures binnen procedures is overschreden (de grens is meer dan 200)

RESULT: (gegeven)
U heeft niet aangegeven wat u wilt doen met het resultaat van een berekening (overigens geeft deze foutmelding u mogelijkheden om even snel iets uit te rekenen).

THE : IS OUT OF PLACE AT (iets)
U heeft ten onrechte ':' gebruikt.

THEN IS OUT OF PLACE
Ten onrechte THEN toegepaste.

THERE IS NO LABEL (gebruikte naam)
THERE IS NO NAME (gebruikte naam)
THERE IS NO PROCEDURE CALLED (gebruikte naam)
LOGO kent respectievelijk het gebruikte label om naar toe te springen, de gebruikte variabelenaam of de gebruikte procedurenaam niet. De laatste melding krijgt u dikwijls als u een tikfout maakt in een van de primitieven.

2.8 Foutmeldingen in LOGO

THERE'S NOTHING TO SAVE
Als u een SAVE commando geeft bij een leeg werkblad voorkomt deze melding dat u uw diskette vult met niets, daarna zou u een lege diskette hebben!

TOO MANY PROCEDURE INPUTS
De grens (circa 100) overschreden.

TOO MUCH INSIDE PARENTHESES
LOGO begrijpt een uitdrukking met haakjes niet.

TURTLE OUT OF BOUNDS
In de NOWRAP mode als de grafische cursor de grens van het scherm zou overschrijden als hij bewoog.

YOU DON'T SAY WHAT TO DO WITH (gegeven)
Er ontbreekt een instructie wat er moet gebeuren. Vergelijkbaar met RESULT: in de direct mode.

8/2.9

LOGO programma structuur

U heeft al gezien dat elk LOGO programma is opgebouwd uit een groep procedures. Al deze procedures wisselen onderling informatie uit via OUTPUT en INPUT. Zelfs in zeer complexe programma's komen zelden procedures voor die langer zijn dan een paar regels. Door de gehele structuur van het LOGO systeem is het mogelijk om procedures afzonderlijk op te zetten en individueel te testen. Als alle onderdelen van een systeem foutloos zijn is de kans groot dat het geheel ook zonder fouten functioneert.

Om dit alles nogmaals te illustreren gaat u nu een spel ontwerpen met de volgende spelregels:
De computer bepaalt een willekeurig 'geheim' punt op het scherm. Door de turtle met behulp van LEFT en FORWARD te bewegen moet de speler zo dicht mogelijk bij het geheime punt komen. Voor elke beurt van de speler wordt de afstand tot het geheime punt door de computer op het scherm afgedrukt. De kunst is in zo weinig mogelijk beurten het doel te bereiken.

Het hart van het spel is een procedure SPEL. SPEL heeft als input Z, het aantal beurten van de speler. Eerst wordt gecontroleerd of de speler gewonnen heeft, zo ja dan wordt het aantal beurten afgedrukt en is het spel afgelopen. Zo nee, dan mag de speler een zet doen en gaat hij naar de volgende beurt. Z is dan met 1 opgehoogd.

```
TO SPEL :Z
TEST GEWONNEN?
IFTRUE (PRINT [JE WON IN] :Z
[BEURTEN])
IFTRUE STOP
MAKEZET
SPEL :Z + 1
END
```

Deze SPEL procedure is heel eenvoudig, want de moeilijke dingen als controleren of er gewonnen is en een zet doen worden naar andere procedures doorgeschoven.

```
TO MAKEZET
PRINT [DRAAI LINKS HOEVEEL?]
LEFT READNUMBER
PRINT [GA VOORUIT HOEVEEL?]
FORWARD READNUMBER
END
```

Om te controleren of er gewonnen is, moet er gekeken worden hoever de grafische cursor van het doel verwijderd is, bijvoorbeeld minder dan 15 stappen. De primitieven XCOR en YCOR geven de coördinaten van de schildpad. De coördinaten van het doel slaan we op in de variabelen XPT en YPT. Als er een procedure AFSTAND is, kunt u de procedure GEWONNEN? schrijven.

2.9 LOGO programmastructuur

```
TO GEWONNEN?
MAKE "D AFSTAND XCOR YCOR
:XPT :YPT
(PRINT [AFSTAND TOT PUNT IS]
:D)
IF :D < 15 OUTPUT "TRUE
OUTPUT "FALSE
END
```

GEWONNEN? geeft als output TRUE of FALSE, dit heeft SPEL nodig om te bepalen of het spel afgelopen is of niet. Het berekenen van de afstand tussen twee punten gaat met behulp van de stelling van PYTHAGORAS.

```
TO AFSTAND :A :B :X :Y
MAKE "AX :A - :X
MAKE "AY :B - :Y
OUTPUT SQRT (:AX* :AX +
:AY* :AY)
END
```

Er is ook nog een procedure nodig om het spel te starten, BEGIN

```
TO BEGIN
CLEARSCREEN
MAKE "XPT RANDOMCOORD
MAKE "YPT RANDOMCOORD
SPEL 0
END
```

Het bepalen van de coördinaten gebeurt in de procedure RANDOMCOORD

```
TO RANDOMCOORD
OUTPUT ((RANDOM 150) - 75)
END
```

Dit geeft coördinaten tussen +75 en -75 mooi centraal op het scherm.
Nu de procedure READNUMBER

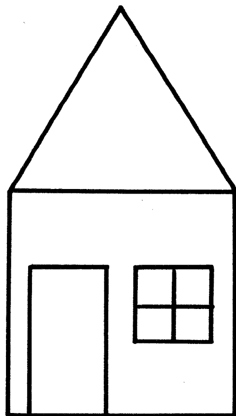
```
TO READNUMBER
OUTPUT FIRST REQUEST
END
```

Het spel is nu klaar. U zou nog kunnen overwegen in de laatste procedure een zodanige wijziging aan te brengen dat bij het intikken van iets anders dan cijfers er een melding komt dat er cijfers ingevoerd moeten worden.

8/2.10

Recursie: perspectief-tekenen

Stelt u zich voor dat u een laan wilt tekenen waaraan huizen staan die door de perspectivische werking steeds kleiner worden. Het beste kunt u uitgaan van een nogal gestyleerd huis, een voorbeeld vindt u in figuur 1.



Als u uw huis goed bekijkt dan ziet u dat het uit elementen is opgebouwd. Als het huis door de perspectivische werking twee maal zo klein wordt, dan worden alle elementen twee maal zo klein. Het is dus duidelijk dat een input in alle procedures die u nodig heeft om te kunnen tekenen de MAAT is. Zet het huis maar in een erg grove procedure op:

```
TO HUIS :MAAT
VOORKANT :MAAT
DAK :MAAT
END
```

Nu kunt u gaan kijken waaruit de voorkant van het huis eigenlijk bestaat. Ook dit gaat in een grove procedure.

```
TO VOORKANT :MAAT
MUREN :MAAT
DEUR :MAAT
RAAM :MAAT
END
```

U moet nu de maatverhoudingen van de diverse elementen ten opzichte van het geheel gaan definiëren. Dit resulteert in een reeks procedures. Duidelijk zal zijn dat hier nogal simpele verhoudingen zijn gebruikt, als u een echte bouwtekening tot uw beschikking heeft wordt het allemaal veel gecompliceerder, maar ook interessanter.

```
TO MUREN :MAAT
VIERKANT :MAAT * 3
END

TO DAK :MAAT
DRIEHOEK :MAAT * 3
END

TO RAAM :MAAT
REPEAT 4[VIERKANT :MAAT/2 RT
90]
END

TO DEUR :MAAT
RECHTHOEK :MAAT * 2 :MAAT
END
```

2.10 Recursie perspectief-tekenen

Er rollen dan nog een paar procedures uit:

```
TO DRIEHOEK :LENGTE
REPEAT 3[FD :LENGTE RT 120]
END
```

```
TO VIERKANT :LENGTE
REPEAT 4[FD :LENGTE RT 90]
END
```

```
TO RECHTHOEK :LENGTE
:BREEDTE
REPEAT 2[FD :LENGTE RT 90 FD
:BREEDTE RT 90]
END
```

Alle elementen van het huis zijn nu aanwezig, er rest nog slechts het 'monteren' en het bepalen waar het eerste huis komt. Met behulp van een procedure SETUP zorgt u voor een groot tekenscherm en brengt u de cursor naar de uitgangspositie:

```
TO SETUP
FULLSCREEN
PU LT 90 FD 135 RT 90 BK 115 PD
END
```

De procedure voor het gehele huis zou kunnen worden:

```
TO HUIS :MAAT
VOORKANT :MAAT
FD :MAAT * 3 RT 30
DAK :MAAT
LT 30 BK :MAAT * 3
END
```

Alleen de voorgevel:

```
TO VOORKANT :MAAT
MUREN :MAAT
RT 90 FD :MAAT / 3 LT 90
DEUR :MAAT
PU RT 90 FD :MAAT * 2 LT 90 FD
 :MAAT * 1.5 PD
RAAM :MAAT
PU BK :MAAT * 1.5 LT 90 FD
 :MAAT * 2 + :MAAT /3 RT 90 PD
END
```

Nu de zaak nog combineren en voorzien van een factor die bepaalt hoe klein het tweede huis ten opzichte van het eerste is en de hele tekening kan worden gemaakt.

```
TO H :MAAT :FACTOR
IF :MAAT < 2 STOP
HUIS :MAAT
PU RT 90 FD :MAAT * 3.4 LT 90
FD :MAAT * 2 PD
H :maat * :FACTOR :FACTOR
END
```

En de laatste procedure:

```
TO HUIZEN :FACTOR
HT
SETUP
H 30 :FACTOR
END
```

De getallen 3.4 en 2 zijn proefondervindelijk bepaald door diverse malen het tekenprogramma te RUNnen.

Printen met LOGO:
Als u aan het tekenen bent of op andere wijze aan het werk met LOGO kunt u, zoals bekend, de printer inschakelen met behulp van het woord PRINTER. De output die anders naar het scherm ging zal dan naar de printer gaan. Als u klaar bent met printen moet u dit aan LOGO mededelen door de instructie NOPRINTER. Het grappige van LOGO is dat u

2.10 Recursie: perspectief-tekenen

van het printen ook een procedure kunt maken, dit gaat als volgt:

```
TO AFDRUK :PROCEDURE
PRINTER
RUN LIST"PRINTOUT:PROCEDURE
NOPRINTER
END
```

Deze procedure kunt u dan vanuit andere procedures aanroepen.

8/2.11

De utility diskette

Naast de diskette met LOGO behoort bij het pakket – behalve een vrij uitgebreid handboek – ook een diskette met utility- en demonstratie-programma's. We geven een korte samenvatting van die files die betrekking hebben op het in deze aanvulling behandelde gedeelte van LOGO.

De utilities:

ARCS
Een reeks procedures om krommen met verschillende straal te tekenen.

BASE
Procedures om getallen van het ene talstelsel naar het andere te converteren.

FOR
Een FOR...NEXT lus.

INSTANT
Een systeem waarin alle LOGO commando's teruggebracht worden tot het aanslaan van een toets.

JOY
Een tekenprogramma om met de joystick te tekenen.

LOG
Het werken met logaritmes en exponenten.

PLOTTER
Een reeks procedures om met de 1620 printer/plotter te kunnen werken. Normaal is LOGO ingesteld op de 801 printer, met de juiste interfaces zijn ook andere printers te gebruiken.

PRINTPICT
Dit is echt een hulpprogramma, het is namelijk geschreven in BASIC en is bestemd om op diskette bewaarde tekeningen op de printer af te drukken. Dit programma kan niet vanuit LOGO gerund worden.

STAMPER
Procedures om tekst op het grafische scherm af te drukken.

TEACH
Systeem om LOGO procedures te schrijven buiten de editor om.

TEXTEDIT
Systeem om de LOGO editor te gebruiken als tekstverwerker.

WHILE
Via deze file krijgt u de beschikking over de commando's WHILE en UNTIL.

Demonstratieprogramma's:

ADVENTURE
Een adventure geschreven in LOGO. Gaat dieper in op het manipuleren van lijsten.

2.11 De utility diskette

ANIMAL
Voorbeeld van een zelf lerend programma. Het heeft een basiskennis die al doende uitgebreid wordt door het opnemen van uw vragen en antwoorden in de vocabulaire van ANIMAL. Via ANIMAL.INSPECTOR kunt u de aanwezige kennis in ANIMAL onderzoeken.

DYNATRACK
Wrijvingloze racebaan. Demonstratie van natuurkundewetten.

GRAMMAR
Gaat dieper in op het genereren van teksten.

INSPI.PIC1 en INSPI.PIC2
Afbeeldingen op de utility diskette om READPCT en SAVEPICT te demonstreren.

PIG
Manipuleren van lijsten en woorden.

SNOW
Sneeuwvlok tekenen met behulp van het veranderen van een driehoek en via recursie.

TET
Recursie gebruiken bij het maken van een ingewikkelde afbeelding.

8/2.12

LOGO wetenswaardigheden

Als u zich verder wilt verdiepen in de taal LOGO zijn er diverse mogelijkheden om u aan te sluiten bij groepen die zich actief met LOGO bezig houden.
De STICHTING LOGO kunt u bereiken via Postbus 1408, 6501 BK in Nijmegen, telefoon 080-238130. Ook binnen de Hobby Computer Club, Postbus 2249, 3500 GE Utrecht is een LOGO gebruikersgroep actief. De HCC is bereikbaar onder telefoonnummer 030-946645.
De officiële Commodoreversie van LOGO is uiteraard engelstalig, maar voor diegenen die het Engels niet goed machtig zijn, bestaan of komen ook diverse mogelijkheden in het Nederlands. Uitgeverij Malmberg brengt het nederlandstallige programma 'SCHILDPAD' uit. Dit is wel leuk, doch heeft helaas erg weinig te maken met de originele LOGO. Een tijdlang is er een, gedeeltelijk in het nederlands vertaalde LOGO versie geweest maar ook deze uitvoering week behoorlijk af van de oorspronkelijke. Binnenkort komt echter de LOGO stichting uit met een volledig nederlandstalige LOGO. Deze heeft de naam pLOGO (voor portable LOGO) gekregen, maar wordt ook LCN LOGO genoemd. In principe zal deze LOGO op alle systemen kunnen draaien.
Voor de liefhebbers onder u zijn er ook diverse boeken over LOGO op de markt. Wij noemen slechts:
Learning Commodore 64 LOGO together, K.P. Goldberg, uitgave Microsoft Press.
Starting with LOGO, C.H. Leigh, uitgave Sigma.
Das trainingsbuch zu LOGO, G. Sauer, uitgave DATA BECKER.
Das Commodore 64 LOGO Arbeitsbuch, H.J. Winter, uitgave Markt und Technik Verlag.
Spiel und Aktion mit Commodore LOGO, C. Wittwehr, uitgave Vogel Verlag.
Commodore 64 LOGO primer, G. Bitter & N. Wattson, uitgave Prentice Hall.
Computers en kinderen, Seymour Papert; uitgegeven door Bert Bakker.
LOGO, H. Pinxteren en J. Ringelberg; uitgegeven door het Spectrum in de Aula reeks.
LOGO PROGRAMMING, Peter Ross; uitgegeven door Addison-Wesley.
LOGISCH LOGO, A. Sikma; uitgave Academic Service.
Learning with LOGO, Daniel Watt, uitgever Antrim.
LOGO and more for the Commodore 64, Lambert, ISBN 088056-3486
Voorts wordt door het LOGO centrum het periodiek SCHILDPAD NIEUWS uitgegeven.

8/2.13

Commando's en primitieven

.ASPECT
Dit primitief wordt gebruikt om de verticale schaal waarin LOGO tekent te beïnvloeden. De standaard aanwezige waarde is 0.768 en hiermee zijn op een normale monitor of televisie de cirkels rond en de vierkanten ook inderdaad vierkant. Heeft u een afwijkend toestel, dan kunt u via .ASPECT N de verticale schaal wijzigen om de vervormingen te corrigeren. Er kunnen bij gebruik van dit primitief problemen optreden als u N een te groot verschil met de standaardwaarde geeft.

.CALL
Dit primitief roept een machinetaalroutine aan, die in het computergeheugen aanwezig is. Er zijn twee inputs nodig, de eerste geeft het startadres van de routine aan, de tweede input is een waarde die in een geheugenlokatie opgeslagen wordt en onderzocht kan worden door de machinetaalroutine. De tweede input is wel noodzakelijk, maar hoeft niet beslist te worden gebruikt.

.CONTENTS
Een primitief dat bruikbaar is voor twee doeleinden. Het geeft u een lijst van alle voor uw LOGO bekende woorden, dat wil zeggen namen van variabelen, van procedures en van woorden in procedures. Behalve om uw geheugen op te frissen, is het ook nuttig als u krap komt te zitten in uw geheugenruimte. Bij het gebruik van .CONTENTS wordt de 'garbage collection' namelijk geblokkeerd. Als u dan de lijst goed bekijkt blijken er misschien veel tikfouten als bekende woorden achtergebleven te zijn. Deze kosten echter allemaal geheugenruimte. U kunt via .GCOLL opruiming houden in het RAM geheugen door een 'garbage collection' te forceren. Ook kunt u in uw werkblad procedures aantreffen die u kunt missen, met ERASE (lijst van procedures) worden deze verwijderd.

.DEPOSIT
Dit primitief vraagt twee inputs. De eerste een geheugenadres en de tweede een waarde. De waarde wordt opgeslagen in het opgegeven geheugenadres (vergelijk POKE in BASIC).

.EXAMINE
Hier dient u een geheugenplaats op te geven en krijgt u te zien welke waarde daarin is opgeslagen (vergelijk PEEK in BASIC).

.GCOLL
Dit primitief forceert een zogenaamde 'garbage collection', dat wil zeggen het geheugen wordt ontdaan van niet noodzakelijke aanwezige gegevens. Indien de computer niet meer schijnt te reageren op uw instructies kan het zijn dat

2.13 Commando's en primitieven

een 'garbage collection' door het systeem wordt uitgevoerd. Via .GCOLL kunt u het moment hiervoor zelf bepalen, zodat u er niet door verrast kunt worden.

.NODES
Verzorgt een opgave van de nog vrije geheugenruimte. Wilt u hiermee exact zijn, dan is het nodig eerst de .GCOLL instructie te geven.

.OPTION
Dit primitief beïnvloedt de werking van diverse primitieven. In hoofdstuk 8/2.14 komen wij uitgebreid op .OPTION terug.

.SPRINT
De input is een getal van 0 t/m 7. Op het scherm verschijnt in rondjes en punten de vorm van de door het cijfer aangegeven sprite. Wordt hoofdzakelijk gebruikt als u met de sprint editor van de utility diskette werkt.

PAUSE
Onderbreekt de uitvoering van een programma en maakt het opvragen van de waarde van variabelen mogelijk. [CTRL] Z heeft dezelfde werking als PAUSE. Met CONTINUE kunt u het onderbroken programma weer voortzetten.

TOPLEVEL
Dit primitief beëindigt de uitvoering van het gehele LOGO gebeuren en geeft de controle terug aan de gebruiker. Het is in zijn werking duidelijk verschillend van STOP, dat alleen de in uitvoering zijnde procedure onderbreekt. TOPLEVEL wordt zelden gebruikt bij het programmeren in LOGO.

STOP
Zie hiervoor de toelichting bij TOPLEVEL. Anders gezegd: STOP beëindigt de in uitvoering zijnde procedure door terug te schakelen naar de controle op een niveau hoger.

ALLOF
Primitief dat minimaal twee inputs vraagt. Het mogen er ook meer zijn, maar dan moet er voor ALLOF een openingshaakje staan en na de inputs moet een spatie volgen voorafgaand aan het sluithaakje. Alleen indien alle inputs WAAR zijn is de output van ALLOF 'TRUE', alle andere gevallen geven 'FALSE' als resultaat.

ANYOF
Het 'spiegelbeeld' van ALLOF. Indien minstens een van de inputs WAAR is: output 'TRUE', anders is het resultaat 'FALSE'.

ELSE
In de eerste aflevering maakte u kennis met de constructie 'IF <uitdrukking> THEN', zoals u reeds kende uit BASIC. LOGO kent ook de constructie 'IF <uitdrukking> THEN <instructie> ELSE <instructie>. Deze constructie maakt het mogelijk om gestructureerder te programmeren dan in BASIC. Hiermee kan, al naar gelang van het resultaat van de uitdrukking, het ene, dan wel het andere blok van het programma uitgevoerd worden.

NOT
Vraagt slechts 1 input. Werking: input 'TRUE', gevolg: output 'FALSE' en omgekeerd: input 'FALSE', dan output 'TRUE'.

2.13 Commando's en primitieven

BLOAD

De input voor BLOAD is een filenaam. Dit bestand wordt ingeladen vanaf de geheugenplaats die in het bestand van die naam op de diskette is opgeslagen. Aan de bestandsnaam bij BSAVE wordt door LOGO niet het achtervoegsel .LOGO toegevoegd, u moet de bestandsnamen bij BLOAD dus exact weten. BLOAD wordt gebruikt voor het laden van machinetaalroutines en spritevormen.

BSAVE

Zorgt voor het vastleggen van de geheugeninhoud van een bepaald geheugengebied op de diskette. Nodig zijn: de de bestandsnaam, het beginadres en het eindadres van het te bewaren geheugengebied. Als het eindadres kleiner is dan het beginadres volgt uiteraard een foutmelding.

TEXT

De procedurenaam is de input, de output is de tekst van de procedure in de vorm van een list. De naam van de procedure moet voorafgegaan worden door ". Als de procedure niet gedefinieerd is, volgt als uitvoer een lege lijst [ ]. Als de input de naam van een primitief is in plaats van de naam van een procedure volgt als uitvoer de naam van dat primitief.

8/2.14

OPTION

.OPTION is een primitief dat werkt als een soort vermenigvuldigingsfactor. Door de diverse mogelijke opties kan een primitief op diverse manieren functioneren. OPTION verhindert u niet om op allerlei manieren met kleuren enzovoorts te werken maar beïnvloedt de default waarden.

De volgende mogelijkheden zijn aanwezig:
.OPTION "DRAW 0 :N. Maakt de standaard achtergrondkleur van het grafische scherm tot kleur N. Bij overgangen tussen SINGLECOLOR en DOUBLECOLOR en tussen NODRAW en DRAW krijgt het scherm de defaultkleur.

.OPTION "DRAW 1 :N. Beïnvloedt de standaard PENCOLOR gebruikt bij DRAW. Bij dezelfde overgangen als boven gaat de PENCOLOR terug naar de default waarde.

.OPTION ".DEPOSIT 0 :N en
.OPTION ".EXAMINE 0 :N gebruikt u bij het machinetaal goochelen. U kunt dan bijvoorbeeld het memory mapped I/O gebeuren uitschakelen en de volledige 64 Kb tot RAM geheugen maken door voor N de waarde 4 te nemen. Bij een waarde van 2 voor N is de Kernel en de 4 Kb karakterset ROM aanwezig inplaats van het I/O gebeuren. Standaardwaarde voor N is gelijk aan 6, dat wil zeggen: Kernel en I/O aanwezig.

.OPTION "EDIT 0 :N en .OPTION "EDIT 1 :N werken precies hetzelfde met het tekstscherm in de EDIT mode als .OPTION "DRAW doet met het grafische scherm in de grafische mode. Het eerste stelt de standaard achtergrondkleur in en het tweede de standaard tekstkleur.

.OPTION "JOYSTICK 0 :N. Normaal is N gelijk aan 0 en betekent JOYSTICK 1 de joystick in poort 1, JOYSTICK 2 uiteraard de joystick in poort 2. Als N gelijk is aan 1 geeft het commando JOYSTICK als output een getal dat gelijk is aan de som van de waarden van de diverse switches van de JOYSTICK. Bijzonderheden hierover vindt u in de COMMODORE Programmer's Reference Guide bladzijde 343 en volgende.

.OPTION "PRINTER 0 :N. Standaard heeft N de waarde 4 en dit betekent dat de printer apparaatnummer 4 heeft. Als u met meer seriële printers werkt kunt u bepalen welke printer er aan het werk wordt gezet door de andere printer(s) een andere waarde voor

2.14 OPTION

N te geven, bijvoorbeeld 5 voor de daisywheelprinter en 6 voor de printer/ plotter.

.OPTION "PRINTER 1 :N bepaalt het secundaire adres van de printer. Bij een 801 printer betekent 0 dat hoofdletters en grafische tekens worden weergegeven, als N de waarde 7 heeft geeft deze printer hoofd- en kleine letters. Voor andere printers moet u het handboek van uw printer raadplegen.

.OPTION "PRINTER 2 :N bepaalt of u de af te drukken tekst alleen op de printer krijgt of dat deze tevens op het scherm te zien is. (Respectievelijk waardes 1 en 0).

.OPTION "RC 0 :N. Normaal is N gelijk aan 0 en dat wil zeggen dat de zogenaamde interrupt characters (CTRL-G, CTRL-Z, CTRL-W, F1, F2 en F3) onmiddellijk verwerkt worden. De controlekarakters hebben 'bitwaardes'. (CTRL-G = 1, CTRL-Z = 2, CTRL-W = 4, F1 = 8, F2 = 16, F3 = 32) Als N gelijk is aan 32 werkt F3 niet meer als interrupt karakter maar als gewoon controlekarakter. Is N gelijk aan 32 + 16 + 1 dan geldt hetzelfde voor F3, F2 en CTRL-G enzovoorts. In het programma INSTANT op de utility diskette wordt van deze .OPTION gebruik gemaakt. Het is uiteraard erg leerzaam om de procedures in de programma's op deze diskette af te drukken voor bestudering.

.OPTION "READ 0 :N. De standaardwaarde voor N is gelijk aan 0. Indien u N gelijk maakt aan 1 kunt u de editor van LOGO eventueel als tekstverwerker gebruiken.

.OPTION "SAVE 0 :N. Ook hier is bij de standaardwaarde 0 alles normaal, maakt u N echter gelijk aan 1 dan wordt de inhoud van de edit buffer op diskette opgeslagen in plaats van de inhoud van het werkblad. Normaal wordt bij een SAVE commando de edit buffer leeggemaakt, de inhoud van het werkblad overgebracht naar de edit buffer en deze inhoud gesaved. U kunt deze .OPTION gebruiken om met behulp van het commando PRINT [string] tekst tussen de inhoud van het werkblad te voegen. Als u de editor met CTRL-G verlaat wordt de betreffende tekst niet naar het werkblad overgebracht maar blijft deze in de buffer. Hierna kunt u de inhoud van de edit buffer saven.

.OPTION "STAMPCHAR 0 :N. De defaultwaarde voor N is gelijk aan 0 en betekent dat het teken dat in de matrix van 8×8 pixels aanwezig was overschreven wordt. N = 1 betekent combineren van de twee karakter via overschrijven. Wordt gebruikt voor het mixen van turtle graphics en character graphics. De waarde 2 zorgt voor REVERSE afbeelden van de 8×8 matrix. Dit stelt u in staat om tekst over graphics af te drukken, de tekst te verwijderen en later de graphics weer op te roepen.

.OPTION "STAMPCHAR 1 :N bepaalt de gebruikte karakterset. Standaardwaarde is 0 en betekent hoofdletters en grafische tekens, de waarde 1 betekent hoofd- en kleine letters.

.OPTION "STAMPCHAR 2 :N bestuurt het al dan niet REVERSE afdrukken van tekens. 0 (standaard) betekent normaal; 1 betekent reverse afdrukken.

2.14 OPTION

.OPTION "TEXTSCREEN 0 :N. Bepaalt welke sprites op het tekstscherm zichtbaar kunnen zijn. Let u vooral op het woord 'kunnen' want de zichtbaarheid van elke sprite wordt individueel bepaald door zijn eigen HIDETURTLE of SHOWTURTLE status. Standaard is geen enkele sprite in het tekstscherm zichtbaar. U herinnert zich dat de zichtbaarheid van elke sprite bepaald wordt door een enkel bit in de betreffende geheugenlocatie, namelijk geheugenplaats 53269, en wel sprite 0 = bit 1, getalwaarde 0 of 1; sprite 1 = bit 2, getalwaarde 0 of 2; sprite 3 = bit 3, getalwaarde 0 of 4 enzovoorts. N = 64 betekent dus dat alleen sprite 7 zichtbaar kan zijn, N = 255 betekent dat alle sprites zichtbaar gemaakt kunnen worden. Zie hiervoor ook deel 10, hoofdstuk 4.

8/2.15

Een behendigheidsspel in LOGO

De bedoeling van het spel dat u gaat schrijven is het besturen van de schildpad over een baan in de vorm van een ovaal. Zodra u in aanraking komt met een van de zijkanten van de baan is het spel afgelopen en krijgt u te zien hoeveel punten u heeft gescoord en wordt gevraagd of u nog eens wilt spelen. Het spel moet na inladen gelijk starten. U begint uiteraard eerst met inventariseren wat er zoal moet gebeuren, welke variabelen er zijn en als u alles op een rijtje heeft staan kunnen de noodzakelijke procedures geschreven worden. Wat is er nodig?

a) Er moet verteld worden wat de bedoeling van het spel is en wat de spelregels zijn.
b) De baan moet worden getekend, dat wil zeggen twee ovalen waarvan de binnenste cirkelbogen heeft met een door de speler te bepalen straal 'r' en de buitenste op een afstand 'b' daarvan verwijderd is. Voor de rechte stukken nemen we een vaste waarde van 100.
c) Er moet een besturingsmechanisme komen voor de schildpad, op elk moment moet worden getest of de schildpad niet tegen de wand botst. Bij een botsing is het spel afgelopen en worden de zaken verder afgewikkeld.

Als u een programma zelfstartend wil maken definieert u op het werkblad de variabele "STARTUP [programmanaam]. Deze variabele SAVED u gelijk met het werkblad en na het inladen via READ "naam werkblad zorgt deze variabele STARTUP er voor dat de procedure 'programmanaam' automatisch wordt uitgevoerd. Dus:

```
MAKE "STARTUP [RACE]
```

De procedure 'race' moet worden uitgevoerd. Deze heeft een begin en speelt zich af op een baan met cirkelbogen met straal 'R' en een breedte 'B', rechte stukken 100. Dus RACE kan er als volgt uit zien:

```
TO RACE
BEGIN
BAAN :R :B
END
```

Bij het begin van het spel wordt eerst het doel ervan uitgelegd (tekst1) en daarna de spelregels (tekst2). Dit alles gebeurt op een geheel tekstscherm.

```
TO BEGIN
NODRAW
HANDLEIDING
END
```

De procedure handleiding zorgt er voor dat de teksten worden afgedrukt.

```
TO HANDLEIDING
TEKST1
TEKST2
END
```

2.15 Een behendigheidsspel in LOGO

Teksten afdrukken is het eenvoudigste door lijsten teksten te laten printen, tekst1 is vrij eenvoudig:

```
TO TEKST1
PRINT [JE MOET PROBEREN OM DE]
PRINT [SCHILDPAD ZO LANG MOGELIJK]
PRINT [IN EEN OVALE BAAN TE HOUDEN]
PRINT [ZONDER DE VANGRAILS TE RAKEN]
PRINT [ ]
PRINT [BESTUREN VAN DE SCHILDPAD MET]
PRINT [BEHULP VAN DE VOLGENDE TOETSEN;]
PRINT [ ]
PRINT ['L' LINKSOM DRAAIEN]
PRINT ['R' RECHTSOM DRAAIEN]
PRINT ['G' VOORUIT IN DE RICHTING WAARIN DE]
PRINT [PUNT VAN DE SCHILDPAD WIJST]
PRINT [ ]
END
```

Tekst2 is iets gecompliceerder. Hierin moet de straal van de binnenste cirkelbogen en de breedte van de baan aan de speler worden gevraagd. Via het primitief 'REQUEST' kunnen we gegevens laten invoeren dit sluiten we af door een druk op de RERURN toets. Om te voorkomen dat u met een hele reeks toestanden te maken krijgt als de speler een reeks getallen invoert isoleert u via FIRST het eerste element uit de lijst en werkt daarmee verder. U voorkomt ook dat de speler letters in plaats van cijfers intikt door te testen of de invoer een cijfer is. Ook voor de voortstuwing van de schildpad maakt u een soortgelijke constructie. De STOP instructies hebben een andere werking dan in BASIC, ze onderbreken weliswaar de in uitvoering zijnde procedure, maar geven de controle terug aan de procedure die de onderbroken procedure heeft aangeroepen, in dit geval dus 'handleiding'. De speler kan dan opnieuw de gegevens invoeren, hopelijk nu correct, zo niet dan weer opnieuw enzovoorts.

```
TO TEKST2
PRINT [GEEF DE STRAAL VAN DE CIRKEL-]
PRINT [BOGEN VAN HET BINNENSTE OVAAL.]
PRINT [(NIET GROTER DAN 50)]
MAKE "R FIRST REQUEST
IF NOT NUMBER? :R TEKST2 STOP
PRINT [HOE BREED WORDT DE BAAN ?]
PRINT [(NIET BREDER DAN 40!)]
MAKE "B FIRST REQUEST
IF NOT NUMBER? :B TEKST2 STOP
PRINT [WAT IS DE VERSNELLING?]
PRINT [(BEGIN VOORZICHTIG MET 1)]
MAKE "VERSNEL FIRST REQUEST
IF NOT NUMBER? :VERSNEL TEKST2 STOP
END
```

De gegevens om de baan te kunnen tekenen zijn nu ingevoerd, zodat naar de procedure BAAN :R :B kan worden gesprongen.

```
TO BAAN :R :B
CLEARINPUT
DRAW PENUP FORWARD 75 HIDETURTLE ; OM DE OVALEN OP
```

2.15 Een behendigheidsspel in LOGO

EEN REDELIJKE PLAATS OP HET SCHERM TE PLAATSEN
PENDOWN RIGHT 90 FORWARD 50
CIRKELBOOG :R ; DE PROCEDURE HIERVOOR MOETEN WE NOG SCHRIJVEN
FORWARD 100
CIRKELBOOG :R
FORWARD 50 LEFT 90 PENUP FORWARD :B PENDOWN
RIGHT 90 FORWARD 50
CIRKELBOOG :R + :B
FORWARD 100
CIRKELBOOG :R + :B
FORWARD 50 PENUP RIGHT 90 FORWARD :R + :B
RIGHT 90 FORWARD 50 + :R + :B / 2 RIGHT 90 PENDOWN
SHOWTURTLE
START ;DE BAAN IS KLAAR, TURTLE STAAT OP BEGINPUNT, NAAR START
CONTROLE 0 ; MET CONTROLE BEHEERSEN WE HET SPEL EN TELLEN DE PUNTEN
END

Een cirkelboog bestaat uit een reeks cirkelsegmenten, elk segment wordt gevormd door iets te draaien, een klein stukje vooruit en weer iets draaien, een klein stukje vooruit en weer iets draaien. Dit moet zolang doorgaan dat de totale draaiing 180 graden bedraagt.
TO CIRKELBOOG :R
REPEAT 36 [SEGMENT :R]
END
De procedure voor het segment is simpel:TO SEGMENT :R
RIGHT 2.5
FORWARD :R * 3.14159 / 36 ;DE OMTREK VAN EEN CIRKEL IS IMMERS 2*PI*R
RIGHT 2.5
END

Voor de procedure start zorgen we dat de weg die de schildpad aflegt niet wordt getekend en we zetten de beginsnelheid zowel in X- als in Y-richting op 0.
TO START
PENUP
MAKE :VX 0
MAKE :VY 0
END

We moeten nu een verband leggen tussen het indrukken van de toetsen L, R en G en wat er met de snelheid van de schildpad gebeurt. Hiervoor gebruiken we drie procedures, namelijk UITLEZEN om de ingedrukte toets te registreren, BEWEGING om de invloed op de TURTLE aan de ingedrukte toets te koppelen en GAS om de invloed op de snelheid uit te rekenen.
TO UITLEZEN
IF RC? OUTPUT READCHARACTER
OUTPUT "
END
TO BEWEGING
LOCAL "SNEL
MAKE "SNEL UITLEZEN
IF :SNEL = " STOP
IF :SNEL = "R RIGHT 30 STOP
IF :SNEL = "L LEFT 30 STOP
IF :SNEL = "G GAS STOP
END

De invloed van een impuls op de snelheid in X- en in Y-richting hangt af van de grootte van de impuls en van de hoek die de bewegingsrichting van de schildpad maakt met de X-as.

2.15 Een behendigheidsspel in LOGO

```
TO GAS
MAKE "VX :VX + :VERSNEL * SIN
HEADING
MAKE "VY :VY + :VERSNEL * COS
HEADING
END
```

We moeten nu de schildpad gaan controleren, dat wil zeggen telkens kijken of hij niet in aanraking komt met de vangrails. In de procedure CONTROLE beginnen we met het oude puntentotaal, we kijken naar de nieuwe coördinaten van de schildpad, we kijken of er gebotst wordt en na opnieuw uitlezen hogen we het puntentotaal met 1 op.

```
TO CONTROLE :T
RACETURTLE
BOTS
BEWEGING
CONTROLE :T + 1
END
```

De nieuwe coördinaten van de schildpad hangen samen met de oude en met de verandering.

```
TO RACETURTLE
SETXY XCOR + :VX YCOR + :VY
END
```

Het botsen controleren we via een botsingsdetectie van de sprites, we zorgen er wel voor dat de toetsenbordbuffer leeg is. U weet dat in geheugenplaats 53279 het met de sprite corresponderende bit geset wordt als de sprite in aanraking komt met een object op het scherm. Na uitlezen wordt locatie 53279 automatisch 'gecleared'.

```
TO BOTS
IF READCHARACTER? STOP
IF KNAL? HIDETURTLE MAKE
"Q .EXAMINE 53269 STOPPEN
END
```

De procedure KNAL? berust op de bekende geheugenlocatie 53279 waarin bijgehouden wordt of een sprite met een pixel die aan staat in aanraking komt.

```
TO KNAL?
LOCAL "Q
MAKE "Q 0
MAKE :Q .EXAMINE 53279
OUTPUT 0 < BITAND :Q 1
END
```

Als er inderdaad wordt gestopt, moet ook aan de speler worden gevraagd of hij nog een keer wil spelen. Hierbij kunnen we dan beginnen bij TEKST2.

```
TO STOPPEN
SPLITSCREEN
PRINT [JE BENT VERONGE-
LUKT!]
PRINT SENTENCE [MAAR JE
HEBT] :T [PUNTEN GESCOORD.]
NOGMAALS?
END
```

De rest is eenvoudig:

```
TO NOGMAALS?
PRINT [WIL JE NOG EEN KEER
SPELEN?]
IF (FIRST FIRST REQUEST) = "J
NOGMAALS STOP
TOPLEVEL
END
TO NOGMAALS
CLEARINPUT
FULLSCREEN
RACE
END
```

Dit programma kunt u natuurlijk zelf interessanter en moeilijker maken door aan de turtle een spritevorm te geven,

2.15 Een behendigheidsspel in LOGO

bijvoorbeeld een raceauto en door de baan moeilijker te maken door er bijvoorbeeld chicanes in te bouwen. Verder kunt u nog verfijningen aanbrengen in het controleren van de invoer, zodat er geen verkeerde waarde voor variabelen ingevoerd kunnen worden.

8/2.16

Sprites

Om gemakkelijker te kunnen begrijpen welke primitieven nodig zijn is het nuttig om even terug te gaan naar deel 10, hoofdstuk 4. U vindt hier dat er:

a) Voor elke sprite een vorm in het computergeheugen moet worden geplaatst.
b) Aan de computer verteld moet worden waar de gegevens van een spritevorm staan. We doen dit in BASIC door het adres van het eerste gegeven in de geheugenplaatsen 2040-2047 te plaatsen (resp. sprites 0 t/m 7) door middel van POKE instructies, LOGO neemt u het een en ander werk uit de handen, de spritegegevens worden namelijk op een vaste plaats opgeslagen, namelijk sprite 0 op geheugenplaats 3072 en volgende (64*48), sprite n vindt u op 64*(n+48).
c) De sprites aan of uit te zetten door het beïnvloeden van de betreffende bits in geheugenplaats 53269.
d) De X- en Y-coördinaten van de sprites te veranderen, zodat de sprites bewegen.
e) Eventueel de sprite(s) in X- of Y-richting te vergroten of de kleur te wijzigen. Vergroten in X-richting gebeurt via geheugenplaats 53276, in Y-richting met behulp van locatie 53271 terwijl geheugenplaatsen 53287 tot en met 53294 de kleurinformatie voor de diverse sprites bevatten. Ook multicolor is mogelijk via de locaties 53285 en 53286.

U ziet dat er voor het manipuleren van sprites behoorlijk wat handelingen nodig zijn, in LOGO is er dan ook een hele reeks primitieven aanwezig die het werken met sprites mogelijk, ja zelfs veel gemakkelijker maakt dan in BASIC.

Primitieven:

TELL

U heeft bij een Commodore 64 de mogelijkheid om met 8 sprites tegelijkertijd te werken. U zult dus moeten vertellen voor welke sprite (0 t/m 7) bepaalde commando's bestemd zijn. Alle na TELL N volgende instructies blijven bestemd voor Sprite n, tot de bestemming van de instructies met een volgend TELL commando gewijzigd wordt. Als u LOGO opstart is sprite 0, de TURTLE, actief. De overige sprites zijn verborgen en hebben de penpunt ingetrokken (zie HIDETURTLE/ SHOWTURTLE en PENUP/PENDOWN).

SETSHAPE

Als een spritevorm definieert, bijvoorbeeld met de sprite editor van de utility

diskette, kunt u met behulp van SETSHAPE N aangeven welk spritenummer de door u ontworpen sprite krijgt. Alleen de TURTLE, sprite 0, kan roteren. De input voor SETSHAPE moet een geheel getal, van 0 t/m 7, zijn.

SHAPE

Geeft u het nummer van de op dat moment actieve sprite. Als u weet dat de gegevens van sprite 0 staan op pagina 48 (dat wil zeggen op geheugenplaats 64 * 48), dan kunt u het begin van de gegevens van sprite N vinden met behulp van de formule 64*(SHAPE + 48).

WHO

Dit primitief is aanwezig voor assistentie aan programmeurs met een slecht geheugen. Het geeft u als output het nummer dat u in de laatste TELL instructie heeft gegeven, dus het nummer van de op dat moment actieve sprite.

DRAWSTATE

Dit primitief is reeds eerder aan de orde gekomen, maar toen is er niet diep op ingegaan. Het is echter belangrijk genoeg voor een uitgebreider behandeling. DRAWSTATE vereist geen input, maar geeft als output een lijst van 9 elementen die u over allerlei zaken informeert. Via het pr mitief ITEM N kunt u het N-de element uit de lijst isoleren om daarmee eventueel verder te werken. Zo'n lijst kan er als volgt uitzien:
[FALSE TRUE 9 3 DRAW DOUBLECOLOR 12 8].
In volgorde hebben deze gegevens de volgende betekenis:

1) FALSE is PEN-UP, TRUE is PEN-DOWN.
2) TRUE als TURTLE zichtbaar is, FALSE bij HIDE TURTLE.
3) Achtergrondkleur.
4) Penkleur, -1 voor PENERASE.
5) DRAW of NODRAW mode.
6) Kleurmode, SINGLE- of DOUBLE-COLOR.
7) TEXT-, FULL- of SPLITSCREEN.
8) Achtergrondkleur tekstscherm.
9) Kleur van tekstkarakters op tekstscherm.

BITAND

Erg belangrijk bij het manipuleren van sprites. Vergelijkt de corresponderende bits van de twee inputwaarden en geeft hun logische 'AND'. Bijvoorbeeld BITAND 3 255 vergelijkt de bits in 00000011 en 11111111. De uitkomst is 00000011 ofwel 3. U kunt dit commando gebruiken om mogelijke zichtbaarheid van sprites uit te schakelen.

BITOR

Vergelijkbaar met BITAND, alleen is het resultaat nu de logische 'OR'. BITOR 5 9 vergelijkt 00000101 en 00001001, het resultaat is 00001101 ofwel 13. Probeert u een paar mogelijkheden met BITOR uit, dan zult u zien dat het resultaat meer keren de waarde 1 bevat dan de inputs, u kunt dit commando dan ook gebruiken om de diverse sprites eventueel (hangt van SHOWTURTLE of HIDETURTLE van de betreffende sprite af) zichtbaar te maken.

BITXOR

Hoort bij de vorige twee primitieven, maar geeft als resultaat de 'Exclusive OR'. Wordt gebruikt om 'opstaande' bits (1) te inverteren, laag (0) te maken. Ook weer te gebruiken bij het manipuleren van sprites.

8/2.17

De utility diskette (vervolg)

Op de schijf met 'gereedschappen', die u het werken met LOGO vergemakkelijken, vindt u behalve de reeds kort besproken procedures nog de volgende:

B&W
Deze procedure zet de output naar het TV-toestel om voor een juiste weergave op een zwart-wit toestel.

CCHANGE, CCHANGE.BIN, CCHANGE.SRC
Deze drie procedures helpen daar waar LOGO eigenlijk tekort schiet, ze geven namelijk rechtstreeks toegang tot de machinetaal. Wilt u deze procedures gebruiken dan moet u de 6502 machinetaal wel meester zijn, anders maakt u ongelukken. Deze drie procedures maken het u onder andere mogelijk om van lijnen die reeds getrokken zijn later de kleuren te veranderen. In grafische programma's natuurlijk uitstekend bruikbaar.

COLORS
Deze procedure bevat tevoren bepaalde variabelenamen voor de diverse kleuren die met de 64 mogelijk zijn.

STAMPFD
Bij de vorige lijst met procedures was STAMPER aanwezig, deze procedure maakte het mogelijk om tekens op het grafische scherm te plaatsen. De procedure STAMPFD maakt het u mogelijk om hetzelfde teken meerdere malen over een bepaalde afstand op het grafische scherm te plaatsen.

8/2.18

Werken met SPRED

Als u de sprite editor inlaadt met het normale commando READ "SPRED wordt een file geladen die alle noodzakelijke procedures en commando's bevat om sprites te kunnen besturen en bewerken. U moet uiteraard via TELL vertellen elke sprite u wilt gaan editen, maar hier schuilt een adder onder het gras. Als u namelijk via TELL 6 sprite 6 aanspreekbaar had gemaakt en deze via SETSHAPE 6 de vorm 6 had gegeven (in .SHAPES files zitten 8 vormen) dan zou toch na een eventueel commando TELL 3 sprite 6 bewerkt worden en niet sprite 3. Ook de TURTLE kunt u bewerken, maar deze gewijzigde vorm kunt u niet op diskette opslaan. Als u klaar bent met inladen en voorbereiden roept u de editor op met het commando EDSH (EDitSHape). Na even geduld oefenen krijgt u een matrix van 24×21 blokken aan de linkerkant van het scherm, waarin balletjes voor die pixels die aan zijn en punten voor de blanco pixels (die dus de kleur van de achtergrond hebben), alsmede de originele vorm van de sprite in het midden van het scherm. U kunt dan naar hartelust gaan tekenen. Net als elke editor heeft ook de sprite editor zijn eigen commando's namelijk:

Cursor naar beneden: Beweegt cursor 1 regel in de grote matrix naar beneden.

Cursor naar boven of ↑ : Beweegt cursor 1 regel in de grote matrix naar boven.

Curcor naar rechts: Beweegt cursor 1 positie naar rechts in de grote matrix.

Cursor naar links of pijl naar links: Beweegt cursor in grote matrix 1 positie naar links.

RETURN: Cursor gaat naar het begin van de volgende regel.

HOME: Cursor gaat naar de linker bovenhoek van de grote matrix.

*: Vult een pixel en beweegt cursor 1 positie naar rechts.

+: Vult pixel, de cursor verandert echter niet van positie.

SPATIE: Wist pixel en beweegt cursor 1 positie naar rechts.

DELETE: Wist pixel en beweegt cursor 1 positie naar links.

-: Wist pixel, cursor verandert niet van plaats.

CLR: Wist gehele matrix, cursor gaat naar linker bovenhoek.

X: Schakelt tussen BIGX en SMALLX, dat wil zeggen de sprite wordt groter of kleiner in horizontale richting.

Y: Idem voor de afmetingen in verticale richting, ofwel BIGY en SMALLY.

CTRL-9: Schakelt tussen normaal en reverse.

CTRL-C of RUN/STOP definiëren de vorm en verlaten de editor. De door u gemaakte sprites zijn dan nog niet op diskette opgeslagen. Met SAVESHAPES

2.16 Werken met SPRED

"NAAM (door u te bepalen) slaat u de nieuwe vormen op onder de door u gegeven naam. Let wel op dat een ander bestand op de diskette met dezelfde naam wordt verwijderd.

8/2.19

Demonstratieprogramma's

SPRITES
Dit programma bevat procedures om files met sprite-vormen in te lezen, afmetingen van sprites te veranderen, instructies aan sprites te geven en botsingen van sprites met een voorwerp op het scherm te ontdekken.

SPRED
Een sprite editor die het u mogelijk maakt om uw eigen sprites te ontwerpen. In hoofdstuk 8/2.18 werd het werken met deze sprite editor reeds besproken.

ANIMALS
Dit programma bevat de sprite-variabelen die corresponderen met het programma ANIMAL.SHAPES en zorgt er tevens voor dat de sprite-variabelen automatisch in dit laatste programma worden ingelezen.

ANIMALS.SHAPES
Zorgt voor een 'dierentuin' van 8 beesten op het scherm

ASSORTED
Ook hier een combinatie van twee elkaar aanvullende programma's, ASSORTED bevat de gegevens van diverse vormen van sprites en zorgt ervoor dat deze ingelezen worden uit het programma ASSORTED.SHAPES.

ASSORTED.SHAPES
Gegevens omtrent de vorm van acht verschillende sprites. Het programma wordt automatisch ingeladen door ASSORTED.

RUNNER
Een voorbeeld van een animatie met behulp van LOGO. Acht verschillende houdingen van een hardloopster zijn gedefinieerd en door beweging en afwisselend zichtbaar maken van de diverse stadia van de beweging wordt een suggestie van een rennende vrouw opgeroepen.

RUNNER.SHAPES
Bevat de gegevens voor de sprite-vormen die in het programma RUNNER gebruikt worden.

SHAPES
Diverse geometrisch gevormde sprites. De gegevens van de vormen staan weer in het bijbehorende SHAPES.SHAPES.

SHAPES.SHAPES
Gegevens voor SHAPES, worden door het vorige programma weer automatisch ingelezen.

2.19 Demonstratieprogramma's

VEHICLES

VEHICLES.SHAPES
Ook hier weer dezelfde combinatie van twee files. Een met gegevens voor de vorm en een met de rest van de instructies. Nu betreft het diverse voertuigen.

DINOSAURS
Een familie dinosaurussen loopt over uw scherm, knabbelt aan bomen etcetera. Interessant voor diegenen die interesse hebben in de gebeurtenissen van de oertijd.

SUBMARINE
Een duikboot op het scherm. Als u een spel gaat programmeren in LOGO kunt u de in dit programma verzamelde wetenschap misschien gebruiken in uw versie van 'ZEESLAG'.

SPRITEDEMOS
In het instructieboek dat u bij het aanschaffen van LOGO ontvangt worden in het hoofdstuk over sprites diverse procedures beschreven. SPRITEDEMOS bevat deze procedures.

VELOCITY
Maakt het u mogelijk om de snelheden van sprites te controleren. Erg nuttig als u denkt over een programma dat auto- of motorracen simuleert.

MUSIC
Procedures die reeksen tonen voortbrengen met behulp van de commodore 64. U kunt de diverse parameters voor attack, decay, sustain en release wijzigen (zie voor een uitgebreide behandeling van deze begrippen hoofdstuk 11 dat de Sound Interface Device behandelt).

SOUND
Dit programma bevat de procedures om het geluid voort te brengen. Wordt automatisch ingelezen door MUSIC.

TWINKLE
De procedures om 'Twinkle, Twinkle, Little Star' te spelen.

Voorts bevat de utility diskette nog vier files die bruikbaar zijn om machinetaal vanuit LOGO te gebruiken, namelijk:

ADDRESSES
Beschrijving van adressen in de LOGO interpreter voor de Assembler.

AMODES
De diverse adresseermogelijkheden van de 6502 processor.

ASSEMBLER
Logo procedures voor Assembler.

OPCODES
De 6502 mnemonics voor de Assembler. In de toekomst gaan wij hier nader op in; kennis van machinetaal is dan noodzakelijk.

8/2.20

Verwarringen in LOGO

THING, afgekort ':' is geen ding maar een WAARDE. PRINT :NUM geeft u de waarde opgeslagen in de geheugenplaats met de naam NUM.

Een variabelenaam kan meerdere malen voorkomen in een programma, maar dit hoeft niet altijd dezelfde variabele te betekenen. Als in de naam van een procedure ook variabelenamen voorkomen, dan gelden deze namen met hun bijbehorende waarden alleen voor deze procedure. Dezelfde namen met andere waarden kunnen daarnaast voorkomen geldend voor andere procedures.

Voor SENTENCE is een woord als input ook een LIST, een 'een-woords list', de output is altijd een LIST, hiermee kunt u dus een woord omzetten zodat het bij een list kan worden gevoegd.
Voor LIST moet de input uit lijsten bestaan.
WORD geeft altijd een woord.

8/2.21

Muziek met LOGO

Vanuit BASIC is het componeren met behulp van de Commodore 64 een behoorlijk gecompliceerde taak waar reeksen en reeksen DATA regels aan te pas komen, of nog erger hele legers POKE instructies. LOGO maakt het u, zoals we verder in dit hoofdstuk zullen zien, erg gemakkelijk. Voor diegenen onder u die het toch liever gecompliceerder maken heeft LOGO ook voldoende mogelijkheden in huis, maar ook hierover later.

Voor de algemene theoretische ondergrond van de muzikale mogelijkheden verwijs ik u naar hoofdstuk 11 van dit boek of naar de Programmer's Reference Guide. Termen als ATTACK, DECAY, SUSTAIN en RELEASE, FILTER, RESONANTIE enzovoorts worden daar duidelijk gemaakt.
Belangrijk in dit kader is dat de SOUND CHIP van de Commodore 64 gesitueerd is op de adressen 54272 tot en met 54300. Voor de goede orde volgen in tabelvorm enige gegevens van deze soundchip.
54272 Low byte van frequentie.
54273 High byte van frequentie.
54274 Low byte van pulsbreedte.
54275 High byte van pulsbreedte.
54276 Golfvorm.
54277 Attack/Decay.
54278 Sustain/Release.
Deze gegevens betreffen de eerste stem, exact dezelfde volgorde heeft de lijst 54279-54285 voor de tweede en 54286-54292 voor de derde stem.
54293 Low byte filterfrequentie, 54294 High byte idem.
54295 Filterresonantie.
54296 Geluidssterkte.
54297 en 54298 Potentiometers 1 en 2.
54298 Oscilatoren, High byte.
54299 Envelope, High byte.

In dit hoofdstuk gaat u praktisch muziek maken met de Commodore 64 door gebruik te maken van de in LOGO aanwezige primitieven en procedures; deze zijn natuurlijk allemaal gebaseerd op het gebruik van DEPOSIT, EXAMINE voor de geheugenadressen en BITAND en BITOR om op die adressen de noodzakelijke bits te beïnvloeden.
U begint met het normaal opstarten van LOGO, waarna u de utility diskette in de diskdrive plaatst. READ "MUSIC <RETURN> laadt de file "MUSIC" in van de diskette en dit bestand zorgt er automatisch voor dat ook het bestand "SOUND" wordt ingelezen.

SOUND bevat de procedure SOUND die wordt gebruikt voor het werkelijke creëren van de geluiden uit de luid-

2.21 Muziek met LOGO

spreker. Deze procedure heeft de volgende vorm:
TO SOUND :PITCH :DUR :AD :SR :WAVEFORM ;er zijn dus 5 inputs nodig.
.DEPOSIT 54296 15 ; geluidssterkte wordt maximaal gemaakt.
.DEPOSIT 54277 :AD ; waarde voor ATTACK/DECAY als input.
.DEPOSIT 54278 :SR ; waarde voor SUSTAIN/RELEASE als input.
.DEPOSIT 54273 QUOTIENT :PITCH 256 ; High byte van frequentie wordt berekend.
.DEPOSIT 54272 BITAND :PITCH 255 ; Low byte van frequentie.
.DEPOSIT 54276 :WAVEFORM ; golfvorm wordt vastgelegd.
REPEAT :DUR* :TEMPO [ ] ; duur van de toon wordt bepaald.
.DEPOSIT 54276 0 ; de luidspreker wordt afgezet.
END

De overige procedures uit "SOUND kunt u ook allemaal bekijken en ze zijn er allen bedoeld om de SOUNDCHIP van de noodzakelijk informatie te voorzien. Alle procedures die u nodig heeft zijn nu aanwezig in het werkblad en u kunt naar hartelust gaan experimenteren.
Een paar voorbeelden:
Een simpel geluid opbouwen:
REPEAT 10 [SSH 20 SSH 10 SSH 20]
of SSHER [8 8 4 4 8] TEMPO 40 SSHER [8 8 4 4 8] TEMPO 10 SSHER [8 8 4 4 8]
of PLAY [0 1 2 3 4 5] [15 15 15 15 15]
of SING [2 4 6 7 9 11 13 14]
U zou bijvoorbeeld PLAY en SING op zich weer in procedures kunnen verwerken.
Nog een voorbeeld:
TO CLEM
PLAY [3 3 3 0 5 5 5 3] [5 5 10 5 5 10 10]
PLAY [3 5 6 6 5 4] [5 5 10 10 5 5 20]
END
Als u enige tijd aan muziek wilt besteden zult u merken dat het creëren van melodieën in LOGO veel gemakkelijker is dan in BASIC.

8/2.22

LOGO en Machinetaal

Voor diegenen onder u die verder willen kijken dan alleen LOGO is er een mogelijkheid om rechtstreeks vanuit LOGO met machinetaal te werken. Indien u geen ervaring heeft met de machinetaal van de Commodore 64 adviseren wij u allereerst het hoofdstuk 9 van dit boek grondig te bestuderen.
Rechtstreeks toegang tot het geheugen van de computer krijgt u zoals reeds eerder besproken, met de Primitieven .EXAMINE en .DEPOSIT, die zoals bekend te vergelijken zijn met PEEK en POKE in BASIC. Deze commando's kunt u vooral gebruiken wanneer u aan de keuze PRINTER/NOPRINTER niet voldoende heeft voor uw in- of uitvoer. De getallen die bij deze commando's worden gebruikt zijn normale getallen uit het tientallige stelsel, u hoeft zich hier niet te vermoeien met de hexadecimale schrijfwijze. U moet echter wel oppassen voor het zogenaamde 'memory mapping' want ogenschijnlijk worden verschillende waarden in dezelfde geheugenplaats opgeslagen. Welke 'map' wordt gebruikt ligt vast in de drie meest rechtse bits van geheugenlocatie nummer 1. Er zijn drie mogelijkheden:
Map 6: toegang tot I/O gebied ($D000-$DFFF) en de KERNEL. De laatste natuurlijk alleen om uit te lezen en gebruik van te maken.
Map 4: dit geeft u toegang tot het opslaggebied van LOGO en maakt het u mogelijk om woorden, lijsten en procedures te bekijken en te veranderen.
Map 2: geeft u onder andere de beschikking over de karakter-ROM.
U moet er echter wel op letten dat de primitieven .EXAMINE en .DEPOSIT verschillende 'mapconfiguraties' hebben, zie hiervoor de bespreking van .OPTION.

8/2.23

Files op de utility diskette

Op de utiliy diskette treft u onder de naam 'ADDRESSES.LOGO' een bestand aan dat u de namen geeft waaronder belangrijke adressen in LOGO bekend zijn. Het is erg nuttig om deze adressen goed te bestuderen want de benamingen zijn eigen voor LOGO en niet te vinden in bijvoorbeeld de Commodore 64 programmers reference guide, bovendien heeft u dit bestand nodig als u daadwerkelijk een door u geschreven machinetaalroutine gaat assembleren. U begint met het bestand 'ASSEMBLER' in te lezen via het commando READ "ASSEMBLER; vervolgens gaat u naar de LOGO editor met het commando 'TO NAAM.CODE' en schrijft uw machinetaalroutine met inachtneming van de verschillen tussen de syntax van een normale assembler en de LOGO assembler zoals hieronder worden beschreven. Nadat u de geschreven procedure heeft gedefinieerd kunt u hem assembleren door de volgende commando's:

READ "ADDRESSES
ASSEMBLE "NAAM.CODE

Als dit is gebeurd kunt u in het vervolg de machinetaal-routine aanroepen met CALL :NAAM 0.

Voorts treft u op de utility diskette een bestand aan met de naam "OPCODES.LOGO". Dit bestand geeft u de operation codes van alle mnemonics die in de LOGO assembler gebruikt worden.

8/2.23.1

De verschillen in Syntax van de LOGO assembler met een normale assembler

Binnen het programma kunnen LABELS gebruikt worden, de naam van het LABEL dient u te laten volgen door een dubbele punt ':'. Als volgt:
HERHAAL: CMP # 0
Verwijzingen naar adressen op de ZERO page moeten een UITROEPTEKEN '!' hebben voor het LABEL of voor de uitdrukking die op de ZERO page staat.
Er moet een spatie volgen na elk uitroepteken of na elk teken '#' dat aangeeft dat er een getal volgt.
De operand van een instructie kan een woord (label), een getal, een lijst of een letter zijn. In het laatste geval moet hij voorafgegaan worden door een stel aanhalingstekens en wordt daadwerkelijk de ASCII waarde van de letter gebruikt.
Binnen een lijst wordt alles geëvalueerd als een normale LOGO uitdrukking. Staat de lijst als eerst op een regel dan mag het evalueren geen waarde uitvoeren, maar wordt er alleen gekeken naar label of PRINT of dergelijke. Volgt de lijst na een instructie dan wordt er één of andere output verwacht. Alle labels zijn LOGO variabelen waarvan de waarde gelijk is aan die van het adres dat op dat moment geassembleerd wordt. De procedures 'LO8' en 'HI8' kunnen ook goed gebruikt worden in lijsten, deze geven als output de lage en hoge 8 bits van een getal.

De procedure '$' vraagt als invoer een woord bestaande uit een hexadecimaal getal en voert de decimale waarde hiervan uit. U kunt dus hexadecimale getallen gebruiken in uw assembler listings door binnen een lijst de procedure '$' aan te roepen.
Door de procedure '$BASE' aan te passen met behulp van 'MAKE' (bijvoorbeeld MAKE "$BASE 8) kunt u ook getallen uit het achttallige stelsel gebruiken. Op dezelfde manier kunt u ook het met binaire getallen werken.
Als u een lang programma probeert te assembleren is het heel waarschijnlijk dat u problemen krijgt met de beschikbare geheugenruimte. De enige oplossing hiervoor is ruimte maken door de file 'OPCODES' in te lezen, de door u overbodig geachte instructies te verwijderen en de ingekorte file als een nieuw 'OPCODES' bestand op diskette op te slaan en met deze diskette verder te werken.

8/2.24

Het opslaan van routines

In afwijking van de standaard procedure om een LOGO programma op diskette op te slaan moet u hiervoor gebruik maken van het primitief 'BSAVE'. Dit vraagt drie inputs, namelijk de bestandsnaam, het startadres en het eindadres van de machinetaalroutine. Het startadres vindt u direct na het assembleren in de LOGO variabele :ORG en het eindadres in de LOGO variabele :END.
Een juiste formulering zou dus zijn:
SAVE "TOOLS.BIN :ORG :END+1
Het inladen gebeurt met het commando:
BLOAD "TOOLS.BIN
U mag nooit vergeten behalve de echte machinecode ook de waardes van de LOGO variabelen, die u immers gebruikt voor de adressen bij 'CALL', vast te leggen. Dit doet u het gemakkelijkst door te tikken 'EDIT NAMEN' en dan in het begin van de edit buffer 'TO INIT' te zetten. Vervolgens gaat u met de cursortoetsen naar het einde van de lijst met variabelen en tikt daaronder:
BLOAD "TOOLS
END
Daarna definiëren met CTRL-C en vervolgens op diskette opslaan met:
<SAVE "TOOLS [INIT]>.
Om de zaak later te gebruiken tikt u:
READ "TOOLS en nadat dit is gebeurt tikt u INIT.

8/2.25

Belangrijke adressen

Tenslotte geven we u een lijst met belangrijke adressen en hun benamingen:
NARG2 Dit is de tweede input voor .CALL. 4 bytes.
FREEPZ De eerste plaats op de ZERO page die vrij beschikbaar is voor de gebruikers. Deze locatie en de volgende 32 zijn beschikbaar (tijdelijk) voor machinetaalroutines van de gebruiker.

Vectoren en vlaggen:
TOPIRQ Deze vector wijst normaal naar de LOGO routine die 60 maal per seconde wordt uitgevoerd als het beeld 'gescanned' wordt op regel 16. U kunt deze vector omleggen mits u hem bij het verlaten van uw machinetaalroutine weer terugzet in de oorspronkelijke waarde.
KERVER Als deze locatie het hexadecimale getal 64 bevat werkt LOGO in zwart/wit.
COLMOD Als er in SINGLE-COLOR mode gewerkt wordt bevat dit adres de waarde 0 en bij DOUBLE-COLOR mode de waarde 1. Als u in dit adres wijzigingen gaat aanbrengen krijgt u vreemde effecten.
VERSION Deze locatie bevat het versie-nummer van de gebruikte LOGO. Elke nieuwe release heeft een ander nummer.
RPTFLG Als dit adres de waarde 128 bevat repeteren alle toetsen, als de waarde 255 is repeteert geen enkele toets. Wilt u persé repeterende toetsen dan zal blijken hoe moeilijk het is om de combinatie SHIFT/COMMODORE aan te slaan. Dit kunt u indien u wenst vervangen door: PRINT1 CHAR 14 en PRINT1 CHAR 142.

ROUTINES en ENTRY/EXIT adressen.
OTPFX2 Als u naar dit adres sprint zorgt .CALL voor het uitvoeren van het gehele getal opgeslagen in de locaties NARG 2 tot NARG2+3.
OTPFIX Als het vorige, maar met terugkeer naar LOGO met deze getalwaarde opgeslagen in het adres in de ZERO page aangegeven door het Y-register en de drie daarop volgende bytes.
PPTTP Wordt gebruikt om uit een machinetaalroutine te springen als er iets mis gaat. PPTTP zorgt ervoor dat het LOGO primitief TOPLEVEL wordt uitgevoerd, waardoor de controle over de machine teruggegeven wordt aan LOGO.
COUT De normale scherm uitvoer routine van LOGO. Het karakter uit de accumulator wordt afgedrukt op het scherm.
U ziet dat dit lijkt op een lijst van nuttige POKE adressen die u in Deel 7/17 vindt. In de bij LOGO geleverde handleiding bevinden zich diverse voorbeelden van

2.25 Belangrijke adressen

machinetaalroutines. Het is echter een valse hoop om te verwachten dat machinetaal programmeren in LOGO mogelijk is zonder grondige kennis van de machinetaal. Nogmaals verwijzen we u naar Deel 9 van dit boek.

8/3

PASCAL

Inhoud

8/3.1 Inleiding
8/3.2 De filosofie achter Pascal
8/3.3 Compiler versus interpreter
8/3.4 Welke Pascal?
8/3.5 Voorzichtig beginnen
8/3.6 Een beetje rekenen
8/3.7 In- en uitvoer
8/3.8 Procedures, functions en lussen
8/3.9 Voorwaardelijke statements
8/3.10 BNF-notatie
8/3.11 Nuttige routines
8/3.12 Voorbeeldprogramma's
8/3.13 Netwerkplanning

8/3.1

Inleiding

De ene taal is de andere niet. Dat geldt niet alleen voor de normale menselijke talen, maar ook voor computertalen. Echter waar menselijke talen – op Esperanto na – gewoon ontstaan zijn, daar zijn computertalen oftwel programmeertalen uitgedacht. Iedere computertaal is het produkt van een lange ontwikkeling, waarbij er lang en breed is nagedacht over het hoe en waarom van zo'n programmeertaal.

Achter een computer steekt meestal een hele filosofie, een heel netwerk van ideeën en redenaties, die uiteindelijk tot de taal geleid hebben zoals die gebruikt wordt. En die filosofie is juist bij Pascal heel belangrijk.

We ontkomen er dan ook niet aan om voor we ook maar één Pascal-commando onder de loep nemen eerst op die achterliggende filosofie in te gaan. Daarbij nemen we dan BASIC, de ingebouwde en meest bekende taal, als vertrekpunt.

BASIC, ook de C-64 BASIC, is een typisch voorbeeld van een vrij simpele, snel aan te leren taal waarin de beginner al gauw resultaten kan bereiken. Er worden weinig eisen aan de opzet en de structuur van het BASIC-programma gesteld. Dat houdt in dat de meeste programma's niet zozeer eerst ontworpen en pas daarna geschreven worden maar in feite al programmerend ontstaan. Hetgeen ideaal is voor de beginnende hobby-programmeur, want al doende leert men.

Maar om een goed doortimmerd programma te schrijven leent BASIC zich toch minder. Juist omdat veel programmeurs hun programma's niet op papier maar rechtstreeks in de computer ontwerpen zijn grotere programma's vaak een ware warboel van sprongen en variabele-namen. Een vaak gebruikte vergelijking is een pan spaghetti, hoewel Gordiaanse knoop de toestand eigenlijk nog beter omschrijft.
Ook voor de programmeur zelf is er na een tijdje geen wijs meer uit te worden.

De reden daarvoor is in feite voor hand liggend; in BASIC is namelijk (bijna) alles toegestaan. De taal is regelnummer-geörienteerd, hetgeen wil zeggen dat alle sprongen naar regelnummers gaan. Of het nu een subroutine betreft of niet maakt weinig uit, in het BASIC-programma staat bijvoorbeeld:

```
1230 GOSUB 2341
```

Wat er gebeurt op regel 2341 moeten we zelf maar uitmaken, uit het programma blijkt het in ieder geval niet.

3.1 Inleiding

Ook wat betreft variabele-namen is het een en ander in BASIC niet echt glashelder te nomen. We mogen die namen weliswaar zolang maken als we willen, slechts de eerste twee tekens zijn significant. Dat will zeggen dat het vrijwel onmogelijk is om met de naam de betekenis van zo'n variabele vast te leggen, dat moet in feite op een los papiertje worden bijgehouden. Wie een bestaand groot BASIC-programma eens wilt uitpluizen loopt daar ook voortdurend tegenop, temeer daar niets de programmeur ervan weerhoudt om zo'n naam – en dus die variabele – voor verschillende doeleinden te gebruiken.

Als we eens naar de soorten variabelen kijken wordt het nog erger. Natuurlijk kent BASIC de array, maar zo'n array, hoeft helemaal niet geDIMensioneerd te worden. Als we een array-variabele gebruiken zonder eerst met een DIM-statement de grootte op te geven, neemt BASIC zonder meer aan dat de dimensie maximaal 10 bedraagt. Aan de ene kant erg makkelijk, maar aan de andere kant bijzonder onduidelijk. Bovendien kent C-64 BASIC weliswaar twee soorten numerieke variabelen – integer en floating point –, in het gebruik ervan is ze echter bijzonder toegevelijk als we die beide types door elkaar heen willen gebruiken. Een regel als:

100 A% = B

waarbij A% een integer-variabele en B een real getal voorstelt is zonder meer toegestaan. BASIC verzorgt automatisch de omzetting van het ene type naar het andere, slechts als de waarde van B te groot of te klein is volgt er een foutmelding.

Zelfs klinklare nonsens als:

2430 A% = 12.25

wordt door BASIC gaccepteerd, terwijl geen enkele programmeur die even doordenkt zoiets zal programmeren.

Komtom, BASIC kent slechts drie types variabelen – integer, floating point en string – maar daarmee is dan ook bijna alles toegestaan. Hetgeen als onvermijdelijk gevolg heeft dat men tijdens het programmeren ook de meest kromme constructies gebruikt.

Iets wat de meeste BASIC-programmeurs al helemaal niet gebruiken is de mogelijkheid om functies te definiëren. Het DEF FN-commando is in Commodore-64 BASIC heel magertjes, zodat het meestal geen nuttige rol kan vervullen. Als we een dergelijke mogelijkheid nodig hebben zullen we meestal naar een subroutine grijpen.
Maar als we tenslotte naar de commando's kijken waarmee de 'intelligentie' in een BASIC-programma wordt gebracht blijkt dat de taal hier wel heel beperkt in is. Slechts stucturen als

IF voorwaarde THEN commando's

en

ON variabele GOSUB/GOTO regelnummers

staan de BASIC-programmeur ter beschikking.
Mogelijk mist u het GOTO-commando als programma-besturing, maar dit speelt eigenlijk een heel andere rol. Door de GOTO kunnen we sommige

3.1 Inleiding

programmalussen maken die anders onmogelijk zouden zijn, maar in een beter gestructureerde taal zouden we dit commando niet nodig hebben.

Al met al kan gezegd worden dat BASIC, hoewel makkelijk te leren, heel wat tekortkomingen vertoont. En dat niet op het gebied van de besturing van geluid of graphics, maar als programmeertaal op zich. Hoewel we niet zover willen gaan als sommige professoren in de informatica – er zijn er die weigeren college te geven aan studenten die BASIC als eerste programmeertaal geleerd hebben, omdat deze studenten volgens hen niet meer in staat zouden zijn om de BASIC-denkwijze af te leren – denken we toch dat BASIC alles behalve ideaal is om goed in te leren programmeren.

BASIC-programmeurs zijn vaak slordige programmeurs, omdat de taal op zich ze geen netheids-eisen oplegt. Alles mag, en wordt dan ook gebruikt, terwijl het daarnaast ten ene male onmogelijk is om een echt zelf-documenterend programma in BASIC te schrijven. De beperkingen die aan variabele-namen opgelegd worden, tezamen met de regelnummer-structuur houden in dat een BASIC-programma zonder veel commentaar in feite onleesbaar is. En ook bij dat commentaar zien we dat slordigheid en luiheid van de programmeur vaak overwint, het commentaar wordt bijna altijd vergeten.

8/3.2

De filosofie achter Pascal

De ontwerpers van Pascal, Kathleen Jensen en Niklaus Wirth, hadden zich als hoofddoel gesteld om een taal te ontwerpen die geschikt zou zijn om het programmeren mee te onderwijzen. Nu hadden de ontwerpers van BASIC (Beginners All Symbolic Instruction Code) weliswaar dezelfde doelstelling, maar in het geval van Pascal gingen de ideeën nog wel een stuk verder.

In de woorden van Wirth zelf:
'A language suitable to teach programming as systematic discipline based on certain fundamental concepts clearly and naturally reflected by the language.' In goed Nederlands kan dat vertaald worden als:
'Een taal die geschikt is om programmeren te onderwijzen als een systematische discipline, gebaseerd op een aantal fundamentele ondergronden die op duidelijke en natuurlijke wijze in de taal tot uiting komen.'

Als we dat nogmaals vertalen, maar dan naar begrijpelijker Nederlands, komt het erop neer dat de taal Pascal ontworpen is om de logica van het programmeren zo duidelijk mogelijk uit te drukken. Waar in BASIC vaak allerlei 'handigheidjes' gebruikt worden die echter alles behalve rechttoe-rechtaan zijn, daar zullen in Pascal dergelijke handigheidjes niet nodig zijn. De structuur van de taal stelt de programmeur in staat het één en ander veel rechtstreekser uit te drukken.

Dat uit zich onder andere in de grote hoeveelheid data-structuren, want in Pascal is het mogelijk gegevens op vele manieren in een programa op te nemen. In de gedachten van Wirth is het (kunnen) kiezen van de juiste data-structuur de allereerste stap in het ontwerpen van een goede programma.

Zo kunnen we in Pascal zelfs eigen variabele-typen ontwerpen, waarbij we precies die namen gebruiken die ons uitkomen. Om een voorbeeld te geven, we kunnen bijvoorbeeld de variabele 'kleur' definiëren, die als waardes 'rood', 'groen', 'geel' en 'blauw' kent. Door nu in een Pascal-programma deze variabele kleur als teller-variabele in een lus te gebruiken kunnen we een regel als:

```
for c:= geel downto rood do ...
```

schrijven. Het is op het eerste gezicht even verwarrend, maar blijkt in de praktijk zeer heldere programma's op te leveren.

Ook de logica-structuren van Pascal

3.2 De filosofie achter Pascal

zijn bijzonder uitgebreid, er zijn vele vormen van lussen mogelijk. Wat echter niet zonder meer kan is de oude trouwe GOTO, wie in een Pascal-programma de GOTO nodig heft maakt volgens Wirth in feite een ontwerpfout. Desondanks heeft deze vernieuwer het niet aangedurfd om de GOTO helemaal buiten spel te zetten, de mogelijkheid is wel aanwezig om een GOTO te gebruiken.

Aan de andere kant mist Pascal de uitgebreide mogelijkheden van BASIC om strings te manipuleren. Wie functies als MID$ of LEFT$ nodig heeft zal ze zelf moeten ontwerpen. In feite is Pascal een raamwerk, een taal waarin alleen de 'primitieve' gereedschappen aanwezig zijn.
Wie meer nodig heeft kan dergelijke uitbreidingen echter vrij simpel in Pascal aanbrengen, de taal is daar meer dan krachtig genoeg voor.

Namen, bijvoorbeeld van variabelen, kunnen zo lang zijn als men maar wilt, waarbij – afhankelijk van de versie van Pascal die men gebruikt – er meestal zo'n 8 of meer tekens worden gebruikt om de namen van elkaar te onderscheiden. Dat houdt in dat die namen zo gekozen kunnen worden dat de betekenis ervan in de naam tot uitdrukking komt.

Maar niet alleen variabelen hebben namen in Pascal, ook de subroutines worden met een naam aangeroepen. Hoewel, de term 'subroutine' mag eingelijk niet gebruikt worden, in Pascal heten het procedures en fuctions.
Zo'n procedure is in feite een programmaatje op zich, wat allerlei extra voordelen met zich meebrengt. Zo kunnen variabelen in Pascal zowel globaal als lokaal gedefinieerd worden, wat te vertalen valt met 'geldig in het hele programma' of 'geldig in slechts een bepaalde procedure'. Dit voorkomt de in BASIC maar al te vaak optredende problemen van dubbel gebruik.
Nog veel belangrijker is de mogelijkheid van recursie, waarbij een bepaalde procedure zichzelf weer aanroept.
Sommige problemen zijn feitelijk alleen via dit recursie-mechanisme op te lossen, hetgeen de BASIC-programmeur voor grote problemen stelt. In Pascal is zoiets echter vloeiend en natuurlijk op te schrijven. De ware kracht hiervan zal echter pas duidelijk worden als we wat voorbeelden gaan bekijken.

Fuctions zijn weer een heel ander hoofdstuk. In BASIC hebben we de beschikking over een heel beperkte DEF FN mogelijkheid, in Pascal kan een funtie zoveel regels (en parameters) omvatten als we zelf willen. Alweer, een function is in feite een soort sub-programma, dat op zijn beurt weer ander sub-programma's kan aanroepen.

Het verloop van een programma kan in Pascal bestuurd worden met een ware weelde aan commando's. Zo zullen we onder meer constructis zoals:

fi ... then ... else
while ... do
repeat ... until
for ... to ... do ...

leren kennen.

De basis van een Pascal-programma is het simpele statement, dat slechts één enekele actie omschrijft. Het is echter

3.2 De filosofie achter Pascal

bijzonder eenvoudig om die enkelvoudige statements tot meervoudige statements samen te voegen, simpelweg door er de zogenaamde 'statementhaakjes' omheen te plaatsen. Nu is de term 'haakjes' wat misleidend, in feite zijn deze haakjes de termen begin en end, die in ieder Pascal-programma vele malen voorkomen. Zometeen, bij de eerste voorbeelden, zal het een en ander al snel glashelder worden.

Pascal is dusdanig ontworpen dat fouten al bij het schrijven van een programma zoveel mogelijk vermeden worden. Zo moeten alle variabelen gedeclareerd worden, met andere woorden, een variabele moet omschreven worden met naam en type vóór deze gebruikt kan worden. Daarvoor is een speciaal gedeelte van het programa gereserveerd, het zogenaamde variable declaration part.

Het is even wat meer werk, maar de programmeur wordt hierdoor wel gedwongen om het programma zelfdocumenterend te maken, terwijl schrijffouten in een variabele-naam onmiddelijk opgemerkt worden.

Daarnaast zal Pascal nooit dubbelzinnig gebruik van variabelen toestaan. Als er een onduidelijkheid is over wat een bepaalde naam betekent, of als de programmeur probeert om bijvoorbeeld een real-waarde aan een integervariabele toe te kennen komt dat op een foutmelding te staan.

Al die aspecten tezamen houden in dat het zonder meer lastig is om een onduidelijk Pascal-programma te produceren. Wie in Pascal een spaghetti-programma wil schrijven zal daar duidelijk zijn of haar best voor moeten doen.

8/3.3

Compiler versus interpreter

Een ander groot verschil tussen de in de 64 ingebouwde BASIC en Pascal is dat de laatste een 'compilertaal' is.
Zoals bekend zal de 64 tijdens het uitvoeren van een BASIC-programma elke regel stuk voor stuk afhandelen, waarbij ieder afzonderlijk commando bekeken wordt en tot de uitvoering van een kleiner of groter stukje machinetaal uit het ROM leidt. Op zich een uitstekend systeem, maar zeker niet zaligmakend.

Pascal – althans bijna alle betere versies – werkt op een geheel andere basis. Een Pascal-programma wordt namelijk eerst in zijn geheel vertaal tot een voor de computer verteerbaarder vorm, waarna die vertaalde versie wordt uitgevoerd. Dat houdt in dat er nog al wat verschillen optreden vergeleken met BASIC.

Zo zal bij BASIC elke fout pas ontdekt kunnen worden tijdens het uitvoeren zelf, terwijl er bij Pascal zowel tijdens dat vertalen (compileren) als tijdens het uitvoeren (executeren) fouten ontdekt kunnen worden. Bovendien kan er natuurlijk veel meer aan fout-detectie gedaan worden dan bij BASIC, de compiler – zoals het vertaal-programma genoemd wordt – beschouwt de Pascaltekst als geheel en kan allerlei tegenstrijdigheden opmerken die een interpreter – zoals BASIC die gebruikt – over het hoofd ziet.

Een eenmaal gecompileerd programma heeft bovendien het voordeel dat het vele malen sneller uitgevoerd kan worden dan een te interpreteren programma. Vaak ook is het mogelijk om een dergelijk vertaald programma – dikwijls object code of binary genoemd – zodanig op te slaan, dat het de volgende keer niet weer eerst vertaald hoeft te worden.

Natuurlijk is het wel zaak om altijd ook de originele, onvertaalde versie te bewaren. Deze source-code kunnen we namelijk desgewenst nog wijzigen, hetgeen met de vertaalde object-code niet kan.

Afhankelijk van welke Pascal-versie gebruikt wordt kunnen er nóg meer mogelijkheden bestaan. Zo kunnen sommige implementaties met van tevoren gecompileerde bibliotheken van subroutines werken, terwijl anderen het mogelijk maken om tijdens het compileren een stuk source-code vanaf een bestand tussen te voegen. Op die wijze kunnen dan allerlei zelf ontwikkelde standaardprocedures worden ingevoegd. Maar nogmaals, alles hangt af van welke Pascal u gebruikt.

8/3.4

Welke Pascal?

Voor de Commodore 64 zijn er heel wat verschillende versies van de Pascal-programmeertaal gemaakt. Al die versies hebben zo hun sterke en zwakke punten, zo zijn er onder meer C-64 Pascals die de grafische mogelijkheden van de 64 kunnen benutten, tot en met de sprites aan toe!

Echter vaak blijken dergelijke uitbreidingen van de standaard ten kosten te gaan van allerlei andere zaken. Sommige van de complexere data-structuren zijn in zo'n 'grafische Pascal' dan niet aanwezig, of er zijn andere beperkingen vergeleken met de officiële standaard.

Aangezien er dus nogal wat verschillen bestaan tussen de mogelijkheden van die grote verscheidenheid aan keuzemogelijkheden hebben de schrijvers van dit boek toch moeten kiezen voor een bepaalde Pascal. Die keuze is gevallen op Oxford Pascal, aangezien deze in alle opzichten voldoet aan de door de ontwerpers van Pascal vastgelegde eisen.

Oxford Pascal stelt vrij veel eisen qua rand-apparatuur, zo is er een disk-drive nodig om deze compiler te kunnen gebruiken. Maar dat zal bij alle werkelijke volledige implementaties van een ingewikkelde taal als Pascal het geval zijn. De cassette-versie heeft bijgevolg enige beperkingen. Het feit dat de disk-versie werkelijk volledig is heeft de doorslag gegeven; om werkelijk in Pascal te kunnen werken zijn structuren als records (met variant-parts) en sets nu eenmaal noodzakelijk.

Oxford Pascal kan op twee manieren gebruikt worden, namelijk als een volledig in het geheugen aanwezig maar ietwat beperkt Pascal-systeem (de enige mogelijkheid bij de cassette-versie) of als werkelijk volledige compiler, waarbij de eigenlijke programmatekst van schijf gelezen wordt.
De eerste methode heeft als voordeel dat alles veel sneller in zijn werk gaat en is daarom juist zeer geschikt als leersysteem. De volledige maar trage compiler wordt dan alleen gebruikt als men inderdaad een volslagen Pascal nodig heeft.

In de zogenaamde resident mode – als dus de bron-tekst, de compiler, de vertaalde code en het run-time system tegelijkertijd in het geheugen aanwezig zijn – kunnen natuurlijk alleen kleinere programma's geschreven worden. Daar zullen we in eerste instantie echter geen problemen mee hebben, er is meer dan genoeg vrij geheugen voor onze eerste experimenten.

3.4 Welke Pascal?

Dat 'run-time system' overigens is een speciaal programma dat de vertaalde Pascal-tekst uitvoert, min of meer te vergelijken met de normale BASIC-interpreter. Met een groot verschil, want die BASIC-interpreter moet werkelijk ieder BASIC-commando vertalen, terwijl bij Pascal de compiler al een soort tussencode heeft afgeleverd. Die tussen-code – P-code genaamd – lijkt al erg veel op machinetaal en kan dus heel snel worden uitgevoerd.

Voor de mensen die al een beetje met Pascal bekend zijn zetten we hier nog wat eigenschappen van Oxford-Pascal op een rijtje:

MAXINT=32767.
Integer-waardes lopen tussen −32768 en +32767.
CHAR-type beslaat de ASCII tekenset, inclusief de grafische tekens. Inverse tekens kunnen niet worden gebruikt.
SET-waardes liggen tussen de 0 en de 127.
REALs zijn accuraat tot 9 cijfers.

Programma-grootte en complexiteit worden slechts beperkt door het beschikbare geheugen.
Identifiers en labels: de eerste 8 tekens moeten uniek zijn.

Uitbreidingen op standaard-Pascal:

Dynamische bestandsnamen.
String-input.
Hexadecimale getallen en −I/O.
Bit manipulatie.
Machine-taal interface.
Geheugen (ook scherm-) toegang.
Run-time I/O error detectie.
Random number generator.
Program-chaining.
64-klok interface.
Linking van programma's.

Voor de kenner zegt dit lijstje heel wat over de capaciteiten van Oxford Pascal, mocht het bovenstaande echter voor u alleen maar een lijstje met onbegrijpelijke kreten zijn, laat u dan niet meteen afschrikken. Pascal – en dus dit hoofdstuk – is inderdaad behoorlijk ingewikkeld. Maar zeker niet onbegrijpelijk, zoals u zult merken. Wel maken we bij voorbaat onze excuses voor het gebruik van allerlei technische Engelse termen, die – met hun verklaring – regelmatig gebruikt zullen worden. Deze termen vertalen zou echter dit hoofdstuk voor mensen die al verder met Pascal gevorderd zijn weer minder leesbaar maken.

8/3.5

Voorzichtig beginnen

Om Oxford Pascal te kunnen gebruiken heeft u, naast uw Commodore 64, een 1541 diskdrive of 1530 cassette-recorder en natuurlijk Oxford Pascal zelf nodig, dat bestaat uit een diskette/cassette en een handboek.
Pas overigens goed op met de diskette, ze is door de fabrikant voorzien van een kopieer-bescherming, u kunt dus geen werk-kopie maken. Dat houdt in dat u altijd met uw originele master-diskette moet werken en als deze beschadigd raakt kan het wel eens een tijdje duren – en het nodige kosten – om die master te vervangen.
Laat dus het write-protect plakkertje op zijn plaats, gebruik die diskette alleen als systeem-disk. Dat houdt in dat al uw eigen bestanden op uw werk-diskettes staan, zowel de Pascal-sources als de vertaalde object-bestanden. Als u op het beeldscherm gevraagd wordt de data-disk en de system-disk te verwisselen moet u daar altijd gehoor aan geven!

De bijgeleverde handleiding is tamelijk summier en zeker niet geschikt om Pascal uit te leren, maar dat zullen we in de loop van dit hoofdstuk verhelpen.

Om te beginnen zet u de computer, de diskdrive, de monitor of televisie en de eventuele printer aan, steek de system-disk in de 1541 drive en tik:

LOAD"*",8

Er wordt dan een klein lader-programmaatje opgehaald dat met RUN moet worden doorgestart. In het geval van de cassette-versie is de SHIFT/RUN-STOP-toets voldoende.
Het eigenlijke laadproces neemt redelijk wat tijd in beslag, maar na enige tijd verschijnt de Oxford Pascal prompt "ready". Inderdaad, dezelfde prompt als u onder BASIC te zien krijgt. Daar is een goede reden voor, Oxford Pascal laat namelijk het BASIC operating system goeddeels intact.
Dat houdt onder andere in dat sommige BASIC commando-series van kracht blijven. Zo kunt u de inhoud van een disk bekijken met het vertrouwde:

LOAD "$",8
gevolgd door
LIST

Probeer niet de gebruikelijke afkorting voor het LIST commando te gebruiken – l gevolgd door shift i – want dat werkt nu net niet, het leidt tot een foutmelding.

Na het starten van Oxford Pascal be-

3.5 Voorzichtig beginnen

vindt u zich in de zogenaamde editor, waarmee u uw programma's schrijven kan.
Het lijkt allemaal heel sterk op de standaard BASIC-editor, maar desondanks heeft deze editor wel wat meer mogelijkheden.
Zo bevindt u zich bij het opstarten in de AUTO-mode, waarbij de regelnummers automatisch gegenereerd worden. Die regelnummers zijn overigens niet voor Oxford Pascal zelf nodig en u zult er dan ook niet naartoe kunnen springen binnen een programma. Ze worden alleen gebruikt door Oxford Pascal om aan te geven waar zich een mogelijke fout in uw programma bevindt.
De scherm-editor gebruikt die regelnummers overigens wel. Ze zijn bijvoorbeeld te gebruiken om aan te geven welk gedeelte van het programma geLIST moet worden.
In de scherm-editor kunnen alle bekende toetsen van de BASIC-editor gebruikt worden. Met de cursor-toetsen kan over het scherm heen en weer gegaan worden, de DEL en INST toetsen werken zoals u dat gewend bent.

Voor we nu een eerste Pascal probeerseltje gaan intikken moeten we weten hoe zo'n Pascal-programma eruit dient te zien. Dat is namelijk aan strenge regels gebonden, veel strikter dan in BASIC.

Zo dient het eigenlijke programma zelf – dus de regels waarbinnen we opdragen wat er moet gebeuren – tussen de twee speciale woorden begin en end te staan. Alles wat tussen een begin en een end staat wordt door de compiler als een blok beschouwd en Pascal programma's bestaan nu eenmaal uit blokken.
Na de laatste end van een programma moet weer een punt staan, ten teken dat dit inderdaad het einde is.

Ons eerste programma zou er dan als volgt uit kunnen zien:

```
10 begin
20 write ('Voorzichtig beginnen')
30 end.
```

Daarbij hoeft u slechts de eerste keer een regelnummer in te tikken, nadat u in regel 10 op de RETURN gedrukt heeft verschijnt automatisch de 20 voor de volgende regel waarbij de cursor op de juiste positie geplaats wordt. Om die regelnummers weer te laten ophouden tikt u na de 40 een enkele RETURN.

Dit programma moet nu nog worden vertaald, voor we het kunnen laten werken. Dat doen we met het commando r, dat voor run staat. Na het intikken van die r, alweer gevolgd door RETURN, zal het programma niet alleen gecompileerd worden maar daarna meteen uitgevoerd. Tenminste, als de compiler geen fouten gevonden heeft. Als alles in orde is ziet u na die r:

```
Compiling
Program xxxx yyyy
0 error(s)
Compilation complete.
Voorzichtig beginnen
```

De bovenste vier regels van dit verhaal zijn afkomstig van Oxford Pascal, de laatste is het resultaat van ons programmaatje. De letters xxxx en yyyy achter Program zullen door getallen vervan-

3.5 Voorzichtig beginnen

gen zijn, op de betekenis daarvan komen we later terug.
Kortom, het systeem meldt eerst dat er gecompileerd wordt, daarna wat er gecompileerd wordt, het aantal eventuele fouten dat tijdens die compilatie gevonden is en tenslotte dat de compilatie beëindigd is. Onmiddelijk daarna wordt ons programmaatje gestart.

Het zou natuurlijk ook kunnen gebeuren dat u een foutje gemaakt heeft bij het intikken. In dat geval zult u allerlei foutmeldingen te zien kunnen krijgen. Kijk in dat geval het voorbeeldje nog eens zorgvuldig na, want ergens wijkt het af van het programma in dit boek. Als u de fout gevonden heeft kunt u deze verbeteren en het nog eens proberen.

Als het compileren de eerste keer echter foutloos was kunt u uw programma natuurlijk nog eens laten uitvoeren, door weer r in te tikken. Alleen, dit maal hoeft het niet meer gecompileerd te worden, dat is al gebeurd. Oxford Pascal houdt zelf bij of er al dan niet gecompileerd moet worden, zolang u het programma niet verandert zal het nu na een r meteen worden uitgevoerd.

Het write-commando in Pascal doet dus blijkbaar hetzelfde als een PRINT in BASIC. Er zijn weliswaar wat kleine verschillen in de manier waarop de tekst achter dat write commando staat vergeleken met PRINT, maar het uiteindelijke effect is hetzelfde.
In Pascal staat een string tussen enkele aanhalingstekens in plaats van dubbele, terwijl iets wat we aan een commando meegeven altijd tussen ronde haakjes moet staan. Of, om de juiste Pascal-termen te gebruiken, het argument 'Voorzichtig beginnen' van de procedure write moet tussen ronde haakjes staan.

Laten we ons programma eens uitbreiden. Voeg eens een regel tussen:

25 write('Dan gaat het goed')

en pas regel 20 aan tot:

20 write('Voorzichtig beginnen');

Als u nu weer een r intikt, dan zal het programma wel weer gecompileerd moeten worden waarna de tekst:

Voorzichtig beginnenDan gaat het goed

verschijnt.
Hetgeen toch niet is wat we zouden verwachten. De beide teksten blijken op dezelfde regel te staan, zonder enige scheiding.

Dat is dan ook een van de eigenschappen van de write-procedure, als we een string – of andere zaken, write is namelijk heel krachtig – met write op het beeldscherm zetten zal dat niet automatisch tot één item per regel beperkt worden. Pas als een regel vol is gaat write door op een nieuwe schermregel, hetgeen ook eigenlijk logisch is.
Als we die nieuwe regel willen hebben moeten we dat specifiek opgeven, door een andere procedure aan te roepen. Naast write bestaat namelijk writeln, waarbij de letters ln voor line (regel) staan. Als we regel 20 wijzigen in:

20 writeln('Voorzichtig beginnen');

3.5 Voorzichtig beginnen

krijgen we wel het verwachte effect.

De puntkomma achter het argument op regel 20 heeft óók een hele specifieke betekenis, het is de zogenaamde statement-scheider. In BASIC doen we dat met een dubbele punt, terwijl het einde van een regel in BASIC automatisch als scheider gezien wordt. Maar in Pascal is alles wat logischer van opzet, er zijn geen automatismen zoals "regeleinde is commando-scheider". We moeten die puntkomma dan ook altijd zetten, ook al staat het statement aan het einde van een regel. De enige uitzondering daarop is voor een end, die in feite een speciale vorm van statement-scheider is.

De term "statement" behoeft ook nog wel enige uitleg. Pascal bestaat namelijk uit een antaal gereserveerde woorden, ieder met hun eigen betekenis. Sommige daarvan zijn procedures zoals write, andere hebben weer andere functies. Uit die woorden bouwen we de statements op, die uit één of meer procedures en andere dingen kunnen bestaan. In ons voorbeeldje hebben we slechts twee enkelvoudige statements, maar het is ook mogelijk om samengestelde statements op te bouwen, zoals we nog zullen zien.

Toch is ons programma nog niet echt kompleet. We hebben nu de "body" van het programma – dat tussen begin en end staat – maar er moet eigenlijk nog een kop boven. In Oxford Pascal is dat weliswaar niet verplicht, maar in vele andere versies wel. Bovendien kunnen we aan zo'n kop, een "program heading" zoals het officieel heet, altijd zien met welk programma we van doen hebben.
Als u toevoegt:

5 program proefeen;

is ons probeerseltje echt compleet. Merk op dat ook hier de puntkomma weer als scheider gebruikt wordt, terwijl het ook al geen toeval is dat de naam precies acht letters lang is.
Elk naam moet namelijk in Pascal ook aan regels voldoen, eentje daarvan is dat de eerste 8 tekens worden gebruikt om die naam van een andere naam te onderschieden. Dat gaat op voor alle namen die we zelf bedenken in een Pascal-programma, ook voor variabele-namen bijvoorbeeld. Er is dan ook weer een officiële term voor, namelijk "identifier".

Tot slot van deze eerste voorzichtige kennismaking met Oxford Pascal is het nog belangrijk om te weten hoe eigen programma's op disk bewaard kunnen worden. Gelukkig gaat dat vergeleken met BASIC wel heel simpel; met:

put proefeen

wordt de eerste poging netjes weg geschreven. Er hoeven geen aanhalingstekens gebruikt te worden noch een apparaatnummer op gegeven te worden; Oxford Pascal gaat er vanuit dat de diskdrive gebruikt wordt.

Terugladen gaat even eenvoudig, met:

get proefeen

kan het programma weer terug gehaald worden.

8/3.6

Rekenen in Pascal

Zoals reeds gesteld, Pascal is iets volkomen anders dan BASIC. Dat blijkt bijvoorbeeld al gauw als we wat simpele berekeningen gaan uitvoeren.

Dat rekenen kan overigens op heel wat manieren, zowel in BASIC als in Pascal. Laten we met een simpel voorbeeld beginnen, waarbij we zogenaamde constantes gebruiken om mee te rekenen. De cijfers waar we mee werken staan dus rechtstreeks in het programma en worden niet eerst in variabelen opgeslagen. In BASIC maakt dat niet zoveel uit, in Pascal des te meer zoals we zullen merken.

Om een BASIC-programmaatje zoals:

10 PRINT 10/2

naar Pascal te vertalen is echter nog eenvoudig genoeg. Weliswaar wordt die Pascal-versie wat langer, omdat we in Pascal nu eenmaal wat netter met een programma omgaan, maar uiteindelijk lijkt het er toch wel heel sterk op.
Kijk maar:

```
1000 program reken1;
1010 begin
1020   write (10/2)
1030 end.

READY.
```

Die ene BASIC-regel is maar liefst vier regels Pascal geworden, maar dat komt omdat we er netjes een programmanaam boven hebben staan en we er bovendien de verplichte begin- en end-statements omheen hebben gezet. Zonder die twee regels, 1010 en 1030, zou de compiler de typische blok-structuur van Pascal niet gevonden hebben.

Bovendien hebben we in deze listing een vaak gebruikte Pascal-gewoonte toegepast; we laten de blokstructuur extra duidelijk uitkomen door een paar spaties in te springen. De begin- en end-commando's bevinden zich op het bovenste (of beter uitgedrukt: buitenste) niveau, terwijl regel 1020 een niveau dieper ligt. Door per niveau één of meer spaties in te springen kan de structuur verduidelijkt worden, wat vooral bij langere programma's de leesbaarheid sterk verhoogt.

Tenslotte ziet u in dit voorbeeld de gebruikelijke regelnummering van Oxford-Pascal, die afwijkt van wat we voor ons allereerste probeersel gebruikt hebben. Immers, in Pascal zijn de regelnummers alleen voor het editen van belang, intern gebruikt het programma ze niet. Ze worden dan ook niet opgeslagen als een programma met het PUT-commando weggeschreven wordt.

3.6 Rekenen in Pascal

Bij het teruglezen met GET worden die regelnummers opnieuw gegenereerd, te beginnen bij 1000 en met een stapgrootte van 10. Om allerlei potentiële problemen te vermijden is het sterk aan te raden om die beide waardes ook zelf altijd te gebruiken. Want anders zouden we met een te verwijderen regelnummer er wel eens naast kunnen zitten en per ongeluk de verkeerde regel weggooien.

Kortom, om dit voorbeeldje in te tikken moeten we zelf het eerste regelnummer, 1000, invoeren. Daarna zal Pascal automatisch de volgende regelnummers (1010, 1020 etc) op het scherm zetten, net zo lang tot we klaar zijn. Op dat moment is een enkele RETURN, zonder verdere tekst, genoeg om weer te stoppen met invoeren.
Als we dit mini-programma eens proberen – door run of alleen de letter r in te tikken – valt ons meteen nog een bijzonderheid op. De uitkomst verschijnt namelijk in een wat ongebruikelijk formaat. Een BASIC-programma zou als resultaat een simpele:

```
1000 program reken2;
1010 var
1020 noemer,
1030 teller,
1040 uitkomst : real;
1050 begin
1060   teller := 10;
1070   noemer := 2;
1080   uitkomst := teller/noemer;
1090   write (uitkomst)
1100 end.

READY.
```

5

afdrukken, maar nu zien we:

5.00000E+00

In feite staat daar gewoon het getal 5, maar dan in de zogenaamde wetenschappelijke notatie. Het getal na de letter E is een macht van 10, waarmee we het getal voor die E moeten vermenigvuldigen.
In dit geval wordt dat dan:

5.00000 maal (10 tot de macht 0)

Omdat 10 tot de macht 0 gelijk aan 1 is staat er een simpele 5.

Het grote voordeel van de schrijfwijze is dat er zonder problemen zowel hele grote als hele kleine waardes mee uitgedrukt kunnen worden, zonder dat dit tot lange reeksen leidt. Later komen we nog terug op deze schrijfwijze, die overigens desgewenst ook heel makkelijk omzeild kan worden. Voor het moment gaan we eerst kijken naar een tweede rekenvoorbeeld.

3.6 Rekenen in Pascal

Feitelijk doet dit programma precies hetzelfde als het vorige voorbeeld, maar nu wordt er niet met constantes maar juist met variabelen gerekend.

In BASIC zou dit heel wat korter zijn:

```
10 A=10
20 B=20
30 C=A/B
40 PRINT C
```

Het voornaamste verschil tussen de BASIC- en de Pascal-versie vloeit voort uit het feit dat we in Pascal verplicht zijn iedere variabele te definiëren voor we deze kunnen gebruiken. Met andere woorden, we moeten de compiler vertellen van welk type een bepaalde variabele is. Als we dat nalaten komt dat onherroepelijk op een foutmelding te staan.

In dit geval gebruiken we drie variabelen, te weten teller, noemer en uitkomst, die alle drie van het type real zijn. In gewoon Nederlands betekent dat breukgetallen.
Zo'n real hoeft geen cijfers achter de komma te bevatten, maar het kan wel. Dit in tegenstelling tot het integer-type, dat alleen maar gehele getallen kan opslaan.

Dat opgeven van de variabelen vindt plaats in de regels 1010 tot en met 1040. In Pascal-termen heet dit het Variable-declaration-part, wat in bijna geen enkel programma zal ontbreken.
Pascal stelt strenge eisen aan de manier waarop we het variable-declaration-part ogpeven.
Het sleutelwoord is VAR, daarmee wordt het variable-declaration-part ingeleid. Daarna dienen per type – in dit geval alleen maar REAL – de namen opgegeven te worden, gescheiden door komma's. Na die namenlijst volgt het eigenlijke type, dat door een dubbele punt van de namen gescheiden wordt. De punt-komma tenslotte wordt gebruikt om de declaraties van de verschillende types van elkaar te onderscheiden.

Maar binnen die eisen kunnen we toch weer voor diverse oplossingen kiezen. Zo zouden de regels 1010-1040 ook als één enkele regel geschreven mogen worden:

```
1010 VAR NOEMER, TELLER, UITKOMST : REAL;
```

De reden waarom we echter iedere naam op een nieuwe regel zetten is alweer: leesbaarheid. Het kost wat meer tikwerk en ruimte, maar is veel makkelijker terug te lezen.

De eigenlijke namen hebben een echte betekenis, noemer, teller en uitkomst staan precies voor wat ze zijn. In een BASIC-programma zouden die namen weliswaar ook gebruikt kunnen worden, maar daar tellen slechts de eerste twee tekens mee om de variabelen van elkaar te onderscheiden. De namen noemer en noemer1 staan in een BASIC-programma voor dezelfde variabele. In een Pascal-programma zijn het echter wel degelijk verschillende variabelen, pas bij erg lange namen – bijvoorbeeld uitkomst1 en uitkomst2 – treden er problemen op.

Na het variable-declaration-part volgt het eigenlijke programma, dat weer

3.6 Rekenen in Pascal

netjes van begin- en end-haken voorzien is en bovendien inspringt om de leesbaarheid weer te verhogen.
Ieder statement staat keurig op een eigen regel, waarbij ze onderling door punt-komma's gescheiden zijn. Alleen na het laatste statement in regel 1090 hoeft die punt-komma niet aanwezig te zijn, want een end is zelf een statementscheider.
Overigens maken de meeste Pascal-programmeurs zich daar niet druk over, ze zetten altijd een punt-komma, ook als die – zoals in regel 1090 – niet vereist is. Het levert geen fouten op en is wel zo veilig.

Wat opvalt in ons voorbeeldje is het verschil tussen BASIC en Pascal in de wijze waarop een waarde aan een variabele wordt toegekend. In BASIC is dat officieel:

LET variabele = waarde

Meestal echter laat men het – niet vereiste – LET-commando weg, zodat een toewijzing er als volgt uit ziet:

variabele = waarde

Met andere woorden, het is-gelijk teken staat in BASIC voor de toewijzing van een waarde – of de uitkomst van een berekening – aan een variabele. Datzelfde is-gelijk teken heeft echter nog een tweede betekenis in BASIC: het wordt ook voor vergelijkingen gebruikt. In een regel als:

IF A$=B$ THEN .....

wordt A$ helemaal niet gelijk gemaakt aan B$, er wordt alleen gekeken of ze gelijk zijn.
BASIC gebruikt de is-gelijk op een wat onduidelijke manier, er kunnen twee totaal verschillende zaken mee bedoeld worden. Vooral beginnende programmeurs liggen daar nog wel eens mee overhoop, terwijl gevorderden er vaak nogal onleesbare trucs mee uithalen. Een regel als:

A=B=1

is weliswaar toegestaan in BASIC, maar bevordert de leesbaarheid niet echt.

Bij het ontwerpen van Pascal zijn dergelijke dubbelzinnigheden juiste koste wat het kost vermeden; de is-gelijk wordt in Pascal alleen maar voor het vergelijken gebruikt. Om een waarde toe te wijzen aan een variabele gebruikt Pascal een combinatie van een dubbele punt en een is-gelijk.

Gehele getallen

We hebben het al kort gehad over de verschillende types variabelen in Pascal. Een van de eigenschappen van de taal is dat die types strikt gescheiden zijn, Pascal is een 'typecasting language'. Dat betekent alleen maar dat de taal steeds bijhoudt van welk type een variabele of de uitkomst van een berekening is en het verboden is om zonder meer verschillende types door elkaar heen te gebruiken.

Als we ons voorbeeld eens herschrijven, dit keer voor gehele getallen – integers – moeten we daar ook rekening mee houden.

3.6 Rekenen in Pascal

```
1000 program reken3;
1010 var
1020 noemer,
1030 teller,
1040 uitkomst : integer;
1050 begin
1060   teller := 10;
1070   noemer := 2;
1080   uitkomst := teller div noemer;
1090   write (uitkomst)
1100 end.

READY.
```

Niet alleen het variabele-type in regel 1040 is veranderd, ook de eigenlijke deling in regel 1080 ziet er nu anders uit, er wordt een andere operatie gebruikt. Pascal kent namelijk twee deel-opdrachten, de / en de DIV.

De gewone / is een standaard deling, zoals we die in BASIC ook gewend zijn. De uitkomst ervan is altijd een real, een gebroken getal. Zelfs al staan er geen cijfers achter de komma, dan nog zal Pascal de uitkomst als real aanmerken – er zou immers best een breukwaarde uit kunnen komen – en is er een real-variabele vereist om die uitkomst in op te slaan.
Natuurlijk mogen we zowel integers als reals gebruiken voor zowel de teller als de noemer. In alle gevallen zal de uitkomst echter een real zijn.

De DIV-operatie is echter een geheel ander soort deling, die 'geheeld delen' genoemd zou kunnen worden. Een DIV is in feite een deling waarbij de eventuele cijfers achter de komma weggegooid worden. Zowel de teller als de noemer moeten integers zijn, de uitkomst is dat ook.

Een paar voorbeelden:

10 DIV 2 geeft 5
10 DIV 3 geeft 3
10 DIV 4 geeft 2

Naast de DIV kent Pascal ook een MOD-operatie. Deze levert juist de rest op van de deling van twee integers.
Voorbeelden:

10 MOD 2 geeft 0
10 MOD 3 geeft 1
10 MOD 4 geeft 2

Later zullen we nog uitgebreid terugkomen op de verschillende rekenkundige operaties en hun gebruik bij de diverse variabelentypes.

8/3.7

In- en uitvoer

Tot nog toe hebben we alleen hele simpele berekeningen uitgevoerd, waarbij we weliswaar de resultaten te zien kregen maar de eigenlijke getallen vast in het programma verankerd lagen.
Maar Pascal kan die getallen natuurlijk ook via het toetsenbord inlezen. Dat gaat zelfs heel simpel:

```
1000 program reken4;
1010 var
1020 noemer,
1030 teller,
1040 uitkomst : integer;
1050 begin
1060   read (teller);
1070   read (noemer);
1080   uitkomst := teller div noemer;
1090   write (uitkomst)
1100 end.

READY.
```

De veranderingen zitten in de regels 1060 en 1070, waar we het read-commando tegenkomen. Deze keer worden de waardes van de teller en de noemer niet zonder meer in het programma toegewezen, maar ingelezen.
Nadat we r – of voluit run – ingetikt hebben verwacht het programma invoer van het toetsenbord. Dat blijkt uit het feit dat we weliswaar de cursor zien, maar niet de normale 'ready'. Tik maar eens twee getallen in, prompt verschijnt de uitkomst, een DIV-deling.

Als u wat experimenteert met dit programma zult u verbaasd staan. Zo blijkt de invoer nogal wat toe te staan, u kunt desgewenst ook getallen met daarin een decimale punt en cijfers daarachter invoeren, zonder dat dit meteen tot een foutmelding aanleiding geeft. Zolang u als tweede cijfer maar geen waarde kleiner dan 1 opgeeft lijkt alles te functioneren.

Ook mag zelf gekozen worden of beide getallen op dezelfde regel gezet worden, met bijvoorbeeld een spatie ertussen om ze van elkaar te scheiden, of dat na ieder getal op de RETURN gedrukt

3.7 In- en uitvoer

wordt. Kortom, er moet heel wat gedaan worden voor er een foutmelding verschijnt. Soms echter – bijvoorbeeld als de tweede waarde een real tussen −1 en +1 is – breekt het programma af met de melding dat u iets door nul probeerde te delen. En nog minder vaak zal de melding:
Integer read error
verschijnen

Toch vallen al die ogenschijnlijke tegenstrijdigheden best te verklaren. Het Pascal read-commando heeft namelijk een aantal hele sterke eigenschappen.
Zo zal het in principe verkeerde invoer gewoon overslaan, waarbij de invoer die wel geldig is gebruikt wordt.
Aangezien we waardes in integer-variabelen lezen, zal dat read-commando alle niet in een integer thuishorende tekens overslaan.
Dat houdt bijvoorbeeld in dat een getal als 34.56 wordt ingelezen als 34, de rest – .56 – wordt gewoon weggegooid.

Bovendien gaat read er niet van uit – zoals we van BASIC gewend zijn – dat een invoer alleen door een RETURN beëindigd kan worden. De RETURN geeft weliswaar het signaal 'dit was het, verder komt er even geen invoer meer', maar een read-commando verwerkt niet die hele regel.
Read – eigenlijk een procedure, niet een commando – leest die ingevoerde regel net zo lang tot er een scheider in staat. Met andere woorden, de read in regel 1060 leest de invoer tot er een niet-geldig teken – geen cijfer – wordt gevonden en gebruikt het tot dan toe gelezen getal als integer. De tweede read begint waar de eerste opgehouden is en leest net zolang tot er een geldige integer gevonden is.

In de praktijk houdt dit in dat we twee manieren tot onze beschikking hebben om de beide integers in te tikken; namelijk op één regel, gescheiden door een spatie, of op twee verschillende regels. De spatie en de RETURN zijn de zogenaamde scheiders.

Zoals reeds gezegd, eigenlijk is read, net als write en writeln, een procedure. Zo'n procedure kan gezien worden als een kant-en-klaar stukje programma, dat in grote lijnen op zichzelf staat. In een van de volgende voorbeelden zullen we zelf procedures gaan schrijven, om bepaalde zaken af te handelen.
Maar deze standaard-procedures zijn voorgedefinieerd, ze hoeven niet zelf meer geschreven te worden. Ze maken echter geen deel uit van de basisdefinitie van Pascal, hoewel ze in iedere Pascal ingebouwd zijn. De reden daarvoor is dat deze procedures verregaand afwijken van wat er in Pascal gebruikelijk is.

We weten al dat Pascal er naar streeft om voortdurend bij te houden of de programmeur geen fouten maakt met het gebruik van de verschillende soorten variabelen. Ook procedures kunnen met variabelen werken, in het geval van ons voorbeeld zijn noemer, teller en uitkomst parameters voor de procedures read en writeln.
Normaal gesproken moeten we echter in de definitie van een procedure opgeven hoeveel parameters er worden meegegeven en van welke types deze zijn. Read, write en writeln zijn daar uitzonderingen op, pas tijdens het uitvoeren van de read-procedure wordt duidelijk of er nu een real of een integer gelezen moet worden. Of meerdere van beide

3.7 In- en uitvoer

types door elkaar, dat is ook nog mogelijk.

Dat verklaart dan ook waarom Pascal – dat in feite een hele strikte taal is – tijdens de invoer opeens zo soepel blijkt te zijn. De standaard-procedures wijken af van de normale benadering.

Behalve de read-procedure bestaat er ook een readln, die iets anders werkt. Aan de hand van het volgende voorbeeld zullen we dat eens onderzoeken.

```
1000 program reken5;
1010 var
1020 noemer,
1030 teller,
1040 uitkomst : integer;
1050 begin
1060   readln (teller);
1070   readln (noemer);
1080   uitkomst := teller div noemer;
1090   write (uitkomst)
1100 end.

READY.
```

De enige verschillen vinden we in de regels 1060 en 1070, waar deze keer readln wordt gebruikt in plaats van read. In de praktijk blijkt dat het programma nog steeds even vergevingsgezind reageert als we per ongeluk een real invoeren, alleen de truc van twee gegevens invoeren op dezelfde regel werkt niet meer.
Readln leest namelijk de regel tot de variabele gevuld is en slaat daarna de rest over.

Laten we het voorbeeld nog eens wat verder uitbreiden en er wat hulpteksten in aanbrengen. Het is immers veel duidelijker wat er gebeurt als het programma ons zelf vertelt wat we moeten doen.

```
1000 program reken6;
1010 var
1020 noemer,
1030 teller,
1040 uitkomst : integer;
1050 begin
1060   write ('Geef waarde:');
1070   readln (teller);
1080   write ('Geef waarde:');
1090   readln (noemer);
1100   uitkomst := teller div noemer;
1110   write ('Uitkomst is:');
```

3.7 In- en uitvoer

```
1120   write (uitkomst)
1130 end.

READY.
```

Met drie extra regels is één en ander geregeld. Nu zegt het programma wanneer er invoer verwacht wordt en wat de uitkomst is. Let er daarbij op dat voor die teksten de write-procedure gebruikt wordt. Writeln zou tot een ongewenste regelopvoer leiden.

8/3.8

Procedures, functions en lussen

Als we eens goed kijken naar de laatste versie van het rekenprogramma dan zien we daar een aantal bijna gelijke regels. Regels 1060 en 1070 lijken als twee druppels water op 1080 en 1090, alleen de naam van de in te lezen variabele verschilt.
In BASIC zouden we er nu over kunnen gaan denken om er een subroutine van te maken, maar dat zou eigenlijk nog een probleem zijn. We kunnen dan namelijk wel de tekst afdrukken en een waarde inlezen, maar moeten daarna toch nog die beide waardes in hun eigen variabelen plaatsen. Dat zou er bijvoorbeeld zo uit kunnen zien:

```
...
140 GOSUB 200: A=D
150 GOSUB 200: B=D
...
200 PRINT "Geef waarde";
210 INPUT D
220 RETURN
```

Pascal echter toont nu zijn kracht, zoals het volgende voorbeeld laat zien.

```
1000 program reken7;
1010 var
1020 noemer,
1030 teller,
1040 uitkomst: integer;
1050 PROCEDURE LEESIN (VAR WAARDE: INTEGER);
1060 begin
1070   WRITE ('Geef waarde:');
1080   READLN (WAARDE);
1090 end;
1100 begin
1110   LEESIN (TELLER);
1120   LEESIN (NOEMER);
1130   uitkomst := teller div noemer;
1140   WRITE ('Uitkomst is:');
1150   write (uitkomst)
1160 end.

READY.
```

3.8 Procedures, functions en lussen

Nieuw is het gedeelte tussen regel 1050 en 1090, waar de procedure leesin wordt gedefinieerd.
Zo'n procedure komt altijd na de var-declaraties en voor het eigenlijke programma, want in Pascal moet alles al bekend zijn voor het gebruikt wordt. Eerst leesin gebruiken, zoals in de regels 1110 en 1120 gedaan wordt en dan pas definiëren leidt tot een foutmelding.

Als we eens kijken naar die definitie van leesin, dan zien we dat zo'n procedure eigenlijk veel op een gewoon Pascal-programma lijkt. Zo'n procedure-blok bestaat in feite uit dezelfde onderdelen als een volledig programma, eerst een kop, daarna wat andere losse zaken – die in dit voorbeeld nog ontbreken – en tenslotte de eigenlijke programma-regels, die door begin- en end-statements omsloten worden.
In dit geval staat er echter geen punt achter de laatste end, dat is alleen aan het einde van het totale programma.

In de procedure-kop staat niet alleen de naam, leesin, maar ook wat er ingelezen moet worden. In dit geval is dat:

(var waarde: integer);

Op de betekenis van dat var komen we nog terug, voor het moment is het belangrijker dat we niet alleen een naam – een identifier – maar ook een type meegeven.
Die naam is niet eerder als variabele gedeclareerd, maar dat hoeft ook niet. Hij wordt alleen gebruikt om de actuele waarde door te geven.
Met andere woorden, waarde bestaat niet echt. Op het moment dat leesin wordt aangeroepen, in de regels 1110 en 1120, wordt de formele parameter waarde vervangen door een van de actuele variabelen noemer of teller. Dat zijn de variabelen waarin uiteindelijk de door leesin gelezen waardes terechtkomen.
Kortom, waarde is slechts een hulpnaam, er bestaat geen variabele van die naam binnen het programma. Het type dat we opgeven wordt echter wel degelijk gecontroleerd; als we zouden proberen leesin aan te roepen met een real-variabele als parameter gaat dat gegarandeerd mis.

We hebben nu met één van de sterkere eigenschappen van Pascal kennis gemaakt, namelijk het parameter-mechanisme. Afhankelijk van de parameter die we meegeven bij de aanroep van procedure leesin komt de ingelezen waarde in een andere variabele terecht, hetgeen zo'n procedure bijzonder flexibel maakt.
De volgende versie van ons deel-programma kent weer wat extra's. Niet alleen gebruiken we nu Pascal commentaren, waarmee de listing nog wat leesbaarder maken, maar bovendien gaan we de waarde van de noemer controleren. Immers, een nul invoeren als tweede waarde levert nog steeds fouten op en een goede programmeur ondervangt dergelijke potentiële problemen.

3.8 Procedures, functions en lussen

```
1000 program reken8;
1010 (*                                    *)
1020 var
1030 noemer,
1040 teller,
1050 uitkomst: integer;
1060 (*                                    *)
1070 procedure leesin (var waarde: integer);
1080 begin
1090   write ('Geef waarde:');
1100   readln (waarde);
1110 end; (* PROCEDURE LEESIN *)
1120 (*                                    *)
1130 begin (* HOOFDPROGRAMMA *)
1140   leesin (teller);
1150   repeat
1160     leesin (noemer);
1170   until noemer <> 0;
1180   uitkomst := teller div noemer;
1190   write ('Uitkomst is:');
1200   write (uitkomst)
1210 end.

READY.
```

Met behulp van de Pascal commentaar-codes – (* begint een commentaar, *) geeft het einde aan – hebben we wat witregels tussengevoegd in de listing. Het is een goede praktijk om de diverse blokken van een programma op zo'n manier van elkaar te scheiden.
Bovendien geven we aan dat de end op regel 1110 slaat op de procedure leesin, wat met zo'n korte procedure eigenlijk nog niet echt nodig is. Bij langere procedures echter kan zoiets de leesbaarheid aanzienlijk bevorderen.

Veel belangrijker echter is de constructie in de regels 1150 tot en met 1170. Hier maken we voor het eerst kennis met een van de lussen die Pascal kent, de repeat until lus.

Alle statements tussen de woorden repeat en until worden namelijk herhaald, tot er aan de voorwaarde achter until voldaan is. In dit geval is die voorwaarde:
noemer $<>$ 0
en tot dat zo is zal de procedure leesin(noemer) worden uitgevoerd. Met andere woorden, het statement op regel 1160 – leesin(noemer) – wordt net zolang herhaald tot het een waarde in de variabele noemer oplevert die niet gelijk aan nul is.

In BASIC zou zo iets alleen met behulp van een IF ... THEN constructie mogelijk zijn, waarbij er na de THEN moet worden teruggesprongen naar een regelnummer. De Pascal-oplossing is heel wat fraaier.

3.8 Procedures, functions en lussen

We zullen de repeat until nog veel vaker tegenkomen, samen met allerlei andere lus-constructies in Pascal. Elk daarvan heeft zo zijn eigen eigenschappen. Zo zal een repeat until de statements tussen de sleutelwoorden repeat en until tenminste één maal uitvoeren, alvorens de voorwaarde achter de until te evalueren.

Behalve procedures, sub-programma's die onder andere variabelen kunnen wijzigen zoals leesin dat doet, kent Pascal ook functions, een soort sub-programma's die rechtstreeks waardes afleveren. In BASIC kennen we weliswaar ook wel functies, zoals SIN en COS, maar de mogelijkheid om zelf functies te definiëren is in Commodore64 BASIC nogal beperkt.
Pascal-functions mogen echter zo complex worden als maar nodig is, een Pascal-function kan een blok van vele regels lengte zijn. Zo'n function kan zelfs parameters meekrijgen, die net als bij de procedures pas bij de aanroep ingevuld hoeven te worden.
Natuurlijk moet ook het type van de functie – levert de functie een real, een integer of nog iets anders af – worden gedeclareerd. Anders zou het typecasting systeem, dat voortdurend controleert of we geen fouten maken met de verschillende types, niet kunnen werken.

In het volgende voorbeeld wordt zo'n function gebruikt om te bepalen of de waarde van de variabele teller inderdaad ongelijk aan nul is. Daarbij zien we meteen een nieuw type, de zogenaamde boolean.
Een boolean-variabele (of -function) kan slechts twee waardes aannemen, waar of onwaar. In het Engels true of false. Booleans worden bij uitstek gebruikt bij logische voorwaardes, zoals we die ook uit BASIC kennen.

Zo zal de expressie:
2=3
de boolse waarde false opleveren, immers, 2 is niet gelijk aan 3.
Nog een voorbeeld, nu met tekst-constantes:
'TEST'< >'EXPERIMENT'
levert als resultaat true op, de beide constantes zijn inderdaad niet hetzelfde.
Het begrip boolean lijkt ingewikkelder dan het is, ook de BASIC-programmeur werkt er voortdurend mee. Alleen, in BASIC zijn de uitkomsten van vergelijkingen niet true of false maar getallen, hoewel we daar meestal niets van merken. De BASIC-regel

IF A=B THEN PRINT "A is gelijk aan B"

test in feite of de uitdrukking A=B wel waar, true, is.

In de praktijk ziet zoiets er als volgt uit is Pascal:

```
1000 program reken9;
1010 (*                                    *)
1020 var
1030 noemer,
1040 teller,
1050 uitkomst: integer;
```

3.8 Procedures, functions en lussen

```
1060 (*                                  *)
1070 procedure leesin (var waarde: integer);
1080 begin
1090   write ('Geef waarde:');
1100   readln (waarde);
1110 end; (* PROCEDURE LEESIN *)
1120 (*                                  *)
1130 function nietnul (waarde: integer): boolean;
1140 begin
1150   nietnul := waarde <> 0
1160 end; (* FUNCTION NIETNUL *)
1170 (*                                  *)
1180 begin (* HOOFDPROGRAMMA *)
1190   leesin (teller);
1200   repeat
1210     leesin (noemer);
1220     if not nietnul (noemer) then writeln
     ('Noemer moet ongelijk nul zijn')
1230   until nietnul (noemer);
1240   uitkomst := teller div noemer;
1250   write ('Uitkomst is:');
1260   write (uitkomst)
1270 end.

READY.
```

Nieuw zijn slechts de regels 1130 tot en met 1160, waar de function nietnul gedefinieerd wordt, alsmede regel 1220, waar deze function gebruikt wordt om eventueel een foutmelding te geven. Bovendien is de rechtstreekse logische voorwaarde in regel 1230 eveneens vervangen door een tweede aanroep van nietnul.
In 1220 komen we de eerste toepassing van de if ... then constructie tegen, die voor BASIC-programmeurs echter volkomen bekend zal zijn.
Het hart van de function nietnul vinden we in regel 1150, die meteen een duidelijk voorbeeld geeft van het verschil tussen de Pascal assignatie (waarde-toekenning) := en de Pascal vergelijking =.
In woorden uitgedrukt staat er:
maak de waarde van de functie nietnul gelijk aan de uitkomst van de vergelijking waarde < >0.
Als de parameter waarde inderdaad ongelijk aan nul is, dan levert de vergelijking – en dus de function – de waarde true af, anders is nietnul false.

Mogelijk heeft u al opgemerkt dat er in de kop van nietnul een wat andere manier gebruikt wordt om de parameter op te geven; het woord var ontbreekt namelijk. De achtergrond hiervan komt echter pas later aan bod, voor het moment geldt dat u deze regel letterlijk moet overnemen.

We kunnen nog een laatste verfraaiing aanbrengen in ons voorbeeld. Tot nog

3.8 Procedures, functions en lussen

toe is het zo dat het programma iedere keer slechts één keer wordt uitgevoerd, om het een tweede keer te starten moeten we weer run of r intikken. Dat kan beter natuurlijk.

```
1000 program reken10;
1010 (*                                *)
1020 var
1030 noemer,
1040 teller,
1050 uitkomst: integer;
1060 (*                                *)
1070 procedure leesin (var waarde: integer);
1080 begin
1090  write ('Geef waarde:');
1100  readln (waarde);
1110 end; (* PROCEDURE LEESIN *)
1120 (*                                *)
1130 function nietnul (waarde: integer): boolean;
1140 begin
1150  nietnul := waarde <> 0
1160 end; (* FUNCTION NIETNUL *)
1170 (*                                *)
1180 function sommaken: boolean;
1190 var
1200 antwoord : char;
1210 begin
1220  writeln; writeln;
1230  write ('Sommetje maken? (j/n)');
1240  repeat
1250   read (antwoord);
1260  until (antwoord = 'j') or (antwoord = 'n');
1270  sommaken := antwoord ='j'
1280 end; (* FUNCTION SOMMAKEN *)
1290 (*                                *)
1300 begin (* HOOFDPROGRAMMA *)
1310  while sommaken do
1320   begin
1330    leesin (teller);
1340    repeat
1350     leesin (noemer);
1360     if not nietnul (noemer) then writeln
('Noemer moet ongelijk nul zijn');
1370    until nietnul (noemer);
1380    uitkomst := teller div noemer;
1390    write ('Uitkomst is:');
1400    write (uitkomst)
1410   end
1420 end.

READY.
```

3.8 Procedures, functions en lussen

Als we dit programma eens bestuderen blijkt dat er nog een blok bijgekomen is, namelijk de function sommaken.
Hoewel deze function, die van het type boolean is, geen parameters kent is ze toch behoorlijk groot. Sommaken is bijna een programma op zichzelf, compleet met een eigen locale variabele van een nieuw type, die in de regels 1190 en 1200 gedefinieerd wordt. Waarom dat is komt straks aan de orde, laten we eerst sommaken zelf eens onder de loep nemen.

De variabele antwoord is van het type char, letterteken. Een char is een typisch Pascal-begrip, het is een enkel teken. In BASIC vinden we niets vergelijkbaars, of het zou een string van slechts een teken lang moeten zijn.
Maar het kenmerkende van het type char is juist dat zo'n variabele nooit of te nimmer langer dan dat enkele teken kan zijn. Dat maakt zo'n chartype-variabele ideaal voor allerlei zaken, zoals het bevatten van een antwoord van de gebruiker van slechts één letter.

Daar wordt antwoord dan ook voor gebruikt, in regel 1250. Eerst stelt sommaken in regel 1230 de vraag "Sommetje maken? (j/n)", waarna de repeat until lus in de regels 1240 tot en met 1260 het eenletterige antwoord op die vraag inleest en controleert. Zo lang de gebruiker een andere letter dan j of n intikt zal het read(antwoord) statement op regel 1250 herhaald worden.
Tenslotte wordt de waarde van sommaken in regel 1270 op true gezet als het antwoord inderdaad en j was.

Om met behulp van de function sommaken het hoofdprogramma te besturen is weer een andere Pascal lus gebruikt, de

while voorwaarde do statement

Deze heeft als eigenschappen dat ze het statement na het do-woord zal blijven uitvoeren zolang de voorwaarde true oplevert, oftewel, zolang sommaken maar true is – de gebruiker heeft een j ingetikt – zal het statement na die do worden herhaald. Pas als sommaken een false aflevert wordt het statement overgeslagen en komen we bij de laatste end op regel 1420.
In tegenstelling tot de repeat until, die de uit te voeren statements tussen twee woorden heeft staan en dan ook meer dan één statement kan herhalen kan de while do slechts één statement besturen. Echter, tussen de while do op regel 1310 en de end op regel 1420 staat ook meer dan één statement.
Dit probleem kan worden kan worden opgelost door een aantal statements samen te voegen tot een enkel samengesteld statement. Dat gebeurt simpelweg door er zogenaamde statement-haken – de welbekende begin en end – omheen te zetten. Het while do op regel 1310 ziet het blok vanaf regel 1320 tot en met 1410 als een enkel – samengesteld – statement.
Een andere mogelijke oplossing was geweest om dit blok statements in een eigen procedure onder te brengen.

Opvallend in dit voorbeeld-programma is het gebruik van een eigen variabele in de function sommaken. In feite is het heel gebruikelijk om in Pascal variabelen binnen een procedure of een function te definiëren. Alle blokken die in een Pascal programma voorkomen

3.8 Procedures, functions en lussen

kunnen ook binnen afzonderlijke procedures en functions voorkomen. Het is desgewenst zelfs mogelijk om een procedure binnen een procedure te definiëren. Het voordeel ervan is dat zo'n identifier, in dit geval de variabele antwoord, alleen binnen dat blok bekend is. Mocht u een regel in het hoofdprogramma opnemen als:
writeln(antwoord)
dan zult u ontdekken dat de variabele antwoord daar niet bekend is. Antwoord bestaat alleen binnen sommaken. Daardoor is het onmogelijk om deze variabele per ongeluk te veranderen, terwijl er alleen geheugenruimte voor antwoord in gebruik is als sommaken wordt uitgevoerd. ▶

Desgewenst zou u zelfs een tweede variabele antwoord kunnen definiëren binnen een procedure, function of zelfs het hoofdprogramma, die op geen enkele wijze met de locale variabele antwoord in sommaken in conflict zou komen. In een volgend hoofdstuk zullen we nadere aandacht aan dit verschijnsel van locale en globale variabelen schenken.

Tot slot nog een laatste voorbeeld. Het nu volgende programma is volstrekt gelijk aan ons vorige voorbeeld, met dien verstande dat de statements weliswaar hetzelfde zijn maar dat de lay-out, het commentaar en de identifiers zo onhandig mogelijk gekozen zijn.

```
1000 program reken10a;
1010 var a,b,c:integer;
1020 procedure een(var x:integer);
1030 begin write('Geef waarde:');readln(x) end;
1040 function twee(x:integer):boolean;
1050 begin twee:=x<>0 end;
1060 function drie:boolean;
1070 var a:char;
1080 begin writeln;writeln;write('Sommetje maken? (j/n)');repeat
     read(a)
1090 until(a='j')or(a='n');drie:=a='j' end;
1100 begin while drie do begin een(a);repeat een(b);
1110 if not twee(b) then writeln('Noemer moet ongelijk nul zijn');
1120 until twee(b);c:=a div b;write('Uitkomst is:');write(c)end
     end.

READY.
```

Probeert u het maar eens, alles werkt precies hetzelfde. Maar van leesbaarheid is niets meer over.

De moraal?
Die is simpel. Ook in Pascal is het best mogelijk om volstrekt onbegrijpelijke ▶ en volkomen onleesbare programma's te schrijven. Hoewel de taal Pascal u een heel eind op weg helpt bij het maken van heldere en overzichtelijke programma's moet u het uiteindelijk zelf willen. Pascal biedt de mogelijkheden, aan u om het te verwezenlijken.

8/3.9

Voorwaardelijke statements

Pascal is – het wordt eentonig – veel en veel sterker dan BASIC als het gaat om het besturen van het verloop van een programma. In feite zijn het juist die sturingen die een programma zijn 'intelligentie' verlenen, zonder die sturingen zou een programma domweg van begin tot eind worden uitgevoerd. Doordat we echter in staat zijn om een programma anders te laten reageren, afhankelijk van de waarde van de verschillende variabelen, kan een computerprogramma op een slimme wijze opgezet worden.

Natuurlijk is geen enkel programma echt 'intelligent', hoewel het er voor iemand die niet weet hoe het allemaal in zijn werk gaat soms wel op kan lijken. Maar ook het 'intelligentste' programma reageert alleen maar volgens de richtlijnen van de programmeur. Altans, als het goed is.

In BASIC is er in feite maar één enkel commando dat het verloop van een programma kan beïnvloeden, het IF . . . THEN commando. Sommige programmeurs maken weliswaar ook wel gebruik van een truc met het FOR . . . NEXT commando, waarbij ze uit de FOR . . . NEXT lus springen afhankelijk van een IF commando, maar dat is feitelijk een slechte stijl van programmeren.
Pascal biedt echter veel en veel meer mogelijkheden om het verloop van een programma te besturen, waarbij dergelijk FOR . . . . NEXT trucs gelukkig totaal overbodig zijn. Sterker nog, ze zijn niet eens toegestaan.

We hebben tot nog toe één van deze stuur-commando's – in Pascal-terminologie 'voorwaardelijke statements' genaamd – in de praktijk leren kennen. Dit is:

if expressie then statement else statement

In dit geval staat 'expressie' voor een boolean expressie, die alleen de 'waarden' waar of nietwaar kan opleveren. Daarnaast kent Pascal echter nog een tweede voorwaardelijk statement, dat een meestal in BASIC vervelend stukje programmeerwerk heel netjes oplost.

Stel dat er in een programma getest moet worden op de inhoud van een bepaalde variabele waarna er afhankelijk van die waarde een bepaalde actie moet worden ondernomen. In BASIC kan dat alleen met een constructie als:

3.9 Voorwaardelijke statements

```
IF N=1 THEN doe dit
IF N=2 THEN doe dat
IF N=3 THEN doe dit en dat
IF N=4 THEN doe dat en dit
etcetera.
```

Helemaal vervelend wordt het als er voor sommige waarden van N veel commando's moeten worden uitgevoerd. GOTO's en GOSUB's zijn dan eigenlijk de enige – en tamelijk onleesbare – oplossing.

Pascal lost dit soort problemen veel eleganter op, met behulp van het case-statement. Het bovenstaande stukje BASIC zou in Pascal als volgt kunnen worden uitgedrukt:

```
case n of
1: doedit;
2: doedat;
3: begin doedit; doedat end;
4: begin doedat; doedit end
end
```

De hele reeks IF-commando's in BASIC worden door een enkel case-statement vervangen. Daarbij zijn er echter wel een paar bijzonderheden op te merken. Zo staat er op de laatste regel een end, die niet met een begin gepaard is. Het case statement zelf dient met een eigen end te worden afgesloten; eigenlijk zou er over het 'case . . . end' statement gesproken moeten worden.

Meer formeel opgeschreven ziet het case-statement er als volgt uit:

```
case expressie of
   waarde(n): statement;
   waarde(n): statement
end
```

De expressie mag iedere uitdrukking zijn die uiteindelijk een integer waarde oplevert, reals zijn niet toegestaan. Dat laatste komt door het simpele feit dat die reals – breukgetallen – door allerlei interne onnauwkeurigheden van de computer nooit exact kloppen. En al is de afwijking pas in de achtste decimaal, voor het case-statement komt de expressie dan niet overeen met de waarde. Overigens mogen er desgewenst ook meerdere waarden na elkaar gezet worden, mits deze door komma's van elkaar gescheiden worden. Als de expressie een van de waarden voor een bepaald statement aanneemt op het moment dat het case-statement wordt uitgevoerd zal het programma-verloop naar dat statement gaan.

Bij dit case-statement moet men zich echter wel realiseren op welke manier dit intern wordt uitgevoerd, iets wat eigenlijk verder haast nooit van belang is.
In het geheugen van de 64 wordt namelijk voor ieder case-statement een tabel opgebouwd. Stel dat er een stukje programma als het onderstaande gebruikt is:

```
case n of
  5: galinks;
  6: garechts;
  8, 9: blijf
end
```

dan wordt er in het geheugen een tabel met 5 posities aangemaakt. Ieder van die posities bevat in feite een sprongadres, zoals die ook in machinetaal-programmering voorkomt. Per positie worden er dan ook twee bytes gebruikt.
Als eerste waarde staat het adres waar-

3.9 Voorwaardelijke statements

heen gesprongen moet worden als de variabele n gelijk aan 5 is, daarna krijgen we het adres dat gebruikt moet worden bij de waarde 6.
De derde positie in de sprongtabel bevat een speciale vlag, die aangeeft dat de waarde 7 niet in het case-statement voorkomt, terwijl de vierde en vijfde positie beiden hetzelfde adres, namelijk voor de waarden 8 en 9, bevatten.

Dit wordt zo gedaan omdat het dit nu eenmaal de snelste en simpelste wijze is om een dergelijk commando uit te voeren. Maar als er eens een gat valt tussen die waarden, dan wordt er voor die waarde een soort vlag – de 'niet van toepassing' vlag – in de sprongtabel geplaatst.
Dat is iets om wel in het achterhoofd te houden bij het schrijven van een Pascal-programma. Een case statement met ver uiteen liggende waarden maakt dus een grote sprongtabel.
Zo zal:

```
case n of
1: doedit;
10000: doedat
end
```

zo'n twintigduizend bytes verspillen aan een sprongtabel.
Niet echt handig, dus.
Sterker nog, zonde van de verspilde ruimte. In zo'n geval kan de programmeur veel beter naar het aloude if-commando grijpen.

Om dit case-statement in de vingers te krijgen is wat praktijk natuurlijk de beste leermeester. Probeer de volgende programma's maar eens uit.

```
program case1;
(*                                  *)
var
getal: integer;
(*                                  *)
procedure vraag (var waarde:integer);
begin
 write ('Geef een getal tussen 0 en 9:');
 readln (waarde)
end; (* PROCEDURE VRAAG *)
(*                                  *)
begin (* HOOFDPROGRAMMA *)
repeat
 vraag (getal);
 case getal of
  0: writeln ('nul');
  1: writeln ('een');
  2: writeln ('twee');
  3: writeln ('drie');
  4: writeln ('vier');
  5,6,7: writeln ('vijf, zes of zeven');
  8: writeln ('acht');
  9: writeln ('negen')
 end;
until getal=10
end.
```

3.9 Voorwaardelijke statements

Een simpel maar doeltreffend voorbeeld van het gebruik van het case-commando. Het programma is opgebouwd uit slechts één procedure en het hoofdprogramma zelf, waarbij die procedure de eigenlijke gebruikers-interactie afhandelt. Een groot woord voor het vragen om een getalletje tussen de 0 en de 9.
Met dat getal in de variabele getal – handig, zo'n lange variabele naam – wordt dan het eigenlijke case-statement uitgevoerd waarbij afhankelijk van de waarde van getal een boodschap afgedrukt wordt. In de zesde case-regel wordt een serie van drie mogelijke waarden genoemd, 5, 6 en 7 die alledrie dezelfde boodschap geven.

Om het programmaatje te stoppen moet er een 10 ingevoerd worden. Althans, dat blijkt uit een van de laatste regels. Maar als er inderdaad een 10 ingetikt wordt blijkt een van de problemen van het case-statement. Als de expressie namelijk een waarde aanneemt die niet in de case-lijst genoemd wordt trakteert Oxford Pascal ons op een case error.
Met andere woorden, zodra de besturende expressie – in dit geval de variabele getal – buiten het bereik van het case-statement valt lijdt dit tot een foutmelding. Spijtig maar waar.

Een manier om dit te ondervangen is bijvoorbeeld:

```
program case2;
(*                                       *)
var
getal: integer;
(*                                       *)
procedure vraag (var waarde:integer);
begin
 write ('Geef een getal tussen 0 en 9:');
 readln (waarde)
end; (* PROCEDURE VRAAG *)
(*                                       *)
begin (* HOOFDPROGRAMMA *)
repeat
 vraag (getal);
 case getal of
  0: writeln ('nul');
  1: writeln ('een');
  2: writeln ('twee');
  3: writeln ('drie');
  4: writeln ('vier');
  5,6,7: writeln ('vijf, zes of zeven');
  8: writeln ('acht');
  9: writeln ('negen');
 10: writeln ('we stoppen ermee')
 end;
until getal=10
end.
```

De tweede versie is wel in staat om die einde-vlag, de waarde 10, te verwerken.

Dit programma bevat een extra regel in het case-statement, de 10-regel. Daar-

3.9 Voorwaardelijke statements

door wordt die waarde 10 nu wel zonder fouten geaccepteerd. Daarbij is die extra regel meteen gebruikt om een keurige einde-melding op het scherm te brengen.

```
program case3;
(*                                  *)
var
getal: integer;
(*                                  *)
procedure vraag (var waarde:integer);
begin
 write ('Geef een getal tussen 0 en 9:');
 readln (waarde)
end; (* PROCEDURE VRAAG *)
(*                                  *)
begin (* HOOFDPROGRAMMA *)
repeat
 vraag (getal);
 case getal of
  0: writeln ('nul');
  1: writeln ('een');
  2: writeln ('twee');
  3: writeln ('drie');
  4: writeln ('vier');
  5,6,7: writeln ('vijf, zes of zeven');
  8: writeln ('acht');
  9: writeln ('negen');
 1000: writeln ('we stoppen ermee')
 end;
until getal=1000
end.
```

Laten we, als demonstratie van het feit dat het gebruik van ver uiteenliggende waarden binnen een case-lijst inderdaad onverstandig is, dat ook eens uitproberen. De volgende listing gebruikt de waarde 1000 als eind-vlag.

Hoewel dit programma nauwelijks langer is dan het vorige – om precies te zijn is de source 4 bytes langer, tweemaal twee nullen – duurt het compileren opeens een stuk langer.
Het vorige programma werd in zo'n 11 seconden vertaald, deze versie kost maar liefst 25 seconden. Hoewel we geen simpele wijze hebben om het geheugengebruik te controleren mag aangenomen worden dat die 14 seconden gebruikt worden om een wel bijzonder lijvige sprong-tabel aan te maken.

Bij deze drie voorbeelden is overigens bewust een stijlfout in het programmeren gemaakt. Want hoewel de tweede en de derde versie alle toegestane waarden – 0 tot en met 9 alsmede 10 of 1000 – goed verwerken wordt in feite iedere integer-waarde geaccepteerd door de procedure vraag. Ook 11, om maar een voorbeeld te geven.
Zo'n waarde ligt echter wel buiten de binnen het case-statement bekende waarden, hetgeen tot een case-error leidt. Een goede programmeur zal ten alle tijden proberen dergelijke fouten te ondervangen. Andere versies van Pascal hebben daar een – overigens geen

3.9 Voorwaardelijke statements

deel van de standaarddefinitie uitmakende – uitbreiding voor. In Oxford Pascal is dat jammer genoeg niet het geval.

Het ondervangen van dergelijke foute invoer, voordat deze tot fouten in het programma-verloop leiden, is echter niet moeilijk. De volgende listing laat een mogelijke oplossing zien:

```
program case4;
(*                                  *)
var
getal: integer;
(*                                  *)
procedure vraag (var waarde:integer);
begin
 write ('Geef een getal tussen 0 en 9:');
 readln (waarde)
end; (* PROCEDURE VRAAG *)
(*                                  *)
begin (* HOOFDPROGRAMMA *)
repeat
 repeat
  vraag (getal)
 until (getal>0) and (getal<11);
 case getal of
  0: writeln ('nul');
  1: writeln ('een');
  2: writeln ('twee');
  3: writeln ('drie');
  4: writeln ('vier');
  5,6,7: writeln ('vijf, zes of zeven');
  8: writeln ('acht');
  9: writeln ('negen');
 10: writeln ('we stoppen ermee')
 end;
until getal=10
end.
```

Door de aanroep van de procedure vraag in het hoofdprogramma op te nemen in een klein repeat . . . until-loopje, waarbij getest wordt of de gekregen invoer wel aan de eisen voldoet, kan men verkeerde invoer tijdig opvangen. Feitelijk moet elke invoer die door de gebruiker ingetikt wordt op een dergelijke wijze gecheckt worden alvorens een programma ze verder verwerkt. De reden daarvoor is simpel, want dergelijke foutieve invoer kan anders veel later – en op een heel andere plek in een programma – tot foutmeldingen leiden. Om dan nog uit te zoeken wat die fout veroorzaakt heeft is veel en veel lastiger dan om elke invoer meteen even te controleren op juistheid. Mocht een programma ondanks die controles afbreken, dan weet men tenminste zeker dat het niet aan een onjuiste invoer lag maar dat er inderdaad een 'bug' in het programma zelf schuilt.

8/3.10

BNF-notatie

Een van de problemen waar men bij het aanleren van een taal als Pascal al snel tegen aanloopt is het feit dat het omschrijven van de mogelijkheden van sommige statements in gewone taal haast niet te doen is.
Gelukkig is dat ook niet nodig, er zijn andere en betere systemen om een programmeertaal te definiëren dan we tot nog toe gebruikt hebben. Een van die manieren in de zogenaamde BNF-notatie, hetgeen staat voor Backus Naur Form.
In hoofdstuk 12/3, een onderdeel van het programmeerkundegedeelte van dit boek, staat een omschrijving van deze BNF-notatie. Met deze notatie zullen we nu allereerst eens op een rijtje zetten wat we al weten van Pascal.
Daarbij zullen de termen worden gebruikt zoals die door de ontwerpers van Pascal gedefinieerd zijn. Ook in allerlei andere literatuur worden deze Engelstalige termen gebruikt.

De onderstreepte woorden vormen daarbij de voor Pascal gereserveerde woorden, termen dus die we nooit voor iets anders dan hun vastgelegde betekenis kunnen gebruiken. Dat lijkt misschien voor de hand te liggen, het feit dat we de Pascal-woorden niet voor andere zaken kunnen gebruiken. Dat is echter niet zo.

Lang niet alle Pascal-woorden zijn gereserveerd, zo kunnen we desgewenst zelf een procedure schrijven met de naam write. In zo'n geval zal de zelf-geschreven procedure uitgevoerd worden als er in dat programma write aangeroepen wordt, en niet de standaard-procedure.

We beginnen met eens te definiëren wat een Pascal-programma eigenlijk is. In BNF:

<program> ::= <program heading><block>.

Een programma bestaat dus altijd uit twee onderdelen, de programma-kop – program heading – en het block, dat het eigenlijke programma omvat. Dat block moet met een punt afgesloten worden, het teken voor de compiler dat het programma afgelopen is.
Die program heading is weer omschreven met:

<program heading> ::= program <identifier> (<file identifier>{,<file identifier>});
<file identifier> ::= <identifier>
<identifier> ::= <letter> {<letter or digit>}
<letter or digit> ::= <letter>|<digit>

Zo'n programmakop bestaat dus uit

3.10 BNF-notatie

het gereserveerde woord program, gevolgd door een identifier, zeg maar de programmanaam. Uit de derde en vierde definitie kunnen we lezen dat zo'n identifier bestaat uit minimaal een letter, eventueel gevolgd door andere letters en cijfers. Let wel, er is geen beperking aan de lengte van een identifier, maar slechts de eerste 8 tekens worden gebruikt om de namen van elkaar te onderscheiden.

Wat we tot nog toe niet tegengekomen waren was het feit dat Pascal wil dat de eventueel gebruikte bestanden ook in die program heading gemeld worden. Ze zijn niet vereist, maar als er bestanden gebruikt worden moeten we ze allemaal vermelden, waarbij de namen door komma's gescheiden worden. Het geheel moet bovendien tussen haakjes staan.
Die bestandsnamen moeten aan dezelfde eisen voldoen als alle identifiers. Sterker nog, het blijkt dat standaard-Pascal op zijn minst één bestandsnaam vereist in de programma-kop. Normaal gesproken zal dit het bestand 'output' zijn, wat gebruikt wordt om onder andere de foutmeldingen op te schrijven. Bij Oxford Pascal is dit echter niet noodzakelijk. Bij deze op interactief gebruik geörienteerde Pascal worden de bestanden 'input' en 'output' zonder meer aanwezig verondersteld, aangezien men anders geen in- en uitvoer zou kunnen plegen.

Dit is echter de enige echte afwijking tussen standaard-Pascal en de Oxford-variant.
De program heading wordt tenslotte afgesloten door de in Pascal altijd als scheider gebruikte punt-komma.

Het bovenstaande toont meteen aan waarom een schrijfwijze als de Backus Naur Form zo handig is; een paar regels BNF zeggen net zoveel als een hele uitleg in tekst.

Het is echter feitelijk onmogelijk om bij een complex geheel als een programmeertaal de BNF-definitie slechts gedeeltelijk te geven. Alle onderdelen grijpen op elkaar in in zo'n samenstelsel van definities.
Vandaar ook dat u in dit hoofdstuk de volledige BNF-notatie van het standaard Pascal zult aantreffen. Weliswaar komen vele van de hier gepresenteerde statements pas in latere afleveringen aan de orde, maar dan kunt u altijd op dit hoofdstuk teruggrijpen.
Een woord van waarschuwing, de hier gegeven definities omschrijven het standaard-Pascal, niet de wat uitgebreidere Oxford-versie. Deze uitbreidingen zullen pas later aan de orde komen.

Standaard-Pascal gedefinieerd in Backus Naur Form
Waarschuwing:
Normaal gesproken maakt de BNF-notatie gebruik van de vierkante haakjes openen en sluiten – [ ] – om aan te geven dat een bepaalde term naar eigen inzicht al dan niet gebruikt kan worden.
Daar deze vierkante haken echter ook een onderdeel vormen van Pascal zelf moeten ze in de onderstaande definities als een gedeelte van de syntax van Pascal zelf gezien worden.

<program> ::= <program heading> <block>.
<program heading> ::= program <identifier> (<file identifier>{, <file

3.10 BNF-notatie

```
identifier>});
<file identifier> ::= <identifier>
<identifier> ::= <letter> { <letter or digit>}
<letter or digit> ::= <letter> | <digit>
<block> ::= <label declaration part> <constant definition part> <type definition part> <variable declaration part> <procedure and function declaration part> <statement part>
<label declaration part> ::= <empty> | label <label> { , <label>} ;;
<label> ::= <unsigned integer>
<constant definition part> ::= <empty> | const <constant definition> {; <constant definition> };
<constant definition> ::= <identifier> = <constant>
<constant> ::= <unsigned number> | <sign> <unsigned number> | <constant identifier> | <sign> <constant identifier> | <string>
<unsigned number> ::= <unsigned integer> | <unsigned real>
<unsigned integer> ::= <digit> {<digit>}
<unsigned real> ::= <unsigned integer> . <digit> {<digit>} | <unsigned integer> . <digit> {<digit>} E <scale factor> | <unsigned integer> E <scale factor>
<scale factor> ::= <unsigned integer> | <sign> <unsigned integer>
<sign> ::= + | −
<constant identifier> ::= <identifier>
<string> ::= ' <character> {<character>} '
<type definition part> ::= <empty> | type <type definition> {; <type definition>} ;
<type definition> ::= <identifier> = <type>
<type> ::= <simple type> | <structured type> | <pointer type>
<simple type> ::= <scalar type> | <subrange type> | <type identifier>
<scalar type> ::= ( <identifier> {, <identifier>} )
<subrange type> ::= <constant> .. <constant>
<type identifier> ::= <identifier>
<structured type> ::= <unpacked structured type> | packed <unpacked structured type>
<unpacked structured type> ::= <array type> | <record type> | <set type> | <file type>
<array type> ::= array [ <index type> {, <index type>} ] of <component type>
<index type> ::= <simple type>
<component type> ::= <type>
<record type> ::= record <field list> end
<field list> ::= <fixed part> | <fixed part> ; <variant part> | <variant part>
<fixed part> ::= <record section> {; <record section>}
<record section> ::= <field identifier> {, <field identifier>} : <type> | <empty>
<variant part> ::= case <tag field> <type identifier> of <variant> {; <variant>}
<tag field> ::= <field identifier> : | <empty>
<variant> ::= <case label list> : ( <field list> ) | <empty>
<case label list> ::= <case label> {, <case label>}
<case label> ::= <constant>
<set type> ::= set of <base type>
<base type> ::= <simple type>
<file type> ::= file of <type>
<pointer type> ::= ^ <type identifier>
<variable declaration part> ::= <empty> | var <variable declaration> {; <variable declaration>} ;
```

3.10 NFB-notatie

```
<variable declaration> ::= <identifier> {, >identifier>} : <type>
<procedure and function declaration part> ::= <procedure or function declaration>
<procedure or function declaration> ::= <procedure declaration> | <function declaration>
<procedure declaration> ::= <procedure heading> <block>
<procedure heading> ::= procedure <identifier> ; | procedure <identifier> ( <formal parameter section> {;<formal parameter section>} ) ;
<formal parameter section> ::= <parameter group> | var <parameter-group> | function <parameter group> | procedure <identifier> { , <identifier>}
<parameter group> ::= <identifier> {, <identifier>} : <type identifier>
<function declaration> ::= <function heading> <block>
<function heading> ::= function <identifier> : <result type> ; | function <identifier> ( <formal parameter section> {; <formal parameter section>} ) : <result type> ;
<result type> ::= <type identifier>
<statement part> ::= <compound statement>
<statement> ::= <unlabelled statement> | <label> : <unlabelled statement>
<unlabelled statement> ::= <simple statement> | <structured statement>
<simple statement> ::= <assignment statement> | <procedure statement> | <go to statement> | <empty statement>
<assignment statement> ::= <variable> := <expression> | <function identifier> := <expression>
<variable> ::= <entire variable> | <component variable> | <referenced variable>
<entire variable> ::= <variable identifier>
<variable identifier> ::= <identifier>
<component variable> ::= <indexed variable> | <field designator> | <file buffer>
<indexed variable> ::= <array variable> { <expression> {, <expression>} ]
<array variable> ::= <variable>
<field designator> ::= <record variable> . <field identifier>
<record variable> ::= <variable>
<field identifier> ::= <identifier>
<file buffer> ::= <file variable> ^
<file variable> ::= <variable>
<referenced variable> ::= <pointer variable> ^
<pointer variable> ::= <variable>
<expression> ::= <simple expression> | <simple expression> <relational operator> <simple expression>
<relational operator> ::= = | < > | < | <= | >= | > | in
<simple expression> ::= <term> | < sign> <term> | <simple expression> <adding operator> <term>
<adding operator> ::= + | − | or
<term> ::= <factor> | <term> <multiplying operator> <factor>
<multiplying operator> ::= * | / | div | mod | and
<factor> ::= <variable> | <unsigned constant> | ( <expression> ) | <function designator> | <set> | not <factor>
<unsigned constant> ::= <unsigned number> | <string> | <constant identifier> | nil
<function designator> ::= <function identifier> | <function identifier> ( <actual parameter> {, <actual para-
```

3.10 BNF-notatie

meter>})
<function identifier> ::= <identifier>
<set> ::= [<element list>]
<element list> ::= <element> {, <element> } | <empty>
<element> ::= <expression> | <expression> . . <expression>
<procedure statement> ::= <procedure identifier> | <procedure identifier> (<actual parameter> {, <actual parameter>})
<procedure identifier> ::= <identifier>
<actual parameter> ::= <expression> | <variable> | <procedure identifier> | <function identifier>
<go to statement> ::= goto <label>
<empty statement> ::= <empty>
<empty> ::=
<structured statement> ::= <compound statement> | <conditional statement> | <repetitive statement> | <with statement>
<compound statement> ::= begin <statement> {; <statement>} end
<conditional statement> ::= <if statement> | <case statement>
<if statement> ::= if <expression> then <statement> | if <expression> then <statement> else <statement>
<case statement> ::= case <expression> of <case list element> {; <case list element>} end
<case list element> ::= <case label list> : <statement> | <empty>
<case label list> ::= <case label> {, <case label> }
<repetitive statement> ::= <while statement> | <repeat statement> | <for statement<
<while statement> ::= while <expression> do <statement>
<repeat statement> ::= repeat <statement> {; <statement>} until expression
<for statement> ::= for <control variable> := <for list> do <statement>
<for list> ::= <initialvalue> to <final value> | <initial value> downto <final value>
<control variable> ::= <identifier>
<initial value> ::= <expression>
<final value> ::= with <record variable list> do <statement>
<record variable list> ::= <record variable> {, <record variable<}

Dat was het dan. De volledige definitie van standaard-Pascal in de Backus Naur Form.
Veel van de hier gedefinieerde onderdelen van Pascal zijn nog niet aan de orde gekomen; ze zullen in komende afleveringen behandeld worden. Deze definitie kan echter ook nu al gebruikt worden om de reeds bekende elementen van Pascal eens als een logisch geheel te bekijken. Als u het samenspel van definities in volgorde leest en probeert te begrijpen zult u tot de vaststelling komen dat de syntax van Pascal inderdaad volledig in elkaar grijpt.

Zo blijken Pascal-programma's te bestaan uit een <program heading> en een <block>, gevolgd door een punt. In dat <block> kunnen onder meer <procedures> en <functions> staan, die op hun beurt weer bestaan uit of een <procedure heading> of een <function heading>, in beide gevallen gevolgd door een <block>. Slechts de punt, die inhoudt dat een programma beëindigd is, ontbreekt.
Met andere woorden, een Pascal-procedure of -functie kan alle elementen van een Pascal-programma zelf bevatten, waarbij ze ook nog genest kunnen worden. Procedures en functies zijn in feite

3.10 BNF-notatie

volledige, eventueel met kleine wijzigingen ook los te gebruiken, Pascal-programma's.
Daardoor is er eigenlijk geen grens meer aan de complexiteit van zo'n Pascal programma. Volledige bestaande programma's kunnen desgewenst zonder veel problemen als onderdeel van een nieuw programma ingezet worden.
Overigens is deze BNF-definitie zeer zeker nuttig te gebruiken. Als u een programmaatje geschreven heeft kunt u met deze definitie heel eenvoudig vaststellen of daar al dan niet syntax-fouten is steken. Bovendien blijkt bij nadere bestudering dat er volgens deze BNF-definitie allerlei constructies zijn toegestaan waar men in eerste instantie niet aan zou denken. En, als het volgens de definitie mag, dan moet iedere volledige Pascal-compiler – zoals Oxford-Pascal – het kunnen verwerken.

8/3.11

Nuttige routines

Het fijne van Pascal is dat de 'woordenschat' van de taal naar believen uitgebreid kan worden. Routines die niet bestaan kunnen zonder problemen zelf geschreven worden. Slechts in enkele gevallen moeten we ons dan tot diep in de machine wenden, dit is afhankelijk van de gebruikte Pascal-implementatie.

Oxford Pascal is een redelijk uitgebreide compiler, er zijn echter bijvoorbeeld geen string-procedures en functies aanwezig. Ook een aantal andere minder belangrijke, maar daarom niet minder nuttige voorzieningen zijn achterwege gelaten. Hier nu echter kunnen we gebruik maken van de Pascal-eigenschap om de taal zelf uit te breiden. In deze paragraaf worden een aantal procedures en functies gepresenteerd waarvan diegene die nodig zijn ondergebracht kunnen worden in eigen programma's. Omdat het mogelijk is dat een programma niet alle routines gebruikt kan zelf besloten worden welke routines wel en welke routines niet in een programma opgenomen worden. Er zijn echter een paar routines die gebruik maken van eerder gedefinieerde zaken, houd hier even rekening mee.

In feite is het mogelijk om oneindig veel zinnige en onzinnige routines te fabriceren. Dit zullen we uiteraard niet doen. Bij de in deze aanvulling gepresenteerde verzameling is uitgegaan van een bestaande Pascal-versie die heel populair is op PC's en CPM-systemen: Turbo Pascal van de firma Borland. Omdat binnen WEKA ook een grote hoeveelheid Turbo Pascal programma's aanwezig is hebben we ervoor gekozen om een aantal van de (standaard) Turbo Pascal routines in Oxford Pascal te implementeren. De function KeyPressed is bijvoorbeeld standaard in Turbo aanwezig. In Oxford Pascal is deze functie niet zo moeilijk om na te maken, hetgeen uit de programmalisting wel zal blijken.

Uiteraard is geen enkele Pascal-implementatie ideaal, zo ook niet Turbo Pascal. Uit de jarenlange ervaring van de auteurs met deze compiler is er, ook voor Turbo Pascal, een standaard-bibliotheek met extra routines gecreëerd. Deze zullen we in latere aanvullingen gaan behandelen, op deze manier wordt het werken met Oxford Pascal gewoon een stuk fijner; er hoeft niet telkens opnieuw het wiel uitgevonden te worden. Wat nodig is wordt gewoon 'geleend'.

Alle routines zijn in één listing samengevat. Deze volgt hieronder, daarna zullen de routines een voor een besproken worden:

3.11 Nuttige routines

```
(*
Declaratie van constantes
*)
const
  Black       = 0;
  White       = 1;
  Red         = 2;
  Cyan        = 3;
  Purple      = 4;
  Green       = 5;
  Blue        = 6;
  Yellow      = 7;
  Orange      = 8;
  Brown       = 9;
  Lightred    = 10;
  Grey1       = 11;
  Grey2       = 12;
  Lightgreen  = 13;
  Lightblue   = 14;
  Grey3       = 15;
  StrLen      = 80;
(*
enige type-declaraties
*)
type
  byte   = 0..255;
  string = packed array[1..StrLen] of char;
(*
en dan nu de procedures etc.
*)
procedure CrtInit; (* Scherminitialisatie etc. *)
begin
  pen(14);
  screen(6);
  border(14);
  page;
end;
(*
*)
procedure CrtExit; (* als CrtInit *)
begin
  CrtInit;
end;
(*
*)
procedure ClrScr; (* maakt het scherm schoon *)
begin
  page
```

3.11 Nuttige routines

```
end;
(*
*)
procedure GotoXY(X,Y:byte); (* positioneert de cursor *)
                            (* X loopt van 1 t/m 40, *)
                            (* Y van 1 t/m 25 *)
  procedure Plot; extern 58732;
begin
  poke(211,X-1);
  poke(214,Y-1);
  Plot;
end;
(*
*)
procedure LowVideo; (* schakelt 'dim'-letters in *)
begin
  pen(14)
end;
(*
*)
procedure NormVideo; (* schakelt 'bright'-letters in *)
begin
  pen(15)
end;
(*
*)
function WhereX : byte; (* retourneert de cursorkolom *)
begin
  WhereX:=peek(211)+1
end;
(*
*)
function WhereY : byte; (* retourneert de cursorregel *)
begin
  WhereY:=peek(214)+1;
end;
(*
*)
procedure Sound(F:integer); (* produceert toontje *)
begin
  envel(3,0,0,15,0);
  poke(54272+15,F div 256);
  poke(54272+14,F mod 256);
  poke(54272+18,33);
  volume(15);
end;
(*
*)
```

3.11 Nuttige routines

```
procedure NoSound; (* zet toontje uit *)
begin
  poke(54272+18,0);
  volume(0);
end;
(*
*)
procedure Delay(D:integer); (* vertragingslus van D ms *)
var
  Teller : integer;
begin
  for Teller:=1 to D do (* niets *);
end;
(*
*)
procedure TextColor(C:byte); (* definieer letterkleur *)
begin
  pen(C)
end;
(*
*)
procedure TextBackground(C:byte); (* definieer achterfrond *)
begin
  screen(C)
end;
(*
*)
procedure FillChar(Loc,Aantal:integer;Waarde:byte);
                (* vult geheugen met waarde *)
var
  Teller : integer;
begin
  for Teller:=Loc to Loc+Aantal-1 do poke(Teller,Waarde)
end;
(*
*)
function Hi(I:integer):byte; (* levert hi-byte van I *)
begin
  Hi:=I div 256
end;
(*
*)
function Lo(I:Integer):byte; (* levert lo-byte van I *)
begin
  Lo:=I mod 256
end;
(*
*)
```

3.11 Nuttige routines

```
function Swap(I:Integer):integer; (* verwisselt hi/lo-byte *)
begin
  Swap:=Hi(I)+256*Lo(I)
end;
(*
*)
function KeyPressed:boolean; (* true indien toets ingedrukt *)
begin
  KeyPressed:=(peek(198)<>0)
end;
(*
*)
function UpCase(C:char):char; (* levert shift-versie *)
begin
  if C in ['a'..'z'] then Upcase:=chr(ord(C)-32)
                     else Upcase:=C
end;
(*
*)
function LoCase(C:char):char; (* levert nite-shift-versie *)
begin
  if C in ['A'..'Z'] then Locase:=chr(ord(C)+32)
                     else Locase:=C
end;
(*
*)
function Length(var S:string):byte; (* retourneert lengte *)
var
  Teller : byte;
begin


  teller:=StrLen+1;
  repeat
    Teller:=Teller-1;
  until S[Teller]<>' ';
  Length:=Teller;
end;
(*
*)
procedure Copy(var S:string;Start,Aantal:byte);
              (* voor MID$, LEFT$ en RIGHT$-simulatie *)
```

3.11 Nuttige routines

```
var
  Dummy : string;
  Teller : byte;
begin
  for Teller:=Start to Start+Aantal-1 do
      Dummy[Teller-Start+1]:=S[Teller];
for Teller:=Aantal+1 to StrLen do Dummy[Teller]:=' ';
  S:=Dummy;
end;
(*
*)
BEGIN
END.
```

De constantes
De constantes worden gedefinieerd omdat tekst altijd leesbaarder is dan cijfertjes. Deze constantes kunnen in zelfgedefinieerde routines terugkomen. Het volgende is dan mogelijk:
TextColor(Red);
Altijd mooier dan TextColor(2) natuurlijk.
De constante StrLen wordt gebruikt om de maximaal gebruikte string-lengte voor de string-functies te definiëren. Deze mag aangepast worden indien grotere strings gebruikt worden.

De types
De types byte en string worden in programma's veel gebruikt. Ze zijn zelfs standaard in Turbo Pascal. Het type byte heeft 1 byte geheugenbesparing tot gevolg in vergelijking tot integers. Het stringtype wordt uiteraard voor tekstmanipulatie en tekstvariabelen gebruikt.

CrtInit en CrtExit;
Deze routines kunnen aan het begin en eind van programma's gebruikt worden. Ze zijn identiek, CrtExit kan eventueel uitgebreid worden met het hires(0)-commando om terug naar tekstmode te keren. Dit in geval gebruik wordt gemaakt van graphics.

ClrScr;
Deze routine wist het scherm en is identiek aan het page-commando. Page is echter in standaard-Pascal alleen voor de printer gedefinieerd (FormFeed). Verder waarborgt de ClrScr-variant de compatibiliteit met Turbo Pascal.

3.11 Nuttige routines

GotoXY(X,Y:byte);
Deze procedure zet de cursor op een willekeurige plaats op het scherm. X loopt van 1 t/m 40, Y loopt van 1 t/m 25.
Voorbeeld:
GotoXY(10,4);
Er wordt in deze procedure gebruik gemaakt van de PLOT-routine die ook vanuit BASIC beschikbaar is middels het SYS 58732-commando. Het extern-commando is een Oxford-extra die het mogelijk maakt om machinetaalroutines toe te passen. Hierover meer in een volgende aanvulling.

LowVideo, NormVideo;
Deze routines schakelen heldere of niet-heldere letters in. Deze routines zijn voornamelijk van belang bij zwartwit-TV's en monochrome monitoren. Op kleurenbeeldschermen kan uiteraard beter gebruik gemaakt worden van de kleuren-commando's (zie verder).

function WhereX:byte;
function WhereY:byte;
Deze twee functies leveren respectievelijk de cursorkolom en de cursorregel. Vooral in samenwerking met GotoXY veel gebruikt. WhereX retourneert waardes van 1 t/m 40, WhereY levert waardes van 1 t/m 25. Voorbeeld:
GotoXY (WhereX+1,WhereY+1);

Sound(F:integer) en NoSound;
Dit zijn commando's voor simpele piepjes en dergelijke. De 64 bezit uiteraard een hifi sound-synthesizer, aan deze commando's valt dit echter niet te merken. In zakelijke en serieuze applicaties wordt hier echter ook geen gebruik van gemaakt, slechts af en toe dient de gebruiker op foute invoer o.i.d. geattendeerd te worden. Sound produceert een toon van (SID)-frequentie F, deze blijft net zolang klinken tot het NoSound-commando gegeven wordt. Voorbeeld:
Sound(10000);
for Teller:=1 to 500 do (* niets *);
NoSound;

Delay(D:integer);
De regels hierboven geven het al aan: om te wachten moet een aparte wachtlus gebouwd worden. De routine Delay vangt dit echter op. In de body van Delay wordt net zoveel keer niets gedaan als de parameter D aangeeft. In praktijk blijkt D het aantal miliseconden aan te geven. Delay(1000) wacht dus 1 seconde en Delay(1000) wacht 10 seconden.

TextColor(C:byte) en TextBackground(C:byte);
Deze routines zijn standaard in Oxford aanwezig, echter onder andere namen. Om de compatibiliteit met Turbo te garanderen worden ze hier gedefinieerd. TextColor staat voor de letterkleur, TextBackground voor de achtergrondkleur. De standaard Oxford-routine Border(C:byte) geeft de rand van het scherm de gewenste kleur. Voorbeeld:
TextBackground(0);
TextColor(Yellow); (* Duidelijker! *)

FillChar(Loc,Aantal:integer;Waarde:byte);
Deze routine vult een geheugengedeelte met de gewenste waarde. Leuk om delen van het scherm te kleuren/vullen. Ook de SID kan op die manier mooi geïnitialiseerd worden:
FillChar (54272,25,0);

function Hi(I:integer):byte;
function Lo(I:integer):byte;
Deze functies retourneren het high-

3.11 Nuttige routines

respectievelijk low-byte van integer I. Heel handig bij het gebruik van adressen.
Voorbeeld:
Write(Hi(49152),Lo($1107));
resultaat
    192   7
Dit omdat 49152 gelijk is aan $C000 waarvan $C0 het high-byte is. Dit is, zoals wellicht bekend, gelijk aan 12*16=192. Idem is het low-byte van $1107 gelijk aan $07=7.

function Swap(I:integer):integer;
Deze functie verwisselt het high- en low-byte van een integer. Dit is belangrijk bij het opslaan van adressen in het geheugen. Swap ($1234) is dan gelijk aan $3412.

function KeyPressed:boolean;
Dit is een functie die TRUE is als er een toets wordt ingedrukt. Handig als er gewacht moet worden tot een gebruiker bijvoorbeeld iets gelezen heeft. Met de volgende constructie wordt er dan gewacht totdat hij op een toets heeft gedrukt:
repeat until KeyPressed;
De volgende constructie leegt de toetsenbordbuffer:
while KeyPressed do Toets:=getkey;
getkey moet dan wel als char gedefinieerd zijn.

function UpCase(C:char):char;
function LoCase(C:char):char;
Deze functies retourneren respectievelijk de hoofd- en kleine letter van het teken C. Alleen dus wanneer C van a t/m z loopt wordt er door UpCase een hoofdletter van gemaakt. Omgekeerd geldt hetzelfde voor LoCase, deze maakt van alle letters A t/m Z een kleine letter. Voorbeeld:
Write(UpCase('a'),LoCase('b'));
resultaat:
Ab (* LoCase('b') is uiteraard 'b'! *)

functionLength(varS:string):byte;
Deze routine levert de lengte van een string-variabele (niet een constante!). Wellicht zullen we in een later stadium een machinetaalversie schrijven omdat deze routine zeer traag is. Het voordeel is dat dit soort routines meestal niet in loops uitgevoerd hoeven te worden, maar eenmalig. De routine maakt gebruik van het feit dat een string wordt aangevuld met spaties totdat zijn formele lengte is bereikt. Voorbeeld:
Len:=Length(Naam);

Copy(var S:string;Start,Aantal:byte);
Copy is een veelzijdige routine die gebruikt kan worden om de BASIC-routines MID$, LEFT$ en RIGHT$ te simuleren. De routine is wederom zeer traag omdat niet gebruikte stukken met spaties aangevuld dienen te worden omdat de formele lengte altijd 80 blijft (in ons geval dan. Maak StrLen kleiner als dat kan). Voorbeelden zijn:
Copy(Naam,1,L);
(* Naam:=LEFT$(Naam,L) *)
Copy(Naam,Length(Naam)-R+1,R);
(* Naam:=RIGHT$ (Naam,R) *)
Copy(Naam,M,N);
(* Naam:=MID$(Naam,M,N) *)
Het is helaas niet mogelijk om Copy als functie te declareren omdat Oxford geen strings als functie-type toestaat. In zoverre wijkt dit Copy-commando iets van de Turbo Pascal versie af, welke wel een functie is.

Voorbeeldprogramma
Tot slot volgt hier dan nog een voorbeeldprogramma waarin de meeste routines gebruikt worden. De regels kunnen direct achter de voorgaande doorgetypt worden. Deze dienen dus ook aanwezig te zijn!

3.11 Nuttige routines

```
BEGIN
  CrtInit;
  GotoXY(2,2);
  write('Hier wat voorbeeldjes...');
  GotoXY(2,4);
  write('Wat is je naam: ');
  NormVideo;
  readln(Naam);
  LowVideo;
  if Naam[1]=' ' then Len:=0 else Len:=Length(Naam);
  GotoXY(2,6);
  write('Jouw naam heeft',Len:3,' letters.');
  GotoXY(2,8);
  TextColor(Black);
  write('Let op...');
  Delay(2000);
  FillChar(1024,40,ord('-'));
  FillChar(1984,40,ord('-'));
  Sound(10000);
  Delay(300);
  NoSound;
  if Len=0 then Naam[1]:='?';
  GotoXY(2,10);
  LowVideo;
  write('Dag ');
  for Teller:=1 to Len do write(UpCase(Naam[Teller]));
  GotoXY(2,12);
  write('Druk nu op een toets...');
  repeat until KeyPressed;
Toets:=getkey;
  ClrScr;
  GotoXY(2,2);
  write('Volgende regel');
  GotoXY(WhereX,WhereY+1);
  write('is hier!');
  GotoXY(2,6);
  write('Geef een getal tussen 1 en 5: ');
  NormVideo;
  read(Teller);
  GotoXY(2,8);
  LowVideo;
  write('De ',Teller:1,'e letter van jouw naam');
  write(' is een ''');
  Copy(Naam,Teller,1);
  write(Naam[1],'''.');
  GotoXY(2,12);
  writeln('Dat was ''t!');
  Delay(7000);
  CrtExit;
END.
```

3.11 Nuttige routines

Aan bovenstaande listing dienen nog wel de VAR-declaraties toegevoegd te worden. Doe dit in programma1 nà de type declaraties:

```
VAR
Teller: byte;
Len: byte;
Naam: string;
```

8/3.12

Voorbeeldprogramma's

Om het werken met Oxford Pascal wat toegankelijker te maken zullen in dit hoofdstuk een aantal voorbeeldprogramma's gepresenteerd worden. Daarbij zal af en toe gebruik gemaakt worden van de eigen routines uit het vorige hoofdstuk. Zij maken het programmeren in Oxford Pascal een stuk aantrekkelijker.

Aflossingsschema

Het eerste programma is een programma om een aflossingsschema mee te maken. Hiermee kunnen diverse leningen, hypotheken en dergelijke met elkaar vergeleken worden omdat er met veel extra zaken rekening gehouden wordt: geldontwaarding (disagio), administratiekosten, rente en aflossingspercentage. Het programma geeft voor elke aflossingsperiode het saldo, de te betalen premie en de opsplitsing van deze premie in rente en aflossing.
Hoe een en ander er uit ziet is in de volgende twee voorbeelden te zien:

Aflossingsschema

Geef de hoogte van het bedrag: 10000
Wat is het disagio? 0
Administratiekosten (%): 1
Rente in%: 8
Aflossingspercentage: 10

Saldo	Premie	Rente	Aflossing
10101.01	1818.18	808.08	1010.10
9090.91	1818.18	727.27	1090.91
8000.00	1818.18	640.00	1178.18
6821.82	1818.18	545.75	1272.44
5549.38	1818.18	443.95	1374.23
4175.15	1818.18	334.01	1484.17
2690.98	1818.18	215.28	1602.90
1088.08	1175.12	87.05	1088.08

Voorbeeld 1

Aflossingsschema

Geef de hoogte van het bedrag: 150000
Wat is het disagio? 2
Administratiekosten (%): 1
Rente in%: 10.8
Aflossingspercentage: 3

Saldo	Premie	Rente	Aflossing
154639.18	21340.21	16701.03	4639.18
150000.00	21340.21	16200.00	5140.21
144859.79	21340.21	15644.86	5695.35
139164.45	21340.21	15029.76	6310.45
132854.00	21340.21	14348.23	6991.97
125862.02	21340.21	13593.10	7747.11
118114.92	21340.21	12756.41	8583.80
109531.12	21340.21	11829.36	9510.84
100020.28	21340.21	10802.19	10538.02
89482.26	21340.21	9664.08	11676.12
77806.14	21340.21	8403.06	12937.14
64869.00	21340.21	7005.85	14334.35
50534.64	21340.21	5457.75	15882.46
34652.18	21340.21	3742.44	17597.77
17054.41	18896.28	1841.88	17054.41

Voorbeeld 2

3.12 Voorbeeldprogramma's

Om een geldbedrag van netto f. 10000.00
te lenen moet er een lening van
f. 10101.01 afgesloten worden.
Het disagio bedraagt f. 0.00.
De administratiekosten zijn f. 101.01.
bij een rentevoet van 8.0% en een
aflossingspercentage van 10.0%.
In totaal wordt daarbij f. 13902.40
aan premie betaald. Dit komt overeen
met LENING * 1.3902.
Hierbij is f. 3801.30 aan rente
inbegrepen.

ready

Voorbeeld 1

Om een geldlening van netto f. 15000.00
te lenen moet er een lening van
f. 15463.92 afgesloten worden.
Het disagio bedraagt f. 309.28.
De administratiekosten zijn f. 154.64.
bij een rentevoet van 10.8% en een
aflossingspercentage van 3.0%.
In totaal wordt daarbij f. 31765.92
aan premie betaald. Dit komt overeen
met LENING* 2.1177.
Hierbij is f. 16302.00 aan rente
inbegrepen.

ready.

Voorbeeld 2

Het programma maakt gebruik van de volgende variabelen:

Disagio	:	disagiopercentage (ca. 1% per jaar)
AdmKosten	:	administratiekosten (%)
Rente	:	rentepercentage
Aflossing	:	aflossingspercentage
OudSaldo	:	vorige saldo
NieuwSaldo	:	nieuwe saldo
Lening	:	te lenen bedrag
PerBedrag	:	te betalen premie per periode
LenTotaal	:	geleende bedrag
R	:	betaalde rente per periode
Afl	:	betaalde aflossing per periode
RTotaal	:	totale rente
PrTotaal	:	totaal betaalde premie
RentePer	:	renteperiode (jaren)

Na het invoeren van alle gegevens bepaalt het programma het totaal benodigde bedrag door de administratiekosten en disagio bij het benodigde bedrag op te tellen. Hiertoe dient de formule:
OudSaldo:=(Lening*100/(100-Disagio-AdmKosten))
De te betalen premie per periode kan als volgt gevonden worden:
PerBedrag:=OudSaldo*(Rente+Aflossing)/100
Ook de betaalde rente is te berekenen, namelijk
R:=OudSaldo*Rente/100
Uit deze twee waardes volgt nu de aflossing, deze is namelijk gelijk aan premie-rente, dus
Afl:=PerBedrag-R

Hiermee is dan het nieuwe saldo te berekenen, dit is namelijk gelijk aan het oude saldo min de aflossing:
NieuwSaldo:=OudSaldo-Afl

Nadat voor iedere periode de bedragen zijn afgedrukt dient er op een toets gedrukt te worden. De computer vertelt dan nog wat er nu eigenlijk precies betaald is (zie eerder voorbeeld).

Bovenstaande listing dient uiteraard onder Oxford Pascal ingetypt te worden en kan met het r-commando gecompileerd en gerund worden. De regelnummers zijn niet belangrijk, deze worden door Oxford Pascal na het SAVEn automatisch geRENUMBERd.

3.12 Voorbeeldprogramma's

```
function Kvar
  Disagio,AdmKosten,Rente,Aflossing : real;
  OudSaldo,NieuwSaldo,Lening,PerBedrag : real;
  LenTotaal,R,Afl : real;
  RTotaal,PrTotaal : real;
  rentePeriode : integer;
(* *)
procedure ClrScr;
begin
  page;
end;
function KeyPressed:boolean;
begin
  KeyPressed:=peek(198)<>0;
end;
procedure KillKbd;
begin
  poke(198,0);
end;
procedure ZetKop;
begin
  writeln(' Saldo   Premie   Rente  Aflossing');
  writeln('----------------------------------');
end;
begin
  ClrScr;
  writeln('Aflossingsschema');
  writeln('----------------');
  write('Geef de hoogte van het bedrag: ');
  readln(Lening);
  write('Wat is het disagio? ');
  readln(Disagio);
  write('Administratiekosten (%): ');
  readln(AdmKosten);
  OudSaldo:=(Lening*100/(100-Disagio-AdmKosten));
  LenTotaal:=OudSaldo;
  write('Rente in %: ');
  readln(Rente);
  write('Aflossingspercentage: ');
  readln(Aflossing);
  ClrScr;
  ZetKop;
  PerBedrag:=OudSaldo*(Rente+Aflossing)/100;
  PrTotaal:=0.0;
  RTotaal:=0.0;
  RentePeriode:=0;
  repeat
    RentePeriode:=RentePeriode+1;
    R:=OudSaldo*Rente/100;
    Afl:=PerBedrag-R;
```

3.12 Voorbeeldprogramma's

```
    NieuwSaldo:=OudSaldo-Afl;
    if NieuwSaldo<0.0 then
    begin
      Afl:=Afl+NieuwSaldo;
      PerBedrag:=R+Afl;
      NieuwSaldo:=0;
    end;
    PrTotaal:=PrTotaal+PerBedrag;
    RTotaal:=RTotaal+R;
    write(OudSaldo:6:2,PerBedrag:6:2,R:5:2);
    writeln(Afl:6:2);
    if RentePeriode mod 20=0 then
    begin
      write('Druk op een toets...');
      repeat until KeyPressed;
      KillKbd;
      ClrScr;
      ZetKop;
    end;
    OudSaldo:=NieuwSaldo;
  until OudSaldo<=0;
  repeat until KeyPressed;
  KillKbd;
  ClrScr;
  writeln('Om een geldbedrag van netto f',Lening:6:2);
  writeln('te lenen moet er een lening van');
  writeln('f',LenTotaal:6:2,' afgesloten worden.');
  writeln('Het disagio bedraagt f',LenTotaal*Disagio/100:4:2,'.');
  write('De administratiekosten zijn f ',LenTotaal*AdmKosten/100:4:2);
  writeln('.');
  writeln('De lening wordt in',RentePeriode:3,' jaar afgelost');
  writeln('bij een rentevoet van',Rente:2:1,' % en een');
  writeln('aflossingspercentage van',Aflossing:2:1,' %.');
  writeln('In totaal wordt daarbij f',PrTotaal:6:2);
  writeln('aan premie betaald. Dit komt overeen');
  writeln('met LENING *',PrTotaal/Lening:1:4,'.');
  writeln('Hierbij is f',RTotaal:6:2,' aan rente');
  writeln('inbegrepen.');
end.
```

8/3.13

Netwerkplanning

Als eerste voorbeeld van een programma in de taal PASCAL gaan we een programmasysteem over netwerkplanning uitwerken. Omdat de PASCAL-cursus zich nog in beginstadium bevindt, zullen we geen programma met alle mogelijke slimme foefjes en trucjes uitwerken maar gaan we ons meer richten op de beginnende PASCAL-programmeur. U moet dit programma over netwerkplanning zien als een groot voorbeeld dat hoort bij de PASCAL-cursus. Het programma is zodanig geschreven dat het met kleine aanpassingen ook overgenomen kan worden door TURBO-PASCAL gebruikers.

Dit hoofdstuk wordt opgedeeld in: het onvermijdelijk gedeelte over de theoretische basisprincipes van netwerkplanning, in het bijzonder de gebruikte netvorm, en verder in een overzichten over: variabelen, hulpprocedures, hoofdprocedures en hoofdprogramma. Later zullen we het programmasysteem uitbreiden met een gedeelte waarmee een grafische voorstelling van het netwerk kan worden gemaakt.

Om plaatsruimte te besparen en ook vanwege de betere programmastructuur van PASCAL hoeven we de afzonderlijke subprogramma's niet te documenteren zoals we dat bij de BASIC-programma's hebben gedaan: hier beperken we ons tot een beschrijving.

Theoretische basisprincipes van netwerkplanning

Netwerkplanning werd in de jaren 50 ter ondersteuning en vereenvoudiging van projectmanagement ontwikkeld. De afkortingen CPM (wat hier staat voor 'critical path method', niet te verwarren met CP/M, het bekende bedrijfssysteem), MPM en PERTH als verschillende vormen van netwerkplanning zijn voor ingewijden in deze techniek zeker bekend. Over netwerkplanning en alles wat daarbij komt kijken valt heel wat te zeggen. Wij beperken we ons tot datgene wat we met betrekking tot ons programmavoorbeeld nodig hebben.

We hebben in het voorafgaande de term projectontwikkeling al gebruikt. Dat klinkt erg abstract, toch is netwerkplanning in de meest uiteenlopende ondernemingen bruikbaar. We kunnen ons bij zo'n project de bouw van een huis voor voorstellen, maar net zo goed de ontwikkeling van een complex programmasysteem. Nemen we als voorbeeld de bouw van een huis. De afzonderlijke werkzaamheden zijn bijvoorbeeld: uitgraven voor de fundering, betonning van de fundering, bouw van de keldermuren, bouw van de verschillende vloeren en muren, bouw van de verdiepingen, aanbrengen van het dak, de afzonderlijke vensters, sanitair enz. Elk van deze onderdelen kan ook

weer als een afzonderlijk project worden gezien.

Over de bouwplaats kan een dak worden gezet, erg zinvol is dat niet. Moet het een goed en stevig huis worden dan zijn de afzonderlijke activiteiten van elkaar afhankelijk. De fundamenten kunnen pas betond worden als de bouwplaats is uitgegraven en er eventueel palen zijn geslagen; het dak kan pas geplaatst worden als de muren van de bovenste verdieping klaar zijn. Hoe gedetailleerd zo'n plan is hangt van het type project af. De installatie van het sanitair kan ook weer worden opgedeeld, bad en wastafels aansluiten en dergelijke evenzo kan dat met de elektrische installatie. De activiteiten en de logische volgorde waarin die moeten worden uitgevoerd worden in een netwerk weergegeven.

Er zijn verschillende typen netwerken. Er zijn er waarin de activiteiten door pijlen worden weergegeven en waarin knooppunten de logische verbindingen voorstellen. Verder zijn er netwerken waarin de activiteiten door knooppunten worden voorgesteld. In ons voorbeeld beperken we ons tot een z.g. volgtijd-knooppuntennet. In dit net worden de activiteiten - met de daarbij behorende gegevens - in knooppunten vastgelegd; de logische verbindingen tussen de knooppunten worden door pijlen weergegeven.

Omdat ons programma geen netwerkplan, zoals dat bijvoorbeeld bij de ontwikkeling van een gevechtsvliegtuig wordt gebruikt, hoeft door te rekenen, zullen we enige restricties aanbrengen. Degene die grotere projecten wil uitvoeren kan het programma heel makkelijke zelf uitbreiden.

Nu de gegevens die we in een knooppunt opnemen. Voor de eenvoud nummeren we de afzonderlijke knooppunten. Het beginknooppunt zal steeds nummer 1 zijn. Vervolgens de beschrijving van de activiteit zoals: grond bouw rijp maken of probleemanalyse voor een computerprogramma. In wat nu volgt gaan we een voorbeeld uitwerken waarbij een redelijk groot programmeerproject in zijn eenvoudigste vorm door een netwerk kan worden voorgesteld.

Goed beschouwd gaat het bij projectplanning met netwerken om zuiver tijdsafhankelijke werkzaamheden. Materiaalplanning blijft buiten beschouwing. Bij de uitvoering van een project in bepaalde tijdstermijnen, iets dat bij projecten in de open lucht maar zelden het geval is, toont een netwerkplan zijn waarde. Vandaar dat we van een activiteit de duur en het aantal keren dat die activiteit voorkomt vastleggen. Bij het bouwen van een huis moet de grond maar één keer worden uitgegraven iets wat bijvoorbeeld 2 dagen duurt. Het inzetten van kozijnen duurt per stuk bijvoorbeeld twee uur. Dat laatste moet een aantal keren gebeuren. In het voorbeeld van het programmeerproject: de probleemanalyse hoeft maar één keer te worden uitgevoerd en duurt, nemen we aan, vijf dagen.

In de regel wordt bij het opgeven van de duur van een activiteit geen tijdseenheid vermeld. Alle tijden hebben betrekking op dezelfde eenheid. Bij kleine objecten zijn dat uren of dagen, bij grotere weken of maanden. Als de tijd wordt opgegeven in dagen is het meestal niet van belang of een activiteit vroeg of laat op die dag klaar is. Overigens komt het maar zelden voor dat een project voor de volle 100 % volgens het netwerk wordt uitgevoerd. Over 'tijden' in het netwerk later meer.

Vervolgens moeten we van elk knooppunt natuurlijk weten welke activiteiten voor dat

3.13 Netwerkplanning

knooppunt komen en welke er na. Om het programma niet te ingewikkeld te maken gaan we ervan uit dat elk knooppunt maximaal drie voorgangers en ook maximaal drie opvolgers heeft; dit laatste geeft geen wezenlijke beperking zoals we later aan de hand van een voorbeeld nog zullen zien.

Normaal gesproken is de duur van de activiteiten tijdens het project nog te beïnvloeden. Welke gegevens u verdere nog bij de knooppunten opneemt is meestal van het project afhankelijk, de genoemde gegevens blijken in de praktijk voldoende te zijn. Als de duur en het aantal malen dat een activiteit voorkomt beïnvloed kan worden door bijvoorbeeld inzetten van extra mankracht, zullen later de gegevens bij de knooppunten nog veranderen. Vaak worden bij zogenaamde kritische activiteiten en bij vertraging nog een analyse opgesteld, iets wat enkele duizenden guldens kan kosten Doel van de analyse is na te gaan of bij uitvoering met hulp van extra mankracht of door andere veranderingen nog tijd kan worden bespaard. Meestal gebeurt dit alleen bij projecten die jaren duren zoals bij militaire projecten (gevechtsvliegtuigen) of in de civiele bouw (neerzetten van flats of ontwikkeling van een nieuwe auto).

Bekijken we nu een concreet, te overzien voorbeeld.

We gaan eerst de grove structuur van een programmeerproject in een netwerk weergeven. Vóór we dat kunnen moeten we eerst bepalen uit welke activiteiten het project bestaat. Om het voorbeeld niet te groot te laten worden beperken we ons tot de belangrijkste activiteiten. We beginnen natuurlijk met de probleemanalyse, waarvoor we 5 tijdseenheden reserveren en dat natuurlijk ook maar een keer wordt gedaan. Vervolgens moeten voor het programma stroomschema's en structuurdiagrammen worden opgesteld, waarvoor we een duur van 4 tijdseenheden nemen en natuurlijk ook weer één keer uitvoeren. Tegelijkertijd kunnen de beeldschermmaskers en de databanken worden ontworpen, waarbij we ervan uitgaan dat door de probleemanalyse het concept daarvan al beschikbaar is. Voor elk beeldschermmasker nemen we 1 tijdseenheid, we moeten er in totaal 8 opstellen. Het opzetten van de databanken duurt iets langer, we nemen daarvoor 2 tijdseenheden, er zijn er 6. Zijn de databanken klaar dan kan meteen het programma voor het afdrukken worden geschreven. Het zijn 9 verschillende programma's (procedures) die elk in 3 tijdseenheden worden uitgevoerd. (Het testen van de lijsten duurt vrij lang.)

Tegelijkertijd met het ontwerpen van beeldschermmaskers en databanken kan het raamprogramma worden geschreven, geschatte programmeertijd 12 tijdseenheden. Is het raamprogramma met de beeldschermmaskers en databanken klaar dat kan een test van het hele systeem worden uitgevoerd (7 tijdseenheden). Ook kan tegelijkertijd met het testen na de probleemanalyse de documentatie van het programma (9 tijdseenheden) en het handboek voor de gebruiker (8 tijdseenheden) geschreven worden, als we er tenminste van uit mogen gaan dat de verschillende werkzaamheden elkaar wederzijds niet beïnvloeden.

In het voorbeeld dat we hier bespreken gaan we er van uit dat een volgende activiteit pas kan beginnen wanneer alle voorgaande activiteiten zijn afgerond. Dit is de praktijk bijna altijd het geval. Er zijn natuurlijk ook activiteiten waarmee reeds een begin kan worden gemaakt als de voorgaande activiteiten nog niet of niet helemaal zijn afgesloten. Bij-

3.13 Netwerkplanning

voorbeeld kan in een bouwwerk de verwarming op de eerste etage al gemonteerd worden terwijl de muren van de tweede etage nog moeten worden opgetrokken. In ons voorbeeld, en ook in het programma, beperken we ons tot de situatie waarin de werkzaamheden in de gewone volgorde worden uitgevoerd. De gegevens zoals we die hebben besproken kunnen als volgt in een tabel worden samengevat:

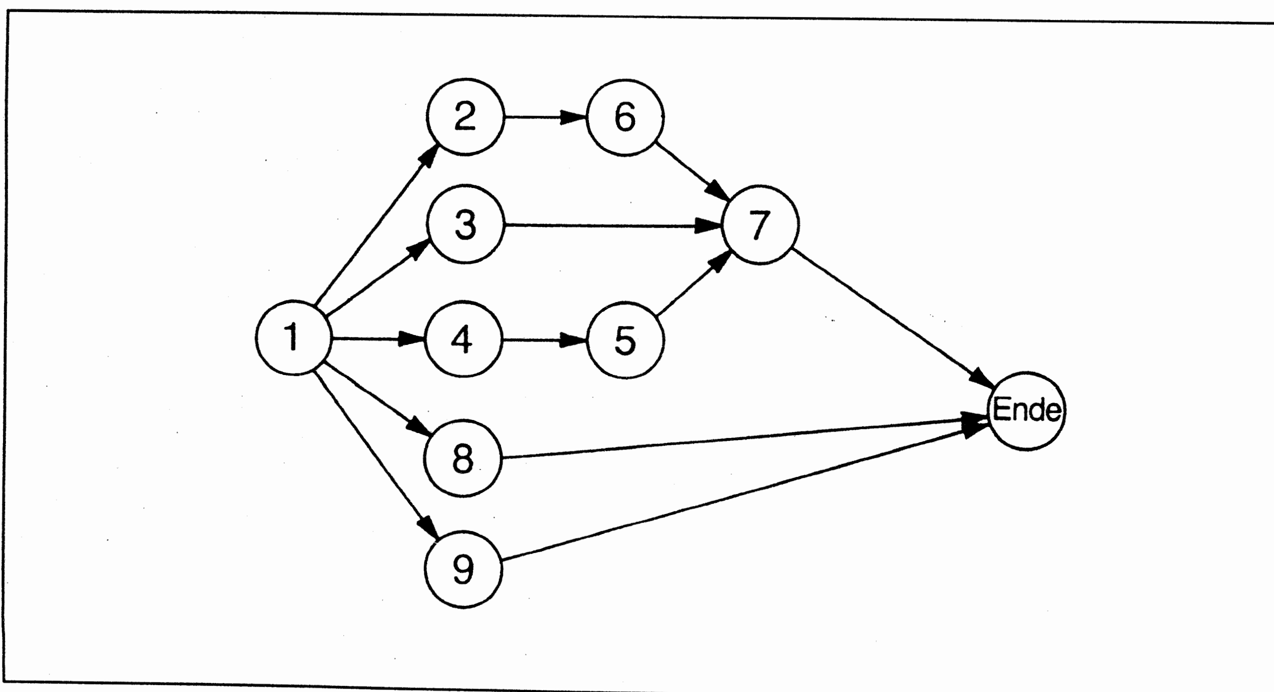
nr.	omschrijving	tijd	voork.
1.	probleemanalyse	5	1
2.	diagrammen opstellen	4	1
3.	beeldschermmaskers	1	8
4.	bestanden aanmaken	2	6
5.	procedures	3	9
6.	raamprogramma	12	1
7.	systeemtest	7	1
8.	documenteren van systeem	9	1
9.	gebruikershandleiding	8	1

Omdat we de activiteit per knooppunt reeds op een rijtje hebben gezet kunnen we nu ook het netwerk tekenen.

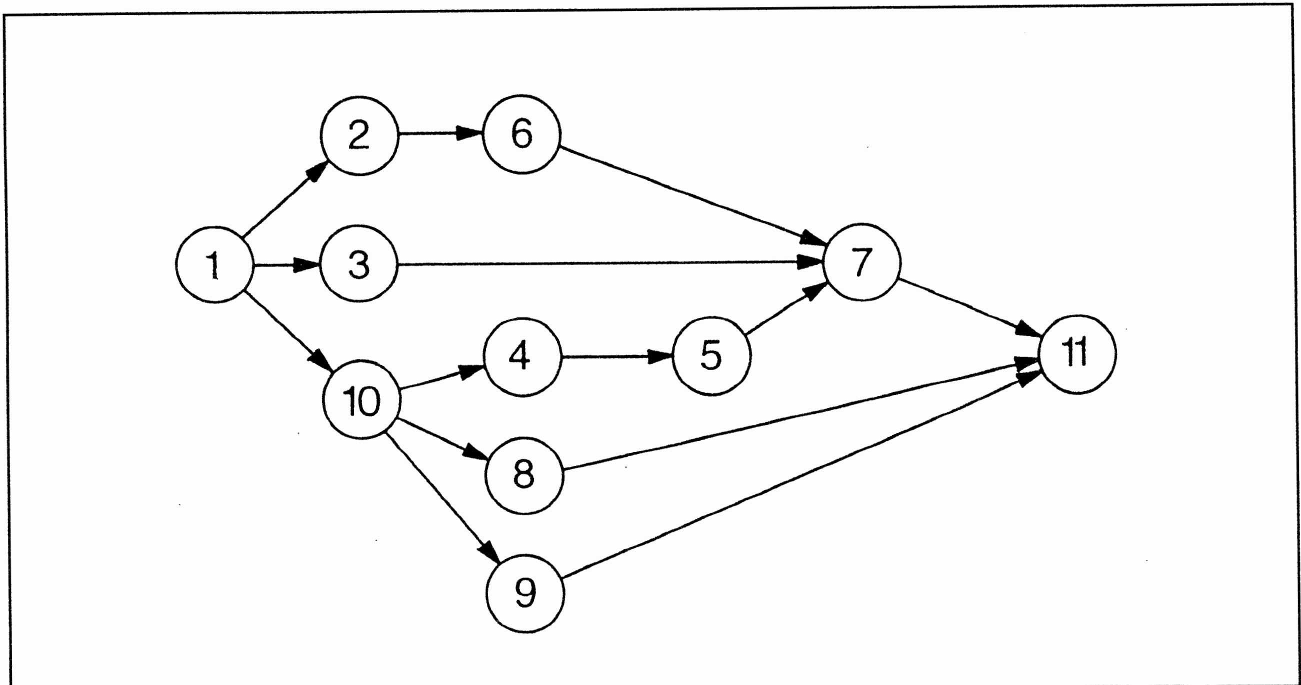
Activiteit 1 (probleemanalyse) heeft als vervolgactiviteiten de nr's 2, 3, 4, 8 en 9; knooppunten 2 en 6 hebben de vervolgactiviteiten 6, resp. 7 enz. Anderzijds heeft activiteit 7 als voorgangers de activiteiten 6, 3 en 5.

Er doet zich nog het volgende probleem voor: een van de uitgangspunten voor het programma was: elk knooppunt zou niet meer dan 3 vervolg en/of voorgaande activiteiten hebben. Aan deze voorwaarde is in bovenstaand netwerk niet voldaan omdat knooppunt 1 verwijst naar 5 vervolgactiviteiten. We lossen dit als volgt op: We voeren een zogenaamd schijnknooppunt in; dit knooppunt heeft en geen tijdsduur en natuurlijk ook is het aantal keren voorkomen niet gedefinieerd. In figuur 8/3.13-2 is het schijnknooppunt ingetekend en is ook meteen de functie ervan duidelijk.

We hebben knooppunt 10 als vervolgknooppunt van 1 ingevoerd, knooppunt 10 heeft als vervolg 4, 8 en 9. Dus heeft knooppunt 1 nu nog maar 3 vervolgactiviteiten. Het "eind-



Figuur 8/3.13-1: Uitgangsnetwerk



Figuur 8/3.13-2: Netwerk met schijnknooppunt

knooppunt" hebben we ook een nummer gegeven (nr. 11). Omdat de computer later op een of andere manier moet vaststellen welk knooppunt moet worden bereikt, wat dus inhoud dat het project is afgerond, bepalen we nu dat de aanduiding van het eindknooppunt "klaar" moet zijn. Dit kan de computer heel makkelijk testen.

Berekeningen

Het opstellen van het netwerk is natuurlijk geen doel op zich; het is iets waaraan we het een en ander willen berekenen. Om te beginnen willen we weten hoelang het totale project gaat duren en wat betreft een knooppunt: wat is het 'vroegst mogelijk voltooïngs- of eindtijdstip', afgekort VME-tijdstip. Om tot het laatste knooppunt '11' te komen moeten we het hele netwerk doorlopen, dus beginnen bij knooppunt 1. Deze rekenmethode heet daarom voorwaartse rekenwijze. Elk knooppunt heeft behalve zijn eigen vroegst mogelijk eindtijdstip, ook zijn vroegst mogelijk begin tijdstip, afgekort VMB-tijdstip. Omdat de totale tijdsduur van een activiteit bepaald wordt door de tijdsduur van de activiteit zelf en het aantal keren dat die activiteit voorkomt kunnen we de VME-tijd van een knooppunt berekenen door bij de VMB-tijd de totale tijdsduur op te tellen.

Omdat we in ons geval alleen met een gewone volgorde te maken hebben is het vroegst mogelijke eindtijdstip van een knooppunt tevens de vroegst mogelijke begintijdstip van al de vervolgactiviteiten. In het schijnknooppunt 10 en doel knooppunt 11 is de VMB-tijd steeds gelijk aan de VME-tijd, omdat deze activiteiten geen tijdsduur hebben.

Bekijken we knooppunt 7, dan stellen we vast dat de activiteiten 3, 5 en 6 aan activiteit 7 voorafgaan. Elk van de knooppunten 3, 5

en 6 heeft een 'vroegst mogelijk eindtijdstip', dat het 'vroegst mogelijk begintijdstip' van knooppunt 7 kan zijn. Nu kan voor de berekening natuurlijk niet de laatst berekende VME-tijd van de activiteiten die aan knooppunt 7 voorafgaan worden genomen. Activiteit 7 kan bij de gewone voortgang pas beginnen als àl zijn voorgaande activiteiten beëindigd zijn. Heeft een knooppunt dus meerdere voorgangers dan is zijn vroegst mogelijk begintijdstip de grootste waarde van alle vroegst mogelijke eindtijdstippen van zijn voorgangers. In ons voorbeeld krijgen we dan voor de VMB- en VME-tijden de volgende waarden:

Nr.	VMB	VME
1	0	5
2	5	9
3	5	13
4	5	17
5	17	44
6	9	21
7	44	51
8	5	14
9	5	13
10	5	5
11	51	51

Klaarblijkelijk wordt de VMB-tijd van knooppunt 7 van de VME-tijd van knooppunt 5 (44) afgetrokken en bij knooppunt 11 (die eveneens drie voorgangers heeft) de grootste waarde van zijn voorgangers (7:51; 8:14; 9:13).

Ons hele project wordt dus na 51 tijdseenheden afgesloten. Nemen we weer de bouw van het huis als voorbeeld dan kunnen we nu aan alle vakmensen opgeven wanneer ze op zijn vroegst met hun werk kunnen beginnen. In ons voorbeeld kan de programmeur van het raamprogramma verteld worden wanneer hij/zij andere werkzaamheden kan verrichten of een paar dagen vakantie kan opnemen, voorzover hij/zij natuurlijk niet betrokken is bij de andere activiteiten van het project.

Omdat een sommige activiteiten meer tijdseenheden in beslag nemen dan anderen die parallel worden uitgevoerd, geeft dat voor een paar knooppunten een zekere speling of speelruimte. Laat men activiteit 6 pas na 15 tijdseenheden beginnen, dan wordt die natuurlijk pas na 27 tijdseenheden beëindigd; de totale tijdsduur van het project zal daardoor niet veranderen om dat activiteit die volgt op 6 pas na 44 tijdseenheden kan beginnen.

Uit het voorgaande volgt meteen de vraag wanneer een activiteit op zijn laatst mag beginnen zó dat de kortste projectduur niet verandert. Om dit te beantwoorden gaan we terugwaarts rekenen. Voor elk knooppunt moet nu berekend worden het 'laatst toelaatbaar begintijdstip' en het 'laatst toelaatbaar eindtijdstip'. Deze tijden zullen we afkorten door LTB- resp. LTE-tijd. Deze methode gaat op de zelfde manier als het voorwaarts rekenen, maar nu beginnen we natuurlijk bij het eindpunt van het project.

Voor de laatst toelaatbaar eindtijdstip van het totale project kunnen we niet de waarde "0" nemen. Logisch is dat daarvoor het vroegst mogelijk eindtijdstip (VME) wordt genomen. Dus heeft de LTE-tijd van knooppunt 11 in ons voorbeeld de waarde 51 en de LTB-tijd van een knooppunt wordt over het algemeen berekend door van het laatst toelaatbaar eindtijdstip de totale tijdsduur van de voorgaande activiteiten af te trekken.

Hebben we bij het voorwaartsrekenen alleen de rekening gehouden met de volgende activiteiten, nu moeten we bij het terugwaartsrekenen de voorgaande activiteiten betrek-

ken. Ook gebruiken we niet begintijdstippen maar de eindtijdstippen van de voorgaande knooppunten. Uitgaande van knooppunt 11 voeren we als LTE-tijd bij de knooppunten 7, 8 en 9 de waarde 51 in en berekenen uit de duur telkens de LTB-tijd. Bij knooppunten 8, 9 en 7 wordt dat resp. 42, 43 en 44. Bij knooppunt 7 hebben de VMB-en LTB-tijden, maar ook de VME- en LTE-tijden dezelfde waarden. Met dit knooppunt is nog iets bijzonders, daarop komen we later terug.

Na het terugwaarts rekenen hebben we de volgende gegevens:

Nr.	LTB	LTE
1	0	5
2	28	32
3	36	44
4	5	17
5	17	44
6	32	44
7	44	51
8	42	51
9	43	51
10	5	5
11	51	51

Bij het voorwaartsrekenen hebben we steeds voor het vroegst mogelijk begintijdstip de grootste waarde van de voorgangers genomen, dus gaan we bij het terugwaarts rekenen voor de laatst toelaatbaar eindtijdstip van een knooppunt de kleinste waarde van de laatst toelaatbare begintijdstip van de opvolgers nemen. We kunnen dit uit de gegevens van de knooppunten 1, 4, 5, 7 en 10 opmaken.

Omdat de begin- en eindtijdstippen van een knooppunt altijd een verschil in de totale tijdsduur uitmaken is ook het verschil tussen VMB- en LTB-tijd enerzijds, en VME- en LTE-tijd anderzijds, altijd gelijk. Dit verschil wordt speelruimte of speling genoemd.

De knooppunten 1, 4, 5, 7 en 10 hebben geen speelruimte. Een vertraging in de activiteiten daarvan geeft automatische een verlenging van de totale projecttijd. De weg van 1 naar 11 via 10,4,5 en 7 wordt ook wel het "kritieke pad" genoemd.

De grootste hoeveelheid werk is klaarblijkelijk de activiteit van knooppunt 5 met 27 tijdseenheden. Omdat hier 9 verschillende programma's/procedures geschreven moeten worden geeft dit knooppunt mogelijkheden heeft om de totale projecttijd te bekorten. Mogelijke is, weliswaar wat overdreven, voor elk programma een aparte programmeur aan te stellen. Omdat het aantal keren dat een bepaald werk voorkomt in principe niet direct betrekking heeft op de activiteit zelf, maar op de duur ervan, wordt het aantal keren voorkomen bij inzetten van 9 programmeurs dan ook 1. De duur van het knooppunt duurt dan nog maar 3 tijdseenheden. Omdat het vroegst mogelijke begintijdstip van activiteit 5 bij 17 tijdseenheden ligt, kan het ook bij 20 tijdseenheden worden afgesloten. Activiteit 7 begint dan niet meer bij tijdseenheid 44, maar ook niet bij tijdseenheid 20 omdat activiteit 6 pas bij tijdseenheid 21 wordt beëindigd. Door deze verandering wordt het kritieke pad naar de activiteiten 1,2,6,7 en 11 verplaatst.

Ook het andere geval is denkbaar n.l. dat een activiteit, dat niet op het kritieke pad ligt, een zo grote vertraging oploopt, dat daardoor het kritieke pad via deze activiteit gaat lopen. Nemen we bijvoorbeeld aan dat voor activiteit 6 niet 12 maar 42 tijdseenheden nodig zijn, dan wordt deze activiteit op zijn vroegst na 51 tijdseenheden beëindigd. Het kritieke

3.13 Netwerkplanning

pad zal dan over 1,2,6,7 en 11 lopen zonder dat de tijdsduur van activiteit 5 werd bekort.

Juist bij grote projecten ontstaan door ziekte, slecht weer of ontbreken van materiaal en dergelijke verschuivingen in het tijdsverloop. Een netwerk opstellen om alleen een paar tijdstippen uit te rekenen komt neer op: met een kanon naar een mug schieten. In situaties als deze, komen de voordelen van het gebruik van computer duidelijk naar voren omdat bij een veranderingen in één activiteit meteen de nieuwe gegevens, in het bijzonder het kritieke pad, kan worden aangegeven. Uit de snelle en precieze berekening (reken maar eens met de hand een wat groter netwerk door!) kan de computer de verkregen resultaten ook op verschillende manieren weergeven en tevens zijn de nieuwste gegevens permanent beschikbaar.

Naast de genoemde speelruimte (ook totale speelruimte genoemd), bestaat er ook een z.g. vrij speelruimte, dat aangeeft hoeveel tijdseenheden een activiteit t.o.v. zijn vroegste begintijdstip verschoven kan worden, zonder het vroegste begintijdstip van de volgende activiteit te beïnvloeden. Deze vrije speelruimte is altijd gelijk aan of kleiner dan de totale speelruimte.

Tot zover voor wat de theorie betreft, laten we ons verder met programmeren in PASCAL bezighouden. Eerst nog iets over programmeren in het algemeen. Overzichtelijke programma's verkrijgt men als de afzonderlijke taken in procedures, die men van mnemonische namen voorziet, worden opgenomen. Over het algemeen moet men het hoofdprogramma zo kort mogelijk houden, zoals we ook in het voorbeeld zullen doen. De afzonderlijke menu-opties van het hoofdmenu worden eveneens door procedures uitgevoerd, die procedures kunnen na een paar voorbereidingen worden opgeroepen. De onderdelen die voor de verschillende hoofdprocedures beschikbaar moeten zijn, of van algemene betekenis zijn, zullen we in het gedeelte over hulpprocedures bespreken. We doen dat volgens de "bottum-up"-methode hetgeen betekent dat we eerst de hulpprocedures zullen uitwerken, vervolgens de hoofdprocedures en tot slot het hoofdprogramma.

Een overzicht van variabelen en procedures
Voordat het programmeren zelf aan de orde komt moeten we ons eerst nog bezig houden met de vraag welke variabelen en welke variabelentypen we in dit geval nodig hebben en in het totale programma moeten kunnen gebruiken.

Omdat we de gegevens (data) van het netwerk, dat bestaat uit de gegevens van de afzonderlijke knooppunten, op een diskette willen opslaan is het zinvol een recordtype te definiëren dat per record alle gegevens van een knooppunt bevat. Voor de eenvoud geven we nu eerst het volledig declaratiedeel; constante-definities, type-definities en variabelen-definities.

```
1   program netwerk;
2    const
4     penkleur = 7;             (* voorgrondkleur:       geel    *)
5     achtgrkl = 0;             (* achtergrond kleur:    zwart   *)
6     randkleu = 6;             (* randkleur:            blauw   *)
7     printadr = 4;             (* printadres:           4       *)
8     printsec = 0;             (* print second. adres:  0       *)
9
```

3.13 Netwerkplanning

```
10  type string    = packed array[1..26] of char;
11       knooppunt = record
12                        nummer: integer;
13                        aktivit:string;
14                        duur,
15                        aantalx,
16                        volger1,
17                        volger2,
18                        volger3,
19                        voorgang1,
20                        voorgang2,
21                        voorgang3,
22                        vmb,
23                        vme,
24                        ltb,
25                        lte:       integer;
26                        opmerking:string;
27                     end;
28
29  var
30    aktmenu:      integer;
31    aktnummer:    integer;
32    bestand:      string;
33    berekend:     boolean;
34    jn:           char;
35    laatste:      integer;
36    mo:           integer;
37    net:          array [0..50] of knooppunt;
38    lst:          text;
39    netbestan:    file of knooppunt;
40    leeg,
41    eind1,
42    eind2:       string;
43
```

Op de programmadefinitie volgt eerst het declaratiedeel voor de constanten. Voor het programma zelf zijn deze constanten niet van betekenis, ze dienen o.a. voor de layout van het schermbeeld. Zoals uit de toelichtingen blijkt zijn de constanten voor het instellen van de kleur van het schermbeeld en het aansturen van de printer. Op deze manier is het heel makkelijk de parameters, indien nodig, aan te passen. De constanten zelf worden later aan het begin van het hoofdprogramma gebruikt. Door het gebruik van mnemonische namen (de betekenis blijkt uit de naam zelf) hoeven we op de elementen van het gegevensrecord "knooppunt" niet meer in te gaan. Behalve de naam en de extra opmerking die bij een knooppunt kunnen worden opgenomen, zijn alle variabelen integers. Tijdsduren kunnen, zoals in dit soort programma's gebruikelijk is, alleen in gehele tijdseenheden worden opgenomen. Het gedeclareerde type 'String' is een type dat

met een tekstvariabele gedefinieerd kan worden.

Nog een kleine toelichting op de variabelen: de variabele 'aktmenu' is voor de gekozen menu-optie; de variabele 'aktnummer' voor het nummer van het knooppunt dat op een bepaald moment wordt bewerkt.

'bestand' is voor de naam van een bestandsnaam en wordt later aan de bestandsnaam 'netbestand' toegewezen. 'i' is een hulpvariabele voor diverse doeleinden en 'jn' is gereserveerd voor het testen van de invoer van één enkele letter op vragen die alleen met 'ja' of 'nee' beantwoord kunnen worden. 'mo' is een hulpvariabele voor het nummer van de menu-optie.

Het netwerk zelf wordt in het veld 'net' opgeslagen waarbij het veld van het type knooppunt is. Alle gegevens kunnen dus met één index worden opgeroepen. In ons geval is de maximale omvang van het netwerk beperkt tot 50 (het knooppunt 0 wordt niet gebruikt en is bedoeld voor testen en dergelijke). Het maximaal aantal knooppunten kan makkelijk gewijzigd worden. 'netbestan' tenslotte is het record voor de knooppunten; waarvan de naam in 'bestand' staat.

We zijn er toe overgegaan het netwerk in een veld te definiëren. Bij grotere netwerken is het nu mogelijk om, na een kleine aanpassing van het programma, de afzonderlijke knooppunten, nadat daarin iets is veranderd, meteen naar de diskette weg te schrijven. Dit komt de beveiliging van de data zeker ten goede.

De procedures

Het programma begint met de naam (netwerk) en dan volgt het declaratiedeel. In de volgorde waarin we de programmadelen beschrijven (en dan meteen ook de listings geven) kunt u ze ook één voor één via de editor invoeren. Zou u de volgorde van de procedures willen veranderen, dan moet u er aan denken dat de naam gedefinieerd is zodra die in het programma opduikt. Vandaar dat alle hulpprocedures meteen aan het begin van het programma staan.

In het bijzonder worden bij de hulpprocedures niet alleen gegevens binnen de procedure gebruikt maar wordt ook verwezen naar globale variabelen, vandaar eerst een korte uiteenzetting over de parameters die in de afzonderlijke procedures gebruikt worden.

Hier volgt een opsomming van de diverse hulpprocedures met de daarin gebruikte parameters:

kop — Actueel menu-optie van type string wordt in de procedure uitgegeven.

menu — Functieprocedure, die het nummer van de gekozen optie genereert (integer); 'mo' fungeert hier als globale variabele.

masker — Hierin worden geen parameters gebruikt omdat er een vast gedefinieerde beeldschermlayout is.

volgnr — Functieprocedure die het nummer van een knooppunt genereert (integer).

netladen — Geen parameters. De array 'net' en de verschillende andere variabelen worden als globale variabelen aangeroepen.

netopslag Geen parameters zie 'volgnr'.

tijdopnul Geen parameters, de overeenkomstige invoer van de velden wordt globaal behandeld.

eindknoop Functieprocedure, die het nummer van de knooppunten met het kenmerk 'klaar' genereert (integer).

toondatkp De invoer is de data van de knooppunten (aktknoop van type knooppunt).

toets Invoerparameter moet hier de regel zijn waarin het bericht 'Met welke toets verder' wordt uitgegeven.

drukregel Als invoerparameter wordt het overeenkomstige element van 'net' ('aktknoop' van type 'knooppunt') aangeroepen.

We zijn nu toe aan het beschrijven van de afzonderlijk procedures.

Hulpprocedures

Het programma wordt geschreven volgens de bottum-up methode, dus moeten nu de programmaonderdelen van het laagste niveau worden geschreven. Op de eerste plaats moeten we bepalen welke programma onderdelen we nodig hebben. Alle delen worden als procedures, in sommige gevallen als functieprocedures, uitgevoerd.

Hulpprocedures voor beeldscherm

Voordat we de hulpprocedures beschrijven eerst een paar korte procedures die, enerzijds het programmeren later wat makkelijker maken, anderzijds een zo goed mogelijke compatibiliteit met TURBO-PASCAL garanderen. TURBO-PASCAL heeft een paar opties, in het bijzonder op het gebied van standaardinstellingen, die OXFORD-PASCAL niet kent.

Die we gebruiken zijn: 'lowvideo', om op negatief beeld over te schakelen, 'normvideo' voor weer terug schakelen naar gewoon beeld, 'clrscr' voor het wissen van het scherm evenals 'gotoxy' om de cursor te sturen. Verder geven we een procedure waarmee de cursor direct kan worden gestuurd. Later in het programma komen we nog een procedure tegen die een lijn (procedure lijn) op het scherm zet.

```
44
45  procedure lowvideo; (* negatief beeld *)
46
47  begin
48    write(chr(18));
49  end; (* lowvideo *)
50
51
52  procedure normvideo; (* gewoon beeld *)
53
54  begin
55    write(chr(146));
56  end; (* normvideo *)
```

3.13 Netwerkplanning

```
57
58
59  procedure clrscr; (* beeldscherm wissen *)
60
61  begin
62    write(chr(147));
63  end; (* clrscr *)
64
65
66  procedure gotoxy(x,y:integer); (* cursor op regel Y kolm X *)
67
68  procedure setcursor; extern $e56c;
69
70  begin
71    poke($d6,y-1);
72    poke($d3,x-1);
73  setcursor;
74  end;
75
```

De procedures 'lowvideo' en 'normvideo' zijn – zoals gezegd – voor de compatibiliteit met TURBO-PASCAL. In die procedures wordt alleen met het write-commando de betreffende stuurcodes uitgegeven, dit is eveneens het geval bij procedure 'clrscr'. Het plaatsen van de cursor wordt door het direct aanspreken van de z.g. zero-page van de C64 bereikt. Zoals bekend, bevindt zich op adres $D3=211 de indexwijzer van de beeldschermkolom en op adres $D6=214 het nummer van de regel waarop de cursor staat.

De eerste, voor het programma belangrijke procedure is die welke de eerste programmaregel toont. Op die regel staat in welk programmaonderdeel of bij welk menu-optie we ons bevinden.

Vervolgens hebben we het hoofdmenu nodig, een beeldschermmasker voor het invoeren, tonen en veranderen van gegevens van de knooppunten, een functie voor het opgeven van het actueel knooppuntnummer en ook nog de procedures voor het laden en opslaan van de netwerkgegevens. Dit zijn procedures die in elk programma met bestandsbeheer voorkomen. Voor ons specifiek probleem hebben we nog de programmaonderdelen nodig voor opzoeken van de laatste knopen en wissen en berekenen van de tijdstippen.

Omdat de gegevens van de knooppunten op verschillende plekken in het programma worden vastgelegd en alle gegevens van een knooppunt op een enkele regel moet worden weergegeven is het niet onverstandig ook hiervoor een aparte procedure te kiezen. Verder is nog een algemene procedure nodig die het bericht 'Druk een toets in' op het scherm zet en dan wacht op een toetsindruk.

Kop - Kopregel neerzetten
Als invoerparameter voor de procedure 'kop' dient de tekst van de actuele menu-optie. De werking van deze procedure zien we meteen als we het schermbeeld in de onderstaande figuur bekijken, daarin wordt meteen ook het hoofdmenu gegeven.

3.13 Netwerkplanning

```
76
77  procedure kop (aktmenu:string); (* opschrift *)
78
79  begin
80    lowvideo;
81    write(chr(19));
82    write (' ':7,aktmenu ,' ':7);
83    normvideo;
84  end; (* kop *)
85
```

```
      Hoofdmenu
1  Knooppunt invoer    :
2  Knooppunt tonen     :
3  Knoopp. veranderen  :
4  Knoopp. verwijderen:
5  Knooppunt weergeven:
6  Tijden berekenen    :
7  Kritiek pad         :
8  Lijsten printen     :
9  Grafische weergave  :
10 Ander netwerk       :
11 Einde               :

Maak uw keuze:
```

Voor de PASCAL-beginners: het PRINT-commando mèt ";"-teken komt overeen met 'write' en een PRINT-commando zonder ";"-teken komt overeen met 'writeln' (write line).

Meteen in het begin hebben we de in het begin al gedefinieerde procedures 'lowvideo' en 'normvideo' nodig.

menu - hoofdmenu neerzetten
Het hoofdmenu, waarin één van de opties moet worden gekozen, is uitgevoerd in de vorm van een functie. De functiewaarde

```
86
87  function menu:integer; (* menu *)
88
89  var mo:integer;
90
91  begin
92   kop('         Hoofdmenu         ');
93   lowvideo;
94   gotoxy(10,5);
95   write(' ':21);
96   gotoxy(10,6);
97   write('1 Knooppunt invoer   :');
98   gotoxy(10,7);
99   write('2 Knooppunt tonen    :');
100  gotoxy(10,8);
101  write('3 Knoopp. veranderen :');
102  gotoxy(10,9);
103  write('4 Knoopp. verwijderen:');
104  gotoxy(10,10);
105  write('5 Knooppunt weergeven:');
```

3.13 Netwerkplanning

```
106  gotoxy(10,11);
107  write('6 Tijden berekenen   :');
108  gotoxy(10,12);
109  write('7 Kritiek pad        :');
110  gotoxy(10,13);
111  write('8 Lijsten printen    :');
112  gotoxy(10,14);
113  write('9 Grafische weergave :');
114  gotoxy(10,15);
115  write('10 Ander netwerk     :');
116  gotoxy(10,16);
117  write('11 Einde             :');
118  gotoxy(10,17);
119  write(' ':21);
120  gotoxy(10,19);
121  write('Maak uw keuze:    ');
122   normvideo;
123
124   repeat
125     gotoxy(29,19);
126     write(' ':5);
127     gotoxy(29,19);
128     readln (mo);
129   until mo in [1..11];
130   menu:=mo;
131 end; (* menu *)
132
```

wordt het nummer van de gekozen optie. Omdat we nog willen testen of een optie op de juiste wijze is gekozen wordt de hulpvariabele 'mp' gebruikt. Is de test positief dan krijgt 'menu' de waarde 'mp'. Vervolgens wordt de procedure 'kop' opgeroepen waardoor de koptekst gecentreerd op het scherm komt.

Ook het menu willen we in negatief beeld weergeven en roepen dus de procedure 'lowvideo' aan. Daarna volgt steeds door opgave van x- en y-positie de positionering van de tekstcursor en wordt het nummer van de optie en bijbehorende tekst op het scherm gezet.

Na een blanco regel komt de opdracht: 'Maak uw keuze' en wordt met 'normvideo' op gewoon beeld teruggeschakeld.

De menupunten geven aan wat we nog allemaal moeten invoeren. Om te beginnen moet mogelijk zijn de gegevens van een knooppunt in te voeren, te tonen en te veranderen en ook het wissen van gegevens van knooppunten moet mogelijk zijn. Verder moeten we de gegevens van de knooppunten in de vorm van een tabel kunnen tonen en afdrukken. Natuurlijk moeten de waarden van VMB, VME, LTB, en LTE worden berekend. Het aangeven van het kritisch pad (welke knooppunten liggen op het kritisch pad, dat is een verkorte knooppuntentabel). Als uitbreiding van het programmasysteem hebben we nog gedacht aan grafisch weer-

geven van het netwerk. Tot slot willen we later nog mogelijk maken van het ene naar het andere netwerk te springen.

In de 'repeat'-lus lezen we telkens een getal in en testen bij 'until' of het getal binnen het aangegeven bereik ligt. Voor elke keus wordt de eerder ingevoerde keus gewist. Tot slot wordt het nummer van de gekozen optie als waarde aan de functieparameter toegekend.

De werking van de procedure 'menu' wordt verder uit de figuur duidelijk.

Lijn - horizontale lijn (40 posities)
Voor we overgaan naar de verschillende menu-opties en de daarbij behorende hulpprocedures, eerst nog een vrij onbelangrijke procedure, maar die we toch vaak zullen gebruiken n.l. het tekenen van een lijn. 'y' is hier een invoerparameter (integer) die bepaalt op welke hoogte de lijn getekend wordt. De variabele 'i' is een gewone hulpparameter voor tellen. Eerst wordt de cursor op de eerste positie van gekozen regel geplaatst en vervolgens wordt 40 maal een grafiekteken (een horizontaal lijntje) gezet. Daarop volgt de regel:

```
POKE(218+y,orb(peek(217+y),128);
```

die regel 'y' tot één enkele regel verklaart. Deze regel is nodig omdat het bedrijfssysteem van de C64 na het zetten van het lijntje op positie 40 de regels als dubbele regels opvat. Het beeldschermmasker werkt dan niet meer zoals we dat willen. Met bovenstaande regel heffen we dit probleem op.

masker - masker voor knooppunt tonen
De procedure 'masker' geeft het beeldschermmasker voor invoeren, tonen en veranderen van de gegevens van knooppunten. Omdat het slechts gaat om het plaatsen van de cursor en tekst op het scherm hoeven we deze procedure niet verder toe te lichten. Het schermbeeld van het masker zien we in onderstaande figuur.

```
   Knoopptnr uitgeven/kiezen
--------------------------------
Nummer :    Duur :  Hoe vaak:
Activiteit  :
--------------------------------
Opvolger 1  :     Opvolger 2  :
Opvolger 3  :
--------------------------------
Voorganger 1 :   Voorganger 2 :
Voorganger 3 :
--------------------------------
VMB :    VME :     LTB :  LTE :
Opmerking :
```

```
133
134 procedure lijn(y:integer); (* lijn tekenen *)
135
136 var i:integer;
137
138 begin
139   gotoxy(1,y);
140   for i:=1 to 40 do write(chr(96));
141   poke(217+y,orb(peek(217+y),128));
142 end; (* lijn *)
143
```

3.13 Netwerkplanning

```
144
145 procedure masker;    (* blanco knooppuntmasker *)
146
147 begin
148   lijn(3);
149   lijn(7);
150   lijn(11);
151   lijn(15);
152   lowvideo;
153   gotoxy(1,4);
154   write ('Nummer :');
155   gotoxy(14,4);
156   write ('Duur   :');
157   gotoxy(27,4);
158   write ('Hoe vaak:');
159   gotoxy(1,5);
160   write ('Aktiviteit  :');
161   gotoxy(1,8);
162   write ('Opvolger 1   :');
163   gotoxy(21,8);
164   write ('Opvolger 2   :');
165   gotoxy(1,9);
166   write ('Opvolger 3   :');
166   gotoxy(1,12);
168   write ('Voorganger 1 :');
169   gotoxy(21,12);
170   write ('Voorganger 2 :');
171   gotoxy(1,13);
172   write ('Voorganger 3 :');
173   gotoxy(1,16);
174   write ('VMB :');
175   gotoxy(12,16);
176   write ('VME :');
177   gotoxy(22,16);
178   write ('LTB :');
179   gotoxy(32,16);
180   write ('LTE :');
181   gotoxy(1,17);
182   write ('Opmerking :');
183   normvideo;
184 end;    (* masker *)
185
```

volgnr - uitgeven/kiezen van knooppuntnr
Omdat zowel bij invoeren, als tonen, als veranderen telkens een knooppuntnummer moet worden uitgegeven (gekozen) en vervolgens een andere taak wordt uitgevoerd, wordt voor het kiezen zelf aparte functie genomen. TURBO-PASCAL doet "moeilijk" als de parameter 'volgnr' meteen als

3.13 Netwerkplanning

```
186
187 function volgnr:integer; (* knooppuntnummer uitgeven/kiezen *)
188
189 var num:integer;
190
191 begin
192   clrscr;
193   kop('Knoopptnr uitgeven/kiezen ');
194   masker;
195   gotoxy(10,4);
196   readln (num);
197   volgnr:=num;
198 end; (* volgnr *)
199
```

argument bij het read-commando wordt gebruikt vandaar dat we een hulpvariabele hebben ingevoerd.

Het zal u zeker zijn opgevallen dat bij de procedure 'masker', het masker zelf wordt uitgegeven zonder dat eerst het scherm is schoongemaakt. Omdat het masker elke keer, dat ook een knooppuntnummer wordt gevraagd, nodig is wordt het schoonmaken van het scherm gestuurd vanuit de procedure 'volgnr'. Daarop aansluitend wordt de kop met de bijbehorende tekst uitgegeven waarna het masker op het scherm wordt gezet. Na het positioneren van de cursor wordt alleen nog het actueel knooppuntnummer bepaald en de waarde daarvan aan de functieparameter toegekend.

netladen - gegevens van een netwerk laden
Bij het laden van de gegevens van een netwerk krijgen we voor de eerste keer in PASCAL te maken met gegevensopslag op een extern geheugen. Vooral beginners in PASCAL moeten aan deze procedure extra aandacht schenken. We moeten steeds in de gaten houden waarom het gaat n.l.: invoeren van de naam van een bestand waarin een netwerk is opgeslagen of nog moet worden opgeslagen.

```
200
201 procedure netladen;    (* gegevens netwerk laden *)
202
203 var i: integer;
204
205 begin
206   kop('      Netwerkbestand      ');
207   gotoxy (1,5);
208   writeln ('Welk bestand wil u bewerken?');
209   write ('==> ');
210   readln  (bestand);
211   for i:=18 downto 3 do
212     bestand[i]:=bestand[i-2];
213   bestand[1]:='0';
```

3.13 Netwerkplanning

```
214   bestand[2]:=':';
215   bestand[19]:=' ';
216   gotoxy(1,8);
217   write('Nieuw bestand? ');
218
219   repeat
220     readln (jn);
221     if jn in ['j','J']
222       then begin
223              rewrite(netbestan,bestand);
224              for i:=0 to 50 do
225              begin
226                with net[i] do begin
227                  nummer      :=i;
228                  aktivit     :=leeg;
229                  duur        :=0;
230                  aantalx     :=0;
231                  volger1     :=0;
232                  volger2     :=0;
233                  volger3     :=0;
234                  voorgang1   :=0;
235                  voorgang2   :=0;
236                  voorgang3   :=0;
237                  vmb         :=0;
238                  vme         :=0;
239                  ltb         :=0;
240                  lte         :=0;
241                  opmerking   :=leeg;
242                end;
243              write(netbestan,net[i]);
244              end;
245              close(netbestan);
246          end;
247 if jn in ['n','N']
248      then begin
249             reset(netbestan,bestand);
250             i:=0;
251             while not eof (netbestan) do
252             begin
253               read(netbestan,net[i]);
254               i:=i+1;
255             end;
256             close(netbestan);
257           end;
258    until jn in ['j','n','J','N'];
259 end;         (* netladen *)
260
```

3.13 Netwerkplanning

Na het inlezen van de bestandsnaam worden in een programmalus alle ingevoerde tekens twee posities naar rechts geschoven waarna '0:' op de eerste twee posities wordt gezet. De strings beginnen dus altijd met '0:' waardoor de disk drive van de C64 wordt geactiveerd. U zult zich misschien afvragen waarom de variabele 'bestand' niet als een 'file of knooppunt' wordt gedeclareerd maar als string zoals uit het declaratiedeel aan het begin van het programma blijkt. Verder zien we daar dat de variabele 'netbestand' als 'file of knooppunt' wordt gedeclareerd. Op deze manier wordt de naam van het bestand onbelangrijk. Met de latere commando's 'rewrite' en 'reset' geven we het programma aan dat we altijd met dat bestand werken waarvan de naam is opgeslagen in de variabele 'bestand', dit laatste gebeurt als we met een bestandscommando 'netbestand' van een bepaalde naam voorzien. Op deze manier wordt 'netbestand' de veralgemening voor een willekeurige 'bestand' waarin gegevens zijn opgeslagen.

Omdat we in de taal PASCAL beslist moeten weten of het gaat om een bestaand bestand of een bestand dat moet worden aangemaakt wordt dat ook eerst gevraagd. De 'repeat'-lus in de regels 219 tot 258 dienen voor het testen van de juiste invoer zodat andere letters dan 'j' of 'n' niet worden geaccepteerd. Gaat het om een nieuw bestand dan wordt het 'if'-blok verwerkt. Met het 'rewrite'-commando treft de diskdrive voorbereidingen het bestand te verwerken en de bestandswijzer wordt daarom aan het begin van het bestand gezet, dus bij het 0'de record. Omdat we het lezen van een gegevensbestand zo eenvoudig mogelijk willen houden en we niet nog eens willen testen of er al record's zijn geschreven en, zoja, hoeveel, gaan we verder met alle records met 'lege'-knooppunten te beschrijven.

Hoewel onze knooppunten pas beginnen te tellen vanaf '1' gaan we toch ook het '0'-de record met een leeg knooppunt beschrijven. Dit record kunnen we later bijvoorbeeld bij verwisselingen van knooppunten of iets dergelijks gebruiken. Behalve dat, maken we het ons gemakkelijker omdat we nu de wijzer van het bestand niet hoeven te positioneren. Het opslaan van het knooppuntnr/recordnr is in principe niet nodig, we doen dit echter toch vanwege de veiligheid. Alle constanten krijgen de waarde '0' in zoverre het gaat om getallen, in het andere geval worden reeksen spaties ingevuld.

Zoals we in de procedure zien, kan bij het 'write'-commando als eerste parameter een bestand worden opgegeven. Als in een bestand geschreven wordt, wordt de bestandswijzer steeds na het schrijven op de volgende component gezet, waardoor het verder positioneren van de wijzer overbodig wordt. Omdat we gelijktijdig met het beschrijven van het bestand alle elementen van het veld 'net' hebben aangepast, hebben we het bestand voorlopig niet nodig en sluiten het daarom met het 'close'-commando. Daarmee wordt ook de directory automatisch aan de nieuwe situatie aangepast. In het geval een nieuw bestand wordt aangelegd wordt niet uit het bestand gelezen, maar wordt die alleen van 'lege' gegevens voorzien en gelijktijdig worden de variabelen voor de knooppunten aangepast. Het 'rewrite'-commando zorgt daarbij voor het openen van het bestand.

Is een gegevensbestand reeds aanwezig dan moet een ander commando voor het openen van het bestand worden genomen namelijk: 'reset'. Wordt met het 'reset'-commando geprobeerd een niet bestaand bestand te openen dan breekt het programma af en wordt de I/O-foutmelding getoond. Dit verklaart

3.13 Netwerkplanning

ook het gebruik van de plausibiliteits-test. De gebruiker moet er zeker van zijn dat bij een bestaand bestand ook de juiste bestandsnaam wordt opgegeven. Als een bestand al bestaat hoeft alleen met een lus en het 'read'-commando een record te worden ingelezen. Net zo als bij het 'write'-commando wordt na het lezen de bestandswijzer op het volgende record gezet.

netopslag - netwerkgegevens wegschrijven
Nieuw aangemaakte- en veranderde gegevens moeten natuurlijk weer naar de diskette worden weggeschreven. Dit gebeurt middels een programmalus.

In principe is het natuurlijk mogelijk elke ingevoerd- of veranderd gegeven weg te schrijven. Dit is in ons geval bij een beperkte aantal gegevens niet nodig. Zowel in de procedure 'netladen' alsook in 'netopslag' wordt het veld 'net' als een globale variabele behandeld.

tijdopnul - tijdstippen op nul zetten
In het theoretisch gedeelte hebben we gezien dat we – om bepaalde uitspraken over het netwerk te kunnen doen – de besproken tijdstippen/tijdsduren moeten kunnen berekenen. Omdat in de betreffende procedure's volgens de recursie-methode gerekend

```
261
262 procedure netopslag; (* netwerk wegschrijven *)
263
264 var i:integer;
265
266 begin
267   rewrite(netbestan,bestand);
268
269   for i:=0 to 50 do
270   begin
271     write(netbestan,net[i]);
272   end;
273
274   close(netbestan);
275 end; (* netopslag *)
276
277
278 procedure tijdopnul; (* tijdenstippen op nul zetten *)
279
280 var i:integer;
281
282 begin
283 for i:=1 to laatste do
284     begin
285        net[i].vmb:=0;
286        net[i].vme:=0;
287        net[i].ltb:=0;
288        net[i].lte:=0;
289      end;
290 end; (* tijdopnul *)
291
```

3.13 Netwerkplanning

```
292
293 function eindknoop:integer; (* laatste knooppunt zoeken *)
294
295 var i:integer;
296
297 begin
298   i:=0;
299   repeat;
300       i:=i+1;
301     until (net[i].aktivit=eind1) or (net[i].aktivit=eind2);
302     eindknoop:=i;
303 end; (* eindknoop *)
304
```

wordt, moeten we er zeker van zijn dat voor het berekenen van bepaalde waarden niet nog verouderde gegevens worden gebruikt. Ook hier kunnen we de variabele 'net' weer als een globale variabele opvatten. De procedure is erg eenvoudig opgebouwd en behoeft geen verdere toelichting.

De variabele 'net' is als – niet expliciet uitgevoerde – invoerparameter op te vatten, de inhoud moet voor het oproepen van de procedure worden ingevoerd. De waarde van 'laatste' wordt door de volgende procedure bepaald.

eindknoop - laatste knooppunt bepalen
We hebben al vastgesteld dat de computer zelf moet uitzoeken hoeveel, van de maximaal 50 knooppunten, voor het actuele netwerk gebruikt gaan worden. Enerzijds voor het wissen van de berekende tijdstippen VMB, VME, LTB en LTE, anderzijds als uitgangspunt voor het terugwaarts rekenen en ook voor de verschillende lijsten moet het nummer van het laatste knooppunt bekend zijn. Het algoritme daarvoor is heel gemakkelijk omdat we bepaald hebben dat het laatste knooppunt het kenmerk 'klaar' zou krijgen. We doorzoeken het netwerk met een 'repeat'-lus tot het kenmerk gevonden is. We gebruiken daarvoor de globale hulpvariabelen 'i', 'eind1' en 'eind2'.

toondatkp - gegevens van knooppunt tonen
Omdat we de regels met gegevens over knooppunten in de totale lijst maar ook bij het opgeven van het kritieke pad nodig hebben loont het de moeite om daarvoor een

```
305
306 procedure toondatkp (aktknoopp:knooppunt); (* regel van een lijst tonen *)
307
308 begin
309   with aktknoopp do
310   begin
311     write(nummer:3);
312     write  (aktivit:27);
313     normvideo;
314     write  (duur:4);
315     writeln(aantalx:4);
```

3.13 Netwerkplanning

```
316     write   (volger1:3);
317     write   (volger2:3);
318     write   (volger3:3);
319     write   (voorgang1:3);
320     write   (voorgang2:3);
321     write   (voorgang3:3);
322     write   (vmb:4);
323     write   (vme:4);
324     write   (lte:4);
325     write   (ltb:4);
326     writeln(ltb-vmb:4);
327   end;
328 end;   (* toondatkp *)
329
```

aparte procedure te schrijven. Alle gegevens van een knooppunt worden daarbij op een enkele regel gezet waarna de cursor nog een regel opschuift. De invoerparameters zijn vanzelf sprekend de gegevens van de actuele knooppunten. Het getal achter de variabele en het ':'-teken geeft het aantal posities van de desbetreffende uitvoer, dat wil zeggen voor alle getallen drie posities en voor de opmerking 27 posities. Omdat het bijna altijd gaat om 2 karakters per gegeven hoeven we niet nog eens apart het spatieteken tussen de verschillende gegevens te zetten.

Door het 'write'-commando te gebruiken besparen we ons nog wat werk omdat dan bij dit commando niet elke keer bijvoorbeeld 'aktknoopp.volger1' of 'aktknoopp.vmb' moet worden geschreven maar alleen de elementen van de knooppunten.

In de lijsten geven we ook de speelruimte per knooppunt op die weliswaar niet zijn opgeslagen maar heel gemakkelijk kunnen worden berekend.

toets - op toestindruk wachten na melding
De procedure 'toets' kan men het best als een algemeen geldige procedure opvatten. Invoerparameter is een regelnummer in het juiste bereik en omdat - door de procedure 'lowvideo' - op negatief beeld is overgeschakeld zal de melding 'Druk een toets in' over de hele breedte van het beeldscherm

```
330
331 procedure toets (regel:integer); (* Op toetsindruk wachten *)
332
333 begin
334   gotoxy(1,regel);
335   write(' ':10);
336   lowvideo;
337   write(' Druk een toets in  ');
338   normvideo;
339   while getkey=chr(0) do;
340 end; (* toets *)
341
```

```
342
343 procedure drukregel(aktknoopp:knooppunt); (* regel printlijst *)
344
345 begin
346   with aktknoopp do
347   begin
348     write(lst,nummer:3);
349     write(lst,aktivit:27);
350     write(lst,duur:3);
351     write(lst,aantalx:3);
352     write(lst,volger1:3);
353     write(lst,volger2:3);
354     write(lst,volger3:3);
355     write(lst,voorgang1:3);
356     write(lst,voorgang2:3);
357     write(lst,voorgang3:3);
358     write(lst,vmb:4);
359     write(lst,vme:4);
360     write(lst,ltb:4);
361     write(lst,lte:4);
362     write(lst,ltb-vmb:4);
363     writeln(lst);
364   end;
365 end; (* drukregel *)
366
```

worden geschreven. Daarna wordt naar gewoon beeld teruggeschakeld en met 'getkey=chr(0)' op het indrukken van een toets gewacht.

drukregel - knooppuntgegevens printeu
De procedure 'drukregel' komt geheel overeen met de procedure toondatkp, behalve dan dat als eerste parameter in het 'write'-commando het uitvoerapparaat 'lst' wordt opgegeven waardoor de uitvoer naar de printer wordt gestuurd.

Hoofdprocedures

Voor de verschillende punten van het menu gaan we nu telkens een of meer procedures opstellen waardoor het hoofdprogramma zo kort en overzichtelijk mogelijk wordt gehouden. De procedures worden in volgorde van voorkomen in het hoofdmenu gegeven.

knpinvoer - invoeren van een knooppunt
Elke menukeuze, zoals ook hier in de procedure 'knpinvoer', wordt van een beeldschermkop voorzien door de procedure 'kop' met als parameter de betreffende tekst aan te roepen. Bij de andere procedures komen we hier niet meer op terug.

Voor de procedure 'knpinvoer' kan worden opgeroepen moet het nummer van het knooppunt bekend zijn. Dit wordt in het hoofdprogramma gerealiseerd door aanroepen van de procedure 'volgnr'. Deze procedure zet ook het beeldschermmasker op het scherm. Elke keer dat een gegeven wordt ingevoerd moet de procedure 'knpinvoer' nog wel voor de positionering van de cursor zorgen alvorens een gegeven kan worden ingelezen. Er vindt verder geen test op de invoer plaats.

3.13 Netwerkplanning

```
367
368 procedure knpinvoer (var aktknoopp:knooppunt); (* Knppuntgegevens lezen *)
369
370 begin
371   kop('     Knooppunt invoeren     ');
372   with aktknoopp do
373   begin
374     gotoxy(15,5);
375     readln(aktivit);
376     gotoxy(22,4);
377     readln(duur);
378     gotoxy(37,4);
379     readln(aantalx);
380     gotoxy(16,8);
381     readln(volger1);
382     gotoxy(36,8);
383     readln(volger2);
384     gotoxy(16,9);
385     readln(volger3);
386     gotoxy(16,12);
387     readln(voorgang1);
388     gotoxy(36,12);
389     readln(voorgang2);
390     gotoxy(16,13);
391     readln(voorgang3);
392     gotoxy(13,17);
393     readln(opmerking);
394   end;
395 end; (* knpinvoer *)
396
```

Verder wijzen we er nog op dat de invoerparameter van het type 'knooppunt' is en, voorzien van de toevoeging 'var', ook als uitvoerparameter kan fungeren omdat de veranderde gegevens door de procedure moeten worden teruggeplaatst.

toon - gegevens van een knooppunt tonen
De programmeeromgeving van de procedure 'toon' is gelijk aan die van 'knpinvoer'. Ook nu moet eerst het nummer van het gewenste knooppunt met de functie 'volgnr' wordt vastgesteld. Het enige verschil met de procedure 'knpinvoer' is dat het commando 'read' door 'write' is vervangen.

verander - wijzigen/aanpassen van gegevens van een knooppunt
Zoals bij de twee laatste procedures werkt ook 'verander' op één enkel knooppunt waarvan het nummer van te voren bekend moet zijn. De parameters worden zowel voor invoer als voor uitvoer gebruikt en omdat de gegevens eventueel ook buiten de procedure verandert kunnen worden is 'aktknoopp' voorzien van de toevoeging 'var'.

Voor het veranderen of aanpassen van gegevens via een beeldschermmasker zijn in principe twee mogelijkheden. Ten eerste: springen van het ene gegeven naar het andere

3.13 Netwerkplanning

```
397
398 procedure toon (aktknoopp:knooppunt); (* Knooppunt tonen *)
399
400 begin
401   kop('      Knooppunt tonen       ');
402   with aktknoopp do
403   begin
404     gotoxy(15,5);
405     write(aktivit);
406     gotoxy(22,4);
407     write(duur:3);
408     gotoxy(37,4);
409     write(aantalx:3);
410     gotoxy(16,8);
411     write(volger1:3);
412     gotoxy(36,8);
413     write(volger2:3);
414     gotoxy(16,9);
415     write(volger3:3);
416     gotoxy(16,12);
417     write(voorgang1:3);
418     gotoxy(36,12);
419     write(voorgang2:3);
420     gotoxy(16,13);
421     write(voorgang3:3);
422     gotoxy(13,17);
423     write(opmerking);
424     gotoxy(7,16);
425     write(vmb:4);
426     gotoxy(17,16);
427     write(vme:4);
428     gotoxy(27,16);
429     write(ltb:4);
430     gotoxy(37,16);
431     write(lte:4);
432     poke(233,orb(peek(233,128));
434    end;
434 end; (* toon *)
435
436
437 procedure verander (var aktknoopp:knooppunt); (* Knoopp.geg. veranderen *)
438
439 var i:integer;
440
441 begin
442   kop(' Knoopp. geg. veranderen ');
443   lowvideo;
444   gotoxy(1,19);
445   write  ('1 Aktiviteit  2 Duur        3 Aantalmaal');
```

3.13 Netwerkplanning

```
446   write  ('4 Volger 1    5 Volger 2    6 Volger 3  ');
447   write  ('7 Voorg. 1    8 Voorg. 2    9 Voorg. 3  ');
448   write  ('10 Opmerking 11 Einde                   ');
449
450   repeat
451     lowvideo;
452     gotoxy(1,24);
453     write('Uw keus :');
454     normvideo;
455
456     gotoxy(12,24);
457     write('  ':5);
458     gotoxy(12,24);
459     readln(i);
460
461     case i of
462       1: begin
463            gotoxy(15,5);
464            readln(aktknoopp.aktivit);
465          end;
466       2: begin
467            gotoxy(22,4);
468            readln(aktknoopp.duur);
469          end;
470       3: begin
471            gotoxy(37,4);
472            readln(aktknoopp.aantalx);
473          end;
474       4: begin
475            gotoxy(16,8);
476            readln(aktknoopp.volger1);
477          end;
478       5: begin
479            gotoxy(36,8);
480            readln(aktknoopp.volger2);
481          end;
482       6: begin
483            gotoxy(16,9);
484            readln(aktknoopp.volger3);
485          end;
486       7: begin
487            gotoxy(16,12);
488            readln(aktknoopp.voorgang1);
489          end;
490       8: begin
491            gotoxy(36,12);
492            readln(aktknoopp.voorgang2);
493          end;
494       9: begin
```

```
495              gotoxy(16,13);
496              readln(aktknoopp.voorgang3);
497           end;
498        10: begin
499              gotoxy(13,17);
500              readln(aktknoopp.opmerking);
501           end;
502        11: begin end;
503         end;
504     until i=11;
505 end; (* verander *)
506
```

en telkens als alleen de RETURN-toets wordt ingedrukt de oude gegevens terug zetten; ten tweede: met een menu vragen naar het gegeven dat verandert moet worden. Wij hebben hier gekozen voor de tweede methode om dat waarschijnlijk telkens maar één enkel gegeven moeten worden aangepast.

Voor een beter overzicht van het schermbeeld wordt het menu zelf in negatief beeld weergegeven, de gegevens in positief beeld. Met de hand kunnen VMB, VME, LTB en LTE niet worden aangepast, die moeten steeds weer opnieuw worden berekend, vandaar dat die parameters ook niet in het keuze-menu voorkomen.

De 'repeat'-lus maakt verandering van een

```
                    Knoopp. geg. veranderen

             ----------------------------------------
             Nummer :  2    Duur   :  4  Hoe vaak:  1
             Activiteit   : Diagrammen opstellen
             ----------------------------------------
             Opvolger 1    :  6 Opvolger 2    :  0
             Opvolger 3    :  0
             ----------------------------------------
             Voorganger 1 :  1 Voorganger 2 :  0
             Voorganger 3 :  0
             ----------------------------------------
             VMB :  5  VME :  9  LTB :  28   LTE : 32
             Opmerking : -

             1 Activiteit  2 Duur        3 Aantalmaal
             4 Volger 1    5 Volger 2    6 Volger 3
             7 Voorg. 1    8 Voorg. 2    9 Voorg. 3
             10 Opmerking 11 Einde

             Uw keus : -
```

Figuur 8/3.13-3: Veranderen van de knooppuntgegevens

3.13 Netwerkplanning

willekeurig gegeven mogelijk zonder dat steeds de menu-optie 3 van het hoofdmenu (knooppunt veranderen) aan geroepen moet worden. 'verander' wordt beëindigd door menu-optie '11' te kiezen. Het uitgeven van een menu-optienummer gaat op dezelfde manier als in de procedure 'menu'. Omdat de hulpvariabele 'i' het nummer van de gekozen optie bevat, kan met het 'case'-commando naar de plaats van het gegeven worden gesprongen dat moet worden veranderd. De cursor wordt naar de juiste positie op het beeldscherm gestuurd waarna de ingetypte gegevens kunnen worden ingelezen.

wissen - gegevens van een knooppunt wissen
Zoals bij alle procedures die we tot nu toe behandeld hebben werkt 'wissen' ook weer op de gegevens van één enkel knooppunt. Door het declareren met 'var' kunnen de veranderde gegevens door de procedure teruggezet. We wijzen er met nadruk op dat het wissen van een knooppunt geen enkele invloed heeft op voorgaande en volgende knooppunten. Vanwege de complexiteit van deze methode zullen we hier later nog op terug komen.

PASCAL-beginners wordt als oefening aanbevolen het terugzetten van de gegevens van een knooppunt op '0', of lege string, in een zelf geprogrammeerde procedure onder te brengen en deze zowel in 'netladen' als 'wissen' te laten oproepen.

lijst - lijst knooppuntgegevens op scherm
Omdat we het samenstellen van een beeldschermregel met de gegevens van een knooppunt al in een aparte procedure hebben ondergebracht hoeven we alleen nog het plaatsen van de regels zelf te behandelen. Natuurlijk wordt eerst weer een opschrift

```
507
508 procedure wissen (var aktknoopp:knooppunt); (*gegevens knooppunt wissen*)
509
510 begin
511   kop('  Knooppuntgeg. wissen    ');
512   with aktknoopp do
513   begin
514     aktivit    :=leeg;
515     duur       :=0;
516     aantalx    :=0;
517     volger1    :=0;
518     volger2    :=0;
519     volger3    :=0;
520     voorgang1  :=0;
521     voorgang2  :=0;
522     voorgang3  :=0;
523     vmb        :=0;
524     vme        :=0;
525     ltb        :=0;
526     lte        :=0;
527     opmerking  :=leeg;
528   end;
529 end; (* wissen *)
530
```

3.13 Netwerkplanning

```
531
532 procedure lijst; (* gegevens knooppunten op scherm zetten *)
533
534 var i:integer;
535
536 begin
537   clrscr;
538   kop(' Gegevens netwerk tonen   ');
539   gotoxy(1,3);
540   writeln('Nr.  Knooppunt                    D.  H.');
541   writeln(' N1 N2 N3 V1 V2 V3 VMB VME LTB LTE SPE');
542   lijn(5);
543   toets(24);
544   gotoxy(1,6);
545
546   for i:=1 to laatste do
547   begin
548     if net[i].ltb-net[i].vmb=0 then lowvideo;
549     toondatkp(net[i]);
550     normvideo;
551     while getkey=chr(0) do;
552   end;
553 end; (* lijst *)
554
```

geplaatst (gegevens netwerk tonen). Omdat er nog al wat gegevens tegelijkertijd op het scherm moeten staan konden we voor de benamingen van de kolommen alleen zeer korte afkortingen gebruiken. Desalniettemin is de tabel zelf toch nog redelijk overzichtelijk.

Na het opschrift komt in regel 23 de melding 'Druk een toets in' op het scherm. Na het indrukken wordt de cursor meteen onder de kop van de lijst gezet. Met een programmalus worden nu alle gegevens van de knooppunten op het scherm gezet. Voor de knooppunten die op het kritieke pad liggen hebben we nog iets bijzonders. Voor die knooppunten geldt dat de waarde van 'speling' nul is, in die gevallen wordt de tekst in negatief beeld op het scherm gezet. Nu volgt het aanroepen van de procedure 'toondatkp', de parameter daarvan moet natuurlijk de waarde van het actuele knooppunt hebben.

Of het nu gaat om een activiteit die op het kritieke pad ligt of niet, bij het terugschakelen naar positief beeld hoeven we dat niet te testen. Na elke regel laten we eerst een toets indrukken, daardoor voorkomen we dat alle knooppunten aan ons voorbij roetsen voordat we ze hebben kunnen bekijken. We moeten dus voor het tonen van alle gegevens per regel telkens een toets indrukken.

Er zijn zeker mooiere lijstprocedures te bedenken, b.v. met heen en weer bladeren. Voorlopig beperken we ons tot deze eenvoudige uitvoering die we later vervangen bij een uitbreiding van het programma-systeem. Op de volgende pagina zien we een schermbeeld van de volledige lijst.

3.13 Netwerkplanning

```
              Gegevens netwerk tonen

   Nr.  Knooppunt                      D.   H.
    N1  N2  N3  V1  V2  V3 VMB VME LTB LTE SPE
   ---------------------------------------------
     1  probleemanalyse                 5    1
     2  3 10   0   0   0   0   5   5    0    0
     2  diagrammen opstellen            4    1
     6  0   0   1   0   0   5   9  32  28   23
     3  beeldschermmaskers              1    8
     7  0   0   1   0   0   5  13  44  36   31
     4  bestanden aanmaken              2    6
     5  0   0  10   0   0   5  17  17   5    0
     5  procedures                      3    9
     7  0   0   4   0   0  17  44  44  17    0
     6  raamprogramma                  12    1
     7  0   0   2   0   0   9  21  44  32   23
     7  systeemtest                     7    1
    11  0   0   3   5   6  44  51  51  44    0
     8  documenteren van systeem        9    1
    11  0   0  10   0   0   5  14  51  42   37

              Druk een toets in
```

Figuur 8/3.13-4: Schermbeeld van de volledige lijst

kritpad - tonen knooppunten op kritiek pad

In grotere netwerken, vooral als het maximaal aantal van 50 door u is veranderd, kunnen niet alle activiteiten tegelijk op een scherm worden getoond. Vandaar dat we een

```
555
556 procedure kritpad; (* Kritiek pad tonen *)
557
558 var i:integer;
559
560 begin
561   clrscr;
562   kop('         Kritiek pad       ')
563   gotoxy(1,3);
564   writeln('Nr. Knooppunt                  D.  H.');
565   writeln(' N1 N2 N3 V1 V2 V3 VMB VME LTB LTE SPE');
566   lijn(5);
567   gotoxy(1,6);
568
```

```
569    for i:=1 to laatste do
570    begin
571      if net[i].ltb-net[i].vmb=0 then toondatkp(net[i]);
572    end;
573    toets (23);
574  end; (* kritpad *)
575
```

afzonderlijke procedure 'kritpad' hebben gemaakt. Deze procedure geeft alleen de activiteiten op het kritieke pad en is dus bijna helemaal gelijk aan de procedure 'lijst'. Ook van deze procedure geven we het schermbeeld, zie de figuur hieronder.

voorwrek - procedure voorwaartsrekenen (recursief)

In de procedure 'verder' hebben we voor de eerste keer te maken met een recursieve rekenmethode. Recursief wil zeggen dat een procedure zichzelf kan oproepen. In het theoretisch gedeelte hebben we gezien dat het vroegst mogelijk eindtijdstip (VME) van een activiteit bij het voorwaarts rekenen ook de vroegst mogelijke begintijdstip (VMB) van de volgende activiteit is. Worden de activiteiten volgens hun nummervolgorde berekend dan is het niet bij elke activiteit zeker dat dan ook het juiste vroegst mogelijke begintijdstip wordt genomen.

Dit is in ons voorbeeld het geval: Bij berekening in nummervolgorde beginnen we met knooppunt 1; daardoor leggen we de vroegst mogelijke begintijdstippen van de activiteiten 2,3 en 10 vast. Na berekenen van activi-

```
                 Kritiek pad

   Nr.  Knooppunt                          D.   H.
    N1  N2  N3  V1  V2  V3  VMB  VME  LTB  LTE  SPE
   --------------------------------------------------
     1  probleemanalyse                      5    1
     2   3  10   0   0   0    0    5    5    0    0
     4  bestanden aanmaken                   2    6
     5   0   0  10   0   0    5   17   17    5    0
     5  procedures                           3    9
     7   0   0   4   0   0   17   44   44   17    0
     7  systeemtest                          7    1
    11   0   0   3   5   6   44   51   51   44    0
    10  scheinknooppunt                      0    0
     4   8   9   1   0   0    5    5    5    5    0
    11  klaar                                0    0
     0   0   0   7   8   9   51   51   51   51    0


             Druk een toets in
```

Figuur 8/3.13-5: Lijst van aktiviteiten op kritiek pad

3.13 Netwerkplanning

```
576
577 procedure voorwrek (aktueel:integer); (* Voorwaats rekenen knpp *)
578
579 var aktvmb,
580     i:      integer;
581
582 begin
583   i:=aktueel;
584   net[i].vme:=net[i].vmb+net[i].duur*net[i].aantalx;
585
586   if net[i].volger1 >0 then
587   begin
588     aktvmb:=net[net[i].volger1].vmb;
589     if aktvmb<net[i].vme then
590     begin
591       net[net[i].volger1].vmb:=net[i].vme;
592       voorwrek (net[i].volger1);
593    end;
594  end;
595
596   if net[i].volger2 >0 then
597   begin
598     aktvmb:=net[net[i].volger2].vmb;
599     if aktvmb<net[i].vme then
600     begin
601       net[net[i].volger2].vmb:=net[i].vme;
602       voorwrek (net[i].volger2);
603     end;
604   end;
605
606   if net[i].volger3 >0 then
607   begin
608     aktvmb:=net[net[i].volger3].vmb;
609     if aktvmb<net[i].vme then
610     begin
611       net[net[i].volger3].vmb:=net[i].vme;
612       voorwrek (net[i].volger3);
613     end;
614   end;
615 end;      (* voorwrek *)
616
```

teiten 2 en 3, die wat dit betreft niet interessant zijn, wordt de VME van activiteit 4 uitgerekend, hoewel een VMB nog niet bekend is.

Als we zo te werk gaan dat, bij elke keer dat een volgend knooppunt voorkomt, de aan dàt knooppunt voorafgaande knooppunten berekend worden zijn we er zeker van dat alle

knooppunten worden behandeld en niet onnodig rekenwerk wordt uitgevoerd. De procedure 'voorwrek' is voor de afhandeling van één enkel knooppunt.

Voor ons voorbeeld geven we de rekenvolgorde: We beginnen natuurlijk met knooppunt 1. Dan wordt de eerste opvolger (knooppunt 2) genomen. Bij knooppunt 2 wordt dan zijn opvolger (knooppunt 6) genomen en dan nog de knooppunten 7 en 11. Elke keer roept de procedure 'voorwrek' zichzelf aan zodat we bij een 'aanroepdiepte' (aantal malen dat de procedure zichzelf heeft aangeroepen zonder terug te springen) 5, bij het laatste knooppunt zijn. Hierbij werd steeds de VME-waarde van de voorganger genomen.

Deze rekenmethode zorgt ervoor dat nu niet verder wordt gegaan met knooppunt 3, maar van knooppunt 11 weer wordt teruggaan naar knooppunt 7, waarbij dan wordt vastgesteld dat van dat knooppunt geen tweede en derde opvolger bestaat. Van 7 terug naar 6 die ook geen tweede en derde opvolger heeft, precies zo bij knooppunt 2. Teruggaand naar knooppunt 1 hebben we dan nog een 'aanroepdiepte' van 1.

Nu wordt de tweede opvolger (knooppunt 3) en dan opnieuw knooppunt 7 berekend! Opnieuw berekenen knooppunt 7 is nodig omdat we niet weten of het vroegst mogelijke eindtijdstip (VME) van knooppunt 6 of die van 3 als VMB-tijdstip van 7 kan worden genomen. Door de eerder uitgevoerde berekening van 7 werd een VMB-tijdstip berekend. We hoeven nu alleen nog na te gaan of de nu berekende VMB groter is dan die vorig berekende VMB. Zo nee, dan hoeven we niet met de opvolgers van 7 verder te gaan omdat die toch geen veranderingen zullen geven. Zo ja, dan moet de nieuwe - grotere - waarde als VMB bij knooppunt 7 worden opgenomen en daarna moet knooppunt 7 opnieuw worden berekend en natuurlijk ook zijn opvolgers. Dit is hier niet het geval omdat knooppunt 3 verder geen opvolgers heeft.

Van knooppunt 1 gaan we naar 10, 4, 5 en 7 en moeten van daaruit nogmaals knooppunt 7 berekenen, omdat de VME van knooppunt 5 groter is dan de tot nu toe berekende VMB-waarde van 7. We gaan de hele weg vanaf 10 weer terug en berekenen 8 en 9, gaan weer terug via 10 naar 1. Hiermee is de procedure afgesloten.

Voor een beter begrip over recursieve eigenschappen van een procedure raden we beginners in PASCAL aan de figuur en de volgende tabel goed te bestuderen. In de tabel wordt in de kolommen 1 t/m 6 de aanroepdiepte, dit is het aantal keren dat de procedure zichzelf aanroept, van dat ogenblik opgegeven. De volgorde van bewerking wordt van boven naar beneden weergegeven; het getal tussen haakjes, dus 1, 2 of 3, staat voor het nummer van de al of niet bestaande opvolger van het zojuist berekende knooppunt. Als aan het bovenstaande is voldaan wordt de procedure 'voorwrek' met het nummer van de opvolger aangeroepen. 'T' geeft het terugspringen naar de procedure aan, vervolgens wordt met het commando na de aanroep verdergegaan, maar dan wel een recursie-niveau hoger.

We kunnen nu de volgende opmerking maken: 'volgerx' en 'voorgangx' (x=1, 2 of 3) hadden we ook als velden van de records van type 'net' kunnen invoeren. Omdat we van elk er maar 3 per knooppunt toelaten hebben we van die methode afgezien, want dan zou de gegevensstructuur duidelijk ingewikkelder worden. De procedures 'voorwrek' en 'terugrek' woorden daardoor wat eenvoudi-

3.13 Netwerkplanning

1	2	3	4	5	6
1 (1)					
	2 (1)				
		6 (1)			
			7 (1)		
				11 (1)	
				11 (2)	
				11 (3)	
				R	
			7 (2)		
			7 (3)		
			R		
		6 (2)			
		6 (3)			
		R			
	2 (2)				
	2 (3)				
	R				
1 (2)					
	3 (1) (knoop 7 wordt niet verwerkt)				
	3 (2)				
	3 (3)				
	R				
1 (3)					
	10 (1)				
		4 (1)			
			5 (1)		
				7 (1)	
					11 (1)
					11 (2)
					11 (3)
					R
				7 (2)	
				7 (3)	
				R	
			5 (2)		
			5 (3)		
			R		
		4 (2)			
		4 (3)			
		R			
	10 (2)				
		8 (1) (knoop 11 wordt niet verwerkt)			
		8 (2)			
		8 (3)			
		R			
	10 (3)				
		9 (1) (knoop 11 wordt niet verwerkt)			
		9 (2)			
		9 (3)			
		R			
	R				
R (naar hoofdprogramma)					

ger te programmeren. De drie bijna identieke blokken worden gekopieerd en vervolgens aangepast. .

Om het vergelijken van de eventueel al bestaande VMB en de actuele VME niet al te gecompliceerd te maken gebruiken we een variabele 'aktvmb' en voor het actuele knooppuntnummer de variabele 'i'. Door deze afkortingen komen de vergelijkingen in het programma op een (tekst)regel, waardoor het programma veel prettiger is te lezen.

De parameter voor de procedure 'voorwrek' is een knooppuntnummer dat wordt uitgegeven door het hoofdprogramma en begint bij nummer 1, het beginknooppunt.

Nu de aanwijzingen voor de procedure zelf: Om te beginnen wordt aan de hulpvariabele 'i' de waarde van het actueel knooppunt gegeven en wordt, volgens de in de theorie besproken manier de - mogelijk al bestaande - waarden van VMB en VME berekend. Er volgen nu drie identieke delen die beginnen met na te gaan of bij elke berekening voor een opvolger een waarde groter dan nul staat. Zo nee, is op deze plaats dus geen opvolger aanwezig, dan kan de volgende opvolger op aanwezigheid worden getest, of de procedure worden beëindigd.

In het andere geval wordt aan de hulpvariabele 'aktvmb' de waarde van het vroegst mogelijk begintijdstip van de betreffende opvolger gegeven. Vervolgens wordt nagegaan of de VMB die aan de opvolger is toegekend kleiner is dan het vroegst mogelijk eindtijdstip van het actueel knooppunt. Zo ja, dan wordt de actuele VME-waarde als VMB-waarde aan de opvolger toegekend en natuurlijk wordt dan voor de opvolger verder gerekend door opnieuw aanroepen van 'voorwrek'.

Op deze plaats wordt ook duidelijk waarom we de hulpprocedure 'tijdopnul' nodig hebben. Van een eerder uitgevoerde berekening kan voor de opvolger reeds een grotere VMB-waarde zijn berekend hoewel bij de actuele berekening nog geen waarde werd toegekend.

Als de tweede en derde opvolger bestaan wordt daarvoor de berekening op precies dezelfde wijze herhaald.

terugrek - terugrekenen (recursief)
De methode van terugwaarts rekenen gaat geheel analoog aan het voorwaarts rekenen, behalve dan dat de invoerparameter voor de eerste aanroep niet '1', maar '11' is (in het algemeen: laatste). Ook de hulpvariabele 'aktvmb' wordt aangepast en heet nu

```
617
618 procedure terugrek (aktueel:integer); (* Terug rekenen knpp *)
619
620 var aktlte,
621     i:      integer;
622
623 begin
624   i:=aktueel;
625   net[i].ltb:=net[i].lte-net[i].duur*net[i].aantalx;
626
627   if net[i].voorgang1 >0 then
628   begin
```

3.13 Netwerkplanning

```
629       aktlte:=net[net[i].voorgang1].lte;
630       if aktlte>net[i].ltb then
631       begin
632         net[net[i].voorgang1].lte:=net[i].ltb;
633         terugrek (net[i].voorgang1);
634       end;
635       if aktlte=0 then
636       begin
637         net[net[i].voorgang1].lte:=net[i].ltb;
638         terugrek(net[i].voorgang1);
639       end;
640     end;
641
642     if net[i].voorgang2 >0 then
643     begin
644       aktlte:=net[net[i].voorgang2].lte;
645       if aktlte>net[i].ltb then
646       begin
647         net[net[i].voorgang2].lte:=net[i].ltb;
648         terugrek (net[i].voorgang2);
649       end;
650       if aktlte=0 then
651       begin
652         net[net[i].voorgang2].lte:=net[i].ltb;
653         terugrek(net[i].voorgang2);
654       end;
655     end;
656
657     if net[i].voorgang3 >0 then
658     begin
659       aktlte:=net[net[i].voorgang3].lte;
660       if aktlte>net[i].ltb then
661       begin
662         net[net[i].voorgang3].lte:=net[i].ltb;
663         terugrek (net[i].voorgang3);
664       end;
665       if aktlte=0 then
666       begin
667         net[net[i].voorgang3].lte:=net[i].ltb;
668         terugrek(net[i].voorgang3);
669       end;
670     end;
671 end; (* terugrek *)
672
```

'aktvme' en gaat het nu niet om 'opvolgers' maar om 'voorgangers'. Nu gaat ook niet getest worden of de LTE-waarde van voorganger kleiner is dan de LTB-waarde van het actueel knooppunt, maar groter. Eigenlijk zou getest moeten worden op: groter dan of

```
673
674 procedure printen; (* Knooppuntlijst naar printerr *)
675
676 var  i:integer;
677
678 begin
679   rewrite(lst,printadr,printsec);
680   kop(' Knooppuntenlijst printen ');
681   writeln(lst,' Gegevens van net    : ',bestand);
682   writeln(lst);
683   write  (lst,'Nr. Knooppunt                         ');
684   writeln(lst,'D. H. N1 N2 N3 V1 V2 V3 VMB VME LTB LTE SPE');
685   write  (lst,'------------------------------');
686   writeln(lst,'------------------------------------------');
687   for i:=1 to laatste do drukregel(net[i]);
688   write(lst,chr(12));
689   close(lst);
690 end; (* printen *)
691
```

gelijk aan. Dit zou, in bepaalde gevallen, aanleiding geven tot een overbodige berekening. Vandaar dat in dit speciale geval, dat de voorganger van het actueel knooppunt nog niet werd berekend, afzonderlijk worden behandeld. In dit geval wordt eenvoudig voor de LTE-waarde, de actuele waarde van de LTB toegekend en in beide gevallen een terugwaarts rekenen uitgevoerd.

printen - activiteitenlijst naar printer sturen
Behalve dat we een lijst met gegevens van knooppunten op het scherm willen hebben willen we ook zo'n lijst laten afdrukken. Net zo als bij de write- en writeln-commando's een bestandsnaam kan worden opgegeven, kan ook een randapparaat worden opgegeven omdat dat voor de computer zelf niets anders dan een bestand is. De standaardinstelling is 'con' dat staat voor: console = beeldscherm. Gaat het om een printer, dan krijgen we de afkorting: lst.

Ook van de procedure 'printen' hebben we er een die daar zeer veel op lijkt, n.l. de procedure 'lijst'. Als we een blik op een lijst werpen kunnen we niet meteen zien om welk netwerkplan het gaat, vandaar dat de naam in de kop wordt vermeld, daarna komt het reeds bekende opschrift voor de tabel. Als laatste opdracht komt de aanroep van een nieuwe pagina (#12). Zo'n lijst met reeds berekende VMB, VME, LTB en LTE-waarden zien we in figuur 6/2.12.1-31. Omdat het beeldscherm van de C64 helaas maar 40 posities kent, is deze lijst veel overzichtelijker dan die we op het scherm kunnen zetten.

Hoofdprogramma

Het hoofdprogramma is, wat de vorm betreft, niets anders dan een procedure zonder parameters. Het declaratiedeel hebben we al in de eerdere listing gezien.

Aan het begin van het hoofdprogramma zien we dat aan een paar variabelen (betreffende kleur en dergelijke) een waarde wordt toegekend. Van de drie hulpvariabelen 'leeg', 'eind1' en 'eind2' krijgt de eerste alleen spaties, de andere twee spaties en het woord-

3.13 Netwerkplanning

Gegevens van net : 0:test

Nr.	Knooppunt	D.	H.	N1	N2	N3	V1	V2	V3	VMB	VME	LTB	LTE	SPE
1	probleemanalyse	5	1	2	3	10	0	0	0	0	5	5	0	0
2	diagrammen	4	1	6	0	0	1	0	0	5	9	32	28	23
3	beeldschermmask.	1	8	7	0	0	1	0	0	5	13	44	36	31
4	bestanden maken	2	6	5	0	0	10	0	0	5	17	17	5	0
5	procedures	3	9	7	0	0	4	0	0	17	44	44	17	0
6	raamprogramma	12	1	7	0	0	2	0	0	9	21	44	32	23
7	systeemtest	7	1	11	0	0	3	5	6	44	51	51	44	0
8	documenteren	9	1	11	0	0	10	0	0	5	14	51	42	37
10	scheinknooppunt	0	0	4	8	9	1	0	0	5	5	5	5	0
11	klaar	0	0	0	0	0	7	8	9	51	51	51	51	0

Voorbeeld van een gegevenstabel

```
693
694 begin (* Hoofdprogramma *)
695   leeg:='                    ';
696   eind1:='klaar               ';
697   eind2:='KLAAR               ';
698   border (randkleu);
699   screen (achtgrkl);
700   pen    (penkleur);
701
702   clrscr;
703   netladen;
704   berekend:=false;
705
706   repeat
707     clrscr;
708     mo:=menu;
709
710     case mo of
711
712     1:begin        (* gegevens knooppunt invoeren *)
713         aktnummer:=volgnr;
714         knpinvoer(net[aktnummer]);
715       end;
716
717     2:begin        (* gegevens knooppunt tonen *)
718         aktnummer:=volgnr;
719         toon(net[aktnummer]);
720         toets(24);
721       end;
722
```

3.13 Netwerkplanning

```
723      3:begin         (* gegevens knooppunt veranderen  *)
724          aktnummer:=volgnr;
725          toon(net[aktnummer]);
726          verander(net[aktnummer]);
727        end;
728
729      4:begin         (* gegevens knooppunt wissen *)
730         aktnummer:=volgnr;
731         wissen(net[aktnummer]);
732       end;
733
734      5:begin         (* tonen - beeldscherm *)
735          laatste:=eindknoop;
736          lijst;
737        end;
738
739      6:begin         (* berekenen *)
740          aktnummer:=1;
741          laatste:=eindknoop;
742          tijdopnul;
743          voorwrek(aktnummer);
744          net[laatste].lte:=net[laatste].vme;
745          terugrek(laatste);
746          lijst;
747          berekend:=true;
748        end;
749
750      7:begin        (* kritiek pad *)
751          laatste:=eindknoop;
752          if berekend=false then
753          begin
754            aktnummer:=1;
755            tijdopnul;
756            voorwrek(aktnummer);
757            net[laatste].lte:=net[laatste].vme;
758            terugrek(laatste);
759            berekend:=true;
760          end;
761          kritpad;
762        end;
763
764      8:begin        (* printen knoopp. netwerk *)
765          laatste:=eindknoop;
766          printen;
767        end;
768
769      9:begin end;
770
771     10:begin        (* nieuw netwerk *)
```

3.13 Netwerkplanning

```
772          netopslag;
773          clrscr;
774          netladen;
775        end;
776
777     11:netopslag;
778    end;
779
780    until mo=11;
781
782 end.
```

je 'klaar', de een met kleine- en de ander met hoofdletters. Met deze variabelen herkent het programma meteen het laatste knooppunt van het netwerk. De volgende drie commando's betreffen de kleur van de rand, scherm en karakters.

Voordat het hoofdmenu kan worden getoond moeten we eerst de gegevens van het netwerk dat we bekijken/bewerken laden. Daarvoor wordt eerst met 'clrscr' het schermbeeld schoongemaakt en met de procedure 'netladen' de gewenste gegevens, inclusief de naam van het netwerk, opgevraagd. Aan de variabele 'gerekend', die van het type boolean is wordt de waarde 'false' toegekend. Deze variabele geeft aan of bij deze sessie de tijdstippen al een keer zijn berekend. We hebben ze later nodig voor het bepalen van het 'kritiek pad'.

Het menu zelf is ondergebracht in een 'repeat'-lus zodat een bepaalde taak kan worden gekozen. Zoals we zien is het eindkriterium de keuze '11'. We weten al dat 'menu' een functieprocedure is die èn kop èn ook het hoofdmenu uitgeeft en als uitvoervariabele het nummer van het gekozen menu-optie geeft. Een pascal-commando dat onze bijzondere aandacht verdient is het vertakkings-commando: "case ... of ...", voor toepassing waarbij gereageerd moet worden op een keuze is dit een uitermate handig commando.

De waarde (menu-optie) waarnaar hier wordt verwezen staat steeds voor een dubbele punt en wordt 'label' genoemd. Labels hoeven niet per se betrekking te hebben op getallen. Deze wijze van programmeren is zeer overzichtelijk, zeker als we een omschrijving van de namen van de procedures als commentaar toevoegen.

Bij invoeren, tonen, veranderen en wissen van een knooppunt moet voor het aanroepen van de betreffende hoofdprocedure het actueel knooppuntnummer bekend zijn. Daarvoor wordt de functieprocedure 'volgnr' aangeroepen die dan een waarde aan de variabele 'aktnummer' toekent. Daardoor kunnen we de inhoud van het veld 'net' als parameter bij de procedures 'knpinvoer', 'toon', 'verander' en 'wissen' gebruiken.

Om ervoor te zorgen dat we de gegevens bij het tonen lang genoeg kunnen bekijken wordt voor elke nieuwe regel, gewacht op een het indrukken van een toets. Dit wordt in regel 24 op het beeldscherm aangegeven. Hetzelfde geldt voor het tonen van gegevens na veranderen. Dit is ook de reden waarom we de procedure 'toets' apart hebben opgenomen en niet in de procedure 'toon'.

De procedures 'lijst', 'voorwrek', 'terugrek', 'kritpad' en 'printen' moeten allemaal weten welk knooppunt van het netwerk het laatste is, vandaar dat aan de variabele 'laatste' elke keer de waarde van de functie 'eindknoop' wordt toegewezen. De berekening wordt telkens uitgevoerd omdat intussen veranderingen hebben kunnen plaatsvinden.

Uit het menu van het hoofdprogramma kunnen we meteen de procedure 'lijst' kiezen die het veld in 'net' als globale parameter gebruikt. Voor het berekenen worden de beide procedures voor voorwaarts- en terugwaarts rekenen na elkaar opgeroepen. In het gedeelte over de theorie werd uiteengezet dat begonnen wordt met de procedure 'voorwrek'. Deze recursieve procedures worden steeds met het eerste resp. laatste knooppunt opgeroepen, vandaar dat de variabele 'aktnummer' als eerste de waarde '1' krijgt. Om fouten in de berekening te voorkomen moeten natuurlijk eerst alle al berekende tijdstippen worden gewist en voor aanroepen van terugwaartsrekenen moeten voor de doelknooppunten de waarden van LTE en VME aan elkaar gelijk worden gesteld.

Omdat men hoogstwaarschijnlijk daarop aansluitend meteen het resultaat wil zien, volgt de procedure 'lijst' en wordt een vlag gezet waaruit blijkt dat reeds een berekening werd uitgevoerd.

De vlag 'berekend' wordt voor het berekenen van het kritiek pad getest omdat het natuurlijk geen zin heeft dit te doen als het netwerk nog niet is doorgerekend. Zijn alle tijdstippen 0, dan is het kritiek pad definitief niet vastgelegd hoewel alle activiteiten bij het uitgeven van de lijst wel worden getoond. Wordt ook geen berekening uitgevoerd dan wordt de analoge daaraan menukeuze 6 uitgevoerd, maar dan zonder afdrukken van de lijst. Tot slot wordt de procedure 'kritpad' opgeroepen en daaropvolgend de gewenste uitvoer gegeven.

Menukeuze nr. 9 is tot nu toe niet geïmplementeerd. Voor het grafisch weergeven is een programma met gecompliceerde algoritmen nodig. We bewaren dit voor een latere uitbreidingen.

Het overschakelen naar een ander netwerk is erg makkelijk. Eerst worden de gegevens van het actuele netwerk opgeslagen en na het schoonmaken van het schermbeeld beginnen we weer van voor af aan.

Met de 'repeat'-lus die we al een keer behandeld hebben is het mogelijk om bij de keuze van menu-optie nr. 11 de gegevens nog op te slaan alvorens het programma wordt verlaten.

Aanwijzingen voor invoeren en compileren

Invoeren

1 Met het 'DISK'-commando de compiler op de diskette aanroepen, om plaats te maken voor de brontekst en de compiler-commando's ter beschikking te hebben.

2 Programma invoeren, het past volledig in het geheugen.

3 Programma met 'PUT NET' op diskette opslaan.

Compileren

1 Met 'COMP NET' kan het programma gecompileerd worden.

2 'EX NET' runt het programma, waarna een test van de diverse onderdelen kan volgen.

3.13 Netwerkplanning

3 Als het programma goed draait kan het met 'COMP NET,C' in een kortere en snellere versie worden omgezet.

4 Vervolgens kan dit programma met 'LOCATE O:NET.BAS=NET' vanuit een BASIC programma worden aangeroepen. Zet het programma vervolgens onder de naam 'NET.BAS' op de diskette.

8/4

C

Inhoud

8/4.1 Introductie
8/4.2 Inleiding
8/4.3 Typen, operatoren en expressies
8/4.4 Control flow
8/4.5 Functies en programmastructuur
8/4.6 Pointers en arrays
8/4.7 Structures
8/4.8 Een groot voorbeeld
8/4.10 Utilities
8/4.11 De standaard bibliotheek
8/4.12 Eigen bibliotheken
8/4.13 Bibliotheekbeheerder

8/4.1

Introductie

C is een 'general purpose' programmeertaal, wat inhoudt dat C voor zeer veel verschillende toepassingen kan worden gebruikt. Deze taal ondersteunt de moderne besturingscontrole (for, while) en data structuren en heeft een rijke verzameling van operaties. C is geen "very high level" taal, zoals Pascal of LISP, noch is C een grote taal qua omvang. Maar omdat C geen beperkingen kent en mede vanwege C's brede toegangsgebied, blijkt C voor vele taken vaak effectiever te zijn dan meer krachtige talen.

C is voor de installatie van het UNIX (*) besturingssysteem ontworpen door Dennie Ritchie. Het UNIX besturingssysteem is tegenwoordig op vele minicomputers geïnstalleerd. Dit is een gevolg van de grote gebruiksvriendelijkheid en de vele mogelijkheden. Het UNIX besturings-systeem, de C-compiler en bijna alle programma's draaiend onder UNIX zijn geschreven in C. C is dus niet aan een bepaalde machine gebonden. Het is trouwens niet moeilijk programma's te schrijven die op andere machines draaien zonder dat deze programma's veranderd moeten worden.

C bevat geen operaties om direct met samengestelde objecten zoals strings, sets, lijsten of arrays als een geheel te werken. C is tenslotte niet als een "very high level" taal geclassificeerd. Het is dus bijvoorbeeld niet mogelijk om met één operatie te checken of het getal nul in een set zit. C kent zelfs geen invoer en uitvoer statements. Deze hogere niveau mechanismen moeten worden toegevoegd en expliciet als functies aangeroepen worden. Eveneens biedt C alleen de simpele besturingscontrole-constructies: testen, loops, subprogramma's, maar geen operaties voor multiprogrammering, parallelle operaties, synchronisatie of co-routines.

Alhoewel het vervelend lijkt dat bepaalde constructies niet in de taal zitten (men moet een functie aanroepen om twee strings te vergelijken) heeft het toch verscheidene voordelen dat de taal op zo'n laag niveau is gebleven. Want C is relatief klein, het kan in beperkte ruimte beschreven worden en de taal is gemakkelijk te leren. Een C-compiler kan simpel en compact zijn. De compilers zijn zo simpel, dat ze in slechts een paar maanden geschreven kunnen worden en dan blijkt dat 80 procent van de code overeenkomt met de code van andere C-compilers. Dit betekent dat de taal erg mobiel is of anders gezegd, dat de taal gemakkelijk van machine naar machine overdraagbaar is. Natuurlijk hoort er bij elke implementatie een

4.1 Introductie

standaard bibliotheek van functies voor de I/O, de string behandeling en geheugenallocatie-operaties, maar ze moeten allemaal expliciet aangeroepen worden.

Voor hen die bekend zijn met andere programmeertalen is het misschien leuk om enkele historische, technische en filosofische aspecten van C te weten, om een en ander met elkaar te kunnen vergelijken.

Veel van de ideeën voor C stammen af van een veel oudere taal: BCPL. Deze is door Martin Richards ontwikkeld. De invloed van BCPL is indirect gegaan via de taal B, die door Ken Thompson in 1970 is gebouwd voor het eerste UNIX besturingssysteem. Alhoewel BCPL en C veel elementen gemeenschappelijk hebben is C zeker geen dialect van BCPL. BCPL en B zijn typeloze talen: het enige datatype is het machinewoord. In C zijn de fundamentele data objecten de tekens, gehele getallen van verschillende grootte en floating point (drijvende komma's) getallen. Tevens bestaan er nog afgeleide data typen, zoals pointers, arrays, structures, unions en functies, waarmee ook weer nieuwe data typen te construeren zijn.

C heeft de fundamentele besturings controle constructies, die noodzakelijk zijn voor goed gestructureerde programma's zoals: het groeperen van statements; test statement (if); herhalings statement met een testconditie (while, for, do) en selectie van een van de mogelijke gevallen (switch).

C ondersteunt pointers en de mogelijkheid om met adressen te manipuleren. De argumenten van functies worden aan de functie doorgegeven door de waarden te kopiëren. Het is voor de aangeroepen functie om de argumenten te veranderen. Als die nog nodig mochten zijn dan kan het 'call by reference' principe worden gebruikt, een pointer naar de te veranderen parameter wordt dan doorgegeven en de functie kan het object veranderen waarnaar de pointer verwijst. De namen van arrays worden zo doorgegeven: met een pointer naar het begin van de array, dit gaat effectiever dan op de andere manier, want bij 'call by reference' behoeft dan niet het gehele object, hier de array, gekopieerd te worden.

Elke functie kan recursief aangeroepen worden (de functie roept zichzelf aan). De definities van de functies mogen niet in elkaars definitie geschreven worden, dit in tegenstelling tot Pascal. De functies in C-programma's kunnen apart gecompileerd worden, zodat eigen functie-bibliotheken aangelegd kunnen worden.

C is geen strikt getypeerde taal, zoals Pascal. Dit houdt in dat C niet zoveel moeite heeft met het veranderen van het type wat bij de uitkomst van een expressie vrijkomt, alhoewel C niet automatisch datatypes verandert, zoals PL/1 dat doet.

Uiteraard heeft C evenals elke andere taal zo zijn tekortkomingen. Sommige delen van de definitie zouden beter kunnen. Er zijn verscheidene talen die deze verbeteringen hebben door gevoerd. Maar niettegenstaande heeft C bewezen dat het een bijzonder effectieve taal

4.1 Introductie

is, waar men zich zeer goed in kan uitdrukken voor een groot aantal toepassingen. Het gevolg is dat er van C weinig dialecten bestaan. Wel is het zo dat men heeft geprobeerd om bepaalde mogelijkheden en gemakken van hoge programmeertalen in C onder te brengen. Enkele zijn hier van het noemen waard: C++ en Classes. Deze talen voegen bepaalde data-abstracties toe. De laatste taal wordt door een compiler vertaald naar C. Deze normale C-code kan dan op zijn beurt weer door de C-compiler gecompileerd worden. Ook is Enhanced C zeker het noemen waard. Deze taal wordt ook met een compiler vertaald naar C-code en bergt vele faciliteiten, die ook in de hogere programmeertalen mogelijk zijn. Het is zelfs zo dat deze taal door de gebruiker uit te breiden is met nieuwe statements, waardoor deze taal op een steeds hoger niveau komt te staan.
Het is zeker de moeite waard om in de toekomst te kijken wat er met Enhanced C gebeurt. Enhanced C is een taal die een grote kans maakt dat hij het gaat 'maken'. Het is een academische taal, die door professor J. Katzenelson van de Universiteit van Haifa wordt ontwikkeld. Ondertussen wordt deze taal al op de TH Delft en op het MIT, een belangrijke Amerikaanse universiteit, gebruikt. Grote programma's zijn er al mee geschreven, zoals een database programma, een IC lay-out ontwerp programma en een programma waarin parallelle processoren (iets waar de grote computers in de toekomst mee gaan werken) in gesimuleerd worden.
De verwachting is, dat Enhanced C zich vanaf de universiteiten naar de industrie zal verplaatsen en misschien zal deze zeer mooie taal nog eens op de Commodore 64 geïmplementeerd worden.

Al deze talen zijn echter gebaseerd op C en gebruiken de taal ook, simpelweg omdat C snel, duidelijk, algemeen, overdraagbaar van machine naar machine en wijd verbreid is. De taal is dus bruikbaar voor zeer veel toepassingen. Goede implementeerbaarheid, efficiëntie, uitdrukkingskracht en enige affiniteit met machinetaal staan meer centraal in C dan 'veiligheid', waar de bescherming van de gebruiker tegen zichzelf mee wordt bedoeld. Als men op een gegeven moment het niveau heeft bereikt van een goede C programmeur dan is C een plezierige taal om mee te werken. C is een nogal cryptische taal, al is het veel duidelijker dan de puur cryptische APL, maar wie C eenmaal beheerst, vindt een C-programma uitstekend leesbaar. Mits uiteraard bepaalde schrijfregels in acht genomen worden. Al met al is het uiterst leerzaam en blikverruimend om C te kunnen verstaan en te kunnen spreken.

(*) UNIX is a Trademark of Bell Laboratories.

8/4.2

Inleiding

We beginnen met een korte inleiding in C. een soortgelijke inleiding kunt u ook vinden in 'The C programming language' van Kernigham en Ritchie.
Eerst zullen de elementen, die ook in echte programma's aan de orde komen, getoond worden, zodat u zo snel mogelijk zelf programma's kan gaan schrijven. Het accent zal vooral liggen op de basiselementen van C: variabelen en constanten, wiskundige berekeningen, de programmabesturing, flow statements, functies en de beginselen van invoer en uitvoer. In het begin worden belangrijke elementen die nodig zijn voor het schrijven van de grotere programma's overgeslagen. Deze elementen zoals pointers, structures, vele C operaties en verschillende besturingsstatements komen later aan de orde. Zoals u reeds opgemerkt zult hebben worden er nogal wat Engelse termen gebruikt, het is handig om deze termen te kennen. Dit vereenvoudigt de communicatie met anderen. Deze Engelse termen zullen uitgelegd worden tegen de tijd dat echt nodig is.

Uiteraard heeft het zo zijn nadelen als niet meteen alles wordt verteld, het kan zelfs misleidend zijn. Ook de voorbeelden zullen niet uitblinken door elegantie. Deze terkortkomingen zijn echter niet van invloed op de 'educatieve inhoud' van dit hoofdstuk.

In elk geval zullen ervaren programmeurs in staat zijn veel informatie uit deze paragraaf te halen. De beginners moeten in staat zijn kleine programma's te schrijven. Beide groepen kunnen dit hoofdstuk als kapstok gebruiken waar de meer gedetaileerde beschrijvingen van latere hoofdstukken aan opgehangen kunnen worden.

8/4.2.1
Hoe te beginnen

Zoals de meeste programmeurs dat doen in elk andere taal zo zullen wij ook beginnen. Het eerste programma moet afdrukken

hallo

Dit is het begin en dit is op zich al lastig genoeg; u zult in staat moeten zijn een programmatekst te creëren, het succesvol te compileren (vertalen naar machinetaal), het te linken (samenvoegen met bibliotheekfuncties), het te laden en te RUNnen. Zodra u weet hoe alles werkt blijft het erg eenvoudig te zijn. In C is het programma dat 'hallo' afdrukt:

```
main()
{
    printf("hallo\n");
}
```

Om dit programma te laten werken moeten de volgende acties ondernomen worden:

4.2 Inleiding

1: Editen (het aanmaken van de tekst)
2: Compileren (het vertalen naar machine code)
3: Linken (het verbinden met al bestaande functies)
4: Loaden (het laden van disk)
5: Runnen

Dit is een heel ander systeem dan BASIC, in BASIC wordt een programma ingetypt en dit kan direct gerund worden. In C en in bijvoorbeeld Pascal moeten de programmateksten eerst gecompileerd worden, bij C moet de uitvoer van de compiler ook nog gelinkt worden en dan kan het programma uiteindelijk geRUNd worden. Het verschil tussen BASIC en C zit hem in de manier waarop de programma's van deze talen geëxecuteerd worden. BASIC programma's worden geïnterpreteerd en C programma's worden eerst gecompileerd en gelinkt, later wordt een stuk machine-code, wat de representatie van de C-code is, geRUNd. Een taal die geïnterpreteerd wordt heeft een interpreter nodig om de interpretatie te verrichten. Interpreteren houdt in dat elk statement van het programma wordt bekeken tijdens het RUNnen en daardoor het bijbehoordende stukje machine-code, wat zich in de interpreter bevindt, uitgevoerd wordt. Bijvoorbeeld het PRINT-statement wordt door de interpreter gelezen en de intepreter roept de functie aan die het PRINT-statement uitvoert. Een taal die gecompileerd wordt heeft dus een compiler nodig. Deze compiler bekijkt elk statement uit een programma afzonderlijk en maakt dan in een corresponderende file een stuk machine-code aan dat overeenkomt met de betekenis van het statement uit het programma. Een gecompileerd programma bestaat dus uit alleen maar machine-instructies. Vandaar dat een gecompileerd programma over het algemeen sneller werkt dan een geïnterpreteerd progamma.

C moet niet alleen gecompileerd worden, maar moet ook nog eens gelinkt worden. Dit linken houdt in dat al reeds bestaande functies, die al gecompileerd zijn, aan het programma vastgemaakt worden zodat het programma over die functies kan beschikken. Als u dus zelf een karakter-print functie maakt, behoeft deze maar één maal gecompileerd te worden en kan deze functie aan alle programma's, die hier gebruik van willen maken, gelinkt worden. Nu kunnen al deze programma's RUNnen zonder dat u de karakter-input functie steeds opnieuw schrijft.
Het is echter niet noodzakelijk om een karakter-print functie te maken, want in de standaard bibliotheek bevinden zich zeer veel functies die al voorgedefinieerd zijn en een karakter-print functie is er een van. Deze functies kunt u met uw eigen geschreven programma linken, dus zelf hoeft u ze dan niet meer te maken. Bij C behoren twee bibliotheken, die soms tot één samen gevoegd worden. Ze heten: STDLIB.L en SYSLIB.L. In de eerste zitten alle standaard functies zoals printf, in de tweede zitten alle systeem-routine calls, die ervoor zorgen dat bijvoorbeeld de disk-I/O goed verloopt.

Bij de C-compilers voor de Commodore 64 (compiler = programma dat tekst van de betreffende taal omzet in machinecode) zitten duidelijke tekst-files die de werking van dat systeem beschrijven. Bij de

4.2 Inleiding

hier gebruikte compiler, die van Pro Line, zit een kleine 'shell' bijgeleverd. Een shell is een kleine commando-interpreter. Als deze shell wordt opgestart verschijnt het dollar-teken '$' als prompt (teken waar de cursor achter staat) in beeld. Het opstarten van de shell gaat met behulp van het "LOAD"SHELL"
8,1" commando. Na het intoetsen van het ED commando wordt de editor opgestart. Met deze editor is de bovenstaande C-tekst aan te maken. Van de editor moet u weten hoe bepaalde tekens zijn gedefineerd:

braces als <shift> + en − ({ })
tilde als <logo> p (~)
underscore als <logo> (_)
vertical bar als <logo> (|)

Als dit is gebeurd en u verlaat de editor verschijnt de prompt weer in beeld. Het programma is nu klaar om door de compiler gecompileerd te worden. Hiertoe moet u "CC FILENAAM.C" intoetsen (de filenaam moet op 'c' eindigen anders wordt de file door de compiler geweigerd, geen nood als dit niet het geval is, met het 'mv' commando is dit probleem te verhelpen). De compiler wordt geladen en de compiler vertelt wat u moet doen. Als dit dan achter de rug is moet het gecompileerde programma nog gelinkt worden. Met het 'link' commando wordt dit voor u gedaan. Bij de linker zal moeten worden opgegeven welke programma's u allemaal wilt linken. Ten eerste natuurlijk het zojuist geschreven programma, vervolgens moeten de standaard bibliotheek en de systeem bibliotheek mee gelinkt worden. Dit gaat als volgt: na de '>' prompt typt u de naam van uw eigen programma in, alleen de '.C' aan het eind vervangt u door '.o'. U drukt op RETURN, dan typt u 'STDLIB.L', gevolgd door return en als laatste voert u 'SYSLIB.L' in, eveneens gevolgd door RETURN. Deze laatste twee files staan op kant twee van de diskette. Als u nu na de volgende '>' prompt RETURN geeft worden de files gelinkt. Nu zal de linker om de filenaam vragen van de te runnen file, als deze op '.sh' eindigt zult u van uit deze shell het programma kunnen opstarten.
Kijken we nu op disk, dan zien we drie files die er bij zijn gekomen, te weten:
filenaam.c (deze heeft u zelf geschreven)
filenaam.o (deze is door de compiler gecrëerd)
filenaam.sh (deze is door de linker gecreëerd)

Na het uittoetsen van 'filenaam' wordt het programma automatisch geladen en gerund.

Nu het gelukt is om het programma te laten draaien is een nadere beschouwing niet ongewenst. Een C programma bestaat altijd uit één of meer functies (bijvoorbeeld de 'GOSUB' in BASIC), die de eigenlijke acties van het programma specificeren. Deze C functies zijn hetzelfde als de procedures van Pascal. In dit voorbeeld is main een functie. Normaal mag elke willekeurige naam voor een functie worden gekozen, maar main is een speciale naam; elk programma begint namelijk met de executie van main. Dus elk programma moet ergens een main hebben.

Informatie voor de functies wordt door parameters of argumenten doorgegeven. De ronde haakjes die na de functie naam volgen bevatten de argumenten;

hier is main een functie zonder arguments, wat door '()' wordt aangegeven. De haakjes { en } groeperen de statements (opdrachten, taal-primitieven of functies) die bepalen wat de functie doet, dit is hetzelfde als BEGIN en END in Pascal. Een functie wordt gebruikt, geëxecuteerd, door hem aan te roepen met de juiste argumenten tussen de ronde haakjes. De regel

printf ("hallo\n");

is tevens een functie-call (functie aanroep) die een function aanroept die printf heet, met als argument "hallo\n". printf is een bibliotheek functie die de output (uitvoer) op de terminal afdrukt (mits anders gespecificeerd).

De groep tekens die door de dubbele quotes "..." wordt ingesloten, wordt een character string of string constant genoemd. De tekens "\n" in de string is de C notatie voor de newline, dit teken zorgt ervoor dat het volgende wat afgedrukt wordt links op een nieuwe regel begint. Als u \n weg laat zult u zien dat het beeld niet scrollt (alle regels gaan een positie omhoog). De printf functie zorgt zelf niet voor een newline of iets dergelijks, zodat ons programma er ook als volgt had kunnen uitzien.

```
{
    printf("hal");
    printf("lo");
    printf("\n");
}
```

N.B. "\n" is dus maar één teken. Met de backslash voor de n zorgt C er voor dat 'onzichtbare' tekens toch in een programma listing te zien zijn. C kent nog meer van deze tekens: "\t" voor een tab (tabulonstap, deze wordt op de C-64 niet gebruikt), "\b" voor backspace (de cursor wordt één positie terug gebracht), "\"" voor het afdrukken van de dubbele quote en "\\" voor de backslash zelf.

8/4.2.2
Variabelen en berekening

Het volgende programma drukt een tabel af van Celcius temperaturen met hun equivalent in Fahrenheit. Hierbij wordt gebruik gemaakt van de formule: F=9/5*C+32.

```
-20    -4.0
-10    14.0
  0    32.0
 90   194.0
100   212.0
```

De eerste regel

/* druk de Celcius-fahrenheit tabel af
voor celcius = -20,-10. . . .,100 */

is commentaar, hier wordt kort verteld wat het programma doet. Alle tekst tussen /* en */ wordt door de compiler overgeslagen; deze commentaren mogen vrijelijk in het programma worden gebruikt om dit te verhelderen. Commentaren mogen op de zelfde plaats staan waar ook blanks(spaties) of newlines mogen staan.

In C moeten alle variabelen, evenals in Pascal, gedeclareerd worden voordat ze worden gebruikt. Dit gebeurt meestal aan het begin van een functie, in ieder geval voordat de statements geëxecuteerd worden. De compiler geeft een foutmelding als u een variabele bent vergeten te declaren. Een declaratie bestaat uit een type en een lijst van variabelen

4.2 Inleiding

```
/* druk de Celcius-fahrenheit tabel af
   voor celcius = -20,-10,....,100 */
main()
{
   int onder,boven,stap;
   float celcius,fahr;

   onder = -20; /* ondergrens van de temperatuur tabel */
   boven = 100; /* boven grens */
   stap  =  10; /* stap grootte */
   celcius = onder;
   while (celcius <= boven ){
      fahr = (9.0/5.0)*celcius+32.0;
      printf("%4.0f %6.1f\n",celcius,fahr);
      celcius = celcius+stap;
   }
}
```

die dat type hebben, zoals

int onder, boven, stap;
float celcius, fahr;

Het type int wil zeggen dat alle variabelen integers (gehele getallen . . ., -2, -1,0,1,2 . . .) zijn, float betekent floating point, dit houdt in dat de getallen een komma bezitten. De precisie van int en float is machine- en compiler-afhankelijk.
De data typen die C ondersteunt zijn:

TYPE	OMSCHRIJVING
char	teken, enkel byte
short	korte integer
int	integer
long	lange integer
float	floating point
double	dubbele precisie float

Tevens zijn er arrays, structures en unions die uit deze datatypen kunnen worden opgebouwd, eveneens bestaat de pointer. We zullen al deze begrippen later tegenkomen.

De werkelijke berekening van de temperatuurs-conversie begint met de assignment (waarde toekenning).

```
onder   =  20; /* ondergrens van
                  temperatuur tabel */
boven   = 100; /* boven grens */
stap    =  10; /* stap grootte */
celcius = onder;
```

zodanig dat de variabelen op hun juiste begin waarden worden ingesteld. De individuele statements worden door semicolons (;) afgesloten.

Elke regel in de tabel wordt op dezelfde manier berekend, dus wordt er een loop gebruikt die per uitvoerregel steeds herhaald wordt;

```
while (celcius = boven) {
   fahr = (9.0/5.0 * celcius + 32.0;
   printf (" %4.0f %6.1f\n", celcius,
   fahr);
   celcius = celcius + stap;
}
```

Eerst wordt de conditie tussen de ronde haakjes getest. Als deze true (waar) is (celcius is kleiner of gelijk aan boven) wordt de body van de loop (al de statements tussen de braces ({ en })) geëxecuteerd. Daarna wordt de conditie opnieuw getest en als deze true is wordt de body opnieuw geëxecuteerd. Als de test false (niet waar) wordt (celcius

4.2 Inleiding

wordt groter dan boven), stopt de loop en wordt het volgende statement na de loop geëxecuteerd. Hier zijn geen statements meer, dus in dit geval stopt hier het programma.

De body van de while kan één of meer statements bevatten, zoals het voorgaande voorbeeld, of een enkele statement zonder de braces zoals

```
while (a > b)
   x = x*2+y;
```

In beide gevallen wordt bij de statements van de body een tab-stop ingesprongen of een drietal spaties, zodat u direct kunt zien welk(e) statement(s) bij de body van de loop behoren. Hoewel het voor C niet uitmaakt hoe de statements geplaatst worden, is een geschikte vorm van inspringen van essentieel belang voor het begrijpen van een programma. Een aanbeveling is om slechts één statement per regel te schrijven en spaties tussen de operatoren (zoals <, =, etc) te laten.
De positie van de braces is minder belangrijk, hier is voor één van de populaire stijlen gekozen. Kies er zelf één, en wees er consequent mee.

Het meeste werk wordt in de body van de loop gedaan. De Fahrenheit temperatuur wordt berekend en aan fahr toegekend door het statement;

```
fahr = (9.0/5.0)*celcius+32.0;
```

9.0/5.0 wordt gebruikt in plaats van 9/5 omdat integer deling in C wordt afgerond en dus het gedeelte achter de komma niet gebruikt wordt. Een decimale punt in een constante duidt op floating point getal, dus 9.0/5.0 is 1.800 en dit is hetgeen wat hier gewenst is.

N.B. dit is trouwens niet efficiënt geprogrammeerd, want de constante 9.0/5.0 wordt tijdens de loop steeds uitgerekend. Beter zou het zijn geweest om deze berekening één keer, voor de loop, te laten plaats vinden; deze constante wordt dan binnen de loop gebruikt. Wel is het zo dat als u met een 'slimme' compiler werkt deze dit soort zaken wel op de zo juist beschreven manier voor u zal doen. Tevens is 32.0 gebruikt inplaats van 32, zelfs ondanks het feit dat er automatisch een int-float conversie (van 32 naar 32.0) zal plaats vinden. Deze conversie wordt automatisch door de compiler gerealiseerd omdat fahr een float is.

In dit programma wordt printf gebruikt. Printf is een functie uit de standaard bibliotheek. Een uitvoerige beschrijving van printf volgt later. Het eerste argument is een besturingsstring. Elk %-teken duidt een argument aan, dit argument wordt dan op de aangegeven manier afgedrukt, zoals in

```
printf("%4.0f %6. 1f\n", celcius, fahr);
```

waarin de conversie van %6.1f beschrijft dat er een float met 6 cijfers voor en 1 cijfer achter de komma afgedrukt moet worden. %4.0f beschrijft dat er vier cijfers voor en geen cijfers achter de komma afgedrukt moeten worden. Gedeelten van deze beschrijving mogen weggelaten worden: %6f betekent dat het getal minstens zes tekens groot is; %.2f betekent twee plaatsen achter de decimale punt, maar er worden geen eisen aan de grootte

gesteld. %f betekent dat er een floating point getal afgedrukt moet worden. Printf herkent tevens %d voor een decimale integer, %c, voor een karakter, %s voor een string, en %% voor % zelf.

Bij elke %-constructie in het eerste argument van printf hoort een eigen waarde. Deze moeten correct volgen anders komen er betekenisloze antwoorden uit. Het weglaten van bijv. fahr in het voorbeeld zou fout zijn.

Tussen haakjes, printf is geen onderdeel van C; in C is geen I/O gedefinieerd. Printf is slechts een handige functie die in de standaard bibliotheek is opgenomen. Deze bibliotheekfuncties zijn voor alle C-implementaties op een zelfde manier aan te roepen. Voorlopig laten we de I/O even met rust, later zal een gedetailleerdere beschrijving volgen.

8/4.2.3
Symbolische constanten

Nog een laatste blik op het temperatuur conversie programma, voordat het met rust gelaten wordt. Eigenlijk is het een slechte gewoonte om 'magische getallen' zoals 100 en -20 in een programma te stoppen, ze zeggen niets voor iemand die het programma later leest, hij kan het programma niet eenvoudig systematisch aanpassen. Gelukkig kunnen deze magische getallen in C vermeden worden. Met de '#define' constructie aan het begin van het programma kan men een symbolische naam of een symbolische constante als zijnde een bepaalde karakter string definiëren. De compiler zal dan alle ongequote plaatsen, waar die naam staat, vervangen door de corresponderende string. De vervangende naam mag uit elke gewenste tekst bestaan, hier bestaan geen beperkingen op.

```
#define  ONDER -20 /* onder grens v/d tabel */
#define  BOVEN 100 /* boven grens */
#define  STAP   10 /* stap grootte */
main()             /* Celcius-Fahrenheit tabel*/
{
   int celcius;

   for (celcius = ONDER ; celcius <= BOVEN ; celcius = celcius+STAP)
      printf("%4.0d %6.1f\n",celcius,(9.0/5.0)*celcius+32);
}
```

ONDER, BOVEN en STAP zijn constanten, ze behoeven dan niet gedeclareerd te worden. Symbolische namen worden meestal in hoofdletters geschreven om ze van kleine variabelen te onderscheiden.
N.B. er is aan het eind van de definitie geen semicolon, omdat de hele regel achter de gedefinieerde naam wordt gesubstitueerd.

8/4.2.4
Nuttige programma's

Input en Output van tekens
De standaard bibliotheek heeft functies voor het lezen en schrijven van tekens. Getchar leest het volgende teken van de standaard input (meestal het toetsenbord) en returnt (waarde afgeven bij beëindiging) dit als waarde.

4.2 Inleiding

Zodat met

c = getchar()

c de waarde van het volgende teken uit de standaard input krijgt. Normaal komen de tekens van het toetsenbord, maar hier wordt later nog op terug gekomen.

De functie putchar(c) is het complement van getchar.

putchar(c)

print de inhoud van variabele c op een outputmedium, dit is meestal de beeldbuis. Putchar en printf mogen door elkaar heen worden aangeroepen: de output zal verschijnen in dezelfde volgorde zoals de functies zijn aangeroepen.

Evenals met printf is er niets speciaals aan getchar en putchar. Ze maken geen deel uit van C, maar ze zijn wel overal in de standaard bibliotheken te vinden.

Het kopiëren van tekst
Als getchar en putchar zijn gegeven, dan is het bijzonder eenvoudig om leuke dingen te doen met I/O zonder iets van de I/O af te weten. Het eenvoudigste voorbeeld is een tekst-kopieer programma. In C wordt dit dan:

```
#include <stdio.h>

main()
{
    int c;
    c = getchar();
    while (c != EOF) {
        putchar(c); c = getchar();
    }
}
```

De != operator betekent 'is niet gelijk aan'.

Het grootste probleem is om het einde van de input te detecteren. Het einde van de invoer wordt door de waarde -1 of 0 aangegeven. In de praktijk wordt een symbolische constante gedefinieerd 'EOF' (End Of File), dat het einde-file teken voorstelt (een file is een gegevensverzameling, deze kan b.v. op disk staan maar kan b.v. ook van het toetsenbord komen). Elk programma moet dus worden voorafgegaan door:

```
#define EOF -1
```

of door

```
#define EOF 0
```

om correct te kunnen werken. Maar ook kunt u boven in uw programma de macro (zo heet dat nu eenmaal) "#include <stdio.h>" opnemen. Met de #include macro wordt de file, die als argument wordt gespecificeerd, gelezen als stond deze tekst in het programma. In stdio.h wordt deze definitie al voor u gedaan zodat u zich daar niet over behoeft te bekommeren.

Nu komt er echter een praktisch probleem om de hoek kijken, vanaf het toetsenbord is het EOF teken met behulp van de getchar()-functie niet te lezen (de integer waarde 4). Normaal wordt voor het EOF teken van het toetsenbord de CTRL-D toets gebruikt, de getchar() functie heeft het echter niet in de gaten als deze toets wordt ingedrukt.

Dit probleem is op twee verschillende manieren op te lossen. Ten eerste kan een speciaal teken, dat door getchar() gelezen kan worden, als EOF teken worden gebruikt. Ten tweede kan de

4.2 Inleiding

getchar() functie herschreven worden. De eerste methode heeft als nadeel dat er een teken verloren gaat voor normaal gebruik en tevens ook de compatibiliteit met de echte C-programmatuur verloren gaat. Het herschrijven van de functie getchar() is niet echt moeilijk. Wel moeten wat systeem-variabelen bekend zijn. Men moet weten of er een toets is ingedrukt, deze variabele staat op adres 198 (hier staat het aantal ingedrukte tekens). Ook moet men de ASCII-waarde van dit ingedrukte teken weten, deze waarde staat aan het begin van de invoer-buffer. Deze buffer loopt van adres 631 tot en met 640. Het programma komt er dan als volgt uit te zien:

```
#include <stdio.h>

#define CURSOR 204
#define CURSORAAN 0
#define CURSORUIT 1

#define JA  1
#define NEE 0

#define BACKSPACE 157

#define BUFFERSIZE 80

int buffervol   = NEE,
    bufferbegin = 0,
    buffereind  = 0;

char buffer[BUFFERSIZE];

#define BUFFERBEGIN   631
#define BUFFEREIND    640
#define READSTATUS    198

readchar()


{
   char c;
   char *charpoint;
   char *status;
   char *i;

   charpoint    = BUFFERBEGIN;
   status       = READSTATUS;

   while (*status == 0) {
      ;
   };
   c = *charpoint;
   *status -= 1;
   for (i = BUFFERBEGIN
        ;i < BUFFEREIND
        ;*i = *++i) { ; };
   return(c);
}

readbuffer()


{
   char c;

   c = buffer[bufferbegin++];
   if (bufferbegin == buffereind) {
      bufferbegin = buffereind = 0;
      buffervol = NEE;
   }



   return(c);
}

fillbuffer()


{
   char c;

   do {
      c = readchar();
      printf(" %c%c",BACKSPACE,c);
      buffer[buffereind++] = c;
   } while (c !='\n' && c != '\04'
        && buffereind != BUFFERSIZE);
   buffervol = JA;
}
```

4.2 Inleiding

```
getchar()        /* RETURNt karakters na
                    een return of CTRL-D
                    of als de buffer vol
                    is                    */
{
   char c;
   char *cursor;

   cursor  = CURSOR;
   *cursor = CURSORAAN;

   if (buffervol == NEE)
      fillbuffer();
   c = readbuffer();
   *cursor = CURSORUIT;
   if (c == '\04')
      return(EOF);
   else
      return(c);
}
```

Ongetwijfeld zult u al enkele dingen herkennen in dit stukje programmatuur. Maar om nu nog niet te diep op de materie in te gaan zullen de functies later uitgelegd worden. Met deze nieuwe routine kunt u dus op het einde van de invoer (EOF of een CTRL-D) van het toetsenbord testen.
N.B. De werkwijze met de linker is even anders. Om niet de 'verkeerde' of oude versie van getchar() te linken (aan het programma te verbinden) zal tijdens het linken de sessie er als volgt uitzien.

```
$ link
> copyl.o
< getchar.o
> stdlib.l
> syslib.l
>
enter object filename: copyl.sh
```

Het kopieerprogramma kan eigenlijk nog compacter geschreven worden. In C kan elk assignment statement (een statement waar een '=' in voorkomt, eigenlijk een waarde-toekennings statement) in een expressie (een uitdrukking o.i.d.) worden opgenomen (zie b.v. hieronder); de waarde van de expressie is dan simpelweg de waarde van de rechter kant van de assignment. Dus in het volgende programma krijgt c de waarde van de functie getchar() en wordt de waarde van het gedeelte rechts van het = teken gebruikt in de

4.2 Inleiding

expressie (denk om de haakjes!). Het is even lastig maar als u het voorbeeld bekijkt wordt alles een stuk duidelijker. Het programma wordt dan als volgt geschreven:

```
#include <stdio.h>

main() {

    int c;

    while ((c = getchar()) != EOF)
        putchar(c);
}
```

Het programma leest een teken, assigneert het aan c en test dan of het teken het einde-file teken is. Als dit niet het geval is dan wordt de body van de while geëxecuteerd, het teken wordt afgedrukt. De while-loop wordt dan herhaald. Als het einde van de input is bereikt dan stopt de while-loop ook en tevens stopt het programma dan.

In deze versie is er maar één input statement tegenover twee en dus is het programma compacter geworden.

Het is belangrijk om op te merken dat de haakjes om de assignment in de conditie echt noodzakelijk zijn. De precedence (een soort Mijnheer Van Dalenregels, maar dan voor meerdere tekens en in het Engels) van != is hoger (zoals * voor + gaat) dan die van =, wat betekent dat als de haakjes niet aanwezig waren de != test eerder wordt uitgevoerd dan de assignment =. Dus het statement

c = (getchar() != EOF)

is equivalent met

c = getchar() != EOF

Een verkeerd gebruik van haakjes heeft een ongewenst effect op de waarde van c, deze kan dan namelijk nog slecht 0 of 1 worden (hier wordt later nog op terug gekomen), alle andere waarden die een teken kan hebben kunnen dan niet meer voorkomen.

Het tellen van tekens
Het volgende programma telt tekens; dit programma is maar een kleine wijziging van het vorige.

```
telchar.c

#include <stdio.h>

main()
{
    long nc;

    nc = 0;
    while (getchar() != EOF)
        ++nc;
    printf("%ld\n",nc);
}
```

Het statement

++nc;

introduceert een nieuwe operator, '++'. Deze nieuwe operator verhoogt het argument met één. U kan ook 'nc = nc + 1' schrijven maar '++nc' is compacter en vaak ook efficiënter. Er is ook een '--' operator en die verlaagt de variabele met een. De operatoren '++' en '--' zijn beide prefix operatoren (++nc, ++ staat voor nc), maar kunnen ook postfix zijn (nc++, ++ staat achter nc), deze twee vormen hebben andere waarden in expressies, zoals later beschreven zal worden, maar beide verhogen nc met één. Nu zullen we ons nog bij de prefix houden.

4.2 Inleiding

Het karaktertel programma gebruikt een long integer in plaats van een integer. Het maximum van een integer ligt op de 32767 en met relatief weinig input zou een programma met een integer variable over zijn maximum heen gaan. Met een long integer is het maximum 2 miljard, hier zal niet direct over heen gegaan worden.

Het tellen van regels
Het volgende programma telt regels van de invoer. Van de regels wordt verwacht dat ze door een RETURN worden afgesloten, dus met het newline (\n) teken.

```
#include <stdio.h>

main()

{
    int c, nr;

    nr = 0;
    while ((c = getchar()) != EOF)
        if (c == '\n')
            ++nr;
    printf("%d\n",nr);
}
```

De body van de while bevat het if-statement, dit statement bepaald de incrementie (verhoging) van nl. Het if-statement test de conditie tussen de haakjes (c = = '\n') en als deze true (waar) is dan wordt het daarop volgende statement of groep van statements tussen de parentheses ({ of }) uitgevoerd.

Het dubbele = teken == is de C notatie voor 'is gelijk aan'. Dit symbool wordt gebruikt om verschil aan te geven tussen de assignment (het = symbool wordt dan gebruikt) en de gelijkheidstest. Elk teken moet tussen enkele quotes (') worden geschreven, dit wordt een character constant (tekenconstante) genoemd. Bijvoorbeeld, 'A' is een character constant; in de ASCII tekenset is 65 hiervan de waarde, ditistevensdeinternerepresentatievanA. De backslash (\) tekens worden eveneens tussen de enkele quotes gezet (zie boven).

Het tellen van woorden
Het derde programma in de handige serie van regel-, woord- en lettertelprogramma's, zal al deze gegevens in eens bepalen. Hier zullen we een woord definiëren als een groep tekens die geen spatie of een newline-teken (return) bevatten.

Elke keer dat het programma het eerste teken van een woord tegen komt wordt het woord geteld. De variabele inwoord houdt bij of het programma in een woord zit of niet; in het begin zit het programma niet in een woord en krijgt inwoord de waard NEE. De symbolische constanten JA en NEE worden gebruikt om het programma beter leesbaar te houden.

De regel

nl = nw = nc = 0;

zet alle drie de variabelen op nul. Dit is geen speciaal geval, maar een direct gevolg van het feit dat een assignment de waarde heeft van het rechter deel. Eigenlijk is dit het hetzelfde als

nc = (nl = (nw = 0));

De operator || betekent OR (of), dus de regel

if (c = = ' ' || c = = '\n')

betekent; als c is een spatie of c is een newline. Er is ook een && operator die staat voor AND (en). Expressies die met && of || worden verbonden worden van links naar rechts uitgerekend en stoppen zodra er een uitspraak over true of false kan worden uitgesproken. Dus als c een spatie is behoeft er niet getest te worden of c een newline bevat en wordt die test

4.2 Inleiding

```
#include <stdio.h>

#define  JA 1
#define NEE 0

main()      /* telt regels, worden, karakters uit de input */

{
    int c, nl, nw, nc, inwoord;

    inwoord = NEE;
    nl = nw = nc = 0;
    while ((c = getchar()) != EOF) {
        ++nc;
        if (c == '\n')
            ++nl;
        if (c == ' ' || c == '\n')
            inwoord = NEE;
        else if (inwoord == NEE) {
            inwoord = JA;
            ++nw;
        }
    }
    printf("%d %d %d\n",nl,nw,nc);
}
```

dan ook niet gedaan. Hier is dat niet van groot belang maar in meer complexere situaties zal dit wel degelijk het geval zijn.

Het voorbeeld laat tevens het C-else statement zien, dit specificeert een alternatieve actie als de conditie bij het if-statement false is. De algemene form is

```
if (expressie)
   statement-1
else
   statement-2
```

Slechts één van de twee statements wordt geëxecuteerd. Als de expressie waar is dan wordt statement-1 geëxecuteerd; anders wordt statement-2 geëxecuteerd. Elk statement kan zo gecompliceerd worden gemaakt als wenselijk is, dan behoeven slechts de afzonderlijke onderdelen (het true en false gedeelte, statement-1 en 2) tussen braces gezet te worden.

8/4.2.5
Arrays

Onderstaand programma is nóg een programma uit de tel-programma familie. Dit programma telt het aantal keren dat een bepaald cijfer wordt gebruikt en al de keren dat een blanco (hier een spatie of een newline) wordt gebruikt, tevens worden de overige tekens ook nog geteld. Met dit simpele programma wordt het array concept uit C geïllustreerd.

Omdat er tien verschillende cijfers zijn is het handig om een array te gebruiken

4.2 Inleiding

die voor elk getal bijhoudt hoe vaak dit al is gebruikt. Dit is eenvoudiger dan het gebruik van tien aparte variabelen die het zelfde doen.
Het programma ziet er dan als volgt uit.

```
#include <stdio.h>

main()    /* tel cijfers, blanco en andere karakters */

{
    int c, i, blanco, ander;
    int cijfers[10];

    blanco = ander = 0;
    for (i = 0;i < 10; ++i)
        cijfers[i] = 0;

    while ((c = getchar()) != EINDEIN)
        if (c >= '0' && c <= '9')
            ++cijfers[c-'0'];
        else
            if (c == ' ' || c == '\n')
                ++blanco;
            else
                ++ander;

    printf("cijfers =");
    for (i = 0;i < 10; ++i)
        printf(" %d", cijfers[i]);
    printf("\nblanco = %d, ander = %d\n",
            blanco,ander);
}
```

De declaratie

int cijfers [10];

declareert (bekend maken aan het programma) een array cijfers met tien integers. Een array begint altijd met element 0 in C, dus de array elementen zijn: cijfers [0], cijfers [1], . . ., [9]. Dit is ook te zien in de for-loop die alle array-elementen op nul intialiseert.

Een array element mag elke integer expressie zijn.

Dit programma maakt bijzonder gebruik van het feit dat tekens intern, in de machine, door getallen worden voorgesteld. Bijvoorbeeld de test:

if (c >= '0' && c <= '9')

bepaalt of het karakter c een getal is. Als dat dan zo is dan is de numerieke waarde van c gelijk aan

c-'0'

Dit werkt alleen als '0', '1', etc. elkaar opvolgen en er geen andere tekens tussen '0' en '9' zitten. Gelukkig is dit het geval voor alle normale tekensets en

4.2 Inleiding

ook voor de tekenset van de Commodore 64.

Per definitie is de berekening met tekens (van het type char) gelijk aan de berekening van integers. Dus c-'0' is een integer expressie met een waarde tussen 0 en 9, die correspondeert met het teken '0' en '9' wat in c zit, dit is dus een geldig element voor de array cijfers.

De beslissing of het teken een cijfer, een blanco of iets anders is wordt gemaakt met de regel:

```
if (c >= '0' && c <= '9')
  ++ cijfers[c-'0'];
else if (c == ' ' || c == '\n')
  ++ blanco;
else
  ++ ander;
```

Het patroon

```
if (conditie)
  statement
else if (conditie)
  statement
else
  statement
```

komt vaak voor in programma's om een meerwegs beslissing uit te drukken. Hier zijn er drie wegen, te weten: cijfer òf blanco òf anders. De code is eenvoudig van boven naar beneden te lezen totdat een conditie true is en alleen het daarmee corresponderende statement wordt geëxecuteerd. Is geen van de condities waar dan wordt het statement na de laatste else geëxecuteerd. Wordt de laatste else met bijbehorend statement weg gelaten dan wordt er niets gedaan als geen van de condities waar is. Er kunnen zoveel

```
else if (condities)
  statement
```

constructies voor komen als u wenst.

Het switch statement, wat later volgt, heeft een soortgelijke functie. Later zal dit zelfde programma nog eens volgen maar dan met het switch statement.

Met de onderstaande regels wordt de waarde op het scherm afgedrukt.

```
printf ('cijfers =');
  for (i = 0;i < 10; ++i)
    printf (" %d", cijfers [i]);
printf ('\nblanco = %d, ander = %d\n",
blanco, ander);
```

8/4.2.6
Functies

In C is de functie hetzelfde als de GOSUB in BASIC, of hetzelfde als de procedure uit Pascal. Een functie is een geschikte vorm om zogenaamde 'black boxes' te maken. Deze 'black box' kan een handeling verrichten zonder dat het voor het programma, dat het aanroept, behoeft te weten hoe deze 'black box' (functie) werkt. Het gebruik van functies is trouwens werkelijk noodzakelijk als men grote programma's gaat schrijven. Als een functie goed ontworpen is, is het voldoende om te weten wat een functie doet in plaats van te weten hoe de functie dat doet. C is dusdanig ontworpen dat men eenvoudig functies kan gebruiken, vaak ziet men ook dat een functie van een paar regels maar één keer wordt aangeroepen, enkel en alleen omdat de C-code dan duidelijker wordt.

4.2 Inleiding

Tot nu toe heeft u alleen nog maar de functies printf, getchar en putchar gezien, maar nu wordt het toch tijd dat er een paar eigen geschreven functies bijkomen. Omdat C niet kan machtsverheffen, zal een machtsverhef functie besproken worden. De functie power (m, n) verheft een integer m tot de positieve integer macht n. Dus power (2,5) is 32.

Hieronder volgt de functie power met een main programma dat het aanroept.

```
main()              /* test power functie */
{
    int i;

    for (i = 0; i < 10; ++i)
        printf("%d %d %d\n", i, power(2,i)
                ,power(-3,i));
}

power(x, n)         /* verhef x tot de macht n: n > 0 */

int x, n;
{
    int i, p;

    p = 1;
    for (i = 1; i <= n; ++i)
        p = p * x;
    return(p);
}
```

4.2 Inleiding

Elke functie heeft de zelfde vorm:

```
naam (variabelen-lijst, als nodig)
variabelen declaratie, als nodig

{
   declaraties
   statements
}
```

De functies mogen in willekeurige volgorde verschijnen, ze mogen zelfs over meerdere source files verdeeld worden. Als ze over verschillende source files verdeeld worden moeten de gecompileerde versies van deze files weer aan elkaar geLINKt worden.

De functie power wordt twee maal geCALLd in de regel:

```
printf ("%d %d %d\n", i, power (2,i),
power (-3,i));
```

printf moet hier drie integerexpressies afdrukken en de beide power functiecalls (aanroepen van een functie) zijn geldige integerexpressies. De functie power geeft hier een integerwaarde af, maar de af te geven waarde mag van elk type zijn, later wordt hier verder op ingegaan.

In power worden de argumenten normaal gedeclareerd, zodanig dat de typen van deze argumenten bekend zijn. Dit gebeurt in de regel:

```
int x, n;
```

deze regel volgt na de functie naam. De declaratie van de argumenten staat tussen de naam van de functie en de eerste open-linker brace ({); elke declaratie wordt afgesloten door een semicolon (;). De namen die power intern voor zijn variabelen gebruikt bestaan enkel en alleen binnen de functie! Dit betekent dat als de functie RETURNt de variabelen en hun waarden niet meer bestaan. Andere functies kunnen ook dezelfde variabele namen gebruiken zonder dat dit tot conflicten leidt. Zo is bijvoorbeeld de variabele i uit power onafhankelijk van de variabele i uit main.

De waarde die power berekent wordt aan main afgegeven door het return-statement. Elke expressie mag tussen de haakjes van return staan. Het is niet noodzakelijk dat een functie een waarde terug geeft; een return zonder expressie geeft geen waarde af maar de functie wordt dan wel beëindigd, het programma-gedeelte dat de functie had aangeroepen gaat dan verder waar het gebleven was. Dit RETURNen zonder waarde is het zelfde als het beëindigen van een functie als de executie tegen de laatste rechter-sluit-brace 'aanloopt'.

8/4.2.7
Argumenten – Call by Value

Een van de aspecten van C die anders zijn in andere talen, is het feit dat alle functie-argumenten 'by value' worden door gegeven. Dit betekent dat de functie de waarden van de argumenten in tijdelijke variabelen (ze staan op stack) opslaat en de functie de waarden van de variabelen, in het stuk programma van de aanroep, niet kan veranderen. Het adres van de variabele wordt namelijk niet met de functie-aanroep doorgegeven maar de waarde van de variabele. Dit is anders dan het 'call by reference' principe dat b.v. in Fortran gebruikt

wordt, waarbij het adres de variabele (een argument is een variabele die in de parameter-lijst van de functie staat) wordt doorgegeven.

Het belangrijkste verschil is dat in C de functie de variabele van het aanroepende stuk programma niet kan veranderen. De variabele binnen de functie is slechts een tijdelijke kopie van de orginele variable en de functie kan alleen zijn eigen tijdelijke variabele veranderen.

Het gevolg van het 'call by value' mechanisme is dat de programma's vaak compact geschreven kunnen worden zonder veel extra variabelen, omdat de argumenten als normale variabelen te gebruiken zijn die reeds geïnitialiseerd zijn. Hieronder volgt een programma dat hiervan gebruik maakt.

```
power(x, n)        /* verhef x tot de
                      macht n: n > 0
                      versie 2       */
int x, n;
{
    int p;

    for (p = 1; n > 0; --n)
        p = p * x;
    return(p);
}
```

Het argument n wordt als tijdelijke variabele gebruikt en telt naar beneden totdat het nul wordt; de variabele i uit het vorige voorbeeld is niet meer nodig. Wat er intern de functie power ook met n gebeurt, het heeft geen invloed op het argument waarmee power orgineel was geCALLd.

Als het noodzakelijk is, is het mogelijk dat een functie een variabele verandert die in de aanroepende routine staat. De CALLER moet dan het adres van de variabele aan de functie door geven (dit is een pointer, een wijzer naar die variabele) en de geCALLde functie moet tevens dat argument als pointer naar een variabele declareren. Die pointer wijst dan naar die variabele. In een later hoofdstuk wordt hier nog op terug gekomen.

Als de naam van een array als argument wordt gebruikt dan wordt alleen het adres van het begin van de array door gegeven (er worden dan geen elementen van de array gekopieerd). Door dit adres te gebruiken kan de functie (bij uitzondering) dus wel elementen van de array veranderen. Dit is het onderwerp van de volgende sectie.

8/4.2.8 Character arrays

Het meest gebruikte type array in C is het character array. Om te illustreren hoe ze gebruikt worden in combinatie met functies wordt het volgende pro-

4.2 Inleiding

gramma geschreven. Het programma leest regels en drukt de langste af. Het hart van de routine is simpel genoeg:

while (er nog een andere regel is)
  if (het langer dan de langste is)
  bewaar het met de lengte
print de langste regel

Dit maakt al direct duidelijk dat het programma in verschillende delen uiteenvalt. Eén gedeelte dat de nieuwe regel leest, een ander dat het test, een volgende dat het bewaart en de rest dat het proces controleert.

Omdat de onderdelen zo mooi uit elkaar vallen is het goed om de onderdelen los van elkaar te schrijven. Getline is een functie die een regel binnen moet lezen, getline is een generalisatie van getchar. Om de functie ook in andere situaties te kunnen gebruiken zal de functie zo flexibel mogelijk geschreven worden. Getline moet de lengte van de regel afgeven of een nul afgeven als het EOF teken wordt gelezen; een regel heeft nooit de lengte nul want deze bestaat altijd uit minstens één teken, namelijk de newline.
Als er een regel wordt gevonden die langer is dan de vorige langste dan moet deze ergens bewaard worden. Een tweede functie, copy, kopieert de nieuwe regel naar een bepaalde buffer.

Uiteindelijk is er nog een programma nodig dan getline en copy bestuurt. Hieronder volgt het resultaat:

```
#include <stdio.h>
#define MAXLENGTE 1000 /* max regel groote */
main()              /* vindt de langste regel */
{
    int  len; /* huidige regel lengte */
    int  max; /* max regel lengte     */
    char line[MAXLENGTE]; /* input regel   */
    char save[MAXLENGTE]; /* langste regel  */

    max = 0;
    while ((len = getline(line, MAXLENGTE)) >0)
        if (len > max) {
            max = len;
            copy(line, save);
        }
    if (max > 0)  /* er was een regel */
        printf("%s", save);
}

getline(s, lim)  /* voer regel in s, return de lengte */
char s[];
int lim;
{
```

4.2 Inleiding

```
    int c, i;

    for (i = 0;i <lim-1 && (c = getchar()) != EOF
          && c != '\n'; ++i)
        s[i] = c;
    if (c == '\n') {
        s[i] = c;
        ++i;
    }
    s[i] = '\0';
    return(i);
}

copy(s1, s2)      /* copieer s1 naar s2;
                     veronderstel s2 groot genoeg */
char s1[],s2[];
{
    int i;

    i = 0;
    while ((s2[i] =s1[i]) != '\0')
        ++i;
}
```

Main en getline communiceren beide met één paar argumenten en een geRETURNde waarde. In getline worden de argumenten gedeclareerd met de regels:

char s[];
int lim;

die specificeren dat het eerste argument een array is en het tweede een integer. De lengte van de array s wordt niet gespecificeerd in getline omdat dit al in main is gedaan. Getline maakt gebruik van return om een waarde naar main terug te sturen net zoals de power functie dit deed. Sommige functies RETURNen een bruikbare waarde, anderen, zoals copy, worden alleen vanwege het effect dat ze teweeg brengen gebruikt en RETURNen geen waarde.

Getline zet het teken \0 (het nul-teken, de waarde hiervan is 0) aan het einde van de string om het einde van deze string aan te geven. De compiler doet hetzelfde en hier wordt bij het afdrukken weer gebruik van gemaakt. Een string:

'hallo \n'

wordt door de compiler als een array van tekens opgeslagen die wordt beëindigd met een nul, zodat printf het einde van de string kan detecteren. Het wordt dan als volgt opgeslagen:

h a l l o \n \0

Het %s formaat specificeert in printf een string als argument en printf verwacht de string in dit formaat. Als u copy bekijkt dan ziet u dan van dit feit gebruik gemaakt wordt.

4.2 Inleiding

Aan dit programmaatje zitten nog wel wat haken en ogen. Wat gebeurt er als er een langere string wordt ingetikt dan de maximum lengte? Hier wordt niets aan het ondervangen van dit probleem gedaan, maar met een simpele test op de maximum lengte en een eventuele foutboodschap is dit probleem al een stuk beter ondervangen.

8/4.2.9
Scope; Externe variabelen

De variabelen in main (line, save, etc) zijn privé of lokaal voor main; ze zijn namelijk in main gedeclareerd, geen enkele andere functie kan deze variabelen direct bereiken. Hetzelfde is waar voor de variabelen van andere functies; bijvoorbeeld de variabele i in getline is ongecorreleerd aan de i uit copy. Elke lokale variabele in een routine bestaat alleen maar in de functie waar deze is gedeclareerd en deze variabele verdwijnt helemaal als de functie beëindigt. Om deze reden worden deze variabelen automatische variabelen genoemd. Deze terminologie zal hier in het vervolg ook gebruikt worden om deze dynamische lokale variabelen aan te duiden.

Omdat deze automatische variabelen komen en gaan met de functie-aanroep, blijven hun waarden niet bewaard tussen de ene en de andere call en moeten ze expliciet bij elk begin van de functie opnieuw een waarde krijgen toegekend.

Een alternatief voor automatische variabelen zijn de external variabelen, deze external variabelen zijn extern voor alle functies, ze zijn dan voor iedere functie toegankelijk door alleen de naam van de variabele te gebruiken. Omdat de externe variabelen vanuit elke functie te gebruiken zijn, kunnen ze gebruikt worden om data tussen de functies door te geven. Omdat de externe variabelen niet verdwijnen met het beëindigen van een functie behouden ze hun waarde.

Een externe variabele moet buiten elke functie gedefiniëerd worden. De variabele moet ook in elke functie gedeclareerd worden, als de functie er tenminste gebruik van maakt. Dit moet gebeuren met het keywoord extern. Het programma langste regel is herschreven om het zojuist vertelde te illustreren, zie hier onder:

```
longestline2.c

#include <stdio.h>
#define MAXLENGTE 1000 /* max regel groote */

char line[MAXLENGTE];   /* input regel       */
char save[MAXLENGTE];   /* langste regel     */
int  max;   /* lengte van de langste regel   */

main()              /* vindt de langste regel
                       speciale versie        */
{
   int  len; /* huidige regel lengte */
```

4.2 Inleiding

```
    max = 0;
    while ((len = getline()) >0)
        if (len > max) {
            max = len;
            copy();
        }
    if (max > 0)   /* er was een regel */
        printf("%s", save);
}

getline(s, lim)   /* speciale versie */
{
    int c, i;
    extern char line[];

    for (i = 0;i <MAXLENGTE-1
          && (c = getchar()) != EOF
          && c != '\n'; ++i)
        line[i] = c;
    if (c == '\n') {
        line[i] = c;
        ++i;
    }
    line[i] = '\0';
    return(i);
}

copy(s1, s2)     /* speciale versie */
{
    int i;
    extern char line[], save[];

    i = 0;
    while ((save[i] = line[i]) != '\0')
        ++i;
}
```

De externe variabelen in main, getline en copy worden gedefinieerd in de eerste regels, met hun type en de ruimte die ze in nemen. Syntactisch worden de externe variabelen op een zelfde manier gedeclareerd als gewoonlijk, maar nu staan ze alleen buiten elke functie en zijn ze dus extern. Voordat een functie gebruik van een externe variabele kan maken moet deze eerst bekend zijn in de functie en dit gebeurt m.b.v. het keyword extern voor de declaratie te schrijven.

In bepaalde omstandigheden kan bij de declaratie extern weg gelaten worden, dit kan gebeuren als de extern variabele in dezelfde file wordt gedeclareerd als de functie die hem aanroept. Hier zijn de keywords extern dus redundant (overbodig).

4.2 Inleiding

N.B. er is een tendens om alle variabelen maar extern te maken, omdat dit de communicatie zou vereenvoudigen, de argument-lijst zou korter kunnen worden en de variabelen zijn er altijd wanneer men dat wil. Externe variabelen zijn er echter ook als dit niet gewenst is. Dit is een foutieve manier van programmeren want het wordt dan moeilijk om programma's te veranderen. De tweede versie van de langste regel is inferieur aan de eerste, gedeeltelijk om deze reden en gedeeltelijk omdat het de generaliteit van twee bruikbare programma's vernietigt door ze aan namen van variabelen vast te knopen.

8/4.2.10 Samenvatting

Op dit punt moet u in staat zijn om met de aangereikte bouwstenen zelf een aardig C-programmaatje te schrijven. U bent nu al enigszins bekend met C, wat oefeningen kunnen u helpen de C-materie beter te begrijpen. Nu is het misschien het juiste moment om eens wat beter naar de geherdefinieerde getchar functie te kijken. U kunt nu nog niet alles begrijpen, maar er moet al veel duidelijk zijn.

Zodra u dit goed onder de knie heeft wordt het tijd om een meer gedetailleerde beschrijving van C te lezen. Deze gedetailleerde beschrijving van C begint in de volgende aanvulling.

8/4.3

Typen, operatoren expressies

Variabelen en constanten zijn de elementaire data objecten waarmee in een programma wordt gemanipuleerd. Met de declaratie worden de variabelen, die gebruikt worden, aan de compiler bekend gemaakt. Het type is dan bekend en eventueel hun initiële waarde. Operatoren specificeren wat er met de variabelen en constanten gedaan wordt. Expressies combineren variabelen en constanten, zodat nieuwe waarden geproduceerd kunnen worden. De hierboven genoemde onderwerpen worden in dit hoofdstuk behandeld.

3/4.3.1
Namen van variabelen

Aan het kiezen van variabele-namen zijn bepaalde restricties verboden. Namen bestaan uit letters en cijfers; het eerste teken is een letter. De underscore "_" telt als een letter; dit teken is erg bruikbaar bij het kiezen van betekenisvolle namen, de underscore verhoogt de leesbaarheid van een naam. C maakt een onderscheid tussen grote en kleine letters; traditioneel worden kleine letters voor variabelen gebruikt en grote letters voor symbolische constanten.

Slechts de eerste acht tekens van een naam zijn significant, al mogen ze wel tot 255 tekens lang zijn. Tevens zijn keywords zoals if, else, int, while, etc., gereserveerd, dus ze mogen niet als naam van een variabele gebruikt worden.

Het is uiteraard verstandig om namen van variabelen te kiezen die iets betekenen, dat ze dus gerelateerd zijn aan het doel waar ze voor gebruikt worden.

3/4.3.2
Data-typen

C kent slechts enkele standaard data-typen:

char	één byte, gebruikt voor de opslag van tekens
int	een integer, hier worden gehele getallen in opgeslagen
float	gebruikt voor enkele preciesie drijvende komma getallen

Er bestaat een aantal qualifiers (extra benamingen) die op ints toegepast mogen worden: short, long en unsigned, maar dit heeft op de Commodore geen effect want het geheugengebruik verandert niet. Het data-type double is op de Commodore 64 gelijk aan de float. Het geheugengebruik per gebruikte variabele is:

Aantal Bytes	Type
1	char
2	short
2	int
2	long

2	unsigned
5	float
5	double
2	pointer

Dus short, int en long zijn synoniem, evenals float en double.
Het geheugengebruik per type is erg machine afhankelijk, bijvoorbeeld op een
Honeywell 6000 neemt het chartype 9 bits in beslag, dit is een hoogst ongebruikelijk aantal bits maar in C zijn deze waardes geheel vrij gelaten.

8/4.3.3 Constanten

Constanten nemen een belangrijke plaats in een programma in. Met int- en float-constanten is al kennis gemaakt, behalve dat de float op verschillende manieren geschreven kan worden.

123.456e-7

en

0.123456E-10
zijn beide legale notaties.

Octale constanten worden met een 0 (nul) aan het begin geschreven, zo is 010 (octaal) gelijk aan 8 (decimaal). Hexadecimale constanten worden met 0x of 0X aan het begin geschreven. Bijvoorbeeld 31 (decimaal) is 037 (octaal) of 0x1f of 0X1F (hexadecimaal).
Een character-constant wordt als een teken tussen single quotes geschreven. Behalve bepaalde niet zichtbare tekens zoals \n (newline), \0 (nul), \\(backslash), \' (enkele quote), etc. Voorbeelden van character constanten zijn dan: 'a' 'Q' '\n'.

Een string-constant is een rij van nul of meer tekens, die door dubbele quotes wordt ingesloten. Zoals

"Ik ben een string"

of

"" /* de null string of lege string */

De quotes maken geen deel uit van de string, als er toch een quote in een string moet komen, moet \" worden gebruikt om een quote in de string te krijgen.

Technisch gezien is een string een array wiens elementen uit enkele tekens bestaan. De compiler plaatst automatisch \0 aan het einde van een string, zodat programma's het einde van de string kunnen bepalen. Met de volgende functie kan dan het einde van een string worden bepaald. De functie krijgt als invoer een char array s, ze telt net zolang tekens totdat het null teken is bereikt.

```
strlen(s) /* lengte van string */
char s[];
{
   int i;

   i = 0;
```

ATTENTIE: merk op dat er een verschil is tussen een character constant ('f') en een string constant met één teken ("f"). De eerste constante is een teken en de tweede constante is een string met een teken (de letter f) en een \0.

8/4.3.4 Declaraties

Alle variabelen moeten gedeclareerd worden voor ze gebruikt kunnen worden, alhoewel sommige declaraties impliciet kunnen gebeuren als ze in de juiste context gebruikt worden. Een declaratie specificeert een type en wordt gevolgd door een lijst van variabelen, die van dat type zijn, zoals

```
int onder, boven, stap;
char c, line[1000];
```

De variabelen mogen ook elk met hun type gedeclareerd worden, zoals

```
int onder;
int boven;
int stap;
char c;
char line[1000];
```

Deze laatste vorm neemt meer ruimte in beslag, maar zo kan er wel commentaar achter elke declaratie geschreven worden.

Variabelen mogen ook geïnitialiseerd worden tijdens hun declaratie. Op de naam van de variabele moet dan een = teken met een constante volgen, bijvoorbeeld

```
char backslash = '\\';
int i = 0;
float eps = 1.0e-5
```

Als de variabelen als external of als static worden gedefinieerd wordt de initialisatie één maal gedaan, namelijk vóór het programma met de executie begint. Als variabelen geen waarde tijdens de declaratie krijgen dan is hun waarde onbepaald.

8/4.3.5 Rekenkundige operatoren

De normale +,–,/,* operatoren en de modulus operator % worden in C gebruikt.
Met de modulus operator wordt de rest bij een deling tussen integers bepaald. De expressie

```
13 % 4
```

levert 1 op, want 13 gedeeld door 4 is 3 rest 1. Deze modulus-deling levert nul op als het eerste argument een veelvoud van het tweede is. Bijvoorbeeld: een jaar is een schrikkeljaar als het deelbaar is door 4, maar niet als het deelbaar is door 100, behalve dan weer als het deelbaar is door 400. Dus

```
if (jaar % 4 == 0 && jaar % 100 != 0 ||
jaar % 400 == 0)
   "het is een schrikkeljaar"
```

else
"het is geen schrikkeljaar"

De modulus operator kan niet op floats of doubles worden toegepast.

De precendence (een soort mijnheer Van Dalen regels) van + en - is gelijk, maar is lager dan de precendence van *,/ en %, welke weer lager is dan de - die voor een variabele of constante staat (negatie-operator). Dus

4*-3/2+1

is gelijk aan

(4*(-3)/2)+1

8/4.3.6 Relationele en logische operatoren

De relationele operatoren zijn

> >= < <=

Ze hebben allemaal dezelfde precedence. Een iets lagere precedence hebben de equality operatoren

== !=

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0

main() {
    int c;

    while ((c = get_int()) != 0 ) {
        printf("%d ",c);
        while (c != 1 && c != 0) {
            if (c % 2) {
                c = (3*c)+1;
            }
```

welke ten opzichte van elkaar weer dezelfde precedence hebben. De relationele operatoren hebben een lagere precedence dan de rekenkundige operatoren, dus de expressie i <j+1 wordt gelezen als i < (j+1), zoals verwacht mocht worden.

De logische operatoren, || (of) en && (en), zijn interesanter. De expressies die door && of || worden verbonden worden van links naar rechts geëvalueerd totdat waar of niet-waar wordt gesignaleerd.

Met al deze kennis kan het onderstaande programma geschreven worden, dat de SYRACUSE rij van een integer bepaalt. Deze reeks verloopt als volgt: als een getal oneven is dan wordt dit met 3 vermenigvuldigd en wordt daar 1 bij opgeteld. Is het getal even dan wordt het door 2 gedeeld. Na deze bewerking wordt op de uitkomst dezelfde procedure toegepast en deze reeks stopt als 1 wordt bereikt. Bijvoorbeeld, de beginwaarde 7 levert de syracuse rij:
7 22 11 34 17 52 26 13 40 20 10
5 16 8 4 2 1

4.2 Typen, operatoren en expressies

```
          else {
             c = c/2;
          };
          printf("%d ",c);
       };
       printf("\n");
    }
}

get_int()
{
    int i;

    scanf("%d",&i);
    return i;
}
```

Het programma stopt als 0 wordt ingevoerd, dezelfde fout in de getchar( ) bibliotheek functie zit in de scanf( ) functie, maar omdat de reeks van 0 niet interessant is wordt 0 voor het beëindigen van het programma gebruikt. Dit programma test niet op een eventuele integer overflow, dit is een integer die groter dan 32767 wordt. Er kan dus alleen met kleine beginwaarden worden begonnen.

8/4.3.7 Type-conversies

Als de operanden uit een expressie van verschillende type zijn, dan worden ze naar hetzelfde type omgezet, volgens een paar kleine regels. In het algemeen worden alleen de conversies uitgevoerd die zinvol zijn, zoals een conversie van een integer naar een floating point. Expressies die zinloos zijn, zoals een float gebruiken als array index, zijn niet toegestaan.

Ten eerste mogen chars en ints door elkaar gebruikt worden, elke char wordt automatisch naar een int geconverteerd. Dit verhoogt de flexibiliteit van de char conversies aanzienlijk. Een voorbeeld hiervan is de functie atoi (dit staat voor alfa to integer), deze functie zet een string van getallen om in de equivalente integer waarde.

4.3 Typen, operatoren en expressies

```
atoi(s) /* converteer s naar int */
char s[];
{
   int i, n;

   n = 0;
   for (i = 0; s[i] >= '0' && s[i] <= '9'; ++i)
      n = 10 * n + s[i] - '0';
   return(n);
}
```

Zoals al eerder is genoemd levert de expressie
s[i]-'0'
de numerieke waarde van het teken dat in s[i] is opgeslagen.

Een ander voorbeeld van een char-naar-int conversie is de functie lower, die een hoofdletter omzet in een kleine letter. Als het teken geen hoofdletter is dan blijft het teken onveranderd.

```
lower(c) /* omzetten naar kleine letters */
int c;
{
   if (c >= 'A' && c <= 'Z')
      return (c + 'a' - 'A');
   else
      return(c);
}
```

Een ander handig gevolg van automatische type-conversie is dat relationele expressies zoals i>j en logische expressies die door && en || worden verbonden de waarde 1 voor true (waar) en 0 voor false (niet waar) hebben. Dus de assignment

isgetal = c>='0' && c<='9';

maakt isgetal tot 1 als het getal is en anders tot 0 (in het test gedeelte van if, while, for, etc. betekent true hetzelfde als niet-nul).

8/4.3.8 Increment- en decrement-operatoren

C heeft twee ongebruikelijke operatoren voor het incrementeren (verhogen) en decrementeren (verlagen) van variabelen. De increment operator ++ telt 1 bij zijn operand op, de decrement operator -- trekt 1 van zijn operand af.

Ook is het ongebruikelijk dat ++ en -- als prefix operator (voor de variabele, bijvoorbeeld ++n), of als postfix operator (na de variabele, bijvoorbeeld n++) bestaan. In beide gevallen wordt n geïncrementeerd. Maar in de expressie ++n

wordt n geïncrementeerd VOORDAT zijn waarde gebruikt wordt, terwijl n++ n pas incrementeert NADAT zijn waarde is gebruikt. Dit betekent dat waar de waarde van n wordt gebruikt n++ niet hetzelfde effect heeft als ++n. Stel n is 5,

x = n++;

assigneert 5 aan x, maar

x = ++n;

assigneert 6 aan x. In beide gevallen wordt n 6. Er zijn dus situaties waar het belangrijk is welke vorm wordt gekozen zoals ook in het onderstaande voorbeeld. Hier worden alle chars c uit de string s verwijderd.

```
remove(s, c) /* verwijder alle c's uit s */
char s[];
int c;
{
    int i, j;

    for (i = j = 0; s[i] != '\n'; i++)
        if (s[i] != c)
            s[j++] = s[i];
    s[j] ='\0';
}
```

Dit is tevens een goed voorbeeld van de complacte schrijfwijze die met deze operatoren valt te behalen. Elke keer dat een niet-c voorkomt, wordt het teken naar de huidige positie van j gekopieerd en alleen dan wordt j geïncrementeerd. Het if-blok is hetzelfde als

```
if (s[i] != c) {
    s[i] = s[i];
    j++;
}
```

8/4.3.9
Logische operatoren op bitniveau

C heeft een aantal operatoren voor operaties op bitniveau, deze mogen niet op floats en doubles worden toegepast.

&	AND
|	OR <logo> *
↑	EXCLUSIVE OR
<<	SHIFT LEFT
>>	SHIFT RIGHT
~	COMPLEMENT, NOT

<logo> p

De AND operator & wordt vaak gebruikt om een bitmasker te zetten, bijvoorbeeld

c = n & 0177; /* octaal, dus 001111111 */

selecteert de 7 minst significante bits van n. De OR operator | wordt juist gebruikt om bits aan te zetten, bijvoorbeeld

x = n | 0xf;

8/4.3.10 Assignment-operatoren en expressies

Expressies zoals

i = i + 2

waarin de linkerkant van de expressie weer in de rechterkant voorkomt, kunnen verkort geschreven worden als

i += 2

Hierbij wordt de assignment-operator += gebruikt.

De meeste binaire operatoren (operatoren met een linker en rechter operand) hebben een corresponderende assignment operator $op=, waar $op een van de volgende binary operatoren is

+ - * / % << >> & ↑ |

Als exp1 exp2 expressies zijn, dan is

exp1 $op=exp2

equivalent met

exp1 = (exp1) $op (exp2)

Dus met de haakjes om exp2 is

x *= y + 1

gelijk aan

x = x * (y + 1)

in plaats van

x = x * y + 1

Als voorbeeld staat hier de functie bitcount, deze functie telt het aantal 1-bits in een integer.

```
bitcount(n) /* telt 1 bits in n */
unsigned n;
{
    int b;

    for (b = 0; n != 0; n >>= 1)
        if (n & 01)
            b++;
    return(b);
}
```

Deze assignment-operatoren hebben als voordeel dat ze veel lijken op de manier zo als mensen denken. Men zegt 'tel 2 op bij i', en niet 'pak i, tel er 2 bij op, stop het resultaat terug in i'.

8/4.3.11 Conditionele expressies

Met de statements

```
if (a > b)
   z = a;
else
   z = b;
```

wordt het maximum van a en b bepaald en wordt dit in z opgeslagen. Een andere manier om conditionele constructies te schrijven wordt geboden door de "?:" operator. In de expressie

```
exp1 ? exp2 : exp3
```

wordt eerst exp1 geëvalueerd. Als deze true is, dan wordt exp2 geëvalueerd en die levert dan de waarde van de conditionele expressie. Anders wordt exp3 geëvalueerd, en die levert deze de waarde. Slechts één expressie van exp 2 en exp3 wordt geëvalueerd. Dus met

```
z = (a > b) ? a : b; /* z = max(a,b) */
```

wordt z het maximum van a en b.

Met de conditionele expressie is het mogelijk zeer compacte code te schrijven. Bijvoorbeeld, de onderstaande loop print N elementen van een array, 4 op een regel, elke kolom wordt gescheiden door een spatie en elke regel wordt afgesloten door een newline.

```
for (i = 0; i <N; i++)
   printf("%5d%c",a[i], (i%4==3 ||
   i ==N-1) ? '\n' : ' ');
```

Een newline wordt na elk vierde getal afgedrukt, alsmede na het N-de (laatste). Alle andere getallen worden door een spatie gevolgd.

8/4.3.12 Precedence en de volgorde van evaluatie

In de onderstaande tabel staat de precedence van alle operatoren en de volgorde waarin ze zelf worden geëvalueerd, tevens staan er ook al enkele operatoren in die nog niet behandeld zijn. Operatoren op dezelfde regel hebben dezelfde precedence, van boven naar beneden neemt de precedence af. Dus bijvoorbeeld, *, / en % hebben dezelfde precedence, die weer hoger is dan de precedence van + en –.

4.3 Typen, operatoren en expressies

OPERATOR	EVALUATIE
( ) [ ] ->.	links naar rechts
! ~ ++ -- - (type) * & sizeof	rechts naar links
* / %	links naar rechts
+ -	links naar rechts
<< >>	links naar rechts
< <= > >=	links naar rechts
== !=	links naar rechts
&	links naar rechts
↑	links naar rechts
|	links naar rechts
&&	links naar rechts
||	links naar rechts
?:	rechts naar links
= += -= etc.	rechts naar links
,	links naar rechts

De operators -> en . worden gebruikt om velden van structures aan te duiden, dit zal samen met sizeof (de grootte van een objekt) in paragraaf 7 uiteen worden gezet. Paragraaf 6 zal de * (indirectie) en & (adres van) behandelen.

8/4.4

Control flow

De control flow-statements of programma besturingsstatements van een taal specificeren in welke volgorde de statements worden uitgevoerd. De normale control flow-statements zijn al eerder in voorbeelden naar voren gekomen, de complete verzameling van deze statements staat in deze paragraaf beschreven, tevens worden de reeds bekende hier meer uitgediept.

8/4.4.1 Statements en blokken

Een expressie zoals x = 0 of i++ of printf(. . .) wordt een statement als het wordt gevolgd door een semicolon, zoals

```
x = 0;
i++;
printf( . . . );
```

De braces { en } worden gebruikt om statements te groeperen. Dit wordt dan een compound statement of een blok genoemd, in het gebruik is zo'n blok exact hetzelfde als een enkelvoudig statement. De braces die de body van een functie bevatten zijn hier een voorbeeld van, ze kunnen ook de control flow statements omvatten. Er volgt nooit een semicolon na de rechter brace van een blok.

8/4.4.2 If-Else

Het if-else statement wordt gebruikt om beslissingen te maken. Formeel is de syntax

```
if (expressie)
   statement-1
else
   statement-2
```

De else met statement-2 is optioneel. De expressie wordt geëvalueerd; als deze true is (de expressie levert een waarde ongelijk nul) wordt statement-1 geëxecuteerd. Als de expressie false was (expressie is nul) en als er een else deel is, dan wordt statement-2 geëxecuteerd.

Omdat een if alleen de numerieke waarde van een expressie bepaalt is het mogelijk om bepaalde afkortingen te maken. De meest gebruikelijke is

```
if (expressie)
```

in plaats van

```
if (expressie !=0)
```

Soms is dit heel natuurlijk en duidelijk, maar het kan in andere gevallen heel cryptisch zijn.

Omdat het else deel optioneel is, kan er onduidelijkheid over bepaalde constructies bestaan. Bijvoorbeeld in de volgende

constructie:

```
if (n > 0)
   if (a > b)
      z = a;
   else
      z = b;
```

Aan het inspringen kan gezien worden dat de else bij de binnenste if behoort. Maar als dit niet gewenst is, dan moeten er braces gebruikt worden om de juiste constructie te krijgen.

```
   if (n > 0) {
      if (a > b)
         z = a;
}
   else

   z = b;
```

Deze tweeslachtigheid wordt precair in de volgende situatie

```
   if (n > 0)
      for (i = 0; i <; i++)
         if (S[i] > 0) {
            printf(". . .");
            return(i);
         }
   else /* DIT IS FOUT */
      printf("error – n is nul\n");
```

Het inspringen laat duidelijk de betekenis zien, die de programmeur er aan geeft, maar de compiler 'weet' dat niet en zal de else met de binnenste if associeren. Dit soort fouten is moeilijk te vinden.

8/4.4.3 Else-if

De constructie

```
   if (expressie)
      statement
   else if (expressie)
      statement
   else if (expressie)
      statement
   else
      statement
```

komt zo vaak voor dat het waard is om hem hier te behandelen. Deze constructie is de algemene manier om meer-weg beslissingen te schrijven. De expressies worden van boven naar beneden geëvalueerd, als een expressie true is wordt het statement dat daar mee geassocieerd is geëxecuteerd en het beëindigt de hele constructie. De code van elk statement is of een enkelvoudig statement of een blok tussen braces.

Het laatste else deel wordt geëxecuteerd als geen van de expressies true is. Als er default geen expliciete actie ondernomen behoeft te worden kan het laatste else deel worden weggelaten, of het kan worden gebruikt om een 'onmogelijke' conditie op te vangen.

Ter illustratie een drie-weg beslissing in de volgende functie. Met deze functie wordt in een gesorteerde array v naar een waarde x gezocht. De elementen van v moeten in oplopende volgorde staan. De functie geeft de positie van x in v af (een getal tussen 0 en n-1) of -1 als x niet in v voorkomt.

4.4 Control flow

```
zoek(x, v, n) /* zoek x in v[0]... v[n-1] */
int x, v[], n;
{
    int laag, hoog, mid;

    laag = 0;
    hoog = n - 1;
    while (laag <= hoog) {
        mid = (laag + hoog) / 2;
        if (x < v[mid])
            hoog = mid - 1;
        else if (x > v[mid])
            laag = mid + 1;
        else    /* gevonden */
            return(mid);
    }
    return(-1);
```

De fundamentele beslissing of x kleiner, groter dan wel gelijk is aan het middelste element v[mid] is normaal voor de else-if.

8/4.4.4 Switch

Het switch element is een speciale meer-weg beslissings constructie, die test welke expressie overeenkomt met een aantal constante waarden en vervolgens de corresponderende expressie(s) uitvoert. In paragraaf 2 staat een programma dat het aantal cijfers, spaties en alle andere tekens telt. Daar werd gebruik gemaakt van de if . . . else if . . . else constructie. Hier is hetzelfde programma, maar dan met de switch.

```
#include <stdio.h>
main() /* telt cijfers, blancos en andere */
{
    int c, i, spatie, blanco, cijfer[10];

    blanco = ander = 0;
    for(i = 0; i < 10 ; i++)
        cijfer[i] = 0;

    while ((c = getchar()) != EOF)
        switch(c) {
        case '0':
        case '1':
        case '2':
        case '3':
        case '4':
        case '5':
        case '6':
        case '7':
```

4.4 Control flow

```
        case '8':
        case '9':
            cijfer[c - '0']++;
            break;
        case ' ':
        case '\n':
            blanco++;
            break;
        default:
            ander++;
            break;
    }
    printf("cijfer =");
    for (i =0; i < 10; i++)
        printf(" %d", cijfer[i]);
    printf("\nblanco = %d, ander = %d\n",
        blanco, ander);
}
```

De switch evalueert de integer expressie tussen de haakjes (hier de char c) en vergelijkt deze met de waarde van alle cases. Elke case moet door een integer, character constante of een constante expressie worden gelabeld. Als een case overeenkomt met de waarde van de expressie dan start de executie van de statements die corresponderen met die case. De case die met default is gelabeld wordt geëxecuteerd als aan geen van de cases wordt voldaan. De default is optioneel; als dan aan geen van de cases wordt voldaan wordt niets geëxecuteerd. De cases kunnen in willekeurige volgorde staan. Alle cases moeten verschillen.

Het break-statement zorgt voor het direct verlaten van de switch. Dit is noodzakelijk omdat de cases slechts als label fungeren en als de code van een case uitgevoerd is, zal de executie door gaan met die van de volgende case, tenzij er een speciale actie worden ondernomen om te ontsnappen. Break en return zijn de normale manieren om de switch te verlaten. Een break kan tevens gebruikt worden om een while loop direct te verlaten, maar hier over later meer.

Door cases heen vallen heeft ook zijn positieve kant, verschillende cases kunnen nu dezelfde actie ondernemen, zonder deze elke keer uit te schrijven. Bijvoorbeeld met de spatie en de newline.

8/4.4.5
Loops – while en for

De while en de for loops zijn al bekend. In

```
while (expressie)
    statement
```

wordt de expressie geëvalueerd. Als

4.4 Control flow

deze niet nul is wordt het statement geëxecuteerd en wordt de expressie gereëvalueerd. Deze cyclus herhaalt zich totdat de expressie nul wordt, dan gaat de executie verder na het statement dat na de loop komt.

Het for statement

```
for (expr1; expr2; expr3)
   statement
```

komt overeen met

```
expr1;
while (expr2) {
   statement;
   expr3;
}
```

Grammaticaal gezien zijn de componenten van het for-statement expressies. Echter meestal zijn expr1 en expr3 assignments of functie-calls en is expr2 een relationele expressievergelijking. Elk van de drie delen mag weggelaten worden, alhoewel er puntkomma's moeten blijven staan. Als expr1 en expr3 weg worden gelaten wordt de for een while. Als ook de test, expr2, wordt weggelaten, krijgt men

```
for (;;) {
   ...
}
```

wat een oneindige loop is, kan deze alleen doorbroken worden door een break of een return.

Als voorbeeld volgt hier een andere versie van atoi (converteer een string naar zijn numerieke equivalent). Deze is algemener: ze houdt rekening met voorafgaande spaties en een optioneel + of - teken.

De structuur van het programma ziet er als volgt uit:

sla alle spaties over, als die er zijn
pak het teken, als dat er is
pak het integer deel, converteer het

Elke stap doet zijn eigen werk en laat de rest voor de volgende stap onverstoord achter. Het proces stopt als er een teken wordt gevonden dat geen deel van een getal kan zijn.

```
atoi(s) /* converteer s naar integer */
char s[];
{
    int i, n, sign;
```

4.4 Control flow

De voordelen van loops komen nog meer naar voren bij het gebruik van geneste loops. De volgende functie sorteert een integer array volgens het Shell sort principe. Het idee achter de Shell sort is dat in elke fase, ver uit elkaar gelegen elementen worden vergeleken, in plaats van buren van elkaar, zoals dat in simpeler sorteer programma's wordt gedaan. Deze methode heeft tot gevolg dat grote wanorde snel wordt geëlimineerd, zodat latere fases minder te doen hebben. Het interval tussen de elementen die vergeleken worden wordt langzamerhand tot één verminderd en op dat punt gaat de Shell sort de buren van een element omdraaien.

```
shell(v, n)  /* sorteer v[0]...v[n-1] */
int v[], n;
{
    int gap, i, j, temp;

    for (gap = n/2; gap > 0; gap /= 2)
        for (i = gap; i < n; i++)
            for (j = i-gap; j >= 0 && v[j] > v[j+gap]; j -= gap) {
                temp = v[j];
                v[j] = v[j+gap];
                v[j+gap] = temp;
            }
}
```

Er zijn drie geneste loops. De buitenste loop bepaalt de afstand tussen de onderlinge elementen, elke keer wordt die afstand een factor twee kleiner, totdat deze nul wordt. De middelste loop vergelijkt elk paar van elementen die op de afstand gap van elkaar liggen. De binnenste loop draait de paren om die niet in de goede volgorde staan.

De laatste C operator is de komma, ",", die het meest in het for statement is te vinden. Een aantal expressies wordt gescheiden door een komma, ze worden van links naar rechts geëvalueerd, het type en waarde van het resultaat zijn het type en de waarde van het rechter operand. Dus in een for statement is het mogelijk om meerdere expressies in de verschillende delen te plaatsen, bijvoorbeeld om twee indices parallel te laten verlopen. Dit wordt door de functie reverse geïllustreerd. Deze functie draait alle tekens uit de string om.

```
reverse(s)   /* draai string s om */
char s[];
{
    int c, i, j;

    for (i =0, j = strlen(s) - 1;
                    i < j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}
```

4.4 Control flow

8/4.46
Loops do-while

De while- en de for-loops testen de beëindigings-conditie aan het begin van de loop, in plaats van aan het einde ervan. De derde loop in C, de do-while, test aan het einde van de loop, nadat de body van de loop geëxecuteerd is. De body van de loop wordt dus zeker één maal geëxecuteerd. De syntax is

```
do
   statement
while (expressie);
```

Het statement wordt geëxecuteerd, dan wordt de expressie geëvalueerd. Als deze waar is wordt het statement weer geëxecuteerd, enzovoorts. Als de expressie niet waar wordt, dan stopt de loop.

De do-while loop wordt minder vaak gebruikt dan de while- en de for-loops, maar desalniettemin is het soms een waardevolle constructie, zoals bijvoorbeeld in de functie atoi. Deze functie converteert een nummer naar een characterstring (het omgekeerde van atoi). De functie maakt eerst een string in omgekeerde volgorde, daarna wordt deze string omgedraaid.

```
itoa(n, s) /* converteer n naar karakterstring s */
char s[];
int n;
{
    int i, teken;

    if ((teken = n) < 0) /* bepaal teken */
        n = -n;          /* maak n positief */
    i = 0;
    do {    /* genereer cijfers in omgekeerde volgorde */
        s[i++] = n % 10 + '0'; /* bepaal volgend getal */
    } while ((n /= 10) > 0);  /* verwijder het         */
    if (teken < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
```

De do-while is hier noodzakelijk, of ten minste handig, omdat er zeker één teken in de array's moet staan. Er zijn braces om de body van de loop geplaatst, ook al is dit niet direkt noodzakelijk. De body bestaat namelijk slechts uit één statement. Echter nu begaat de haastige lezer niet de fout om te denken dat het while-gedeelte het begin van een while loop is.

8/4.4.7
Break

Het kan soms handig zijn om een loop

te beëindigen op een andere plaats dan aan het begin of aan het einde. Het break statement zorgt voor het eerder verlaten van de for-, while-, do- en switch-loops. Een break zorgt ervoor dat de binnenste loop direkt verlaten wordt.

8/4.4.8 Continue

Het continue-statement lijkt veel op de break, het zorgt ervoor dat de volgende iteratie van de binnenste loop (for, while, do) begint. In geval van de while en de do betekent dit dat het testgedeelte direkt wordt geëvauleerd. In geval van de for wordt verder gegaan bij de reïnitialisatiestap (de continue werkt slechts op loops en niet op de switch. Een continue binnen een switch binnen een loop zorgt voor de volgende loop iteratie.

Een voorbeeld, dit onderdeel van een programma werkt alleen op de positieve elementen van de array; de negatieve elementen worden overgeslagen.

```
for (i = 0; i < N; i++) {
   if (a[i] < 0) /* sla negatieve
   elementen over */
      continue;
   . . . . /* verwerk de positieve
   elementen */
}
```

Het continue-statement wordt vaak gebruikt als de loop die volgt gecompliceerd is.

8/4.4.9 Goto's en labels

C ondersteunt ook het altijd verwarrende goto-statement, alsmede de labels om naar toe te springen. Eigenlijk is het gebruik van de goto nooit noodzakelijk en in de praktijk is het bijna altijd mogelijk om code zonder de goto te schrijven. Desalniettemin zijn er bepaalde situaties waarin de goto een plaats kan vinden. De meest gebruikelijke is om een proces van diep geneste loops direkt te verlaten. Hier kan de break niet helpen want deze verlaat slechts de binnenste loop.

```
for(. . .)
   for( . . .) {
      . . .
      if (ramp)
         goto fout;
   }
. . .

fout:

   ruim de rommel op
```

Op deze manier is het gebruik van de goto handig en dit is zeker het geval als de fouten op verschillende plaatsen op kunnen treden. Een label heeft dezelfde vorm als de naam van een variabele, het wordt gevolgd door een dubbele punt, ":". Het kan aan elk statement vooraf gaan in DEZELFDE functie als de goto naar dat label. Het is niet mogelijk om naar een label te springen in een andere functie al ondersteunen de setjmp( ) en de longjmp( ) standaard bibliotheek functies wel zo'n gedrag.

8/4.5

Functies en programmastructuur

Functies worden gebruikt om grote programma's in kleinere stukken stukken op te splitsen en maken het mensen mogelijk om gebruik te maken van het werk dat al door anderen is gedaan. Met geschikte functies is het mogelijk om bepaalde details weg te stoppen, zodat het programma dat deze functies gebruikt niets van deze details hoeft te weten. Dit maakt het geheel duidelijker en ook eenvoudiger om te veranderen.

C is ontworpen om functies efficiënt en eenvoudig in het gebruik te maken. Een C-programma bestaat in het algemeen uit vele kleine functies in plaats van enkele grote. Een programma mag over verschillende sourcefiles verdeeld worden, deze files mogen afzonderlijk gecomplileerd worden, later kunnen ze dan samen met reeds bestaande gecompileerde en/of bibliotheekfuncties gelinkt worden en als een geheel gedraaid worden.

De meesten onder ons zijn reeds bekend met de bibliotheekfuncties voor invoer en uitvoer, zoals getchar() en putchar(). In dit hoofdstuk zult u meer zien over het zelf schrijven van programma's.

8/4.5.1
De basis

Elke functie heeft de vorm:

```
naam(parameter lijst, als deze er is)
declaratie van de argumenten, als ze er zijn
{
    delaraties en statements,
    als ze er zijn
}
```

Uit de hierboven beschreven vorm blijkt dat verscheidene delen afwezig mogen zijn. De minimale functie is:

```
niets () { }
```

wat ook niets doet. De functienaam mag vooraf worden gegaan door het type wat de functie afgeeft (retourneert), in het geval dat dit iets anders is dan een integer.

Een programma is slechts een verzameling van verschillende functiedefinities. De communicatie tussen de functies gaat via de argumenten en de waarden die de functies retourneren, ook kan de communicatie via externe variabelen gaan. De functies mogen in willekeurige volgorde in de sourcefile verschijnen en het programma kan over verschillende files verdeeld worden, zolang er geen functies gesplitst worden.

Het return-statement zorgt ervoor dat een waarde van een functie geretourneerd wordt aan de caller. Elke expressie kan

4.5 Functies en structuur

als argument van return worden mee gegeven:

return(expressie)

De caller (die de functie callt) behoeft niets met de geretourneerde waarde te doen. Ook behoeft er geen expressie na het return statement te volgen, in dat geval wordt er geen waarde geretourneerd aan de caller. De executie gaat ook verder bij de caller zonder geretourneerde waarde als de functie aan het einde van de functie de laatste rechter brace tegenkomt. Het is niet foutief, maar het duidt op moeilijkheden, als een functie op een plaats een waarde retourneert en op een andere plaats niets retourneert. In elk geval is de "waarde" die een functie retourneert ongedefineerd als niet expliciet een return-statement met expressie wordt geëxcuteerd.

8/4.5.2
Return waarden

Tot nu toe is nog in geen van de programma's een functie gedeclareerd. Dit komt omdat een functie automatisch en impliciet gedeclareerd wordt zodra deze gebruikt wordt in een expressie of in een statement, zoals:

while (functie(arg1, arg2, 1) › j)

Als een naam, die nog niet eerder is gedeclareerd, in een expressie voorkomt en wordt gevolgd door een haak openen, wordt deze beschouwd als een functienaam. Tevens wordt dan aangenomen dat het een functie is die een integer-waarde retourneert. Omdat een char een int wordt in een expressie is er geen reden om een functie te declareren die een char retourneert. Deze veronderstellingen dekken het grootste gedeelte van alle gevallen en tevens alle hierboven beschreven functies.

Maar wat gebeurt er als een functie een ander type moet retourneren? Veel numerieke functies zoals sqrt, sin en cos retourneren floats, andere functies kunnen andere waardes retourneren. Dit wordt geïllustreerd aan de hand van de functie wortel, deze functie bepaalt de wortel van een getal.

Ten eerste dient wortel het type van de waarde dat ze retourneert te declareren, omdat het geen int is. De typenaam gaat aan de functienaam vooraf.

```
#include <stdio.h>
float verbeter();
float gemiddelde();
float fabs();

float wortel(x)
float x;
```

4.5 Functies en structuur

```
{
    float schatting = 1.0;

    while (!goegenoeg(schatting,x)) {
        schatting = verbeter(schatting,x);
    }
    return (schatting);
}

int goegenoeg( a, b)
float a,b;
{
    return ((fabs(a * a - b)) < (b / 1.0e8));
}

float verbeter(s,x)
float s,x;
{
    return(gemiddelde(s,x/s));
}

float gemiddelde(a,b)
float a,b;
{
    return ((a + b) / 2.0);
}

main()
{
    float x;

    scanf("%f",&x);
    while(x > 0){
        printf("%f %f\n", x, wortel(x));
        scanf("%f",&x);
    }
}
```

Ten tweede moet de routine die wortel aan roept, hier main, weten dat wortel geen integer-waarde retourneert. In dit voorbeeld weet main dit omdat de functie wortel aan main vooraf gaat. Maar wortel maakt zelf gebruik van verbeter, wat ook een float-retournerende functie is. Om dit nu aan wortel bekend te maken wordt verbeter als float-functie gedeclareerd:

float verbeter();

Voor gemiddelde geldt hetzelfde als voor goedgenoeg, want goedgenoeg roept verbeter aan en moet ook weten dat verbeter een float-retournerende functie is.

De wortel wordt in deze functie met behulp van de methode van Newton be-

paald. Er wordt net zolang een schatting van de wortel verbeterd totdat de maximale preciesie van de Commodore 64 is bereikt. De schatting wordt verbeterd door het gemiddelde van de schatting en het gevraagde gedeeld door de schatting te nemen. De wortel ligt ten slotte tussen beide waarden in.

8/4.5.3 Functie argumenten

Eerder is al beschreven dat functie-argumenten "by value" worden doorgegeven, dit houdt in dat een functie een eigen, tijdelijke kopie van elk argument krijgt en niet het adres van dat argument. Dit betekent dat een functie niet de oorspronkelijke waarde van dat argument kan aantasten. Binnen de functie is elk argument eigenlijk een lokale variabele die geïnitialiseerd wordt met de waarde waarmee de functie aangeroepen wordt.

Als de naam van een array als argument van een functie wordt gebruikt, dan wordt het adres van het begin van de array doorgegeven, de elementen worden niet gekopieerd. De array is dus een uitzondering op de bovenstaande regel. De functie kan array-elementen veranderen. Dit komt omdat arrays "by reference" worden doorgegeven. In het volgende hoofdstuk wordt beschreven hoe, met behulp van pointers, argumenten die geen array zijn, toch veranderd kunnen worden.

Het is trouwens niet goed mogelijk om functies te schrijven, die een variabel aantal argumenten accepteren, omdat er geen goede methode is om te bepalen hoeveel argumenten tijdens de functie aanroep werden doorgegeven. Het is dus bijvoorbeeld niet mogelijk een functie te schrijven die het maximum van een willekeurig aantal argumenten kan bepalen, zoals dit wel met de MAX functie van Fortran of een max-functie uit LISP kan.

Alhoewel een variabel aantal argumenten niet goed mogelijk is, is het echter niet ónmogelijk zolang, de aangeroepen functie maar geen argument gebruikt wat niet bestaat en als de typen maar overeenstemmen. printf is de meest gebruikte C functie met een variabel aantal argumenten, het gebruikt informatie van het eerste argument om te bepalen hoeveel andere argumenten aanwezig zijn en wat voor type ze hebben. printf gaat hevig in de fout als de caller (de functie die aanroept) niet genoeg argumenten meegeeft of de typen niet zijn wat het eerste argument aangeeft. Hierbij komt nog dat de functie niet zomaar overgebracht kan worden naar een andere computer, de functie zal dan aangepast moeten worden.

8/4.5.4 Externe variabelen

Een C-programma bestaat uit een verzameling externe objecten, die óf variabelen óf functies zijn. Het voorvoegsel extern wordt gebruikt in tegenstelling tot intern, wat de argumenten en de automatische variabelen binnen een functie beschrijft. Externe variabelen worden buiten functies gedefinieerd en zijn beschikbaar voor vele functies. Functies zijn altijd extern, C laat tenslotte niet toe dat functies binnen andere functies worden gedefinieerd, in tegenstelling tot bijvoorbeeld Pascal. Automatisch zijn externe variabelen "global", zodat alle referenties naar zo'n variabele door dezelfde naam naar hetzelfde object refereren.

Omdat externe variabelen globaal toegankelijk zijn, bieden zij een alternatief

4.5 Functies en structuur

voor functie-argumenten en geretourneerde waarden om voor de communicatie tussen functies te zorgen. Elke functie kan externe variabelen gebruiken, lezen en schrijven, mits de naam van die variabele ergens gedeclareerd is.

Als een groot aantal variabelen door verschillende functies moet worden gedeeld is het gebruik van externe variabelen vaak handiger en efficiënter dan het gebruik van lange argumentlijsten. Maar om redenen uiteengezet in hoofdstuk 2 moet dit met enige zorg gedaan worden, omdat het een slecht effect op de programmastructuur kan hebben en tot programma's kan leiden met veel data-verbindingen tussen de functies.

Een andere reden om externe variabelen te gebruiken is de levensduur. Automatische variabelen bestaan slechts wanneer een functie is aangeroepen en ze verdwijnen wanneer deze verlaten wordt. Externe variabelen zijn daarentegen permanent. Zij worden niet gecreëerd waarna ze na afloop weer verdwijnen, zij blijven bestaan en behouden hun waarde tussen de ene functiecall en de andere. Dus als twee functies data moeten delen zonder dat de een de ander aanroept, is het heel geschikt om de gemeenschappelijke data op te slaan in externe variabelen in plaats van deze data via de argumenten in en uit te laten gaan.

Aan de hand van het volgende voorbeeld zal het een en ander verduidelijkt worden. Het probleem is om een calculatorprogramma te schrijven. De volgende operaties kunnen ermee uitgevoerd worden: +, -, *, / en = (druk het antwoord af). Om het programma wat eenvoudiger te houden zal de calculator de omgekeerd Poolse notatie gebruiken in plaats van infix. Omgekeerd Poolse notatie wordt wel gebruikt op HP rekenmachines, ook is de taal Forth hier op gebaseerd. In de omgekeerd Poolse notatie wordt elke operator na zijn operanden geplaatst, een infix notatie zoals:

(1 - 2) * (4 + 5) =

wordt dan als volgt geschreven:

1 2 - 4 5 + * =

Haakjes zijn hier dan niet nodig. Elke operand wordt op een stack (stapel) gezet (push) en wanneer een operator arriveert wordt het benodigde aantal operanden (twee voor de gewone operatoren) van stack gehaald (pop) en de operatie wordt op de operanden uitgevoerd. Het resultaat wordt daarna op de stack terug gezet. In het bovenstaande voorbeeld worden 1 en 2 op stack gepusht en vervangen door hun verschil, -1.
Vervolgens worden 4 en 5 gepusht en die worden vervangen door hun som, 9. Het produkt van -1 en 9, -9 vervangt hen weer op de stack. Met de = operator wordt het bovenste element van de stack bekeken, zonder het er vanaf te halen.

De push en pop operaties zijn simpel, maar om ze ook foutbestendig te maken en op die fouten netjes te laten reageren zijn ze groot genoeg geworden om ze in functies onder te brengen. Ook moet er een aparte functie zijn om de volgende input — operator of operand — op te halen. Dus dan wordt de structuur van het programma als volgt:

while (volgende operator of operand is niet einde invoer)

4.5 Functies en structuur

```
if (nummer)
      push het op stack
else if (operator)
      pop de operands
      doe de operatie
      push het resultaat
else
      foutmelding
```

```
#include <stdio.h>

#define MAXOP     20   /* max grote van operand of operator */
#define NUMMER    '0'  /* signaal dat een nummer is gevonden */
#define TEGROOT   '9'  /* signaal dat de string is te groot */


main()      /* omgekeerde Polisch desk calculator */
{
    int type;
    char s[MAXOP];
    float op2, atof(), pop(), push();

    while ((type = getop(s, MAXOP)) != EOF)
        switch (type) {

        case NUMMER:
            push(atof(s));
            break;
        case '+':
            push(pop() + pop());
            break;
        case '*':
            push(pop() * pop());
            break;
        case '-':
            op2 = pop();
            push(pop() - op2);
            break;
        case '/':
            op2 = pop();
            if (op2 != 0.0)
                push(pop() / op2);
            else
                printf("Delen door nul is niet toegestaan\n");
            break;
```

4.5 Functies en structuur

```
        case '=':
            printf("    %f\n",push(pop()));
            break;
        case 'c':
            clear();
            break;
        case TEGROOT:
            printf("%.20s ... is te groot\n",s);
            break;
        default:
            printf("onbekend commando %c\n", type);
            break;
        }
}

        clear();
        return(0);
    }
}

clear()     /* leeg de stack */
{
    sp = 0;
}
#define MAXDIEPTE 100 /* stack diepte */

int sp = 0;     /* stack pointer */
float val[MAXDIEPTE]; /* stack */

float push(f)   /* push f op de stack */
float f;
{
    if (sp < MAXDIEPTE)
        return(val[sp++] = f);
    else {
        printf("Error: stack is vol\n");
        clear();
        return(0);
    }
}

float pop()       /* pop de top van de stack */
{
    if (sp > 0)
        return(val[--sp]);
    else {
        printf("Error: stack leeg\n");
```

Met het c-commando wordt de stack vrijgemaakt, dit gebeurt met behulp van de clear functie die ook door push en pop wordt gebruikt in geval van een fout. Op getop wordt straks terug gekomen.

Zoals al eerder is besproken, is een variabele extern zodra deze buiten de body van een functie is gedefinieerd. Dus de stack en de stackpointer, die door push, pop en clear gedeeld moeten worden, zijn buiten deze drie functies gedefinieerd. Maar main zelf gebruikt de stack en de stackpointer niet, ze zijn voor main verborgen. Dus de = operator moet

```
push (pop());
```

gebruiken om de top van de stack te bekijken zonder deze te veranderen.

N.B. bij de + en de * operatoren is het niet van belang in welke volgorde de operanden van stack gepopt worden, maar voor de - en de / operatoren is dit wel van belang, er moet onderscheid gemaakt worden tussen de linker en de rechter operator.

8/4.5.5
Werkruimte van variabelen en functies

De functies en externe variabelen die samen een C-programma vormen, behoeven niet allemaal tegelijk gecompileerd te worden, de source van het programma mag over verschillende files verdeeld worden en routines die al eerder gecompileerd waren mogen gelinkt worden. Hieruit resulteren twee belangrijke vragen:

1. Hoe moeten declaraties geschreven worden, zodat variabelen juist gedeclareerd worden tijdens de compilatie?

2. Hoe worden de declaraties opgebouwd, zodanig dat de losse stukken goed met elkaar verbonden worden tijdens het linken?

De werkruimte of scope van een naam is het gedeelte van het programma waarvoor de naam gedefinieerd is, dus waar deze naam in te gebruiken is. Van een automatische variabele die aan het begin van een functie gedefinieerd is, is de scope de functie waarbinnen de naam is gedeclareerd en variabelen met dezelfde naam in andere functies staan hier los van. Het zelfde geldt voor de argumenten van een functie.
De scope van een externe variabele begint op het punt waar deze gedeclareerd wordt tot aan het einde van de file. Bijvoorbeeld, als val, sp, push, pop en clear in een file worden gedefinieerd, in de volgorde hier boven beschreven,

```
int sp = 0;
float val[MAXGROOTTE];

float push(f) {.....}

float pop() {.....}

clear() {.....}
```

dan mogen de variabelen val en sp in push, pop en clear gebruikt worden, door ze gewoon aan te roepen. Er zijn geen verdere declaraties nodig.

Aan de andere kant, als een externe variabele wordt gebruikt voordat hij gedefinieerd is of dat hij in een andere file wordt gebruikt dan waar hij in gedefinieerd is, dan is de extern declaratie verplicht.

Het is belangrijk het verschil te zien tus-

4.5 Functies en structuur

sen de declaratie van een externe variabele en zijn definitie. Een declaratie duidt het type aan, een definitie doet dit ook maar veroorzaakt dat er ruimte voor de variabele wordt gereserveerd. Als de regels

```
int sp;
float val[MAXGROOTTE];
```

buiten een functie staan, dan definiëren ze de externe variabelen sp en val, zorgen ervoor dat er ruimte wordt gereserveerd en zorgen voor de declaratie tot aan het einde van de file. Aan de andere kant, de regels

```
extern int sp;
extern float val[];
```

declareren voor de rest van de file dat sp een int en dat val een float-array is (waarvan de grootte ergens anders is bepaald). Ze creëren dus geen variabelen en reserveren er geen ruimte voor.

Er mag slechts een definitie van een externe variabele zijn in alle files die een programma vormen, ander files mogen extern declaraties gebruiken om ze te bereiken. De initialisatie van externe variabelen kan alleen gebeuren bij de definitie. De grootte van arrays moet gespecificeerd worden bij de definitie, maar is niet nodig bij een externe declaratie.

Alhoewel het geen gebruikelijke manier is voor zo'n klein programma, kunnen val en sp in één file gedefinieerd en geïnitialiseerd worden en de functies push, pop en clear in een andere file gedefinieerd worden. Dit gaat er dan als volgt uit zien:

In file 1

```
int sp = 0;   /* stack pointer */
float val[MAXGROOTTE] /* stack
*/
```

In file 2

```
extern int sp;
extern float val[];

float push(f) {.....}

float pop() {.....}

clear() {.....}
```

Omdat de extern declaraties in file 2 aan de drie functies voorafgaan, gelden ze voor alle drie, dus deze declaraties zijn voldoende voor de gehele file 2.

Hieronder zal de getop functie, die in de vorige paragraaf al is gebruikt, worden besproken. De functie haalt de volgende operator of operand. De primaire taak is eenvoudig: sla alle spaties en newlines over. Als het volgende teken geen cijfer of een punt (decimale punt) is retourneer dit dan, het is een commando. Anders moet een string van cijfers verzameld worden en moet NUMMER geretourneerd worden, het signaal dat een nummer is verzameld.

De routine wordt ingewikkelder, omdat de situatie waarbij de nummers te groot zijn fatsoenlijk wordt afgehandeld. De getop functie leest getallen (met misschien een tussenliggende punt) totdat hij ze niet meer ziet, maar slaat alleen dat gedeelte op wat past. Als het getal past wordt NUMMER en een string van getallen geretourneerd. Maar als het nummer te groot was dan slaat getop de rest van de regel over zodat de gebruiker slechts de regel behoeft over te typen vanaf het punt van de fout, de functie retourneert TEGROOT als een fout signaal.

4.5 Functies en structuur

```
#include <stdio.h>
#define NUMMER  '0'                   /* signaal nummer is gevonden */
#define TEGROOT '9'                   /* signaal nummer is te groot */

getop(s, lim)              /* lever volgende operator of operand */
char s[];
int lim;
{
   int i, c;

   while ((c = getch()) == ' ' || c == '\n')
      ;                            /* sla spaties en newlines over */
   if (c != '.' && (c < '0' || c > '9'))
      return(c);                         /* geen nummer dus commando */
   s[0] = c;
   for (i = 1; (c = getchar()) >= '0' && c<= '9'; i++)
      if (i < lim)
         s[i] = c;                          /* stop cijfer in array */
   if (c == '.') {                          /* verzamel fractie     */
      if (i < lim)
         s[i] = c;
      for (i++; (c = getchar()) >= '0' && c <= '9'; i++)
         if (i < lim)
            s[i] = c;
   }
   if (i < lim) {                                /* nummer is goed */
      ungetch(c);
      s[i] = '\0';
      return(NUMMER);
   } else {/* het is te groot, sla de rest van de regel over */
     while (c != '\n' && c != EOF)
        c = getchar();
     s[lim-1] = '\0';
     return(TEGROOT);
   }
}
```

4.5 Functies en structuur

Wat doen getch en ungetch? Vaak is het zo dan een programma input leest en nog niet kan bepalen of het genoeg gelezen heeft todat het werkelijk te veel heeft gelezen. Een voorbeeld hiervan is het verzamelen van tekens die samen een nummer vormen, namelijk zolang er nog geen niet-cijfer is gezien is het nummer nog niet compleet. Maar zodra het programma een teken te ver leest weet het dat het nummer reeds binnen is maar kan dan nog niets met het volgende teken doen.

Dit probleem zou zijn opgelost als er een "on-lees" commando zou bestaan. Elke keer als het programma een letter te veel leest, dan zou het teken terug in de input kunnen stoppen, zodat de rest van het programma niet in de gaten heeft dat het ooit gelezen is. Helaas is er geen standaard functie die dit doet, zodat dit gesimuleerd moet worden door twee samenwerkende functies. getch levert het volgende teken af en ungetch stopt het teken terug in de invoer, zodanig dat bij de volgende aanroepen van getch dit teken weer wordt gelezen.

Het mechanisme hiervoor is vrij simpel. ungetch stopt de teruggestopte tekens in een gezamelijke buffer, een char array. getch leest van die buffer als er iets in zit en als deze leeg is dan callt ze getchar. Er moet ook een index zijn die de positie aangeeft van het laatste teken in de buffer.

Omdat de buffer en de index door getch en ungetch gedeeld moeten worden en hun waarden behouden moeten worden tussen de functiecalls door, moeten ze beiden extern aan de routines zijn. Dit ziet er dan als volgt uit:

```
#define BUFFERGROOTTE  100

char buf[BUFFERGROOTTE]; /* buffer voor de ungetch */
int  bufp = 0;           /* volgende vrije positie */

getch() /* pak een teken, mogelijk terug gestopt */
{
   return((bufp > 0) ? buf[--bufp] : getchar());
}

ungetch(c)        /* stop teken terug in de invoer */
int c;
{
   if (bufp > BUFFERGROOTTE)
      printf("Ungetch: te veel tekens\n");
   else
      buf[bufp++] = c;
}
```

Er wordt gebruik gemaakt van een array zodat er meer dan een teken terug gestopt kan worden.

8/4.5.6
Statische variabelen

De statische variabele is de derde methode van data opslag, eerder in dit hoofdstuk zijn de externe en automatische variabele al besproken.

Statische variabelen mogen extern dan wel intern zijn. Interne statische variabelen zijn lokaal aan een bepaalde functie zoals automatische variabelen dit zijn, maar in tegenstelling tot de automatische variabelen blijven ze bestaan in plaats van te verdwijnen en weer gecreëerd te worden iedere keer als een functie wordt geactiveerd. Dit betekent dat interne statische variabelen voor de eigen, permanente opslag van een functie zorgen.

Een externe statische variabele is bekend in de rest van de file waarin deze gedeclareerd is, maar niet in een andere file. De externe statische variabelen vormen dus een mechanisme om variabelen te verbergen. Dus de variabele-namen buf en bufp in de getch-ungetch combinatie, die extern moeten zijn om gedeeld te kunnen worden, kunnen onzichtbaar gemaakt worden voor de gebruikers van getch en ungetch, zodat het onmogelijk is dat er conflicten kunnen ontstaan met andere variabelen die buf of bufp heten. De variabelen zijn voor de "buitenwereld" onzichtbaar als ze in een file als volgt worden gecompileerd:

```
static char [BUFFERGROOTTE]
static int bufp = 0;

getch() {.....}

ungetch() {.....}
```

Statische variabelen worden gespecificeerd met het woord static voorafgaand aan de normale declaratie. De variabele is extern als hij buiten elke functie is gedefinieerd en is intern als hij binnen een functie wordt gedefinieerd.

In C staat 'static' niet alleen voor permanente data opslag, maar ook tot op een bepaalde hoogte voor 'privacy'. Interne statische objecten zijn alleen bekend binnen een functie, externe statische objecten (dit kunnen zowel variabelen als functies zijn) zijn alleen bekend binnen de file waarin ze verschijnen en kunnen geen problemen opleveren met variabelen of functies met dezelfde naam in andere files.

Externe statische objecten vormen een manier om data-objecten en interne routines, die die objecten manipuleren, te verbergen. Bijvoorbeeld getch en ungetch vormen een "module" voor tekeninvoer en -terugstoppen: buf en bufp moeten statisch zijn zodat ze ontoegankelijk zijn van buiten. Op dezelfde manier vormen push, pop en clear een module voor stack operaties, val en sp moeten ook static/extern (???) zijn om ontoegankelijk van buiten af te zijn.

8/4.5.7
Registervariabelen

De vierde en laatste methode van dataopslag heet register. Op de Commodore 64 worden alle registerdeclaraties genegeerd.
De C-compiler op elke machine bepaalt hoeveel registervariabelen aan registers van de onderliggende processor worden toegekend en op deze machine wordt er geen één toegekend. Als registervariabe-

len wél in machineregisters worden gestopt dan wordt aanbevolen om variabelen als zodanig te declareren als deze zeer vaak gebruikt wordt. In dat geval resulteert het gebruik van registervariabelen in snellere programma's.

Op een machine die wel iets met registervariabelen doet en waar hetzinnig kan zijn deze te gebruiken ziet een declaratie er als volgt uit:

```
register int x;
register char c;
```

Alleen automatische variabelen en parameters van een functie mogen als register gedeclareerd worden. Dat ziet er dan als volgt uit:

```
f(c,n)
registerintc,n;
{
    registerint i;
    .....

}
```

N.B. Ook al kent de Commodore 64 de registervariabele om C-programma's te versnellen dan niet, toch kunnen programma's versneld worden door de volgorde van de variabele-declaraties slim te kiesen. De eerste integers, characters en pointers die binnen een functie worden gedeclareerd worden namelijk in de zero page geplaatst, terwijl de rest op de stack in het gewone, langzamere geheugen wordt geplaatst. De variabelen die dus vaak gebruikt worden kunnen dus het best vooraan in de declaratie lijst staan.

8/4.5.8 Blok structuur

C is geen blok-gesturctureerde taal zoals Pascal dit is, het is in C niet mogelijk om functies binnen andere functies te definiëren.

Aan de andere kant kunnen variabelen wél op een blok-gestructureerde manier gedefinieerd worden. Declaraties van variabelen mogen na de linker brace van elk meervoudig statement volgen en niet alleen aan het begin van een functie. Variabelen die zo gedefinieerd worden maken gelijkgenaamde variabelen uit buitenliggende blokken onzichtbaar en ze blijven bestaan tot aan de corresponderende rechter brache. Bijvoorbeeld:

```
i + + ;
if (n › 0) {
   int i;   /* definitie van een
   nieuwe i */
   for (i { 0; i ‹ n; i + +
     .....

}
```

de scope of werkruimte van de nieuwe i bestaat uit het blok behorend bij de truetak van het if statement. Deze i is niet gerelateerd met de i die boven het if statement staat.
Blok structuur is tevens van toepassing op externe variabelen. Bij de volgende declaraties

```
int x;

f()
{
   float x;
   .....
}
```

is referentie binnen de functie f naar x een referentie naar de floating point variabele, buiten f is dit een referentie naar de externe integer. Hetzelfde geldt voor parameters:

```
int z;

f(z)
float z;
{
   .....

}
```

Binnen de functie f, refereeert z naar de parameter, niet naar de externe variabele.

8/4.5.9
Initialisatie

Initialisatie is al een aantal malen genoemd, maar dit was dan altijd als onderdeel van een ander onderwerp. In deze paragraaf worden de regels samengevat, nu dat de verschillende data-opslag-mechanismen bekend zijn.

Als er niet expliciet wordt geïnitialiseerd, worden de externe en statische variabelen met nul geïnitialiseerd, automatische en registervariabelen hebben en ongedefinieerde waarde.

Eenvoudige variabelen (geen array's of structures) mogen worden geïnitialiseerd wanneer zij gedefinieerd worden, door na de naam het "=" teken met een constante expressie te laten volgen:

```
int x = 1;
char letter = 'a';
float dag = 24 * 60 * 60;
```

Externe en statische variabelen worden slechts één maal geïnitialiseerd, namelijk bij het opstarten van het programma. Automatische en register variabelen worden geïnitialiseerd elke keer wanneer een functie of een blok wordt binnengegaan.

Voor automatische en register variabelen geldt niet de restrictie, dat ze alleen met constante expressies mogen worden geïnitialiseerd. In feite mogen ze door elke expressie, zelfs een functie-call, worden geïnitialiseerd. Bijvoorbeeld de initialisatie van het binaire zoekprogramma uit sectie 4 kan als volgt worden geschreven:

```
zoek (x, v, n)
int x, v[], n;
{
   int laag = 0;
   int hoog = n - 1;
   int mid;
   .....
}
```

in plaats van

```
zoek(x, v, n)
int x, v[], n;
{
   int laag, hoog, mid;
   laag = 0;
   hoog = n - 1;
   .....
}
```

Het effect van de initialisatie is het zelfde als de assignments na de declaratie. Welke vorm gebruikt wordt is enkel en alleen een kwestie van smaak.

Automatische arrays mogen niet geïnitialiseerd worden. Externe en statische arrays mogen wel geïnitialiseerd

worden, de initialisatoren staan tussen braces en worden gescheiden door komma's. Bijvoorbeeld het teken-tel programma uit sectie 2, dat als volgt begon:

```
main() /* tel cijfers, blanco
en andere karakters */
{
   int c, i, blanco, ander;
   int cijfers[10];

   blanco = ander = 0;
   for (i = 0;i < 10; + +i)
      cijfers[i] = 0;
   .....

}
```

kan ook anders geschreven worden

```
int blanco = 0;
int ander = 0;
int cijfers[10] = { 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0};

main()  /*tel cijfers, blanco
en andere karakters */
{
   int c, i;
   .....

}
```

Hier zijn deze initialisaties eigenlijk niet nodig omdat ze toch automatisch allemaal op nul worden geïnitialiseerd, maar het is duidelijker om dit expliciet te doen. Als er minder initialisatoren zijn dan de grootte van de array specificeert, dan is de rest nul. Het is fout om te veel initialisatoren te hebben. Helaas bestaat er geen manier om herhaling van de initialisator aan te geven, noch bestaat er een methode om een element uit het midden van een array te initialiseren zonder alle voorafgaande waardes te specificeren.

Character arrays kunnen op een speciale manier worden geïnitialiseerd, een string mag worden gebruikt in plaats van de braces met komma's notatie:

```
char string[] = "text";
```

Dit is de verkorte notatie voor

```
char string[] =
{ 't', 'e', 'x', 't', '/0'};
```

Als de grootte van een array wordt weggelaten, dan wordt dit het aantal initialisatoren. In dit geval is de grootte 5.

8/4.5.10
Recursie

C-functies mogen recursief gebruikt worden, dit betekent dat functies zichzelf mogen aanroepen, direct dan wel indirect. Een traditioneel voorbeeld is dat van het afdrukken van een cijferstring. Zoals al eerder is gezegd worden de tekens in de verkeerde volgorde geproduceerd: de laagwaardige cijfers zijn eerder beschikbaar dan de hoogwaardige cijfers, maar ze moeten in de omgekeerde volgorde worden afgedrukt.

Er zijn twee methodes om dit probleem op te lossen. De eerste slaat de cijfers op in een array op het moment dat ze worden gegenereerd en print ze later in omgekeerde volgorde af, zoals in sectie 4 met de itoa functie is gedaan. De eerste versie van printd volgt deze methode.

4.5 Functies en structuur

Het alternatief is een recursieve methode, waarin printd eerst zich zelf aanroept met de voorafgaande cijfers en daarna het achterliggende cijfer print.

```
printd(n)    /* print n decimaal af */
int n;
{
    char s[10];
    int i;

    if (n < 0) {
        putchar('-');
        n = -n;
    }
    i = 0;
    do {
        s[i++] = n % 10 + '0'; /* pak volgende teken */
    } while ((n /= 10) > 0);
    while (--i >= 0)
        putchar(s[i]);
}
```

Als een functie zich recursief aanroept, krijgt de functie iedere keer een verse nieuwe set van automatische variabelen. Dus in printd(123) geldt voor de eerste printd n = 123. Het geeft 12 door aan de volgende printd en print 3 als deze returnt. Op dezelfde manier geeft de tweede printd 1 door aan de derde (die 1 print) en drukt 2 af.

Recursie is niet sneller dan de niet recursieve variant. Maar recursieve code is compacter en vaak eenvoudiger te schrijven en gemakkelijker om te begrijpen. Recursie is uitermate geschikt voor recursief gedefinieerde data-structuren zoals bomen.

```
printd(n)    /* print n decimaal af (recursief) */
int n;
{
    int i;

    if (n < 0) {
        putchar('-');
        n = -n;
    }
    if ((i = n/10) != 0)
        printd(i);
    putchar(n % 10 + '0');
}
```

8/4.5.11
De C Preprocessor

C heeft een simpele macro preprocessor. De # define mogelijkheid die al gebruikt is, is de meest gebruikte faciliteit die de preprocessor biedt, een andere mogelijkheid is om files op te nemen tijdens de compilatie.

File opname
Files kunnen ingelezen worden door een regel met een # include op te nemen. Zo'n regel ziet er als volgt uit:

```
# include "filenaam"
```

of

```
# include <filenaam>
```

voor de Commodore 64 is er geen verschil, maar op andere systemen wordt bij de eerste versie de file in de huidige directory gezocht, terwijl de file uit de tweede versie uit de standaard file inclusion bibliotheek wordt gezocht (zoals bijvoorbeeld stdio.h). Vaak komt zo'n regel voor in het begin van een source file. # include's mogen genest voorkomen.

Macro substitutie
Een definitie zoals

```
# define JA 1
```

roept een simpel macro-substitutie mechanisme aan, de naam wordt door een aantal tekens vervangen. Namen die door # define worden gedefinieerd hebben dezelfde vorm als C variabele-namen en de vervangende text is willekeurig. Normaal is de vervangende text de rest van de regel. De scope van de gedefinieerde naam is vanaf dat punt tot aan het einde van de file. Namen mogen geherdefinieerd woren en een definitie mag vorige definities gebruiken. Substituties worden niet uitgevoerd als de naam tussen quotes staat. Bijvoorbeeld als JA een gedefinieerde naam is, dan vindt er geen substitutie plaats in printf ("JA").

Omdat de werking van # define uit een macro voorbewerking bestaat en geen onderdeel van de eigenlijk compiler is, zijn er weinig grammaticale beperkingen aan wat er zoal gedefinieerd kan worden. Pascal-fans kunnen bijvoorbeeld het volgende doen:

```
# define then
# define begin{
# define end  ;}
```

en dan schrijven

```
if (I > 0) then
  begin
    a = 1;
    b = 2
  end
```

Het is ook mogelijk om macro's met argumenten te definiëren, zodat de vervangende tekst afhankelijk is van de manier waarop de macro wordt aangeroepen. Als voorbeeld een macro max:

```
# define max (A ,B) ((A) > (B) ?
(A) : (B))
```

Nu wordt de regel

```
a = max(q+2,c*d);
```

vervangen door de regel

```
a = ((q+2) > (c*d) ?
(q+2) : (c*d));
```

4.5 Functies en structuur

Dit is de maximum functie die op de regel wordt geëxpandeerd in plaats van een functiecall. Voorzichtigheid is geboden bij het gebruik van haakjes om er voor te zorgen dat de evaluatie correct wordt uitgevoerd. Neem bijvoorbeeld de macro

```
# define kwadraad(x) x * x
```

en roep deze aan als kwadraad(z + 1). Het resultaat is niet het kwadraat (z + 1 * z + 1 = 2 * z + 1)!
Niet te min zijn macro's toch heel waardevol.

8/4.6

Pointers en arrays

Een pointer is een variabele die het geheugenadres van een andere variabele bevat. Pointers worden veel in C gebruikt, gedeeltelijk omdat het soms niet mogelijk is een berekening zonder pointers uit te voeren en gedeeltelijk omdat het gebruik van pointers tot een meer compacte en efficiëntere codering leidt.

Pointers worden wel eens geassocieerd met het goto-statement; pointers en de goto zijn fantastische instructies om een programma volkomen onbegrijpelijk te maken. Dit is zeker waar als ze ondoordacht worden gebruikt, het is trouwens eenvoudig om pointers te creëren die maar wat in het wilde weg wijzen. Maar als pointers verstandig gebruikt worden, leveren ze duidelijke en eenvoudige programma's op. In de volgende paragrafen zal dit geïllustreerd worden.

8/4.6.1
Pointers en adressen

Omdat een pointer een adres van een object bevat, is het mogelijk het object "indirect" via de pointer te bereiken. Veronderstel dat x een variabele van het type int is en dat px een pointer is. De & operator geeft het adres van een object, zodat het statement

```
px = &x;
```

het adres van x aan de variabele px toekent; px wijst nu naar x. De &-operator kan alleen worden toegepast op variabelen, constructies als &(x+1) of &3 zijn verboden.

De *-operator beschouwt zijn operand als het adres van de te gebruiken waarde. Dus als y een integer is dan geeft

```
y = *px;
```

y de waarde waarnaar px wijst. Dus de statements:

```
px = &x;
y = *px;
```

geven y dezelfde waarde als wat het volgende doet:

```
y = x;
```

Het is altijd noodzakelijk om de variabelen die gebruikt worden te *declareren*.

```
int x, y;
int *px;
```

De declaratie van de pointer px is nieuw. Het betekent dat de combinatie *px een integer is.

Pointers kunnen voorkomen in expressies.

Bijvoorbeeld: als px naar een integer x wijst, dan kan *px overal voorkomen waar x kan staan.

y = *px + 1;

kent aan y de waarde x + 1 toe,

printf("%d /n", *px);

drukt de waarde van x af. Pointers mogen ook aan de linkerkant van het = teken staan. Als px naar x wijst, dan wordt x door

*px = 0;

op nul gezet, en door

*px + = 1; of (*px)++;

met één opgehoogd. De haakjes zijn noodzakelijk bij het laatste voorbeeld. Zonder deze haakjes zou de expressie px ophogen in plaats van waar px naar wijst, omdat de unaire operatoren, zoals * en ++, van rechts naar links worden geëvalueerd.

Tenslotte: omdat pointers variabelen zijn kan ook met hen gemanipuleerd worden zoals dat met variabelen kan. Als py ook een pointer van het type int is, dan kopiëert het statement

py = px;

de inhoud van px naar py, dus py zal gaan wijzen waar px naar wijst.

8/4.6.2
Pointers en functie argumenten

Omdat de argumenten van C-functies met het "call by value" mechanisme (de waarde van de argumenten wordt doorgegeven) worden doorgegeven, is er geen manier voor de functie om de variabelen die aan de functie worden mee gegeven te veranderen. Wat moet er gebeuren als een functie werkelijk een argument wil veranderen? Neem bijvoorbeeld een sorteerprogramma, dat twee foutief geplaatste elementen met een functie verwissel moet omdraaien. Het is niet genoeg om te schrijven

verwissel (a, b);

waarbij de functie verwissel als volgt gedefinieerd zou zijn

```
swap(x, y)   /* FOUT */
int x, y;
{
    int temp;

    temp = x;
    x = y;
    y = temp;
}
```

Vanwege het call by value mechanisme, kan de opdracht verwissel de argumenten a en b, die de routine aanriepen niet veranderen.

Er is wel een manier om het gewenste effect te bereiken. Het programma dat verwissel aanroept geeft pointers door waarvan de waarde wordt verwisseld.

verwissel (&a, &b);

werkt, omdat de & operator het adres van een variabele doorgeeft; &a is een pointer. In verwissel zelf moeten de argumenten zelf ook als pointers worden gedeclareerd,

en de werkelijke variabelen, a en b, worden daardoor gemanipuleerd.

```
swap(px, py) /* draai *px en
                *py om          */
int *px, *py;
{
    int temp;

    temp = *px;
    *px = *py;
    *py = temp;
}
```

8/4.6.3
Pointers en arrays

Er bestaat in C een sterke relatie tussen pointers en arrays. Elke operatie die door array-selectoren gedaan wordt, kan ook door pointers worden gedaan. De pointer versie is sneller, maar ook wel wat moeilijker om te begrijpen.

De declaratie

int a[10];

definiëert een array a met 10 elementen, met opvolgende objecten die a[0], a[1], ... , a[9] heten. De notatie a[i] betekent: het element van array a, i posities van het begin. Als pa een pointer is naar een integer dan kent de assignment

pa = &a[0];

het adres van het eerste element van a toe aan pointer pa. De assignment

x = *pa;

zal de waarde van a[0] aan x toekennen.

Als pa naar een element uit de array a wijst, dan wijst pa+1 per definitie naar het volgende element uit die array en wijst pa+i naar het i-de element voor pa en pa+i naar het i-de element na pa. Dus als pa naar a[0] wijst, dan refereert

*(pa+1)

aan de waarde van a[1], pa+1 is het adres van a[1], *(pa+1) is de waarde van a[1].

Dit geldt voor elk type dat in array a zit. Alle pointer arithmetiek wordt aangepast aan de grote van objecten die in een array worden opgeslagen. Dus in pa+i wordt i vermenigvuldigd met de grote van de objecten waar pa naar wijst, voordat de optelling bij pa plaats vindt.

Array indexering en pointer arithmetiek liggen dicht by elkaar. In feite wordt een array referentie door de compiler omgezet naar een pointer bij het begin van de array. Het gevolg is dat een arraynaam eigenlijk een pointer-expressie is. Omdat de naam van een array het synoniem is voor het adres van het nul-element, betekent dat, dat de assignment

pa = &a[0]

ook geschreven kan worden als

pa = a

Dit betekent dat een referentie naar a[i] tevens geschreven kan worden als *(a+i). Als a[i] geëvalueerd wordt, converteert C het direkt naar *(a+i), de twee vormen zijn volledig identiek. Wel zit er een verschil tussen een array naam en een pointer naam. Een pointer is een variabele, dus pa=a en pa++ zijn zinnige operaties. Maar een array naam is een constante en

4.6 Pointers en arrays

geen variabele: constructies zoals a=pa en a++ of p=&a zijn verboden.

Als een array naam wordt doorgegeven aan een functie, dan wordt de locatie van het begin van de array doorgegeven. Binnen de aangeroepen functie is dit argument een variabele, zoals elk argument. De naam van de array is dan een pointer, dus een variabele die een adres bevat. Met deze kennis kan een nieuwe versie van strlen geschreven worden. Deze functie bepaalt de lengte van een string.

```
strlen(s) /* retoeneer de lengte
             van een string      */
char *s;
{
   int n;

   for (n = 0; *s != '\0'; s++)
      n++;
   return(n);
}
```

Het verhogen van s is volkomen legaal, omdat het een pointervariabele is; s++ heeft geen effect op de stringpointer in de functie die strlen aanroept, maar het heeft wel effect op strlen's eigen copy van het adres.

De parameters in de functie-definitie

```
char s[];
```

en

```
char *s;
```

zijn equivalent. Als een arraynaam wordt doorgegeven aan een functie, dan kan die functie dit zien als een array of een pointer, zoals dat binnen die functie het beste uitkomt. De functie kan zelfs de twee vormen door elkaar gebruiken, als dit handiger of duidelijker is.

Het is mogelijk om een deel van een array aan een functie door te geven. Dit wordt gedaan door een pointer naar het begin van de sub-array door te geven. Bijvoorbeeld als a een array is, dan geven

```
f(&a[2])
```

en

```
f(a+2)
```

beide aan functie f het adres van het element a[2] door, omdat &a[2] en a+2 beide pointer expressies zijn, die wijzen naar het derde element uit array a. Binnen f kan het argument op twee manieren gedeclareerd worden:

```
f(arr)
int arr[ ];

   ...
```

of

```
f(arr)
int *arr;

   ...
```

De functie f ziet niet dat het doorgegeven argument in werkelijkheid naar een deel van een grotere array wijst. In feite is het ook niet van belang.

8/4.6.4 Adresberekeningen

Als p een pointer is, dan incrementeert p++ p tot het punt dat naast p ligt. Deze en gelijksoortige constructies zijn de eenvoudigste vormen van adresberekeningen.

De integratie van pointers, arrays en adresberekeningen is een van de sterkste punten van C. Dit wordt geïllustreerd aan de hand van een simpele storage allocator. Er zijn twee routines: alloc(n) geeft een pointer p terug naar gebied van n opeenvolgende characters, en free(p). Deze maakt ruimte vrij zodat hij later weer gebruikt kan worden. Het is een eenvoudige allocator omdat free in omgekeerde volgorde moet worden aangeroepen m.b.t. alloc.
Met andere woorden, het storagemanagement werkt met behulp van het stackmechanisme; last-in first-out. De standaard C-bibliotheek heeft de functies alloc en free die deze beperking niet heeft. Hier zal later nog op worden teruggekomen.

Deze implementatie maakt gebruik van een grote character array, waar door alloc de geheugenstukjes uit worden gehaald. Deze array is alleen toegankelijk voor alloc en free. Omdat de functies aan de buitenkant alleen met pointers werken en niet met array indices, behoeven andere routines niet van het bestaan van de character array af te weten. Deze array kan dus als static gedeclareerd worden. Dit houdt in dat de array alleen zichtbaar is in de file waarin gedefinieerd.

Er moet ook bijgehouden worden hoeveel er van allocbuf is gebruikt. Dit wordt gedaan door een pointer allocp die naar het eerste vrije element wijst. Als aan alloc n characters worden gevraagd, dan checkt alloc of er nog genoeg vrije ruimte is in allocbuf. Als dit het geval is dan retourneert alloc de huidige waarde van allocp en hoogt het allocp op met n, zodat allocp weer naar het begin van de vrije ruimte wijst. free(p) zet alleen allocp op de waarde van p als p binnen allocbuf ligt.

Over het algemeen kan een pointer zoals elke andere variabele geïnitialiseerd worden, alhoewel de enige zinnige waardes NULL of een expressie waar adressen in voorkomen zijn die al eerder gedefinieerd zijn.

Machines en ook compilers zijn niet altijd compatibel met de standaard. Ook hier blijkt dat weer. De C pre-processor heeft een mechanisme om deze machine- en/of compiler afhankelijke stukken binnen de programmatuur conditioneel te compileren. Hier wordt dit gedaan met behulp van de pre-processor statements:
ifdef, wat test of een macro variabele naam is gedefinieerd,
else, heeft dezelfde werking als de bekende else # endif, die de pre-processor expressie afsluit.

De Pro-line compiler laat niet toe dat statische variabelen bij de declaratie geïnitialiseerd worden. Door voor die initialisatie een aparte routine te schrijven, die bij het opstarten van het programma direct geëxecuteerd wordt, wordt deze onvolkomenheid opgelost.

De declaratie:

```
# ifdef PROLINE

static char *allocp;/* volgende vrij positie */

allocinit()
```

4.6 Pointers en arrays

```
#define PROLINE

#define NULL 0 /* pointer waarde voor fout indicatie */
#define ALLOCSIZE 1000 /* grote van de beschikbare ruimte */

static char allocbuf[ALLOCSIZE]; /* ruimte voor alloc */
#ifdef PROLINE

static char *allocp; /* volgende vrij positie */

allocinit()
{
    allocp = allocbuf; /* initialiseer */
}

#else

static char *allocp = allocbuf; /* volgende vrije positie */

#endif

char *alloc(n) /* retoeneerd een pointer naar n characters */
int n;
{
    if (allocp + n <= allocbuf + ALLOCSIZE) { /* past */
        allocp += n;
        return(allocp -n); /* oude waarde van allocp */
    } else                /* niet genoeg ruimte */
        return(NULL);
}

free(p) /* vrije ruimte begint bij p */
char *p;
{
    if (p >= allocbuf && p < allocbuf + ALLOCSIZE)
        allocp = p;
}
```

allocp=allocbuf;/*initialiseer*/

else

static char *allocp = allocbuf;
/* volgende vrije positie */

endif

definieert allocp en laat allocp naar het begin van allocbuf wijzen, wat de eerste vrije positie is, als het programma start. Een ander manier om dit te schrijven is

static char *allocp = &allocbuff[0];

Deze declaratie wordt echter alleen uitge-

voerd als de pre-processor PROLINE niet gedefinieerd is. Als de declaratie

define PROLINE

is opgenomen, dan wordt het eerste gedeelte, tussen de # if en de # else gecompileerd en zal de main routine in het begin de alloc init functie moeten aanroepen.

De test

```
if(allocp+n <= allocbuf + ALLOCSI-
ZE)
```

checkt of er genoeg ruimte is, om aan het verzoek voor n characters te voldoen. Als er geen ruimte is, moet alloc een signaal afgeven om dit kenbaar te maken. C garandeert dat geen enkele pointer, die naar geldige data wijst, nul zal bevatten. Dus de return-waarde nul kan worden gebruikt om aan te geven dat er iets bijzonders aan de hand is; in dit geval geen ruimte. De macro NULL wordt gebruikt om 0 aan te geven, het is duidelijker dan het cijfer 0. In het algemeen heeft het geen betekenis om een integerwaarde aan pointers toe te kennen.

De testen zoals

```
if (allocp + n <= allocbuf + ALLOCSIZE)
```

en

```
if (p >= allocbuf && p < allocbuf
+ ALLOCSIZE)
```

laten enkele facetten van pointerberekening zien. Ten eerste mogen pointers vergeleken worden. Als p en q naar dezelfde array wijzen, dan werken de relaties <, >=, ==, != enz. uitstekend. Het is niet zinnig om pointers te vergelijken, die in verschillende arrays zitten. Waar arrays namelijk in het geheugen geplaatst worden is volkomen machine afhankelijk. Een programma kan dus op de ene machine prima lopen, maar op een andere machine op vreemde wijze 'crashen'.

Ten tweede kunnen pointers en integers bij elkaar worden opgeteld of van elkaar worden afgetrokken, zoals al eerder besproken is. De compiler zorgt er, door middel van scaling voor, dat het juiste element aangewezen wordt.

8/4.6.5 Character pointers en functies

Een string geschreven als

```
"Dit is tekst"
```

is een array van characters. Intern wordt dit opgeslagen met een \ 0 aan het einde, zodat programma's het einde van het programma kunnen vinden. De string is dus één byte groter dan het aantal tekens tussen de quotjes. De meest voorkomende plaats van string constanten is waarschijnlijk als functie-argument, zoals in

```
printf ("Hallo \n");
```

Als een string zoals deze voorkomt in een programma, dan wordt deze string via een pointer gebruikt, printf krijgt dus een pointer naar de charcaterarray door.

Als tekst gedeclareerd is door:

```
char *tekst;
```

dan assigneert het statement:

```
tekst = "Dit is tekst";
```

4.6 Pointers en arrays

aan tekst een pointer toe die naar de tekens van die string wijst. Dit is dus geen kopie van de string. Er worden hier alleen pointers gebruikt. C heeft geen operatoren om hele strings te manipuleren.

Meer facetten van pointers en arrays zullen geïllustreerd worden aan de hand van enkele standaard bibliotheek routines. De eerste functie strcpy(a, b) kopieert de string b naar a. De array versie van deze functie is:

```
strcpy(s, t) /* copieer t naar s */
char s[], t[];
{
    int i;

    i = 0;
    while ((s[i] = t[i]) != '\0')
        i++;
}
```

De tweede versie van strcpy die pointers gebruikt is

```
strcpy(s, t) /* copieer t naar s
                pointer versie 1 */
char *s, *t;
{
    while ((*s = *t) != '\0') {
        s++;
        t++;
    }
}
```

Omdat de argumenten met het "call by value" mechanisme worden doorgegeven, kunnen s en t binnen strcpy als variabelen gebruikt worden. Hier worden ze gebruikt als geïnitialiseerde pointers, die over de strings heen lopen, todat het \ 0 teken wordt bereikt. De string is dan namelijk gekopieerd.

Een andere manier die meer eigen aan C is, is de volgende vorm van strcpy.

```
strcpy(s, t) /* copieer t naar s
                pointer versie 2 */
char *s, *t;
{
    while ((*s++ = *t++) != '\0')
        ;
}
```

In deze versie worden s en t binnen het test-gedeelte geïncrementeerd. Het character dat door t wordt aangewezen is de waarde van *t++; de postfix waarde ++ hoogt t pas op nadat het character is opgehaald. Een soorgelijke redenering gaat op voor s. Het character wat gekopieerd wordt wordt vergeleken met \ 0 en de uitkomst van deze test bepaalt of de lege loop doorlopen moet worden. Het totale effect hiervan is dat alle characters van t gekopieerd worden tot en met het laatste character, de \0.

De tweede pointerversie van strcpy was al korter dan de eerste en de derde en laatste pointer versie van strcpy is nog iets korter. De vergelijking met \ 0 is namelijk overbodig, dus de functie kan ook als volgt worden geschreven:

4.6. Pointers en arrays

```
strcpy(s, t) /* copieer t naar s
                pointer versie 3 */
char *s, *t;
{
   while (*s++ = *t++)
}
```

Alhoewel dit er op het eerste gezicht nogal cryptisch uitziet, is het toch een veel gebruikte notatie en is het handig om deze schrijfwijze te leren, ook al is het alleen maar omdat deze notatie vaak in C-programma's voorkomt. C is een expressionele taal, wat inhoudt dat elke expressie een waarde afgeeft, die gebruikt kan worden. Meestal wordt van die vrijkomende waarde geen gebruik gemaakt, door er bijvoorbeeld een puntkomma achter te schrijven. In het bovenstaande geval wordt de waarde die van de expressie vrijkomt gebruikt als test conditie. Als die waarde 0 is dan is, de conditie FALSE en als die waarde niet 0 is, dan is de conditie TRUE. De waarde die bij een expressie vrijkomt is de waarde van de laatste assignment.

De tweede routine is strcmp(s, t), die vergelijkt de characters in een string met elkaar en retourneert een negatieve, nul of een positieve waarde als s lexicografisch kleiner, gelijk of groter is dan t. De geretourneerde waarde wordt verkregen door de characters van elkaar af te trekken, die op de eerste positie staan waar s en t niet overeenkomen.

```
strcmp(s, t) /* retoeneer <0 als
                s<t, 0 als s==t,
                >0 als s>t       */
char s[], t[];
{
   int i;

   i = 0;
   while (s[i] == t[i])
      if (s[i++] == '\0')
         return(0);
   return(s[i] - t[i]);
}
```

De bovenstaande functie is de arrayversie van strcmp, de pointer versie van deze functie volgt hier onder:

```
strcmp(s, t) /* retoeneer <0 als
                s<t, 0 als s==t,
                >0 als s>t       */
char *s, *t;
{
   for ( ; *s == *t; s++, t++)
      if (*s == '\0')
         return(0);
   return(*s - *t);
}
```

8/4.6.6 Pointers zijn geen integers

Vaak wordt gedacht dat pointers hetzelfde zijn als integers. Dit is een misverstand dat bij de overdracht van programma's van de ene naar de andere machine steeds weer aan het licht komt. Dit misverstand wordt in de hand gewerkt, doordat op ver-

scheidene machines pointers en integers ongestraft door elkaar gehaald kunnen worden. Vaak worden er routines geschreven die pointers retourneren, waarbij de pointer-declaratie vergeten wordt. Deze fout moet zoveel mogelijk vermeden worden. Neem bijvoorbeeld de functie strsave(s). Deze functie kopiëert de string s naar een veilige plaats, die verkregen wordt met behulp van alloc. strsave en retourneert een pointer naar de gekopieerde string. De functie behoort als volgt geschreven te worden.

```
#define NULL 0

char *strsave(s) /* save string */
char *s;
{
   char *p, *alloc();

   if ((p = alloc(strlen(s) + 1))
               != NULL)
      strcpy(p, s);
   return(p);
}
```

In de praktijk bestaat er een sterke tendens om de functie als volgt te schrijven:

```
#define NULL 0

char *strsave(s) /* save string */
{
   char *p;

   if ((p = alloc(strlen(s) + 1))
               != NULL)
      strcpy(p, s);
   return(p);
}
```

Dit werkt op vele machines, omdat het default type voor functies en argumenten de integer is en integers op die machines veilig met pointers uitgewisseld kunnen worden. Desalniettemin moet dit toch afgewezen worden, omdat het correct werken van zo'n functie afhankelijk is van de machine. Het is beter om alle declaraties in het programma op te nemen.

8/4.6.7 Meer-dimensionale arrays

In deze paragraaf zullen de meerdimensionele arrays aan de orde komen. De meer-dimensionale arrays zullen worden verduidelijkt aan de hand van een datum conversie probleem; van de dag van de maand naar de dag van het jaar en vice versa. Bijvoorbeeld wereld dierendag, 4 oktober, is de 277ste dag van een gewoon jaar en de 278ste dag van een schrikkeljaar. Stel er zijn twee functies: dag-jaar converteert de maand en de dag in de dag van het jaar en maand-dag converteert de dag van het jaar in de maand en de dag. Omdat de laatste functie twee waarden retourneert, namelijk de maand en de dag, zullen de argumenten pointers zijn:

maand-dag(1987, 277, &m, &d)

Het resultaat van deze aanroep zal zijn dat m de waarde 10 krijgt en d de waarde 4 (4 oktober).

Beide functies hebben dezelfde informatie nodig; namelijk een tabel met het aantal dagen die elke maand heeft. Omdat het aantal dagen per maand verschilt voor schrikkeljaren en normale jaren is er een array nodig met twee rijen voor de verschillende jaren. De array en de functies die voor de conversie zorgen volgen hier onder:

4.6. Pointers en arrays

```
static int dag_tabel[2][13] ={
   {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
   {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
};

dag_jaar(jaar, maand, dag) /* geef dag van het jaar */
int jaar, maand, dag;      /* van de maand en de dag */
{
   int i, schrikkel;

   schrikkel = jaar%4 == 0 && jaar%100 != 0 || jaar%400 == 0;
   for (i = 1; i < maand; i++)
      dag += dag_tabel[schrikkel][i];
   return(dag);
}

maand_dag(jaar, dag_jaar, pmaand, pdag) /* geef maand, dag */
int jaar, dag_jaar, *pmaand, *pdag; /* van de dag van het jaar */
{
   int i, schrikkel;

   schrikkel = jaar%4 == 0 && jaar%100 != 0 || jaar%400 == 0;
   for (i = 1; dag_jaar > dag_tabel[schrikkel][i]; i++)
      dag_jaar -= dag_tabel[schrikkel][i];
   *pmaand = i;
   *pdag = dag_jaar;
}
```

De array dag-tabel moet extern zijn voor zowel dag-jaar als maand-dag, zodat ze er beide gebruik van kunnen maken.

In C worden meer dimensionale arrays als "arrays van arrays" gedeclareerd. De indices van de arrays worden ook geschreven op een manier die daar op duidt:

dag-tabel[i][j]

in plaats van

dag-tabel[i, j]

zoals dit in de meeste andere talen gebeurt. De bovenstaande notatie wordt door de compiler omgezet in een equivalente pointer notatie. Bijvoorbeeld de expressie dag tabel[2][5] wordt geëxpandeerd tot de volgende vorm:

```
*(*dag-tabel+2)+5)
```

wat als volgt geëvalueerd wordt:
dag-tabel
een 2 bij 13 array, die geconverteerd wordt in een pointer naar de eerste array met dertien elementen.
dag-tabel+1
een pointer naar de tweede array van dertien elementen.
*(dag-tabel+1)
(de tweede) 13-elements array van integers die geconverteerd wordt naar een pointer naar de eerste integer uit deze array.
*(dag tabel+1)+5
een pointer naar de zesde integer in de tweede 13-elements array.
((dag tabel+2)+5)
(de zesde) integer (in de tweede 13-elements array).

Een array wordt geïnitialiseerd met een lijst van initialisators tussen braces. Elke rij van de twee-dimensionale array wordt geïnitialiseerd door een daarmee corresponderende sub-lijst. De array dag-tabel wordt gestart met een kolom 0 zodat voor de maandnummers de werkelijke nummers gebruikt kunnen worden die van 1 tot 12 lopen in plaats van 0 tot 11.

Als een twee-dimensionale array wordt doorgegeven aan een functie, dan moet de argument-declarie in die functie tevens de kolomdimensie bevatten. De dimensie van de rijen is irrelevant, want hetgene dat doorgegeven wordt is een pointer. Dus als de array dag-tabel aan een functie wordt doorgegeven, dan is de declaratie:

```
f(dag-tabel)
int dag-tabel[2][13];
  ...
```

De declaratie van het argument kan ook de onderstaande vorm hebben

```
int dag-tabel[ ][13];
```

omdat het aantal rijen irrelevant is. Ook is het mogelijk om het de volgende vorm te geven

```
int (*dag-tabel)[13];
```

Dit wil zoveel zeggen als: het argument is een pointer naar een array van 13 integers. De haakjes zijn noodzakelijk omdat de brackets [ ] een hogere precedence hebben dan *. Zonder haakjes

```
int *dag-tabel[13];
```

betekent de declaratie iets anders, hier is dit een array van 13 pointers, zoals dat beschreven wordt in de volgende paragraaf.

8/4.6.8
Pointer arrays en pointer naar pointers

Pointers zijn variabelen: dit betekent dat ze ook in arrays kunnen worden opgeslagen. Aan de hand van een sorteerprogramma, dat regels in alfabetische volgorde zet, zal dit geïllustreerd worden.

In sectie 4 is de Shell-sorteermethode besproken, die een array van integers sorteert. Hetzelfde algorithme zal hier gebruikt worden. Met dit verschil dat er nu met regels tekst wordt gewerkt in plaats van met integers. Er is een datarepresentatie nodig, die handig regels tekst van variabele lengte kan opslaan.

Hiervoor is de array van pointers uitermate geschikt. Als de te sorteren regels allemaal in een grote character array worden

4.6. Pointers en arrays

opgeslagen (alloc kan dit bijhouden), dan kan iedere regel bereikt worden door een pointer naar zijn eerste character. Zo'n pointer kan worden opgeslagen in een array. Twee regels kunnen worden vergeleken, door hun pointers door te geven, met behulp van strcmp. Als twee regels omgedraaid moeten worden, kunnen de pointers in de pointerarray worden omgedraaid en niet de tekst zelf. Dit bespaart de overhead van het verplaatsen van de werkelijke regels.

Het sorteerproces bevat drie onderdelen:

Lees alle regels
Sorteer ze.
Druk ze in volgorde af.

Zoals gewoonlijk is het het best om het programma in functies te verdelen, die het meest overeenkomen met de natuurlijke versie en met een hoofdroutine die alles bestuurd.

Om te beginnen zullen eerst de invoer en uitvoer routines besproken worden en daarna de sorteer routine. De invoer routine moet alle characters van elke regel opslaan, een array van regelpointers opbouwen, omdat deze informatie nodig is voor het sorteren en het afdrukken. Omdat de invoerroutine maar een aantal regels op kan slaan, kan het ook een foutboodschap afgeven. In dit geval is dat –1, om aan te geven dat geen invoer kan worden gegeven. De uitvoer functie behoeft slechts de regels af te drukken zoals ze in de array van pointers staan.

De newline aan het einde van de regel- wordt er afgestript, om er voor te zorgen dat de regels niet in de verkeerde volgorde worden gezet tijdens het sorteren.

4.6 Pointers en arrays

```
#define PROLINE
#define NULL 0
#define LINES 100 /* maximum aantal regels */

main()     /* sorteer de invoer */
{
   char *lineptr[LINES]; /* pointer naar de regels */
   int  nlines;          /* aantal gelezen regels */

#ifdef PROLINE

   allocinit();

#endif

   if ((nlines = readlines(lineptr, LINES)) >= 0) {
      sort(lineptr, nlines);
      writelines(lineptr, nlines);
   }
   else
      printf("invoer te groot om te sorteren\n");
}

#define MAXLEN 1000

readlines(lineptr, maxlines) /* lees de invoer */
char *lineptr[];
int  maxlines;
{
   int len, nlines;
   char *p, *alloc(), line[MAXLEN];

   nlines = 0;
   while ((len = getline(line, MAXLEN)) > 0)
      if (nlines >= maxlines)
         return(-1);
      else if ((p = alloc(len)) == NULL)
         return(-1);
      else {
         line[len-1] = '\0'; /* verwijder de newline */
         strcpy(p, line);
         lineptr[nlines++] = p;
      }
   return(nlines);
}
```

```
writelines(lineptr, nlines) /* schrijf de uitvoer */
char *lineptr[];
int  nlines;
{
   int i;

   for (i = 0; i < nlines; i++)
      printf("%s\n", lineptr[i]);
}
```

4.6. Pointers en arrays

Een nieuw commando is de declaratie van lineptr:

char *lineptr[LINES];

wat betekent dat lineptr een array van LINES elementen is, waarvan elk element een character pointer is. Dus lineptr[i] is een character pointer en *lineptr[i] is een character.

Omdat lineptr zelf een array is, die wordt doorgegeven aan writelines, kan het als een pointer worden behandeld op precies dezelfde manier als in de vorige voorbeelden. De functie kan dan als volgt geschreven worden.
De pointer *lineptr wijst in het begin naar de eerste regel, na elke incrementie wijst het naar de volgende regel, terwijl nlines gedecrementeerd wordt.

Nu de invoer en de uitvoer geregeld zijn, kan het sorteren worden besproken. Het Shell sorteerprogramma uit sectie 4 behoeft slechts een weinig gewijzigd te worden en het vergelijken dient door een aparte functie te worden gedaan. De basis van het algorithme blijft hetzelfde:

```
writelines(lineptr, nlines) /* schrijf de uitvoer */
char *lineptr[];
int  nlines;
{
   while (--nlines)
      printf("%s\n", *lineptr++);
}
```

```
sort(v, n) /* sorteer strings v[0] ... v[n-1] */
char *v[]; /* in oplopende volgorde */
int n;
{
   int gap, i, j;
   char *temp;

   for (gap = n/2; gap > 0; gap /= 2)
      for (i = gap; i < n; i++)
         for (j = i-gap; j >= 0; j -= gap) {
            if (strcmp(v[j], v[j+gap]) <= 0)
               break;
            temp = v[j];
            v[j] = v[j+gap];
            v[j+gap] = temp;
         }
}
```

Omdat het individuele element v een character pointer is, moet temp dit ook zijn zodat de een naar de ander gekopieerd kan worden.

Het geschreven programma is rechttoerechtaan, zodat het (bijna) direkt zal werken. Het kan sneller gemaakt worden door bijvoorbeeld de inkomende regels direkt in een array te stoppen in plaats van de plaats ervan te verbergen door alloc te gebruiken. Maar het is altijd verstandig om eerst een ontwerp van iets simpels te maken en je pas later bezig te gaan houden met de "efficiëntie".

```
6.18.c

char *maand_naam(n)  /* retoeneer
                        de naam van de
                        n-de maand     */
int n;
{
   static char *naam[] = {
       "foutive maand",
       "Januari",
       "Februari",
       "Maart",
       "April",
       "Mei",
       "Juni",
       "Juli",
       "Augustus",
       "September",
       "Oktober",
       "November",
       "December"
   };

   return((n < 1 || n > 12) ? naam[0] : naam[n]);
}
```

8/4.6.9 De initialisatie van pointer arrays

Veronderstel dat er een functie maand-naam(n) moet worden geschreven, die functie retourneert een pointer naar een characterstring die de naam van de n-de maand bevat. Dit is de ideale applicatie om een interne statische array te gebruiken. maand-naam heeft een eigen array van characterstrings en retourneert een pointer naar een van de strings als ze correct wordt aan geroepen. Deze paragraaf laat zien hoe array's worden geïnitialiseerd.

De syntax is gelijksoortig als vorige initialisaties:

4.6. Pointers en arrays

De declaratie van naam, wat een array van character pointers is, is hetzelfde als die van lineptr in het sorteervoorbeeld. De initialisator is een lijst van character strings; elk van die strings correspondeert met een positie in de array. Omdat de grootte van de array niet gespecificeerd is, moet de compiler het aantal initialisatoren tellen en het aantal elementen zelf invullen.

8/4.6.10 Pointers en meer dimensionale arrays

Diegene, die voor het eerst met C in aanraking komen, hebben soms problemen met het verschil tussen een tweedimensionale array en een array van pointers, zoals het bovenstaande voorbeeld. Gegeven de declaraties:

```
int a[10][10];
int *b[10];
```

Het gebruik van a en b kan hetzelfde zijn, a[4][3] en b[4][3] zijn beide legale referenties naar een integer. Maar a is een echte array: alle 100 integers zijn beschikbaar en de normale indexberekening wordt uitgevoerd om een bepaald element te vinden. Daarentegen reserveert de declaratie van b alleen maar 10 pointers; elke pointer moet geset worden om naar een array van integers te wijzen. Veronderstel dat elke pointer naar een array van 10 integers wijst, dan is er ruimte voor 100 integers gereserveerd, plus de ruimte voor 10 pointers. Dus een array van pointers gebruikt iets meer ruimte en heeft een extra initialisatiestap nodig. Maar dit heeft twee voordelen: een array element wordt bereikt door indirectie in plaats van door vermenigvuldigen en optellen. De rijen van de array mogen van verschillende lengte zijn. Dit houdt in, dat niet elk element van b hoeft te wijzen naar een 10-elements vector; sommige mogen naar een 2-elements, sommige naar een 20-elements en sommige mogen zelfs nergens naar wijzen.

8/4.6.11 Commando regel argumenten

Bij elk systeem dat C ondersteunt, is er een mechanisme wat het mogelijk maakt argumenten van de commando regel door te geven als parameters voor het te executeren programma. main wordt aangeroepen met twee argumenten. Het eerste (dat altijd argc wordt genoemd) geeft het aantal commando argumenten aan en het tweede (argv) is een pointer naar een array van character strings die de argumenten bevat.

Een eenvoudige illustratie van de noodzakelijke declaraties en het gebruik is het programma echo, wat slechts de commando regel argumenten echo't. Dus als het commano

```
echo hallo, wereld
```

wordt gegeven, dan ziet het resultaat er als volgt uit:

```
hallo, wereld
```

Per definitie is argv[0] de naam van het programma, dus argc is tenminste 1. In het bovenstaande voorbeeld is argc 3 en argv[0], argv[1] en argv[2] zijn respectievelijk 'echo', 'hallo,' en 'wereld'.
Het eerste echte argument is argv[1] en het laatste is argv[n-1]. Als argc gelijk is aan 1, dan zijn er geen argumenten gegeven na de naam van het programma. Het programma echo illustreert dit.

4.6 Pointers en arrays

```
main(argc, argv) /* druk argumenten af versie 1 */
int argc;
char *argv[];
{
    int i;

    for (i = 1; i < argc; i++)
        printf("%s%c", argv[i], (i <argc-1) ? ' ' : '\n');
}
```

Omdat argv een pointer is naar een array van pointers zijn er verscheidene manieren om dit programma te schrijven. Hier onder volgen nog twee voorbeelden.

```
main(argc, argv) /* druk argumenten af versie 2 */
int argc;
char *argv[];
{
    while (--argc > 0)
        printf("%s%c", *++argv, (argc > 1) ? ' ' : '\n');
}
```

Omdat argv een pointer is die naar het begin van de arcumenten array wijst, zal argv na deze te hebben geïncrementeerd (++argv) gaan wijzen naar argv[1]. Bij elke incrementie zal het naar het volgende argument wijzen; *argv is de pointer naar dat argument. Tegelijkertijd wordt argc gedecrementeerd en zodra het gelijk wordt aan nul zijn er geen argumenten meer om af te drukken. De derde versie volgt hier onder.

```
main(argc, argv) /* druk argumenten af versie 3 */
int argc;
char *argv[];
{
    while (--argc > 0)
        printf((argc > 1) ? "%s" : "%s\n", *++argv);
}
```

Deze versie toont dat het format argument van printf ook een expressie kan zijn.

Een tweede voorbeeld is een programma wat een patroon in een tekst zoekt en de regel afdrukt waar dit patroon in staat.

```
#define MAXLINE 1000

main(argc, argv) /* vindt het patroon */
int argc;
char *argv[];
{
   char line[MAXLINE];

   if (argc != 2)
      printf("Gebruik: zoek patroon\n");
   else
      while (getline(line, MAXLINE) > 0)
         if (index(line, argv[1]) >= 0)
            printf("%s", line);
}
```

8/4.6.12 Pointers naar functies

In C is een functie zelf geen variabele, maar het is mogelijk om een pointer naar een functie te definiëren; waarmee gemanipuleerd kan worden, kan worden door gegeven aan functies, in arrays worden geplaatst, etc. Dit zal worden geïllustreerd aan de hand van het sorteer programma dat al eerder in deze sectie besproken is. Dit programma krijgt een optioneel argument -n, als dit wordt gebruikt dan sorteert het numeriek in plaats van lexicografisch.

Een sorteerroutine bestaat uit drie delen: een vergelijking die de ordening van de objecten bepaalt, een verwisseling die de objecten verwisselt en een sorteer algorithme dat de vergelijkingen en de verwisselingen uitvoert totdat de objecten in de juiste volgorde staan.

Het sorteer algorithme is onafhankelijk van de vergelijk- en de verwisseloperaties, dus door verschillende vergelijk- en verwisselfuncties aan het sorteer programma mee te geven, kan op andere criteria gesorteerd worden. Deze methode wordt gebruikt bij de nieuwe sorteer routine.

De lexicografische vergelijking van twee regels wordt gedaan door strcmp en het verwisselen door swap. De numerieke vergelijking van twee regels zal door numcmp worden uitgevoerd. Deze drie functies worden gedeclareerd in main en pointers naar ze worden doorgegeven aan sort. sort roept op zijn beurt de functies via de pointers aan.

4.6 Pointers en arrays

```
#define LINES 100   /* maximaal aantal te sorteren regels */

main(argc, argv) /* sorteer de invoer */
int argc;
char *argv[];
{
   char *lineptr[LINES]; /* pointers naar de regels */
   int  nlines;          /* aantal gelezen invoer regels */
   int  strcmp(), numcmp(); /* vergelijkings functies */
   int  swap();          /* verwissel functie */
   int  numeric = 0;     /* 1 als er numeriek gesorteerd
                            moet worden */

   if (argc > 1 && argv[1][0] == '-' && argv[1][1] == 'n')
      numeric = 1;
   if ((nlines = readlines(lineptr, LINES)) >= 0) {
      if (numeric)
         sort(lineptr, nlines, numcmp, swap);
      else
         sort(lineptr, nlines, strcmp, swap);
      writelines(lineptr, nlines);
   } else
      printf("Invoer te groot om te sorteren\n");
}
```

strcmp, numcmp en swap zijn adressen van functies, omdat bekend is dat ze functies zijn behoeft de &-operator niet gebruikt te worden. De compiler zorgt ervoor dat de adressen van de functies worden doorgegeven.

De volgende stap is het aanpassen van sort:

4.6. Pointers en arrays

```
sort(v, n, comp, exch) /* sorteert v[0] ... v[n-1] */
char *v[];
int n;
int (*comp)(), (*exch)();
{
    int gap, i, j;

    for (gap = n/2; gap > 0; gap /= 2)
        for (i = gap; i < n; i++)
            for (j = i - gap; j >= 0; j -= gap) {
                if ((*comp)(v[j], v[j + gap]) <= 0)
                    break;
                (*exch)(&v[j], &v[j + gap]);
            }
}
```

De declaratie

int (*comp) ()

duidt aan dat comp een pointer naar een functie is die een integer retourneert. Het eerste paar hakjes is noodzakelijk, zonder ze zou

int *comp()

betekenen dat comp een functie is die een pointer naar een integer retourneert, wat duidelijk iets anders is.

Het gebruik van comp in de regel

if ((*comp)(v[j], v[j+gap]) <= 0)

is in overeenstemming met de declaratie: comp is een pointer naar een functie, *comp is de functie en

(*comp)(v[j], v[j+gap])

roept de functie aan.

Hier onder volgt numcmp, deze functie vergelijkt twee strings op hun numerieke waarde.

```
numcmp(s1, s2) /* vergelijk
s1 en s2 */ char *s1, *s2;
{
    float atof(), v1, v2;

    v1 = atof(s1);
    v2 = atof(s2);
    if (v1 < v2)
        return(-1);
    else if (v1 > v2)
        return(1);
    else
        return(0);
}
```

4.6 Pointers en arrays

De laatste stap is om een functie swap toe te voegen die twee pointers verwisselt. Zo iets dergelijks is al eerder in deze sectie getoond.

```
swap(px, py) /* verwissel *px en *py */
char *px[], *py[];
{
    char *temp;

    temp = *px;
    *px = *py;
    *py = temp;
}
```

8/4.7

Structures

Een structure is een verzameling van één of meer variabelen, die van een verschillend type mogen zijn en samengevoegd zijn onder een naam. In Pascal worden structures records genoemd.

Het traditionele voorbeeld van een structure is de loonstrook van een werknemer, die wordt bepaald door een aantal kenmerken zoals: naam, adres, fiscaal nummer, salaris, etc. Enkele van deze kenmerken kunnen op zich ook weer structures zijn, zoals bijvoorbeeld het adres.

Structures helpen om gecompliceerde data te organiseren, zeker in gecompliceerde programma's omdat grote programma's in veel gevallen zeer geschikt zijn om data te groeperen. In deze paragraaf wordt het gebruik van structures geïllustreerd. De programma's die hiervoor gebruikt worden zijn over het algemeen groter dan de meeste andere die u nu al bent tegen gekomen, toch blijven ze nog van een bescheiden omvang.

8/4.7.1
De Basis

De datum conversie routines uit paragraaf 6 zijn zeer geschikt om structures in te gebruiken. De datum bestaat uit verschillende delen, zoals de dag, maand en het jaar, maar ook de dag van het jaar, en de naam van de maand. Deze vijf variabelen kunnen allemaal in een enkele structure worden gevangen, zoals:

```
struct  datum{
    int   dag;
    int   maand;
    int   jaar;
    int   dag_jaar;
    char  maand_naam[4];
};
```

Het keywoord struct introduceert een declaratie van een structure, de declaratie is de lijst van declaraties tussen de braces. Een optionele naam, die een structure tag wordt genoemd, mag na het woord struct volgen (in dit geval is dat datum). De tag geeft deze structure een naam en kan gebruikt worden als een verkorte vorm voor de gedetailleerde declaratie.

De elementen of variabelen die in een structure voorkomen worden members genoemd. Een structure member of tag of een gewone variabele kunnen dezelfde naam hebben, zonder dat dit tot conflicten leidt, omdat er altijd een onderscheid gemaakt kan worden. Dit door naar de context te kijken waarin ze gebruikt worden. Het is natuurlijk een kwestie van stijl, dat men alleen voor sterk gerelateerde objecten dezelfde naam zal kiezen.

De rechter brace, welke het einde van de

4.7 Structures

lijst van members bepaalt, kan worden gevolgd door een lijst van variabelen. Bijvoorbeeld

```
struct{.....} x, y, z;
```

Dit is syntactisch analoog aan

```
int x, y, z;
```

Elk statement declareert x, y en z als variabelen van een type met een naam, met als gevolg dat geheugenruimte voor ze wordt gealloceerd.

Een structure declaratie die niet wordt gevolgd door een lijst van variabelen alloceert geen geheugenruimte, ze beschrijft in dit geval slechts de vorm van de structure. Als de structure wordt getagged, dan kan deze tag worden gebruikt om variabelen van die structure te definiëren. Bijvoorbeeld, gegeven de bovenstaande declaratie van datum,

```
struct datum d;
```

definieert dan een variabele d. d is een variabele van het type datum. Een externe of statische structure kan worden geïnitialiseerd, met een lijst van initialisatoren.

```
struct datum d={ 4, 7, 1776, 186,
'Jul'};
```

Een member uit een structure wordt door middel van de volgende constructie gebruikt:

```
structure-naam.member
```

De structure member operator '.' verbindt de naam van de structure met de naam van de member. Beschouw het schrikkeljaar voorbeeld uit paragraaf 6 nogmaals,

```
schrikkel= d.jaar % 4 && d.jaar % 100
            != 0
       ||   d.jaar % 400==0;
```

of om de maand namen te checken,

```
if (strcmp(d.maand_naam, 'Aug')==0)
.....
```

of om het eerste teken van de naam van de maand om te zetten naar kleine letters,

```
d.maand_naam[0]= lower
(d.maand_ naam[0];
```

Structures mogen worden genest, een salaris strook kan er als volgt uitzien

```
struct persoon{
   char    naam[NAAMGROOTTE];
   char    adres[ADRESGROOTTE];
   int     fiscaal_nr;
   unsigned salaris;
   struct datum geboorte_dag;
   struct datum trouw_dag;
};
```

De structure persoon bevat een datum. Als piet, zoals hieronder wordt gedeclareerd,

```
struct persoon piet;
```

dan refereert

```
piet.geboorte_dag.maand
```

naar de maand van de geboorte. De structure member operator . werkt van links naar rechts.

8/4.7.2
Structures en functies

Er bestaan een aantal restricties voor de C-structures. De belangrijkste regels bepalen dat de enige operaties die op structures kunnen worden uitgevoerd zijn: die operaties die betrekking hebben op het adres met de & operator en die operaties die betrekking hebben op structure members. Dit heeft tot gevolg dat structures nooit als een geheel mogen worden gekopieerd of een waarde mogen krijgen en dat ze niet door- of terug kunnen worden gegeven aan of door een functie. (Dit is een oorspronkelijke eis van C, in nieuwe C-versies bestaat deze beperking niet meer). Pointers naar structures vallen (uiteraard) niet onder deze restricties. Een regel ten aanzien van structures is dat alleen de externe en statische structures geïnitialiseerd kunnen worden.
Evenals de automatische arrays worden de automatische structures niet geïnitialiseerd.

Met behulp van de herschreven datum conversie functies zullen enkele van de hierbovenstaande punten duidelijk gemaakt worden. Omdat de regels het verbieden dat een structure direct aan een functie wordt doorgegeven, moeten òf alle elementen apart worden doorgegeven, òf moet een pointer naar het geheel worden doorgegeven. De eerste functie die wordt herschreven is dag-jaar (zie ook 8/4.5). De oude versie gaf bijvoorbeeld het onderstaande te zien:

```
d.jaar_dag= dag_jaar(d.jaar, d.maand,
d.dag);
```

Een andere manier om dit te doen, gaat met behulp van het doorgeven van een pointer. Als trouwdag is gedeclareerd als

```
struct datum trouwdag;
```

en dag_jaar is herschreven, dan is het mogelijk

```
trouwdag.jaar_dag= dag_jaar (&trouw-
dag);
```

De functie moet worden herschreven, omdat zijn argument nu een pointer is in plaats van een lijst van variabelen.

```
dag_jaar(pd) /* maak de dag van het jaar met maand en dag */
struct datum *pd;
{
    int i, dag, schrikkel;

    dag = pd->dag;
    schrikkel = pd->jaar % 4 == 0 && pd->jaar % 100 != 0
                || pd->jaar % 400 == 0;
    for (i = 1; i < pd->maand; i++)
        dag += dag_tabel[schrikkel][i];
    return(dag);
}
```

De declaratie

```
struct datum *pd;
```

bepaalt dat pd een pointer is naar een structure van het type datum. De notatie:

4.7 Structures

pd->jaar

is nieuw. Als p een pointer naar een structure is, dan refereert

p->member-van-de-structure

naar die member in het bijzonder.

Omdat pd naar een structure wijst, kan het jaar member ook gerefereerd worden door

(*pd).jaar

maar omdat pointers naar structures zo vaak worden gebruikt, is de -> notatie in de taal ingevoerd als zijnde een handige korte notatiewijze. De haakjes zijn noodzakelijk in (*pd).jaar omdat de precedence van de structure operator '.' hoger is dan '*'. Zowel -> als . werken van links naar rechts, zodat

p->q->member
piet.geboorte_dag.maand

en

(p->q)->member
(piet.geboorte_dag).maand

equivalent zijn.

Voor de volledigheid volgt hier de andere functie herschreven, maand_dag, om gebruik te kunnen maken van de structure faciliteiten die C biedt.

```
maand_dag(pd) /* maak maand en dag van de dag van het jaar */
struct datum *pd;
{
   int i, schrikkel;

   schrikkel = pd->jaar % 4 == 0 && pd->jaar % 100 != 0
                    || pd ->jaar % 400 == 0;
   pd->dag = pd->dag_jaar;
   for (i = 1; pd->dag > dag_tabel[schrikkel][i]; i++)
      pd->dag -= dag_tabel[schrikkel][i];
   pd->maand = i;
}
```

De structure operatoren -> en . samen met () voor argument lijsten en [] voor array indexes, vormen de hoogste precedence in de precedence hierarchie. Dus verbinden zij de operanden erg sterk. Neem bijvoorbeeld de volgende declaratie

```
struct{
   int x;
   int *y;
} *p;
```

dan incrementeert

++p->x

x en niet p, omdat er haakjes 'gelezen' kunnen worden: ++(p->x). Haakjes kunnen gebruikt worden om deze impliciete binding te veranderen: (++p)->x incrementeert p na afloop (de laatste haakjes zijn niet noodzakelijk!).

Op dezelfde manier haalt *p->y op waar y naar wijst. *p->y++ incrementeert y nadat y gebruikt is. (*p->y)++ verhoogt y en *p++->y incrementeert p nadat y is gebruikt.

8/4.7.3
Arrays van structures

Arrays van structures zijn bijzonder handig om gerelateerde informatie in te beheren. Beschouw bijvoorbeeld een programma dat het aantal keren dat een C-keywood in een C-programma voorkomt telt. Hiervoor is een array van character strings nodig om de namen van de keywoorden in op te slaan en er is een array van integers nodig om het aantal malen van optreden bij te houden. Een mogelijkheid is om twee parallelle array's keywoord en keyaantal te gebruiken, zoals in

```
char *keywoord[NKEYS];
int    keyaantal[NKEYS];
```

Maar het simpele feit dat de arrays parallel zijn duidt erop dat een andere organisatie mogelijk is. Elk keywoord bestaat eigenlijk uit één paar:

```
char *keywoord;
int    keyaantal;
```

Dit levert dus een array van paren. De structure declaratie.

```
struct key{
    char *keywoord;
    int    keyaantal;
}keytabel[NKEYS];
```

definieert een array keytabel van structures van boven beschreven type en alloceert geheugenruimte voor deze array van structures. Elk element van de array is een structure. De bovenstaande definitie kan ook als volgt worden geschreven:

```
struct key{
    char *keywoord;
    int    keyaantal;
};

struct key keytabel [NKEYS];
```

Omdat de structure keytabel eigenlijk een verzameling van constante namen bevat is het het eenvoudigst om deze direct te initialiseren. De structure initialisatie is analoog aan eerdere initialisaties: de definitie wordt gevolgd door een lijst van initialisatoren, ingesloten door braces. De proline compiler kan deze initialisatie, evenals vorige initialisaties, niet aan. Hier rekening mee houdend leidt dit tot het volgende stuk code.

```
#ifndef PROLINE

struct key {
    char *keywoord;
    int  keyaantal;
} keytabel[] = {
    "break", 0,
    "case", 0,
    "char", 0,
    "continue", 0,
    "default", 0,
    /* ... */
    "unsigned", 0,
    "while", 0
};
```

4.7 Structures

```
#else

struct key {
    char *keywoord;
    int  keyaantal;
} keytabel[7];

proline_init()
{
    keytabel[0].keywoord = "break";
    keytabel[0].keyaantal = 0;
    keytabel[1].keywoord = "case";
    keytabel[1].keyaantal = 0;
    keytabel[2].keywoord = "char";
    keytabel[2].keyaantal = 0;
    keytabel[3].keywoord = "continue";
    keytabel[3].keyaantal = 0;
    keytabel[4].keywoord = "default";
    keytabel[4].keyaantal = 0;
    /* ...
                                    ... */
    keytabel[5].keywoord = "unsigned";
    keytabel[5].keyaantal = 0;
    keytabel[6].keywoord = "while";
    keytabel[6].keyaantal = 0;
}

#endif
```

4.7 Structures

De initialisatoren worden paarsgewijs opgegeven. Het is exacter om de initialisatoren van elke rij of structure in braces af te sluiten, zoals:

{ 'break', 0},
{'case', 0},

De binnenste braces zijn echter niet noodzakelijk als de initialisatoren simpele variabelen of character strings zijn en ze allemaal aanwezig zijn. Zoals gewoonlijk zal de compiler het aantal entries uit de array keytabel tellen, als de initialisatoren aanwezig zijn en de brackets [ ] leeg gelaten zijn.

Het keywoord-tel-programma begint met de definitie van keytabel. De main routine leest de input door gebruik te maken van de functie getword, dat elk woord afzonderlijk uit de invoer leest. Voor elk woord wordt gekeken of het in keytabel zit, met behulp van de binaire zoekroutine uit paragraaf 4.

```
#define MAXWORD 20

main() /* tel C keywoorden */
{
    int n, t;
    char word[MAXWORD];

#ifdef PROLINE

    proline_init();

#endif

    while ((t = getword(word, MAXWORD)) != EOF)
        if (t == LETTER)
            if ((n = binary(word, keytabel, NKEYS)) >= 0)
                keytabel[n].keyaantal++;
    for (n = 0; n < NKEYS; n++)
        if (keytabel[n].keyaantal > 0)
            printf("%4d %s\n", keytabel[n].keyaantal, keytabel[n].keywoord);
}
```

4.7 Structures

```
binary(word, tab, n) /* vindt word in tab[0] ... tab[n-1] */
char *word;
struct key tab[];
int n;
{
   int low, high, mid, cond;

   low = 0;
   high = n - 1;
   while (low <= high) {
      mid = (low + high) / 2;
      if ((cond = strcmp(word, tab[mid].keywoord)) < 0)
         high = mid - 1;
      else if (cond > 0)
         low = mid + 1;
      else
         return(mid);
   }
   return(-1);
}
```

De functie getword zal zo worden besproken, op dit moment is het voldoende om te weten dat het LETTER retourneert zodra een woord wordt gevonden en dat het het woord in het eerste argument zet.

De waarde van NKEYS is het aantal keywoorden uit keytabel. Alhoewel dit aantal met de hand kan worden bepaald, is het veel gemakkelijker en veiliger om dit door de machine te laten verrichten, zeker wanneer de lijst lang is. Een mogelijkheid is om de lijst af te sluiten met een NULL pointer, de keytabel te doorlopen totdat het einde gevonden is.

Dit kan echter eenvoudiger, omdat de grootte van de array al tijdens de compilatie bekend is. Het aantal entries is:

grootte van keytabel /
grootte van de structure key

C bevat de operator sizeof, die gebruikt kan worden om de grootte van een object te bepalen tijdens de compilatie.
De expressie

sizeof (object)

levert een integerwaarde af ter grootte van het gespecificeerd object (de grootte wordt aangegeven in bytes, de grootte van een char object is 1). Het object kan een variabele zijn, een array of een structure, of de naam van een basis type zoals int of double. Het kan ook de naam van een afgeleid type zoals een structure zijn.

4.7 Structures

In dit geval is het aantal keywoorden de array grootte gedeeld door de grootte van de array elementen. Deze berekening kan worden gebruikt in een # define statement om de waarde van NKEYS te definieëren:

define NKEYS (sizeof(keytabel) / sizeof (struct key))

Nu volgt de functie getword. Hier onder volgt een iets meer algemene getword dan nodig is voor het programma, maar het is er niet veel ingewikkelder op geworden. De functie getword retourneert het volgende woord uit de invoer, waarbij een woord een string is van letters en cijfers, beginnende met een letter. Een woord kan ook een enkel teken zijn.
Het type van het object is de returnwaarde van de functie, dit is LETTER als het een woord is, EOF voor einde van de file, of het teken zelf als het niet in het alfabet voorkomt.

```
getword(w, lim) /* haal het volgende woord uit de invoer */
char *w;
int lim;
{
    int c, t;

    if (type(c = *w++ = getch()) != LETTER) {
        *w = '\O';
        return(c);
    }
    while (--lim > 0) {
        t = type(c = *w++ = getch());
        if (t != LETTER && t != DIGIT) {
            ungetch(c);
            break;
        }
    }
    *(w - 1) = '\O';
    return(LETTER);
}
```

Getword maakt gebruik van getch en ungetch die in paragraaf 5 beschreven zijn. Als de verzameling van alfabetische tekens stopt, is getword een teken te ver gegaan. Door ungetch aan te roepen wordt dat teken in de invoer terug gestopt en kan het later weer gebruikt worden.

getword roept type aan om het type van elk individueel teken te bepalen. Hieronder volgt een versie voor het ASCII en Commodore (PETSCI) alfabet.

De symbolische constanten LETTER en DIGIT moeten waarden hebben, die niet in conflict zijn met de non-alfanumerieke tekens en EOF. Voor de hand liggende keuzes zijn dan

define LETTER 'a'
define DIGIT 'o'

De C-standaard bibliotheek bevat de routines: isalpha en isdigit, die op soort gelijke wijze werken.

4.7 Structures

```
type(c) /* retoerneer het type van het ASCII teken */
int c;
{
   if (c >= 'a' && c <= 'z' || c >= 'A' && c <= 'Z')
      return(LETTER);
   else if (c >= '0' && c <= '9')
      return(DIGIT);
   else
      return(c);
}
```

8/4.7.4
Pointers naar structures

Het gebruik van pointers naar structures zal aan de hand van het keywoord-tel-programma worden geïllustreerd. Daartoe zal dit programma gewijzigd worden. De externe declaratie van keytabel verandert niet, maar main en binary moeten wel veranderd worden.

```
#define MAXWORD 20

main() /* tel C keywoorden, pointer versie */
{
   int t;
   char word[MAXWORD];
   struct key *binary(), *p;

#ifdef PROLINE

   proline_init();

#endif

   while ((t = getword(word, MAXWORD)) != EOF)
      if (t == LETTER)
         if ((p = binary(word, keytabel, NKEYS)) != NULL)
            p->keyaantal++;
      for (p = keytabel; p < (keytabel + NKEYS); p++)
         if (p->keyaantal > 0)
            printf("%4d %s\n", p->keyaantal, p->keywoord);
}
```

4.7 Structures

```
struct key *binary(word, tab, n) /* vindtt woord
                                    in tab[0] ... tab[n] */
char *word;
struct key tab[];
int n;
{
   int cond;

#ifndef PROLINE

   struct key *low = &tab[0];
   struct key *high = &tab[n-1];
   struct key *mid;

#else

   struct key *low;
   struct key *high;
   struct key *mid;

   low = &tab[0];
   high = &tab[n - 1];

#endif

   while (low <= high) {
      mid = low + (high - low) / 2;
      if ((cond = strcmp(word, mid->keywoord)) < 0)
         high = mid - 1;
      else if (cond > 0)
         low = mid + 1;
      else
         return(mid);
   }
   return(NULL);
}
```

Een aantal dingen vallen hier op. Ten eerste, de declaratie van binary. Hier wordt aangegeven dat de functie een pointer naar de structure key retourneert, in plaats van een integer. Dit is zowel in main als in binary gedeclareerd. Als binary het woord vindt, dan retourneert het een pointer daar naar toe. Als het woord niet vindt, dan retourneert het NULL.

Ten tweede, alle elementen van keytabel worden gebruikt door gebruik te maken van pointers. Dit heeft tot gevolg dat er een belangrijke wijziging in binary plaats vindt: de berekening van het middelste element kan niet langer, door middel van de volgende expressie, plaats vinden.

```
mid= (low+ high) / 2
```

Dit omdat de optelling van twee pointers nooit een zinnig resultaat oplevert (zelfs als het door 2 wordt gedeeld niet); het is in feite een illegale operatie. Dit moet gewijzigd worden in

```
mid= low+ (high - low) / 2
```

wat mid naar het element, halverwege low en high, laat wijzen.

In main staat het volgende statement geschreven:

```
for (p= keytabel;
p < keytabel+ NKEYS; p++)
```

Als p een pointer naar een structure is, dan werkt dit statement uitstekend. Er wordt namelijk rekening gehouden met de grootte van het object, zodat p++ p op correcte wijze ophoogt om het volgende element uit de array van structures te verkrijgen. Maar verwacht niet dat de grootte van de structure gelijk is aan de som van de grootte van de afzonderlijke elementen: door alignment (delen van een structure laten beginnen op bijvoorbeeld een even geheugen adres) van de verschillende members kunnen er 'gaten' binnen de structure optreden.

8/4.7.5
Zich zelf refererende structures

Veronderstel dat het bovenstaande probleem gegeneraliseerd dient te worden. Niet een selecte verzameling van woorden uit de invoer moet geteld worden, maar van alle woorden, die in de invoer voorkomen, moet worden bij gehouden hoe vaak deze voorkomen. Omdat de lijst van woorden in het begin niet bekend is, kan geen gebruik worden gemaakt van een normale binaire zoekmethode. Tevens kan ook niet lineair, in de volgorde van binnenkomst, worden gezocht, om te testen of dit woord al eerder is voorgekomen. Het programma zou dan bijna oneindig lang gaan duren (om precies te zijn zal de verwachte executie tijd kwadratisch met het aantal invoer woorden stijgen). Hoe moet een programma dan worden georganiseerd om toch efficiënt te werken met een onbekende lijst van woorden?

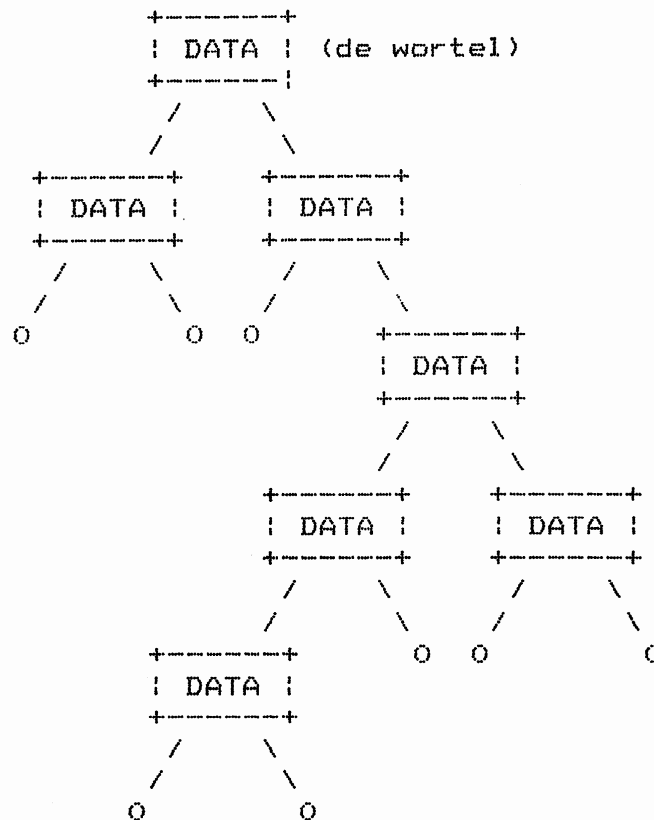
Een oplossing is om de verzameling van woorden altijd gesorteerd te houden, door elk woord in de juiste positie op te slaan, zodra het binnenkomt. Dit moet niet gebeuren door die woorden steeds binnen een lineaire array op te schuiven, omdat dat te lang zal gaan duren. In plaats daarvan zal gebruik gemaakt worden van een binaire boom.

Een binaire boom bestaat uit een aantal knooppunten die, via een bepaalde regel, met elkaar verbonden zijn. Een knooppunt uit een binaire boom ziet er bijvoorbeeld als volgt uit:

een pointer naar de tekst van het woord
een tel waarde, die het aantal malen van optreden bijhoud
een pointer naar een linker-kind knooppunt
een pointer naar een rechter-kind knooppunt

4.7 Structures

Een knooppunt mag niet meer dan twee kinderen hebben, het mag er wel nul of één hebben. Elk knooppunt kan dus kinderen hebben, ook heeft elk knooppunt een ouder, dit is er altijd één. Uitzondering is de wortel, deze heeft geen ouder, alleen eventuele kinderen. Een binaire boom kan er dan als volgt uitzien.



De knooppunten zijn zo georganiseerd dat elke linker subboom van een knooppunt alleen woorden bevat, die kleiner zijn dan het woord in het knooppunt; de rechter subboom bevat alleen woorden, die groter zijn. Om uit te zoeken of een nieuw woord al reeds in de boom is opgenomen, begint de routine tree het nieuwe woord te vergelijken met het woord dat is opgeslagen in de wortel van de boom. Als het resultaat hiervan positief is, dan is het antwoord bekend. Is het nieuwe woord kleiner dan het woord uit de boom, dan wordt het linker kind onderzocht, anders wordt het rechter kind onderzocht. Als er geen kind in de gewenste richting bestaat, dan was het nieuwe woord niet in de boom opgenomen en is tevens de juiste plaats voor het nieuw toe te voegen kind gevonden. Het zoekproces is recursief, omdat het zoekt vanaf een knooppunt naar één van de kinderen. Vanwege deze recursiviteit zien de toevoeg- en afdrukroutines er logisch uit.

De knooppunts beschrijving van het knooppunt, die hier gebruikt gaat worden ziet er als volgt uit:

4.7 Structures

```
struct tnode {          /* basis knooppunt */
   char *word;          /* pointer naar de tekst */
   int  count;          /* aantal malen van optreden */
   struct tnode *left;     /* linker kind */
   struct tnode *right;    /* rechter kind */
};
```

Deze 'recursieve' declaratie van een knooppunt mag er dan gewaagd uitzien, maar in feite is deze declaratie helemaal correct. Het is illegaal om binnen een structure een member te hebben, die als type de structure zelf heeft (de structure zou zich zelf 'opblazen'), maar

struct tnode *left;

declareert left als zijnde een pointer naar een knooppunt en niet het knooppunt zelf.

De code voor het gehele programma is erg klein, gegeven de ondersteunende routines die als eerder in dit boek voorkomen, te weten: getword, om een invoer woord op te halen en alloc, om geheugen ruimte van de heap af te geven.

De main routine leest woorden met behulp van getword en plaatst ze in de boom door middel van tree.

```
#define MAXWORD 20

main()   /* tel het aantal malen dat een woord voorkomt */
{
   struct tnode *root, *tree();
   char word[MAXWORD];
   int t;

   root = NULL;
   while ((t = getword(word, MAXWORD)) != EOF)
      if (t == LETTER)
         root = tree(root, word);
   treeprint(root);
}
```

De functie tree is recht toe recht aan. Een woord wordt door main doorgegeven aan de top van de boom. Elke keer wordt dit woord vergeleken met een woord dat reeds in de boom is opgeslagen en wordt het naar beneden gestuurd, naar òf de linker òf de rechterkant van de boom. Dit gebeurt door middel van een recursieve aanroep van tree. Uiteindelijk komt het woord overeen met een woord wat reeds in de boom stond, of de NULL pointer wordt gevonden, wat aangeeft dat een knooppunt gecreëerd moet worden en aan de boom moet worden toegevoegd. Als

4.7 Structures

een nieuw knooppunt is gecreëerd, retourneert tree een pointer naar dat knooppunt, wat verbonden is aan zijn ouderknooppunt.

```
struct tnode *tree(p, w) /* installeer w aan of onder p */
struct tnode *p;
char *w;
{
    struct tnode *talloc();
    char *strsave();
    int cond;

    if (p == NULL) {    /* een nieuw woord is binnen gekomen */
        p = talloc();   /* creer een nieuw knooppunt */
        p->word = strsave(w);
        p->count = 1;
        p->left = NULL;
        p->right = NULL;
    } else if ((cond = strcmp(w, p->word)) == 0)
        p->count++;     /* woord bestond reeds */
    else if (cond < 0) /* ga lager in de linker subboom */
        p->left = tree(p->left, w);
    else                /* ga lager in de rechter subboom */
        p->right = tree(p->right, w);
    return(p);
}
```

De geheugenruimte voor een nieuw knooppunt wordt door de routine talloc verzorgd, wat een aanpassing van de alloc routine is, welke al eerder beschreven is. Het retourneert een pointer naar vrije ruimte, dat geschikt is om een knooppunt in op te slaan.
Het nieuwe woord wordt door strsave naar een 'verborgen' plaats gekopieerd, de count wordt geïnitialiseerd en de twee kinderen worden NULL gemaakt. Dit gedeelte van de code wordt alleen dan geëxecuteerd als een nieuw knooppunt wordt toegevoegd. Aan error checking wordt hier niet gedaan, de return waarden van strsave en talloc kunnen duiden op fouten.

De functie treeprint drukt een boom van links naar rechts af, het drukt eerst de linker subboom, dan het woord zelf en daarna de rechter subboom af. Dit is een van de eenvoudigste recursieve routines die maar te bedenken is.

N.B. Als de boom 'uit balance' raakt, omdat de woorden niet in random volgorde binnen komen, kan de executie tijd van het programma snel toenemen. In het ergste geval, als de woorden reeds gesorteerd zijn, dan voert dit programma een 'dure' simulatie van een lineaire zoekactie uit. Er zijn generalisaties van binaire bomen, B-bomen, B+-bomen, AVL-bomen, etc. die geen last hebben van deze slechtste situa-

4.7 Structures

tie, hier worden ze echter niet beschreven.

```
treeprint(p)   /* print de boom recursief */
struct tnode *p;
{
    if (p != NULL) {
        treeprint(p->left);
        printf("%4d %s\n", p->count, p->word);
        treeprint(p->right);
    }
}
```

Voordat er wordt overgegaan op een nieuw onderwerp, is het de moeite waard om even stil te blijven staan bij een probleem dat aan storage (geheugenruimte) allocators is verbonden. Het is duidelijk te prefereren dat er maar één storage allocator in een programma is, zelfs als het verschillende objecten alloceert. Maar als een allocator gebruikt wordt om bijvoorbeeld pointers naar char's en pointers naar struct tnode's te alloceren, dan rijzen er twee vragen. Ten eerste, hoe wordt er omgegaan met de restricties die de meeste grotere computers hebben ten aanzien van de alignment van de objecten (bijvoorbeeld: integers dienen vaak op even adressen te beginnen)? Ten tweede, hoe wordt er omgegaan als objecten van verschillende types moeten worden gealloceerd?

De alignment restricties kunnen eenvoudig worden opgelost, ten koste van wat geheugenruimte verspilling, door er voor te zorgen dat elk gealloceerd object op een gealigneerde geheugenplaats begint. Het enige verlies dat optreedt, treedt alleen dan op als een character string wordt gealloceerd van oneven lengte. Dit heeft tot gevolg dat alloc niet portabel (overdraagbaar) geschreven kan worden, maar het gebruik van alloc is wel overdraagbaar.

De vraag van de typedeclaratie van alloc is een probleem voor elke taal, die typechecking serieus neemt. In C is het de beste procedure om alloc een pointer naar een char te laten retourneren, dan deze pointer expliciet te co-ercen naar een pointer van het verlangende type, door middel van de cast operator. Bijvoorbeeld als p als volgt is gedeclareerd.

```
char *p;
```

dan converteert

```
(struct tnode *) p
```

het naar een tnode pointer binnen een expressie. Dus talloc ziet er als volgt uit.

4.7 Structures

```
struct tnode *talloc()
{
   char *malloc();

   return((struct tnode *) malloc(sizeof(struct tnode)));
}
```

8/4.7.6
Opzoeken in tabellen

In deze paragraaf zullen tabel-zoekroutines geschreven worden, die nog een aantal aspecten van de structures zullen illustreren. De code lijkt zeer veel op de code, die kan worden teruggevonden in de macro preprocessor van bijvoorbeeld de C-compiler. Neem bijvoorbeeld het C # define statement. Wanneer tijdens de compilatie een regel als

define JA 1

wordt tegengekomen, dan wordt de naam JA en de vervangende tekst 1, in een tabel opgeslagen. Later, wanneer de naam JA in een statement voorkomt, zoals

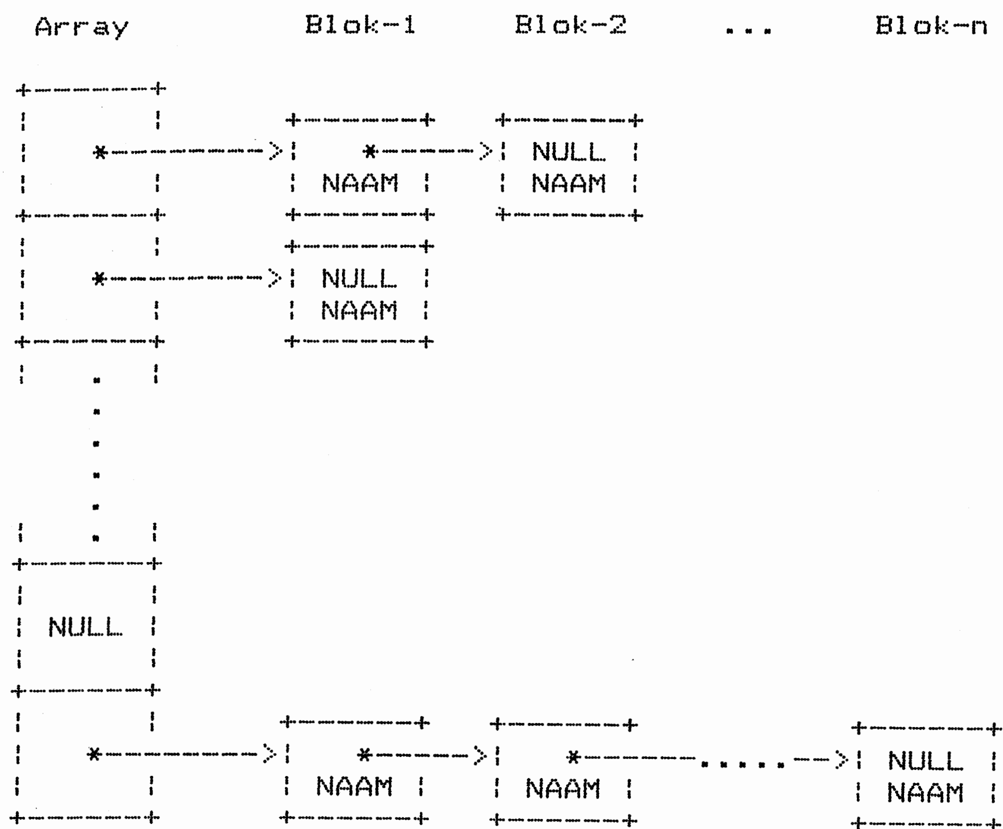
switch = JA;

dan moet deze naam worden vervangen door 1.

Er zijn twee belangrijk routines die de namen met hun vervangende tekst manipuleren. De functie install(s, t) noteert de naam s en de vervangende tekst t in een tabel, s en t zijn character strings. lookup(s) zoekt naar S in de tabel en retourneert een pointer naar de plaats waar deze gevonden is, of NULL als het niet in de tabel te vinden was.

Het algorithme maakt gebruik van een hash functie, deze hash functie converteert de inkomende naam naar een kleine positieve integer, die als index wordt gebruikt voor een array van pointers. Een array element wijst naar het begin van een ketting van blokken die de namen beschrijven welke die hash waarde hebben. Als de pointer NULL is dan zijn er geen namen die die hash waarde hebben. De structuur ziet er dan bijvoorbeeld als volgt uit.

4.7 Structures



Een blok in de ketting is een structure van pointers naar de naam, de vervangende tekst en het volgende blok.

De NULLwaarde in de volgende-blok member geeft het einde van ketting aan.

```
struct nlist { /* basis ketting onderdeel */
    char *name;
    char *def;
    struct nlist *next; /* pointer naar volgende schakel */
};
```

De pointer array bestaat uit niet meer dan deze definitie:

```
#define HASHSIZE 100

static struct nlist *hashtab[HASHSIZE]; /* pointer tabel */
```

4.7 Structures

De hash functie, die gebruikt wordt in zowel lookup als in install, telt eenvoudig weg de character waarden uit de string bij elkaar op en retourneert de modulo array grootte van deze som. Dit is een van de methoden om te hashen, er bestaan betere hash methoden. Enkele hash methoden zullen hier kort beschreven worden, alleen de reeds beschreven methode van hashing wordt later geïmplementeerd, alhoewel dit dus niet de beste methode is (deze methode is hier effectief genoeg en springt er uit door zijn eenvoud).

De priemgetaldeling gaat er van uit dat een getal (zoals boven de som) op een bepaalde manier is verkregen. Door een modulo deling van een priemgetal dat, qua grootte, in de buurt van de array grootte ligt, wordt de index in de array gevonden. Door namelijk een iets andere manier van hashing te nemen kan de spreiding over de array gelijkmatiger worden. Hoe gelijkmatiger die spreiding is, des te beter wordt de responsie tijd van het opzoeken van een element uit de gehashte array.

Bij hashing door vermeningvuldiging wordt er uitgegaan van een getal. Tevens wordt er een woordlengte W gedefinieerd (bijvoorbeeld 1 000 000). Is het getal groter dan de woordlengte (bijvoorbeeld 56874487620), dan wordt dit getal opgevouwen:

```
487620
 56874
----------+
544494
```

en 544494 wordt nu als sleutel S beschouwd. Deze sleutel wordt met een konstante K vermenigvuldigd. Dit geeft een dubbellengte product, waarvan de kop wordt weggelaten. De volgende grootheid wordt dan bepaald:

Q= (K * S) %W

De index in de array wordt dan bepaald door:

Index= Array-grootte * Q / W

Er zijn twee voordelen aan deze methode. Ten eerste blijven tot aan de laatste vermenigvuldiging alle karakteristieken van de sleutel S nog aanwezig, mits de konstante K zo gekozen wordt dat K en W onderling ondeelbaar zijn, dus als de grootste-gemene-deler van K en W, 1 is. Ten tweede worden door de vermenigvuldiging de opeenvolgende sleutel-waarden sterk verspreid over het index gebied, terwijl de eerder beschreven methoden opeenvolgende sleutels opeen laten volgen binnen de gehashte array. Aansluitende sleutels komen in de praktijk nog al eens voor, doordat het laatste karakter vaak een soort volgnummer is, bijvoorbeeld

aanvulling-7, aanvulling-8, aanvulling-9
of groep1, groep2, groep3, groep4

De eerst beschreven hash functie, die het getal alleen deelt door de grootte van de array, ziet er als volgt uit.

De hash functie produceert een index voor de array hashtab. Als de string ergens gevonden kan worden, dan zit die string in de ketting die met die index begint. De zoekactie wordt door lookup uitgevoerd.
Als lookup de string vindt, retourneert het een pointer naar het blok; anders retourneert het NULL.

4.7 Structures

```
hash(s)   /* vorm de hash waarde uit de string s */
char *s;
{
   int hashval;

   for (hashval = 0; *s != '\0'; )
      hashval += *s++;
   return(hashval % HASHSIZE);
}
```

```
struct nlist *lookup(s) /* zoek s in de hashtabel */
char *s;
{
   struct nlist *np;

   for (np = hashtab[hash(s)]; np != NULL; np = np->next)
      if (strcmp(s, np->name) == 0)
         return(np);      /* gevonden */
   return(NULL);          /* niet gevonden */
}
```

De functie install gebruikt lookup om te bepalen of de naam reeds aanwezig is. Als dat zo is dan moet de nieuwe definitie de oude vervangen. Anders moet een geheel nieuw blok worden gecreëerd. Install retourneert NULL als er om één of andere reden geen ruimte is voor een nieuw object.

De gebruikte routines zoals strsave en free (geheugen allocatie functies) zijn in vorige paragrafen beschreven.

8/4.7.7 Bitvelden

De Proline C-compiler ondersteunt de bitvelden faciliteit uit C niet, om desalniettemin volledig te wezen, worden de bit velden hier toch beschreven.

Bitvelden worden gebruikt als er gebrek aan geheugenruimte is, het kan dan noodzakelijk zijn om verschillende objecten bij elkaar, in één machine woord te plaatsen. Maar ze worden ook gebruikt voor het aansturen van hardware registers, waar bijna altijd de registers in bits zijn onderverdeeld en waarbij de afzonderlijke bits allen een eigen betekenis hebben.

Bitvelden worden op een soortgelijke wijze als structures gedefinieerd. Neem bijvoorbeeld een hardware register uit de Complex Interface Adapter. Van de CIA

4.7 Structures

```
struct nlist *install(name, def) /* plaats name en def
                                        in de hash tabel */

char *name, *def;
{
   struct nlist *np, *lookup();
   char *strsave(), *malloc();
   int  hashval;

   if ((np = lookup(name)) == NULL) { /* niet gevonden */
      np = (struct nlist *) malloc(sizeof(struct nlist));
      if (np == NULL)
         return(NULL);
      if ((np->name = strsave(name)) == NULL)
         return(NULL);
      hashval = hash(np->name);
      np->next = hashtab[hashval];
      hashtab[hashval] = np;
   } else                          /* wel gevonden */
      free(np->def);               /* free vorige definitie */
   if ((np->def = strsave(def)) == NULL)
      return(NULL);
   return(np);
}
```

is het Peripheral Control Register het meest interresante register, met behulp van bitvelden zou dit register als volgt (in de software) gedefinieerd kunnen worden:

```
struct{
   unsigned CA1-int-control  : 1;
   unsigned CA2-control      : 3;
   unsigned CB1-int-control  : 1;
   unsigned CB2-control      : 3;
}peri-control;
```

Het aantal bits dat voor zo'n veld wordt gebruikt, wordt achter de dubbele punt gedefinieerd. De velden zijn als unsigned gedeclareerd, om aan te geven dat het hier om tekenloze eenheden gaat.
De individuele velden worden gerefereerd door peri-control.CA1-int-control, peri-control.CA2-control, etc. Deze velden gedragen zich als kleine tekenloze integers en arthmetische operaties kunnen hier op worden uitgevoerd.

8/4.7.8
Unions

Een union is een variabele die objecten van verschillende types en groottes mag bevatten. Op een bepaald moment kan een union dus een integer bevatten en op het volgende moment kan deze waarde zijn vervangen door een float.

Ter illustratie wordt hier een symbooltabel uit een compiler gebruikt. Veronderstel dat er alleen constanten zoals integers, floats of character pointers mogen zijn. Dan moet de waarde van elke constante in een variabele van het juiste type worden opgeslagen. Het is het meest efficiënt om de ruimte die zo'n constante nodig heeft ook als zodanig te reserveren in een tabel, onafhankelijk van waar het wordt opgeslagen en wat voor type het is. Dit is het idee achter de union, om voor een variabele te zorgen, die legaal elke waarde van een aantal te specificeren types kan opslaan. De syntax is gelijk aan die van de structures.

4.7 Structures

```
union u_tag{
    int     ival
    float   fval
    char    *pval;
}uval;
```

De variabele uval zal groot genoeg zijn om de grootste van de drie types in op te slaan, onafhankelijk van de machine waar dit op wordt gecompileerd. Elke waarde van deze drie types mag aan uval worden toegekend en dan later in een expressie worden gebruikt, zolang het gebruik maar consistent is: het type wat wordt opgehaald moet hetzelfde zijn als het type wat er als laatste is ingestopt.
Het is de verantwoordelijkheid van de programmeur om bij te houden wat het huidige type is, dat nu in de union is opgeslagen.

Syntactisch worden members als volgt gebruikt

```
union-name.member
```

of

```
union-pointer->member
```

op dezelfde manier als dit voor structures gebeurt. Als de variabele utype wordt gebruikt om bij te houden wat het type dat het laatst in uval is opgeslagen, dan kan een stukje code er zo uitzien.

```
if (utype == INT)
   printf("%d\n", uval.ival);
else if (utype == FLOAT)
   printf("%f\n", uval.fval);
else if (utype == STRING)
   printf("%s\n", uval.pval);
else
   printf("verkeerd type %d in utype\n", utype);
```

Unions kunnen voorkomen in structures en arrays en vice versa. De notatie voor de access van de members van een union in een structure is gelijk aan die van de geneste structures. Bijvoorbeeld, in de structure array gedefinieerd door:

```
struct {
   char *name;
   int  flags;
   int  utype;
   union {
      int  ival;
      float fval;
      char *pval;
   } uval;
} symtab[nSYM];
```

wordt aan de variabele ival gerefereerd door

```
symtab[i].uval.ival
```

en het eerste character van de string pval door

```
*symtab[i].uval.pval
```

8/4.7.9 Typedef

In C kunnen ook nieuwe types gedefinieerd worden, met behulp van de laatst nog niet besproken C faciliteit: typedef. Bijvoorbeeld de declaratie

```
typedef int LENGTE;
```

maakt de naam LENGTE synoniem aan int. Het 'type' LENGTE kan in declaraties, cast, etc. worden gebruikt, op exact dezelfde manier als int gebruikt kan worden.

```
LENGTE len, maxlen;
LENGTE *lengte[];
```

Het zelfde is de declaratie

```
typedef char *STRING
```

deze declaratie maakt STRING synoniem voor char * of voor een character pointer, die dan gebruikt mag worden in declaraties zoals

```
STRING p, lineptr[LINES],
alloc();
```

Merk op dat het type dat wordt gedeclareerd zich bevindt op de plaats van de variabele naam, niet direct achter het woord typedef. Syntactisch is typedef gelijk aan de storage groepen extern, static, etc.
Een iets complexer voorbeeld, typedef's kunnen gebruikt worden om boom van knooppunten te maken.

```
typedef struct tnode{        /*basis knooppunt */
char *word;                  /* wijst naar de tekst */
int count;                   /* aantal malen van optreden */
struct tnode *left;          /* linker kind */
struct tnode *right;         /* rechter kind */
} TREENODE, *TREEPTR;
```

Dit creëert twee nieuwe type-keywoorden, die TREENODE (EEN structure) en TREEPTR (een pointer naar een structure) heten.
Dan kan de functie talloc er als volgt uit gaan zien.

```
TREEPR talloc();
{
   char *alloc();

   return((TREEPTR) alloc(TREENODE)));
}
```

8/4.8

Een groot voorbeeld

Tot nu toe zijn de voorbeeld programma's steeds klein geweest, om de behandelde stof te illustreren is het nooit nodig geweest om deze voorbeelden erg groot te maken. Op dit moment is er al veel van C behandeld en is dit het goede moment om een wat groter programma aan dit boek toe te voegen.

Het frame van dit programma is hetzelfde als de al eerder besproken Poolse desk calculator. Deze desk calculator rekent echter niet met integers of floating point getallen, maar met bignums.

Bignums zijn getallen, die eigenlijk grote integers representeren. De grootste integer die de Commodore 64 kent is 32767 of een factor twee groter, de grootste unsigned integer: 65535. Met bignums is het mogelijk om alle gehele getallen te representeren. Dus 65536 kan als bignum geschreven worden, maar ook -240564041062 kan als bignum geschreven worden.

Met dit programma is het mogelijk om de getallen zoals boven beschreven op te tellen, af te trekken, te vermenigvuldigen en ook te delen. Bij gebruik van het programma zal het opvallen, dat de tijd die nodig is om deze operaties uit te voeren ook in die volgorde oploopt. Het delen van bignums neemt zeer veel tijd in beslag, zodra deze enige grootte van betekenis krijgen.

De routines voor optellen, aftrekken, vermenigvuldigen en delen zijn geen standaard routines en moeten voor dit programma geschreven worden. Deze routines nemen ongeveer de helft van de ruimte van de listings in beslag. Al de routines zijn gestructureerd en duidelijk gezet, zodat voor diegene die ze voor het eerst ziet, het niet al te moeilijk zal zijn om zichzelf wegwijs te maken in het totale programma.

Voordat aan de beschrijving van al de verschillende routines kan worden begonnen, moet eerst de representatie van de bignums uiteen gezet worden. De representatie die hier gebruikt wordt bestaat uit drie delen: het teken, de lengte en de waarde. Het teken duidt het teken van de bignum aan en gebruikt één byte. De lengte geeft het aantal bytes van de waarde aan en gebruikt twee bytes zodat de maximale lengte van het waarde gedeelte, 65536 bytes is. Het waarde gedeelte bevat de eigenlijke waarde van de bignum; iedere byte heeft een waarde tussen de 0 en de 99. Het laagstwaardige byte staat vooraan en wordt aangeduid door Bignum-Bottom. Het hoogstwaardige byte staat achteraan en wordt aangeduid door Bignum-Top. Een bignum heeft dan de volgende vorm:

4.8 Een groot voorbeeld

```
+----+----+----+-----------------------------------------+
|SIGN|HLEN|LLEN|        VALUE                            |
+----+----+----+--^--------------------------------------^--+
                  |                                       |
              Bignum-Bottom                           Bignum-Top
```

Om het programma leesbaar te maken zijn er een aantal macro's gedefinieerd. Deze staan in de listing hiernaast en naar de file die hier mee correspondeert zal voortaan met de naam bignum.h worden verwezen. Dit is van belang omdat de file bignum.h bij compilatie van elke functie 'ge-*inlcude*' moet worden.

De meeste macro's spreken voor zich, alleen de laatste zijn misschien wat cryptisch, al geeft de naam precies aan wat ze moeten doen. Later zal hier nog op worden terug gekomen.

HET HOOFDPROGRAMMA

De main routine volgt hieronder. Deze routine is qua vorm identiek aan de main routine van de al eerder beschreven Poolse desk calculator, maar inwendig is het toch anders. Ten eerste worden de rekenroutines allemaal gedeclareerd, deze routines retourneren stuk voor stuk een pointer naar het resultaat, de bignum. Zo'n bignum is weer opgebouwd uit bigdigit's, dus elke functie retourneert een pointer naar een bigdigit. Ten tweede wordt er gebruik gemaakt van de functie free. Dit heeft met de storage-allocatie van de bignums te maken. De nieuwe bignums worden op de heap gealloceerd en zodra ze niet meer gebruikt worden (dus na een operatie) kunnen ze weer beschikbaar worden gesteld aan de heap, dit gebeurt dan met de free functie. Ten derde is het '=' commando vervangen door het 'p' van lings print commando. Het '=' commando is een vergelijkingsoperatie geworden die de top van de stack met het element wat daar onder zit vergelijkt.

De getop functie is hier iets gewijzigd. getop retourneert slechts een waarde en geeft niet via een pointer-argument de waarde van het eventuele bignum door. Er zijn namelijk C-compilers in omloop die dit niet op correcte wijze kunnen uitvoeren en aangezien dit ook anders op te lossen valt, is daar hiervoor gekozen.

De functies die voor de stack-manipulatie zorg dragen zijn bijna exact gelijk aan die van de reeds beschreven desk calculator. De clear functie geeft alle ruimte vrij die de bignums, die op de stack staan, in gebruik nemen.

DE INVOER EN UITVOER

De twee volgende functies zorgen voor de in- en uitvoer van de gegevens. De eerste functie print, voert een bignum uit naar standaard output. De waarde van de afzonderlijke bigdigit's kan zo worden geïnterpreteerd als integer-waarde, met dien verstande dat als deze waarde kleiner dan 10 is er een '0' voor moet worden uitgevoerd. Deze regel gaat alleen niet op voor het eerste bigdigit, want als dit kleiner is dan 10 moet de '0' niet worden afgedrukt.

4.8 Een groot voorbeeld

```
#define POSITIVE 1
#define NEGATIVE 0

/*
 * +----+---+---+---------------------------+
 * |SIGN|LEN    |          VALUE            |
 * +----+---+---+^-------------------------^+
                 |                         |
           Bignum-Bottum              Bignum-Top
 */
#define bigdigit char

#define SIGN(Bignum)               (Bignum[0])
#define READ_LEN(Bignum)           (Bignum[1] << 8 | Bignum[2])
#define WRITE_HLEN(Bignum)         (Bignum[1])
#define WRITE_LLEN(Bignum)         (Bignum[2])
#define ALLOCLEN(Bignum)           (READ_LEN(Bignum) + 3)
#define Bignum_Bottom(Bignum)      (&(Bignum)[3])
#define Bignum_Top(Bignum)         (&(Bignum)[READ_LEN(Bignum)] + 2)
#define HARDLENGTH(Length)         (Length + 3)
#define HEADER(Bignum)             (Bignum - 3)

#define POS_BIGNUM(Bignum)         (SIGN(Bignum) == POSITIVE)
#define NEG_BIGNUM(Bignum)         (SIGN(Bignum) == NEGATIVE)
#define ZERO_BIGNUM(Bignum)        (READ_LEN(Bignum) == 0)

#define EQUAL      0
#define ONE_BIGGER 1
#define TWO_BIGGER 2

#define Categorize_Sign(ARG1, ARG2) ((SIGN(ARG1) << 1) | SIGN(ARG2))
#define BOTH_NEGATIVE 0
#define ARG1_NEGATIVE 1
#define ARG2_NEGATIVE 2
#define BOTH_POSITIVE 3

#define RADIX 100
#define MAX_DIGIT_SIZE (RADIX - 1)
#define Get_Carry(lw)      ((lw > MAX_DIGIT_SIZE) ? 1 : 0)
#define Get_Digit(lw)      ((lw > MAX_DIGIT_SIZE) ? lw - RADIX : lw)
#define Ge_mul_Carry(lw)   (lw / 100)
#define Ge_mul_Digit(lw)   (lw % 100)
```

4.8 Een groot voorbeeld

```
#define NUMMER   '0' /* signaal dat een nummer is gevonden */

main()      /* omgekeerde Poolse desk calculator */
{
    int type, comp;
    bigdigit *s, *op1, *op2;
    bigdigit *pop(), *push();
    bigdigit *plus_signed_bignum();
    bigdigit *minus_signed_bignum();
    bigdigit *mul_signed_bignum();
    bigdigit *div_signed_bignum();

    while ((type = getop()) != EOF)
        switch (type) {
        case NUMMER:
            s = getnum();
            push(s);
            break;
        case '+':
            op1 = pop();
            op2 = pop();
            push(plus_signed_bignum(op1, op2));
            free(op1);
            free(op2);
            break;
        case '*':
            op1 = pop();
            op2 = pop();
            push(mul_signed_bignum(op1, op2));
            free(op1);
            free(op2);
            break;
        case '-':
            op2 = pop();
            op1 = pop();
            push(minus_signed_bignum(op1, op2));
            free(op1);
            free(op2);
            break;
        case '/':
            op2 = pop();
            if (!ZERO_BIGNUM(op2)) {
                op1 = pop();
                push(div_signed_bignum(op1, op2));
                free(op1);
                free(op2);
            }
            else {
                printf("Delen door nul is niet toegestaan\n");
                free(op2);
```

4.8 Een groot voorbeeld

```
            }
            break;
        case 'p':
            op2 = pop();
            print(op2);
            push(op2);
            printf("\n");
            break;
        case 'c':
            clear();
            break;

        case '=':
            op1 = pop();
            op2 = pop();
            comp = big_compare(op1, op2);
            if (comp == ONE_BIGGER)
                printf("De top is groter\n");
            else if(comp == TWO_BIGGER)
                printf("De top is kleiner\n");
            else
                printf("Ze zijn gelijk\n");
            push(op2);
            push(op1);
            break;
        default:
            printf("Onbekend commando %c\n", type);
            break;
        }
}
```

Page 1

```
getop()             /* lever volgende operator */
{
    int c;

    while ((c = getch()) == ' ' || c == '\n')
        ;                       /* sla spaties en newlines over */
    if (!isdigit(c))
        return(c);                  /* geen nummer dus commando */
    ungetch(c);
    return(NUMMER);
}

#define BUFFERGROOTTE   100
```

4.8 Een groot voorbeeld

```
char buf[BUFFERGROOTTE]; /* buffer voor de ungetch */
int  bufp = 0;           /* volgende vrije positie */

getch() /* pak een teken, mogelijk terug gestopt */
{
   return((bufp > 0) ? buf[--bufp] : getchar());
}

ungetch(c)        /* stop teken terug in de invoer */
int c;
{
   if (bufp > BUFFERGROOTTE)
      printf("Ungetch: te veel tekens\n");
   else
      buf[bufp++] = c;
}
```

```
#define MAXDIEPTE 100 /* stack diepte */

int sp = 0;     /* stack pointer */
bigdigit *val[MAXDIEPTE]; /* stack */

bigdigit *push(f)   /* push f op de stack */
bigdigit *f;
{
   if (sp < MAXDIEPTE)
      return(val[sp++] = f);
   else {
      printf("Error: stack is vol\n");
      clear();
      return return_bignum_zero();
   }
}
```

4.8 Een groot voorbeeld

```
bigdigit *pop()       /* pop de top van de stack */
{
    if (sp > 0) {
        return(val[--sp]);
    }
    else {
        printf("Error: stack leeg\n");
        clear();
        return return_bignum_zero();
    }
}

clear()      /* leeg de stack */
{
    while (sp > 0)
        free(val[--sp]);
}
```

```
print(arg)
bigdigit *arg;
{
    bigdigit *top;
    int digit;

    if (SIGN(arg) == NEGATIVE) printf("-");
    if (ZERO_BIGNUM(arg)) {
        printf("0");
        return;
    }
    top = Bignum_Top(arg);
    arg = Bignum_Bottom(arg);

    /* print digit pairs */
    digit = *top--;
    printf("%d", digit);
    while (top >= arg) {
        digit = *top--;
        if (digit < 10)
            printf("0%d", digit);
        else
            printf("%d", digit);
    }
}
```

4.8 Een groot voorbeeld

De tweede I/O routine, getnum, zorgt ervoor dat bignums kunnen worden ingelezen. De functie retourneert een pointer naar het ingelezen bignum. In deze functie wordt gebruik gemaakt van een nieuwe operator, de cast operator.

result = (bigdigit*)
malloc(HARDLENGTH(length));

De cast operator (bigdigit *), zorgt ervoor dat het type wat de functie malloc retourneert (char), wordt omgezet in een bigdigit pointer. De functie van de cast operator is het omzetten van het ene type naar het andere type. Op de cast operator zal later nog worden ingegaan.

```
bigdigit *getnum()
{
     char *malloc();
     bigdigit temp_digit[100];
     bigdigit *copy, *result;
     int sign;
     int digit, i = 0, length = 0;
     char c, s[100];

     /* lees getal */
     c = getch();
     if (isdigit(c) || c == '-' || c == '+')
         s[i++] = c;

     else
         printf("Error: geen bignum\n");
     while (isdigit(c = getch())) s[i++] = c;
     s[i] = '\0';

     i = 0;
     sign = POSITIVE;
     if (s[0] == '+') s[i++] = '0';
     if (s[0] == '-') {
         sign = NEGATIVE;
         s[i++] = '0';
     }

     /* ga naar het laatste cijfer */
     while ((c = s[i++]) != '\0') ;
```

4.8 Een groot voorbeeld

```
        i--;
        while (i != 0) {
            c = s[--i];
            if (isdigit(c)) {
                digit = c - '0';
            }
            if (i != 0) {
                c = s[--i];
                if (isdigit(c))
                    digit = digit + (c - '0') * 10;
            }
            temp_digit[length++] = digit;
        }

        /* verwijder voorafgaande nullen */
        while (temp_digit[length-1] == 0 && length > 0)
            length--;

        result = (bigdigit *)malloc(HARDLENGTH(length));
        SIGN(result) = sign;
        WRITE_HLEN(result) = length >> 8;
        WRITE_LLEN(result) = length;

        copy = Bignum_Bottom(result);

        for (i = 0 ; i < length ; i++) *copy++ = temp_digit[i];
        return (result);
}
```

DE REKEN ROUTINES

De volgende acht functies verrichten het rekenwerk. Voor elke operatie zijn er twee routines: de eerste voert een selectie uit en de tweede doet het eigenlijke rekenwerk. Eerst wordt de optelling, daarna de aftrekking, de vermenigvuldiging en uiteindelijk de deling behandeld.

Optellen

De eerste functie bepaalt het teken van beide bignums en aan de hand daarvan wordt een ongesigneerde optel- dan wel aftrek-routine uitgevoerd. Aan deze routine wordt naast de twee bignums ook het teken van het resultaat mee gegeven.

De werkelijke optelling wordt door plus_unsigned_bignum() uitgevoerd. Een groot gedeelte van deze functie is de voorbewerking van allerlei variabelen. In de volgende loop wordt de optelling uitgevoerd.

4.8 Een groot voorbeeld

```
#include <bignum.h>

/* Optellen */

bigdigit *plus_signed_bignum(arg1, arg2)
bigdigit *arg1, *arg2;
{
    bigdigit *plus_unsigned_bignum();

    if (ZERO_BIGNUM(arg1) && ZERO_BIGNUM(arg2))
        return return_bignum_zero();
    switch(Categorize_Sign(arg1, arg2)) {
        case BOTH_POSITIVE : return(plus_unsigned_bignum(arg1, arg2,
POSITIVE));
        case ARG1_NEGATIVE : return(minus_unsigned_bignum(arg2, arg1,
POSITIVE));
        case ARG2_NEGATIVE : return(minus_unsigned_bignum(arg1, arg2,
POSITIVE));
        case BOTH_NEGATIVE : return(plus_unsigned_bignum(arg1, arg2,
NEGATIVE));
        default : printf("Error: plus_signed_bignum\n");
    }
}
```

```
while (top2 >= arg2) {
    sum        = *arg1++ + *arg2++ + Get_Carry(sum);
    *answer++ = Get_Digit(sum);
}
```

In deze loop krijgt sum de waarde van een bigdigit uit arg1 gesommeerd bij de waarde van een bigdigit uit arg2, hier wordt tevens een carry of overflow bij opgeteld als de vorige optelling een resultaat had dat groter dan 99 is geweest. Hier komt dan ook de definitie van Get_ Carry() uit bignam.h naar voren.

```
#define Get_Carry(lw)    ((lw > MAX_DIGIT_SIZE) ? 1 : 0)
```

Hierin is de MAX_DIGIT_SIZE gelijk aan 99. De sum krijgt de waarde van het gedeelte wat in een byte kan worden opgeslagen, dus het gedeelte onder de 100. Hier komt de definitie van Get_Digit() uit bignum.h naar voren.

```
#define Get_Digit(lw)    ((lw > MAX_DIGIT_SIZE) ? lw - RADIX : lw)
```

4.8 Een groot voorbeeld

Hierin is de RADIX gelijk aan 100. De functie van de beide macro's zal in dit verband nu duidelijk zijn.

Het overige gedeelte van de functie zorgt ervoor dat het resultaat correct wordt doorgegeven.

```
bigdigit *plus_unsigned_bignum(arg1, arg2, sign)
bigdigit *arg1, *arg2;
bigdigit sign;
{
    char *malloc();
    int sum;
    bigdigit *result, *answer;
    bigdigit *top1, *top2;

    /* Draai arg1 en arg2 om zodat arg1 langer is */
    if (READ_LEN(arg1) < READ_LEN(arg2)) {
        answer = arg1;
        arg1 = arg2;
        arg2 = answer;
    }

    /* Alloceer ruimte voor het resultaat */
    result = answer = malloc(ALLOCLEN(arg1) + 1);
    WRITE_HLEN(answer) = (READ_LEN(arg1) + 1) >> 8;
    WRITE_LLEN(answer) = READ_LEN(arg1) + 1;
    SIGN(answer) = sign;

    /* Initialiseer pointers */
    top1 = Bignum_Top(arg1);
    top2 = Bignum_Top(arg2);
    arg1 = Bignum_Bottom(arg1);
    arg2 = Bignum_Bottom(arg2);
    answer = Bignum_Bottom(answer);
    sum = 0;

    /* Start optelling */
    while (top2 >= arg2) {
        sum        = *arg1++ + *arg2++ + Get_Carry(sum);
        *answer++ = Get_Digit(sum);
    }
```

4.8 Een groot voorbeeld

```
    /* Laat de carry doorwerken */
    while ((top1 >= arg1) && (Get_Carry(sum) != 0)) {
        sum        = *arg1++ + 1;
        *answer++ = Get_Digit(sum);
    }

    /* Copieer de rest naar answer */
    while (top1 >= arg1) {
        *answer++ = *arg1++;
    }
    *answer = Get_Carry(sum);

    /* Trim het answer */
    if (*answer == 0) {
        sum = READ_LEN(result) - 1;
        WRITE_HLEN(result) = sum >> 8;
        WRITE_LLEN(result) = sum;
    }

    return(result);
}
```

Aftrekken

Het aftrekken valt in twee soortgelijke routines uiteen. De eerste bepaalt het teken voor het resultaat en selecteert daar ook de juiste routine bij.

```
/* Aftrekken */

bigdigit *minus_signed_bignum(arg1, arg2)
bigdigit *arg1, *arg2;
{
    bigdigit *minus_unsigned_bignum();

    if (ZERO_BIGNUM(arg1) && ZERO_BIGNUM(arg2))
        return return_bignum_zero();
    if (big_compare(arg1, arg2) == EQUAL)
        return return_bignum_zero();

    switch(Categorize_Sign(arg1, arg2)) {
        case BOTH_POSITIVE : return(minus_unsigned_bignum(arg1, arg2,
POSITIVE));
        case ARG1_NEGATIVE : return(plus_unsigned_bignum(arg1, arg2,
NEGATIVE));
        case ARG2_NEGATIVE : return(plus_unsigned_bignum(arg1, arg2,
POSITIVE));
        case BOTH_NEGATIVE : return(minus_unsigned_bignum(arg2, arg1,
POSITIVE));
        default : printf("Error: minus_bignum sign error\n");
    }
}
```

4.8 Een groot voorbeeld

De tweede routine verricht de eigenlijke aftrekking. Het aftrekken wordt in de hier onderstaande loop uitgevoerd.

```
while (top2 >= arg2) {
    diff      = *arg1++ + (MAX_DIGIT_SIZE - *arg2++) + Get_Carry(diff);
    *answer++ = Get_Digit(diff);
}
```

Deze loop heeft een soortgelijke vorm als de loop die het optellen verricht. De overige stukken C-tekst uit de functie zijn nodig om de gehele aftrekking correct te laten verlopen.

```
bigdigit *minus_unsigned_bignum(arg1, arg2, sign)
bigdigit *arg1, *arg2;
bigdigit sign;
{
    char *malloc();
    bigdigit *result, *answer, *trim;
    bigdigit *top1, *top2;
    int diff;

    if (compare_unsigned(arg1, arg2) == TWO_BIGGER) {
        answer = arg1;
        arg1 = arg2;
        arg2 = answer;
        sign = !sign;
    }

    result = answer = malloc(ALLOCLEN(arg1));
    WRITE_HLEN(answer) = READ_LEN(arg1) >> 8;
    WRITE_LLEN(answer) = READ_LEN(arg1);
    SIGN(answer) = sign;

    top1 = Bignum_Top(arg1);
    top2 = Bignum_Top(arg2);
    arg1 = Bignum_Bottom(arg1);
    arg2 = Bignum_Bottom(arg2);
    answer = Bignum_Bottom(answer);
    diff = RADIX;
```

4.8 Een groot voorbeeld

```
    /* Loops voor minus_unsigned_bignum */
    while (top2 >= arg2) {
        diff       = *arg1++ + (MAX_DIGIT_SIZE - *arg2++) + Get_Carry(diff);
        *answer++ = Get_Digit(diff);
    }
    while ((top1 >= arg1) && (Get_Carry(diff) == 0)) {
        diff       = *arg1++ + MAX_DIGIT_SIZE;
        *answer++ = Get_Digit(diff);
    }
    while (top1 >= arg1) *answer++ = *arg1++;

    /* trim bignum, verwijder de voorafgaande nullen */
    trim = --answer;
    while (*trim-- == 0) {
        diff = READ_LEN(result) - 1;
        WRITE_HLEN(result) = diff >> 8;
        WRITE_LLEN(result) = diff;
    }

    return(result);
}
```

Vermenigvuldigen
Vermenigvuldigen is stukken ingewikkelder dan optellen of aftrekken. De vermenigvuldiging is weer in twee routines opgesplitst, die beiden een overeenkomstige taak hebben als bij optellen of aftrekken.

```
#include <bignum.h>

/* vermenigvuldigen */

bigdigit *mul_signed_bignum(arg1, arg2)
bigdigit *arg1, *arg2;
{
    bigdigit *mul_unsigned_bignum();

    if (ZERO_BIGNUM(arg1) && ZERO_BIGNUM(arg2))
        return return_bignum_zero();
    switch (Categorize_Sign(arg1, arg2)) {
        case BOTH_POSITIVE : return(mul_unsigned_bignum(arg1, arg2,
POSITIVE));
        case ARG1_NEGATIVE : return(mul_unsigned_bignum(arg2, arg1,
NEGATIVE));
        case ARG2_NEGATIVE : return(mul_unsigned_bignum(arg1, arg2,
NEGATIVE));
        case BOTH_NEGATIVE : return(mul_unsigned_bignum(arg1, arg2,
POSITIVE));
        default : printf("Error: mul_signed_bignum\n");
    }
}
```

4.8 Een groot voorbeeld

De vermenigvuldiging wordt uitgevoerd zoals iedereen dit op de lagere school heeft geleerd. Om dit nog even in herinnering te roepen volgt hier een voorbeeld:

```
  2405
    64
------ *
  9620
144300
------ +
153920
```

Eerst wordt 2405 met 4 veremnigvuldigd, daarna wordt het product van 2405 en 60 daarbij opgeteld. Op deze wijze wordt in deze routine de vermenigvuldiging ook uitgevoerd, met dien verstande dat er bigdigits met elkaar vermenigvuldigd worden (deze lopen van 0 tot 99) en het product van elke bigdigit-vermenigvuldiging wordt direct bij het al bestaande resultaat opgeteld.

Dit resulteert tot het volgende stuk C tekst:

```
while (top2 >= arg2) {
    arg1 = arg1_old;      /* herstel oude waarde */
    answer = answer_old++;
    while (top1 >= arg1) {
        prod       = (*arg1++ * *arg2) + *answer + Ge_mul_Carry(prod);
        *answer++ = Ge_mul_Digit(prod);
    }
    /* Laat de carry doorwerken */
     .....

    arg2++;
}
```

4.8 Een groot voorbeeld

De regel die praktisch al het werk doet is:

```
prod       = (*arg1++ * *arg2) + *answer + Ge_mul_Carry(prod);
```

In deze regel wordt per bigdigit de vermenigvuldiging uitgevoerd met de optelling van het al bestaande resultaat, plus dan nog wat er is overgebleven van een voorgaande bewerking. De Get_Carry() macro kan hier niet voor gebruikt worden, al is ook hier de fractie nodig die groter dan 100 is, maar dit kan niet worden verkregen door, als prod groter dan 100 is, de waarde 1 te nemen. Hier voor wordt de volgende macro definitie gebruikt.

```
#define Ge_mul_Carry(lw)  (lw / 100)
```

Deze macro zou voor optellen of aftrekken exact hetzelfde effect hebben gehad. Alleen zou daar een gedeelte van de computertijd worden verspild door een getal tussen 199 en 100 te delen door 100 of een getal kleiner dan 100 te delen door 100 (het resultaat hiervan is 1 of 0). Om dezelfde reden is ook hier van een nieuwe macro gebruik gemaakt om de fractie kleiner dan 100 te verkrijgen.

```
#define Ge_mul_Digit(lw)  (lw % 100)
```

De totale listing van de functie mul_unsigned_bignum() volgt hier onder.

```
bigdigit *mul_unsigned_bignum(arg1, arg2, sign)
bigdigit *arg1, *arg2;
bigdigit sign;
{
   char *malloc();
   int  prod;
   bigdigit *result, *answer, *answer_old, *temp;
   bigdigit *top1, *top2, *top3, *arg1_old;

   /* Draai arg1 en arg2 om, zodat arg1 langer is */
   if (READ_LEN(arg1) < READ_LEN(arg2)) {
      answer = arg1;
      arg1 = arg2;
      arg2 = answer;
   }

   /* Alloceer ruimte voor het resultaat */
   result = answer = malloc(HARDLENGTH(READ_LEN(arg1) + READ_LEN(arg2)));
   WRITE_HLEN(answer) = (READ_LEN(arg1) + READ_LEN(arg2)) >> 8;
   WRITE_LLEN(answer) = READ_LEN(arg1) + READ_LEN(arg2);
   SIGN(answer) = sign;
```

4.8 Een groot voorbeeld

```
    /* Initialiseer pointers */
    top1 = Bignum_Top(arg1);
    top2 = Bignum_Top(arg2);
    top3 = Bignum_Top(answer);
    arg1_old = arg1 = Bignum_Bottum(arg1);
    arg2 = Bignum_Bottum(arg2);
    temp = Bignum_Bottum(answer);
    answer_old = answer = Bignum_Bottum(answer);
    prod = 0;

    /* Bereidt het resultaat voor */
    while (top3 >= temp) *temp++ = 0;

    /* Start vermenigvuldiging */
    while (top2 >= arg2) {
       arg1 = arg1_old;    /* herstel oude waarde */
       answer = answer_old++;
       while (top1 >= arg1) {
          prod       = (*arg1++ * *arg2) + *answer + Ge_mul_Carry(prod);
          *answer++ = Ge_mul_Digit(prod);
       }
       /* Laat de carry doorwerken */
       while (Ge_mul_Carry(prod)) {
          prod       = *answer + Ge_mul_Carry(prod);
          *answer++ = Ge_mul_Digit(prod);
       }
       arg2++;
    }

    /* Trim bignum, verwijder de voorafgaande nullen */
    temp = Bignum_Top(result);
    while (*temp-- == 0) {
       prod = READ_LEN(result) - 1;
       WRITE_HLEN(result) = prod >> 8;
       WRITE_LLEN(result) = prod;
    }

    return(result);
}
```

4.8 Een groot voorbeeld

Delen
Het delen van bignums kost de meeste tijd in vergelijking met de andere operaties, dit is ook wel begrijpelijk als de verschillende routines naast elkaar worden gelegd. De optel en aftrek routines bevatten while-loops die niet genest zijn. De vermenigvuldigroutine bevat een enkelvoudig geneste while loop en de deelroutine bevat een dubbel geneste while loop. dit houdt in dat het programma de volgende structuur in zich heeft.

```
while {
   .....
   while {
      .....
      while {
         .....

      }
   }
}
```

Als regel kan ruwweg wel gesteld worden dat hoe dieper de nesting van loops is hoe meer tijd er voor zo'n routine nodig is om hem uit te voeren.

De eerste van de twee routines ziet er als volgt uit:

```
/* delen */

bigdigit *div_signed_bignum(arg1, arg2)
bigdigit *arg1, *arg2;
{
   bigdigit *div_unsigned_bignum();

   if (ZERO_BIGNUM(arg1))
      return return_bignum_zero();
   if (ZERO_BIGNUM(arg2)) {
      printf("Error: Delen door nul is niet toegestaan\n");
      return(-1);
   }
   switch (Categorize_Sign(arg1, arg2)) {
      case BOTH_POSITIVE : return(div_unsigned_bignum(arg1, arg2,
POSITIVE));
      case ARG1_NEGATIVE : return(div_unsigned_bignum(arg1, arg2,
NEGATIVE));
      case ARG2_NEGATIVE : return(div_unsigned_bignum(arg1, arg2,
```

4.8 Een groot voorbeeld

```
NEGATIVE));
        case BOTH_NEGATIVE : return(div_unsigned_bignum(arg1, arg2,
POSITIVE));
        default : printf("Error: div_signed_bignum\n");

    }
}
```

De deling verloopt evenals de veremnigvuldiging op de wijze zoals die op de lagere school is onderlegd. Om dit nog even in herinnering te roepen volgt hier een voorbeeld:

```
  62 / 1004 \ 0016 rest 12
        0
        --
        10
         0
        ---
        100
         62
        ----
         384
         372
         ---
          12
```

Het resultaat is dus 16. Op soortgelijke wijze wordt de deling ook door de routine verricht. Door herhaald aftrekken wordt bepaald hoe vaak de deler uit het te delen getal gaat. Door gebruik te maken van een vergelijkings routine die delen van bignums met elkaar kan vergelijken wordt bepaald wanneer een einde is gekomen aan dit herhaald aftrekken. De onderstaande listing moet duidelijk genoeg zijn om inzicht te geven in de werking ervan.

```
bigdigit *div_unsigned_bignum(arg1, arg2, sign)
bigdigit *arg1, *arg2;
bigdigit sign;
{
   char *malloc();
   int  diff, count, length2;
   bigdigit *result, *answer, *temp;
   bigdigit *semi_top1, *top2, *bottom2;
   bigdigit *bottom1, *semi_bottom1;

   if (READ_LEN(arg1) < READ_LEN(arg2)) return return_bignum_zero();
```

4.8 Een groot voorbeeld

```
    /* Alloceer ruimte voor het resultaat */
    result = answer = malloc(HARDLENGTH(READ_LEN(arg1) - READ_LEN(arg2) +
1));
    WRITE_HLEN(answer) = (READ_LEN(arg1) - READ_LEN(arg2) + 1) >> 8;
    WRITE_LLEN(answer) = READ_LEN(arg1) - READ_LEN(arg2) +1;
    SIGN(answer) = sign;

    /* Initialiseer pointers */
    semi_top1 = Bignum_Top(arg1);
    bottom1 = Bignum_Bottom(arg1);
    semi_bottom1 = Bignum_Top(arg1) - READ_LEN(arg2) + 1;

    top2 = Bignum_Top(arg2);
    length2 = READ_LEN(arg2);
    bottom2 = Bignum_Bottom(arg2);

    answer = Bignum_Top(answer);

    /* Start deling */
    while (bottom1 <= semi_bottom1) {
       count = 0; /* initeel aantal aftrekkingen */
       while (comp_part_bignum(semi_top1, top2, (semi_top1 - semi_bottom1
+ 1), length2) != TWO_BIGGER) {
          /* trek "arg2" van "arg1" af */
          diff = RADIX;
          arg1 = semi_bottom1;
          arg2 = bottom2;

          /* Loops om af te trekken */
          while (top2 >= arg2) {
             diff    = *arg1 + (MAX_DIGIT_SIZE - *arg2++) +
Get_Carry(diff);
             *arg1++ = Get_Digit(diff);
          }
          /* Laat de carry doorwerken */
          while ((semi_top1 >= arg1) && (Get_Carry(diff) == 0)) {
             diff    = *arg1 + MAX_DIGIT_SIZE;
             *arg1++ = Get_Digit(diff);
          }
          count++; /* increment het aantal aftrekkingen */
       }
       *answer-- = count;

       /* pas de pointers aan */
       if (*semi_top1 == 0) semi_top1--;
       semi_bottom1--;
   }
       /* Trim bignum, verwijder de voorafgaande nullen */
       temp = Bignum_Top(result);
       while (*temp-- == 0) {
          diff = READ_LEN(result) - 1;
          WRITE_HLEN(result) = diff >> 8;
          WRITE_LLEN(result) = diff;
       }

       return(result);
   }
```

4.8 Een groot voorbeeld

VERGELIJKINGS EN OVERIGE ROUTINES

De laatste functies van dit programma zijn een aantal vergelijkingsroutines en een functie die het nul-bignum retourneert. De eerste twee functies behoren bij elkaar en vergelijken twee bignums. De eerste voert de vergelijking uit aan de hand van het teken, als deze er niet uit mocht komen dan vergelijkt de tweede functie de corresponderende bigdigits totdat er uitsluitsel gegeven kan worden over de uitkomst van de vergelijking.

```
/* Bignum vergelijkings primitieven */

big_compare(arg1, arg2)
bigdigit *arg1, *arg2;
{
   switch(Categorize_Sign(arg1, arg2)) {
      case BOTH_NEGATIVE : return compare_unsigned(arg1, arg2);
      case BOTH_POSITIVE : return compare_unsigned(arg1, arg2);
      case ARG1_NEGATIVE : return TWO_BIGGER;
      case ARG1_NEGATIVE : return ONE_BIGGER;
      default : printf("Error: bignum_compare\n");
   }
}

compare_unsigned(arg1, arg2)
bigdigit *arg1, *arg2;
{
   bigdigit *limit;

   if ((READ_LEN(arg1)) > (READ_LEN(arg2))) return ONE_BIGGER;
   if ((READ_LEN(arg1)) < (READ_LEN(arg2))) return TWO_BIGGER;
   if ((READ_LEN(arg1)) == 0)             return EQUAL;
   limit = Bignum_Bottum(arg1);
   arg1  = Bignum_Top(arg1);
   arg2  = Bignum_Top(arg2);
   while (arg1 >= limit) {
      if (*arg1 > *arg2) return ONE_BIGGER;
      if (*arg1 < *arg2) return TWO_BIGGER;
      arg1 -= 1;
      arg2 -= 1;
   }
   return EQUAL;
}
```

4.8 Een groot voorbeeld

De volgende vergelijkingsroutine wordt door de deel-routine gebruikt om gedeeltes van de bignums met elkaar te vergelijken.

```
comp_part_bignum(arg1, arg2, length1, length2)
bigdigit *arg1, *arg2;
int length1, length2;
{
    if ((*arg1 != 0) && (length1 > length2)) return ONE_BIGGER;
    if ((*arg1 == 0) && (length1 > length2)) {
        arg1 -= 1;
        length1--;
    }
    if (length1 != length2) printf("Error: comp_part_bignum\n");
    while (length1--) {
        if (*arg1 > *arg2) return ONE_BIGGER;
        if (*arg1 < *arg2) return TWO_BIGGER;
        arg1 -= 1;
        arg2 -= 1;
    }
    return EQUAL;
}
```

De laatste functie die bij dit programma behoort, retourneert het nul-bginum en wordt door verschillende functies gebruikt.

```
bigdigit *return_bignum_zero()
{
    bigdigit *Result;

    Result = malloc(HARDLENGTH(0));
    SIGN(Result) = POSITIVE;
    WRITE_HLEN(Result)  = 0;
    WRITE_LLEN(Result)  = 0;
    return(Result);
}
```

4.8 Een groot voorbeeld

Dit totale programma moet u een idee hebben gegeven hoe een programma van enige omvang op goed gestructureerde en overzichtelijke wijze kan worden opgezet. Het is altijd verstandig om programma's op zo'n manier te schrijven, want een programma moet onderhouden worden als het gebruikt wordt. Als een programma lang mee gaat dan worden er fouten in ontdekt, eisen van de gebruikers ten aanzien van het programma wijzigen, er wordt op een andere machine overgeschakeld. Er zijn dus legio redenen om een programma goed en doordacht op te zetten.

Ook dit programma zal ongetwijfeld fouten bevatten en zal toch ook wel wat uigerbeid moeten worden als men er een praktische desk calculator van wil maken, maar voor dit boekwerk is dit allemaal van ondergeschikt belang. Het gaat erom dat u zelf straks de programma's kunt gaan schrijven die u wilt hebben. Daarbij is dit programma een leuk uitgangspunt, maar zeker niet de enige en beste weg.

8/4.10

Utilities

Deze paragraaf bevat tal van handige programma's en de listings van menig handige utility. Met de beschrijving van deze programma's zullen tal van praktische en machine-afhankelijke problemen worden opgelost. Uiteraard zal er bij de beschrijving van de programma's van worden uitgegaan dat u al op de hoogte bent van de hiervoor verschenen paragrafen, omdat de beschrijving voornamelijk over het functionele deel van de routines gaat en niet zozeer over de gebruikte C constructies.

Het printen van C listings
Met de door Proline geleverde shell is het mogelijk om op eenvoudige wijze tekst of een listing op papier te zetten, dit kan met het pr commando. Zoals hier onder,

$ pr programma.c »

Door de twee groter-dan tekens achter elkaar te plaatsen drukt de printer de listing af. Aan deze methode kleven echter enkele nadelen:

1) Alles wordt achter elkaar afgedrukt, zonder dat er met enige pagina indeling rekening wordt gehouden.
2) De speciale C tekens zoals {, }, |, \, en ~worden niet juist afgedrukt, deze worden als Commodore tekens afgedrukt.
3) De naam van de file staat niet op papier.
4) Het is niet mogelijk diverse files achter elkaar met één commando te laten afdrukken.

Uit deze nadelen vloeit dan automatisch het volgende wensenlijstje voort.

1) Er dient rekening te worden gehouden met de pagina indeling.
2) Deze pagina indeling moet instelbaar zijn.
3) De speciale C tekens dienen correct te worden afgedrukt.
4) De naam van de file dient op papier te staan.
5) Meerdere files moeten achter elkaar kunnen worden afgedrukt.

Deze eisen leveren, samen met de eis, dat de invoer gegeven dient te worden aan de Unix standaard tot het programma: lpr. De synopsis (aanroepbeschrijving) van lpr is:

$ lpr [-rs] [-number] [-tnumber] [-mnumber] [-bnumber] [filename] [filename] ...

Waarin de opties de volgende betekenissen hebben:

4.10 Utilities

-r : reset de printer
-s : elke file begint op een nieuwe pagina.
-lnumber : stel de pagina lengte in, default 72.
-tnumber : stel het aantal blanco regels voor de header in, default 3.
-mnumber : stel het aantal blanco regels na de header in, default 2.
-bnumber : stel de regel die de laatste is waarop wordt geprint, default 66.

De default waarden die hier genoemd zijn kunnen door een simpele ingreep in het programma veranderd worden.

De main routine van het programma bestaat uit de definitie van de defaultwaarden die hierboven genoemd zijn. Het grootste gedeelte van main bestaat uit het afhandelen van de opties. In de laatste paar regels van main worden te printen files geopend en worden ze door de functie pr_file afgedrukt.

```
#include <stdio.h>

#define MAXLINE 81 /* max number of chars a line */
#define MAXLEN     256
#define MAXLINES  1000
#define TOP        3
#define MARGIN     2
#define PAGELEN 72
#define BOTTOM   66
#define YES        1
#define NO         0

unsigned top, margin, pagelen, bottom, separate;
unsigned pageno, lineno;
int printer;

main(argc, argv)
unsigned argc;
char **argv;
{
   char *s;
   FILE fin;
   char *a[MAXLINES];
   int  n;

   printer  = init_pr();
```

4.10 Utilities

```
top        = TOP;
margin     = MARGIN;
pagelen    = PAGELEN;
bottom     = BOTTOM;
separate   = NO;
pageno     = 0;
lineno     = 0;

while (--argc && **++argv == '-')
    for (s = *argv + 1; *s; s++)
        switch (*s) {
        case 'l': /* adjust default page length */
            pagelen = atoi(++s);
            bottom  = pagelen - 6;
            if (pagelen == 0)
                error();
            while (isdigit(*s)) s++;
            s--;
            break;
        case 't': /* adjust default top skipping */
            top = atoi(++s);
            if (top > pagelen)
                error();
            while (isdigit(*s)) s++;
            s--;
            break;
        case 'm': /* adjust default margin skipping */
            margin = atoi(++s);
            if ((margin + top) > pagelen)
                error();
            while (isdigit(*s)) s++;
            s--;
            break;
        case 'b': /* adjust default bottom skipping */
            bottom = atoi(++s);
            if (bottom > pagelen)
                error();
            while (isdigit(*s)) s++;
            s--;
            break;
        case 'r': /* reset printer */
            reset_pr();
            break;
        case 's': /* seperate file, separate page */
            separate = YES;
            break;
        default:
            error();
            break;
        }
```

4.10 Utilities

```
    while (argc--) {
        if ((fin = fopen(*argv++, "r")) == NULL || ferror()) {
            printf("can't open %s\n", *(argv - 1));
            exit(1);
        }
        n = read_lines(fin, a, MAXLINES);
        pr_file(a, *(argv - 1), n);
    }

    close_pr(printer);
}
```

De functie pr_file neemt als invoer een FILE variabele en een character pointer, de FILE variabele wordt gebruikt door de standaard functie fgets en de character pointer wijst naar een string waar de naam van de af te rukken file staat. De naam van de file wordt in de header afgedrukt.

De functie pr_file drukt een file af op de printer. Hierin zijn de volgende stappen te onderscheiden:

1) Als de file op een aparte pagina moet beginnen, ga dan naar het begin van een pagina.
2) Zolang de file nog niet geheel is afgedrukt doe dan de volgende stappen.
3) Sla een aantal (top) regels over.
4) Druk de header af, met het juiste pagina nummer.
5) Sla een aantal (margin) regels over.
6) Print een aantal regels uit de file.
7) Sla een aantal (margin - bottom) regels over, zodanig dat de printer kop aan het begin van de pagina staat.

Binnen pr_file zorgt de functie lprint ervoor dat een regel wordt afgedrukt.

```
pr_file(b, file, max)
char *b[];
char *file;
int  max;
{
    char inline[MAXLINE + 1];
    int i = 0;
```

4.10 Utilities

```
    if (separate == NO && lineno != 0)
        if (top + margin + 1 + lineno >= pagelen) {
            skip(pagelen - lineno);
            lineno = 0;
        }
        else {
            skip(top);
            head(file, pageno);
            skip(margin);
            lineno += top + margin + 1;
        }

    while (i < max) {
        strcpy(inline, b[i++]);
        if (lineno == 0) {
            skip(top);
            pageno++;
            head(file, pageno);
            skip(margin);
            lineno = top + margin + 1;
        }
        lprint(inline);
        lineno++;
        if (lineno >= bottom) {
            skip(pagelen - lineno);
            lineno = 0;
        }
    }
    if (separate == YES) {
        pageno = 0;
        if (lineno > 0) {
            skip(pagelen - lineno);
            lineno  = 0;
        }
    }
}
```

De functie lprint drukt een regel af. Maar een regel kan niet onbewerkt naar de printer worden gestuurd, want de bovengenoemde speciale C tekens worden dan foutief. afgedrukt. Deze dienen dus uit de regel toworden gefilterd en dienen op een speciale manier te worden afgedrukt. De PETSCII waarde van deze tekens is voor:

```
{  : 219
}  : 221
/  :  92
|  : 175
~  : 223
```

Als deze characters in een regel zitten dan worden de desbetreffende afdruk routines uitgevoerd. Als zo'n character niet in de

4.10 Utilities

regel zit dan wordt de regel naar de printer gestuurd.

De functies skip, head en error slaan respectievelijk een regel over, drukken de header af en brengen een foutmelding op het scherm bij foutief gebruik.

De laatste twee routines van lpr lezen de file in, dit zijn routines die al in een iets andere vorm beschreven zijn. Deze routines lezen de informatie niet van standaard input maar van een te specificeren file.

De routines die voor het correct aansturen van de printer zorgen staan in 8/4.9.3: Printer functies.

```
lprint(inline)
char inline[];
{
    char *s, *p, outline[MAXLINE + 1];

    s = inline;
    p = outline;
    while (*s) {
        switch (*s) {
        case 219:
            *p = '\0';
            print_lb(outline);
            s++;
            p = outline;
            break;
        case 221:
            *p = '\0';
            print_rb(outline);
            s++;
            p = outline;
            break;
        case 92:
            *p = '\0';
            print_bs(outline);
            s++;
            p = outline;
            break;
        case 175:
            *p = '\0';
            print_ti(outline);
            s++;
            p = outline;
            break;
```

4.10 Utilities

```
        case 223:
            *p = '\0';
            print_vb(outline);
            s++;
            p = outline;
            break;
        default:
            *p++ = *s++;
            break;
        }
    }
    *p = '\0';
    fprintf(printer, "%s", outline);
}
```

```
skip(n)
int n;
{
    while (n--)
        fprintf(printer, "\n");
}

head(name, pageno)
char *name;
unsigned pageno;
{
    fprintf(printer, "%-30sPage %d\n", name, pageno);
}

error()
{
    printf("usage: lpr [-rs] [-lnumber]\n");
    printf("           [-tnumber] [-mnumber] [-bnumber]\n");
    printf("           [filename] [filename] ...\n");
    exit(1);
}

read_lines(fin, lineptr, maxlines) /* get input */
FILE fin;
char *lineptr[];
int  maxlines;
{
```

4.10 Utilities

```
    int len, nlines;
    char *p, *malloc(), line[MAXLEN];

    nlines = 0;
    while ((len = getline(fin, line, MAXLEN)) > 0)
        if (nlines >= maxlines)
            return(-1);
        else if ((p = malloc(len)) == NULL)
            return(-1);
        else {
            strcpy(p, line);
            lineptr[nlines++] = p;
        }
    return(nlines);
}

getline(fin, s, lim) /* get line into s, return length */
FILE fin;
char s[];
int lim;
{
    int c, i;

    i = 0;
    while (--lim && (c = fgetc(fin)) != EOF && c!= '\n')
        s[i++] = c;
    if (c == '\n')

        s[i++] = c;
    s[i] = '\0';
    return(i);
}
```

File concatenatie
Vaak bestaat er de behoefte aan een programma dat twee of meer files achter elkaar in een nieuw te vormen file plaatst. In deze sectie zal zo'n file concatenatie programma worden besproken.
Dit programma is tevens in staat om de naam van de file voor iedere regel af te drukken. Ook kan het programma het regelnummer van elke regel afdrukken. Als beide opties worden gebruikt dan ziet een regel uit de file fiets er bijvoorbeeld als volgt uit:

fiets: 213: Dit is regel 213, uit de file fiets.

De synopsis van het programma cat is:

4.10 Utilities

$ cat [-f] [-n] [filename] [filename] ...

Waarin de opties de volgende betekenis hebben:

-f : Print de naam van de file voor iedere regel.
-n : Print het regelnummer voor iedere regel.

De main routine van cat behandelt op bekende wijze de verschillende opties en roept de functie cat aan. De functie cat krijgt een FILE variabele mee, twee parameters die aangeven welke opties gebruikt worden en de naam van de file.

```
#include <stdio.h>

#define MAXLINE 256 /* max number of characters a line */
#define YES 1;
#define NO  0;

char line[256];

main(argc, argv) /* concatenate files */
unsigned argc;
char **argv;
{
   int number, prname;
   char *s;
   FILE fin;

   number = NO;
   prname = NO;

   while (--argc && ***++argv == '-')
      for (s = *argv + 1; *s; s++)
         switch (*s) {
            case 'f':          /* print file name */
                 prname = YES;
                 break;
            case 'n':          /* print line numbers */
                 number = YES;
                 break;
            default:
                 printf("cat: illegal option %c\n", *s);
                 printf("usage: cat [-n] [-f] [filename] [filename] ...\n");
                 break;
```

4.10 Utilities

```
        }
    if (argc == 0)
        cat(stdin, number, prname, "stdin");
    else
        while (argc--) {
            if ((fin = fopen(*argv++, "r")) == NULL || ferror()) {
                printf("can't open %s\n", *argv);
                exit(1);
            }
            cat(fin, number, prname, *(argv - 1));
        }
    exit(0);
}
```

4.10 Utilities

De functie cat test de variabelen die de waarde van de opties aangeven en drukt de corresponderende informatie af indien gewenst en drukt uiteindelijk de regel af uit de file. Deze handelingen worden voor alle regels uit de desbetreffende file gedaan.

```
cat(fin, number, prname, file)
FILE fin;
int number, prname;
char *file;
{
    int lineno;

    lineno = 0;
    while (fgets(line, MAXLINE, fin) != NULL) {
        lineno++;
        if (prname)
            printf("%s: ", file);
        if (number)
            printf("%d: ", lineno);
        printf("%s", line);
    }
    fclose(fin);
}
```

Het kopiëren van files

Files moeten vaak verplaatst worden van de ene diskette naar de andere. De Proline shell voorziet niet in een routine die dat doet. Dit is de reden waarom hieronder een programmaatje staat wat deze simpele bewerking doet.

De eerste file wordt van de diskette gelezen en in het geheugen van de Commodore geplaatst. Dan wordt verzocht om de target diskette in de drive te plaatsen en wordt, na een druk op de RETURN toets de file op de target diskette geplaatst.

De synopsis van cp is:

$ cp filename filename

```
#include <stdio.h>
#define ELSIZE 1 /* size of elements */
#define NELEM  1 /* number of elements */
#define MAX_FILE_SIZE 20000

main(argc, argv) /* copy a file */
int argc;
char *argv[];
{
   char s[MAX_FILE_SIZE], *c;
   int i, top;
   FILE f1, f2;

   if (argc != 3) {
      printf("Usage: cp filename1 filename2\n");
      exit(1);
   }
   f1 = fopen(argv[1], "r"); /* open first file */
   if (ferror(f1)) {
      printf("Error: %s can't be opened\n",argv[1]);
      exit(1);
   }
   i = 0;

   while (fread(c, ELSIZE, NELEM, f1))
      s[i++] = *c;

   top = --i;

   printf("insert target disk\n");
   getchar();
```

4.10 Utilities

```
    f2 = fopen(argv[2], "w"); /* open second file */
    if (ferror(f2)) {
       printf("Error: %s can't be opened\n",argv[2]);
       exit(1);
    }

    i = 0;

    while (i <= top) {
       *c = s[i++];
       fwrite(c, ELSIZE, NELEM, f2);
    }

    exit(0);
}
```

Clear screen

Het programmaatje clr, maakt het scherm schoon. Het schrijft een 'hartje' of PETSCII 147 naar het scherm.

```
main()
{
   clrscrn();
}

clrscrn() /* clear screen */
{
    putchar (147);
}
```

Kolom manipulatie

Computers worden meestal gebruikt om gegevens te manipuleren. Een bepaalde vorm van gegevensopslag is de tabel. Tabellen worden gebruikt om gegevens te groeperen en duidelijk weer te geven. Zodra men met tabellen werkt komt men vaak tot de conclusie dat tabellen wel eens gecombineerd kunnen worden of dat een kolom uit een tabel verplaatst moet worden. Om nu te voorkomen dat al de gegevens opnieuw ingetikt moeten worden bestaan er programma's, die met behulp van de oude gegevens nieuwe tabellen kunnen maken.

In deze sectie zijn twee programma's opgenomen: merge en split, om respectievelijk tabellen te mengen en te splitsen.

Het programma merge voegt twee files samen tot een nieuwe file, waarbij de eerste regel uit de file voor de eerste regel uit de twee file wordt gezet. Op deze manier worden alle corresponderende regels bij elkaar gevoegd. Stel da in tabellen alle regels even lang zijn, dan heeft dit tot gevolg dat als merge op twee van zulke files wordt los gelaten er een nieuwe tabel zal ontstaan.

De synopsis van merge is:

$ merge filename filename

De uitvoer wordt naar standaard output geredigeerd.

4.10 Utilities

```
#include <stdio.h>
#define CHARS 200 /* max number of characters a line */

main(argc, argv) /* merge two files */
int argc;
char *argv[];
{
    char *src1, *src2, *tmp;
    FILE f1, f2;

    if (argc != 3) {
        printf("Usage: merge filename filename\n");
        exit(1);
    }
    f1 = fopen(argv[1], "r"); /* open first file */
    if (ferror(f1)) {
        printf("Error: %s can't be opened\n",argv[1]);
        exit(1);
    }
    f2 = fopen(argv[2], "r"); /* open second file */
    if (ferror(f2)) {
        printf("Error: %s can't be opened\n",argv[2]);
        exit(1);
    }

    src1 = malloc(CHARS); /* allocate space for strings */
    src2 = malloc(CHARS);


    while (fgets(src1, CHARS, f1) && fgets(src2, CHARS, f2)) {
        tmp = src1;
        while (*tmp != '\n') tmp++; /* search for newline */
        *tmp = '\0';                /* replace by end of line */
        printf("%s%s", src1, src2);
    }
    while (fgets(src1, CHARS, f1))
        printf("%s", src1);
    while (fgets(src2, CHARS, f2))
        printf("%s", src2);
    fclose(f1);
    fclose(f2);
    exit(0);
}
```

4.10 Utilities

Het tweede programma is split, split splitst een tabel in twee stukken. De grens van de twee stukken wordt bepaald door een kolom nummer wat als parameter dient te worden meegegeven.

De synopsis van split is:

$ split -cnumber filename filename

De invoer wordt van standaard input genomen.

Beide programma's zijn eenvoudig van structuur en opzet en behoeven geen verdere uitleg.

```
#include <stdio.h>
#define CHARS 200 /* max number of characters a line */

main(argc, argv) /* split a file into two files */
int argc;
char *argv[];
{
   char *src, *tmp;
   FILE f1, f2;
   int column, i;

   if (argc != 4) {
      printf("Usage: split -cnumber filename filename\n");
      exit(1);
   }
   if (argv[1][0] != '-' && argv[1][1] != 'c') {
      printf("Usage: split -cnumber filename filename\n");
      exit(1);
   }
   column = atoi(argv[1] + 2);

   f1 = fopen(argv[2], "w"); /* open first file */
   if (ferror(f1)) {
      printf("Error: %s can't be opened\n",argv[2]);
      exit(1);
   }
   f2 = fopen(argv[3], "w"); /* open second file */
   if (ferror(f2)) {
      printf("Error: %s can't be opened\n",argv[3]);
      exit(1);
   }

   src  = malloc(CHARS);
```

4.10 Utilities

```
    while (gets(src)) {
        tmp = src;
        i = 0;
        while (*tmp++) i++;
        if (i <= column) { /* no string for second file */
            fprintf(f1, "%s\n", src);
            fprintf(f2, "\n");
        }
        else { /* string for second file is present */
            fprintf(f2, "%s\n", (src + column));
            *(src + column) = '\0';
            fprintf(f1, "%s\n", src);
        }
    }
    fclose(f1);
    fclose(f2);
    exit(0);
}
```

STRUCT

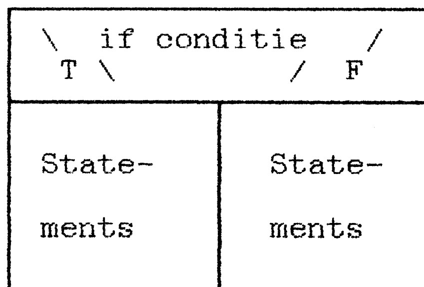
Inleiding
Tegenwoordig wordt er veel gepraat over het documenteren van programma's. Een handig hulpmiddel daarvoor kan bijvoorbeeld de Nassi-Sneider diagram zijn. Deze diagrammen kunnen tevens tijdens het ontwerpen van een programma een nuttig rol spelen. Nassi-Sneider diagrammen geven snel een overzicht van de control flow van de programmatuur, zonder dat de programmatuur daarvoor zelf gelezen behoeft te worden.

Nassi-Sneider diagrammen zijn opgebouwd uit enkele basis figuren:

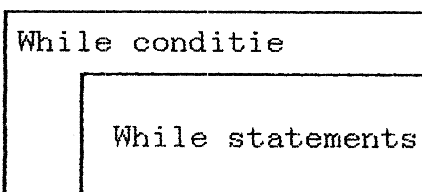
Het statement:

```
Statement
```

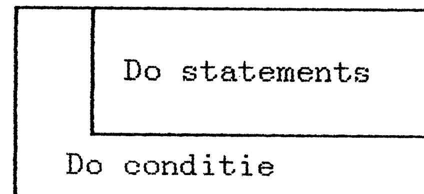
Het if-statement:



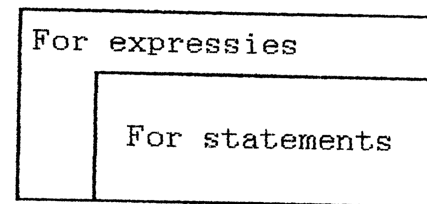
De while-loop:



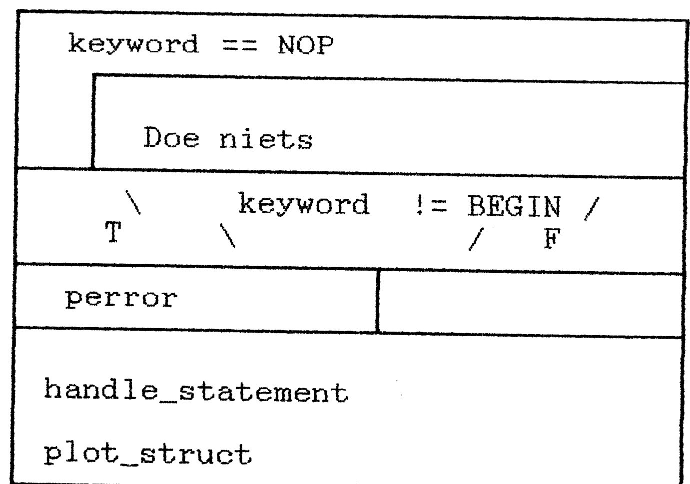
De do-loop:



De for-loop:



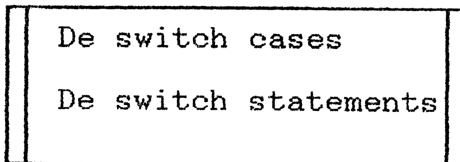
Met deze basis figuren kunnen de meeste programma's goed grafisch worden weergegeven. Ons hoofdprogramma bijvoorbeeld, zie verder, ziet er in een Nassi-Sneider diagram als volgt uit:



C kent een aantal constructies waar geen Nassi-Sneider basisfiguur voor bestaat, denk hier bijvoorbeeld aan de switch en de case. Het break-, continue- en return-statement zijn statements die ook niet direct of hele-

maal niet in een Nassi-Sneider diagram gevangen kunnen worden. Maar in het algemeen wordt er vaak gebruik gemaakt van dit soort C-features, zodat deze diagrammen ook voor de taal C zeer zinvol kunnen zijn.

Het switch statement kan beschouwd worden als een verzameling if statements, die zonder probleem met een Nassi-Sneider diagram gerepresenteerd kunnen worden. Om toch aan te geven dat het om een switch gaat en niet om een verzameling if statements wordt er een extra basis figuur aan de boven staande verzameling toegevoegd, als volgt:



Het programma
Het programma valt in ruwweg twee delen uiteen. Een parser en een plotter. Het eerste gedeelte de parser, 'leest' de standaard input, dit moet een compileerbare C-source zijn en bouwt een programma flow-boom op. Het tweede gedeelte, de plotter, heeft als input de boom en verwerkt die zodanig dat er een Nassi-Sneider diagram uitkomt.

De *parser* begint al in het hoofdprogramma, hier wordt naar het begin van een functie gezocht. Zodra dit is gevonden kan de eigenlijke parsing van de functie beginnen.

De header-files die worden ge-*include* zijn defs.h en struct.h.

De functie *scan_until_next_keyword* leest de standaard input totdat er weer een nieuw keywoord is gevonden, alle tweede argu-

```
#include "struct.h"
#include "defs.h"

char *malloc();

main()
{
  struct node *root;
  enum node_type keyword;

  while ((keyword = scan_until_next_keyword()) == NOP) {
       ;
  }
  if (keyword != BEGIN) {
       perror("STRUCT - BEGIN not found\n");
       exit(-1);
  }
  root = (struct node *)malloc(sizeof(struct node));
  root -> next = STR_NULL;
  handle_statement(root);
  plot(root);
  exit(0);
}
```

Listing Struct.C

4.10 Utilities

```
#define enum
#define node_type int
#define IF             0
#define FOR            1
#define WHILE          2
#define DO             3
#define SWITCH         4
#define STATEMENT      5
#define CASE           6
#define CASE_BREAK     7
#define CASE_DEFAULT   8
#define ELSE           9
#define END           10
#define BEGIN         11
#define NOP           12

struct node {enum node_type                          type;
     struct node *next;
     struct node *primary;
     struct node *secundary;
  } ;

typedef char bool;

extern enum node_type handle_statement();
extern enum node_type handle_case();

extern void handle_if();
extern void handle_for();
extern void handle_while();
extern void handle_do();
extern void handle_switch();
extern void handle_end();

extern enum node_type scan_until_next_keyword();
extern enum node_type scan_while();
extern void push_keyword_back();

extern struct node *create_node();

extern bool scan();
extern bool match();
extern bool keyword_match();

extern void print_node_type();

extern char *malloc();
```

Listing Struct.H

```
#define NULL 0
#define STR_NULL (struct node *)0;

#define TRUE  1
#define FALSE 0
```

Listing Defs.H

menten van match zijn de keywoorden waarna gezocht wordt. De functie *push_keyword_back* 'duwt' een keywoord terug, zodat dit keywoord weer door scan_until_next_keyword gelezen kan worden. Zie de listing hieronder:

```
#include "defs.h"
#include "struct.h"
#include <stdio.h>

static bool buf_full = FALSE;
static enum node_type buffer;

enum node_type scan_until_next_keyword()
{
  char s[100];
  int i;

  if (buf_full == TRUE) {
      buf_full = FALSE;
      return(buffer);
  }
  while (TRUE) {
      i = scanf("%s", s);
      if (i == EOF) {
     printf("EOF reached\n");
     exit(-1);
      } else if (i == 0) {
     printf("SCAN - error with scanf\n");
     exit(-1);
      }
#ifdef DEBUG1
      printf("%s", s);
#endif
      if (match(s, "{")) {
     return(BEGIN);
      } else if (match(s, "}")) {
     return(END);
      } else if (match(s, "if")) {
     return (IF);
      } else if (match(s, "else")) {
     return(ELSE);
      } else if (match(s, "for")) {
```

Listing Scan.C

```
        return(FOR);
         } else if (match(s, "while")) {
        return(WHILE);
         } else if (match(s, "do")) {
        return(DO);
         } else if (match(s, "switch")) {
        return(SWITCH);
         } else if (match(s, "case")) {
        return(CASE);
         } else if (match(s, "break")) {
        return(CASE_BREAK);
         } else if (match(s, "default")) {
        return(CASE_DEFAULT);
         } else {
        return(NOP);
         }
    }
}

void push_keyword_back(keyword)
enum node_type keyword;
{
    buffer = keyword;
    buf_full = TRUE;
}
```

Listing Scan.C

De *match* functies vergelijken twee strings met elkaar, zoals strcmp dit ook doet, met dit verschil dat er een groter aantal verschillen in de strings mogen zitten om toch te 'matchen'. De listing maakt een en ander duidelijk:

```
#include "defs.h"
#include "struct.h"
#include <ctype.h>

bool match(word1, word2)
char word1[], word2[];
{
   int i, j, k;
   bool matches = FALSE;

#ifdef DEBUG1
   printf("word1: #%s#\nword2: #%s#\n", word1, word2);
#endif
   if ((strlen(word1) == 0) || (strlen(word2) == 0)) {
     return(FALSE);
   }
```

4.10 Utilities

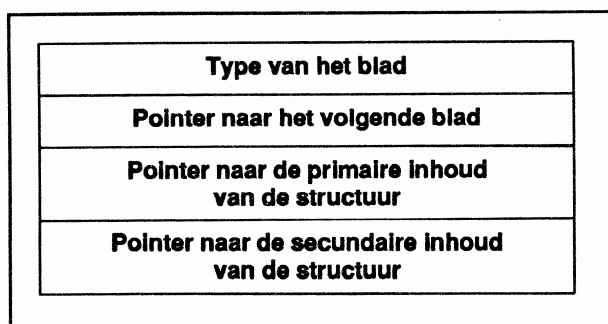
```
    if (strlen(word1) <= strlen(word2)) {
       for (i = 0; i <= strlen(word2) - strlen(word1); i++) {
          k = i;
          matches = TRUE;
          for (j = 0; j < strlen(word1); j++) {
             if (word1[j] != word2[k++]) {
                matches = FALSE;
                return(FALSE);
             }
          }
       }
    } else {
       for (i = 0; i <= strlen(word1) - strlen(word2); i++) {
          k = i;
          matches = TRUE;
          for (j = 0; j < strlen(word2); j++) {
             if (word2[j] != word1[k++]) {
                matches = FALSE;
                return(FALSE);
             }
          }
       }
    }
    return(TRUE);
 }

 bool keyword_match(string, keyword)
 char string[], keyword[];
 {
    int i, j, k;
    bool matches = FALSE;

 #ifdef DEBUG1
    printf("word1: #%s#\nword2: #%s#\n", word1, word2);
 #endif
    if ((strlen(string) == 0) || (strlen(keyword) == 0)) {
         printf("KEYWORD_MATCH - one of the strings is empty\n");
         exit(-1);
    }
    if (strlen(string) < strlen(keyword)) {
         return(FALSE);
    } else {
       for (i = 0; i <= strlen(string) - strlen(keyword); i++) {
          k = i;
          matches = TRUE;
          for (j = 0; j < strlen(keyword); j++) {
             if (keyword[j] != string[k++]) {
                matches = FALSE;
                return(FALSE);
             }
```

Listing Match.C

```
            }
        }
        /*
         *    So far so good
         */
        if (string[strlen(keyword) + 1] == '\0') {
           return(TRUE);
        } else if (isspace(string[strlen(keyword) + 1])) {
           return(TRUE);
        } else {
           return(FALSE);
        }
    }
}
```

Listing Match.C

De spil van de parser is de routine *handle_statement*. Bij elk nieuw statement wordt er een blad aan de boom gehangen. Dit blad van het type struct node is een structure, die er zo uitziet:



Het type is één van: IF, FOR, WHILE, DO, SWITCH, STATEMENT. De pointer naar het volgende blad naar het volgende statement, het huidige statement, wordt van begin tot het einde als één statement beschouwd. Dus als er bijvoorbeeld een if-statement wordt gevolgd door een for statement, en dat for statement volgt direct op de laatste bracket van het else gedeelte van de if, dan wijst de pointer naar het volgende blad van het if-statement naar het blad van het for-statement. De pointer naar de primaire inhoud van de structuur wijst naar het eerste statement dat binnen het huidige statement valt, dus als we het bovenstaande voorbeeld nemen dan wijst deze pointer naar het blad van het eerste statement wat uitgevoerd wordt als de if-expressie waar is. De pointer naar de secundaire structuur wijst naar die statements die niet bij de primaire inhoud van de structuur horen, dus om het bovenstaande voorbeeld weer te gebruiken, deze pointer wijst naar het eerste statement van het else-gedeelte van de if-expressie.

De functie *Handle_statement* zoekt naar het volgde keywoord en laat de juiste routine de juiste handelingen verrichten die bij dat keywoord nodig zijn.

De routine *handle_if* verwerkt de if statements. De functie moet rekening houden met het optionele else gedeelte van het if-statement. Daarvoor moet de routine een stukje van de invoer lezen en test of het volgende keywoord ELSE is of niet. Als het niet ELSE is, is het niet handig dat dit keywoord is gelezen, want dit moet op een andere plaats in het programma gebeuren, daarom is er een functie (push_keyword_back) om deze lees actie ongedaan te maken. Dit is te vergelijken met de functies getch en ungetch.

4.10 Utilities

```
#include "struct.h"
#include "defs.h"

enum node_type handle_statement(current)
struct node *current;
{
    enum node_type keyword;
    static short int depth = 0;

    depth++;
    printf("HANDLE_STATEMENT - depth: %d\n", depth);
    while (TRUE) {
  while ((keyword = scan_until_next_keyword()) == NOP) {
      ;
  }
#ifdef DEBUG
  print_node_type("HANDLE_STATEMENT", keyword);
#endif
  if (keyword == BEGIN) {
      current = create_node(current, STATEMENT);
      keyword = handle_statement(current);
      if ((keyword == END) || (keyword == ELSE)) {
     depth--;
#ifdef DEBUG
  printf("HANDLE_STATEMENT - return depth: %d\n", depth);
#endif
     return(END);
      }
  } else {
      switch (keyword) {
      case IF:
     current = create_node(current, IF);
     handle_if(current);
#ifdef DEBUG1
     printf("HANDLE_STATEMENT - handle_if is returned\n");
#endif
     break;
      case FOR:
     current = create_node(current, FOR);
     handle_for(current);
     break;
      case WHILE:
     current = create_node(current, WHILE);
     handle_while(current);
     break;
      case DO:
     current = create_node(current, DO);
     handle_do(current);
     break;
      case SWITCH:
```

Listing Handle_Statement.C

4.10 Utilities

```
      current = create_node(current, SWITCH);
      handle_switch(current);
      break;
       case ELSE:
      break;
       case END:
         depth--;
#ifdef DEBUG
         printf("HANDLE_STATEMENT : return depth: %d\n", depth);
#endif
         return(keyword);
      break;
       case CASE:
      break;
       case CASE_BREAK:
      break;
       case CASE_DEFAULT:
      break;
       default:
      perror("HAND_STATEMENT: Wrong case\n");
      exit(-1);
      break;
       }
  }
     }
}
```

Listing Handle_Statement.C

```
#include "defs.h"
#include "struct.h"

void handle_if(current)
struct node *current;
{
  enum node_type keyword;

  keyword = handle_statement(current->primary);
#ifdef DEBUG
  printf("HANDLE_IF - handle_statement is returned\n");
#endif
  if (keyword != END) {
      perror("HANDLE_IF: end not found");
  }
#ifdef DEBUG
  printf("HANDLE_IF - search for optinal keyword else\n");
#endif
  keyword = scan_until_next_keyword();
  if (keyword == ELSE) {
#ifdef DEBUG
```

Listing Handle_If.C

4.10 Utilities

```
  printf("HANDLE_IF - optinal keyword else found\n");
#endif
      /*
       * handle else part
              */
           keyword = handle_statement(current->secundary);
        } else {
      push_keyword_back(keyword);
           current -> secundary = STR_NULL;
  }
#ifdef DEBUG
  printf("HANDLE_IF - returned\n");
#endif
}
```

Listing Handle_If.C

De routine *handle_while* verwerkt de while-statements. While-statements hebben geen secundair gedeelte. De listing blijft dan ook eenvoudig:

```
#include "struct.h"
#include "defs.h"

void handle_while(current)
struct node *current;
{
  if (handle_statement(current -> primary) != END) {
      perror("HANDLE_WHILE: end not found");
  }
  current -> secundary = STR_NULL;
}
```

Listing Handle_While.C

Handle_for en *handle_do* doen iets soortgelijks als de voorgaande functie. Vandaar ook dat de beide listings weinig nieuws zullen onthullen:

```
#include "struct.h"
#include "defs.h"

void handle_for(current)
struct node *current;
{
  if (handle_statement(current -> primary) != END) {
      perror("HANDLE_FOR: end not found");
  }
  current -> secundary = STR_NULL;
}
```

Listing Handle_For.C

```
#include "struct.h"
#include "defs.h"

void handle_do(current)
struct node *current;
{
  if (handle_statement(current -> primary) != END) {
      perror("HANDLE_DO: end not found");
  }
  if (scan_until_next_keyword() != WHILE) {
      perror("HANDLE_DO: while not found");
  }
  current -> secundary = STR_NULL;
}
```

Listing Handle_Do.C

Met *handle_switch* moet er iets meer gedaan worden dan bij de bovenstaande routines, maar in feite komt deze routine daar toch in grote lijnen mee overeen.

```
#include "struct.h"
#include "defs.h"

void handle_switch(current)
struct node *current;
{
  enum node_type keyword;
  struct node* p;

  /*
   *    Never a secundary node
   */
  current -> secundary = STR_NULL;
  keyword = handle_statement(current ->primary);
  switch (keyword) {
  case CASE:
      keyword = handle_case(current -> primary, CASE);
      break;
  case CASE_DEFAULT:
      keyword = handle_case(current -> primary, CASE_DEFAULT);
      break;
  case END:
      break;
  default:
      perror("HANDLE_SWITCH: wrong case\n");
      exit(-1);
      break;
  }
}
```

Listing Handle_Switch.C

4.10 Utilities

De *handle_case* routine is vrij gecompliceerd omdat er binnen een switch behoorlijk wat mogelijkheden zijn. De case behandelt op gestructureerde manier al deze gevallen.

```
#include "struct.h"
#include "defs.h"

enum node_type handle_case(current, keyword)
struct node *current;
enum node_type keyword;
{
  struct node* p, dummy;

  switch (keyword) {
  case END:
      return (END);
      break;
  case CASE:
      p = (struct node *)malloc(sizeof(struct node));
      p -> type = CASE;
      current -> primary = p;
      /*
       *    Search for next keyword (a case has always a primary)
       */
            keyword = handle_statement(current -> primary);
      if (keyword == CASE_BREAK) {
     /*
      *  Break found
                 */
     keyword = handle_case(current -> secundary, CASE_BREAK);
     current -> next = STR_NULL;
       } else if (keyword == CASE) {
     /*
      * No break found
      */
     keyword = handle_case(current -> next, CASE);
     current -> secundary = STR_NULL;
      } else if (keyword == CASE_DEFAULT) {
     /*
      * Default found
      */
      keyword = handle_case(current -> secundary, CASE_DEFAULT);
      current -> next = STR_NULL;
      } else if (keyword == END) {
     return (END);
      } else {
     printf("HANDLE_CASE: end not found within switch\n");
     exit(-1);
      }
      return (keyword);
      break;
```

Listing Handle_Case.C

```
    case CASE_DEFAULT:
        keyword = handle_statement(current -> secundary);
        return (keyword);
        break;
    case CASE_BREAK:
        keyword = handle_statement(dummy);
        if (keyword == END) {
       return (keyword);
        } else if (keyword == CASE) {
       keyword  =handle_case(current, CASE);
       return (keyword);
        } else {
       perror("HANDLE_CASE: not proper keyword found in case_break\n");
        }
        break;
    default:
        perror("HANDLE_CASE: wrong case found\n");
        exit(-1);
        break;
    }
}
```

Listing Handle_Case.C

De functie *create_node* creëert een nieuw blad voor boom, dit blad wordt op de heap met malloc toegewezen en wordt aan de vorige vast gemaakt.

```
#include "struct.h"

struct node *create_node(node_in, keyword)
struct node *node_in;    /* pointer to pointer to a node */
enum node_type keyword;
{
        struct node *p;
        char **n;

  p = (struct node *)malloc(sizeof(struct node));
  p -> type = keyword;
  node_in -> next  = p;
  return(p);
}
```

Listing Create_Node.C

Een handige functie tijdens het debuggen of om de werking van dit programma te volgen is *print_node_type*, te vinden op de volgende pagina

4.10 Utilities

```
#include "struct.h"

void print_node_type(s, keyword)
char *s;
enum node_type keyword;
{
  printf("%s keyword: ", s);
  switch (keyword) {
  case BEGIN:
      printf("BEGIN");
      break;
  case END:
      printf("END");
      break;
  case IF:
      printf("IF");
      break;
  case ELSE:
      printf("ELSE");
      break;
  case FOR:
      printf("FOR");
      break;
  case WHILE:
      printf("WHILE");
      break;
  case DO:
      printf("DO");
      break;
  case SWITCH:
      printf("SWITCH");
      break;
  case CASE:
      printf("CASE");
      break;
  case CASE_BREAK:
      printf("CASE_BREAK");
      break;
  case CASE_DEFAULT:
      printf("CASE_DEFAULT");
      break;
  default:
      perror("PRINT_NODE_TYPE: wrong case");
      break;
  }
  printf("\n");
}
```

Listing Print_Node.C

Dump

Dump is een eenvoudig file-dump programma, dat de hexadecimale waarde samen met de character representatie van een byte laat zien. De file-dumper is niet interactief, zodat het ook niet mogelijk is om voor- en achteruit door de file te lopen. De Proline C-implementatie heeft daar ook niet de mogelijkheid toe. Andere C-implementaties hebben in de standaard bibliotheek functies om *random* door files te kunnen 'lopen', zoals de functie lseek(). De synopsis van Dump is:

```
$ dump filenaam
```

Het werk wordt in de functie pr_line verzet. Pr_line drukt een regel, in een algemeen gebruikt formaat, af. Eerst wordt de positie van het eerste byte uit de regel afgedrukt. Daarna volgen 8 (CHAR_ON_LINE) hexadecimale waarden, de bytes die daarvoor uit de file worden gelezen, worden tevens in de array value geplaatst. Zodra de 8 bytes zijn afgedrukt worden de waarden van de array value gebruikt om de character representatie van deze bytes af te beelden. Aan het einde van de file wordt 'EOF' afgedrukt, wat overigens geen deel van de file uitmaakt die gedumpt wordt.

```
#include <stdio.h>

#define CHAR_ON_LINE 8

main(argc, argv)
int argc;
char *argv[];
{
     FILE *fin;

     fin = fopen(argv[1], "r");   /* open file */
     if (ferror(fin)) {
          printf("Error: %s cann't be opened\n", argv[1]);
          exit(1);
     }
     dump(fin);
     exit(0);
}

dump(fin)
FILE *fin;
{
     int ok;
/*
0000   01 02 03 04 05 06 07 08   12345678
 */
     do {
          ok = pr_line(fin);
     } while (ok);
}

pr_line(fin)
FILE *fin;
```

```
{
    static int count = 0;
    int value[CHAR_ON_LINE];
    int i, c;

    printf("%04X  ", count);
    for (i = 0; i < CHAR_ON_LINE; i++) {
        c = fgetc(fin);
        value[i] = c;
        if (c == EOF) {
            printf("EOF");
            for (; i < CHAR_ON_LINE - 1; i++) {
                printf("   ");
            }
            break;
        }
        printf("%02X ", c);
    }
    printf(" ");
    for (i = 0; i < CHAR_ON_LINE; i++) {
        if (value[i] == EOF) {
            printf("\n");
            return(0);
        }
        printf(isprint(value[i]) ? "%c" : ".", value[i]);
    }
    count += CHAR_ON_LINE;
    printf("\n");
    return(1);
}
```

Dos-functies

Deze listing bevat de Dos functies zoals we die ook al uit BASIC kennen. Al deze functies zijn erg systeem-afhankelijk, behalve de functie *rename*. Deze functie komt ook op andere systemen voor. De functies dos_copy, dos_scratch en dos_new zijn waarschijnlijk niet onder deze naam op andere systemen te vinden, maar kunnen daar zeker gemaakt worden.

```
dos_new(name, id)
char *name, *id;
{
    open(7, 8, 15); /* open command channel */
    fprintf(7, "n:%s,%s", name, id);
    close(7);
}

dos_copy(old_name, new_name)
char *old_name, *new_name;
{
    open(7, 8, 15); /* open command channel */
```

4.10 Utilities

```
    fprintf(7, "c:%s=0:%s", new_name, old_name);
    close(7);
}

dos_rename(old_name, new_name)
char *old_name, *new_name;
{
    open(7, 8, 15); /* open command channel */
    fprintf(7, "r:%s=0:%s", new_name, old_name);
    close(7);
}

rename(old_name, new_name)
char *old_name, *new_name;
{
    open(7, 8, 15); /* open command channel */
    fprintf(7, "r:%s=0:%s", new_name, old_name);
    close(7);
}

dos_scratch(name)
char *name;
{
    open(7, 8, 15); /* open command channel */
    fprintf(7, "s:%s", name);
    close(7);
}

dos_initialize()
{
    open(7, 8, 15); /* open command channel */
    fprintf(7, "i");
    close(7);
}

dos_validate()
{
    open(7, 8, 15); /* open command channel */
    fprintf(7, "v");
    close(7);
}
```

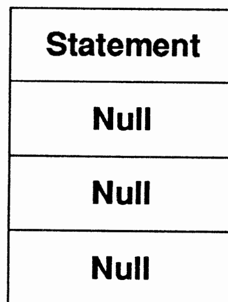
STRUCT (vervolg)

De plotter

Het tweede gedeelte van de Nassi-Schneider diagram generator beeld de boom die in het eerste gedeelte is gecreëerd grafisch uit. De bovengenoemde boom representeert de control flow van het programma, dat als invoer dient van struct. Voor dit gedeelte van struct, is deze boom één van de belangrijkste ingang variabelen. Laten we wat beter naar deze boom gaan kijken. Voor een eenvoudig programmaatje als:

```
main()
{
      printf("Hallo, wereld\n");
}
```

ziet de boom er simpel uit; zie figuur 1.



Figuur 1: simple statement

Omdat er geen volgend blad is, is deze pointer NULL. Er is ook geen primaire inhoud dus ook deze pointer is NULL. Omdat er tevens geen primaire inhoud is, is er automatisch geen secundaire inhoud en is ook deze pointer NULL.

Uit een iets ingewikkelder programmaatje zoals dit hieronder, volgt figuur 2.

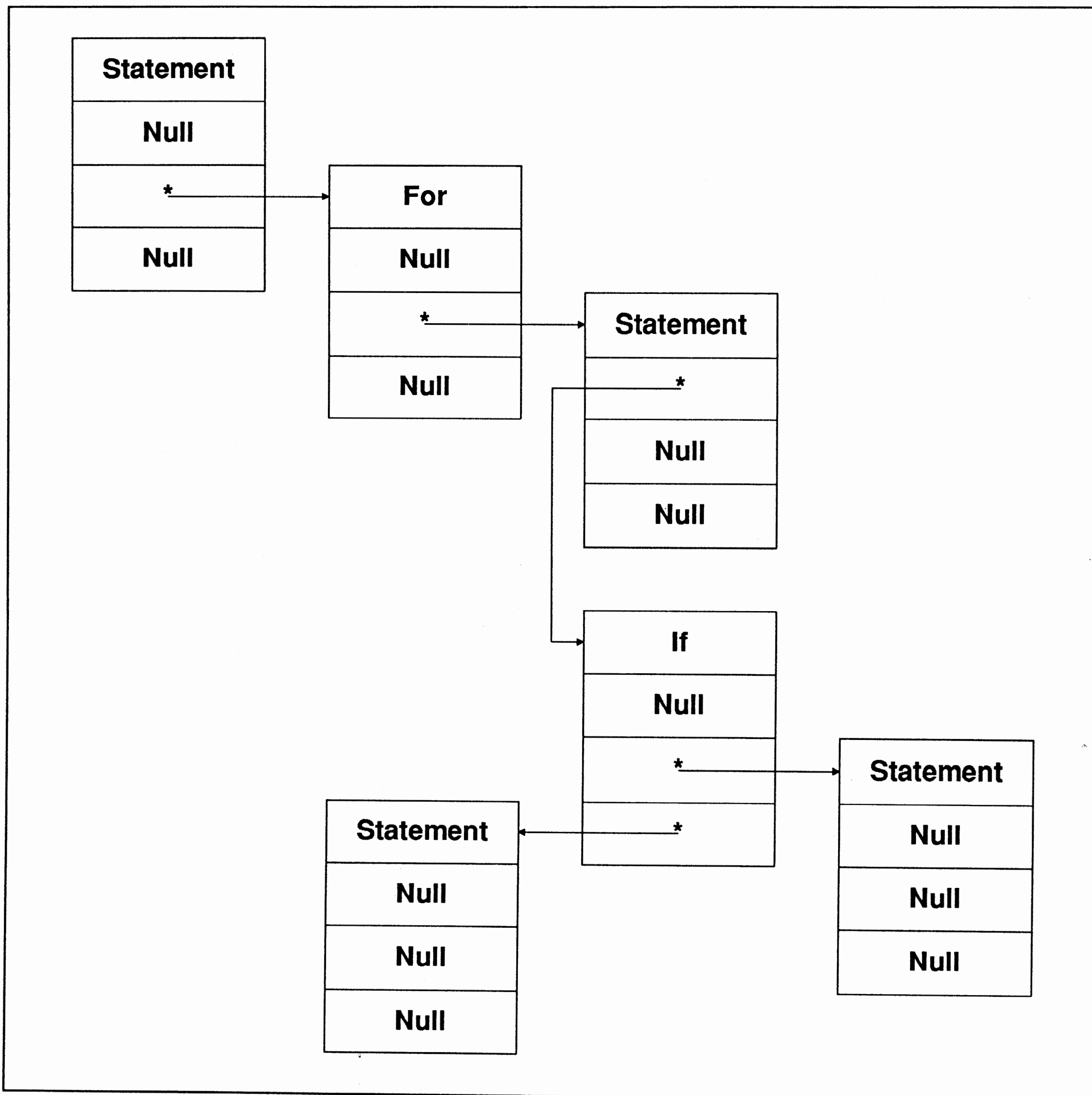
```
main()
{
   int i, a;
   for (i = 0; i < 10; i++) {
     a = i;
     if ((a % 2) == 0) {
       printf("%d is even\n",
                a);
     } else {
       printf("%d is oneven\n",
                a);
     }
   }
}
```

Deze direct al veel complexere boomstructuur, moet straks worden omgezet in een Nassi-Schneidermann diagram zoals in figuur 3 is afgebeeld.

In een laatste voorbeeld wordt gebruik gemaakt van het switch statement, waarvoor een eigen manier van afbeelden is gemaakt.

```
main()
{
   int i;

   scanf("%d", &i);
   switch(i) {
     case 1:
       printf("1\n");
       break;
     case 2:
       printf("2\n");
       break;
     case 3:
       printf("3\n");
     case 4:
       printf("3 & 4\n");
       break;
     case 5:

     case 6:
       printf("5 of 6\n");
```

4.10 Utilities



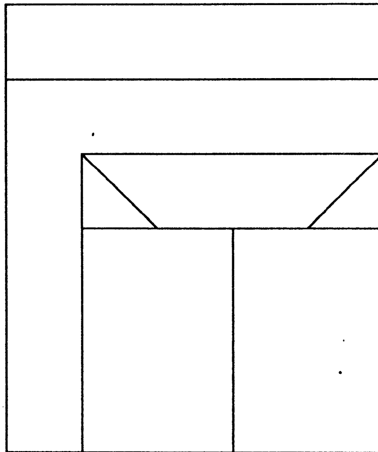
Figuur 2: if- en for-constructies

```
            break:
        default:
            printf("Iets anders\n);
            break;
    }
}
```

Is equivalent aan de onderstaande C code:

```
main()
{
   int i;
```

4.10 Utilities



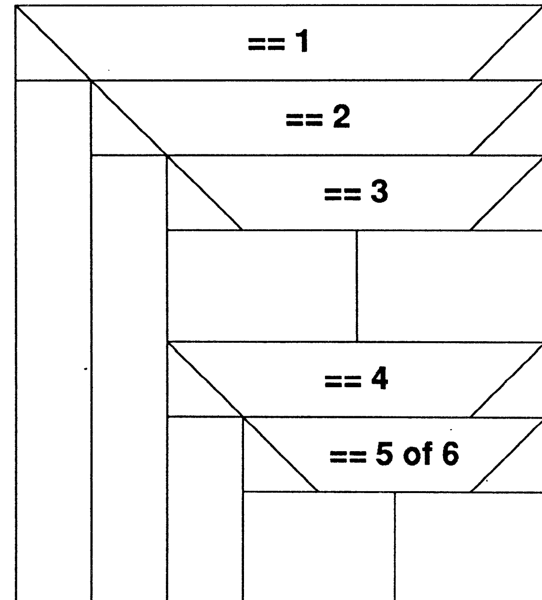
Figuur 3: N-S-diagram

```
    scanf("%d", &i);
    if (i == 1) {
      printf("1\n");
    } else if (i == 2) {
      printf("2\n");
    } else if (i == 3) {
      printf("3\n");
      if (i == 4) {
      printf("3 & 4\n");
      }
    } else if ((i==5) || (i==6)){
      printf("5 of 6\n");
    } else {
      printf("Iets anders\n");
    }
}
```

Deze codes worden beide op een gelijksoortige wijze door de structuurboom gerepresenteerd. Allen de switch vorm wordt in figuur 4 afgebeeld.

Het Nassi-Schneider diagram dat van deze boom gecreëerd wordt is te zien in figuur 5.

Nu duidelijk is wat het tweede deel van het programma moet doen, kunnen de routines die het werk moeten verrichten worden uit-



Figuur 5: N-S-diagram

gelegd. Een belangrijk onderdeel hierin vormt de include file: nassi.h. In deze file wordt een nieuwe data-structuur gedefinieerd en met een aantal macro definities voor constanten, die door de gebruiker zelf zijn te veranderen, om de Nassi-Schneider diagram uitvoer erna eigen voorkeur uit te laten zien.

De NASSI structuur heeft drie elementen. Met deze elementen wordt een diepte en breedte van een structuur aangegeven en tevens de inhoud van de structuur. Deze inhoud bestaat uit een depth aantal strings, die elk de lengte width hebben. Een structuur kan er bijvoorbeeld als in figuur 6 uitzien.

Deze structuur kan tijdens het runnen van het programma zichtbaar worden gemaakt met de functie print_object.

Met behulp van de functies print_tree en tree is het mogelijk om de invoer boom voor het tweede gedeelte zichtbaar te maken.

4.10 Utilities

Statement
*
Null
Null

Switch
*
Null
Null

Case_break
Null
*
*

Case_break
Null
*
*

Statement
Null
Null
Null

Statement
Null
Null
Null

Case
*
*
Null

Statement
Null
Null
Null

Case_break
Null
*
*

Case_break
Null
*
*

Statement
Null
Null
Null

Default
Null
Null
Null

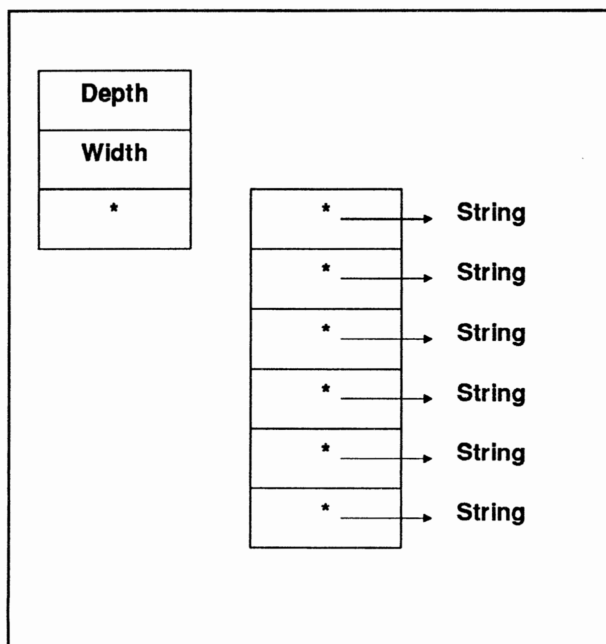
Statement
Null
Null
Null

Figuur 4: case-statement

4.10 Utilities

```
struct NASSI {
  short int depth;
  short int width;
  char **strings;
};

#define FOR_WIDTH  3
#define WHILE_WIDTH   FOR_WIDTH
#define DO_WIDTH   FOR_WIDTH
#define SWITCH_WIDTH  2
#define IF_WIDTH   1
#define STATEMENT_WIDTH 0

#define BASIC_FOR_DEPTH  5
#define BASIC_WHILE_DEPTH       BASIC_FOR_DEPTH
#define BASIC_DO_DEPTH          BASIC_FOR_DEPTH
#define BASIC_SWITCH_DEPTH 4
#define BASIC_IF_DEPTH   6
#define BASIC_STATEMENT_DEPTH 3

/* #define NULL_OBJECT  (struct NASSI *)0 */

#define MAX_BEGIN_STR   20
#define MAX_STR_WIDTH   80
```

Listing Nassi.H



Figuur 6: depth-elementen

De functies die met deze NASSI structuur manipuleren zullen hier onder eerst worden behandeld. Het NASSI-NULL object wordt binnen het programma veel gebruikt, de functie make_null_object creëert zo'n object.

De meeste van deze NASSI objecten bestaan slechts voor korte tijd, daarom bestaat er een functie object_free om de datastructuren die op de heap waren gealloceerd, weer vrij te geven.

Het plot gedeelte van het programma doorloopt de boven beschreven invoer boom en aangekomen aan de uiteinden, de zogenaamde "leafs" moeten er basis NASSI objecten worden gecreëerd. Deze basis NASSI objecten hebben alle een standaard vorm. Deze

4.10 Utilities

```
void print_tree(node)
struct node *node;
{
   tree(0, node);
}

void tree(count, node)
int count;
struct node *node;
{
    char s[20];
    int i;

    if (count > 20) {
        printf("TREE - count too big, something is wrong\n");
        exit();
    }
    strcpy(s, "");
    for (i = 0; i < count; i++) {
     strcat(s, " ");
    }
    print_node_type(s, node -> type);
#ifdef DEBUG
        printf("%s ->%d\n", s, node -> primary);
        printf("%s ->%d\n", s, node -> secundary);
        printf("%s ->%d\n", s, node -> next);
#endif
    if (node -> primary != STR_NULL) {
     tree(count + 1, node -> primary);
    }
    if (node -> secundary != STR_NULL) {
       tree(count + 1, node -> secundary);
    }
    if (node -> next != STR_NULL) {
     tree(count, node -> next);
    }
}

void print_object(object)
struct NASSI *object;
{
    int i;

    printf("PRINT_OBJECT - BEGIN \n");
    for (i = 0; i < object -> depth; i++) {
     printf("%s\n", object -> strings[i]);
    }
    printf("PRINT_OBJECT - END\n");
}
```

Listing Print.C

4.10 Utilities

```
#include "plot.h"
#include "nassi.h"
#include <stdlib.h>
#include <stdio.h>

struct NASSI *make_null_object()
{
    struct NASSI *p;

    p = malloc(sizeof(struct NASSI));
    p -> width = 0;
    p -> depth = 0;
    p -> strings = NULL;
    return(p);
}
```

Listing Mak_null.C

```
#include "nassi.h"
#include "plot.h"
#include <stdlib.h>
#include <stdio.h>

void object_free(object)
struct NASSI *object;
{
    int i;
    char *p;

#ifdef DEBUG1
    printf("OBJECT_FREE \n");
#endif
    for (i = 0; i < object -> depth; i++) {
  p = object -> strings[i];
  if (p != NULL) {
        free(p);
  }
    }
    if (object -> strings != NULL) {
     free(object -> strings);
    }
    free(object);
}
```

Listing Obj_free.C

standaard vorm kan al naar gelang behoefte groter worden gemaakt in de lengte als ook in de breedte. Al de basis vormen hebben echter ook minimale afmetingen waar rekening mee gehouden dient te worden. Zo is er voor elke vorm een minimale breedte, een

4.10 Utilities

```
#include "struct.h"
#include "nassi.h"

int min_node_width(current_node)
struct node *current_node;
{
     switch (current_node -> type) {
      case FOR:
           return(FOR_WIDTH);
           break;
      case WHILE:
           return(WHILE_WIDTH);
           break;
      case IF:
           return(IF_WIDTH);
           break;
      case SWITCH:
           return(SWITCH_WIDTH);
           break;
      case DO:
           return(DO_WIDTH);
           break;
           case STATEMENT:
                return(STATEMENT_WIDTH);
                break;
           case NOP:
                return(0);
                break;
           default:
                printf("MAKE_NASSI - Wrong node type\n");
                print_node_type(current_node->type);
                exit();
                break;
      }
}
```

Listing Node_wid.C

minimale header diepte en een minimale basis diepte. De minimale breedte wordt bepaald door de functie min_node_width.

Onder de minimale header diepte van een basis Nassi-Schneidermann object wordt het aantal regels verstaan dat nodig is om de bovenkant van de vorm, waar de test of conditie staat, in af te beelden. Dit wordt bepaald door de functie object_header_depth.

Onder de minimale basis diepte wordt het aantal regel verstaan dat nodig is de onderkant van de vorm, waar de statements staan, in af te beelden. Dit wordt bepaald door de functie basic_depth.

Om te weten hoe breed al de Nassi-Schneider basis vormen moeten worden in het uiteindelijke Nassi-Schneider diagram, wordt er eerst een schatting gemaakt van de breed-

4.10 Utilities

```
short int object_header_depth(type)
enum node_type type;
{
    switch (type) {
     case WHILE:
        case DO:
        case FOR:
            return(2);
        case IF:
            return(4);
  case STATEMENT:
      return(0);
        default:
            printf("OBJECT_DEPTH - wrong input type\n");
            print_node_type("node type is:", type);
            exit();
    }
}
```

Listing Obj_dpth.C

```
#include "nassi.h"
#include "plot.h"
#include "struct.h"
#include <stdio.h>

void make_basic_nassi(type, width, object)
enum node_type type;
short int width;
struct NASSI *object;
{
     int i, j;
     char *p;

     printf("MAKE_BASIC_OBJECT - width: %d\n", width);
     print_node_type("                  - type:", type);
     object -> strings = (char **)malloc(sizeof(char *) * basic_depth(type));
     for (i = 0; i < basic_depth(type); i++) {
         object -> strings[i] = NULL;
     }
     for (i = 0; i < basic_depth(type); i++) {
        object -> strings[i] = (char *)malloc(width + 1);
     }
     object -> width = width;
     object -> depth = basic_depth(type);
     switch (type) {
        case STATEMENT:
```

Listing Make_bas.C (deel 1)

4.10 Utilities

```
            /*
             *   First line
             */
            p = object -> strings[0];
            *p++ = '+';
            for (i = 1; i < (width - 1); i++) {
        *p++ = '-';
         }
            *p++ = '+';
            *p = '\0';
            /*
             *   Next lines
             */
            for (j = 1; j < basic_depth(type) - 1; j++) {
             p = object -> strings[j];
                *p++ = '|';
                for (i = 1; i < width -1; i++) {
                    *p++ = ' ';
                }
                *p++ = '|';
                *p = '\0';
            }
            /*
             *   Last line
             */
            p = object -> strings[basic_depth(type) - 1];
            *p++ = '+';
            for (i = 1; i < (width - 1); i++) {
        *p++ = '-';
         }
            *p++ = '+';
            *p = '\0';
            break;
        default:
            printf("MAKE_BASIC_OBJECT - wrong type\n");
            print_node_type("Type is: " ,type);
            exit();
    }
}
```

Listing Make_bas.C (deel 2)

te die elke vorm later gaat hebben. Een vrij cryptische functie estimate_width berekent het maximale aantal basis vormen wat in een deel van de invoer boom naast elkaar staan en de daarvoor minimaal benodigde ruimte. Deze functie is haast van zelf sprekend recursief, zodat informatie over linkerkant en van de rechter kant van de subboom op eenvoudige wijze verkregen kan worden. Dit moet ook voor het vervolg, het next gedeelte, van deze subboom worden gedaan, en voor diens next gedeelte, enzovoorts. Het

4.10 Utilities

maximum van al die op één volgende delen is het resultaat van deze functie.

In estimate_width wordt gebruik gemaakt van een functie number_of_boxes. Deze functie heeft als resultaat het aantal naast elkaar staande basis vormen voor een subboom.

```
#include "nassi.h"
#include "defs.h"
#include "struct.h"
#include "plot.h"

void estimate_width(current_node, min_recuired_width, max_boxes)
struct node *current_node;
short int *min_recuired_width,
    *max_boxes;
{
    short int width, boxes;
    short int prim_recuired_width, prim_max_boxes;
    short int sec_recuired_width, sec_max_boxes;

#ifdef DEBUG
    if (current_node == STR_NULL) {
        printf("ESTIMATE_WIDTH node is NULL\n");
    } else {
        print_node_type("ESTIMATE_WIDTH node_type:", current_node ->type);
    }
#endif
    *max_boxes = 0;
    *min_recuired_width = 0;
    while (current_node != STR_NULL) {
        width = min_node_width(current_node);
        boxes = number_of_boxes(current_node);
        estimate_width(current_node -> primary, &prim_recuired_width,
          &prim_max_boxes);
        estimate_width(current_node -> secundary, &sec_recuired_width,
          &sec_max_boxes);
  *min_recuired_width = ((*min_recuired_width >
     (width + prim_recuired_width + sec_recuired_width)) ?
     *min_recuired_width :
     (width + prim_recuired_width + sec_recuired_width));
        *max_boxes = ((*max_boxes >
          (boxes + prim_max_boxes + sec_max_boxes)) ? *max_boxes :
          (boxes + prim_max_boxes + sec_max_boxes));
        current_node = current_node -> next;
    }
}
```

Listing Est_wdth.C

4.10 Utilities

```
#include "nassi.h"
#include "defs.h"
#include "struct.h"
#include "plot.h"

short int number_of_boxes(current_node)
struct node *current_node;
{
    switch (current_node -> type) {
        case NOP:
            return(0);
        case WHILE:
        case DO:
        case FOR:
        case STATEMENT:
            return(number_of_boxes(current_node -> primary));
        case IF:
            return(2 + number_of_boxes(current_node -> primary) +
                    number_of_boxes(current_node -> secundary));
        default:
            printf("NUMBER_OF_BOXES - switch error\n");
            print_node_type("Type is:", current_node);
            exit();
    }
}
```

Listing Num_box.C

8/4.11

De standaard bibliotheek

De standaard bibliotheek, die ook wel de "run-time library" wordt genoemd, bestaat uit een groot aantal functies. De standaard bibliotheken, die bij andere C systemen horen bevatten over het algemeen meer functies, dan de bibliotheek van het Proline systeem. Maar de verzameling die Proline gemaakt heeft is toch zeer compleet te noemen. De niet opgenomen functies kunnen over het algemeen eenvoudig met behulp van de reeds bestaande functies worden samengesteld.

De Proline standaard bibliotheek is onder te verdelen in 7 secties, te weten:
1) Character behandelings functies.
2) String behandelings functies.
3) Mathematische functies.
4) Geheugenbeheersfuncties.
5) standaard in- en uitvoer.
6) Overige standaard functies.
7) Commodore functies.

Deze functies zullen hieronder steeds op een zelfde manier beschreven worden. Eerst komt de naam van de functie in hoofdletters. Daarna komt de synopsis, ofwel de definitie van de interface, zoals de gebruiker die functie kan gebruiken. Als laatste volgt een korte beschrijving van de werking van de functie, met in enkele gevallen een voorbeeld. Bijvoorbeeld:

```
STRLEN
# include <string.h>
int strlen(s)
   char *s;
```

strlen retourneert de lengte van een string, dit is het aantal tekens, dat voor het NULL character van string s, voorafgaat.

De functie strlen heeft één parameter, dit is een pointer naar een character. Dit character zal een onderdeel van een string zijn (anders is dit een zinloze operatie). De functie retourneert een waarde van het type int. De file waarin de aanroep naar de functie strlen in is opgenomen, moet voordat deze functie wordt aangeroepen, de header file string.h includen. Met het preprocessor statement "# include <string.h>" wordt dit gedaan.

8/4.11.1
Character behandelings functies

De character behandelings functies dienen voor de klassificatie van een character. Sommige standaard bibliotheken bevatten ook character conversie functies, maar bij de PROLINE compiler zijn deze functies niet opgenomen. Deze functies zullen hier dan ook niet behandeld worden.

4.11 De standaard bibliotheek

ISALPHA
int isalpha(c)
  char c;

isalpha retourneert een waarde ongelijk aan NULL, als c een element van het alfabet is, een van:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z

ISASCII
int isascii(c)
  char c;

isascii retourneert een waarde ongelijk aan NULL, als c een waarde heeft die kleiner is dan 0200 (De grootte van de standaard ASCII character set), anders NULL.

ISCNTRL
int iscntrl(c)
  char c;

iscntrl retourneert een waarde ongelijk aan NULL, als c de code van een "control character" heeft.

ISDIGIT
int isdigit(c)
  char c;

isdigit retourneert een waarde ongelijk aan NULL, als c een decimaal cijfer is, een van:

0 1 2 3 4 5 6 7 8 9

ISLOWER
int islower(c)
  char c;

islower retourneert een waarde ongelijk aan NULL, als c een element uit het "kleine letter" alfabet is, een van:

a b c d e f g h i j k l m n o p q r s t u v w x y z

ISPRINT
int isprint(c)
  char c;

isprint retourneert een waarde ongelijk aan NULL, als c een afdrukbaar teken is, dit is elk ASCII teken met uitzondering van de control characters: de spatie en een van:

! " # $ % & ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z [\ ] ^ _
` a b c d e f g h i j k l m n o
p q r s t u v w x y z { | } ~

ISPUCNT
int ispunct(c)
  char c;

ispunct retourneert een waarde ongelijk aan NULL, als c een afdrukbaar teken is, mits het geen alfanumeriek of control character is. Deze tekens zijn de spatie en de onderstaande tekens.

! " # $ % & ' ( ) * + , - . / :
; < = > ? @ [\ ] ^ _ ` { | } ~

ISSPACE
int isspace(c)
  char c;

isspace retourneert een waarde ongelijk aan NULL, als c een blanco teken is. Anders wordt de waarde NULL geretourneerd.

ISUPPER

```
int isupper(c)
   char c;
```

isupper retourneert een waarde ongelijk aan NULL, als c een element is van het hoofdletter alfabet, een van:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

8/4.11.2
String behandelings functie

Per definitie hebben strings in C een variabele lengte en eindigen ze op het NULL character. Dit NULL character wordt automatisch door de computer toegevoegd achter alle string-constanten. Maar het is de verantwoordelijkheid van de programmeur om zich er van te vergewissen dat zelfgemaakte strings, in character arrays, op het NULL character eindigen.

Alle standaard bibliotheek functies, die hier worden beschreven, gaan er vanuit dat strings met het NULL character eindigen. Wanneer bij een bepaalde functie strings verplaatst worden, moet de programmeur er voor zorgen dat er in de "destination string" (de string waar het resultaat in komt te staan) genoeg ruimte is, want er wordt door de functies geen overflow test uitgevoerd. Bij de te reserveren ruimte dient rekening gehouden te worden, met de extra byte die het NULL character inneemt.

ATOI

```
int atoi(iptr)
   char *iptr;
```

atoi converteert de string iptr naar een integer, dit is de geretourneerde waarde van de functie.

ATOF

```
float atof(fptr)
   char *fptr;
```

atof converteert de string fptr naar een float, dit is de geretourneerde waarde van de functie.

INDEX

```
# include <string.h>
char *index(s, c)
   char *s, c;
```

index retourneert een pointer naar de eerste plaats waar het teken c in de string s voorkomt. Als het teken c niet in de string voorkomt wordt NULL geretourneerd.

RINDEX

```
# include <string.h>
char *rindex (s, c)
   char *s, c;
```

rindex retourneert een pointer naar de laatste plaats waar het teken c in de string s voorkomt. Als het teken c niet in de string voorkomt wordt NULL geretourneerd.

STRCAT

```
# include <string.h>
char *strcat(s1, s2)
   char *s1, *s2;
```

strcat plaatst de inhoud van de string s2 achter string s1. Een pointer naar het eerste teken van s1 wordt geretourneerd.

STRCMP

```
# include <string.h>
int strcmp(s1, s2)
   char *s1, *s2;
```

strcmp vergelijkt de inhoud van string s1 met de inhoud van s2 op lexicografische wijze. De functie retourneert een integerwaarde kleiner dan nul, als s1 kleiner dan s2 is; gelijk aan nul, als s1 gelijk is aan s2; groter dan nul, als s1 groter is dan s2.

Beide strings zijn gelijk, als ze dezelfde lengte hebben en op elke overeenkomende positie een zelfde teken hebben staan. De string s1 is lexicografisch groter dan de string s2, als aan een van de onderstaande voorwaarden is voldaan.

1) De inhoud van de strings zijn gelijk tot op een zekere positie en het teken dat daarop volgt uit de verschillende strings is bij string s1 groter dan bij string s2.

2) De string s1 is langer dan string s2, en de overeenkomende inhoud is gelijk.

STRCPY
```
# include <string.h>
char *strcpy(s1, s2)
   char *s1, *s2;
```

strcpy kopieert de inhoud van string s2 naar string s1. Een pointer naar het eerste teken van s1 wordt geretourneerd.

STRLEN
```
# include <string.h>
int strlen(s)
   char *s;
```

strlen retourneert de lengte van een string. Dit is het aantal tekens van string s, dat aan het NULL character, voorafgaat.

STRNCAT
```
# include <string.h>
char *strncat(s1, s2, n)
   char *s1, *s2;
   int n;
```

strncat voegt n tekens uit string s2 toe aan het einde van string s1. Een pointer naar het begin van s1 wordt geretourneerd.

STRNCMP
```
# include <string.h>
int strncmp(s1, s2, n)
   char *s1, *s2;
   int n;
```

strncmp vergelijkt lexicografisch de eerste n tekens uit string s1 met die van string s2. De functie retourneert een waarde kleiner dan NULL, als s1 kleiner is dan s2; een waarde gelijk aan NULL, als s1 is gelijk aan s2; een waarde groter dan NULL, als s1 is groter dan s2. Zie ook strcmp.

STRNCPY
```
# include <stdio.h>
char *strncpy(s1, s2, n)
   char *s1, *s2;
   int n;
```

strncpy kopieert n tekens van string s2 naar string s1. Als er minder dan n tekens in s2 zijn, worden er NULL characters in s1 geschreven, achter de niet NULL characters, totdat er in totaal n tekens zijn geschreven.

8/4.11.3 Mathematische functies

De standaard bibliotheek ondersteunt een zeer complete verzameling van mathematische functies.

ABS
```
# include <math.h>
int abs(i)
   int i;
```

abs retourneert de absolute waarde van de integer i. Als i positief is wordt i zelf geretourneerd, als i negatief is wordt de waarde -i geretourneerd.

ACOS

```
# include <math.h>
float acos(x)
  float x;
```

acos retourneert de arccosinus van x. Het resultaat ligt tussen 0 en pi. De waarde van x mag niet kleiner dan -1.0 of groter dan 1.0 zijn.

ASIN

```
# include <math.h>
float asin(x)
  float x;
```

asin retourneert de arcsinus van x. Het resultaat ligt tussen -pi/2 en pi/2. De waarde van x mag niet kleiner dan -1.0 of groter dan 1.0 zijn.

ATAN

```
# include <math.h>
float atan(x)
  float x;
```

atan retourneert de arctangens van x. Het resultaat ligt tussen -pi/2 en pi/2.

ATAN2

```
# include <math.h>
float atan2(x, y)
  float x, y;
```

atan2 retourneert de arctangens van x/y. Deze functie is van belang bij de conversie van het normale coördinaten stelsel naar het polaire of Cartesische stelsel. Het resultaat van atan2 is de hoek tussen de positieve x-as en de lijn, die door het punt (0,0) en het punt (x,y) getrokken wordt. Het resultaat ligt tussen -pi en pi.

CABS

```
# include <math.h>
float cabs(c)
  struct{
    float x, y;
  } *c;
```

cabs retourneert de hypotenusa van c. De hypotenusa wordt gedefinieerd als: sqrt (x * x + y * y).

CEIL

```
# include <math.h>
float ceil(x)
  float x;
```

ceil retourneert de kleinste integer, die niet kleiner is dan x.

COS

```
# include <math.h>
float cos(x)
  float x;
```

cos retourneert de cosinus van x. Het resultaat ligt tussen -1.0 en 1.0

COSH

```
# include <math.h>
float cosh(x)
  float x;
```

cosh retourneert de cosinus hyperbolicus van x. De wiskundige formule voor cosh(x) is:

$$\cosh x = \frac{e^{x} + e^{-x}}{2}$$

EXP
```
# include <math.h>
float exp(x)
   float x;
```

exp retourneert de exponentiële macht van x, beter bekend als e^x.

FABS
```
# include <math.h>
float fabs(x)
   float x;
```

fabs retourneert de absolute waarde van x.

FLOOR
```
# include <math.h>
float floor(x)
   float x;
```

floor retourneert de grootste integerwaarde niet groter dan x.

HYPOT
```
# include <math.h>
float hypot(x, y)
   float x, y;
```

De functie hypot retourneert de hypotenusa van x en y. De hypotenusa wordt gedefinieerd als: sqrt(x *x + y * y).

FREXP
```
# include <math.h>
float frexp(x, ptr)
   float x;
   int *ptr;
```

frexp splitst x in een mantissa, waarvan de waarde tussen 1.0 en 0.5 ligt, en een exponent, zodat geldt: x = mantissa * $2^{exponent}$. De mantissa is de geretourneerde waarde van de functie, de exponent is een zijeffect van de functie, en wordt op de plaats, die door ptr wordt aangegeven, opgeslagen.

LDEXP
```
# include <math.h>
float ldexp(x, n)
   float x;
   int n;
```

ldexp retourneert x * 2^n.

LOG
```
# include <math.h>
float log(x)
   float x;
```

log retourneert de natuurlijke logarithme van x.

LOG10
```
# include <math.h>
float log10(x)
   float x;
```

log10 retourneert de logarithme van de 10-macht, van x.

MODF
```
# include <math.h>
float modf(x, ptr)
   float x, *ptr;
```

modf splitst x in een fractioneel deel (f) en een integer deel (n). Het fractionele deel heeft een positieve waarde, tussen 0.0 en 1.0, en wel zodanig dat geldt: x = f + n. De functie retourneert het fractionele deel, het integer deel wordt op het adres dat door ptr wordt aangegeven geplaatst.

POW
```
# include <math.h>
float pow(x, y)
   float x, y;
```

pow berekent de y-macht van x en retourneert dus x^y.

RANDOM
int random ()

random retourneert een integer, die ligt in het gebied van 0 tot en met de grootste integer. Deze waarde is gegenereerd met een pseudorandom generator.

SIN
include <math.h>
float sin(x)
  float x;

sin retourneert de sinus van x. Het resultaat ligt tussen -1.0 en 1.0

SINH
include <math.h>
float sinh(x)
  float x;

sinh retourneert de sinus hyperbolicus van x. De wiskundige formule voor sinh(x) is:

$$\sinh x = \frac{e^x - e^{-x}}{2}$$

SQRT
include <math.h>
float sqrt(x)
  float x;

sqrt retourneert de wortel van x.

SRANDOM
srandow(n)
  int n;

srandom wordt gebruikt om de pseudorandom generator te initialiseren. Als srandom twee maal met dezelfde waarde wordt geïnitialiseerd, worden dezelfde pseudorandom getallen gegenereerd.

TAN
include <math.h>
float tan(x)
  float x;

tan retourneert de tangens van x.

TANH
include <math.h>
float tanh(x)
  float x;

tanh retourneert de tangens hyperbolicus van x. De wiskundige formule voor tanh(x) is:

$$\tanh x = \frac{\cosh x}{\sinh x}$$

8/4.11.4
Geheugenbeheersfuncties

Geheugenbeheersfuncties hebben allemaal betrekking op de "heap" van de computer. De heap is een geheugengebied dat direct voor een bepaalde hoeveelheid vrije ruimte kan zorgen. Deze ruimte kan voor velerlei doeleinden gebruikt worden, en als een bepaalde ruimte wordt teruggegeven, kan deze ruimte weer worden hergebruikt. Deze laatst genoemde faciliteit van de zo genoemde heap manager is onzichtbaar voor de gebruiker.

CALLOC
char *calloc(nelem, elsize)
  unsigned nelem, elsize;

calloc retourneert een pointer naar een blok geheugen, wat gevuld is met nullen, dat op zijn minst nelem * elsize bytes bevat.

FREE

```
free(ptr)
   char*ptr;
```

free geeft het blok geheugen, dat door ptr wordt aangewezen, terug aan de heap. De pointer ptr moet naar een geheugenblok wijzen, dat door malloc, calloc of realloc is vrijgegeven.

MALLOC

```
char*malloc(size)
   unsigned size;
```

malloc retourneert een pointer naar een blok geheugen, dat op zijn minst size bytes bevat.

REALLOC

```
char *realloc(ptr, size)
   char *ptr;
   unsigned size;
```

realloc kopieert het geheugenblok, dat door ptr wordt aangewezen, naar een geheugenblok van tenminste size bytes. De pointer naar dit nieuwe geheugenblok is de return-waarde van de functie. De pointer ptr moet naar een geheugenblok wijzen, dat door malloc, calloc of realloc is vrijgegeven.

8/4.11.5
Standaard in- en uitvoer

Tot de standaard in- en uitvoerfuncties behoren al die functies, die betrekking hebben op het schrijven naar het beeldscherm, meestal standaard uitvoer (stdout), het lezen van het toetsenbord (stdin) en het lezen of schrijven naar disk.

EOF

```
# include <stdio.h>
# define EOF  -1
```

De macro EOF wordt gebruikt om het einde van de file aan te geven. EOF heeft gewoonlijk een waarde, die niet als waarde voor een character wordt gebruikt, in deze implementatie is dat -1.

FILE

```
# include <stdio.h>
typedef int FILE;
```

Het datatype FILE wordt gebruikt om informatie over een file of meer algemeen een stream, op te slaan. Een stream is meestal een file, maar kan ook een andere vorm van data zijn, waaronder ook het toetsenbord en het beeldscherm gerekend worden.

FCLOSE

```
# include <stdio.h>
fclose(stream)
   FILE stream;
```

De functie fclose sluit een stream. Deze stream zal geopend moeten zijn voor invoer of uitvoer.

FEOF

```
# include <stdio.h>
int feof (stream)
   FILE stream;
```

De functie feof detecteert of het einde van de stream is bereikt tijdens het lezen daarvan. Als het einde is bereikt retourneert de functie een waarde ongelijk aan NULL, anders wordt de waarde NULL geretour-

4.11 De standaard bibliotheek

neerd.

FERROR
include <stdio.h>
int ferror()

ferror retourneert een waarde ongelijk aan nul, als er tijdens de laatste diskoperatie een fout is opgetreden.

FGETC
include <stdio.h>
int getc(stream)
  FILE stream;

getc retourneert een teken van de gespecificeerde stream. De functie retourneert EOF als het einde van de stream is bereikt.

FGETS
include <stdio.h>
char *fgets(s, n, stream)
  char *s
  int n;
  FILE stream;

De functie fgets leest een string van de gespecificeerde stream. De string begint op het adres dat door de pointer s wordt aangegeven. Er worden maximaal n-1 character van de stream gelezen, mits er geen newline teken in de stream voorkomt. Als er een newline in de stream voorkomt is het newline-teken het laatste teken dat in de string wordt geplaatst. Achter het laatste teken uit de string wordt altijd een NULL geplaatst. De functie retourneert s bij de normale werking of NULL als het einde van de stream wordt bereikt.

FOPEN
include <stdio.h>
FILE fopen(filename, mode)
  char *filename, *mode;

fopen opent een file op de disk voor lezen of schrijven. De string filename bevat de naam van de file. Het eerste teken van de stringmode specificeert lezen of schrijven, respectievelijk 'r' of 'w'. De default file is sequentieel, maar een program file mag ook worden geopend. De functie retourneert een file nummer, wat van dit moment af als een stream wordt beschouwd. Dit stream nummer kan later weer gebruikt worden bij in- en uitvoeropdrachten. De functie retourneert NULL als de file niet geopend kon worden.

Voorbeelden
include <stdio.h>

FILE f;

/* open een sequentiële file voor schrijven */
f = fopen("test", "w");

/* open een program file voor lezen */
f = fopen("test,p", "r");

/* open en vervang een sequentiële file */
f = fopen("@ 0:test", "w");

FPRINTF
# include <stdio.h>
fprintf(stream, format, arg1, arg2,...)
  FILE stream;
  char *format;

fprint is een uitvoerfunctie, de uitvoer wordt naar een stream gestuurd. De format string bepaalt hoe de uitvoer er uitziet, tevens bepaalt deze string het aantal argumenten wat nog volgt in de lijst van argumenten. Deze functie werkt hetzelfde als printf, het enige verschil is dat fprintf de uitvoer naar een gespecificeerde stream stuurt, terwijl printf de uitvoer altijd naar de standaard uitvoer stuurt. Zie

printf voor de beschrijving van de formattering operaties.

FPUTC
```
# include <stdio.h>
fputc(c, stream)
   char c;
   FILE stream;
```

fputc schrijft het character c in de gespecificeerde stream, die voor uitvoer geopend moet zijn.

FPUTS
```
# include <stdio.h>
fputs(s, stream)
   char *s;
   FILE stream;
```

fputs schrijft de string s in de gespecificeerde stream, die voor uitvoer geopend moet zijn.

FREAD
```
# include <stdio.h>
fread(ptr, elsize, nelem, stream);
   char *ptr;
   int elsize, nelem
   FILE stream;
```

fread leest een array, die nelem elementen bevat, die elk van grootte elsize zijn en slaat dit op, vanaf het adres dat door ptr wordt aangegeven. De array wordt uit de gespecificeerde stream gelezen.

FREOPEN
```
# include <stdio.h>
FILE freopen(filename, mode, stream)
   char *filename, *mode:
   FILE stream;
```

freopen opent een file zoals fopen dit doet. Maar met dit verschil dat eerst de file stream wordt gesloten, vervolgens wordt de file filename geopend waaraan de oude stream wordt toegekend.
Het belangrijkste doel van deze functie is het omleiden van de in- en uitvoer streams: stdin en stdout.

Voorbeeld:
```
# include <stdio.h>

FILE f;

/* leidt de standaard uitvoer om naar een
file op disk */
f = freopen ("disk-uitvoer", "w", stdout);
```

FSCANF
```
# include <stdio.h>
int fscanf(stream, control, arg1, arg2, ...)
   FILE stream;
   char *control;
```

fscanf leest tekens uit de stream, het voert conversies uit zoals die in de controlstring staan beschreven en plaatst het resultaat in de juiste argumenten.
Zie scanf voor een beschrijving van de controlstring.

FWRITE
```
# include <stdio.h>
fwrite(ptr, elsize, nelem, stream)
   char *ptr;
   int elsize, nelem;
   FILE stream;
```

fwrite schrijft een array, die nelem elementen bevat, elk van de grootte elsize naar de stream. De array begint op het adres dat door ptr wordt aangegeven.

GETC
```
# include <stdio.h>
int getc(stream)
```

4.11 De standaard bibliotheek

FILE stream;

De functie getc leest een character van de gespecificeerde stream, en retourneert dit character. De functie retourneert EOF als het einde van de file is bereikt.

GETCHAR
include <stdio.h>
int getchar()

getchar leest een character van de standaard invoer. De functie retourneert EOF als het einde van de file is bereikt.

GETS
include <stdio.h>
char *gets(s)
char *s;

De functie gets leest characters van de standaard invoer totdat een newline wordt gelezen. De newline wordt vervangen door NULL. Deze characterstring begint bij s. Van s wordt verwacht dat het naar het begin van een character array wijst, waarin genoeg ruimte is om de binnen te lezen string op te slaan.

GETW
include <stdio.h>
int getw(stream)
FILE stream;

De functie getw retourneert een integer waarde (twee bytes) uit de gespecificeerde stream. De functie retourneert EOF als het einde van de file is bereikt.

PRINTF
# include <stdio.h>
printf(control, arg1, arg2, ...)
char *control:

De functie printf voert een optionele lijst van argumenten uit naar de standaard uitvoer, in overeenstemming met het formaat, dat in de controlstring is gespecificeerd.

De control string mag bestaan uit gewone tekens, die rechtstreeks in de uitvoer verschijnen, en conversie operatoren, die specificeren hoe een argument moet worden uitgevoerd. Elke conversie operator begint met het percentage teken "%" en wordt gevolgd door:

* Een optioneel min-teken "-", dat aangeeft dat het argument links gejustificeerd moet worden in het uitvoerveld. Normaal wordt er rechts gejustificeerd.

* Een optioneel nummer, dat de minimale breedte van het uitvoerveld bepaalt. Een geconverteerd argument zal niet worden afgebroken, ook al past het niet in het gespecificeerde uitvoerveld. Als het eerste cijfer van het nummer een 0 is, wordt het uitvoerveld aan de voorkant opgevuld met nullen, anders zal het aan de voorkant opgevuld worden met spaties. De maximum breedte van het uitvoerveld is 128.

* Een optionele punt "." gevolgd door een nummer, geeft de precisie van een floating point of string-argument aan. Bij een floating point geeft de precisie aan, hoeveel cijfers er achter de decimale punt moeten worden afgedrukt (de default waarde is 6). Als de precisie expliciet op 0 wordt gezet, wordt er geen decimale punt afgedrukt. Bij strings geeft de precisie het aantal af te drukken tekens aan (default wordt de gehele string afgedrukt).

* Een letter, die aangeeft welk con-

4.11 De standaard bibliotheek

versietype moet worden uitgevoerd.

+ d - een integer argument wordt afgedrukt, dit is een "signed integer".

+ u - een integer argument wordt afgedrukt, dit is een "unsigned integer".

+ o - een integer argument wordt afgedrukt, als een octaal getal.

+ x - een integer argument wordt afgedrukt, als een hexadecimal getal.

+ f - een float argument wordt afgedrukt.

+ s - een character pointer argument, waarvan verwacht wordt, dat deze naar een string wijst, wordt afgedrukt.

+ c - een integer argument wordt als een character afgedrukt.

* Bij elk conversie commando behoort een corresponderend argument, wat van het juiste type moet zijn.

* Om een procent teken af te drukken, moet "%%" worden gebruikt.

* Het vermenigvuldigingsteken "*" kan gebruikt worden om de uitvoerveldbreedte of de precisie met behulp van een argument te laten variëren. De waarde zal dan dus van een integer argument worden genomen.

Voorbeelden

```
print ("%s", "Dit is C");
/* uitvoer: Dit is C */

printf ("%d %f", 123, 3.45);
/* uitvoer: 123, 3.450000 */

printf ("05o", 0678);
/* uitvoer: 00678 */

printf ("abc%-*.*fxyz", 5, 2, 12.3456);
/* uitvoer: abc12.34 xyz */
```

PUTC

```
# include <stdio.h>
putc(c, stream)
   char c;
   FILE *stream;
```

putc schrijft het character c naar de gespecificeerde stream.

PUTCHAR

```
# include <stdio.h>
putchar(c)
   char c;
```

putchar schrijft het character c naar de standaard uitvoer.

PUTS

```
# include <stdio.h>
puts(s)
   char *s;
```

puts schrijft de string s naar de standaard uitvoer.

PUTW

```
# include <stdio.h>
putw(i, stream)
   FILE stream;
```

putw schrijft een integer (twee bytes) naar de gespecificeerde stream.

SCANF

```
# include <stdio.h>
scnaf(control, arg1, arg2, ...)
```

Deze functie leest characters en conver-

4.11 De standaard bibliotheek

teert ze, zoals gespecificeerd is in de controlstring. De geconverteerde waarden worden indirect via de pointer-argumenten geretourneerd.

De controlstring mag spaties en newlines bevatten, die overeen mogen komen met eventuele spaties en newlines uit de invoer. Ook mag de controlstring naast de conversie-operatoren gewone tekens bevatten, deze moeten overeenkomen met corresponderende tekens uit de invoer. Elke conversie-operator begint met het procent-teken "%" en wordt gevolgd door:

* Een optioneel vermenigvuldigings teken "*", dat voorkomt dat de geconverteerde waarde aan een argument wordt toegekend.

* Een optioneel nummer, dat de maximale grootte van het invoerveld specificeert. Tekens worden gelezen tot het eerstvolgende teken dat niet wordt herkend, voor de conversie die op het moment plaatsvindt, of tot het gelezen aantal tekens gelijk is aan de maximale grootte van het invoerveld. Als er geen invoerveld-grootte is gespecificeerd, dan worden de tekens gelezen tot het eerste niet herkenbare teken, voor de huidige conversie.

* Een letter, die het type van conversie, dat op een bepaalde plaats vindt, bepaalt.

+ d - er wordt op het invoerveld een decimaal getal verwacht, dit getal is mogelijk negatief. Het getal wordt geconverteerd naar een integer.

+ x - er wordt op het invoerveld een hexadecimaal getal verwacht, het wordt geconverteerd naar een integer.

+ o - er wordt op het invoerveld een octaal getal verwacht, het wordt geconverteerd naar een integer.

+ f - er wordt op het invoerveld een decimal getal verwacht, met mogelijk een decimale punt en mogelijk een exponent, dit wordt geconverteerd naar een float.

+ s - er wordt geen conversie uitgevoerd. Het invoerveld wordt gekopieerd naar het string-argument uit de parameter-lijst, een NULL character wordt aan het einde toegevoegd.

+ c - het invoerveld bevat een teken, dit wordt gekopieerd naar het character-argument.

* Voor elke conversie-operator (behalve voor diegene waarbij de toekenning wordt onderdrukt met het "*" teken) moet er een corresponderend argument zijn, die een pointer naar een variable van het juiste type is. Bijvoorbeeld, een d-conversie vereist dat er een pointer-argument naar een integer is.

* Om het procent teken uit de invoer overeen te laten komen met de controlstring, moet in de controlstring "%%" gebruikt worden.

De functie retourneert EOF als het einde van de file is bereikt, anders retourneert de functie het aantal correct uitgevoerde conversies.

Om strings binnen te lezen, waar spaties in zijn opgenomen, kan niet van scanf gebruik gemaakt worden, dan moet de

functie gets gebruikt worden.

Bijvoorbeeld:

```
# include <stdio.h>
int i;
float f;
char s[80], c;

scanf ("%d %f", &i, &f);
/*
 * invoer: 123 456
 * resultaat: i = 123, f = 456.0
 */

scanf ("%3d %5f", &i, &f);
/*
 * invoer: 123456.7890
 * resultaat: i = 123, f = 456.78
 */

scanf ("%10s staat %c", s, &c);
/*
 * invoer: In Holland staat een huis ...
 * uitvoer: s = "In Holland", c = 'e'
 */
```

SPRINTF

```
# include <stdio.h>
sprintf (s, control, arg1, arg2, ...)
   char *s, *control;
```

De functie sprintf heeft dezelfde functionaliteit als printf, met dat verschil dat sprintf de uitvoer in de string s plaatst en niet in de standaard uitvoer.

SSCANF

```
# include <stdio.h>
sscanf (s, control, arg1, arg2, ...)
   char *s, *control;
```

De functie sscanf heeft dezelfde functionaliteit als scanf, met dat verschil dat sscanf de invoer uit de string s haalt en niet uit de standaard invoer.

STDIN

```
# include <stdio.h>
```

stdin is de stream, die direct al tijdens de executie van het programma geopend is en waar al de standaard invoer uit wordt gelezen. Dit is praktisch altijd het toetsenbord.

STDOUT

```
# include <stdio.h>
```

stdout is de stream, die direct al tijdens de executie van het programma geopend is en waar al de standaard uitvoer in wordt geschreven. Dit is praktisch altijd het beeldscherm.

8/4.11.6
Overige standaard functies

De overige standaard functies zijn niet in te delen in één van de eerder besproken groepen. Deze functies komen wel in elke standaard bibliotheek voor en vallen buiten de typische Commodore 64 functies, die in de volgende paragraaf besproken zullen worden.

ABORT

```
abort()
```

De functie abort stopt het programma. Alle files die geopend zijn worden gesloten.

EXIT

```
exit()
```

De functie exit stopt het programma. Alle files die geopend zijn worden gesloten.

LONGJMP

```
# include<setjmp.h>
```

```
longjmp(env, val)
   jmp_buf env;
   int val;
```

De functie longjmp herstelt de stack, die door de functie setjmp is gesaved en retourneert op zo'n manier dat het lijkt alsof de functie setjmp is geretourneerd met als geretourneerde waarde val. De functie-call naar setjmp en longjmp mogen in verschillende functies voorkomen, maar de setjmp-call moet wel aan de call van longjmp vooraf gaan. De faciliteit die door longjmp en setjmp wordt geleverd staat bekend als een non-local exit. Dit kan gezien worden als een goto opdracht, die over functiegrenzen heen gaat.

Voorbeeld

```
# include <setjmp.h>

int errno, error;
jmp buf_env;

errno = setjmp(env);
if (errno !=0)
/*   printf ("error %d", errno);
 *   exit();
 */
/* ...*/

if (error)
   longjmp(env, 1)
```

QSORT

```
qsort(base, nel, elsize, comp)
   char *base;
   int nel, elsize;
   int (*comp) ();
```

qsort sorteert een array, die begint op base, nel elementen heeft, die elk van grootte elsize zijn. comp wijst naar een functie, die de array elementen vergelijkt. Die functie moet twee pointer-argumenten vergelijken en een integer kleiner dan, gelijk aan, of groter dan NULL retourneren, als het eerste element respectievelijk kleiner, gelijk aan dan wel groter is dan het tweede element.

Voorbeeld

```
/* sorteer een array met floats */

# define NELEM 100

float t[NELEM];
int fcomp();

qsort(t, NELEM, sizeof(float), fcomp);

/* .. */

fcomp(p1, p2)
float *p1, *p2;
{
   if (*p1 < *p2)
     return(-1);
   else if (*p1 = *p2)
     return(0);
   else
     return(1);
}
```

SETJMP

```
# include <setjmp.h>
int setjmp(env)
   jmp buf env;
```

setjmp slaat de stack-omgeving op in env en retourneert NULL. Deze functie wordt gebruikt in combinatie met de functie longjmp. Zie longjmp voor meer informatie.

8/4.11.7
Commdore functies

Proline heeft een aantal functies in de standaard bibliohteek opgenomen, die

4.11 De standaard bibliotheek

specifiek voor de Commodore 64 zijn. Zodra van deze functies gebruik gemaakt gaat worden, zijn de programma's niet meer overdraagbaar naar een andere computer, zonder de programmatuur te herschrijven. Om deze reden is het aan te bevelen om zo min mogelijk gebruik van deze functies te maken. Aan de andere kant maken deze functies een groot aantal zaken mogelijk, die anders in C op de Commodore 64 niet haalbaar waren geweest.

BCMP
```
int bcmp(p1, p2, len)
   char *p1, *p2;
 int len;
```

bcmp vergelijkt len bytes van de strings p1 en p2 en retourneert NULL als ze gelijk zijn, anders een waarde ongelijk aan NULL.

BCOPY
```
bcopy(p1, p2, len)
   char *p1, p2;
   int len;
```

bcopy kopieert len bytes van string p1 naar string p2.

BZERO
```
bzero(p, len)
   char *p;
   int len;
```

bzero vult string p op, met len nullen.

CLOSE
```
close(fileno)
 int fileno;
```

De functie close sluit een file, met file nummer fileno, dit is hetzelfde file nummer als gebruikt is om met de open functie de corresponderende file te openen. Zie ook open.

CLOSEDIR
```
# include <dir.h>
closedir()
```

De functie closedir sluit de geopende directory.

DEVICE
```
device(n)
   int n;
```

De functie device verandert de default disk drive. De files die na de device call worden geopend, staan of komen te staan op device n. Het default device is device 8.

Bijvoorbeeld
```
/* open een file op device (disk drive) 9 */


device(9);
f = fopen ("1:bestand", "r");
```

FFS
```
int ffs(i)
   int i;
```

ffs retourneert de positie van het eerste bit wat geset (1) is in i. De bits zijn genummerd te beginnen bij 1. Als het argument 0 is retourneert ffs -1.

HIGHMEM
```
highmem(adres)
   unsigned adres;
```

highmen verandert het hoogste adres, wat een C programma kan gebruiken. De stack en de heap zullen niet boven dit

adres komen. De waarde van het argument moet één byte groter zijn dan het gewenste adres. Als highmem niet wordt aangeroepen, is de default adres waarde 0xd000, wat betekent dat het hoogste adres wat gebruikt kan worden, 0cfff is.

KERNAL

```
kernal(flag)
   int flag;
```

Als de flag nul is, wordt al het RAM geheugen wat zich in de Commodore 64 bevindt, bereikbaar en tevens worden alle interrupt requests gedisabled, op interrupts wordt dan dus niet meer gereageerd (ook niet van het toetsenbord). Bij een andere waarde van flag, wordt de kernal ROM en de I/O ruimte beschikbaar (van 0xd00 tot en 0xfff) en worden de interrupts geenabled. In de default instelling is de kernal en de I/O ruimte bereikbaar. Het BASIC ROM staat altijd uit, behalve wanneer er floating point operaties plaats vinden.

OPEN

```
int open(fileno, device, secaddr, name)
   int fileno, device, secaddr;
   char *name;
```

De functie open, opent een file op de BASIC manier, met een file nummer, device nummer, een secondair adres en een file naam, wat ook de argumenten zijn van het Basic OPEN commando. Raadpleeg deel 6 of 7 voor de betekenis van de argumenten. De functie retourneert NULL als de file niet geopend kan worden, anders wordt een waarde ongelijk aan NULL geretourneerd.

Voorbeeld

```
/* beeld een file af op het scherm */
# include <stdio.h>

int c;

open(5, 8, 5, "file-naam,s,r");

for ((c = getc(5)) != EOF)
   putchar(c);

close(5);
```

OPENDIR

```
# include <dir.h>
opendir()
```

De functie opendir opent een directory om die te lezen. Als de directory niet geopend kan worden wordt NULL geretourneerd.

READDIR

```
# include <stdio.h>
struct direct *readdir()
```

readdir leest het volgende gedeelte uit de directory en retourneert een pointer naar het gelezen gedeelte. In de header-file dir.h wordt de structuur van zo'n directory-gedeelte beschreven. Deze structuur ziet er als volgt uit:

4.11 De standaard bibliotheek

```
struct direct{
    char type;              /* file type */
    char track;             /* track of link */
    char sector;            /* sector of link */
    char name[16];          /* file name */
    char sstrk;             /* relative file side sector track */
    char sssec;             /* relative file side sector sector */
    char recsize;           /* relative file record size */
    char unused[6]
unsigned nblocks;           /* file size */
                            };
```

Hierin is ondermeer de positie van de eerste track, van de file, op de disk te vinden, het type van de file, de file naam en de grootte van de file.

Bijvoorbeeld
/* Laat de inhoud van de directory zien. */

```
# include <dir.h>
# include <stdio.h>

struct direct *dp;
opendir();
for (dp = readdir(); dp != NULL; dp =
readdir())
    puts (dp->name);
closedir();
```

REWINDDIR

```
# include <dir.h>
rewinddir();
```

rewinddir zorgt er voor dat als de volgende keer readdir wordt aangeroepen, het eerste gedeelte van de directory wordt gelezen.

SYS

```
sys(adres, a, x, y)
    unsigned adres;
    char *a, *x, *y;
```

De functie sys roept een machinetaal routine aan. sys laadt de accumulator, de x en y registers van de microprocessor met de waarden, waar a, x en y naar wijzen. Dan roept het de subroutine aan, die zich bevindt op de aangegeven waarde van de parameter adres. Wanneer de machinetaal routine beëindigd wordt, worden de mogelijke nieuwe waarden van de registers op de adressen geplaatst, die door de parameters zijn aangegeven. De functie retourneert NULL als de carry flag van de processor 0 is, anders 1.

Voorbeeld

```
char *s, x, y;

s = "dit is een voorbeeld-string";

while (*s++)
    sys (0xffd2, s, &x, &y);

/*
 * 0xffd2 is de kernal routine,
 * die tekens op het scherm afdrukt
 */
```

8/4.12

Eigen bibliotheken

Het aanleggen van een eigen bibliotheek met zelf geschreven routines behoort tot de mogelijkheden van C. C ontleent juist uit deze mogelijkheid zijn grote kracht. Er kunnen grote bibliotheken van functies worden aangelegd, waarvan de opbouw en de inhoud al lang niet meer bekend is, maar waarvan de functie nog wel bekend is. De functie van de routines kan eenvoudig gedefinieerd worden in een user's guide van die bibliotheek, zoals dit bijvoorbeeld door Proline is gedaan.

Printer functies voor de MPS 1000

Met de MPS 1000 en met andere printers is het niet mogelijk om met de Commodore karakter set, de speciale C karakters af te drukken. Deze speciale C karakters zijn: {,}, , ˜en. Om deze karakters op juiste wijze af te drukken zijn in deze sectie een aantal routines opgenomen.

Tevens staan er in deze paragraaf nog een aantal andere functies voor het OPENen, CLOSEn, reseten, enzovoorts van de printer. De functionele beschrijving van de printer functies zal gegeven worden op de standaard functie beschrijvings methode, zoals dit bijvoorbeeld ook door Proline is gebruikt. Tevens zal naast de functionele beschrijving de listing van de functie worden gegeven.

NAAM
 init_pr – open printer device

SYNOPSIS
 include<stdio.h>

 FILE init_pr()

BESCHRIJVING
Init_pr opent het printer device en retourneert de file identifier waarde, deze is altijd 6. Tevens wordt het programmeerbare character channel geopend, deze identifier is altijd 5.

```
FILE init_pr() /* open printer channels */
{
   open(5, 4, 5, "");       /* open programmable character channel */
   open(6, 4, 7, "");       /* open printer channel */
   return(6);
}
```

8/4.12
Eigen bibliotheken

NAAM
close_pr — close printer device

SYNOPSIS
include <stdio.h>

close_pr(dev)
FILE dev;

BESCHRIJVING
Close_pr sluit het printer device dat met open_pr is geopend.

```
close_pr(dev) /* close printer channels */
FILE dev;
{
    if (dev != 6) {
        printf("can't close open printer device: %d\n", dev);
        exit(1);
    }
    close(5);
    close(6);
}
```

NAAM
reset_pr — reset printer device

SYNOPSIS
reset_pr()

BESCHRIJVING
reset_pr reset het printer device dat met open_pr is geopend.

```
reset_pr() /* reset printer */
{
    open(7, 4, 10, "");
    fprintf(7, " ");
    close(7);
}
```

NAAM
nlq_on — zet printer device in near letter quality mode

SYNOPSIS

nlq_on()

BESCHRIJVING
Nlq_on zet het printer device in near letter quality mode, dat met open_pr is geopend.

```
nlq_on()
{
    fprintf(6, "%c", 31);
}
```

NAAM
nlq_off — zet de near letter quality mode van het printer device uit.

SYNOPSIS

nlq_off()

BESCHRIJVING
Nlq_off zet de near letter quality mode van het printer device uit, dat met open_pr is geopend.

```
nlq_off()
{
    fprintf(6, "%c", 159);
}
```

8/4.12
Eigen bibliotheken

NAAM

print_lb, print_rb, print_bs, print_ti, print_vb – print een string met een speciaal C teken.

SYNOPSIS

print_lb(s)
char *s;

print_rb(s)
char *s;

print_bs(s)
char *s;

print_ti(s)
char *s;

print_vb(s)
char *S;

BESCHRIJVING

Print_lb print de string s gevolgd door een left brace: {

Print_rb print de string s gevolgd door een right brace:}

Print_bs print de string s gevolgd door een back slash:

Print_ti print de string s gevolgd door een tilde: ~

Print_vb print de string s gevolgd door een vertical bar:|

```
print_lb(s) /* print left brace: { */
char *s;
{
   fprintf(5, "%c%c%c%c%c%c%c%c", 0, 16, 108, 130, 130, 0, 0, 0);
   fprintf(6, "%s%c", s, 254);
}

print_rb(s) /* print right brace: } */
char *s;
{
   fprintf(5, "%c%c%c%c%c%c%c%c", 0, 130, 130, 108, 16, 0, 0, 0);
   fprintf(6, "%s%c", s, 254);
}

print_bs(s) /* print back slace: \ */
char *s;
{
   fprintf(5, "%c%c%c%c%c%c%c%c", 128, 64, 32, 16, 8, 4, 2, 0);
   fprintf(6, "%s%c", s, 254);
}

print_ti(s) /* print tilde: ~ */
char *s;
{
   fprintf(5, "%c%c%c%c%c%c%c%c", 64, 128, 128, 64, 32, 32, 64, 0);
   fprintf(6, "%s%c", s, 254);
}
```

4.12 Eigen bibliotheken

```
print_vb(s) /* print vertical bar: ¦ */
char *s;
{
    fprintf(5, "%c%c%c%c%c%c%c%c", 0, 0, 0, 238, 0, 0, 0, 0);
    fprintf(6, "%s%c", s, 254);
}
```

Bibliotheek toegang-files

Tijdens het linken van een programma zijn er twee verschillende manieren om een object module mee te linken. Bij de eerste manier wordt de naam van de module ingetypt, deze naam heeft meestal als extensie .obj of .o. Bij de tweede manier wordt de naam van een bibliotheek toegang-file ingetypt, de object module moet dan wel in deze bibliotheek zijn opgenomen.

Als de tweede manier gebruikt wordt dan zal de linker in de bibliotheek toegang-file zoeken naar die modules, waar functies in staan die tot nog toe niet in het geheugen zijn geladen. De linker zal zodra hij zo'n module vindt, deze module in het geheugen laden. De linker blijft zoeken totdat hij niets meer kan vinden in de desbetreffende bibliotheek toegang file. Neem nu als voorbeeld een eenvoudig programmaatje als:

```
#include <stdio.h>
main()
{
printf("Hallo, wereld\n");
}
```

Als dit is gecompileerd dan kan het linken op de twee verschillende manieren worden uitgevoerd. Bij de eerste manier ziet een link sessie er als volgt uit:

```
$ link
 hallo.o

unresolved external
references:
c$start    c$105    c$1102
printf     c$106
 printf.obj
*
*
*
```

De enige functie uit de standaard bibliotheek die mee-gelinkt moet worden heet printf, deze functie zit in een module die printf.obj heet (de functienaam en de naam van de module hoeven niet altijd overeen te komen). De overige functies zitten in de systeembibliotheek (syslib). Bij de tweede manier om dit programma te linken moeten de beide bibliotheken opgegeven worden tijdens het linken:

```
$ link
 hallo.o
 stdlib.l
 syslib.l
*
*
```

Het linken is op de tweede manier veel gebruikersvriendelijker. Maar wat nu te doen als er van veel zelfgemaakte functies gebruikt wordt gemaakt. Als het niet mogelijk is om zelf bibliotheek toegang-files te creëren, dan zullen bij het linken altijd de namen van de object modules moeten worden ingetypt. Met het utility-programma Libr uit hoofdstuk 8/4.13 kunnen deze bibliotheek toegang-files gecreëerd worden.

De definitie van de bibliotheek toegang-file, zoals PROLINE die gebruikt, wordt hieronder weergegeven:

size	identifier	\0	filename	\0	identifier	\0	filename

De grootte van elk onderdeel wordt daaronder in bytes vermeld. De betekenis van de onderdelen is:

- size : Het aantal identifier/filename paren.
- identifier: De naam van de functie.
- filename: De naam van de file waarin de corresponderende functie staat.

4.12 Eigen bibliotheken

De '\0'-waarden zijn byte-seperatoren, met waarde 0, tussen de filename en identifier strings in.

8/4.13

Bibliotheekbeheerder

De functies uit de standaard bibliotheek worden tijdens het linken samen met één of meerdere modules waar een programma uit bestaat tot een werkend programma samengevoegd. De linker vindt de functies, die over een groot aantal bestanden staan verspreid, door middel van een bibliotheek toegang file (zie ook 8/4.12, blz. 5 en verder).

Met het utility programma Libr kunnen deze bibliotheek toegang-files worden gecreëerd. Libr's belangrijkste functies zijn het creëren van die bestanden, het toevoegen en verwijderen van modules aan de desbetreffende toegang-file. Libr maakt een backup van de oorspronkelijke toegang-file, voordat de nieuwe toegang-file wordt weggeschreven. De backup moet voordat de nieuwe toegang file wordt aangepast wel worden verwijderd.

De synopsis van het programma is:

```
$ libr [filenaam]
```

waarin filenaam de naam van de bibliotheek toegang-file is. Als deze naam niet wordt meegegeven bij de aanroep van het programma kan deze met een intern commando nog bepaald worden.

Het hoofdmenu wordt in de main-routine afgebeeld, er zijn acht commando's te geven. Deze worden met een éénletterig commando ingegeven. Deze letter komt overeen met de eerste letter van elke regel uit het keuze menu. Commando's kunnen met kleine of met hoofdletters gegeven worden. De listing van het hoofdprogramma is hieronder weergegeven.

```
#include <stdio.h>
#include <dir.h>

#define   MAX_LIB_SIZE   255
#define   MAX_ID          20
#define   MAX_LINE        81
#define   NAME_SIZE       20

char ids[MAX_LIB_SIZE][MAX_ID],      /* array to store identifiers */
     files[MAX_LIB_SIZE][MAX_ID];    /* array to store filenames   */
char size;                           /* number of identifiers      */

main(argc, argv)
int argc;
char *argv[];
```

4.13 Bibliotheekbeheerder

```
{
    char lib_name[NAME_SIZE];
    char inbuf[MAX_LINE];
    size = 0;

    strcpy(lib_name, "");
    if (argc > 2) {
        printf("Usage: libr [library-name]\n");
        exit(1);
    }
    if (argc == 2) {
        strcpy(lib_name, argv[1]);
        get_lib(lib_name);
    }
    for (;;) {
        clrscrn();
        printf("\n\n");
        printf("        Library: %s\n\n\n", lib_name);
        printf("        Get library\n");
        printf("        Display library\n");
        printf("        List directory\n");
        printf("        Clear library\n");
        printf("        Add modules to library\n");
        printf("        Remove modules from library\n");
        printf("        Put library\n");
        printf("        Quit\n\n");
        printf("        Enter Choice: ");
        gets(inbuf);
        clrscrn();
        switch (*inbuf) {
        case 'g':
        case 'G':
            printf("\n\nlibrary file name: ");
            gets(lib_name);
            get_lib(lib_name);
            break;
        case 'd':
        case 'D':
            display();
            break;
        case 'l':
        case 'L':
            list_dir();
            break;
        case 'c':
        case 'C':
            size = 0;
            break;
        case 'a':
        case 'A':
```

4.13 Bibliotheekbeheerder

```
                add();
                break;
            case 'r':
            case 'R':
                remove();
                break;
            case 'p':
            case 'P':
                put_lib(lib_name);
                break;
            case 'q':
            case 'Q':
                exit();
            }
        }
}

clrscrn()
{
    putchar(147);
}
```

Get_lib opent een bibliotheek toegang-file, leest deze en sluit de file. De file is opgebouwd zoals beschreven is in 8/4.12.

Openlib opent een file. Fget_string leest een string uit een file en plaatst het resultaat in het eerste argument.

```
get_lib(lib_name)
char *lib_name;
{
    FILE f;
    unsigned i;

    if ((f = openlib(lib_name, "r")) == NULL)
        return;
    size = getc(f);
    for (i = 0; i < size; i++) {
        fget_string(ids[i], f);
        fget_string(files[i], f);
    }
    fclose(f);
}
FILE openlib(lib_name, mode)
char *mode;
char *lib_name;
{
    FILE f;
```

4.13 Bibliotheekbeheerder

```
    f = fopen (lib_name, mode);
    if (f == NULL || ferror()) {
        f = NULL;
        printf("\ncan't open %s\n", lib_name);
        wait();
    }
    return f;
}

fget_string(s, f)
char *s;
FILE f;
{
    while (*s++ = getc(f))
      ;
}
```

Display toont de inhoud van de bibliotheek toegang-file. Display toont maximaal 24 regels tegelijkertijd. Om te vervolgen moet op de Return-toets worden gedrukt.
List_dir laat de inhoud van de disk-directory op het scherm zien. Wat opvalt aan de C-code, is dat de functie begint met een closedir(). Zonder deze closedir werkt de routine niet naar behoren, blijkbaar laat het systeem de disk directory geopend, zodat de directory alleen na te worden gesloten weer geopend kan worden.

```
display()
{
   unsigned i, line;

   line = 1;
   for (i = 0; i < size; i++) {
       if (line == 1) {
           clrscrn();
           printf("%-20s%-19s\n\n", "IDENTIFIER", "FILE");
           line = 3;
       }
       printf("%-20s%-19s\n", ids[i], files[i]);
       line++;
       if (line == 24) {
           wait();
           line = 1;
       }
   }
   wait();
}
```

4.13 Bibliotheekbeheerder

```
list_dir()
{
    int i, line, col;
    struct direct *dp;

    col = 1;
    line = 1;
    closedir();         /* make sure directory is closed */
    opendir();
    for (dp = readdir(); dp != NULL; dp = readdir()) {
        if ((line == 1) && (col == 1)) {
            clrscrn();
        }
        printf("%15s", dp->name);
        if (col == 2) {
            line++;
            col = 1;
            printf("\n");
        } else {
            col++;
        }
        if (line == 24) {
            wait();
            line = 1;
        }
    }
    wait();
    closedir();
}
```

Add voegt een module toe aan de toegang-file. De inhoud van de module wordt gelezen. De functienamen worden gezocht in de module en deze namen worden aan de array ids, waar alle functie namen in staan, toegevoegd.

```
add()
{
    unsigned n;
    FILE f;
    char inbuf[MAX_LINE];

    printf("\nhit RETURN when done\n\n");
    for (;;) {
        printf("object file name: ");
        gets (inbuf);
        if (!isalpha(*inbuf)) {
            return;
        }
        f = fopen (inbuf, "r");
        if (f == NULL || ferror()) {
```

4.13 Bibliotheekbeheerder

```
            printf("can't open %s\n", inbuf);
        } else {
            n = getw(f);
            while (n--) {
                getc(f);
            }
            n = getw(f);
            while (n--) {
                getw(f);
            }
            n = getw(f);
            while (n-- && size < MAX_LIB_SIZE) {
                fget_string(ids[size], f);
                if (isalpha(*ids[size]) && isprint(ids[size][1])) {
                    strcpy (files[size++], inbuf);
                }
                getc(f);
                getw(f);
            }
            fclose (f);
            if (size == MAX_LIB_SIZE) {
                printf("\nlibrary is full\n");
                wait();
                return;
            }
        }
    }
}
```

Remove verwijdert een module uit de toegang-file. Alle functies die eventueel in de desbetreffende module staan worden verwijderd.

```
remove()
{
    unsigned i, j;
    char inbuf[MAX_LINE];

    printf("\nhit RETURN when done\n");
    for (;;) {
        printf("\nfile to delete: ");
        gets (inbuf);
        if (!isalpha(*inbuf)) {
            return;
        }
        for (i = 0; i < size && strcmp(files[i], inbuf) != 0; i++)
            ;
        if (i == size) {
            printf("file not found\n");
        }
        else {
```

```
          while (i < size && strcmp (files[i], inbuf) == 0) {
             for (j = i; j+1 < size; j++) {
                 strcpy (files[j], files[j+1]);
                 strcpy (ids[j], ids[j+1]);
             }
             size--;
          }
       }
    }
}
```

Put_lib schrijft de bibliotheek toegang-file naar disk. Eerst wordt een backupnaam voor de oorspronkelijke file gemaakt, daarna krijgt die file een andere naam, waarna de nieuwe file naar disk wordt geschreven.

Create_name maakt een nieuwe naam: de extensie '.l' wordt vervangen door '.bak'.

Dos_rename, wait en getch spreken voor zich.

```
put_lib(lib_name)
char *lib_name;
{
   FILE f;
   unsigned i;
   char backup_name[NAME_SIZE];

   if (strcmp(lib_name, "") == 0) {
       printf("\n\nlibrary file name: ");
       gets(lib_name);
   }
   if (!create_name(backup_name, lib_name)) {
      return;
   }
   dos_rename(lib_name, backup_name);
   if ((f = openlib(lib_name, "w")) == NULL)
      return;
   putc (size, f);
   for (i = 0; i < size; i++)
      fprintf(f, "%s%c%s%c", ids[i], 0, files[i], 0);
   fclose(f);
}
```

4.13 Bibliotheekbeheerder

```
create_name(backup, name)
char *backup, *name;
{
    char *p;

    strcpy(backup, name);
    p = backup;
    while (*p) {
        p++;
    }
    while (*p != '.') {
        if (p == backup) {
            printf("Error in library name\n");
            wait();
            return(0);
        }
        p--;
    }
    strcpy(p, ".bak");
    return(1);
}

dos_rename(old_name, new_name)
char *old_name, *new_name;
{
    open(7, 8, 15); /* open command channel */
    fprintf(7, "r:%s=0:%s", new_name, old_name);
    close(7);
}

wait()
{
    printf("\nhit RETURN to continue");
    while (getch() != '\n') {
        ;
    }
}

getch()
{
    char a, x, y;

    do {
        sys (0xffe4, &a, &x, &y);
    } while (a == 0);
    return a;
}
```

8/5

FORTH

Inhoud

8/5.1 Onze eerste schreden in FORTH
8/5.2 Ons eerste programma
8/5.3 FORTH basiswoordenschat
8/5.4 Programmastrukturen

8.5 FORTH

FORTH is meer dan zomaar een programmeertaal, de uitdrukking 'programmeermilieu' is heel wat beter van toepassing omdat FORTH eigenlijk een synthese is van programmeertaal, operating systeem en testhulpmiddelen.

FORTH werd in het begin van de jaren '70 ontwikkeld door de Amerikaan Charles Moore die een taal voor ogen had die snel en eenvoudig te hanteren zou zijn. Moore gebruikte FORTH op een PDP-11 voor het besturen van astronomische telescopen. Het operating systeem van deze computer kon echter maar bestandsnamen van maximaal 5 tekens verwerken, op grond waarvan de oorspronkelijk bedachte naam 'FOURTH' (taal van de vierde generatie) werd afgekort tot FORTH. Ook vandaag nog wordt FORTH overwegend toegepast in besturingen, zelfs in de ruimtevaart. Maar met FORTH kunnen ook andere taken tot een goed einde worden gebracht.

Om de belangstelling voor deze taal nog verder aan te wakkeren beginnen we nu met een aantal vragen.

* Kunt u zich een programmeertaal voorstellen waarbij u vanaf een bepaald niveau uw aanwijzingen aan de computer nagenoeg in de omgangstaal kunt formuleren? Een voorbeeld daarvan in de donkere kamer zou de besturing van de belichtingstijd kunnen zijn. Het zou toch fantastisch zijn als de computer een zin als 'Lampen aan, 10 seconden wachten, lampen uit' zou verstaan.
* Kunt u zich een programmeertaal voorstellen waarin u niet alleen direkt op machineniveau maar ook in algemeen beschaafd Nederlands kunt programmeren en die dat in alle gevallen structureert, ook op machineniveau?
* Kunt u zich een programmeertaal voorstellen waarvan u de compiler, dus de omvang van de taal, zelf kunt uitbreiden?
* Kunt u zich een programmeertaal voorstellen die de ontwikkelingstijd van uw programma's drastisch bekort?

Luidt uw antwoord op een van deze vragen 'neen', dan moet u beslist doorlezen want FORTH is die taal.

8/5.1

De eerste schreden in Forth

We gaan nu onze eerste schreden in FORTH zetten. Het beste kunt u nu uw computer inschakelen en het FORTH-systeem laden omdat u de voorbeelden dan onmiddellijk kunt controleren. In onze voorbeelden wordt gebruik gemaakt van het C 64-FORTH van Performance Micro Productions. Omdat FORTH echter een zeer goed gestandaardiseerde taal is kunnen de voorbeelden ook op elk ander FORTH-systeem worden gecontroleerd. Maar denk eraan: FORTH is anders en vertoont maar weinig verwantschap met andere programmeertalen. Maar vooral: heeft men de FORTH-koorts eenmaal te pakken dan vergeet men talen als BASIC maar al te graag. FORTH werkt verslavend. Maar zonder gekheid, inmiddels zal de FORTH compiler wel geladen en gestart zijn en heeft hij zich met een eenvoudig 'OK' gemeld. Dat doet ergens aan BASIC denken, ook hier bedoelt de computer gewoon 'READY'. FORTH is een minder afgeronde taal, in FORTH is eigenlijk alles 'OK'. Om zo terloops de omvang van de taal te demonstreren voeren we maar eens het woord VLIST in en drukken de RETURN-toets in. Hopelijk valt u nu niet van schrik van uw stoel want wat er nu allemaal op het scherm van de computer verschijnt is de basiswoordenschat van FORTH. Dat kunnen zo rond de 250-400 woorden zijn. Maar geen paniek! Tenslotte herkennen we naast enkele bekende woorden als '+' en '–' ook al andere rekenkundige symbolen. FORTH kan dus kennelijk rekenen . . .

We zullen ons dan ook maar gelijk met enkele rekenvoorbeelden bezighouden. Daarmee beleven we dan ook onmiddellijk de eerste eigenaardigheid van FORTH. Deze taal rekent namelijk met OPN. Wat dat is? OPN is de afkorting van 'Omgekeerde Poolse Notatie'. Geen angst, u hoeft geen Pools te leren om OPN te kunnen begrijpen. De naam is ontleend aan het feit dat een Pools wiskundige als eerste dit idee opperde. Om berekeningen aan een computer te formuleren kan men verschillende schrijfwijzen hanteren. Bijvoorbeeld 2 + 3; hier wordt dus eerst de eerste operand (2) aangegeven, dan de bewerking (+) en tenslotte de tweede operand (3). Deze vorm is in de meeste programmeertalen gebruikelijk.
OPN wil echter niets anders zeggen dan dat men eerst alle operanden aangeeft gevolgd door de bewerking die erop moet worden uitgevoerd. zo werken bijvoorbeeld de calculators van Hewlett Packard volgens dit principe.
In het genoemde voorbeeld zou men dus 2 3 + moeten schrijven. Voert men dit in FORTH in dan volgt het vertrouwde OK, tenzij u de spaties tussen de operanden en de operator hebt weggelaten. In FORTH is de spatie namelijk erg belangrijk; hij scheidt de afzonderlijke woorden in een reeks van instrukties.

5.1 De eerste schreden in Forth

OK, prima, maar waar blijft nu de uitkomst? We zullen u niet in spanning laten, u toetst eenvoudig de decimale punt (.) in en FORTH levert het juiste antwoord, namelijk 5.
Wat is er nu eigenlijk gebeurd? Bij het invoeren van het getal 2 zet FORTH dat getal op een datastack. De datastack is een geheugengebied dat letterlijk als stapelplaats fungeert. Men kan er informatie inzetten en er ook weer vanaf nemen, maar altijd volgens een bepaald systeem: het LIFO-systeem (Last In First Out). Dit betekent alleen maar dat de informatie die er als laatste werd ingezet, er als eerste weer wordt afgenomen. Dit stapelen is het beste te vergelijken met een stapeltje kaarten waar men alleen kaarten bovenop kan leggen en van bovenaf kaarten kan wegnemen. In ons voorbeeld hebben we er dus een kaart met het getal 2 bovenop gelegd. Bij de volgende handeling gebeurt hetzelfde, maar met het getal 3. Er liggen dan twee getallen op de stack, de 3 bovenop. De + zorgt er nu voor dat de beide bovenste waarden van de stack worden gehaald, dat ze bij elkaar worden opgeteld en dat de uitkomst weer op de stack wordt gezet. De beide operanden gaan daarbij overigens verloren. Met . wordt het bovenste element vanaf de stack op het scherm gezet en daarbij eveneens van de stack verwijderd. We zullen nog enkele rekenvoorbeelden nalopen:

```
25 4 - .   geeft als uitkomst 21
5 6 * .    geeft als uitkomst 30
10 3 / .   geeft als uitkomst de waarde 3
```

Vooruit maar, uit de laatste berekening had eigenlijk 3.33333333 moeten komen! Maar dat is dan de tweede karakteristieke eigenschap van FORTH: het kan voorlopig alleen nog met integers (hele getallen) tussen -32768 en 32767 werken. Maar dat is geen bezwaar zoals we later nog zullen zien.

De OPN-schrijfwijze van FORTH bepaalt verder dat de haakjes uit rekenkundige bewerkingen mogen worden weggelaten. Ook hiervan een voorbeeld:

```
2 3 + 3 5 + *
```

Heel eenvoudig, of niet? Eerst worden de twee uitdrukkingen tussen haakjes berekend, daarna volgt de vermenigvuldiging. In gewone taal: eerst worden de getallen 2 en 3 op de stack gezet en bij elkaar opgeteld (+). Vervolgens worden de getallen 4 en 5 op de stack gezet en eveneens opgeteld (+). In de stack staan nu de uitkomsten van beide optellingen die tenslotte met * worden vermenigvuldigd. De uitkomst daarvan blijft bovenop de stack staan.

Wie dit omslachtig vindt moet wel weten dat:

* De alom bekende firma Hewlett Packard calculators vervaardigt die uitsluitend op dit systeem zijn gebaseerd om het bijzonder efficiënt werkt.
* Compilers voor hogere programmeertalen de programma's meestal in een tussencode vertalen die FORTH heel dicht benadert en bij rekenkundige problemen volgens hetzelfde systeem werkt. Een schoolvoorbeeld daavan is het bekende UCSD-Pascal (UCSD= University of California at San Diego). De door UCSD-Pascal geleverde, zogenaamde P-code is eigenlijk hetzelfde als FORTH.

8/5.2

Een eerste programma

We zullen er nu eens goed tegenaan gaan en een klein programma in FORTH schrijven. Ook hierbij vertoont FORTH weinig overeenkomst met conventionele programmeertalen. Een programma in FORTH schrijven wil niets anders zeggen dan uitbreiden van de woordenschat van het FORTH-systeem (die met VLIST kan worden uitgelijst). Ter illustratie kiezen we het omrekenen van decimale getallen in hexadecimale getallen, een opgave die met FORTH heel elegant kan worden opgelost. Een mogelijke oplossing zou er als volgt uit kunnen zien:

: D->H DUP DECIMAL . "DECIMAAL" . ." = HEXADECIMAAL " HEX . DECIMAL ;

Wat hebben we nu hier en daar gedaan? De dubbele punt leidt de definitie van een nieuw woord in. In FORTH-kringen noemt men zoiets een 'Colon-Definition'. Nu volgt de naam van het nieuw te definiëren woord, in ons geval is dat "D->H". Daarna volgt de reeks FORTH-woorden die bij het oproepen van het nieuwe woord moeten worden uitgevoerd. In dit geval wordt het bovenste stack-element gedupliceerd (DUP, wat wil zeggen dat het bovenste stack-element twee maal beschikbaar is). Vervolgens wordt het in het decimale stelsel afgegeven. Nu wordt FORTH op het hexadecimale stelsel ingesteld (HEX) en het (door het dupliceren twee maal beschikbare) bovenste element in dit talstelsel afgegeven. Tenslotte wordt FORTH weer op het decimale stelsel ingesteld. De puntkomma sluit de colon-definition af.
Het nieuwe woord is nu onmiddellijk uitvoerbaar, bovendien staat het bij het uitvoeren van de VLIST-instruktie op de eerste plaats van de woordenschat.

In FORTH hoort men nieuwe woorden te documenteren door de invloed ervan op de stack te noteren. Dat wil zeggen men noteert de definitie van het woord en de daarbij optredende verandering van de stack. Daarvoor is de volgende schrijfwijze ingeburgerd geraakt:

woord (a/b/c - d/e) commentaar

De letters "a", "b" en "c" stellen de stack-inhoud voor vóór de uitvoering van het woord, de letters "d" edn "e" de stack-inhoud na het uitveren ervan. De meest rechtse letter stelt hierbij het bovenste stack-element voor. In ons voorbeeld is dus vóór de uitvoering van het woord de "c" het bovenste stack-element en na de uitvoering de "e". Tenslotte kan nog een commentaar worden geschreven dat aangeeft wat het woord bewerkstelligt. Voor ons rekenvoorbeeld zouden we dus het volgende kunnen schrijven:

D ->H (n –) geeft n in decimale en in hexadecimale vorm af

5.1 Een eerste programma

Tenslotte moeten we nog testen of ons nieuwe woord aan zijn doel beantwoordt. Voeren we nu geslaagde definitie 194 D->H in, dan moet het volgende antwoord verschijnen:

DECIMAAL 194 = HEXIDECIMAAL C2

Het woord D->H is nu vast bestanddeel van het FORTH-vocabulaire geworden. Het kan op elk gewenst moment ook in andere woorden worden gebruikt. Willen we bijvoorbeeld een omrekentabel afdrukken waarin de decimale waarden van 0 tot 255 zijn omgerekend dan moet dit als volgt worden geschreven:

```
: TABEL CR ." DEC-HEX CONVERSIE" CR 256 0 DO I D->H CR LOOP;
```

Hier duiken weer enkele nieuwe FORTH-woorden op. CR zorgt voor een regeltransport. DO leidt een lus in, op soortgelijke wijze als de FOR . . .NEXT-lus in BASIC. De lus begint met de waarde 0 en wordt beëindigd als de waarde 256 wordt bereikt (en niet overschreden zoals in BASIC). LOOP is het NEXT-equivalent in FORTH. Met de instruktie I wordt alleen de lusindex op de stack gezet. TABEL zorgt vervolgens voor het afdrukken van de omrekentabel.

Deze voorbeelden leren ons het volgende:

* Programmeren in FORTH bestaat uit het wel doordacht uitbreiden van de basiswoordenschat van FORTH.
* FORTH-woorden hebben de status van 'subprogramma's', dat wil zeggen ze kunnen door andere woorden worden opgeroepen.
* Omdat uitsluitend reeds bestaande woorden worden opgeroepen is programmeren in FORTH een klassieke 'bottom-up' techniek wat wil zeggen dat ook de woorden voor heel eenvoudige acties eerst gedefinieerd moeten worden (als subprogramma's van het laagste niveau), dan de bijbehorende oproeper enz.
* Deze stijl van programmeren leidt tot een goede structurering van het programma, maar bovenal tot een zeer compact programma.
* omdat nieuw gedefinieerde woorden op elk gewenst moment getest kunnen worden is de kans dat fouten binnensluipen eigenlijk minimaal.

Hiermee hebben we een punt bereikt waarop het nuttig is een samenvatting te geven van de FORTH-woordenschat (verkort geschreven en met stack-informatie).

8/5.3

Forth basiswoordenschat

Operanden:		
	n, nl . . .	16-bit integers (met voorteken)
	d, dl . . .	32-bit integers (met voorteken)
	u, ul . . .	16-bit integers (unsigned: zonder voorteken)
	ud . . .	32-bit integers (unsigned: zonder voorteken)
	addr.	16-bit adres
	b	8-bit byte
	c	7-bit ASCII (karakter)
	f	Boolean of logische vlag (<> 0 komt overeen met "waar")

TOS (Top Of Stack) geeft het bovenste stack-element aan; SEC het tweede stack-element

Stack-manipulatie

-DUP	(n - n/?)	Dupliceert TOS alleen dan als TOS niet gelijk is aan nul.
>R	(n -)	Zet TOS op de return-stack.
DROP	(n -)	Neemt het bovenste stuk-element weg.
DUP	(n - n/n)	Dupliceert het bovenste stack-element.
OVER	(n1/n2 - n1/n2/n1)	Zet SEC bovenaan de stack.
R	(- n)	Kopieert de return-stack naar TOS; de return-stack blijft ongewijzigd.
R>	(- n)	Haalt TOS van de return-stack.
ROT	(n1/n2/n3 - n2/n3/n1)	Roteert de drie bovenste elementen.
SWAP	(n1/n2 - n2/n1)	Verwisselt de twee bovenste stack-elementen.

Talstelsels

BASE	(-addr)	Zet het adres van de USER-variabele BASE op TOS.
DECIMAL	(-)	Zet BASE op 10 (decimaal stelsel).
HEX	(-)	Zet BASE op 16 (hexadecimaal stelsel).

8.3 Forth basiswoordenschat

Rekenkundige bewerkingen

*	(n1/n2 - produkt)	16-Bit vermenigvuldiging (uitkomst: 16-bit).
*/	(n1/n2/n3 - quotiënt)	Als */MOD maar zonder restberekening.
*/MOD	(n1/n2/n3 - rest quot)	Vermenigvuldigen gevolgd door delen (n1 * n2/n3).
+	(n1/n2 - som)	Telt TOS en SEC bij elkaar op tot een nieuwe TOS (16-bit).
-	(n1/n2 - verschil)	Aftrekken (16-bit).
/	(n1/n2 - quotiënt)	Delen van 16-bit heel getal (uitkomst: 16-bit)
/MOD	(n1/n2 - rest quot)	Berekent quotiënt en rest.
1+	(n - n+1)	Telt 1 op bij TOS.
2+	(n - n+2)	Telt 2 op bij TOS.
ABS	(n - u)	Bepaalt de absolute waarde van TOS.
D+	(d1/d1 - som)	Optellen (32-bit).
DABS	(d - ud)	Bepaalt de absolute waarde van een 32-bit getal).
DMINUS	(d - - d)	Niet-bewerking van een 32-bit getal.
M/MOD	(ud1/u2 - u3/ud4)	Delen van een 32-bit getal met overdracht van rest/quotiënt.
MAX	(n1/n2 - maximum)	Handhaaft de maximale waarde van n1 en n2 op de stack.
MIN	(n1/n2 - minimum)	Handhaaft de minimale waarde van n1 en n2 op de stack.
MINUS	(n - - n)	Niet-bewerking van TOS.
MOD	(n1/n2 - rest)	Berekent de rest van de deling n1/n2.

Boole- of logische bewerkingen

AND	(n1-n2 - logische EN)	Logische EN-funktie tussen TOS en SEC.
OR	(n1/n2 - logische OF)	Logische OF-funktie tussen TOS en SEC.
XOR	(n1/n2 - exclusief OF)	Exclusief OF-funktie tussen TOS en SEC.

Vergelijkende bewerkingen

0<	(n - f)	"waar" als n <0
0=	(n - f)	"waar" als n=0
<	(n1/n2 - f)	"waar" als n1<n2.
=	(n1/n2 - f)	"waar" als n1=n2.
>	(n1/n2 - f)	"waar" als n1>n2.

5.3 Forth basis-woordenschat

Geheugengerelateerde instrukties

@	(addr - n)	Haalt van het aangegeven adres de 16-bit inhoud op.
!	(n addr -)	Slaat op het aangegeven adres een 16-bit getal op.
BLANKS	(n1/n2 -)	Bezet, vanaf adres n1, n2 bytes met blanco karakters (CHR$(32)).
C@	(n1/n2 -)	Haalt de 8-bit inhoud van een geheugen plaats op (PEEK).
C!	(b addr -)	Slaat een 8-bit getal op in een geheugen-plaats (POKE).
CMOVE	(n1/n2/n3 -)	Brengt n3 karakters over van adres 1 naar adres 2.
ERASE	(n1/n2 -)	Bezet, vanaf adres n1, n2 bytes met nul.
FILL	(n1/n2/b -)	Vult, vanaf adres n1, n2 bytes met waarde b.
SP@	(- addr)	Zet de positie van de stack-wijzer op TOS.
TOGGLE	(n1/n2 -)	XOF-funktie tussen n2 en de byte op adres n1.

Struktuurwoorden

BEGIN . . . UNTIL	(f)	Voert de lus net zolang uit tot f="waar".
BEGIN WHILE REPEAT	(f)	De lust wordt uitgevoerd als f="waar".
BEGIN . . . AGAIN	(-)	Eindeloze lus tussen BEGIN en AGAIN.
DO . . . LOOP	(n1/n2 -)	Initialiseert een lus van n2 tot n1 met increment 1.
DO . . . + LOOP	(n1/n2 -)	Als DO . . . LOOP maar met TOS als increment.
I	(- n)	Zet de lusindex op TOS.
IF . . .ELSE . . ENDIF	(f)	Gestruktureerde IF-controle.
LEAVE	(-)	Verlaat de lus bij eerstvolgende LOOP.

Invoer-/Uitvoer-bewerkingen

.	(n -)	Drukt TOS af.
."	(-)	Geeft tekst af tot de eerstvolgende".
.R	(n1/n2 -)	Druk n1 af met n2 plaatsen recordlengte.
?	(addr -)	Drukt de inhoud van het adres af.
?TERMINAL	(- f)	Is "waar" als een toets wordt ingedrukt.
COUNT	(addr - addr+1/u)	Zet een stirng met lengtebyte om in TYPE-vorm.
CR	(-)	Geeft een wagenterugloop (CHR$(13)).

8.3 Forth basiswoordenschat

D.	(d -)	Drukt 32-bit getal af.
D.R	(d/n -)	Drukt 32-bit getal met een n positie af.
EMIT	(c -)	Geeft een karakter af.
EXPECT	(addr/n -)	Verwacht n karakters en slaat die op vanaf het adres.
KEY	(- c)	Leest het indrukken van een toets.
SPACE	(-)	Geeft een spatie af.
SPACES	(n -)	Geeft n spaties af.
TYPE	(addr/u -)	Geeft, vanaf het adres, u bytes af.
WORD	(c -)	Leest in het invoer-buffer tot het eerstvolgende karakter c.

Formatteren van getallen

#	(d - d)	Zet een cijfer om in een string.
#>	(d - addr/u)	Beëindigt de omzetting, string kan worden afgegeven met TYPE.
#S	(d - O O)	Zet resterende cijfers om in een string.
<#	(-)	Leidt het omzetten van getallen tot een strng in.
HOLD	(c -)	Plaats het karakter c op de eerstvolgende plaats in een numerieke string.
NUMBER	(addr - d)	Zet een string, vanaf het adres, om in een 32-bit getal.
SIGN	(n/d - d)	Voegt een voorteken aan de numerieke string toe.

Massageheugen-instrukties

-	(-)	Compileert eerstvolgende screen-inhoud.
B/BUF	(- n)	Geeft aantal bytes per buffer.
BLK	(- addr)	Systeemvariabele (feitelijke bloknummer).
BLOCK	(n - addr)	Brengt disk-blok over naar het adres.
EMPTY-BUFFERS	(-)	Alle buffers worden als "leeg" aangemerkt.
FLUSH	(-)	Alle 'gewijzigde' blokken worden naar disk teruggeschreven.
LIST	(n -)	Lijst screen met nummer n uit.
LOAD	(n -)	Compileert screen met nummer n.
SCR	(- addr)	Systeemvariabele (feitelijke screen-nummer).
UPDATA	(-)	Het laatst gebruikte blok wordt als 'gewijzigd' aangemerkt.

5.3 Forth basis-woordenschat

Defenitiewoorden

Woord	Stack	Omschrijving
: xyz	(-)	Woord xyz wordt opnieuw gedefinieerd.
;	(-)	Einde definitie van een woord.
BUILDS		
DOES	(-)	Voor het definiëren van definitiewoorden.
CODE xyz	(-)	Er wordt een machineprogramma opgesteld.
CONSTANT xyz	(n -)	Aan de constante xyz wordt waarde n toegekend.
IMMEDIATE	(-)	Het laatst gedefinieerde woord wordt als immediate (onmiddellijk uitvoerbaar) aangemerkt.
SMUDGE	(-)	Het laatst gedefinieerde woord wordt kenbaar gemaakt.
VARIABELE xyz	(n -)	Een variabele xyz wordt gedeklareerd en krijgt de waarde n toegekend.

Woordenlijst

Woord	Stack	Omschrijving
ASSEMBLER	(-)	Roept assembler-woordenlijst op.
CONTEXT	(- addr)	Geeft het adres van de Context-woordenlijst (zoek-woordenlijst)
CURRENT	(- addr)	Geeft het adres van de Current-woordenlijst (definities-woordenlijst).
DEFINITIONS	(-)	Zet CURRENT op CONTEXT.
EDITOR	(-)	Roept Editor-woordenlijst op.
FORTH	(-)	Zet CONTEXT op FORTH.
VLIST	(-)	Lijst alle woorden van de Context-woordenlijst uit.
VOCABULARY xyz	(-)	Definieert nieuwe woordenlijst xyz.

Systeemwoorden

Woord	Stack	Omschrijving
'xxx	(- addr)	Zoekt het adres van woord xxx in de adreslijst.
(	(-)	Begin van een commentaar; te eindigen met).
,	(n -)	Slaat 16-bit getal op in de adreslijst.
ABORT	(-)	Forceert fout-onderbreking.
ALLOT	(n -)	Bezet n bytes ub de adreslijst.
C,	(b -)	Slaat 8-bit getal op in de adreslijst.
FORGET abc	(-)	'Vergeet' alle woorden vanaf abc (inclusief abc).

5.3 Forth basiswoordenschat

HERE	(- addr)	Verwijst naar de eerstvolgende vrijepositie in de adreslijst.
PAD	(- addr)	Buffert 64 bytes hoger dan HERE.

8/5.4

Programmastrukturen

Nu we ons met de belangrijkste woorden van FORTH vertrouwd hebben gemaakt is het tijd iets te zeggen over gestruktureerd programmeren. Het fundamentele woord voor programmastrukturen is een "als . . . dan"-controle die natuurlijk ook in de woordenschat van FORTH voorkomt.

Hier meteen een principiële eigenschap van controles in FORTH. Om aan te geven of een voorwaarde "waar" of "niet waar" is dient in FORTH ook weer het bovenste stack-element TOS. Een waarde niet gelijk aan nul komt overeen met de toestand "waar". Testen op voorwaarden levert normaal op de stack een 0 of -1 (16 binaire enen) op. Tenslotte blijft bij het op ongelijkheid testen van twee getallen het verschil van die getallen over.

Genoeg theorie nu, eerst maar eens een praktisch voorbeeld: we willen een woord definiëren dat test of een getal deelbaar is door een ander getal en dat het resultaat daarvan niet-gecodeerd afgeeft. De definitie is uiterst eenvoudig:

```
: DEELBAAR? MOD IF ." NIET "ENDIF
." DEELBAAR " ;
```

Wat gebeurt hier nu? Daarvoor slaan we ons instructie-overzicht op: MOD berekent bij een deling de rest, met andere woorden gaat de deling op dan is de rest nul. Een rest zorgt op de stack voor een getal niet gelijk aan nul. IF bekijkt nu het bovenste stack-element (waarbij het wordt weggenomen). Een waarde anders dan nul heeft tot gevolg dat de programmatekst wordt uitgevoerd tot de eerstvolgende ENDIF. In ons voorbeeld wordt bij het voorkomen van een rest het woord "niet" afgedrukt gevolgd door de melding "deelbaar". Uiteraard kent FORTH ook nog een andere (in het BASIC van de C 64 helaas niet bekende) vorm van de IF-controle, namelijk de IF-controle met ELSE-vertakking, in ongecodeerde vorm ongeveer te formuleren als "als . . . dan . . . anders". Ons voorbeeld zou men dus ook kunnen formuleren als:

```
: DEELBAAR? MOD IF ." NIET DEEL-
BAAR " ELSE ." DEELBAAR " ENDIF ;
```

Is de voorwaarde "waar" dan wordt het gedeelte tussen IF en ELSE uitgevoerd, zo niet dan het deel tussen ELSE en ENDIF. In veel FORTH-versies wordt in plaats van het woord ENDIF ook wel het woord THEN gebruikt. Dergelijke controles zijn natuurlijk ook aan elkaar te rijgen.

Andere hulpmiddelen voor het structureren van een programma zijn de lussen. FORTH kent alle denkbaar mogelijke lussen. In ons geval beginnen we echter met de eenvoudigste (en in bijzondere gevallen zo mogelijk ook de nuttigste) lus, de eindeloze

5.4 Programmastrukturen

lus. Het volgende woord is hiervan een voorbeeld:

: ENDELOOS BEGIN ." IK BEN EEN EINDELOZE LUS" CR AGAIN ;

Bij het oproepen daarvan wordt voortdurend de tekst "IK BEN EEN EINDELOZE LUS" afgegeven, net zolang tot de stroom uitvalt of de programmeur een RESET geeft. De structuur is duidelijk, alles tussen BEGIN en AGAIN wordt eeuwig herhaald. Hier dient opgemerkt te worden dat structuurwoorden alleen in definities zijn toegestaan omdat ze spronginstrukties inleiden. In de normale invoermodus zou zonder meer invoeren van BEGIN ." EEN OF ANDER" AGAIN een foutmelding tot gevolg hebben.
Als volgende dienen lussen te worden genoemd die door het optreden van voorwaarden kunnen worden verlaten. Ook hiervan een voorbeeld:

: VOORBEELD1 BEGIN ." TOETS INDRUKKEN" ?TERMINAL UNTIL ;

Dit woord geeft net zolang de tekst "TOETS INDRUKKEN" te zien tot de gebruiker aan de verzoek voldoet. Het woord ?TERMINAL controleert of een toets werd ingedrukt (bij veel C-64-FORTH-versies wordt gecontroleerd of de STOP-toets werd ingedrukt). Is dat het geval dan wordt "waar" op de stapel gezet. De instrukties tussen BEGIN en UNTIL worden dan net zolang herhaald tot de door UNTIL geteste voorwaarde "waar" is. De instruktie wordt dus minstens eenmaal afgewerkt.

Een andere controle voor een onderbreking blijkt uit het volgende voorbeeld:

: VOORBEELD2 BEGIN ?TERMINAL 1-WHILE ." TOETS INDRUKKEN" REPEAT ;

Ook hier wacht het programma tot een toets wordt ingedrukt. De mededeling "TOETS INDRUKKEN" wordt echter alleen afgegeven als dat feit nog niet eerder werd geonstateerd. De instructies tussen BEGIN en WHILE worden afgewerkt en leveren de voorwaarde-incidicatie (vlag) op de stack. De instructies tussen WHILE en REPEAT worden alleen afgewerkt als de voorwaarde "waar" is. Na REPEAT wordt dan teruggekeerd naar BEGIN en wordt de voorwaarde opnieuw onderzocht. De lus test dus de onderbrekingsvoorwaarde vóór de uitvoering en hoeft dus niet beslist helemaal uitgevoerd te worden.

Tenslotte dienen nog de lussen vermeld te worden waarvan het aantal doorgangen tevoren wordt bepaald. Een dergelijke constructie vinden we ook in DO . . . LOOP. DO verwacht twee getallen op de stack; als bovenste de aanvangswaarde van de lus en als SEC de eindwaarde ervan. LOOP verhoogt de index met één en herhaalt de lus als de eindwaarde nog niet bereikt werd. In het volgende voorbeeld zouden dus de getallen 1 t/m 10 worden afgegeven:

: GETALLEN 11 1 DO I . LOOP ;

De lus houdt op als de teller de waarde 11 bereikt en begint bij 1. De instructie I zet alleen de feitelijke lusindex op de stack om die weer af te kunnen geven.

5.4 Programmastrukturen

DO . . . LOOP kan ook als DO . . . +LOOP worden gebruikt. In dat geval is het increment (de waarde waarmee de index wordt verhoogd) vrij kiesbaar. Aftellen van 10 tot 1 zou als volgt moeten worden geformuleerd:

: AFTELLEN 0 10 DO I. -1 +LOOP ;

Hier wordt dus telkens -1 bij de lusindex opgeteld, net zolang tot deze de waarde 0 bereikt. DO . . . LOOP constructies zijn de traagste lussen in FORTH, maar altijd nog ongeveer een faktor 10 sneller dan het vergelijkbare FOR . . . NEXT in BASIC.

Hiermee zijn de belangrijkste structuurwoorden van FORTH toegelicht. Gekunstelde structuurwoorden als GOTO treft men in FORTH niet aan. Wie met GOTO wil programmeren heeft het nut van FORTH nog niet begrepen. Een GOSUB kan men zich besparen omdat elk FORTH-woord toch al een subprogramma voorstelt en eenvoudig door het noemen ervan kan worden opgeroepen.

8/6

COMAL

Inhoud

8/6.1 Introduktie

8/6.1

Introduktie

COMAL is een programmeertaal, waarin de positieve aspecten van BASIC, PASCAL en LOGO verenigd zijn. De taal is in 1973 ontstaan, maar werd pas onlangs interessant voor P.C.-gebruikers. Zoals voor bijna iedere naam van een programmeertaal het geval is, gaat ook achter COMAL een afkorting schuil, n.l. 'COMmon Algorithmic Language'. De huidige versie voor de C64, bestaat uit een programma, dat in het geheugen geladen wordt en dan een gebruikerssysteem vormt. Het gaat hier dus niet om een compiler die het geschreven programma omzet in machinetaal. Deze taalversie heeft echter ook een nadeel tot gevolg: na het starten van het COMAL-programma blijven er namelijk nog maar 9902 Bytes voor de programmeur ter beschikking over. De zojuist genoemde versie is overigens vrij van auteursrechten, dat betekent dat het gecopieerd mag worden en dat het tegen kostprijs te verkrijgen is.
Naas deze versie bestaat er een ROM-versie, die over meer vrijgeheugen beschikt. Deze bevindt zich in een insteekmodule die niet erg goedkoop is. Hierna zal de programma-versie van COMAL 0.14 beschreven worden.

ELEMENTEN van COMAL uit ANDERE TALEN
COMAL moet men niet als een volledig nieuwe programmeertaal opvatten, want op de keper beschouwd, zijn slechts enkele gedeelten ervan werkelijk nieuw. Veeleer heeft men gepoogd, alle positieve facetten van enkele programmeertalen te verenigen om daaruit een nieuwe taal te creëren.

BASIC is de basistaal van COMAL en er bestaat een hiërarchische compabiliteit van BASIC naar COMAL. Dat betekent dat iedereen, die BASIC enigszins beheerst, ook met COMAL kan omgaan. Geleidelijk aan zal een ieder zich de fijne kneepjes van COMAL eigen maken. Er wordt vaak beweerd, dat BASIC de programmeerstijl van de gebruiker bederft; ondanks de nauwe relatie tussen BASIC en COMAL, is dit element uit COMAL weggenomen.
Het zal de meer ervaren BASIC-programmeur opvallen, dat er in COMAL geen mogelijkheden zijn om een commando af te korten. Dat komt, doordat in COMAL de mogelijkheid bestaat nieuwe commando's te definiëren.

Pascal
COMAL heeft uit PASCAL vooral de mogelijkheid tot gestructureerd programmeren over genomen. Dit onderwerp wordt nog in een afzonderlijk hoofdstuk uitvoerig behandeld. Gestructureerde instructie-reeksen zijn gemakkelijker te programmeren, dan de onvermijdelijke

6.1 Introduktie

omschrijvingen in BASIC, door middel van omslachtige 'IF'-instructiereeksen.
Met instructies zoals bijvoorbeeld 'REPEAT...UNTIL' of 'CASE...WHEN...', worden instructiewoorden toegepast, die dichter bij de gewone spreektaal en de normale manier van denken staan, dan de omschrijvingen met behulp van 'IF'.
Het programma wordt daardoor gemakkelijker leesbaar en het is uitgesloten, dat men zijn eigen programma niet meer kan lezen. Het is tenslotte toch de bedoeling van de programmeur dat zijn programma niet alleen de oplossing van een probleem levert, maar tevens dat het overzichtelijk en goed leesbaar is weergegeven.

Logo
Ook de programmeertaal LOGO leverde essentiële delen aan COMAL. In COMAL bestaat namelijk eveneens de mogelijkheid om respectievelijk nieuwe commando's te definiëren en subprogramma's door middel van hun namen aan te roepen.
Ook kan men *functies* definiëren, die niet, zoals in BASIC, uit één regel bestaan, maar als *subprogramma's* met een willekeurig aantal regels. Ook dit bevordert de overzichtelijkheid en vooral de leesbaarheid enorm. In BASIC moet men de functie van een subprogramma met een 'REM'-regel kenbaar maken. In COMAL daarentegen, kan men het subprogramma aanroepen met een naam, die met zijn functie overeenkomt.
Ook de bekende 'TURTLE-graphics' van LOGO is overgenomen. Deze is eenvoudig te programmeeren en aan de menselijke wijze van denken aangepast. Op zijn allereenvoudigst gaat het hier om een tekenpen (TURTLE), die om zijn eigen as kan draaien en heen en weer bewogen kan worden. De naam 'TURTLE' stamt uit de begintijd van computergraphics, waarbij de cursor gelijkenis met een schildpad vertoonde.
Een nauwkeurige beschrijving van deze grafische mogelijkheden volgt later in een apart hoofdstuk.

Behandeling van fouten
Zoals reeds werd opgemerkt is de vrije geheugenruimte van de C64 bij gebruik van COMAL 0.14 erg krap. Om deze reden zijn de teksten van foutmeldingen naar een bestand op diskette overgeheveld en steeds, wanneer er een fout optreedt, wordt deze tekst uit dat bestand gelezen. In COMAL wordt n.l. iedere regel direct na invoer op zijn syntactische juistheid getest. Toetst men een spelfout of een onjuiste instructie in, dan slaat het drijfmechanisme van de diskette aan, hetgeen velen algauw op de zenuwen gaat werken. aarom is er een commando 'SETMSG-', dat, indien er een fout gemaakt is, alleen het nummer van de fout op het beeldscherm aangeeft. Als men dus vloeiend met COMAL wil werken, kan men het best een lijst van foutmeldingen met hun nummers laten printen en bij iedere foutmelding deze lijst raadplegen. Wil men weer terugschakelen naar de normale foutmelding dan kan dat met de instructie 'SETMSG+'.

FORMATEN, LADEN en OPSLAG
COMAL-programma's kunnen in twee verschillende formaten worden opgeslagen. Enerzijds is er het normale formaat, waarbij de commando's (instructies), net als in BASIC, in afgekorte en gecodeerde vorm worden opgeslagen. Deze codering is specifiek voor het COMAL 0.14-formaat en daarvoor staan de uit BASIC bekende commando's 'LOAD' en 'SAVE' ter beschikking. De werkadressen op het

6.1 Introductie

drijfmechanisme moet men niet aangeven: dus *niet* 'LOAD grafiek,8,' invoeren, maar alleen 'LOAD grafiek'. Daarnaast is er nog het commando 'CHAIN', waarmee programma's geladen en daarna automatisch gestart kunnen worden. Syntactisch is deze instructie identiek met 'LOAD'.

In BASIC is het zeer lastig programma's, die geschreven zijn in verschillende versies van BASIC, onderling uit te wisselen. Om deze situatie in COMAL te ondervangen, kunnen programma's in tekstvorm opgeslagen worden.
De programma-instructies worden dan niet gecodeerd maar als tekst in het geheugen geplaatst. Daardoor kan dit tekstbestand ook door een andere versie van COMAL geladen en daarna opnieuw in het geheugen gecodeerd worden. Het commando voor uitlezen is in dit geval: 'LIST‹programmanaam›' en het commando voor inlezen:
'ENTER‹programmanaam›'
Programma's, die in dit formaat worden uitgevoerd en door middel van 'ENTER' worden ingelezen, kunnen zonodig aan een reeds in het geheuge aanwezig programma worden toegevoegd.
Het commando 'ENTER' werkt dus op dezelfde wijze als het in BASIC gebruikelijke commando 'MERGE'. Dergelijke tekstbestanden zijn, door het onbreken van de codering, aanzienlijk langer dan gecodeerde programmabestanden.

Gestructureerd programmeren
Gestructureerd programmeren is identiek met overzichtelijk, eenvoudig en goed leesbaar programmeren. Dit alles wordt in BASIC dikwijls verwaarloosd en daarom moet de echte BASIC-programmeur zich serieus met gestructureerd programmeren bezig houden.

Gestructureerd listen
COMAL beschikt evenals BASIC over het commando 'LIST'. Daarmee kan het opgeslagen programma uit het geheugen gehaald worden. Analoog aan BASIC zijn de volgende opties van 'LIST' mogelijk:

Instructie:	Resultaat:
LIST	toont het gehele programma
LIST rnl-	toont het programma vanaf regelnummer rnl
LIST -rnl	toont het programma tot rnl
LIST rnl-rn2	toont het programma van rnl tot rn2

Gedurende het uitlezen kan de weergave zoals in BASIC door de CONTROL-toets vertraagd worden. Het weergaveproces kan eveneens met de spatietoets gestopt en met een volgende druk op die toets weer voortgezet worden.
Het grote verschil met het LIST-commando van BASIC is echter, dat het programma gestructureerd wordt weergegeven. Dat betekent dat overal waar structurele instructies – zoals b.v. lussen en conditionele instructies – toegepast zijn het programmagedeelte, waarin zo'n structuur voorkomt, één positie naar rechts is verschoven.
Hierdoor wordt de programmalisting natuurlijk veel overzichtelijker en eenvoudiger te lezen. Ook geneste structuren zijn in één oogopslag duidelijk, doordat deze meerdere posities naar rechts zijn verschoven.
De gestructureerde opmaak is daarom van zo'n grote betekenis, omdat er in COMAL aanzienlijk meer structurele instructies zijn dan in BASIC. Het is echter ook mogelijk het programma ongestructureerd weer te geven, dus zoals in BASIC. Dit bereikt men met het commando

6.1 Introduktie

'EDIT', dat syntactisch overeenkomt met het commando 'LIST', maar het programma zodanig weergeeft dat de structuren niet langer in de listing te herkennen zijn.

Programmalussen
Programmalussen zijn elementaire middelen bij het programmeren die iedere computer kan uitvoeren. Er bestaan echter verschillende vormen van deze faciliteit. In BASIC komt alleen de FOR...NEXT-lus voor. Deze beperking is in COMAL verholpen. In die programmeertaal kan men namelijk de ingewikkelde omschrijvingen vergeten.

FOR...STEP...ENDFOR
Zoals in BASIC, is het ook in COMAL mogelijk lussen te programmeren. In tegenstelling tot BASIC heeft het afsluitingscommando van de lus niet 'NEXT', maar 'ENDFOR'. Gebruikt men desondanks toch 'NEXT', dan vervangt COMAL dat door 'ENDFOR'. In een daarop volgende listing verschijnt echter op zulke plaatsen een 'ENDFOR', dit om de hiërarchieke compatibiliteit van BASIC naar COMAL te handhaven.

De instructie 'FOR' moet op een aparte regel staan. Aan het einde van die regel staat 'DO' om de lijst van instructies in te leiden. 'DO' moet men *niet* invoeren, daar dit automatisch door COMAL wordt toegevoegd.
De lus-structuur ziet er dus als volgt uit:

```
FOR‹lusvariabele›:=‹beginwaarde› TO
‹eindwaarde› (STEP‹stapgrootte›) DO
...
‹lusinstructies›
...
ENDFOR(‹lusvariabele›)
```

REPEAT...UNTIL
Met het REPEAT...UNTIL-commando kan men een lus beschrijven, die bij het programmeren vrij vaak wordt toegepast, maar die in BASIC alleen door een omschrijving gerealiseerd kan worden.
In het programma ziet dit commando er als volgt uit:

```
REPEAT...
...
‹lusinstructies›
...
UNTIL‹conditie›
```

De instructies binnen een lus worden net zo vaak herhaald tot een conditie die achter 'UNTIL' staat **waar** is. Voorbeeld:

```
REPEAT
INPUT 'TOETS EEN CODEWOORD
IN, S.V.P.:';CODE$
UNTIL CODE$='COMAL'
```

De computer vraagt u hier net zo lang een codewoord in te voeren, totdat u het juiste codewoord 'comal' heeft ingevoerd.

WHILE...ENDWHILE
Ook de WHILE...ENDWHILE-instructie dient er voor programma-ideeën, die zeer dicht bij het gewone taalgebruik staan, zo eenvoudig mogelijk op de computer te kunnen realiseren. In het programma wordt de instructie als volgt toegepast:

```
WHILE‹conditie›
...
‹lusinstructies›
...
ENDWHILE
```

Zolang de achter 'WHILE' aangegeven

6.1 Introduktie

conditie **waar** is, worden de lusinstructies uitgevoerd. Is deze **niet langer waar**, dan wordt het programma voortgezet op de regel die direct volgt na het lusbegrenzingscommando 'ENDWHILE'.
Er is echter ook een éénregelige versie van deze instructie beschikbaar, die *geen* begrenzingsinstructie nodig heeft. Deze wordt als volgt toegepast:

```
WHILE‹conditie› DO ‹lusinstructie›
```

Men kan deze verkorte versie gebruiken als het slechts om één lusinstructie gaat. Nadat de conditie in de instructie niet-waar is geworden, gaat het programma op de volgende regel verder.

Vertakkingen
Ook in COMAL beschikt men over vertakkingen, resp. conditionele instructies. In tegenstelling tot BASIC mogen zulke conditionele structuren uit meer dan één regel bestaan. De neiging om een vertakking in een bepaald gedeelte van een programma door middel van GOTO te realiseren, is hier dus verlaten; in plaats daarvan probeert men bepaalde structuren, die onder een 'ware' conditie uitgevoerd moeten worden, direct ter plaatse te verwerken. Daarmee wordt een dwangmatig heen- en weerspringen – zoals in BASIC – in hoge mate vermeden.

IF...ELSE...ELIF...ENDIF
Evenals BASIC kent COMAL ook de conditionele instructie 'IF'. In BASIC kan men die alleen in één regel toepassen; wil men desondanks meerregelige instructiereeksen invoeren, dan moet dat omslachtig beschreven worden. Dit is in COMAL niet nodig. Om wille van de hiërarchische compabiliteit is de eenvoudige éénregelige variant eveneens beschikbaar. De syntax ervan komt met die van BASIC overeen en ziet er als volgt uit:

```
IF‹conditie› THEN ‹instructie›
```

De meest eenvoudige syntaktische vorm van een meerregelige vertakking ziet er zo uit:

```
IF‹conditie› THEN
  ‹instructie 1›
  ‹instructie 2›
  ...
  ‹instructie n›
ENDIF
```

Men hoeft dus alleen na 'THEN' op een nieuwe regel te beginnen en de lijst met instructies met·ENDIF' af te sluiten.
De meest ingewikkelde vorm ziet er zo uit:

```
IF ‹conditie› THEN
  ‹instructies›
ELIF ‹conditie› THEN
     ‹instructies›
ELIF ‹conditie› THEN
     ‹instructies›
ELSE
     ‹instructies›
ENDIF
```

Zoals men ziet, kunnen naar believen vele 'ELIF'-instructies ingevoegd worden. De evaluatie van deze structuur gaat als volgt in zijn werk: eerst wordt de 'IF'-conditie getoetst. Als deze waar is, worden de instructies onder 'IF' uitgevoerd en het programma wordt na 'ENDIF' voortgezet.
Gaat de 'IF'-conditie niet op, dan worden succesievelijk de 'ELIF'-condities getoetst. Komt de computer er bij één, die waar is, dan worden de daarbij horende instructies uitgevoerd en de computer

6.1 Introduktie

werkt daarna de rest van het programma na 'ENDIF' af. Als alle 'IF'-en 'ELIF'-condities niet waar zijn, dan worden de instructies na het 'ELSE'-commando uitgevoerd. Hierbij een voorbeeld:

```
INPUT 'Voer een getal in s.v.p.: '; getal
IFgetal < 100 THEN
   PRINT 'Het getal is kleiner dan 100'
ELIF getal < 200 THEN
   PRINT 'Het getal ligt tussen 100 en
   200'
ELIF getal < 300 THEN
   'Het getal ligt tussen 200 en 300'
ELIF getal < 400 THEN
   PRINT 'Het getal ligt tussen 300 en
   400'
ELSE
   PRINT 'Het getal is groter dan 399'
ENDIF
END
```

Er wordt nog maals op gewezen, dat indien een 'IF'- of 'ELIF'-conditie waar is, géén van de andere 'ELIF'-condities getoetst worden.

CASE...WHEN...OTHERWHISE... ENDCASE

De 'CASE'-instructie werkt op vergelijkbare wijze als de 'IF'-instructie; opnieuw is geprobeerd, de bewoording en de vorm van de programmastructuur aan de gewone taal aan te passen. De algemene vorm van de 'CASE'-instructie is:

```
CASE <expressie> OF
WHEN <waarde1>
        <instructies>
WHEN <waarde2>
        <instructies>
OTERWISE
        <instructies>
ENDCASE
```

Natuurlijk kunnen er ook bij de 'CASE'-instructie gedeelten worden weggelaten. Minimaal moet echter de 'CASE'- en de 'ENDCASE'-instructie aanwezig zijn. Bij de CASE-instructie wordt getoetst of de expressie, die uit een numerieke waarde of uit een reeks symbolen kan bestaan, een bepaalde waarde heeft. Deze expressie wordt achter de 'CASE'-instructie gezet. Daarop volgen na 'WHEN' één of meer waarden; worden er meerdere equivalente waarden in de conditie opgenomen dan dienen deze onderling door komma's gescheiden te zijn. Komt een van deze waarden in de expressie voor, dan worden de instructies in de volgende regels uitgevoerd. Het programma wordt dan na de 'ENDCASE'-instructie voortgezet. Heeft echter de expressie geen van de waarden die achter de 'WHEN'-instructies voorkomen dan worden de instructies na 'OTHERWISE' uitgevoerd.

Voorbeeld:

```
...
INPUT 'Zal ik deze instructie uitvoe-
ren?:' antw$
CASE antw$ OF
WHEN 'j', 'ja', 'y', 'yes'
PRINT 'instructie wordt uitgevoerd'
...
WHEN 'n', 'nee', 'no'
PRINT 'instructie wordt niet uitge-
voerd.'
...
OTHERWISE
PRINT 'Invoer niet begrepen.'
PRINT 'Herhalen s.v.p.'
...
ENDCASE
```

Eerst wordt gevraagd of een bepaalde instructie uitgevoerd moet worden. Daarna komt de evaluatie met 'CASE'. Is het ant-

6.1 Introduktie

woord 'j', 'ja', 'y' of 'yes' dan wordt de instructie uitgevoerd. Is het antwoord 'n', 'nee' of 'no' dan wordt de instructie niet uitgevoerd. Als het antwoord niet met een van de 7 keuzemogelijkheden overeenstemt dan worden de instructies na 'OTHERWISE' uitgevoerd (deze zijn hier alleen maar aangeduid).

SUBPROGRAMMA'S en VARIABELEN
In tegenstelling tot BASIC kunnen in COMAL nieuwe commando's en functies in afzonderlijke structuren (subprogramma's) gedefinieerd worden. Voor gebruikers van BASIC zal de verschillende behandeling van variabelen nieuw zijn; terwijl BASIC alleen algemene variabelen kent, bestaan er in COMAL algemene èn locale variabelen.

DEFINITIE van NIEUWE COMMANDO's (PROCEDURES)
In COMAL is het - in tegenstelling tot BASIC - mogelijk nieuwe commando's te definiëren. Eenvoudig gezegd kan men zulke commando's zo toepassen, dat een subprogramma niet met een regelnummer, maar met een naam wordt aangeroepen. Als men van deze constructiemogelijkheid gebruik maakt om nieuwe commando's te definiëren dan kan men, - net als bij gewone commando's - een lijst met parameters en variabelen met de naam van het subprogramma meegeven. In andere programmeertalen wordt deze constructie ook wel met 'PROCEDURE' aangeduid (b.v. in PL / 1 en PASCAL). In een typisch geval ziet de volgorde van de commando's er als volgt uit:

```
PROC ‹naam van de structuur› (‹lijst met parameters en/of variabelen›)
›lijst met instructies›
ENDPROC‹naam van de structuur›
```

Ter verduidelijking volgt een eenvoudig voorbeeld:

```
INPUT 'Aantal regels over te slaan:'
;regels
REGELOPVOER(regels)
PRINT 'Regelopvoer uitgevoerd.'
...
...
PROC REGELOPVOER(opvoer)
FOR a:= 1 TO opvoer DO
PRINT
ENDFOR a
ENDPROC REGELOPVOER
```

In dit programmavoorbeeld wordt eerst gevraagd hoeveel regels er overgeslagen moeten worden. De ingegeven waarde van de variabele 'regelopvoer' wordt opgeslagen. Daarna wordt het nieuwe commando 'REGELOPVOER; en zijn variabele 'regels' aangeroepen. Deze wordt in het daaropvolgende subprogramma gedefiniëerd. Hierin wordt de overgedragen parameter verwerkt. Dan wordt – in overeenstemming met de variabele 'opvoer' (het aantal regels, dat overgeslagen moet worden), door middel van een lus met een 'PRINT'-instructie uitgevoerd. Als de instructie 'ENDPROC' bereikt is, wordt de verdere uitvoering van het hoofdprogramma voortgezet.
Als het taalelement 'REF' bij de definitie van de 'PROCEDURE' vóór het argument van de variabele gezet wordt (b.v.: PROC REGELOPVOER REF opvoer), dan worden tijdens de uitvoering van de 'PROCEDURE' niet de **actuele waarden** van de parameters, maar de **namen** van de parameters gebruikt (call bij reference). Dit is vooral belangrijk als bij het aanroepen van de Procedure de actuele variabele namen binnen de Procedure getalwaarden moeten worden toegekend.

6.1 Introduktie

DEFINITIE van NIEUWE FUNCTIES
Op ongeveer gelijke wijze kunnen eveneens functies als commando's gedefinieerd worden. Er mogen in het argument van de functie een willekeurig aantal variabelen voorkomen en de reeks van opeenvolgende regels mag een willekeurige lengte hebben. De functiewaarde, die door de 'FUNCTIE' berekend is, wordt met het commando 'RETURN<variabele>' aan het hoofdprogramma teruggeven.
Hiervan een voorbeeld:

```
INPUT 'integer' ;z
PRINT z
PRINT frac(z)
...
FUNC frac(a)
  fr=a - int(a)
  return(fr)
ENDFUNC(frac)
```

In het eerste gedeelte van dit voorbeeld, wordt de invoer van een integer gevraagd, die in de variabele 'z' wordt opgeslagen en meteen op het beeldscherm verschijnt. Daarna wordt door middel van een 'PRINT'-commando de functie frac() aangeroepen, die vervolgens wordt gedefinieerd. Aan het begin staat het commando 'FUNC en daarna volgen de naam van de functie (frac) en de functievariabele (a), zoals deze in dit subprogramma worden gebruikt. Dan wordt de vorm van de functie aan gegeven (fr = a – int(a)). De waarde ervan wordt met 'RETURN' aan het hoofdprogramma teruggegeven. Aan het slot staat 'ENDFUNC' met de naam van de functie, om deze ruimtelijk af te sluiten.

LOKALE en ALGEMENE VARIABELEN(CLOSED)
In BASIC komen alleen algemene variabelen voor. Dat zijn variabelen, die op ieder moment in ieder programmasegment aangeroepen kunnen worden. Locale variabelen komen bij het definiëren van Commando's en FUNCTIES voor. De argumenten van FUNCTIES zijn altijd lokaal:

```
PROC DELEN(naam$)
FUNC PRIEMGETAL(getal)
```

Dit betekent dat de gekenmerkte variabelen ook in het hoofdprogramma kunnen voorkomen, maar dat hun waarde echter niet veradert, indien de waarden van functieargumenten in het subprogramma veranderd worden. De beide variabeletypen zijn dus ondanks hun gelijke namen onafhankelijk. Hiervan een programmavoorbeeld:

```
a=3 : b=4
PRINT 'a=';a ; 'b= ',b
product(a)
PRINT 'a='a;,'b=',b
...
PROC product(a)
a=a*b:b=5
PRINT 'a=' , a;'b=',b
ENDPROC product
```

Hieruit krijgt men het volgende resultaat:

```
a=3    b=4
a=12   b=5
a=3    b=5
```

De waarde van 'a' wordt in het subprogramma veranderd. Desondanks heeft 'a' naderhand in het hoofdprogramma opnieuw dezelfde waarde als helemaal aan het begin. In het subprogramma was 'a' dus een lokale variabele. De waarde van b

6.1 Introduktie

werd in het subprogramma nog gewijzigd van waarde. 'b' is dus een globale variabele (zoals altijd in BASIC).

Er bestaat echter ook de mogelijkheid alle outputvariabelen van het subprogramma om te zetten in lokale variabelen. Dit is met het commando 'CLOSED' mogelijk, dat dan aan het einde van de beginregel staat, zoals bijvoorbeeld:

```
PROC product(a) CLOSED
```

Wanneer het vorige programma aldus gewijzigd wordt, dan is 'b' in het subprogramma een locale variabele geworden; de uitvoer van dit programma ziet er nu zo uit:

```
a=3    b=4
a=12   b=5
a=3    b=4
```

Thans heeft ook 'b' in het hoofdprogramma opnieuw zijn oorspronkelijke waarde teruggekregen, ondanks de verandering van zijn waarde in het subprogramma. 'b' is nu een lokale variabele in het subprogramma.

GRAFISCH SYSTEEM MET HOGE RESOLUTIE

Gezien zijn grootte beschikt de Commodore 64 over een zeer goed oplossend vermogen. Dit wordt echter in BASIC met geen enkel commando ondersteund. In COMAL is dat door een hele reeks grafische instructies verholpen. Met het commando 'SETGRAPHIC' kan men naar de grafische modus omschakelen. Toetst u 'SETGRAPHIC 0' in, dan werkt u op een beeldscherm met een hoge resolutie, in twee grijstonen (voorgrond en achtergrond) en met een matrix van 320 x 200 beeldpunten.

Met 'SETGRAPHIC 1' werkt u op een kleurenbeeldscherm met 4 kleuren (3 voorgrond en 1 achtergrond) en met een resolutie van 160 x 200 beeldpunten.
Als u het grafische systeem reeds eerder ingesteld heeft, is het commando 'SETGRAPHIC' (zonder argument) voldoende om naar het daarvoor reeds vastgelegde grafische beeldscherm om te schakelen.

OPBOUW VAN HET BEELDSCHERM EN DE MODI

Zoals reeds vermeld werd, is het oplossend vermogen afhankelijk van de ingestelde grafische modus. Bij vele grafische commando's is een punt direct adresseerbaar; hierbij moet u erop verdacht zijn, dat het punt met coördinaten (0,0) zich in de linker beneden hoek bevindt (dit in tegenstelling tot de 'normale' coördinatenverdeling van de C64). De verticale uitbreiding met 200 punten loopt tot het punt (0,199) in de linker bovenhoek.
Bij de invoer van een reeks punten dient u altijd eerst de x-coördinaat en daarna - gescheiden door een komma - de y-coördinaat in te voeren.
De uitbreiding naar rechts is afhankelijk van de ingestelde grafische modus en bedraagt 320 of 160 beeldpunten. De rechter benedenhoek komt respectievelijk overeen met de coördinaten (319,0) en (159,0).
Met het commando 'SETTEXT' kan het grafische beeldscherm weer op het tekstbeeldscherm teruggeschakeld worden. Op het grafische beeldscherm zijn er twee verdelingsmodi mogelijk; met het commando 'FULLSCREEN' beslaat het grafische beeldscherm het hele beschikbare beeldscherm. Vooral in de directe modus is het interessant het commando 'SPLITSCREEN'

6.1 Introduktie

in te voeren; daarmee worden aan de bovenrand van het beeldscherm twee regels ingevoegd, waarop uitvoerinstructies ingevoerd kunnen worden. In het vierkleurensysteem is dit echter niet mogelijk.

Het grafische systeem TURTLE, dat ook in de programmeertaal LOGO toegepast wordt, is een zeer eenvoudige en gemakkelijk te begrijpen toepassing van een grafisch systeem. Oorspronkelijk werd het voor kinderen ontworpen; het betreft hier uiteraard een meer uitgebreide en verbeterde versie.
Na het inschakelen van het grafische systeem verschijnt er in het midden van het beeldscherm een driehoek, die TURTLE (Ned.: schildpad) genoemd wordt. Met eenvoudige commando's kan deze schildpad nu telkens om een bepaald aantal stappen gedraaid, vooruit, achteruit of naar links of rechts bewogen worden. Aan deze elementaire commando's zijn nog enkele toegevoegd, waarmee men de schildpad ook direct naar een bepaald punt kan bewegen.
Tenslotte kan voor iedere beweging van de schildpad vastgelegd worden of deze zich alleen over het beeldscherm moet bewegen of dat deze daarbij tevens een lijn moet tkenen, dat wil zeggen een soort spoort moet achterlaten.

DE GRAFISCHE COMMANDO'S
In alfabetische volgorde worden alle tot dusver nog niet genoemde grafische commando's opgesomd en in het kort uitgelegd. Aan het slot volgen nog enkele, die met grafische faciliteiten te maken hebben.

BACK ‹numerieke expressie›

Met deze instructie beweegt de schildpad het aantal stappen, dat na BACK is opgegeven, naar achteren. Het gaat hier om een relatieve instructie t.o.v. de actuele positie van de schildpad.

CLEAR

Met de instructie CLEAR wordt het grafische beeldscherm schoon gewist; dat wil zeggen: alle beeldpunten krijgen de kleur van de achtergrond (BACKGROUND).

DRAWTO ‹x-coördinaat›,‹y-coördinaat›

Met DRAWTO beweegt de schildpad vanaf zijn actuele positie naar een punt, waarvan de coördinaten zijn opgegeven (absolute positionering). Hierbij wordt altijd een lijn getekend.

FILL ‹x-coördinaat›,‹y-coördinaat›

Met dit commando wordt, uitgaande van de opgegeven coördinaten, een vlak op het beeldscherm ingevuld. De coördinaten van het punt dienen binnen dit vlak te liggen. Indien de omgrenzing van het vlak ergens niet geheel gesloten is, wordt ook buiten het gewenste vlak ingevuld.

FORWARD ‹aantal stappen›

Met deze instructie kunt u de schildpad een bepaald aantal stappen (d.w.z. beeldschermpunten) naar voren bewegen.

HIDETURTLE

Met HIDETURTLE wordt de schildpad, die de positie van de grafische cursor aangeeft onzichtbaar gemaakt, om b.v. een voltooide grafiek ook zonder schildpad te kunnen bekijken.

6.1 Introduktie

HOME

Deze instructie brengt de schildpad naar het midden van het grafische beeldscherm.

LEFT‹hoek in graden›

Met deze instructie wordt de schildpad over het aantal opgegeven graden naar links gedraaid. Na een daaropvolgend commando 'FORWARD' zal de schildpad zich in een andere richting bewegen. Een volledige draaiing komt met 360 graden overeen.

MOVETO‹x-coördinaat›,‹y-coördinaat›

Met MOVETO kan de schildpad naar een gewenste coördinatenpositie verschoven worden. Daarbij wordt *geen* lijn getekend. Het is dus de tegenhanger van DRAWTO.

PENDOWN

Door middel van dit commando kan de schildpad in de tekenmodus gezet worden. Dat betekent, dat bij alle instructies, zoals b.v. 'FORWARD' of 'BACK', een lijn wordt getekend. De denkbeeldige pen wordt daarmee op het schrijfvlak gezet (down).

PENUP

Deze instructie doet precies het tegengestelde van PENDOWN. Wordt daarna met een instructie als b.v. 'FORWARD' of 'BACK' de schildpad bewogen, dan wordt er geen lijn getekend, d.w.z. er wordt geen spoor achtergelaten.

PLOT‹x-coördinaat›,‹y-coördinaat›

Met 'PLOT' wordt op de aangegeven positie één punt op het grafische beeldscherm getekend.

PLOTTEXT‹x-coördinaat›,
‹y-coördinaat›, ‹tekst›

Met deze instructie kan men ook op het grafische beeldscherm gewone tekst afbeelden. De opgegeven coördinaten bepalen het punt, waar de tekst begint; de af te drukken tekst wordt eenvoudigweg achter dat punt door een reeks tekens weergegeven. De begincoördinaten worden door de computer zodanig afgerond dat de tekst wordt afgedrukt volgens de regels, die gelden voor het tekstmodusbeeldscherm. In de vierkleurenmodus kunt u deze instructie niet toepassen.

RIGHT‹hoek in graden›

Met dit commando kan de schildpad over een opgegeven aantal graden rechtsom gedraaid worden (vergelijk commando 'LEFT')

SETHEADING‹hoek in graden›

Met dit commando kan de schildpad in een bepaalde richting gedraaid worden, waarbij 0, resp. 360 naar 'boven' en 90 graden naar 'rechts' betekent.

SETXY‹x-coördinaat›,‹y-coördinaat›

Met deze instructie is het mogelijk de schildpad naar een door coördinaten vastgelegde positie te bewegen. In tegenstelling tot 'DRAWTO' en 'MOVETO' is het hier doorslaggevend of het systeem zich in de 'PENUP' of 'PENDOWN'-status bevindt.

6.1 Introduktie

SHOWTURTLE

Het gaat hierbij om het tegendeel van de instructie 'HIDETURTLE'. Als de schildpad met 'HIDETURTLE' onzichtbaar gemaakt is, kan deze met SHOWTURTLE opnieuw zichtbaar gemaakt worden.

TURTLESIZE‹grootte›

Met deze instructie kan de schildpad een bepaalde grootte gegeven worden. Deze grootte varieert van 0 tot 10, waarbij 10 overeenkomt met de grootste afmeting, die men met COMAL kan instellen. Gaat u dus met SETGRAPHIC 0 de grafische modus in en voert u dan b.v. TURTLESIZE5 in, dan wordt de schildpadgrootte tot de helft verkleind. Met TURTLESIZE10 krijgt deze zijn oorspronkelijke grootte terug.

Nu volgen drie commando's, waarmee u de kleuren in verschillende gebieden van het beeldscherm kan instellen.

BACKGROUND‹kleurcode›

Met BACKGROUND kan de achtergrondkleur, - dat is de kleur waarop de eigenlijke grafiek getekend wordt -, veranderd worden. Het codenummer van een bepaalde kleur kunt u elders in dit werk vinden.

BORDER‹kleurcode›

Met deze instructie kunt u het gebied buiten het vlak waarin getekend wordt, - de omlijsting dus -, een andere kleur geven.

PENCOLOR‹kleurcode›

Met deze instructie kunt u de kleur waarmee u wilt gaan tekenen veranderen. Als u dus via deze instructie een nieuwe kleur opgeeft en u tekent daarna een lijn, dan verschijnt deze in de nieuwe kleur op het beeldscherm.

8/7

Superbase

INHOUD

8/7.1 Inleiding

8/7.1

Inleiding

De meeste bezitters van een Commodore 64 of een Commodore 128 kennen het programma Superbase.
Een veel kleinere groep heeft zich ook bezig gehouden met het programma als gebruiker.
Een groot deel van deze gebruikers valt echter weer af omdat het opzetten van een bestand (bijv. een adressenbestand) meestal nog wel lukt maar om er daarna ook nog een mooie print-out van te krijgen is voor velen te lastig.
Met name bij dit onderdeel blijkt dat de gebruiksaanwijzing wel bijzonder uitgebreid is maar op het onderdeel programmeren duidelijk te kort schiet.
Wellicht wordt dit veroorzaakt door het gegeven dat programmeren in Superbase niet met het normale Commdore BASIC gaat maar met een geheel eigen serie BASIC-opdrachten.
En met name deze eigen BASIC opdrachten maken Superbase zo fijn om mee te werken.
Twee simpele voorbeelden die het uitprinten met Superbase veraangenamen: PLEN en TLEN.
Met de opdracht PLEN in het start- of lijstprogramma kan nu standaard de papierlengte opgegeven worden en met TLEN de hoeveelheid tekstregels die op 1 pagina moet staan. Superbase houdt nu perfekt de regelteller bij en transporteert ook keurig naar een volgende pagina.

In de volgende hoofdstukken zal stap voor stap worden verteld hoe men een ledenadministratie kan maken met de daarbij behorende lijsten.

Het zal overigens wel duidelijk zijn dat hier sprake is van een *algemene* ledenadministratie.
Een voetbalvereniging zal andere detailinformatie willen hebben dan bijvoorbeeld een biljartvereniging of een tafeltennisvereniging. Het hierna uitgewerkte voorbeeld is aan te passen.

Achtereenvolgens zal behandeld worden:
1) het gereed maken van een bestandsschijf,
2) het start-programma,
3) het definiëren van de database,
4) het definiëren van het bestand,
5) het maken van een programma voor:
   a) het maken van labels
   b) het maken van overzichten en lijsten
   c) het maken van selekties

Gereed maken bestandsschijf
Als Superbase is ingeladen verschijnt op het scherm het startbeeld met twee mogelijkheden:
a) druk op return voor het inlezen van een bestandsschijf
b) druk op F1 voor het maken van een bestandsschijf.

7.1 Inleiding

Als de database al gemaakt is en het Startprogramma is overgezet op de bestandsschijf zal Superbase na het indrukken van de returntoets het Startprogramma zoeken, inlezen en runnen.
Superbase zal nu vragen welke database u wilt gebruiken en daarna welke file gebruikt gaat worden. Dat dit makkelijker kan zal in het volgende hoofdstuk worden toegelicht.

Als nu gekozen wordt voor de functie F1 kost het vrij veel tijd om een nieuwe schijf te formateren en enkele hulpfiles van de programma schijf over te zetten naar de bestandsschijf.

Deze tijd en moeite wordt bespaard met de volgende oplossing:
a) doe een nieuwe schijf in de diskdrive
b) druk op RETURN: na enige tijd verschijnt boven in het scherm de mededeling 'File not found (press return to continue)'
c) nu 2 maal RETURN indrukken, MENU2 van Superbase verschijnt dan, kies daar voor de functie F6 MAINTAIN
d) kies nu voor de optie F8 OTHERS en tik de volgende instructie in: N:aaaaa,99 waarbij 'aaaaa' de naam van de diskette is en 99 het ID-nummer.
e) het programma vraagt nu 'Are you sure' en tik in 'Y'.

Superbase formateert nu keurig de bestandsschijf en het programma blijft gewoon in de computer zitten.
Denk er wel aan dat Superbase 64 de schijf maar aan 1 kant formateert en Superbase 128 de schijf aan 2 kanten formateert. Ga nu weer terug naar MENU 1 door middel van de RETURN toets.

Het start programma
Uitgaande van het gegeven dat Superbase ALTIJD het startprogramma inleest kunnen de database en de file ook direct ingelezen worden. Hiertoe moeten we het startprogramma wat aanpassen.

Hierna volgt een voorbeeld van een startprogramma dat EN de database 'leden' (zie regel 150) EN de file 'leden' (regel 160) EN het printprogramma 'lijsten' inleest (zie regel 170).

Tevens is hier van de gelegenheid gebruik gemaakt om de basistekst die Superbase presenteert aan te passen naar eigen inzicht. Zie hiervoor de regels 20 t/m 110.

Daarnaast is er een kleine 'beveiliging' ingebracht tegen het onbevoegd gebruik van de database. Als hier een fout password wordt ingegeven wordt meteen naar regel 260 gesprongen waar de QUIT opdracht staat.
Als gevolg hiervan zal de computer gereset worden en is ook het programma Superbase uit de machine.
Voor iemand die dit na 1 keer te drastisch vindt kan er natuurlijk een teller worden bijgemaakt zodat de quit opdracht bijv. pas na 3 keer in werking treedt.

Om te voorkomen dat iemand het programma gaat LISTen om achter het password te komen kan gebruik gemaakt worden van het Superbase commando 'PROTECT'.

Het startprogramma zou natuurlijk via de PROG functie (F5 in MENU2) geheel opnieuw kunnen worden ingetikt. Het is echter ook mogelijk om het Startprogramma dat op de programmaschijf staat te wijzigen.

7.1 Inleiding

Hiertoe dient vanuit menu 1 rechtstreeks het commando LOAD START te worden ingetikt. Via de PROG-functie (F5 in MENU 2) komt nu het startprogramma op het scherm.
Hierna kan het programma gewijzigd worden en kan er uiteraard een eigen password ingezet worden.
Nadat alle wijzigingen zijn aangebracht kan met behulp van de toetsen F1 en RUN/STOP weer terug naar MENU1 gesprongen worden vanwaaruit direct het commando SAVE START danwel PROTECT START gegeven kan worden.
In beide gevallen wordt het startprogramma weggeschreven op de bestandsdiskette. Bij SAVE START kan het programma eventueel weer gewijzigd worden.
Bij PROTECT START IS DAT NIET MEER MOGELIJK.

```
10 brkon:gosub 180
20 display @10,3" ┌──────────────────────┐"
30 display @10,4"│ WEKA SOFT AMSTERDAM │"
40 display @10,5" └──────────────────────┘"
50 display @10,8" ┌──────────────────────┐"
60 display @10,9"│     (c) juli 1987     │"
70 display @10,10" └──────────────────────┘"
80 display @3,14"________________________________________"
90 display @3,16"────────────────────────────────────────"
100 display @0
110 display @3,15@+"L E D E N A D M I N I S T R A T I E"
120 display @10,18"Uw passwoord a.u.b."
130 ask @29,18kz$:display chr$(147)
140 if kz$<>"weka"then 260
150 database "leden",8,0
160 file "leden"
170 load "lijsten"
180 lmarg 1:rmarg 80
190 plen 72:tlen 66
200 pdev 4:pdef 0
220 lfeed 0:cont 1
230 space 0:across
240 screen 0
250 return
260 quit
```

7.1 Inleiding

Definiëren van de database
Vanuit MENU 1 kan rechtstreeks de opdracht 'DATABASE' ingetikt worden. Superbase vraagt nu om de naam van de database. Hier kunt u iedere naam intikken die u als HOOFDNAAM aan de database wilt geven. Onthou de naam echter goed want u hebt hem later nog nodig. Probeer in ieder geval een logische naam te bedenken zoals 'leden' of de naam van de vereniging of gewoon 'adressen' enz.. De maximale lengte van de naam is 16 tekens.

Zodra de naam is ingetikt en op RETURN is gedrukt constateert Superbase dat deze database niet bestaat en komt de vraag of deze aangemaakt moet worden. Wanneer u nu 'Y' opgeeft maakt Superbase de database onder de naam die u hebt opgegeven.

Definiëren van het bestand
De database is nu bekend bij de computer, echter nu moet ook het bestand nog gedefinieerd worden. Dit wordt gedaan door eerst naar MENU 2 te gaan en daar te kiezen voor de optie F1: FILE.
Superbase vraagt nu om de file-naam. Handig is om hier gewoon de naam 'bestand' te gebruiken, maar iedere andere naam tot 12 posities is mogelijk.
Kies echter ook hier weer voor logische namen.
Bij een ledenadministratie waar bijv. aparte bestanden moeten worden aangemaakt voor de leden en oud-leden kunnen beter de namen 'leden' en 'oud-leden' gebruikt worden dan bijv. bestand1 en bestand2.
In het hier uitgewerkte voorbeeld is als bestandsnaam gekozen voor 'LEDEN'.

Na het intikken van deze naam constateert Superbase dat het bestand er nog niet is en of hij het moet creëren.
Tik hier nu 'Y' in en een blanko formatscherm verschijnt op de monitor.
Voor de opzet van de ledenadministratie is in het voorbeeld gewerkt met vier verschillende schermen die ieder voor zich een bepaalde soort informatie opleveren.

Het eerste scherm is het scherm met de algemene informatie

Het tweede scherm geeft de specifieke sport informatie

Het derde scherm geeft de contributie informatie

Het vierde scherm biedt de mogelijkheid voor extra informatie

Alhoewel in de gebruiksaanwijzing van Superbase uitstekend wordt uitgelegd hoe men een scherm kan creëren toch een paar opmerkingen ter toelichting.

De kaders op het scherm zijn gemaakt met de CBM-toets + de toetsen A ; S ; Z en X voor de hoeken en de SHIFT * voor de strepen.

Het key-veld
Het veld 'Naam' wordt hier gebruikt als 'KEY-veld'.
Voordeel hiervan is dat alle leden automatisch alfabetisch op de naam worden gesorteerd.
Nadeel hiervan kan zijn dat eerst de achternaam en daarachter de voorletters moet worden ingebracht.
Superbase sorteert namelijk vanaf de eerste positie.
A. Zwaan komt hier dan VOOR
W. Adriaansen.

7.1 Inleiding

Voor een juiste sortering op achternaam dus invoeren als:
Adriaansen W en Zwaan A.

Natuurlijk an dit in verband met een wellicht minder fraaie adressering opgelost worden door gebruik te maken van twee velden: één voor de achternaam en een aparte voor de voorletters en evt. tussenvoegsels zoals 'van der'.
Consequentie hiervan is dat met het uitprinten wel alles mooi op volgorde geprint kan worden maar het printprogramma bijzonder moeilijke oplossingen moet bevatten voor het feit dat bij ieder adres dezelfde velden gebruikt moeten worden om te printen ook al wordt een veld bij een adres niet gebruikt.

Voorbeeld:
Veld Naam is 25 posities:
Velde en Jansen
Veld Voorletters is 5 posities: PKM en K
Veld Tussenvoegsel is 10 posities: van der

Bij het uitprinten levert dit op:
PKM /van der /Velde (de streepjes niet meetellen)
en
K / /Jansen (de streepjes niet meetellen)

Dit komt omdat voor iedereen het veld met de tussenvoegsels afgedrukt wordt OOK AL STAAT ER NIETS IN, dit nog afgezien van de variabele lengte van de inhoud. Denk maar aan 'van der' en gewoon 'de'. In theorie zou dit allemaal op te lossen zijn door voor het printen eerst per record na te gaan of een bepaald veld wel of niet gevuld is en op basis daarvan met behulp van variabele printposities alles uit te printen.

Consequentie hiervan is dat het printprogramma aanzienlijk langzamer zal lopen en u het risiko loopt dat het totale programma groter wordt dan 4K.

De veld-lengte
In de praktijk blijkt dat een veldlengte van 25 posities bijna altijd voldoende is voor de Namen, Adressen en Woonplaatsen. Dit gaat zeker op als gebruik gemaakt worden van de officiële afkortingen van de PTT. Deze zijn te vinden in het postcodeboek.

Voor de postcode is een ruimte gereserveerd van 7 posities. Tussen de 4 cijfers en de 2 letters moet namelijk 1 spatie zitten. Bij het invoeren van de gegevens er overigens op letten dat de 2 letters van de postcode en de plaats in HOOFDLETTERS worden ingebracht. Dit is een van de voorwaarde van de PTT voor het verkrijgen van evt. kortingen bij het aanbieden van grotere hoeveelheden gesorteerd op postcode.
Bij de opmaak van de printlay-out er tevens op letten dat er 2 spaties moeten zitten tussen de postcode en de woonplaats.

Op alle schermen komt het veld 'Naam' terug. dit veld hoeft echter maar een keer ingevuld te worden (het key-veld op scherm 1). Superbase zet de inhoud van het keyveld nu automatisch over in de andere velden met de omschrijving 'Naam'.

Het voordeel hiervan zal duidelijk worden als bijv. de penningmeester de contributie gegevens wil inbrengen en dan gaat 'bladeren' binnen de database.
Ook op scherm 2 kan hij dan direct zien of hij wel het juiste lid op het scherm heeft staan.

7.1 Inleiding

Op de volgende bladzijden staat een afdruk van de vier schermen. Van de eigen database kan overigens heel eenvoudig ook zo'n afdruk gemaakt worden., haal het gewenste scherm op de monitor en druk op de toetsen 'CONTROL' en 'P'. Als de printer aanstaat verschijnt een keurige schermafdruk op papier.

Achter de schermafdrukken staat nog een overzicht van alle rubrieken die in het voorbeeld zijn gebruikt.
Van de eigen database kunt u een dergelijke lijst krijgen door de volgende acties.

a) vanuit MENU1 tik rechtstreeks in: PLEN72: TLEN 66: PRINT ACROSS en druk op RETURN
b) ga naar MENU2 met behulp van RETURN en kies optie F6
c) kies nu de optie F1 STATUS en Superbase print keurig een overzicht met alle velden, type veld, de lengte en de evt. berekeningen die moeten worden gemaakt.

Door de papier- en tekstlengte mee te geven in deze opdracht zal Superbase ook rekening houden met de bladsprong.

d) Ga na het uitprinten terug naar MENU1 en type in: DISPLAY met return.

VERGEET DIT NIET AANGEZIEN U HET RISIKO LOOPT DAT ALLE UITVOER NIET MEER NAAR HET SCHERM GAAT MAAR NAAR DE PRINTER.

```
mode : Entry                        :#   1    n

 L E D E N A D M I N I S T R A T I E

          |ALGEMENE INFORMATIE|

Lidnr     <         > Soortlid <         >

Naam      <                            >

Adres     <                            >

Postcode  <         >

Plaats    <                            >

Telefoon  <              >

Gebdat    <         > Geslacht < >

Lidvanaf  <         > Eindelid <         >

Clubblad  < >
```

```
mode : Entry                          :#   14   d

                   |SPORT INFORMATIE|

          | Naam <                           > |

Keuring <        >

 Speelt <                             >

Positie <                             >
```

7.1 Inleiding

```
mode : Entry                :#  18  n

        ┌─────────────────────┐
        │FINANCIELE GEGEVENS │
        └─────────────────────┘

   ┌───────────────────────────────────┐
   │ Naam <                          > │
   └───────────────────────────────────┘

 Contr87  <        >  Bet87 <        >

 Contr88  <        >  Bet88 <        >

 Contr89  <        >  Bet89 <        >

 Contr90  <        >  Bet90 <        >

 Contr91  <        >  Bet91 <        >
```

```
mode : Entry                :#  29  t

        ┌──────────────────┐
        │EXTRA INFORMATIE │
        └──────────────────┘

   ┌───────────────────────────────────┐
   │ Naam <                          > │
   └───────────────────────────────────┘

 Opmerk1 <                             >

 Opmerk2 <                             >

 Opmerk3 <                             >
```

File Definition : leden/leden

#	name	type	Format/ Calculation	
1	Lidnr	numeric	+#######	
2	Soortlid	text	Length	8
3	Naam	key	Length	25
4	Adres	text	Length	25
5	Postcode	text	Length	7
6	Plaats	text	Length	25
7	Telefoon	text	Length	12
8	Gebdat	date	Length	7
9	Geslacht	text	Length	1
10	Lidvanaf	date	Length	7
11	Eindelid	date	Length	7
12	Clubblad	text	Length	1
13	Naam	text	Length	25
14	Keuring	date	Length	7
15	Speelt	text	Length	25
16	Positie	text	Length	25
17	Naam	text	Length	25
18	Contr87	numeric	+####.##	
19	Bet87	date	Length	7
20	Contr88	numeric	+####.##	
21	Bet88	date	Length	7
22	Contr89	numeric	+####.##	
23	Bet89	date	Length	7
24	Contr90	numeric	+####.##	
25	Bet90	date	Length	7
26	Contr91	numeric	+####.##	
27	Bet91	date	Length	7
28	Naam	text	Length	25
29	Opmerk1	text	Length	27
30	Opmerk2	text	Length	27
31	Opmerk3	text	Length	27

7.1 Inleiding

Het maken van een programma
Het programma is er en alle gegevens kunnen nu worden ingevoerd. Prettig is natuurlijk als de gegevens nu ook nog een beetje gestructureerd op papier kunnen komen.
Op de volgende pagina's staat een compleet programma om verschillende lijsten uit te draaien.
Dit programma wordt nu stap voor stap doorlopen.

De opbouw van het programma is als volgt:
A) Het MENU gedeelte
regels 10 - 120
B) INVOEREN
regel 130
C) Aanmaak LABELS
regels 140 - 350
D) Aanmaak TELEFOONLIJST
regels 400 - 550
E) Aanmaak LEDENLIJST
regels 600 - 690
F) Aanmaak BETAALLIJST
regels 700 - 780
G) Aanmaak DIVERSEN
regels 800 - 880
H) EINDE programma
regels 900 - 960

Het menu gedeelte
Op regel 10 wordt gestart met de opdracht screen 0. Dit is in feite een standaard opdracht op grond waarvan het programma altijd naar het eerste scherm gaat van een lid. Dit in tegenstelling tot de opdracht in regel 700 waar het programma direct naar het derde scherm gaat.
Even opletten dus: 0 (nul) is het eerste scherm en 2 is het derde scherm.
De tweede opdracht op deze regel maakt het scherm schoon.

Op regel 20 wordt aangegeven dat dat op de lijsten moeten worden afgedrukt volgens de europese methode van dag, maand, jaar. De namen van de maanden worden overigens wel in het Engels afgedrukt. Door de toevoeging van ',n' wordt iedere datum alvorens te worden geprint gecontroleerd op bestaanbaarheid. De datum 31APR78 wordt dus niet geaccepteerd.

Op de regels 30 t/m 90 staat nu het menu zoals op het scherm moet worden weergegeven.

In de inleiding is al gesteld dat Superbase enkele opdrachten kent die logischer overkomen dan de overeenkomstige opdrachten in Commodore BASIC.

Bij Commodore BASIC kan een PRINT opdracht 2 dingen betekenen: printen op papier en printen op het scherm. Dit afhankelijk van welk print-kanaal er open staat.
Bij Superbase ligt dit makkelijker: printen is ook werkelijk printen op papier en anders is het display voor op het scherm.
De opdracht van regel 30:
display @ 5, 4+"HOOFDMENU" houdt in:

> zet op de vijfde positie horizontaal en op de vierde regel vertikaal in reverse de tekst HOOFDMENU.

Op regel 400 staat dus de opdracht:

> zet op de vijfde positie horizontaal en op de achtste regel vertikaal de tekst 2 labels.

De syntax van een display opdracht is dan ook: display @ x, y waarbij x is de positie horizontaal (max. 40 of 60) en y de vertikale regel (max. 22).

De betreffende display opdrachten hoe-

7.1 Inleiding

ven overigens niet in de normale volgorde te staan.
Zet de opdrachten op de regels 30 t/m 90 maar een in een andere volgorde bijv.:

30 display @ 5, 14"5 Openst. contributie"
35 display @ 5, 10"3 Telefoonlijst"
40 display @ 5, 8"2 Labels" enz.

Het volledige menu verschijnt op het scherm, alleen wordt er niet van boven naar beneden geprint maar gaat alles door elkaar.
Regel 90 eindigt met de opdrachten '@ 1,0@ 1,0': dit zijn twee spatie-regels op het scherm.
Regel 100 geeft weer een andere opdracht te zien t.o.v. de normale Commodore BASIC opdracht: de term 'ASK'.
Deze opdracht komt in de plaats van de BASIC opdracht INPUT.
De regels 110 en 120 spreken voor zich zelf.

Invoeren/muteren
Op regel 130 staat de opdracht MENU. Als gevolg hiervan gaat Superbase automatisch naar het Superbase MENU1. Van hieruit kan dan bijv. de ENTER functie worden gekozen voor invoeren of de SELECT functie voor muteren.

Aanmaak labels
Als vanuit het hoofdmenu gekozen wordt voor de optie labels gaat Superbase naar regel 200 en geeft hier een submenu voor aanmaak van labels.
Hiervoor is in deze versie bewust gekozen; als er namelijk meerdere bestanden op dezelfde diskette staan (bijv. de OUD-LEDEN) kan volstaan worden met een en dezelfde printroutine voor beide bestanden.

Regel 300 geeft weer een nieuw soort opdracht: REPORT Z$.
Hiermee wordt aangegeven dat er een uitdraai moet worden gemaakt van velden in het bestand met de naam Z$. Voor de waarde van Z$ zie de regels 260 en 270.
Voor het maken van etiketten kunnen het best de opdrachten PLEN en TLEN gebruikt worden.
Een simpele methode om het aantal printregels voor etiketten te bepalen is ze naast een computer uitdraai te leggen en het aantal regels te tellen. Tel er dan één spatie regel bij op en klaar is kees.
In het voorbeeld is dus uitgegaan van etiketten met acht printregels op het etiket plus een spatieregel.
Dan een belangrijke opdracht op regel 310: PRINT ACROSS.
Dat is de opdracht die aangeeft dat alle uitvoer nu naar de printer moet worden gestuurd. De toevoeging ACROSS geeft aan dat er naast elkaar zal worden geprint, dit in tegenstelling tot de opdracht PRINT DOWN.
De regels 320 t/m 340 bevatten nu de hele print routine voor etiketten.
Regel 320 begint met de opdracht OUTPUT ALL. Dit houdt in dat alle records uit het bestand op volgorde van keyveld worden afgedrukt.
Voor het positioneren van de print worden dezelfde commando's gebruikt als bij de DISPLAY opdracht.

Let op dat de veldnamen tussen rechte haken staan ([....]). Dit is voor Superbase een aanduiding dat hier sprake is van een veld uit een bepaalde database.
Twee kleine dingen extra: achter het positioneringsteken @ 5 staat het '&' teken.
Dit houdt in dat de Superbase alleen die posities print waar ook werkelijk iets staat.
Dit klinkt heel logisch maar denk er aan

7.1 Inleiding

dat Superbase hier geen tekst strings print maar inhoud van velden.
Als in een naamveld van maximaal 25 posities alleen maar An R staat wil Superbase toch alle 25 posities printen (dus ook 21 lege posities). Door de toevoeging '&' worden deze lege posities weggelaten en zal derhalve de print sneller verlopen.
De regel 320 en 330 eindigen beide met de opdracht PLUS.
Deze opdracht vertelt Superbase dat de regels 320 t/m 340 bij elkaar horen als één record. Deze drie regels geven de gevraagde inhoud per record op een etiket.
In Commodore BASIC is dit het gedeelte dat binnen een FOR/NEXT loop staat.
De Superbase versie hiervan is deze PLUS opdracht in combinatie met de opdracht van regel 350: EOF goto 10. Oftewel bij End Of File naar regel 10 en anders het volgende record.

Telefoonlijst
De regels 400 t/m 510 moeten gezien het voorgaande bekend voorkomen.
Op de regels 520 en 530 staat de opdracht voor het maken van een kopregel boven ieder blad. Dit d.m.v. de opdracht TITLE.
Als gevolg van de PLUS opdracht op regel 520 wordt de volgende regel automatisch door Superbase als tweede kopregel herkend.
Regel 540 begint met de opdracht DETAIL ALL.
Dit is een alternatief voor de opdracht OUTPUT ALL en geeft dan ook hetzelfde resultaat.
Op deze regel overigens ook weer wat nieuws: voor het veld ADRES staat nu als positionering @ 53&13.
Betekenis: vanaf positie 53 de eerste 13 tekens van de inhoud van het veld ADRES.

Ledenlijst
De regels 600 t/m 690 bevatten op zich geen nieuwe opdrachten. De inhoud van een ledenlijst is uiteraard naar eigen behoefte aan te passen.
Een klein attentiepuntje echter m.b.t. de tekstlengte.
Per lid worden vier regels afgedrukt plus een spatieregel tussen elk lid (zie regel 680).
Voor de kopregel zijn in totaal 3 regels nodig: één print regel plus twee spatie regels.
Uitgaande van 12" papier met 72 regels blijven er derhalve 69 regels over voor de leden. Bij 5 regels per lid derhalve maximaal 13 leden op een blad.
Nu even terugrekenen: 13 leden x 5 regels + 3 regels kop is totaal 68 printregels. Zie regel 620 TLEN68.

Aanmaak betaallijst
De in dit voorbeeld gedefinieerde betaallijst is bedoeld als hulpmiddel voor de penningmeester om na te gaan wie de contributie nog niet heeft betaald.
Om dit te kunnen bereiken moet er dus een selectie gemaakt worden van die records (leden) die nog niet hebben betaald.
Op regel 700 staan de opdrachten SCREEN 2 en FIND "BETAALLIJST".
De eerste opdracht zorgt er voor dat Superbase automatisch naar het derde gedefinieerde scherm gaat. In dit geval dus het scherm met de betaalgegevens.
De tweede opdracht heeft tot gevolg dat er van dit scherm een selectie scherm verschijnt waarop u per veld kunt aangeven wat het selectie criterium is.

Voorbeeld:
U wilt alle leden hebben die hun contributie over 1988 nog niet hebben betaald.

7.1 Inleiding

Ga nu met de cursor naar veld CONTR88 en zet daarin '›1'; ga vervolgens naar het veld BET88 en zet daar in '↑' (het pijltje boven de CONTOL-toets.
Druk nu op SHIFT en RETURN. Op de bovenste regel van het scherm vraagt Superbase om de selectie datum. Geef deze op en Superbase zal alle leden selecteren die aan de voorwaarden voldoen.

Toelichting:
De waarde '1' in het veld CONTR88 is om te voorkomen dat er ook leden worden geselekteerd waar nog geen contributie bedrag is ingevuld en dus ook nog niet betaald kunnen hebben.
In het veld BET88 moet een datum worden ingevuld MET een selectie criterium (‹ kleiner dan of › groter dan).
Het volledige criterium van bijv. '‹31mar88' past niet in het veld vandaar de oplossing via het pijltje.
Na de hiervoor beschreven handelingen zal Superbase een bestandje aanmaken met de key-velden van alle leden die nog niet hebben betaald. Dit bestandje wordt op de schijf weggeschreven onder de naam 'BETAALLIJST'.
Na het volledig afwerken van de opdracht FIND BETAALLIJST zal Superbase verder gaan met de volgende regels.
Hierbij komt Superbase dan bij regel 760 weer een nieuwe opdracht tegen: DETAIL FROM "BETAALLIJST".
Met deze opdracht wordt aangegeven dat alleen die leden moeten worden afgedrukt die met behulp van de selectie in het bestandje BETAALLIJST zijn weggeschreven.

Diverse lijsten
Bij het onderdeel betaallijst is al gesproken over de selectie mogelijkheden die Superbase heeft.
Dit programma onderdeel is qua opzet hetzelfde als een betaallijst aanmaken doch is meer bedoeld voor een variabele inhoud.
Om dit te kunnen bereiken moeten er twee dingen gebeuren: ten eerste moet er een uitgebreide selectie mogelijk zijn. Vandaar de opdracht SCREEN0 op regel 800. Het programma gaat nu naar het eerste scherm. Ten tweede zullen de printregels 840 t/m 870 aangepast moeten worden naar gelang de behoefte van deze ene specifieke lijst.
Als blijkt dat bepaalde lijsten regelmatig noodzakelijk zijn is het natuurlijk logischer dat deze lijsten dan standaard in het programma komen en vanuit het HOOFDMENU rechtstreeks worden uitgevoerd.

Einde bewerkingen
In de regels 900 t/m 960 is een korte routine opgenomen die het gehele programma Superbase uit de computer haalt.
Superbase kan namelijk niet met behulp van een RESET knop uit de computer worden gehaald. Blijven er derhalve nog twee mogelijkheden over: 1 door middel van de QUIT opdracht (regel 960) of het eenvoudig uitzetten van de computer.
Aan u de keus.

SAVEn en testen van het programma
Nadat het gehele programma is ingetikt moet het nog geSAVED worden op de schijf.
Om uit de PROG optie te komen intikken de functietoest F1 + Q.
Superbase komt nu terug in MENU1. Hier kunt u nu rechtstreeks intikken SAVE "LIJSTEN".
Om het programma te laten lopen kan vanuit MENU1 rechtstreeks de F7 toets (EXECUTE) ingedrukt worden.

7.1 Inleiding

Als het programma niet foutloos is of moet worden aangepast op grond van bijv. de eigen behoeftes kan het na de wijziging op de zelfde wijze worden geSAVEd.
Superbase doet indien noodzakelijk zelf de REPLACE functie.

Slotmerkingen
Dit was het hele programma dat nodig is om de verschillende lijsten, labels e.d. aan te maken.

Zoals al een paar maal is opgemerkt kan het programma aangepast worden aan de eigen wensen. Later komen we op e.e.a. nog terug.

```
10 screen 0:display chr$(147)
20 date "01jan85",n
30 display @5,4@+" HOOFDMENU "
35 display @5,6"1 Invoeren/muteren"
40 display @5,8"2 Labels"
50 display @5,10"3 Telefoonlijst"
60 display @5,12"4 Ledenlijst op A4"
70 display @5,14"5 Openst.contributie"
80 display @5,16"6 Diverse lijsten"
90 display @5,18"7 Einde bewerkingen"@1,0@1,0
100 ask &1@5,20"Geef het juiste nummer op. ";x
110 if x<1or x>7then 100
120 on xgoto 130,200,400,600,700,800,900
130 menu
200 display chr$(147)
210 display @5,4@+" LABELMENU "
220 display @5,6"1 Van de leden"
230 display @5,8"2 Van de oud-leden"
240 display @5,10"3 Terug naar hoofdmenu"
250 ask &1@5,12"Geef juiste nummer ";z
260 if z=1then z$="leden":goto 300
270 if z=2then z$="oud leden":goto 300
280 if z=3then 10
290 goto 250
300 report z$:plen 9:tlen 3
310 print across
320 output all @5&[naam]plus
330 @5&[adres]plus
340 @5[postcode]@15&[plaats]
350 eof goto 10
400 display chr$(147)
410 display @5,4@+" TELEFOONLIJST "
420 display @5,6"1 Van de leden"
430 display @5,8"2 Van de oud-leden"
440 display @5,10"3 Terug naar hoofdmenu"
450 ask &1@5,12"Geef juiste nummer ";z
460 if z=1then z$="leden":goto 500
470 if z=2then z$="oud leden":goto 500
480 if z=3then 10
```

7.1 Inleiding

```
490 goto 450
500 report z$:plen 72:tlen 66
510 print across
520 title @27"T E L E F O O N L I J S T"@1,0@1,0plus
530 "Naam"@27"Adres"@53"Plaats"@67"Tel.nr."@1,0
540 detail all [naam]@27[adres]@53&13[plaats]@67[telefoon]
550 eof goto 10
600 print across
610 report "leden"
620 plen 72:tlen 68
630 title @30"L E D E N L I J S T"@1,0@1,0
640 detail all @3"Lidnr."@14&6,0[lidnr]@50"Srt lid"@60[soortlid]plus
650 @3"Naam"@15&[naam]@50"Tel."@60&[telefoon]plus
660 @3"Adres"@15&[adres]@50"Gebdat"@60[gebdat]plus
670 @3"Plaats"@15&[postcode]@25&[plaats]@50"Lid vanaf"@60[lidvanaf]plus
680 @1,0
690 eof goto 10
700 screen 2:find"betaallijst"
710 print across
720 report "leden"
730 plen 72:tlen 66
740 title @26"LIJST OPENSTAANDE CONTRIBUTIE"@1,0plus
750 @6"LIDNR"@15"NAAM"@45"SPEELT"@62"BEDRAG"@1,0
760 detail from "betaallijst"@5&6,0[lidnr]@15&[naam]@47&2,0[speelt]plus
770 @55[contr87]@1,0
780 eol goto 10
800 screen 0:find"diversen"
810 print across
820 report "leden"
830 plen 72:tlen 66
840 title @6"LIJST DIVERSEN"@1,0@1,0plus
850 @6"LIDNR"@15"NAAM"@40"SPEELT"@58"SOORT LID"@1,0
860 detail from "diversen"@5&6,0[lidnr]@15&[naam]@42&2,0[speelt]plus
870 @58[soortlid]@1,0
880 eol goto 10
900 display chr$(147)
910 display @11,6"Weet u het zeker ?"
920 display @11,8"Het programma gaat"
930 display @11,10"uit de machine !!"
940 ask @11,12"Graag j of n ";kz$
950 if kz$<>"j"then 10
960 quit
```

8/8

PILOT

Inhoud

8/8.1 Inleiding
8/8.2 Starten van PILOT
8/8.3 Een regel in PILOT
8/8.4 Conditionals
8/8.5 Opcodes
8/8.6 GRAPHICS
8/8.7 Sprites
8/8.8 Record-keeping
8/8.9 Geluid
8/8.10 De routine SYSX
8/8.11 Operatoren en functies
8/8.12 Vergelijkingen, voorwaarden

8/8.1

Inleiding

In Nederland en ook daarbuiten is het gebruik van de computer nog lang niet in alle lagen van het onderwijs doorgedrongen, dit in tegenstelling met wat men een paar jaar geleden verwachtte. Als we de kranten mogen geloven zou dat komen doordat er nog niet voldoende software beschikbaar is en dat wat er wel is, heeft niet de kwaliteit om op grote schaal in het onderwijs gebruikt te worden. Het schijnt dat veel computers in het lager onderwijs ongebruikt in de kast staan behalve dan op die scholen waar een hobbyist aktief is. Velen staan nogal onwennig tegen het apparaat aan te kijken. Dat is echter niet zo verwonderlijk als we ons bedenken dat aan de gebruiksvriendelijkheid van de (betaalbare) computer nog heel wat valt te doen. Uitermate frustrerend is het als door een klein bedieningsfoutje het ding toch niet doet wat het zou moeten doen en dan altijd in een situatie waarbij de leerkracht ook nog wel wat anders aan zijn/haar hoofd heeft dan alleen het bedienen van een computer. Er zullen heel wat onderwijsgevenden zijn die, nadat ze geprobeerd hebben een programmaatje in elkaar te zetten (bijna altijd in BASIC) toch niet uit de wirwar van GOTO's kunnen komen en als gevolg daarvan er voor altijd maar van af zien zelf iets te produceren. Het is natuurlijk ook een groot misverstand te denken dat een echt goed onderwijsprogramma even in de vrije tijd kan worden gemaakt in een taal die daarvoor niet is gemaakt. In de ontwikkeling van talen, compilers e.d. zitten manjaren werk. Zouden wat ingewikkeldere toepassingen even op een vrije avond te maken zijn?

Ondanks dit wat minder optimistische begin stellen we u hier voor de taal PILOT (Programmed Inquiry Learning Or Testing). In een bekend populair wetenschappelijk tijdschrift stond laatst (juni 1987) dat de taal PILOT ontwikkeld zou zijn om de leraar een nog gemakkelijker te leren taal dan BASIC aan te bieden. Dat is natuurlijk onzin. Zou een willekeurige nederlandse leraar moeite hebben met het aanleren van 20, 40 of 60 woorden? De problemen bij het programmeren zitten zeg maar nooit in het aanleren van de taal maar komen daarna of dat nu BASIC, PASCAL of welke taal dan ook is. Wat wel het geval is: in een taal die ontwikkeld is voor b.v. het bijhouden van een administratie is niet zo makkelijk en prettig te tekenen!

PILOT is dus speciaal voor het onderwijs ontwikkeld en ontleent zijn kracht aan een makkelijk te programmeren interaktie (inspelen op vraag en antwoord) met de leerling. We gaan in dit hoofdstuk er van uit dat de lezer al eens heeft kennis gemaakt met b.v. BASIC of PASCAL. Van

8.1 Inleiding

de taal PILOT zijn een aantal versies voor diverse computers in de handel. Wij bespreken hier de versie voor de COMMODORE 64.

In dit hoofdstuk zullen we het steeds hebben over de leerling, afgekort 'll' en de leraar, degeen die het programma de ll. aanbiedt.

Voor we ook maar iets duidelijk maken over de taal zelf, laten we de lezer met PILOT kennis maken door onderstaand scherts programma(tje). We geven zowel het programma zelf als het beeldscherm wat een ll. op het scherm krijgt bij het doorlopen van het programma. De bedoeling is dat de drie listings aandachtig worden bestudeerd.

We hebben twee ll. n.l. Pienter Hendrikdirk, die er snel uit was, en Hendrik Dommerik, die er wat langer over deed, het programma, aangeboden.

```
        d:n$(20)
        d:k$(2)
        t:      Vandaag gaan we wat rekenen, niet te veel natuurlijk,
        t:      maar voor we beginnen willen we weten hoe je heet.
        th:      Tik je naam in :
        a:n$
        th:      En ook nog even je klas :
        a:k$
        t:      O.k. Ben je klaar, zet je schrap, we beginnen:
*begin  t:      Wat is 5 + 2
        a:
        m:111
        ty:      Dat antwoord is goed in het tweetallig
        ty:      stelsel. Nu nog in het antwoord in tientallig stelsel!
        jy:begin
        m:7
        ty1:      Dat is goed en dat voor de eerste keer! Goed zeg!
        ty1:      Eigenlijk was dat niet de bedoeling. Want nu kan
        ty1:      dit programma jou niets meer leren. Jammer.
        jy1:eind
        ty:      Dat is een goed antwoord, onthou je het!
        jy:eind
        tn5:      Je hebt nu al 5 keer een fout antwoord gegeven.
        tn5:      Kennelijk weet je het niet. Het goed antwoord
        tn5:      is 7!! gewoon 2 meer dan 5, weet je wel.
        jn5:eind
        tn:      dat is fout
        tn:      probeer het nog eens
        jn:begin
*eind   t:      Beste $n$  uit klas  $k$ .
        t:      Ben je moe, voor vandaag vinden we het meer dan genoeg!
        t:
        t:      We stoppen er mee. Doe je thuis de groeten.
        e:
```

8.1 Inleiding

```
        Vandaag gaan we wat rekenen, niet te veel natuurlijk,
        maar voor we beginnen willen we weten hoe je heet.
        Tik je naam in : Hendrik Dommerik
        En ook nog even je klas : 5
        O.k. Ben je klaar, zet je schrap, we beginnen:
        Wat is 5 + 2
5
        dat is fout
        probeer het nog eens
        Wat is 5 + 2
2
        dat is fout
        probeer het nog eens
        Wat is 5 + 2
111
        Dat antwoord is goed in het tweetallig
        stelsel. Nu nog in het antwoord in tientallig stelsel!
        Wat is 5 + 2
i
        dat is fout
        probeer het nog eens
        Wat is 5 + 2
10
        Je hebt nu al 5 keer een fout antwoord gegeven.
        Kennelijk weet je het niet. Het goed antwoord
        is 7!! gewoon 2 meer dan 5, weet je wel.
        Beste Hendrik Dommerik uit klas  5.
        Ben je moe, voor vandaag vinden we het meer dan genoeg!

        We stoppen er mee. Doe je thuis de groeten.


        Vandaag gaan we wat rekenen, niet te veel natuurlijk,
        maar voor we beginnen willen we weten hoe je heet.
        Tik je naam in : Pienter Hendrikdirk
        En ook nog even je klas : 1
        O.k. Ben je klaar, zet je schrap, we beginnen:
        Wat is 5 + 2
7
        Dat is goed en dat voor de eerste keer! Goed zeg!
        Eigenlijk was dat niet de bedoeling. Want nu kan
        dit programma jou niets meer leren. Jammer.
        Beste Pienter Hendrikdirk uit klas  1.
        Ben je moe, voor vandaag vinden we het meer dan genoeg!

        We stoppen er mee. Doe je thuis de groeten.
```

8.1 Inleiding

Toelichting:
Het zal duidelijk zijn dat de letters 'y' en 'n' na de letter 't' voor resp. yes(goed) en no(fout) staan. In de volgende voorbeelden zullen we steeds zowel het programma als het schermbeeld geven. De bedoeling is dat het programma steeds wordt vergeleken met de tekst op het scherm. Ook zullen we in de programma's steeds de ingetikte tekst door de ll. aan de kantlijn laten zetten, en de tekst van de computer (programma, leraar) 5 spaties laten inspringen. Op deze manier is steeds makkelijk na te gaan wat de ll. heeft ingetikt en de reaktie van het programma daarop.
Om niet van die lange listings te krijgen en ook om de leesbaarheid van de programma's en schermbeelden te vergroten hebben we bij de programma's ons niet gehouden aan de maximale regelbreedte van de COMMODORE 64. Wil men de programma's in dit hoofdstuk dus intikken, dan zal men steeds de regelbreedte moeten aanpassen.

Met het schertsprogramma hebben we aangegeven dat vraag en antwoord snel en overzichtelijk is te programmeren. We moeten wel bedenken dat dit programma voor een gewoon antwoord op een simpele vraag erg uitgebreid is, natuurlijk zal meestal worden volstaan met goed/fout en als gevolg daarvan wel of niet naar de volgende vraag. Echter, het is een vergissing te denken dat het programma zo even achter de machine is bedacht en ingetikt. Dat kunnen alleen de zeer ervarenen die ook goed met de logika van zo'n programma om kunnen gaan. Wat bij elke taal geldt, geldt ook voor PILOT. Een statement op de verkeerde plaats heeft soms desastreuze gevolgen. Dus moet eerst een stroomdiagram getekend worden. Pas als dit met zorg is gedaan kan het intikken van het programma beginnen. Vooral het goed doordenken van de werking van een programma is ontzettend belangrijk. Het tekenen van een stroomdiagram kost wel wat tijd maar dat verdienen we bijna altijd ruimschoots terug. Ook voorkomen we daarmee de ergenis als blijkt dat een simpel ingetikt stukje programma het uiteindelijk toch niet doet, we vertellen hiermee voor degene die al wat ervaring heeft, niets nieuws.

8/8.2

Starten van PILOT

PILOT wordt geleverd op een enkel schijfje met een in het engels gestelde handleiding (ruim 100 pag.) We kunnen zeker zeggen: het boekje is uitstekend en helemaal toegesneden op de Commodore versie.
De man achter PILOT is een zekere Larry Kheriaty en er bestaat een gebruikersgroep die te bereiken is op het volgende adres: C/O PILOT Users' Group
Computer Center, Room 334
Western Washington University
Bellingham, WA 98225

Hebben we een Commodore 64, dan moeten we dus ook een drive hebben (een bandje is vanwege capaciteit en toegangstijd in dit geval een volledig achterhaalde zaak). Hebben we b.v. een Commodore 128, dan schakelen we hem in de stand C64.

Op het schijfje staan de volgende file's:

PILOT, PILOTR, BOX, ALPHABET, LA, SPRITES, SOUND.

De programma's PILOT en PILOTR hebben we nodig om een programma in de taal PILOT te kunnen runnen. Het verschil tussen PILOT en PILOTR is dat met het laatste programma een PILOT-programma alleen kan worden gerund terwijl met het eerste ook een programma kan worden aangemaakt. In feite is het programma PILOT de editor voor de taal PILOT.

BOX, ALPHABET en LA zijn drie voorbeeldprogramma's. Alle drie zijn ze wat pover. In het programma BOX b.v. zien we niet meer dan een zeer simpele animatie met de vlakken van een kubus.
SPRITES is een sprite-editor en met SOUND kunnen we geluid genereren.

Starten van PILOT op de COMMODORE
We starten de machine en komen in BASIC. Om iets in PILOT te kunnen doen moeten we of PILOT of PILOTR laden. We willen een programma aanmaken dus laden we PILOT.
We tikken in:

LOAD 'PILOT',8
daarna RETURN.

Na de gebruikelijke SEARCHING for ..., (even wachten) LOADING (even wachten) en READY tikken we in: RUN.

We komen na enige tijd in de zg. COMMAND-mode. Boven aan het scherm staat wat informatie en daaronder staat: PILOT: We kunnen nu van deze mode naar drie andere door het intikken van de letters e, r en i. Er zijn dus vier mode's.

8.2 Starten van PILOT

COMMAND	laden, wegschrijven en printen van programma's
EDIT	voor aanmaken en editten van programma's
RUN	voor runnen van het geladen of aangemaakt programma
IMMIDIATE	in deze mode kunnen we meteen uitproberen wat een stukje programma doet. Vooral prettig als we de taal moeten leren.

Pas op! Maken we of editten we een programma dan kunnen we niet van de IMMEDIATE-mode gebruik maken. Doen we dat toch dan zijn we het programma waarmee we bezig waren kwijt!! Willen we van mode veranderen dan altijd eerst terug naar COMMAND door intikken van de RUN/STOP-toets. Daarna intikken van of e, r of i.
Dat we in de COMMAND-mode zijn kunnen we herkennen aan de tekst: PILOT. De machine staat te wachten op een instruktie.

Gaan we voordat we iets geladen hebben naar de EDIT-mode dan treffen we daarin een stukje programma van 4 regels aan. We kunnen deze vier regels zien als we in de EDIT-mode naar beneden bladeren. Indrukken van SHIFT en CRSR (vertikaal). Wat dit stukje programma doet begrijpen we pas als we de zg. EXECUTE-INDIRECT hebben uitgelegd.

Zijn we in de COMMAND-mode dan kunnen we een programma laden door intikken van alleen een 'l'. Op het scherm verschijnt dan achter: PILOT:, L: en we tikken de naam van het programma in. Door intikken van r, e of i wordt het programma gerund of kunnen we het editten of komen we in de IMMIDIATE-mode.

Voor de commando's van de EDIT- en COMMAND-mode zie de betreffende paragrafen.

8/8.3

Een regel in PILOT

We verwijzen in deze paragraaf naar regels uit het schertsprogramma.

Een regel kan bestaan uit:

1. wel of niet een label
2. een opcode
3. geen, een of meer modifiers
4. geen, een of meer conditionals
5. teller
6. een colon
7. het veld

Label
Een label is een woord bestaande uit maximaal 6 letters/cijfers/leestekens. Voorafgegaan door een '*'. Voorbeeld '*begin' en *eind'. Andere voorbeelden '*les1', '*vraag6' enz. De naam van een label moet beginnen met een letter en mag geen spaties bevatten. Vanuit andere plaatsen van een programma kan naar die labels gesprongen worden. Eventueel kan een regel bestaan uit alleen een label. (Zie verder bij opcode J; JUMP.)

Opcode
Een letter die staat voor een bepaalde bewerking op het veld van de regel. Er zijn ongeveer 20 verschillende opcode's. Enkele voorbeelden: 't', 'd', 'j', die staan voor resp. TEKST, DIMENSION en JUMP. Ze worden één voor één in paragraaf 5 besproken.

Modifiers
Na een opcode kunnen één of meer modifiers volgen, dit zijn letters, die aan de werking van een opcode iets veranderen, b.v. 'h', van 'hang', toegevoegd aan een 't', zorgt ervoor dat de cursor niet meteen naar een volgende regel op het scherm springt, maar blijft staan op de positie na de laatst op het scherm gebrachte letter (zie de regel waar de naam van de ll. wordt gevraagd). De modifiers zijn specifiek voor elke opcode, vandaar dat ze bij elke opcode apart besproken worden. Het aantal modifiers varieert van 0 b.v. bij de 'd'-opcode (DIMENSION) tot 3 bij b.v. de 't'-opcode TEKST.

Conditional
Conditionals zijn de letters 'y' en 'n', een getal of uitdrukking die meteen na de opcode staan.
B.v.: na een antwoord op een vraag wordt een 'vlag' bijgehouden of de vraag goed of fout was. Die conditie 'y'(goed) of 'n'(fout) kunnen we later gebruiken om te anticiperen op het antwoord dat op een vraag is gegeven (zie regel met: Dat antwoord is goed in het tweetallig...). Conditionals zijn erg belangrijk, vandaar dat we daar, voor de bespreking van de opcodes, uitvoerig op terug komen.

Teller
Een teller, dit is eigenlijk ook een condi-

tional, die telt van 1 t/m 9. De teller houdt bij hoe vaak de laatste vraag is beantwoord (zie regel waarin de ll. er op wordt gewezen dat er al 5 keer een fout antwoord is gegeven).

Colon
Een colon ':', dubbele punt, moet na een opcode (met z'n modifiers en conditionals), één keer voorkomen.

Veld
Tekst of uitdrukking na de colon noemen we het veld.

8/8.4

Conditionals

Er zijn 3 soorten conditionals:

a. een enkele letter
Als een antwoord is gegeven op een vraag, dit gebeurt met de opcode a: ACCEPT, wordt tijdens het matchen, d.i. testen (we komen hier op terug), een vlag gezet. Deze vlag heeft twee standen nl. fout (n) en goed (y). De stand van de vlag kan later in het programma gebruikt worden om op de juiste manier op het antwoord te anticiperen. In het schertsprogramma komt een paar keer deze conditional voor. Een conditional bepaalt of de regel wel of niet zal worden uitgevoerd. We kunnen dus heel makkelijk een antwoord op een vraag geven in het geval de vraag goed was door bij de tekst-opcode de 'y'-conditional te plaatsen, even makkelijk als de vraag fout is beantwoord door het plaatsen van de 'n'-conditional. Plaatsen we die conditional niet dan wordt die de tekst na de t: TEKST-opcode altijd op het scherm gezet,

b. een getal 1 t/m 9
Tijdens het matchen wordt een teller bijgehouden. Elke keer dat een bepaalde ACCEPT 'wordt gepasseerd', wordt deze teller verhoogd met één. Met deze teller kunnen we heel makkelijk anticiperen op het aantal keren dat een vraag fout is beantwoord. Door alleen het plaatsen van een getal (1-9) na de opcode zal de regel alleen worden uitgevoerd als het aantal keren dat de ACCEPT is gepasseerd overeenkomt met dat getal.

c. een uitdrukking (of vergelijking)
Na evolueren van de uitdrukking blijkt deze waar of niet waar te zijn. Is de uitdrukking waar, dan wordt de regel uitgevoerd, bij niet waar wordt de regel dus niet uitgevoerd. In deze uitdrukkingen kunnen ook één of meer logische symbolen worden gebruikt. Zie daarvoor ook de paragraaf over uitdrukkingen/vergelijkingen.

De conditionals kunnen volgen op elke opcode in elke willekeurige volgorde. Voor de overzichtelijkheid wordt meestal de volgende volgorde aangehouden: letter, dan getal en als laatste de uitdrukking. De uitdrukking staat tussen haakjes.
Een voorbeeld is:

```
tn5(x>12):dit is een voorbeeld
```

Er zijn in dit voorbeeld dus 3 conditionals. De tekst wordt alleen op het scherm gezet als aan de 3 conditie's is voldaan nl.
1. de laatste keer bleek tijdens het matchen de vraag fout te zijn beantwoord;
2. het was eveneens de 5e keer dat de ACCEPT is gepasseerd
3. ook was 'x' één of andere variabele,

8.4 Conditionals

groter dan 12!!!
(We zullen de conditionals heel vaak tegen komen, vrijwel in elke programmaatje zitten ze).

8/8.5

Opcodes

In dit hoofdstuk bespreken we één voor één alle opcodes met zijn modifiers. We geven de opcodes in volgorde van belangrijkheid, uiteraard is dit dan een erg subjectieve volgorde.

In educatieve programma's draait het in feite toch allemaal om de tekst (uitleg, vragen e.d.), het antwoord op een vraag en wat we verder in dit hoofdstuk het 'matchen' zullen noemen. Onder dat matchen verstaan we het nagaan in hoeverre het antwoord door de ll. in overeenstemming is met het antwoord dat wordt verwacht.

We beginnen achtereenvolgens met de 't'-opcode, TEKST, de 'a'-opcode, ACCEPT en de 'm'-opcode, MATCH.
We kunnen wel zeggen dat de kracht van de taal PILOT schuilt in die drie opcode's *met* het handig toepassen van de conditionals en modifiers.

't' TEKST
Teksten komen natuurlijk veel voor in interaktieve programma's. De in het veld van alleen de opcode 't:' staande tekst komt precies zo op het scherm, als dat tenminste op de regel past, zoniet, dan wordt naar de volgende regel gesprongen en wordt daar de rest van de tekst geschreven. Na de laatste tekst-regel springt de cursor naar de volgende regel.
Een 'h', hang modifier, zorgt ervoor dat de cursor niet naar de volgende regel springt maar a.h.w. blijft hangen na de tekst die op het scherm geschreven is. Zijn we aan de onderkant van het scherm, dan 'scrollt' de tekst op het scherm net zo lang naar boven tot de hele tekst van de regel op het scherm staat (zie vensters).

In de tekst kunnen we variabelen en constanten e.d. opnemen. Een gedimensioneerde string die een inhoud heeft kunnen we op het scherm zetten door het '$'-teken aan die string voor af te laten gaan, constanten (getallen) moeten vooraf gaan door een ' # '-teken. Een string wordt altijd in het programma gevolgd door een '$'-teken.

Een 's'-modifier, zorgt ervoor dat eerst het scherm (werken we in een venster, het venster) wordt schoon gemaakt. De tekst begint dan dus in de linker bovenhoek van het scherm (venster). Als we een wat langere tekst willen opschrijven hoeven we niet steeds de opcode 't' te herhalen, we kunnen volstaan met alleen een colon.

In onderstaand programma komt van elke situatie een voorbeeld voor.

8.5 Opcodes

```
d:naam$(10),getal(3)
c:naam$="Pieter";getal=99
tx:Deze regel begint dus
 : op een schoongemaakt scherm.
t:Die naam is $naam$  en het
:getal #getal  .
th:Nu springt
t:de cursor niet naar
:de volgende regel.
:    Een spatie tussen springt
:en de was wel handig geweest.
e:
```

```
Deze regel begint dus
 op een schoongemaakt scherm.
Die naam is Pieter en het
getal 99.
Nu springtde cursor niet naar
de volgende regel.
   Een spatie tussen springt
en de was wel handig geweest.
```

Opmerkingen:
'ts' maakt dus het scherm schoon, in andere verhalen van PILOT kan dat 'tx' zijn.
In de eerste 3 regels wordt de naam Pieter en het getal 99 vastgelegd (dus een gedimensioneerde string, die een inhoud heeft, wordt steeds voorafgegaan en gevolgd door het '$'-teken. Een constante wordt voorafgegaan door het ' # '-teken). Na n en # g, volgt op het scherm een spatie.

> Als je je het volgende niet realiseert of weet kan je met de TEKST-opcode voor raadsels komen te staan en raadsel oplossen kost nu eenmaal tijd!

Na de 't:' wordt de tekst ingetikt die op het scherm moet komen INCLUSIEF DE SPATIES. Staan er spaties aan het eind van de geschreven tekst dan komen daarna alsnog de spaties ook als er op die regel geen plaats meer is. Automatisch wordt dan naar een volgende regel gesprongen en die is dus blanco. Bij het samenstellen van schermbeelden komt het heel vaak voor dat we (vanwege de beperkte regellengte, zeker in vensters) een woord weghalen of vervangen door een korter woord. In de tekst van het programma zien we de spaties niet meer maar ze staan er wel! Dus met het weghalen van een woord ook de spaties weghalen!
(zie ook: Kleur)

'a' ACCEPT
Op een vraag komt een door de ll. ingetikt antwoord. Met de 'a'-opcode komt alles wat de ll. intikt in wat we noemen een systeemvariabele '%b' of antwoordbuffer. De inhoud van die buffer heeft een bepaalde grootte, nl. 80 tekens. Even in de gaten houden bij vragen van een stuk(je) tekst! Er is slechts één antwoordbuffer, dus elke keer dat een 'a'-opcode in het programma voorkomt wordt de antwoordbuffer leeg gemaakt en komt alles wat ingetikt wordt tot het indrukken van een return in die buffer terecht. Later kan gekeken worden in hoeverre dat antwoord goed is. We kunnen op verschillende manieren dat wat in de buffer komt, beïnvloeden. Om te beginnen met de modifier 's'.

Het toevoegen van een 's', singel, heeft tot

8.5 Opcodes

gevolg dat het programma, zodra een toets is ingedrukt, doorgaat, er hoeft dan dus geen return te komen. Belangrijk als het gaat om snel te reageren op een vraag of iets dergelijks. Het antwoord ondergaat geen enkele bewerking voor het in de antwoordbuffer terechtkomt. We kunnen door het vermelden van één of meer strings of numerieke variabelen het antwoord meteen behalve in de antwoordbuffer ook meteen in die string- of numerieke variabele zetten. Komen meer strings- of numerieke variabelen voor dan krijgt alleen de eerste een inhoud.
Wat dat betreft werkt PILOT niet zo prettig als b.v. de wat betere versies van b.v. BASIC. Daarbij kunnen met één enkele INPUT een aantal variabelen een verschillende inhoud krijgen.
Als we willen nagaan of een antwoord goed of fout is, is het veel makkelijker te testen als het antwoord òf uit alleen hoofdletters òf alleen kleine letters bestaat. Door de 'p'-opcode, PROBLEM, kunnen we er voor zorgen dat alles wat in de antwoordbuffer komt eerst in hoofd- of kleine letters komt. Onderstaand programma begint met zo'n statement: pr:u. De 'u' staat voor uppercase (hoofdletter). Zie verder MATCH en PROBLEM.

```
        pr:u
        d:naam$(20),leeft(1)
        th:       Hoe heet je? :
        a:$naam$
*leef   th:       Hoe oud ben je? :
        a:#leeft
        te:       We verwachten een getal!
        je:leef
        t:       Dus je heet $naam$  en bent #leeft  jaar.
        e:
```

```
Hoe heet je? :Pienter Hendrik
Hoe oud ben je? :11 jaar
Dus je heet PIENTER HENDRIK en bent 11 jaar.
```

In bovenstaand programma zien we o.a. dat PILOT niet onmiddellijk met een foutmelding komt als er iets niet klopt. Dat heeft zo zijn voor- en nadelen.
Er wordt gevraagd naar een getal, dat blijkt uit het veld van de ACCEPT. Er komt echter als antwoord een getal en tekst nl. het woord 'jaar'. Zou je zoiets proberen in FORTRAN dan jaag je daarmee het hele apparaat op stang. PILOT geeft geen krimp. Als even later gevraagd wordt dat getal op te schrijven komt met ' # 1' gewoon het getal op het scherm. Kijken we wat in de antwoordbuffer zit dan blijkt daar wel die '11 jaar' in te zitten wat door de PROBLEM-opcode (pr:u) '11 JAAR' is geworden.
Natuurlijk heeft bovenstaande eigen-

8.5 Opcodes

schap in een interaktief programma meer voor- dan nadelen omdat bijna alle vragen meer (goede) alternatieven hebben. We komen hier nog nader op terug.

We geven nog een voorbeeld waarbij de inhoud van de antwoordbuffer door de 'p'-opcode wordt beïnvloed (we hebben het dus nu even niet over de 'a'-opcode). Vaak is het zo dat een antwoord goed is als de naam van iets zonder daarbij te letten op het gebruik van hoofd/kleineletters, goed is.

Ook kan een vraag goed zijn beantwoord bij gebruik van meer of minder spaties. Het is erg moeilijk antwoorden te testen waarbij het aantal hoofdletters, kleine letters en spaties niet precies vast ligt. In onderstaand voorbeeld zien we: 'pr:us'. De 's' zorgt ervoor dat ook alle spaties uit het antwoord verdwijnen. Het zal dus duidelijk zijn dat antwoorden in b.v. alleen hoofdletters en zonder spaties veel makkelijker te testen zijn. Onderstaand programma geeft weer een paar voorbeelden.

```
p:us
d:p$(30)
d:a$(30)
d:i$(30)
t:      Hoe heet je buurvrouw?
a:$a$
c:p$=%b
t:      In de antwoordbuffer zit:
t:      $p$  en ook:
t:      $a$ !!!
p:z
t:      Hoe heet je buurvrouw?
a:$a$
c:p$=%b
t:      Nu zit buf er zoals ze het gewend
t:      is in n.l. $p$
t:
t:      Tik snel je naam in:
as:
c:i$=%b
t:      we hebben nu dus alleen je eerste
t:      letter in de antw. buffer
t:      n.l. $i$
w:300
e:
```

8.5 Opcodes

```
      Hoe heet je buurvrouw?
C.J.A.M. Naaste geboren Buur
      In de antwoordbuffer zit:
      C.J.A.M.NAASTEGEBORENBUUR en ook:
      C.J.A.M.NAASTEGEBORENBUUR!!!
      Hoe heet je buurvrouw?
C.J.A.M. Naaste geboren Buur
      Nu zit buf er zoals ze het gewend
      is in n.l. C.J.A.M. Naaste geboren Buur

      Tik snel je naam in:
c      we hebben nu dus alleen je eerste
      letter in de antw. buffer
      n.l. c
```

Opmerkingen:
De tweede regel zorgt ervoor dat alle spaties uit het antwoord verdwijnen en alle kleine letters worden omgezet in hoofdletters. Maar ook uit a$ zijn de spaties en kleine letters verdwenen!

Met "p:z" maken we het eerste problemstatement weer ongedaan.

In het laatste gedeelte krijgen we niet de gelegenheid om de naam 'computeraar' in te tikken! De machine blijkt die man trouwens niet te kennen! Tikken we 'computeraar' snel in dan gebeurt er van alles en nog wat en even later staat onder aan het scherm 'raar'. Dat is overigens niet zo raar als het lijkt want: bij de eerste letter gaat het programma verder en op het scherm verschijnt dan de tekst 'We hebben...', wat onmiddellijk weer verdwijnt omdat een van de volgende toetsindrukken aan het wachten een eind maakt. Programma is dus afgelopen en we zitten weer in de COMMAND-mode. Er gebeuren nu rare dingen op het scherm maar even later wordt een 'e' ingetikt, we gaan dus naar de EDIT-mode en daarin wordt het woordje 'raar' ingetikt.

Zie ook 'p'-opode PROBLEM.

'm' MATCH
Na een ACCEPT zit in de antwoordbuffer

8.5 Opcodes

'%b', het te testen antwoord. Testen gebeurt met de 'm'-opcode. De m staat voor: MATCH, zoiets als vergelijken. Tijdens het 'matchen' wordt een vlag gezet, goed (y) of fout (n). Deze vlag kan later gebruikt worden om te zien of het antwoord goed of fout was. Zolang geen nieuwe MATCH komt blijft de vlag op dezelfde stand.

Het matchen gaat als volgt:

We hebben twee letter/cijfer patronen nl. na de ACCEPT en na de MATCH. Er wordt nu van voor naar achteren in het letter/cijferpatroon van de ACCEPT gekeken of hetzelfde letter/cijferpatroon van de MATCH voorkomt.

We geven een voorbeeld:
patroon ACCEPT b.v.: ABCDEFGHIJKLM
patroon MATCH b.v.: FGH

1. ABC - FGH past niet
2. BCD - FGH past niet
3. CDE - FGH past niet enz. tot dat
4. FGH - FGH past wel en meteen gaat de MATCH-vlag op goed (y).

Is het letter/cijfer-patroon van de MATCH groter dan van de ACCEPT dan wordt dus nooit een passend patroon gevonden en blijft de vlag op fout (n).

In het volgende programma laten we de buurvrouw nog weer even optreden.

Toelichting:

Bij de eerste match wordt het ingetikte antwoord zonder meer vergeleken (inclusief spaties, punten, komma's e.d.) met het veld van de 'm'-opcode. In het eerste voorbeeld werd de naam van de buurvrouw precies goed gespeld. De MATCH-vlag staat dus op goed (y) en alle opcodes met 'y' worden dus uitgevoerd dus ook: 'jy:eind". Programma springt dus naar laatste label en is afgelopen. Is het antwoord fout dat wordt elke opcode met een conditional 'n' uitgevoerd (tweede schermbeeld).
Bij de tweede MATCH wordt het symbool '&' gebruikt, wat staat voor elk willekeurig patroon (met spaties). Op die plek zou dus 'geboren' mogen staan of '-' enz.
In het laatste voorbeeld wordt het symbool '*' gebruikt. Dit is het symbool voor een z.g. wild-card. Elk teken wat op de plek van het sterretje staat wordt geaccepteerd.

Met de volgende symbolen kunnen we het matchen beïnvloeden. (minder precies maken)

1. '*' Elk ingetikte karakter wordt als goed geaccepteerd. (wild card) Laatste match in het voorbeeld.
2. '&' Een willekeurig aantal ingetikte karakters wordt als goed geaccepteerd. (serie wild cards) Tweede match in het voorbeeld. Dus tussen 'Naaste' en 'Buur' mag van alles en nog wat komen.
3. '%' Matchen van een spatie of begin of eind van een antwoord. B.v.: '%Naaste-Buur' laat niet een voorletter toe (wel spaties).
4. '!' Matcht zowel wat voor als wat na het '!'-teken komt. B.v.: Naaste!Buur. Eerst wordt 'Naaste' vervolgens 'Buur' getest.

We moeten goed in de gaten houden dat bij elke match geldt dat gekeken wordt of het veld van de 'm'-opcode voorkomt in de antwoordbuffer. De tweede keer is in de antwoordbuffer zowel voor als na de

8.5 Opcodes

```
     Hoe heet je buurvrouw?
Ik dacht C.J.A.M. Naaste-Buur
     Die ken ik al van het vorige programma
     Een aardig mens, niet?
```

```
     Hoe heet je buurvrouw?
Ik dacht: mevrouw C.J.A.M. Naaste geboren Buur of zo iets
     Nee, die heb ik nog nooit ontmoet.
     Maar heb je het wel goed geschreven?
     Ik dacht van wel.
     Oh, Ze heet eigenlijk mevrouw:
     C.J.A.M. Naaste-Buur
     Ik dacht dat haar voorletters
     niet C.J.A.M. waren maar C.J.I.M.
     Dus hoe heet ze nu eigenlijk?
Ik heb me vergist, ze heet : C.J.Q.M.Naaste-Buur
     Nu weet ik het weer
     Een aardig mens, niet?
```

```
      t:      Hoe heet je buurvrouw?
      a:
      m:C.J.A.M. Naaste-Buur
      ty:      Die ken ik al van het vorige programma
      jy:*eind
      tn:      Nee, die heb ik nog nooit ontmoet.
      tn:      Maar heb je het wel goed geschreven?
      tn:      Ik dacht van wel.
      m:C.J.A.M. Naaste&Buur
      tn:      Nee hoor, onbekend
      ty:      Oh, Ze heet eigenlijk mevrouw:
      ty:      C.J.A.M. Naaste-Buur
      ty:      Ik dacht dat haar voorletters
      ty:      niet C.J.A.M. waren maar C.J.I.M.
      ty:      Dus hoe heet ze nu eigenlijk?
      a:
      m:C.J.*.M Naaste&Buur
      t:      Nu weet ik het weer
*eind t:      Een aardig mens, niet?
      w:300
      e:
```

8.5 Opcodes

naam van de buurvrouw het een en ander ingetikt. De vlag is dus ook nu op goed gezet. Dat op deze manier gematcht wordt is bijzonder prettig. De ll. kan een heel verhaal als antwoord intikken, waar het om gaat is of het veld van de 'm'-opcode in het antwoord voorkomt!!

Toevoegen van een 's' aan de 'm'-opcode maakt het mogelijk kleine spelfouten over het hoofd te zien. Het antwoord mag één karakter afwijken van het veld van de 'm'-opcode (het antwoord mag dus niet een karakter langer of korter worden).

Meestal worden aan het begin van een programma met de 'p'-opcode alle ingevoerde antwoorden òf in hoofdletter òf in kleine letter gezet. Dit vergemakkelijkt het matchen aanzienlijk. Zie verder 'p'-opcode.

Tot slot: toevoegen van een 'j' geeft automatisch een sprong naar de volgende match indien het antwoord fout is. Dit dus om achtereenvolgens een aantal keren een mogelijk antwoord te matchen.

We geven nog een voorbeeld om te laten zien hoe makkelijk een match in Pilot kan worden uitgevoerd.

```
       pr:u
       t:       Noem eens 5 landen in Noord Europa:
*begin a:
       m:N&N!Z&N!&IJ&D!R&D!U&R!D&N!S&E!F&D
       tn:       Nee dat land ligt niet in Noord Europa
       ty:       Dat is goed.
       jy5:*eind
       t:       Nog een?
       j:begin
*eind  t:       Zijn dat er 5??
       e:
```

8.5 Opcodes

```
        Noem eens 5 landen in Noord Europa:
Noorwegen
        Dat is goed.
        Nog een?
ZWEDEN
        Dat is goed.
        Nog een?
Denemarken
        Dat is goed.
        Nog een?
Rusland
        Dat is goed.
        Nog een?
USSR
        Dat is goed.
        Zijn dat er 5??
```

Zouden we dit matchen in b.v. één of andere versie van BASIC doen dan zijn we nog wel even een poosje bezig. De kracht van PILOT zit nl. in het makkelijk kunnen inspelen op antwoorden van de ll. Overigens kan het met bovenstaand voorbeeld helemaal mis gaan, en wel door meerdere oorzaken.
Bijv. 5 x 'denemarken' intoetsen wordt niet opgemerkt als een wat vreemde manier van antwoorden op de gestelde vraag en de regel: 'jy5:eind'. We bedoelen natuurlijk dat naar 'eind' wordt gesprongen als 5 keer een goed antwoord is gegeven, maar dat gebeurt hier natuurlijk niet. Er wordt gesprongen na het 5e antwoord als tevens dat laatste antwoord goed is!!!
Bovenstaande manier van matchen kunnen we beter reserveren voor het matchen van alternatieven b.v. Rusland, Sowjet Unie, U.S.S.R., USSR.
Ondanks de mogelijkheden die PILOT de programmeur biedt moet elke match goed doordacht worden.

We geven nog een voorbeeld:
Stel: iemand heeft drie neefjes, Jan, Cor en Han. We kunnen die namen natuurlijk een voor een laten intikken, we moeten dan wel tenminste twee keer vragen: 'noem er nog eens een'. Veel mooier is natuurlijk:

```
         ps:1
         t:       Hoe heten je neven?
         a:
         m:jan&cor&han!jan&han&cor!cor&jan&han
         jy:kenik
         m:cor&han&jan!han&jan&cor!han&cor&jan
*kenik   ty:      Ja, die ken ik allemaal.
         tn:      Nee, die ken ik niet allemaal.
         w:300
         e:
```

```
        Hoe heten je neven?
Ik dacht Han, jan en ook nog COR
      Ja, die ken ik allemaal.
```

8.5 Opcodes

We kunnen de 6 verschillende mogelijkheden van het intikken van de drie namen niet op één regel zetten.
COMMODORE-PILOT laat slechts 40 karakters toe, vandaar dat we het matchen in twee stappen doen.

We zouden het ook als volgt kunnen doen (eigenlijk veel mooier). Vooral bij een flink aantal neven, in dit geval 4.

```
      ps:1
      t:      Hoe heten je neven?
      a:
      m:jan
      my:cor
      my:han
      my:ben
      jn:fout
      ty:      Ja die ken ik allemaal.
      e:
*fout t:      Nooit geweten!
      e:
```

```
     Hoe heten je neven?
Jan, Henk, Cor en Ben
     Nooit geweten!
```

Op de meest simpele vragen kan meestal een scala van antwoorden komen die allemaal goed zijn. Die verschillende antwoorden opnemen in de 'match' is onnodig als we de ll. dwingen op een bepaalde manier op de vraag te antwoorden. Meestal doen we dat ook. De vraag: ben je klaar? kunnen we het beste aanvullen met ja of nee. We verwachten nu of: ja, of nee en niet: bijna, nog even, momentje nog enz.

Ook bij het matchen van een simpel getal kunnen we in de fout gaan:

```
ps:u
t:      Hoeveel potloden heb je?
a:
m:1
ty:      Dat is niet veel!
t:      Zo gaat het wel!
t:      Hoeveel potloden heb je?
a:
m:%1%
ty:      Dat is niet veel
tn:      Dat is in ieder geval meer dan een.
e:
```

8.5 Opcodes

```
     Hoeveel potloden heb je?
11
     Dat is niet veel!
     Zo gaat het wel!
     Hoeveel potloden heb je?
11
     Dat is in ieder geval meer dan een.
```

In het voorbeeld wordt zonder voorzorgsmaatregelen 11 dus als goed antwoord geaccepteerd. Door '%'-tekens voor en achter het antwoord te plaatsen wordt getest op ' 1 ' (we bedoelen: 'spatie,1,spatie'). Nu moet het goede antwoord beginnen met: ' 1' ('spatie,1') en eindigen op '1 ' ('1,spatie'). Nu is er geen ander goed antwoord mogelijk dan een enkel potlood, en dat is inderdaad niet veel.
We kunnen het ook zo zeggen: willen we iets exact matchen dan kan dat door plaatsen van het '%'-teken voor en na het woord in het matchveld.

Als we bij de ACCEPT expliciet naar een getal vragen kan dat door het plaatsen van het ' # '-teken voor een letter. De letter stelt dan de naam van de variabele voor. Wordt nu iets anders ingetikt dan een getal, gaat de goed/fout-vlag altijd op fout. Maar .... ook nu gaat het matchen weer verkeerd. Zie onderstaand voorbeeld.

```
        p:u
*begin  t:      Nogmaals, hoeveel potloden heb je?
        a:#n
        m:1
        t:      Maar #n  potloden?
        ty:     O.K.
        tn:      Ik vroeg toch naar potloden!
        jn:begin
        w:300
        e:
```

8.5 Opcodes

```
      Nogmaals, hoeveel potloden heb je?
11
      Maar 0 potloden?
      Ik vroeg toch naar potloden!
      Nogmaals, hoeveel potloden heb je?
o
      Maar 0 potloden?
      Ik vroeg toch naar potloden!
      Nogmaals, hoeveel potloden heb je?
11
      Maar 11 potloden?
     O.K.
```

Het antwoord zit na een accept dus in de antwoordbuffer '%b'. Mochten we het antwoord later nodig hebben dan kunnen we het nu opslaan in een of andere van te voren gereserveerde string. Zie 'c'-opcode (COMPUTE).

'p' of 'pr' PROBLEM
Bij ACCEPT hebben we reeds met de 'p'-opcode kennis gemaakt. Met de PROBLEM-opcode kunnen we op de volgende manieren het antwoord in de antwoordbuffer beïnvloeden; in feite ondergaat het antwoord voor het in de buffer geplaatst wordt een bewerking.

u, alle bij de accept ingetikte letters worden hoofdletters (uppercase)
l, alle bij de accept ingetikte letters worden kleineletters (lowercase)
s, alle spaties worden verwijderd
e, maakt ESCAPE-functies aktief
z, (alleen voorkomend) maakt alle vorige opties van de laatste PROBLEM-opcode ongedaan.

Hebben we een of meer opties aktief gemaakt dan worden bij een volgende PROBLEM-opcode de bestaande opties ongedaan gemaakt en de bij de nieuwe PROBLEM-opcode vermelde opties aktief. Dus als eerst b.v. komt: "pr:ls' en later stuit het programma op: 'pr:u', dan worden de spaties dus niet meer genegeerd. Bij MATCH en ACCEPT hebben we al een paar maal een 'pr'-opcode gebruikt.

Voor de optie 'e' (ESCAPE) verwijzen we naar: 'p'-opcode (JUMP).

8.5 Opcodes

'd' DIMENSION
Aan het begin van een programma moet met de DIMENSION–opcode geheugenruimte worden gereserveerd voor de z.g. strings die later in het programma gebruikt worden. Een string bestaat uit een aantal letters/cijfers/codes; b.v. de naam van een gebruiker, een zinnetje, een ingetikt antwoord enz. Het aantal geheugenplaatsen, dus ook de lengte van een string, kan maximaal 255 zijn. In de loop van het programma kan de inhoud van een string worden veranderd. Zie verder bij de paragraaf: Stringmanipulatie.

In de meeste programmeertalen kunnen voor de naam van strings een aantal letters (b.v. 6 of 8 of nog meer) worden gebruikt. Hebben we strings nodig voor b.v. naam, adres en woonplaats dan zouden we dat als volgt kunnen doen:

d:naam$(20); adres$(20); plaats$(20)

Dus de naam van een string wordt gevolgd door het '$'–teken en daarachter komt het aantal te reserveren geheugenplaatsen. Het aantal staat tussen haakjes. De COMMODORE–versie van PILOT staat dit (helaas) op deze manier niet toe. Om te beginnen kan per regel, dus per 'd' – opcode maar één string worden gedimensioneerd en ook als naam slechts één letterteken worden gebruikt. Dus we krijgen dan:

```
d:n$(20)
d:a$(20)
d:p$(20)
```

Vooral het gebruik van maar één enkele letter voor de namen van strings maakt een programma veel minder prettig leesbaar en ook wordt daardoor het aantal te definiëren strings erg klein.

Zoals bijna alle programmeertalen wèl, kent de COMMODORE–versie van PILOT geen numerieke array's. Dit is een flink nadeel want we kunnen daardoor niet een aantal getallen op een gemakkelijke en snelle manier met b.v. een soort DO–LOOP opslaan. Elke variabele moeten we dus afzonderlijk een naam geven. Kent PILOT geen numerieke array's, dan kunnen we de z.g. meerdimensionale array's wel helemaal vergeten. Het rekenen met matrices gaat in talen zoals BASIC en FORTRAN redelijk eenvoudig. In PILOT is dit vrijwel niet te doen.

We kunnen wel stellen dat de COMMODORE–versie van PILOT ons niet uitnodigt ingewikkelde berekeningen uit te voeren. Om te beginnen hebben we daar het gereedschap niet voor (in de vorm van de juiste opcode's en meerdimensionale array's) en ook omdat het rekenwerk nogal traag verloopt. Dit is allemaal geen bezwaar omdat in bijna geen enkel educatief programma uitgebreid rekenwerk zit.

'c' COMPUTE
Het gewone rekenwerk
Optellen, aftrekken, vermenigvuldigen en delen gaat in PILOT nagenoeg op dezelfde manier als in de meeste andere talen (BASIC b.v.).
Maar pas op: De COMMODORE–versie van PILOT kent geen gebroken getallen, althans kent geen gebroken getallen als resultaat van een berekening.

Rechts van het '='–teken staat een uitdrukking die wordt uitgerekend. Het resultaat wordt toegekend aan de variabele links van het '='–teken. (Het '='–teken kunnen we het best lezen als: stop in de variabele die links staat het resultaat van de berekening die rechts staat).

8.5 Opcodes

Dus, in vergelijk met BASIC, niets nieuws.

Wèl nieuw is:

Links van het '='−teken kan een stringvariabele staan terwijl rechts een numerieke variabele staat of omgekeerd. Bij de meeste versies van BASIC geeft dat een foutmelding. PILOT geeft geen foutmelding maar voert de 'berekening' uit alsof er rechts van het '='−teken dezelfde type variabele staat als links. Er kunnen dus 'vreemde' uitkomsten komen uit zo op het oog eenvoudige sommetjes. We zijn dit al eens eerder tegengekomen. PILOT herstelt dit soort foutjes. We noemen dit: auto−conversion.

Met de 'c'−opcode kunnen we ook z.g. stringmanipulaties uitvoeren. Natuurlijk is PILOT wat zijn mogelijkheden betreft daarin wel goed voorzien. Zie volgende paragraaf: Stringmanipulatie.

Vooral als we al veel in andere talen hebben geprogrammeerd doen de uitkomsten van die simpele sommetjes wat vreemd aan.

-- voorbeeld 1 Niets nieuws, gewone numerieke variabelen hoeven dus niet te worden gedimensioneerd.

-- voorbeeld 2 We zouden kunnen verwachten: 'een vreemd getal'. Door de optelling is de inhoud van de b$ '0' geworden!! (auto−conversion)

-- voorbeeld 3 We verwachten misschien 0,666666. Zelfs geen goede afronding!! (geen gebroken getallen als uitkomst)

-- voorbeeld 4 Niets nieuws

-- voorbeeld 5 Het grootste getal waarmee we kunnen werken is: 32767 en het kleinste is −32768. Komen we boven resp. beneden deze grenzen dan ontstaat z.g. 'wrap−around', hetgeen in dit geval wil zeggen: op het grootste getal volgt het kleinste, dan het op één na kleinste enz. De andere kant op: op het kleinste getal volgt het grootste, dan het één na grootste enz.
Vandaar dat we i.p.v. 32768 het getal −32768 krijgen. Zouden we hebben laten berekenen 32767 + 10 dan zouden we −32759 hebben gekregen. We zien ook dat deze versie van PILOT geen z.g. wiskundige notatie (die met exponent 10) kent. PILOT reserveert voor elk getal, numerieke variabele, 2 bytes.
Omdat elke byte bestaat uit 8 posities voor een '0' of '1', is een groter getal dan 32767 niet mogelijk. (Een positie wordt niet voor het vastleggen van het getal gebruikt).

-- voorbeeld 6 Aantal haakjes klopt maar antwoord is wel erg onnauwkeurig!

-- voorbeeld 7 Deze versie van PILOT kent geen goniometrische funkties zoals b.v. de sinusfunktie. We kunnen in een betrekkelijk makkelijke routine de waarde van een sinus met behulp van een machtreeks wel berekenen. We krijgen op het eerste gezicht toch wel een wat erg grof afgeronde sinustabel. Het 'ruwe' resultaat is echter niet het gevolg van afrondingen maar de berekening:
z=x/57.295 (omzetten van graden in radialen) gaat al helemaal mis. Het getal 57,295 kunnen we niet invoeren, laat staan dat er van de verdere berekening iets terecht komt!

De voorbeelden in nevenstaand programmaatje geven duidelijk aan dat

8.5 Opcodes

```
         t:We rekenen een paar sommetjes uit:
         :zie het programmaatje zelf!!
         d:b$(20)
         c:b$="een vreemd getal"
*vb1     c:a=3+5
         t:#a
*vb2     c:b$=b$ + b$
         t:$b$
*vb3     c:c=2/3
         t:#c
*vb4     c:d=32766  +  1
         t:#d
*vb5     c:e=32767  +  1
         t:#e
*vb6     c:f=10/(((1+1)+1)/1)
         t:#f
*vb7     t:een sinustabel:
         c:x=0
*begin   u:sin
         c:x=x+10
         j(x<100):begin
         w:300
*end     e:
r:============================================
*sin     r:routine voor sinus (maar 3 termen)
         c:z=x/57.295
         c:y=z-(z*z*z)/6+(z*z*z*z*z)/120
         t:sin  #x  = #y
         e:
```

```
We rekenen een paar sommetjes uit:
zie het programmaatje zelf!!
8
0
0
32767
-32768
3
een sinustabel
sin 0 = 0
sin 10 = 0
sin 20 = 0
sin 30 = 0
sin 40 = 0
sin 50 = 0
sin 60 = 1
sin 70 = 1
sin 80 = 1
sin 90 = 1

A>
```

8.5 Opcodes

de mogelijkheden op het gebied van rekenen op zijn zachtst gezegd nogal pover zijn. Bovendien is het steeds uitkijken. Wat we misschien al jaren gewend zijn in andere talen te doen gaat niet in deze versie van PILOT en dat geeft ook niet altijd een foutmelding!
We doen er verstandig aan een berekening eerst even uit te testen in de IMMEDIATE–mode. We weten dan meteen of een bepaalde regel wel of niet werkt.

Moet in een regel een numerieke variabele worden opgenomen, b.v.:

Dus Laurens gaat verder met vraag 2.

Dan kan dat door de naam van de variabele, voorafgegaan door het '# '–teken in het veld van de t-opcode op te nemen. Dus:

t:Dus n gaat verder met vraag # v.

Dit gaat dus analoog aan de manier waarop we stringvariabelen in een zin opnemen (Het '$'–teken vóór de string is optioneel. We vermelden steeds het '$'–teken omdat in andere versies van PILOT het '$'–teken dan wèl moet worden geplaatst).

Stringmanipulatie.

Het belangrijkste verschil tussen een numerieke variabele en stringvariabele is dat voor een numerieke variabele steeds 2 bytes worden gebruikt en voor een stringvariabele zoveel bytes als bij de dimension–opcode wordt opgegeven. De numerieke variabele kan dus alleen getallen bevatten. De stringvariabele kan zowel getallen als letters als andere symbolen bevatten. Een stringvariabele bestaat uit maximaal 255 karakters.

Met: '!!' kunnen we strings samenvoegen. We noemen het '!!'–teken de z.g. concatenation–operator.

Het toekennen van een gedeelte van een string aan een andere string gaat als volgt: achter de string waarvan we een deel willen toekennen aan de andere string zetten we twee getallen tussen haakjes, gescheiden door een komma. Dus b.v. A$(3,5). Het eerste getal, de lengte van de substring die aan de andere string wordt toegekend.

-- voorbeeld 1 dit is een gewone concatenation

-- voorbeeld 2 de eerste 6 letters van de rechter string worden gezet in de linker string. De oorspronkelijke inhoud van de linker string gaat geheel verloren!!

-- voorbeeld 3 dit werkt bij de COMMODORE –versie van PILOT niet, wel b.v. in de I.B.M.–versie.

-- voorbeeld 4 zo werkt het wel, echter er worden maar twee '–'–tekens in de string geplaatst en niet wat we zouden verwachten, n.l. 3.

-- voorbeeld 5 gewone toekenning van één enkel symbool aan een string

-- voorbeeld 6 zelfde resultaat als voorbeeld 5, dus, vermelden we het aantal karakters niet, dan wordt één karakter in de string geplaatst.

'r' REMARK
Opcode voor het opnemen van teksten (op– en aanmerkingen) voor de lezer van het programma. Na 'r:' komt de tekst.

8.5 Opcodes

```
r:stringmanipulatie
r:er zijn 4 strings n.l. n, a, h en r
      d:n$(15)
      d:a$(15)
      d:h$(40)
      d:r$(5)
      c:n$="Voornaam  "
      c:a$="Achternaam"
      c:r$="12x--"
*vb1  c:h$=n$!!a$
      t:vb1:      $h$
*vb2  c:h$=a$(1,6)
      t:vb2:      $h$
*vb3  c:h$(8,4)=n$
      t:vb3:      $h$
*vb4  c:h$(5,3)="---"
      t:vb4:      $h$
*vb5  c:h$=r$(4,1)
      t:vb5:      $h$
*vb6  c:h$=r$(4)
      t:vb6:      $h$
      w:300
      e:
```

```
vb1:      Voornaam Achternaam
vb2:      Achter
vb3:      Achter Voor
vb4:      Acht---Voor
vb5:      -
vb6:      -
```

Vergelijk met REM in BASIC.
Zodra het programma een 'r'—opcode tegenkomt wordt alles wat daarachter op die regel staat genegeerd. Het is dus mogelijk om op een regel waar al een opcode met zijn veld staat ook nog een 'r'—opcode met tekst te plaatsen. Vaak zal dat niet voorkomen omdat de regellengte beperkt is tot 40 posities.

Het dimensionvoorbeeld zou dan kunnen worden:

```
d:n$(20)    r:naam
d:a$(20)    r:adres
d:p$(20)    r:plaats
```

waardoor het programma veel prettiger te lezen is.

'u' USE
Opcode voor het aanroepen van een subroutine. Na de opcode met colon volgt een label. Het programma springt naar dat label en gaat daar verder tot het eind van die subroutine. Dat eind wordt aangegeven met de 'e:'—opcode. Vervolgens gaat het programma verder op de regel volgend op de 'u'—opcode.

De subroutines kunnen genest worden. We bedoelen hiermee dat vanuit een subroutine weer naar een volgende subroutine gesprongen kan worden. De diepte (het aantal keren dat vanuit een routine verwezen wordt naar een volgende) is beperkt tot 4.
Belangrijk is dat we in de gaten houden dat we niet zomaar van de ene subroutine

8.5 Opcodes

naar de andere kunnen springen. Als we het goed doen komen we altijd weer terug in de subroutine vanwaaruit we een andere routine hebben aangeroepen.

In onderstaand programma geven we een voorbeeld waarbij vanuit de 'eerste subroutine' gesprongen wordt naar 'tweede subroutine'. Is het programma aangekomen in de 'tweede subroutine' dan kunnen we alleen via de 'eerste subroutine' weer terug naar het hoofdprogramma.

```
      r:voorbeeld van nesten van subroutines
r:=============================================
      t:hoofdprogramma
      u:sub1
      t:weer in het hoofdprogramma
      u:sub2
      t:en weer terug in het hoofdprogramma
      w:300
      e:
r:=============================================
      r:subroutine "eerste"
*sub1 t:      dit is de eerste subroutine
      u:sub2
      t:      nog steeds de eerste subroutine
      e:
r:=============================================
      r:subroutine "tweede"
*sub2 t:           dit is de tweede subroutine
      e:
```

```
hoofdprogramma
     dit is de eerste subroutine
          dit is de tweede subroutine
     nog steeds de eerste subroutine
weer in het hoofdprogramma
          dit is de tweede subroutine
en weer terug in het hoofdprogramma
```

'j' JUMP
De 'j'−opcode veroorzaakt een sprong naar de na de colon vermelde label.
Deze opcode kunnen we vergelijken met de bekende GOTO in BASIC. Wees er spaarzaam mee want voor je er erg in hebt raak je het spoor in het programma kwijt.
In de eerste versies van PASCAL kwam de GOTO niet voor, juist vanwege de filosofie achter PASCAL. Elke GOTO in een programma geeft aan, vonden de ontwerpers van PASCAL, dat een programma niet goed is doordacht. In de nieuwere versies van PASCAL komt vanwege een steeds groter wordende flexibiliteit de GOTO wel voor.

8.5 Opcodes

Als we na de 'j'—opcode conditionals gebruiken hebben we in plaats van een gewone GOTO een uiterst plezierig te gebruiken opcode in handen waarmee we voorwaardelijke sprongen kunnen maken. Zeker als nog gebruikt wordt gemaakt van de logische operatoren: & en ! blijkt dat we met de 'j'—opcode een uiterst krachtig verwijs/voorwaarden hulpmiddel in handen hebben. We komen daarop terug in het hoofdstuk: Operatoren.

In het volgende programma geven we nog een simpel voorbeeld (de in het programma voorkomende verwijzing is op een aantal manieren te maken).

We willen zeker niet beweren dat we met dit programma een elegant raamwerk laten zien voor een programma dat bestaat uit twee series vragen (de verwijzing naar de volgende vraag gaat nogal stuntelig met steeds maar die regel: c(v='x'):v=v+1. Er is een opcode die daar een uiterst elegante oplossing voor heeft. Zie de 'x'—opcode, EXECUTIVE INDIRECT).

Ook zien we in het programma een aardig voorbeeld van de manier waardoor het toch allemaal minder mooi wordt dan we zouden willen.
De leerling heet LAURENS. Even later wordt Laurens, die volgens eigen 'intikken' toch al 12 jaar is, aangesproken met

'O.K. Laurensje gaat verder met vraag ..'.

Zou Laurens de machine niet een wat houtaine manier van aanspreken kunnen verwijten? Eén enkele spatie met een komma achter de naamstring voorkomt dat Laurens zo onheus wordt aangesproken.

Er is nog een methode waarmee we snel naar het vervolg van het programma kunnen springen, vooral als het er om gaat snel te kunnen anticiperen op vragen. Het gaat het : '@ '—teken.
De volgende mogelijkheden bestaan:

j:@ a	veroorzaakt een sprong naar de laatste ACCEPT
j:@ p	veroorzaakt een sprong naar de volgende PROBLEM
j:@ m	veroorzaakt een sprong naar de volgende MATCH

In onderstaand programma geven we een voorbeeld.

Het systeem achter deze manier van matchen is het volgende :
eerst wordt het goede antwoord getest, is inderdaad een goed antwoord gegeven dan gaan we meteen met 'jy: p' naar de volgende vraag. Vervolgens worden voor de hand liggende foute antwoorden in groepjes getest. Met 'ty:....' wordt steeds geanticipeerd op een specifieke fout. Is die fout inderdaad gemaakt dan weer met 'jy: a' terug naar de ACCEPT.

In dit geval wordt nog op de schrijfwijze van het antwoord ingegaan. Bij de eerste MATCH wordt alleen een antwoord geaccepteerd als daarvan ook de spelling goed is. Bij de laatste MATCH worden ook goede antwoorden die verkeerd zijn gespeld getest.
Is inderdaad een spelfout gemaakt dan gaan we met 'jy: a' weer terug naar de ACCEPT.

Het voordeel van deze methode in plaats van het gebruik van labels is, dat om te beginnen de machine niet uit alle labels naar het juiste label hoeft te zoeken. Dat geeft dus een stukje tijdwinst, maar wat belangrijker is: als we ons eraan wennen de vragen met een PROBLEM te beginnen,

8.5 Opcodes

```
        d:n$(20)
        d:j$(20)
        t:      Hoe heet je?
        a:$n$
        t:      Hoe oud ben je?
        a:#l
        c(l<10):j$="jonger dan 10 jaar."
        c(l>=10):j$="vanaf 10 jaar."
        t:      Met de hoeveelste vraag wil je verder?
        t:      Er zijn in totaal 5 vragen.
        t:      Dus tik in: 1,2,3,4 of 5.
        a:#v
        t:      O.K. $n$ je gaat verder met vraag #v  die
        t:      bestemd is voor ll.  $j$
        j(l<10):*jong
        j(l>=10):*oud
*jong   r:Hier beginnen de vragen voor ll. jonger dan 10 jaar.
        t:      Een simpel vraagje.
*jovr1  t(v=1):     De eerste vraag.
        c(v=1):v=v+1
*jovr2  t(v=2):     De tweede vraag.
        c(v=2):v=v+1
*jovr3  t(v=3):     De derde vraag.
        c(v=3):v=v+1
*jovr4  t(v=4):     De vierde vraag.
        c(v=4):v=v+1
*jovr5  t(v=5):     De vijfde vraag.
        c(v=5):v=v+1
        r:etc.
        w:300
        e:
*oud    r:Hier beginnen de vragen voor ll. vanaf 10 jaar.
        t:      Een moeilijke vraag.
        r:etc.
*ouvr1  t(v=1):     De eerste vraag.
        c(v=1):v=v+1
*ouvr2  t(v=2):     De tweede vraag.
        c(v=2):v=v+1
*ouvr3  t(v=3):     De derde vraag.
        c(v=3):v=v+1
*ouvr4  t(v=4):     De vierde vraag.
        c(v=4):v=v+1
*ouvr5  t(v=5):     De vijfde vraag
        c(v=5):v=v+1
        r:etc.
        w:300
        e:
```

8.5 Opcodes

```
        Hoe heet je?
Laurens
        Hoe oud ben je?
12
        Met de hoeveelste vraag wil je verder?
        Er zijn in totaal 5 vragen.
        Dus tik in: 1,2,3,4 of 5.
2
        O.K. Laurensje gaat verder met vraag 2 die
        bestemd is voor ll.  vanaf 10 jaar.
        Een moeilijke vraag.
        De tweede vraag.
        De derde vraag.
        De vierde vraag.
        De vijfde vraag
```

kunnen de vragen heel gemakkelijk onderling verwisseld, tussengevoegd èn weggehaald worden. Het geheel wordt uiterst flexibel.
Met dit systeem kunnen we, als het alléén gaat om series vragen, vanachter het toetsenbord een heel programma schrijven.

Als we deze methode vergelijken met het schertsprogramma waar we mee begonnen zijn dan zien we dat PILOT nog wel uitermate geschikt is om te gebruiken in onderwijsprogramma's.

Er zijn nog drie hulpmiddelen die zeer van pas kunnen komen in onderwijsprogramma's n.l. de routine SYSX, de K-RECORD en de EXECUTE INDIRECT. De laatste twee hulpmiddelen zijn gewone opcode's.

'x' EXECUTE
Al eerder is opgemerkt dat we kunnen beschikken over een zeer krachtige opcode waarmee we naar door de ll. opgegeven delen van het programma kunnen springen. Maar niet alleen daarvoor kan die opcode gebruikt worden, ook is het mogelijk met die opcode nieuwe programmaregels aan te brengen. Dit laatste kan de programmeur gebruiken tijdens het testen of uitvoeren van een gedeelte van het programma. Waar we het over hebben is de opcode 'x', die we EXECUTE zullen noemen.

Eigenlijk is het niets anders dan een opcode die aangeeft:

> doe wat in het veld van mijn eigen opcode staat.

Dat veld kunnen we met een ACCEPT tijdens de verwerking van het programma invoeren. Simpel niet? Dus, als we het volgende doen:

```
t:Voer dat in wat je wil laten uitvoeren
a:u$
x:u$
```

wordt dat wat in u$ staat uitgevoerd.

8.5 Opcodes

```
r:een paar JUMP voorbeelden met het "@"-teken
        pr:u
        ts:         In welke stad bevindt zich het
        :         Internationaal Gerechtshof?
        a:
        m:DEN HAAG!'S GRAVENHAGE
        ty:         Inderdaad, dat is goed!
        jy:@p
        m:PARIJS!BRU&EL!B&N!AMS&DAM
        ty:         Dat is een hoofdstad in Europa maar,
        ty:         daar is het Intern. Gerechtshof
        ty:         niet gevestigd.
        jy:@a
        m:NE&RK!LOS&LES!WASH&N
        ty:         De stad die we bedoelen ligt in
        ty:         Europa!
        jy:@a
        m:STRAATSBURG
        ty:         Daar is het Europese Parlement
        ty:         gevestigd, maar niet het Int.
        ty:         Gerechtshof!
        jy:@a
        m:DE&AG!&VENHA&
        ty:         Zo schrijf je die naam toch niet!
        jy:@a
        t:
        :         Je weet het kennelijk niet. Dat
        :         gerechtshof ligt in Den Haag.
        pr:
        t:
        :         Het Internationaal Gerechtshof
        :         is een Gerechtshof van de ...
        a:
        m:UNO!UN&TIONS!V&N&
        ty:         Goed
        jy:@p
        m:EUR&SCHAP!EEG
        ty:         Foei
        jy:@a
        pr:
*end    t:        enz.
        e:
```

8.5 Opcodes

```
          In welke stad bevindt zich het
          Internationaal Gerechtshof?
New York
          De stad die we bedoelen ligt in
          Europa!
Straatsburg
          Daar is het Europese Parlement
          gevestigd, maar niet het Int.
          Gerechtshof!
Brussel
          Dat is een hoofdstad in Europa maar,
          daar is het Intern. Gerechtshof
          niet gevestigd.
Londen

          Je weet het kennelijk niet. Dat
          gerechtshof ligt in Den Haag.

          Het Internationaal Gerechtshof
          is een Gerechtshof van de ...
Verenigde Naties
          Goed
         enz.


          In welke stad bevindt zich het
          Internationaal Gerechtshof?
Brussel
          Dat is een hoofdstad in Europa maar,
          daar is het Intern. Gerechtshof
          niet gevestigd.
SGravenhage
          Zo schrijf je die naam toch niet!
's Gravenhage
          Inderdaad, dat is goed!

          Het Internationaal Gerechtshof
          is een Gerechtshof van de ...
Europese Gemeenschap
          Foei
Verenigde Naties
          Goed
         enz.
```

U$ zou bijvoorbeeld kunnen zijn: 'j:begin' of 'j:vra1' maar net zo goed als dat zo uitkomt 'd:a$(20)' of 'm:natriumsulfaat' of ja,..... bedenk zelf maar wat!

Belangrijk is dat we ons realiseren dat deze opcode, ons enorme mogelijkheden biedt, juist in educatieve programma's. We geven van dit laatste nog een voorbeeld.

Eerst herschrijven we het programma uit paragraaf 5.i, waarin de ll. gevraag wordt met welke vraag hij/zij verder wil.

8.5 Opcodes

```
        d:n$(20)
        d:j$(20)
        d:q$(7)
        d:v$(1)
        t:      Hoe heet je?
        a:$n$
        t:      Hoe oud ben je?
        a:#l
        c(l<10):j$="jonger dan 10 jaar."
        c(l<10):q$="j:jovr"
        c(l>=10):j$="vanaf 10 jaar.'
        c(l>=10):q$="j:ouvr"
        t:      Met de hoeveelste vraag wil je verder?
        t:      Er zijn in totaal 5 vragen.
        t:      Dus tik in: 1,2,3,4 of 5.
        as:$v$
        c:q$= q$ !! v$
        t:
        t:      O.K. $n$  je gaat verder met vraag $v$  die
        t:      bestemd is voor ll.   $j$
        x:q$
*jong   r:Hier beginnen de vragen voor ll. jonger dan 10 jaar.
*jovr1  t:      De eerste vraag.
*jovr2  t:      De tweede vraag.
*jovr3  t:      De derde vraag.
*jovr4  t:      De vierde vraag.
*jovr5  t:      De vijfde vraag.
        r:etc.
        w:300
        e:
*oud    r:Hier beginnen de vragen voor ll. vanaf 10 jaar.
        r:etc.
*ouvr1  t:      De eerste vraag.
*ouvr2  t:      De tweede vraag.
*ouvr3  t:      De derde vraag.
*ouvr4  t:      De vierde vraag.
*ouvr5  t:      De vijfde vraag
        r:etc.
        w:300
        e:
```

8.5 Opcodes

```
      Hoe heet je?
pieter
      Hoe oud ben je?
13
      Met de hoeveelste vraag wil je verder?
      Er zijn in totaal 5 vragen.
      Dus tik in: 1,2,3,4 of 5.
3
      O.K. pieter je gaat verder met vraag 3 die
      bestemd is voor ll.  vanaf 10 jaar.
      De derde vraag.
      De vierde vraag.
      De vijfde vraag
```

We hebben dus niets anders gedaan dan het programma daar waar dat nodig is veranderd, zó dat we de EXECUTE kunnen toepassen.

Nu hebben we een veel fraaier raamwerk voor de twee series vragen, ook is die ongelukkige methode van verwijzen naar een bepaalde vraag verdwenen.
In het begin hebben we twee nieuwe strings moeten definiëren n.l. 'q$' en 'v$'. In 'q$' komt de regel (opcode met veld) die we willen laten uitvoeren, dus de JUMP naar de gevraagde label. Het veld wordt in twee stappen opgebouwd. Eerst de verwijzing naar de juiste serie (ouder/jonger dan 10 jaar) en daarna wordt in de regel van de COMPUTE met behulp van een CONCATENATIE het vraagnummer aan de gekozen serie toegevoegd. Omdat we met een CONCATENATIE alleen karakterstrings kunnen samenvoegen, hebben we van de numerieke string '@ v' een karakterstring 'v$' gemaakt.

In plaats van over de opcode EXECUTE te spreken wordt ook wel over de opcode EXECUTE–INDIRECT gesproken, dit vanwege de indirecte manier van het uitvoeren van een opcode.

n.b. In het veld van een 'x'–opcode mag niet nog eens een 'x'–opcode voorkomen.

In hoofdstuk 2, Starten van PILOT, hebben we gezegd dat we het stukje programma dat we in de IMMEDIATE–mode aantreffen pas kunnen verklaren als we de EXECUTE–INDIRECT hebben uitgelegd.

Welnu: starten we PILOT en gaan we meteen naar de IMMEDIATE–mode dan vinden we daarin het volgende stukje programma:

```
TS:IMMEDIATE MODE...
A:
X:%B
J:@ A
```

Dit stukje programma doet niets anders dan uitvoeren wat bij de ACCEPT wordt ingevoerd. Is dat gebeurd, dan weer terug naar de ACCEPT.

In het volgende programma wordt de ll. gevraagd steeds een bepaald zinnetje in een andere persoon te zetten.
Het zinnetje luidt (1e persoon, enkelvoud)

8.5 Opcodes

Ik ga naar school en heb mijn tas bij me.
~ ~~~~~~ ~

De woorden die in dat zinnetje veranderen worden getest, dat zijn er dus 5. Het persoonlijk voornaamwoord waarmee het zinnetje begint wordt steeds door het programma gekozen.

Programma's over taal vereisen toch wel een heel andere aanpak dan b.v. rekenprogramma's. In rekenprogramma's kan heel simpel een vraag herhaald worden door met een randomgenerator dezelfde vraag (het sommetje) met andere getallen te herhalen. Voor het rekenen op basisschool—niveau worden dat relatief eenvoudige programma's.
In taalprogramma's kan hetzelfde stramien niet zo simpel worden herhaald. De onregelmatigheden in zinsbouw, spelling van uitgangen van woorden (enk. meerv.), persoon, tijden e.d. zorgen ervoor dat ze vrij lastig zijn te programmeren, althans in vergelijk met rekenprogramma's.
Het matchen in taalprogramma's moet heel precies worden aangepakt. In het programma wordt daarvan een voorbeeld gegeven.

Na: 'r:Nu organiseren we de goede MATCH' worden de 6 verschillende series van 5 antwoorden allemaal in de 't$' gestopt. De strings i$ (ik), j$ (jij) enz beginnen allemaal met het persoonlijk voornaamwoord, dat wordt gekozen en dus niet hoeft te worden getest. Verder zijn alle goede antwoorden a.h.w. 'ingebed' in voldoende en de juiste '%'—'&'—tekens.

In de regel met de label '*nog' wordt een pers. voornaamwoord gekozen. De betreffende deelstring geschreven in de string 'r$' en het eerste woord wordt op het scherm gezet.
De ll. maakt nu de zin af. Steeds wordt een woord uit de deelstring gepakt en in de routine 'ROUTI' getest. In de routine wordt gebruik gemaakt van de EXECUTE. Die EXECUTE werkt hier zo prettig omdat een regel zoals: m:x nu eenmaal niet werkt. Zouden we die EXECUTE niet kunnen gebruiken dan zou in dit toch simpele voorbeeld al 30!! maal een MATCH voorkomen. Wel gemakkelijk te programmeren maar nu niet bepaald elegant.
Deze methode maakt het mogelijk ook andere zinnetjes in te voeren. Het enige waaraan moet worden voldaan is de struktuur van de strings i$, j$, h$ enz. Is een woord in de ene zin wat langer dan in de andere zin. B.v. ziekenhuis i.p.v. school, dan kunnen de deelstrings worden aangepast door voldoende '&'—tekens te plaatsen.

Zo'n programma kan ondanks de voorzorgen toch op alle mogelijke manieren mis gaan. Hier b.v. in een van de zinnen komt het woord 'jullie' twee keer voor. Een antwoord:

> Jullie gaan naar school en hullie hebben hun tas bij hun.

Geeft niet het juiste aantal fouten, wel amusant voor de leraar (als het niet te vaak gebeurt), maar denkbeeldig?

Het programma aanpassen op twee keer hetzelfde woord in één zin is niet zo moeilijk, maar als er nog een paar van dat soort probleempjes zijn wordt zo'n programma wel meteen erg lang.
Ook is de kans groot dat bij het invoeren van een nieuw zinnetje, dat zinnetje weer zijn specifieke onregelmatigheden heeft, waar dan ook weer op moet worden gean-

8.5 Opcodes

```
        pr:1
        d:i$(40)
        d:j$(40)
        d:h$(40)
        d:w$(40)
        d:u$(40)
        d:z$(40)
        d:t$(240)
        d:r$(55)
        d:m$(40)
        d:x$(40)
        d:q$(55)
        d:y$(10)
        d:k$(4)
        c:k$="m:&"
        r:voorbeeld programma taal!!!!
*begin  tx:       We hebben het volgende zinnetje:
        :
        :       Ik ga naar school en heb mijn tas bij me.
        :          ~~                ~~~ ~~~~ ~~~      ~~
        :       Zeg die zin in jezelf een paar keer na!!!!
        :
        w:60
        t:       Je moet dit zinnetje in een van de 6 verschillende
        :       persoonsvormen zetten. Dus een van de mogelijkheden
        :       zou kunnen zijn:
        :
        w:60
        t:       Wij gaan naar school en hebben onze tassen bij ons.
        :           ~~~~                ~~~~~~ ~~~~ ~~~~~~     ~~~
        w:60
        t:       Alles duidelijk? Ja of nee?
        a:
        j(%b="nee"):begin
r:Nu organiseren we de goede MATCH
        c:i$="ik    %ga&&&heb&&&&mijn&&&tas&&&&me%%%%"
        c:j$="jij   %gaat&hebt&&&je&&&&&tas&&&&je%%%%"
        c:h$="hij   %gaat&heeft&&zijn&&&tas&&&&zich%%%"
        c:w$="wij   %gaan&hebben&onze&&&tassen&ons%%%"
        c:u$="jullie%gaan&hebben&jullie&tassen&je%%%%"
        c:z$="zij   %gaan&hebben&hun&&&&tassen&zich%%%"
        c:t$=i$!!j$!!h$
        c:t$=t$!!w$!!u$!!z$
```

8.5 Opcodes

```
        t:
        :       Wij kiezen de persoon die je moet nemen!
        :       Dus, wij beginnen met b.v.: Jullie ....
        :       en dan maak jij de zin af met: .....
        :       gaan naar school en hebben jullie tassen bij je.
        :
        :       We beginnen!
        :
        w:60
        tx:
*nog    c:a=(rnd(6)*40)+1
        c:r$=t$(a,40)
        c:x$=r$(1,6)
        th:$x$
        a:
        c:q$=%b
        c:x$=r$(8,4)
        u:routi
        c:x$=r$(13,6)
        u:routi
        c:x$=r$(20,6)
        u:routi
        c:x$=r$(27,6)
        u:routi
        c:x$=r$(34,6)
        u:routi
        t:
        th:nog een? toets 1 in:
        a:#w
        j(w=1):nog
        e:

*routi  c:x$=k$!!x$
        x:x$
        tnh:  fout
        tyh:  goed
        e:
```

8.5 Opcodes

```
        We hebben het volgende zinnetje:

        Ik ga naar school en heb mijn tas bij me.
           ~~                 ~~~ ~~~~ ~~~     ~~
        Zeg die zin in jezelf een paar keer na!!!!

        Je moet dit zinnetje in een van de 6 verschillende
        persoonsvormen zetten. Dus een van de mogelijkheden
        zou kunnen zijn:

        Wij gaan naar school en hebben onze tassen bij ons.
            ~~~~               ~~~~~~ ~~~~ ~~~~~~     ~~~
        Alles duidelijk? Ja of nee?
 ja

         Wij kiezen de persoon die je moet nemen!
         Dus, wij beginnen met b.v.: Jullie ....
         en dan maak jij de zin af met: .....
         gaan naar school en hebben jullie tassen bij je.

         We beginnen!


 ik     ging naar huis en had zijn tas meegenomen
   fout   fout   fout   goed   fout
 nog een? toets 1 in:  1
 jullie gaan naar school en hebben jullie tassen bij je
   goed   goed   goed   goed   goed
 nog een? toets 1 in:  1
 wij    gaan naar school en hebben onze tassen bij ons
   goed   goed   goed   goed   goed
 nog een? toets 1 in:  1
 jij    gaat naar school en heeft zijn pukkeltje bij zich
   goed   fout   goed   fout   fout
 nog een? toets 1 in:  0
```

ticipeerd. Door bij een ACCEPT maar een zeer beperkt aantal b.v. maximaal 3 woorden te testen kunnen we dit soort problemen binnen de perken houden.

De moeilijkheden moeten ook weer niet worden overdreven. Stel dat we bovenstaand programma uitbreiden met nog 4 zinnetjes. Die binnen het raamwerk van het eerste zinnetje passen, dan hebben we op basisschool–niveau toch al een programma, waar voor de basisschool–leerlingen aardig wat valt te leren!!!

'n' NEWCHAR
Komt een bepaald letterteken, symbool of figuurtje vaak voor en is het niet een van de tekens uit de gewone ASCII–lijst dan kan met de 'n'–opcode zo'n figuurtje gemaakt worden. Het gaat als volgt: na de opcode met colon volgt een getal en daarna 8 series van 8 punten of kruisjes (het x–teken). Het getal moet tussen de 32 en 127 zitten. Het resultaat is dat als we de toets indrukken die correspondeert met de ASCII–waarde van dat getal, niet het gewone symbool wordt afgebeeld maar het

8.5 Opcodes

symbool dat overeenkomt met het patroon van punten en kruisjes. De punten en kruisjes mogen op een regel maar ook onder elkaar in groepjes van 8.

In een bepaald programma is een pijltje nodig dat aangeeft dat de returntoets moet worden ingedrukt. Nergens in het programma komen vierkante haken voor. Die twee vierkante haken (ASCII–waarden 91 en 93) worden in onderstaand programma veranderd in een pijltje, dat dus twee lettertekens groot is, dus 16 x 8 pixels.

```
r: een pijltje 1
      d:a$(2)
      n:91 ...XX.....XXX....XXXX...XXXXXXXX.XXXXXXX..XXX......XX.......X...
      n:93 ..............XX......XXXXXXXXXXXXXXXX........................
      c:a$="↵"
      t:en nu een pijltje:  $a$
      t:dit geeft ook een pijltje:↵
      t:en dit ook: #91#93
      w:300
       e:
```

```
r: een pijltje 1
      d:a$(2)
      n:91 ...xx...
           ..xxx...
           .xxxx...
           xxxxxxxx
           .xxxxxxx
           ..xxx...
           ...xx...
           ....x...
      n:93 ........
           ........
           ......xx
           xxxxxxxx
           xxxxxxxx
           ........
           ........
           ........
      c:a$="↵"
      t:en nu een pijltje:  $a$
      t:dit geeft ook een pijltje:↵
      t:en dit ook: #91#93
      w:300
      e:
```

Beide programmaatjes geven dus drie keer een pijltje.

'b' BIT-PATTERN

Opcode voor het definiëren van sprites. Met sprites kunnen figuurtjes worden gemaakt die over het scherm kunnen schuiven en door het in een bepaalde volgorde steeds vervangen van verschillende sprites kan de indruk van een beweging worden gesuggereerd. De Commodore 64 geeft nogal wat mogelijkheden om met SPRITES te werken. In educatieve programma's worden ze veelvuldig toegepast in animaties. Het werken met sprites gaat altijd in twee stappen. Ergens, meestal in het begin van het programma moet het patroon van een sprites met de opcode BIT–PATTERN worden gedefinieerd, later als in de loop van het programma de sprite moet komen opdraven moet dat gebeuren met de 's'–opcode. Na de 's'–opcode komen één of meerdere instructies

8.5 Opcodes

(die we bij de opcode GRAPHICS bespreken) die aangeven waar en hoe de sprite op het scherm moet komen. De 'b'–opcode werkt precies zo als de NEW–CHAR–opcode. In plaats van 8x8 punten en kruisjes krijgen we nu een patroon van 24x21 punten en kruisjes. Het getal meteen na de opcode heeft nu niets te maken met ASCII–waarde of iets dergelijk maar is het nummer van de sprite. Er kunnen in een programma maximaal 8 (0–7) verschillende sprites voorkomen.
In onderstaand programma laten we een balletje diagonaal over het scherm lopen. Het heeft natuurlijk geen zin om van dit programma een schermbeeld te geven.

Wat we op het scherm zien maakt nog niet de indruk van een echt draaiend balletje, daarvoor is deze animatie te eenvoudig. In het eerste deel van de listing worden de twee sprites gedefinieerd. In het laatste deel wordt het balletje over het scherm gestuurd. Een sprite over het scherm laten bewegen gaat altijd door het herhalen van de volgende drie stappen: coördinaten kiezen, sprite plaatsen, sprite weghalen, nieuwe coördinaten bepalen, sprite plaatsen, sprite weghalen enz..
Op de codes die in het laatste deel van het programma worden gebruikt komen we nog uitvoerig terug in het hoofdstuk SPRITES.

Draaien we het programma dan vallen wel een paar zaken op. Om te beginnen zien we in de linker benedenhoek het balletje eerst a.h.w. aangroeien. Het verplaatsen van een sprite kunnen we zo opvatten: stel de sprite voor als een papiertje waar een plaatje op staat. Dit papiertje zit in de hoek links–boven vast aan een coördinaat die het papiertje laat verschuiven. Zit de coördinaat helemaal links–beneden (0,0) dan valt het hele plaatje, de sprite dus, buiten het scherm. Verschuift de coördinaat langs de diagonaal van het scherm dan komt een steeds groter (horizontaal) stuk van de sprite te voorschijn.

Het scherm bestaat uit 318 pixels horizontaal en 192 vertikaal. Loopt de waarde van x en y in gelijke stappen op dan komt de sprite eerder aan de bovenrand dan aan de rechter rand van het scherm. De sprite verdwijnt dus aan de bovenkant, terwijl de waarde van x en y gelijkmatig blijven toenemen. Komt de y–waarde boven de 192, dan wordt van die waarde 192 afgetrokken en komt even later de sprite dus weer aan de onderkant van het scherm tevoorschijn. Omdat de stap waarmee de x– en y–coördinaat toeneemt 4 is, komt de waarde van x=318 niet voor en dus wordt steeds naar '*ga' gesprongen. Nu wordt steeds van de x–waarde 318 afgetrokken en we zien de sprite weer beneden verschijnen en weer in de richting van de diagonaal schuiven. Dus: steeds zien we de sprite in de rechter benedenhoek even langs komen want steeds worden veelvouden van 312 voor de x– en 192 voor de y–richting van de coördinaatwaarden afgetrokken.
Het programma komt dus in een oneindige lus terecht.

'v' VOICE
Met de 'v'–opcode die gevolgd wordt door een instruktielijst wordt de SID–chip, ofwel de geluidchip van de COMMODORE 64 aangestuurd. In het hoofdstuk GELUID komen we daarop terug.

'g' GRAPHICS
De 'g'–opcode wordt gebruikt voor het tekenen van figuurtjes op het scherm, ook hierop komen we in een apart hoofdstuk (GRAPHICS) op terug.

8.5 Opcodes

```
r:een balletje rolt langs diagonaal over het scherm
r:eerste sprite
      b:0 ........................
          ........................
          ........................
          ...........X............
          ........X.X.X.X.........
          .....X.X.X...X.X.X......
          ..X.X.X.X.X.X.X.X.X.X...
          .X.X.X.X.X...X.X.X.X.X..
          X.X.X.X.X.X.X.X.X.X.X.X.
          .X...X...X...X...X...X..
          X.X.X.X.X.X.X.X.X.X.X.X.
          .X.X.X.X.X...X.X.X.X.X..
          ....X.X.X.X.X.X.X.X.....
          .......X.X...X.X........
          ..........X.X...........
          ........................
          ........................
          ........................
          ........................
          ........................
          ........................
r:tweede sprite
      b:1 ........................
          ........................
          ........................
          ...........X............
          ........X.X.X.X.........
          .......X.X.X.X.X........
          ..X.X...X.X.X.X...X.X...
          .X.X.X.X...X...X.X.X.X..
          X.X.X.X.X.X.X.X.X.X.X.X.
          .X.X.X.X.X...X.X.X.X.X..
          X.X.X.X...X.X...X.X.X.X.
          .X.X.X.X.X.X.X.X.X.X.X..
          ....X...X.X.X.X...X.....
          .......X.X.X.X.X........
          ..........X.X...........
          ........................
          ........................
          ........................
          ........................
          ........................
          ........................
```

8.5 Opcodes

```
        c:x=0
        c:y=0
*ga     s:0;y1;e1;lx,y
        w:2
        s:0;e0;lx,y
        s:1;y1;e1;l(x+2),(y+1)
        w:2
        s:1;e0;l(x+2),(y+1)
        c:x=x+4
        c:y=y+4
        j(x<>318):ga
        t:klaar
        w:300
```

'l' LINK

Een lang programma kan beter worden opgedeeld in een paar deelprogramma's. Het werken met deelprogramma's werkt veel prettiger vooral tijdens het programmeren, foutzoeken en testen. Vaak maakt men een hoofdprogramma, waarin de algemene zaken worden geregeld, zoals opvragen van naam en klas van ll. en uitleg voor de bediening en gebruik van het programma. In afzonderlijke deelprogramma's bevinden zich steeds afgeronde delen van het totale programma. Met de 'l'– opcode kan een ander (deel) programma worden opgeroepen.

Na de colon volgt de naam van het programma dat wordt opgeroepen. De in het aanroepende gedefinieerde en/of gebruikte variabelen blijven hun waarde behouden en kunnen dus weer worden gebruikt.

Zodra een programma langer wordt dan 12 K moeten we het in stukken opdelen omdat het geheugen geen grotere programma's kan verwerken. Educatieve programma's bevatten doorgaans nogal wat tekst, zodat die 12 K al vrij vlot vol zijn. Gaan we er vanuit dat een regel gemiddeld 25 karakters lang is, dan kunnen we dus ruim 400 regels in een deelprogramma onderbrengen. Stukken van ongeveer 400 regels zijn lang genoeg om mee te werken en blijven toch nog wel hanteerbaar.

Om een idee te krijgen over de lengte van een goed afwisselend programma wat door een ll. in ongeveer 1 uur kan worden gedaan komen we toch al wel gauw op ruim 100 K. Het gaat dan om een programma dat bestaat uit kaartjes, sprites, verschillende animaties, uitleg en b.v. nog een kleine overhoring. Het gaat dan dus niet om een programma genaamd: Veel van hetzelfde. Het totale programma moet dan worden opgedeeld in minstens een 10–tal module's. Het opdelen van een programma moet wel natuurlijk met overleg plaatsvinden. Bij het aanroepen van een nieuwe module blijven de waarden van de gedefinieerde strings en numerieke–variabelen behouden. Omdat we in totaal over maar 26 verschillende karakterstrings kunnen beschikken zal vaak in een nieuwe module de naam van een string van een voorgaande module gebruikt moeten worden. Dat is geen enkel bezwaar als we daar maar rekening mee houden.

In onderstaand voorbeeld roept het hoofdprogramma (HOOFD) eerst MODULE1 aan en vervolgens MODULE2.

Afgezien van het feit dat het werken met een variabele (a) die in het begin van het programma 'HOOFD' nog geen waarde heeft gekregen minder fraai is, zien we dat de strings 'naam' en 'klas' en de variabelen 'b' en 'c' steeds hun waarden blijven houden.

8.5 Opcodes

```
r:hoofd programma voor module 1 en module 2
       j(a=2):uit1
       j(a=3):uit2
       d:n$(20)
       d:k$(4)
       d:r$(20)
       t:        Hoe heet je?
       a:$n$
       t:        In welke klas zit je?
       a:$k$
       t:        Nu volgt nog het een en ander
       w:30
       t:        We gaan nu naar het eerste module
       l:module1
*uit1  t:        We zitten weer in hoofd
       t:        We gaan nu naar het tweede module
       l:module2
*uit2  t:        We zitten weer in hoofd en zijn klaar
       :
       :       ll. $n$  uit klas $k$  heeft
       :       #b  en #c  als antwoorden gegeven.
       t(b=2&c=4):      Niet slecht!!
       t(b<>2!c<>4):      Kan beter!!
       w:300
       e:


r:dit is module een bij hoofd.
      t:        De eerste som $n$
      t:        Wat is 1 + 1 ?
      a:#b
      t:        O.k. terug naar hoofd
      c:a=2
      l:hoofd
      e:


r:dit is module twee bij hoofd.
      t:        De tweede som $n$
      t:        Wat is 2 + 2 ?
      a:#c
      t:        O.k. terug naar hoofd
      c:a=3
      l:hoofd
      e:
```

8.5 Opcodes

```
        Hoe heet je?
Alfred Jakobus Kwak
        In welke klas zit je?
4e
        Nu volgt nog het een en ander
        We gaan nu naar het eerste module
        De eerste som Alfred Jakobus Kwak
        Wat is 1 + 1 ?
3
        O.k. terug naar hoofd
        We zitten weer in hoofd
        We gaan nu naar het tweede module
        De tweede som Alfred Jakobus Kwak
        Wat is 2 + 2 ?
4
        O.k. terug naar hoofd
        We zitten weer in hoofd en zijn klaar

        ll. Alfred Jakobus Kwak uit klas 4e heeft
        3 en 4 als antwoorden gegeven.
        Kan beter!!
```

'f' FILE

In een programma kan het nodig zijn dat bepaalde gegevens die in een file op de schijf staan worden gebruikt, eventueel worden veranderd en vervolgens weer naar de schijf worden weggeschreven. Voor deze handelingen is de 'f'—opcode. De 'f' wordt gevolgd door een 'i' voor het inlezen van een file van de schijf, en gevolgd door een 'o' voor het wegschrijven van een file.

Voor het bijhouden van de score van ll. is de 'f'—opcode onmisbaar. Het bijhouden zelf van b.v. de score van een aantal ll. vereist een zeer preciese manier van file's oproepen, oude file van de schijf wissen, nieuwe file bijwerken en weer wegschrijven. We komen op deze techniek uitgebreid terug in het hoofdstuk KEEP RECORDS.

Wordt aan de 'f' een 'x' toegevoegd dan kan achter de colon een instruktie van het disk—operatingsysteem komen. Die instruktie wordt meteen uitgevoerd. Formateren van een disk, verwijderen en veranderen van namen van files kan dan worden uitgevoerd.

b.v.:

fx:n:naam,getal	formateert een nieuwe schijf. De naam mag 16 karakters lang zijn en het getal mag maximaal 2 bytes in beslag nemen, te gebruiken voor volgnummers e.d.
fx:s:naam	verwijdert de genoemde file
fx:r:nieuw=oude	verandert oude naam in nieuwe naam

8.5 Opcodes

'z' CALL
In educatieve programma's kan gebruik gemaakt worden van apparatuur waarop voor de ll. relevante gegevens staan, denk b.v. aan dia's, video, recorders, c.d. e.d. De 'z'—opcode die gevolgd wordt door een getal is nodig voor de koppeling tussen computer en aan te sturen apparaat. We gaan op deze koppeling niet uitgebreid in. We volstaan met het volgende voorbeeld.

Stel dat voor het aanleren van een vreemde taal men aan een computerprogramma uitspraakvoorbeelden van bepaalde woorden/zinnen uit die taal wil koppelen. Voor het aansturen van de recorder is een apart stukje programma nodig. Dit stukje programma wordt vanuit het PILOT—programma aangeroepen en laat de band tot een bepaald stopteken verder lopen. Dit stukje programma moet voor het starten van het PILOT—programma worden ingelezen op een bepaalde plek van het geheugen, dat speciaal voor gebruikersroutines is gereserveerd. Aan het eind van die routines moet weer een sprong naar het aanroepende programma staan.

Stuit het PILOT—programma op een 'z ' — opcode gevolgd door een getal, dan komt een sprong naar de lokatie in het geheugen dat door dat getal wordt aangegeven. Het getal wordt opgegeven in absolute notatie. De routine wordt doorlopen, en dus wordt het apparaat aangestuurd waarna weer terug gesprongen wordt en het programma verder gaat op de plaats waar het gebleven was.

In het geheugen zijn twee gebieden voor dit soort routines gereserveerd:

3584—4096	($0E00—$0FFF)	512 bytes
28160—28671	($6E00—$6FFF)	512 bytes

'e' END
De 'e'—opcode staat aan het eind van een programma en subroutine. Wordt ergens met een 'u'—opcode naar een subroutine gesprongen dan wordt bij de eerstvolgende 'e'—opcode teruggesprongen naar de regel volgend op de regel waarvandaan is gesprongen naar de subroutine. De 'e'—opcode aan het eind van een programma zorgt voor een overschakeling naar de COMMAND—mode. Aan het eind van de programma's staat bijna altijd 'w:300' om ons nog even de gelegenheid te geven de resultaten van het programma te bekijken alvorens wordt overgeschakeld naar de COMMAND—mode en dan dus alle gegevens van het scherm zijn verdwenen. Volgt op de colon van de 'e'—opcode de naam van een label, dan wordt naar die plek gesprongen. Na de colon kan ook: a, p en m staan. Er wordt dan steeds naar de laatst ACCEPT, PROBLEM of MATCH gesprongen.

8/8.6

GRAPHICS

8/8.6.1
De grafische codes

Omdat in onderwijzersprogramma's plaatjes, tekeningetjes en bepaalde symbolen een belangrijke rol spelen wijden we aan dit onderwerp een apart hoofdstuk. In het algemeen geldt dat voor het tekenen van een figuurtje de "g"-opcode gebruikt wordt. In het veld van de opcode staat een aantal codes die gescheiden worden door het ";"-teken. Met die verschillende codes wordt als het ware een pen, al of niet schrijvend, over het scherm gestuurd.
Getekend wordt over het scherm dat is opgebouwd uit 320x200 puntjes, de z.g. pixels, 320 horizontaal en 200 vertikaal. Elk pixel kan apart worden aangegeven (aangestuurd) met een code in het veld van de "g"-opcode. In PILOT kunnen we niet alle pixels gebruiken, n.l. met de y-coördinaat kunnen we niet verder dan 192 en voor de x-coördinaat niet verder dan 319. De oorsprong van het assenstelsel, zoals de wiskundigen dat zo mooi noemen, ligt in de linker beneden hoek.

In onderstaand voorbeeld wordt een kader om het scherm getekend. Het kader bestaat uit twee evenwijdige lijntjes. Het tekenen begint in de linker benedenhoek en gaat dan rechts om.

```
r:programma tekent kader
g:e;p0,1;d0,192;d319,192;d319,1;d0,1
w:30
g:p2,3;d2,190;d317,190;d317,3;d2,3
e:
```

8.6 Grafische codes

In het veld van de "g"-opcode staat dus een aantal instrukties, die één voor één worden uitgevoerd. Die codes worden zoals al is opgemerkt gescheiden door een ";"-teken. Komen in de codes coördinaten x- en y voor dan worden die gescheiden door een ",". In de eerste regel staat meteen na de : een "e". Die "e" geeft de opdracht: maak scherm schoon. Vervolgens moet de pen als het ware worden neergezet; dit gebeurt met de "p"-code (put) gevolgd door de coördinaten van het punt (pixel) waar dat moet gebeuren. Daarna wordt met de "d"-code (draw) een lijn getrokken naar het aangegeven punt.

De eerste regel tekent dus een lijn langs de uiterste rand van het scherm. Na intikken en runnen van bovenstaand programmaatje valt ons meteen op dat de lijnen vertikaal en horizontaal op geen stukken na even dik zijn. Ook zien we dat de tweede kaderlijn in horizontale richting verschilt van die in vertikale richting. De tweede horizontale kaderlijn komt "los" van de eerste, wat we ook verwachten, (2 pixels opgeschoven!); de vertikale lijnen worden niet gescheiden. Het resultaat is wel fraai maar niet wat we op grond van de gebruikte coördinaten zouden verwachten. We kunnen het zó opvatten: kennelijk zijn de vertikale pixels wèl, maar de horizontale pixels niet afzonderlijk aan te sturen.

Dus:

g:p22,30 en g:p22,31

geven op het scherm beiden hetzelfde punt en:

g:p22,30 en g:p23,30

geven verschillende punten op het scherm.

In onderstaande lijst staan alle codes die in het veld van een "g"-opcode kunnen voorkomen.

-- e   Maakt het scherm schoon en zet de pen links beneden (0,0).

-- px,y   Zet de pen neer op de plek (x,y) en zet daar een punt.

-- dx,y   Tekent een lijn van het punt waar de pen op dat moment staat naar het punt (x,y).

-- mx,y   Verplaatst de pen, die we nu maar weer cursor zullen noemen, naar het punt (x,y) zonder een zichtbaar gevolg. Dus, geen punt zetten en ook niet een lijn van het oorspronkelijke punt naar (x,y).

-- rx,y   Verwijder de lijn die is getrokken van het oorspronkelijke punt waar de cursor staat naar het punt (x,y).

-- qx,y   Haal op de plek (x,y) het gezette pixel weg en laat de cursor op die plek staan.

-- bn   Code voor achtergrondkleur. Het getal "n" staat voor een bepaalde kleur. Zie verder: KLEUR

-- cn   Code voor voorgrondkleur. Dus de kleur van de figuurtjes die we tekenen. Het getal "n" staat voor een bepaalde kleur. Zie verder: KLEUR.

-- xn   Code voor kleur van de rand van het scherm. Die rand is voor het programma niet te gebruiken. Het getal "n" staat voor een bepaalde kleur. Zie verder: KLEUR.

-- fx,y   Tekent een rechthoek en vult dat op met de dan geldende voorgrondkleur. Een van de hoekpunten is op de plek waar de cursor staat, het overstaande hoekpunt is de plek die wordt aangegeven

8.6 Grafische codes

door de coördinaten bij de "f"-code. Na afloop blijft de grafische cursor op de plek staan.

-- t Er zijn in feite twee cursors in het spel n.l. de tekstcursor en de grafische cursor. Met de "t"-code springt de tekstcursor naar de karakterpositie waarin de grafische cursor zich bevindt (ergens op het scherm staat de grafische cursor, dat is natuurlijk altijd binnen een gebiedje van 8x8 pixels waar we een karakter kunnen zetten).

-- ox,y De absolute waarden van de coördinaten kunnen met de "o"-code worden veranderd. De "o" staat voor "offset". We kunnen ook zeggen: de oorsprong van het coördinatiestelsel wordt op een andere plek van het scherm gelegd. Zodra dat is gedaan zal elke keer dat een m,d,f,p,of q-code wordt met die nieuwe oorsprong van de coördinaten rekening worden gehouden.

-- sn Splitst het scherm in twee delen. Omdat deze code voor een onderwijsprogramma zo belangrijk is wijden we daar de hele volgende paragraaf aan.

Het schoonmaken van het scherm kan op twee verschillende manieren worden gedaan, n.l. met de t-opcode en met de screen-modifier, dus met "ts:" en met de "e"-code. Tussen de twee methoden is een klein verschil. De "ts"-opcode zet alleen de tekstcursor in de hoek links boven, de grafische cursor blijft dan op zijn plek staan. De "e"-code zet de grafische cursor in de hoek links beneden.
Onderstaande programmaatje geeft dus twee verschillende lijntjes.

```
r:scherm schoonmaken
       g:p0,100
       t:er staat nu wat op het scherm
       w:50
r:nu maken we het scherm schoon en trekken
r:een lijntje
       ts:
       g:d300,190
       w:50
r:nu maken we weer het scherm schoon en trekken
r:hetzelfde lijntje?
       t:iets op het scherm
       g:e;d300,190
       w:300
       e:
```

8/8.6.2
SPLIT-code (vensters)

Bijna alle moderne versies van de meest gebruikte talen kennen de zogenaamde vensters. Een venster is een gedeelte van het scherm waarin kan worden gewerkt zonder dat de rest van het scherm daardoor wordt beïnvloed.
Bijvoorbeeld: in een bepaald programma willen we de ll. de gelegenheid geven steeds een bepaald stukje informatie te bekijken. Stel dat in een bepaald programma een gedeelte van kaart van Noord-Holland, waarin de plekken zijn aangegeven waar elke ochtend filevorming rond Amsterdam is, voorkomt. Het probleem is: hoe los je die filevorming op? Het betreft een les over ruimtelijke ordening. Dus de bedoeling is dat elk moment dat de ll. dat wenst, meteen dat kaartje op het scherm verschijnt.
Dat kan zo:
Een van de functietoetsen (hoe dat in zijn werk gaat wordt in het hoofdstuk SYSX behandeld) roept een subroutine op waarin het venster wordt gedefinieerd, dus als het ware wordt een stuk papier op het scherm gelegd. Vervolgens wordt om het geheel een wat fraaier aanzien te geven een kader getekend en daarna het kaartje op een of andere manier opgeroepen in de vorm van b.v. sprites. In de subroutine komt vervolgens een ACCEPT. Zolang geen toets is ingedrukt blijft het kaartje dus op het scherm. Is een toets ingedrukt dan gaat de routine verder met het weghalen van het kaartje. Het programma gaat daarna weer verder op de volgende regel.

De COMMODORE-versie van PILOT kent geen echte vensteropcode maar wel een die daar veel op lijkt. Met de "s"-code (SPLIT), die dus staat in het veld van een "g"-opcode, wordt het scherm gesplitst in twee horizontale delen. Het onderste gedeelte kunnen we als tekstvenster gebruiken. Het bovenste voor plaatjes, figuurtjes enz.

Stel dat we vragen willen stellen over een of andere scheikundige reactie. Op het scherm tekenen we, b.v. met sprites de reaktie (vanwege indices e.d. kan zo'n reaktie niet zomaar van het toetsenbord worden ingetikt). Vervolgens splitsen we het scherm met:

```
g:s2
```

De vragen kunnen nu met gewone type-instructies op het scherm worden gezet en antwoorden van ll. kunnen worden ingetikt enz. Dat laatste gebeurt alleen in het onderste deel van het scherm. Als het onderste deel vol is wordt automatisch naar boven gescrold.
Het bovenste deel van het scherm blijft onveranderd. Met het getal (0-5), de zogenaamde splitswaarde, achter de "s" code, wordt aangegeven hoe de splitsing is. Zie onderstaande lijst:

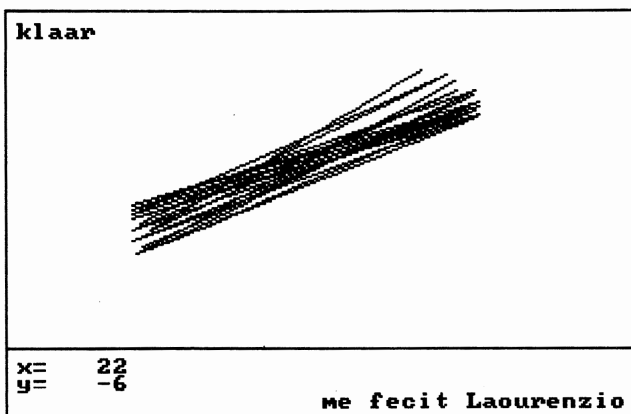
splits-waarde	aantal regels boven	aantal regels beneden
0	0	24
1	3	21
2	7	17
3	11	13
4	15	9
5	19	5

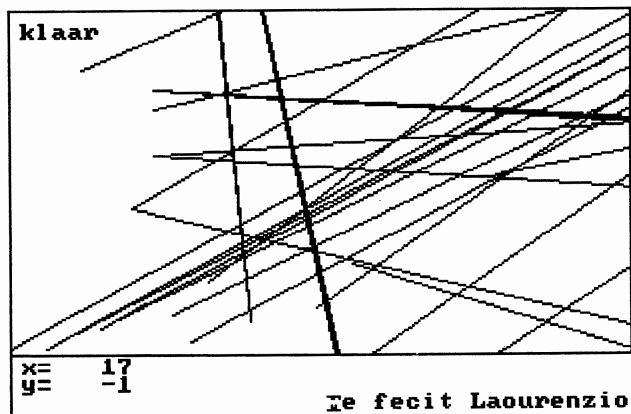
Bekijken we de tabel dan zouden we dus kunnen zeggen: de "default"-splitswaarde is 0. Zodra een splits-code voorkomt wordt de tekstcursor in de hoek links boven van het tekstvenster gezet.

8.6 Grafische codes

In onderstaande programma's komt een aantal keren de "g"-opcode voor.

```
r:compositie 25 lijntjes
        c:a=0
        c:x=0
        c:y=0
        g:e;p80,50
*herha  g:d(240+x),(150+y)
        c:x=(rnd(63))-31
        c:y=(rnd(31))-15
r:      g:o(5*x),(5*y)    *******
        c:a=a+1
        g:s5
        ts:
        :
        :x=     #x
        :y=     #y
        :a=     #a
        g:s0
        g:d(80+x),(50+y)
r:      w:50              *******
        j(a<>25):herha
r:      w:50              *******
        t:klaar
        g:s5
        t:
        :
        :
        :                     me fecit Laourenzio
w:300
e:
```





8.6 Grafische codes

De compositie van 25 lijntjes geeft duidelijk aan hoe met de graphics-codes wordt gewerkt. De gedachte achter het programma is: trek steeds 2 lijnen, waarbij het beginpunt van de tweede lijn het eindpunt van de eerste lijn is.
Eerst wordt natuurlijk de pen neergezet in het punt (80,50). Vervolgens wordt de eerste lijn getrokken naar (240,150). Daarna worden nieuwe "willekeurige" waarden voor het eindpunt van de tweede lijn bepaald en de lijn naar dat punt getrokken. Vóór die tweede lijn getrokken wordt willen we de waarden van x, y en a (aantal paren lijnen) weten. Om de "compositie" niet te verstoren splitsen we het scherm, maken het tekstscherm schoon, schrijven de x, y en a op en sluiten het tekstscherm weer.
Daarna herhaalt de procedure zich. Tot slot wordt nog even in het tekstscherm de compositie gesigneerd.

Het programmaatje is nogmaals gedraaid waarbij de "o"-code is gebruikt om ook het beginpunt van de tweede lijn willekeurig te kiezen. We krijgen nu een volledig andere compositie.

De "composities" zijn met een veel korter programmaatje te maken. Waar het ons om gaat is dat door wachtlussen in te voegen we precies kunnen zien hoe de "SPLIT"-code werkt.

Het scherm kan enerzijds worden opgevat als een stuk papier met hokjes van 40 (horizontaal) x 24 (vertikaal) voor karakters die gedefinieerd zijn volgens de ASCII-tabel en anderzijds door een stuk grafiekenpapier van 320 (horizontaal) x 192 (vertikaal). Als een stukje tekst in een grafiek moet worden gezet is het prettig te weten welke grafische coördinaten behoren bij, laten we maar zeggen, karaktercoördinaten en omgekeerd. In onderstaande tabellen kunnen die waarden worden afgelezen. Voor de y-richting wordt ook de s-waarde (splits-waarde) gegeven.

x-richting

kolom	x-waarde	kolom	x-waarde	kolom	x-waarde	kolom	x-waarde
1	0-7	11	80-87	21	160-167	31	240-247
2	8-15	12	88-95	22	168-175	32	248-255
3	16-32	13	96-103	23	176-183	33	256-263
4	24-31	14	104-111	24	184-191	34	264-271
5	32-39	15	112-119	25	192-199	35	272-279
6	40-47	16	120-127	26	200-207	36	280-287
7	48-55	17	128-135	27	208-215	37	288-295
8	56-63	18	136-143	28	216-223	38	296-303
9	64-71	19	144-151	29	224-231	39	304-311
10	72-79	20	152-159	30	232-239	40	312-319

8.6 Grafische codes

y-richting

s0		s1		s2		s3		s4		s5	
rij/	y-w	rij/	y-w	rij/	y-w	rij/	y-w	rij/	y-w	rij/	y-w
1	184-191	4	160-167	8	128-135	12	96-103	16	64-71	20	32-39
2	176-183	5	152-159	9	120-127	13	88-95	17	56-63	21	24-31
3	168-175	6	144-151	10	112-119	14	80-87	18	48-55	22	16-23
		7	136-143	11	104-111	15	72-79	19	40-47	23	8-15
										24	0-7

8/8.6.3 Kleur

Door op een verstandige manier gebruik te maken van kleuren kunnen programma's een stuk levendiger, fraaier een ook veel duidelijker worden. Belangrijk is dat de gekozen kleuren voor de achtergrond en de kleur van de tekst duidelijke kontrasten geven waardoor de beeldschermen prettiger zijn te lezen. Verder is nog belangrijk dat een eenmaal gekozen kleurcombinatie tijdens het hele programma niet verandert en ook, dat een spaarzaam gebruik van de verschillende kleuren de tekst veel beter ondersteunt dan het gebruik van veel verschillende kleuren. Aan de functie van layout en de daarbij gebruikte kleuren kan een hele studie worden gewijd. Hier geven we alleen de manier van gebruik.

De volgende 16 kleuren kunnen worden gebruikt

0	zwart	4	purper	8	oranje	12	grijs 2
1	wit	5	groen	9	bruin	13	licht-groen
2	rood	6	blauw	10	licht-rood	14	licht-blauw
3	cyaan	7	geel	11	grijs 1	15	grijs 3

Zoals we zien worden de verschillende kleuren aangegeven met een cijfercode. De ingestelde code van PILOT is: 13 en 0, d.w.z. dat als geen instructies om de kleur te veranderen worden gegeven de rand- en achtergrondkleur licht-groen en de tekst zwart is.

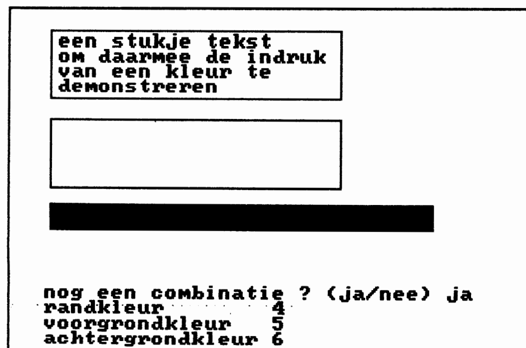
Van die 16 verschillende kleuren moet men zich niet te veel voorstellen. Vooral als een T.V. als monitor wordt gebruikt, hangt de kleur ook erg af van de instelling van het toestel.

Met onderstaand programma kan een kleurcombinatie worden uitgetest. Het programma begint met de ingestelde kleurcombinatie van PILOT, waarna elke andere kleurcombinatie kan worden gekozen. De zwarte balk in het midden van het schermbeeld laat (alleen in werkelijkheid natuurlijk) de verschillende kleuren zien. In de 9e regel van het programma zien we hoe de kleuren met de g-opcode worden ingesteld. In het programma zien we ook een voorbeeld van het gebruik van "fx,y". In de do-loop waarin steeds een volgende rechthoek met een

8.6 Grafische codes

bepaalde kleur wordt getekend (regel 23) wordt bij "ct" steeds een nieuwe kleur aktief, waarna in "fx,14" het rechthoekje wordt gekleurd. Het rechthoekje heeft als overstaande hoeken de plaats waar de grafische cursor staat vòòr het inkleuren en de coördinaat "x,14". Vervolgens zorgt "px,o" ervoor dat de grafische cursor weer terugspringt naar de goede y-coördinaat.

```
      r:kleur             p=randkleur
      r:                  q=voorgrondkleur
      r:                  r=achtergrondkleur
      c:p=13
      c:q=13
      c:r=0
      d:j$(2)
*nog  g:xp;cq;br
      g:s0
      ts:
      :
      :     een stukje tekst
      :     om daarmee de indruk
      :     van een kleur te
      :     demonstreren
      g:o26,134
      g:p1,1;d170,1;d170,50;d1,50;d1,1
      g:o26,74
      g:p1,1;d170,1;d170,50;d1,50;d1,1
      c:x=26
      c:t=0
*her  g:ct;fx,14;px,0
      c:t=t+1
      c:x=x+14
      j(t=16):ver
      j:her
*ver  g:s5
      ts:
      th:   nog een combinatie? (ja/nee)
      j(j$<>"ja"):eind
      th:   randkleur
      a:#p
      th:   voorgrondkleur
      a:#q
      th:   achtergrondkleur
      a:#r
      j(j$="ja"):nog
*eind e:
```



8/8.7

Sprites

In hoofdstuk 5.1 hebben we al kennis gemaakt met sprites. Een sprite is, zoals daar al is uitgelegd, een tekeningetje dat we over het scherm kunnen laten schuiven. Het tekeningetje wordt aan het begin van een programma eerst gemaakt (gedefinieerd) met de "b"-opcode. In een programma kunnen maximaal 8 verschillende sprites voorkomen. De nummering (0-7) vindt plaats bij de "b"-opcode. Het getal na de : geeft het nummer van de sprite. Voor het plaatsen en weghalen van sprites wordt de "s"-opcode gebruikt. Die code werkt ongeveer zoals de "g"-opcode. Evenals bij die opcode komen in het veld de verschillende codes die aangeven waar en hoe een sprite moet worden geplaatst.

Na de colon van de "s"-opcode komt het nummer van de sprite, daarna volgen de verschillende code's gescheiden door een ";". In het veld van de "s"-opcode kunnen net zoveel code's voorkomen als voor de manipulatie van de sprite nodig is. Let wel: bij elke "s"-opcode gaat het alleen om de sprite waarvan het nummer achter de colon staat.

In het veld van de opcode kunnen de volgende code's voorkomen.

-- e1	maakt de sprite zichtbaar dus: sprite neerzetten
-- e0	maakt de sprite onzichtbaar dus: sprite weghalen
-- x1	verdubbel de horizontale afmeting
-- x0	zet sprite weer in gewone horizontale afmeting
-- y1	verdubbel de vertikale afmeting
-- y0	zet sprite weer in gewone vertikale afmeting
-- p1	zet de sprite achter de eventueel voorkomende tekening
-- p0	zet de sprite voor de eventueel voorkomende tekening
-- cn	zet de voorgrondkleur van de sprite op kleur n
-- lx,y	zet de sprite op plaats met coördinaten x en y op het scherm. De coördinaten corresponderen met de linker bovenhoek van de sprite. x: 0-318, y:1-192. Voor x en y kunnen ook uitdrukkingen staan die dan eerst worden uitgerekend.

De volgende 4 code's zijn voor de z.g. multicolor-mode (zie hierna):

-- m1	zet de sprite in de multicolor-mode
-- m0	de gewone color-mode (voorgrond- en achtergrondkleur)
-- rn	code voor multicolor-mode
-- qn	code voor multicolor-mode

8.7 Sprites

Zonder bijzondere voorzorgen komt een sprite in 2 kleuren op het scherm. Een "." en "x" in het bit-patroon wordt de achtergrond - respectievelijk voorgrondkleur. In de z.g. multicolor-mode kan een sprite in vier kleuren op het scherm worden gezet. Met de code "m1" schakelen we over naar de multicolor-mode.

In de mode wordt het bit-patroon in een regel van 24 posities bij de "b"-opcode opgedeeld in twaalf groepjes van 2 posities. Elk groepje van 2 posities krijgt de kleur die bepaald wordt door de combinatie van punten en kruisjes van dat groepje. De vier mogelijke combinaties zijn:

".." de dan geldende achtergrondkleur
"xx" de geldende voorgrondkleur gezet door de code "cn"
"x." de kleur gezet door code "rn"
".x" de kleur gezet door de code "qn"

Het werken in de multicolor-mode vereist extra aandacht voor wat betreft de indeling van de punten en kruisjes in de spritedefinitie. Het is aan te bevelen eerst de sprites te tekenen op een raster van 24x21 hokjes. De rijen van 24 posities moeten daarbij gezien worden als 12 groepjes van 2 posities. Pas op: zijn door de code's "qn" en "rn" de kleuren eenmaal gekozen, dan gelden die kleuren voor alle sprites en dus niet alleen voor de sprite waarvoor de code's zijn gedefinieerd!
Voor het definiëren van sprites kan de sprite-editor op het PILOT-schijfje worden gebruikt. In feite is het niets meer dan een programma dat een set van 8 blanco sprites netjes naast en onder elkaar op het scherm zet. Als de sprites zijn gedefinieerd kan het overbodig geworden deel van het programma worden weggehaald en de overgebleven definitie in een ander programma worden opgenomen.

In onderstaand programma zien we hoe met sprites wordt gewerkt. Een poppetje neemt, al lopend over het scherm, zijn hoed af. Het schermbeeld geeft natuurlijk een momentopname.

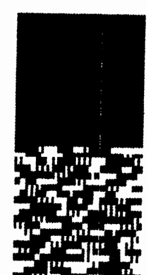
```
r:demonstratie van sprites, poppetje loopt
r:over het scherm
g:c0;b13;x13;e
g:e;m0;16,t
u:def
r:tonen van sprites
s:2:c0;x1;y1;m0;1000,180;e1
w:20
s:4:c0;x1;y1;m0;1060,180;e1
w:20
s:3:c0;x1;y1;m0;1120,180;e1
w:20
s:5:c0;x1;y1;m0;1180,180;e1
w:20
s:1:c0;x0;y0;m0;1240,180;e1
w:20
c:x=0
s:1:c0;x1;y1;m0;1060,90;e1
s:6:c0;x1;y1;m0;1060,48;e1
w:40
```

8.7 Sprites

```
*loop  s:2;c0;x1;y1;m0;lx,90;e1
       s:3;c0;x1;y1;m0;lx,48;e1
       w:5
       s:2;e0
       s:3;e0
       s:4;c0;x1;y1;m0;l(x+8),90;e1
       s:5;c0;x1;y1;m0;l(x+8),48;e1
       w:5
       j(x>120):eind
       s:4;e0
       s:5;e0
       j:loop
*eind  w:300
       e:
*def
r:definitie van de sprites
b:1
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
b:2
........................
........................
........................
............XXX.........
.........XXX...X........
........XX...XXX........
........XXXXX..XX.......
.....XXXXXX....XX.......
....XX.XXXX.X.X..X......
...X..XXX........X......
..X...X..XX.XX.XX.......
```

8.7 Sprites

```
.X...X....X....X........
.X....X..XXXXXXXX.......
..XX...XXX.....X.X......
....XX....XXXXX..X......
......X..........X......
.......X.......X.X......
........X.....XX.X......
.........X.....X.X......
.........X.....X.X......
.........X.....X.X......
b:4
.........XX.............
.......XX..X............
.....XX..XX.............
....X..XX...............
....XXX....XXXX.........
...XXX....XXXXXX........
..XXXX...XXXX..XX.......
.XXX.....XX....XX.......
.X.X.....XX.X.X..X......
.X.X....X........X......
.X.X.....XX.XX.XX.......
.X.XX.....X....X........
.X...XX..XXXXXXXX.......
..XX...XXX.....X.X......
....XX....XXXXX..X......
......X..........X......
.......X.......X.X......
........X.....XX.X......
.........X.....X.X......
.........X.....X.X......
.........X.....X.X......
b:3
.........X.....X.X......
.........X.....X.X......
.........XXXXXXX.X......
........X......XXX......
........X.......XXX.....
.......X........XXX.....
......X..........XX.....
......X....X.....X......
.....X....X.XX...X......
.....X...X....X..X......
......X..X....X..X......
......X..X...X..X.......
.......X..X.X...X.......
.......X.XX.XXXX........
.......XX..X.X.XXXX.....
......XXXX.XXXX....X....
.....X....X.X......X....
.....XXXXX..XXXXXXXX....
........................
........................
........................
b:5
.........X.....X.X......
.........X.....X.X......
.........XXXXXXX.X......
........X......XXX......
........X.......XXX.....
.........X......XXX.....
.........X...X...XX.....
..........X...X..X......
..........X....XXX......
...........X.....X......
............XX....XX....
.............XXX...X....
............X...X.XXXX..
............XXXX.XXX..X.
.............X.XXXX..X..
............XXX.X...X...
............X...X..X....
............XXXXXXX.....
........................
........................
........................
e:
```



8.7 Sprites

Toelichting
Het poppetje heeft de afmeting van 2 vergrote sprites (met x1 en y1 wordt de horizontale- en vertikale afmeting verdubbeld). Eerst worden de sprites naast elkaar op het scherm gezet. Naast deze 4 sprites staat nog een sprite die echter niet is gedefinieerd. Draaien we het programma dan zien we dat, zodra een op het scherm staande sprite nogmaals wordt aangeroepen, de eerste weer verdwijnt. Een en dezelfde sprite kan dus maar één keer op scherm voorkomen.
In het programmaatje regelen de variabele "x" en de wachtlussen de snelheid waarmee het poppetje loopt. In de regels waarin een bepaalde sprite op het scherm wordt gezet geldt dat de kleurcode voor de sprite, in het programma steeds "c0", alleen voor die enkele sprite geldt.

Laten we sprites over elkaar heen schuiven dan hangt het van het spritenummer af welke sprite voor of achter de andere sprite langs gaat. Hoe lager het spritenummer hoe hoger de prioriteit van de sprite, wat hier betekent dat die sprite beter zichtbaar is. Bijvoorbeeld: stel dat sprite nr 2 over het scherm schuift waar achtereenvolgens de sprite's 1 en 3 staan. Het blijkt dan dat sprite 2 achter 1 en voor 3 langs schuift. Draaien we het programmaatje dan zien we iets dergelijks met de boven- en onderkant van het poppetje als het langs de stilstaande sprites schuift. Behalve voor en achter andere sprites schuiven, kunnen spirtes ook voor en achter op het scherm staande figuren en karakters schuiven. Met de code "p0" komen sprites voor, en met "p1" achter de figuren op het scherm.

Animaties in de multicolor-mode zijn erg moeilijk te maken. Wat wel in die mode lukt is sprites steeds over elkaar heen leggen waardoor bepaalde effecten worden bereikt. Vaak zal dit echter niet worden toegepast omdat het maken van dergelijke sprites een erg tijdrovend werk is. Of nu wel of niet in de multicolor mode wordt gewerkt, het verdient altijd aanbeveling de sprites eerst op een ruitjespapier te ontwerpen. Ook moet terdege rekening worden gehouden met de nummering van de sprite. Definiëren van bijvoorbeeld spritenummer 3 heeft tot gevolg dat lagere spritenummers (0,1,2) die in het programma niet zijn gedefinieerd ook het patroon van sprite 3 bevatten en dus niet als blanco sprites kunnen worden gebruikt.

8/8.8

Record-keeping

In hoofdstuk 5.p, waarin de "f"-opcode werd besproken, is reeds opgemerkt dat in onderwijsprogramma's het vaak nodig is, de score van de ll. bij te houden. Ook is, bij het uittesten van een programma, belangrijk te weten, wat een ll. op de verschillende vragen heeft geantwoord. Door het bestuderen van die antwoorden kunnen onduidelijkheden, fouten, vergissingen e.d. worden opgespoord. Dit laatste is erg belangrijk omdat een onderwijsprogramma pas goed is als het goed anticipeert op de meest voorkomende *foutieve* antwoorden. Pas als een programma een paar keer door verschillende ll.'n is doorlopen en alle gegeven antwoorden goed zijn bestudeerd kan iets gezegd worden over het functioneren van het programma in een leersituatie.

De I.B.M.-versie van PILOT heeft voor het bijhouden van antwoorden van ll.'n een aparte opcode. Dat werkt bijzonder prettig als het erom gaat een gegeven antwoord op de schijf te zetten. Het enige wat na een antwoord (een accept a:) in het programma moet staan is een regel met: "k:%b". Als op de schijf een file staat met genaamd "k", dan wordt in die file de inhoud van de antwoordbuffer bijgeschreven. Verder hoeven we niets te doen, althans niet voor het inlezen, wegschrijven, organisatie binnen de file e.d. Is die file nog niet aanwezig dan wordt die alsnog aangemaakt. Het enige waarop gelet moet worden is dat die file niet al te snel groeit, dus dat niet teveel antwoorden in die file worden opgenomen, waardoor het schijfje (te) snel vol is. Om later de file makkelijk te kunnen lezen, moeten wat simpele voorzorgen genomen worden.

Helaas kent de Commodore-versie van PILOT de "k"-opcode niet. Maar, wat hiervoor is beschreven, kan ook met de file-opcode worden uitgevoerd. Echter moeten dan wel een paar voorzorgen worden genomen. Hoe een en ander werkt laten we in een voorbeeld zien.

Eerst nog de volgende opmerkingen:

1. Alle communicatie tijdens het runnen van een programma met de schijf gaat via de z.g. "filebuffer". Deze buffer kan in het programma worden aangeroepen met de systeem variabele "%f".
2. De lengte van "%f" en dus ook de inhoud van de buffer kan maximaal 4096 bytes lang zijn.
3. Inlezen van een file van de schijf naar de buffer gaat met: "fi:naam". De naam mag 16 lettertekens lang zijn.
4. Naar de schijf schrijven van de inhoud van de filebuffer gaat met: "fo:naam".
5. In het statement van de filebuffer "%f" komen altijd twee indices. De eerste

8.8 Record-keeping

stelt de positie en de tweede de lengte van de substring voor. De posities lopen van 0 tot 4095 en de lengte van 1 tot 255. Een regel in een programma zou kunnen zijn:

c:%f(20,2)="aa".

Op de posities 21 en 22 van de filebuffer komt de letter "a" te staan (niet de posities 20 en 21 want de posities tellen vanaf 0).

6. Meestal is het prettig als op een of andere manier het eind van de gegevens in de filebuffer wordt aangegeven. Het handigst is het eind van de gegevens aan te geven met een return. (ASCII-waarde 13). Dus:

   c:%f(e+1,1)=chr(13).

   "e" staat dan voor de laatste relevante positie in filebuffer.
7. Een file met een naam die al op de schijf voorkomt kan niet naar de schijf worden weggeschreven. De meeste operatingsystems laten dat wel toe. De file op de schijf wordt beschouwd als de verouderde versie en daaroverheen wordt a.h.w. de nieuwe versie geschreven. Als de file op de schijf, die wordt herschreven, meteen na het inlezen van de schijf wordt verwijderd kunnen gegevens verloren gaan als er met het programma iets mis gaat. Ook is het niet zo handig steeds na elke verandering (backup) de oude naam van de file te vervangen door een nieuwe naam.

Op de volgende manier voorkomen we beide problemen.

Op de schijf staat de file GEGEVENS. De gegevens in die file worden in het programma bijgewerkt. In het programma komen de volgende regels:

```
fi:GEGEVENS
fx:r:GEGEVENSOUD=GEGEVENS
```

Nu kan in het programma de bewerking op de gegevens plaatsvinden waarna de volgende regels komen:

```
fx:s:GEGEVENSOUD
fo:GEGEVENS
```

Vanuit PILOT is door het veranderen van de naam een z.g. back-up copy van de file gemaakt.

In het nuvolgende programma laten we zien hoe het bijhouden van de score van de ll.'n in een file, die na elk antwoord wordt bijgewerkt, in zijn werk gaat.

In de file komen de volgende gegevens: volledige naam en (om het gemakkelijk te houden) twee getallen van elk twee posities. Deze twee getallen worden door de ll. gekozen. De naam is maximaal 30 posities lang en elke score is een getal van 2 posities. De file op de schijf heet GERADEN.

Om te beginnen moeten we op de schijf het begin van de file zetten die dan elke keer dat het programma gedraaid wordt kan worden bijgewerkt. Omdat de file niet langer kan worden dan 4096 posities en de lengte van de file moet worden bijgehouden reserveren we de eerste 4 posities in de file voor een getal dat de huidige lengte van de file voorstelt. We maken een programmaatje dat deze file aanmaakt. Het programmaatje heet: MAAKFILE en luidt:

```
r:zet het begin van GERADEN op de schijf
c:%f(0,4)-"0005"
c:%f(4,1)=chr(13)
c:%f(5,1)=chr(13)
fo:GERADEN
```

8.8 Record-keeping

Het getal "0005" wordt afgesloten met twee returns. De tweede return zorgt ervoor dat bij het uitlezen van de file in de edit-mode alleen het relevante stuk van de file op het scherm komt.
Eigenlijk is de file niet "5" maar "6" posities lang. Die 6e positie wordt tijdens het bijwerken overschreven.

In het programma zelf moet de teller (1) voor de lengte worden bijgehouden en ook gereageerd worden voor het overschrijden van de lengte (1>4095).

```
    r:programma demonstreert RECORD KEEPING
    d:e$(2)
    d:t$(2)
    d:n$(30)
    c:n$=""
    fi:geraden
    fx:r:geranew=geraden
    t:      We willen weten hoe je heet, dus
    :      tik je voor en achternaam in:
    a:$n$
    t:      O.k. $n$, je mag nu twee getallen
    :      (beneden de 100) kiezen. Het eerste
    :      getal:
    a:$e$
    t:      En dan nu het tweede getal:
    a:$t$
    t:      Prima, dat is alles! Bedankt.
    r:de file bijwerken, eerst de lengte lezen
    c:l=%f(0,4)
    j(l>4095):eind
    c:%f((l+1),30)=n$
    c:%f((l+32),1)=chr(13)
    c:%f((l+33),2)=e$
    c:%f((+35),1)=chr(13)
    c:%f((l+36),2)=t$
    c:%f((+38),1)=chr(13)
    c:%f((+39),1)=chr(13)
    c:l=l+39
    c:%f(0,4)=str(l)
    r:tot slot: oude file verwijderen en de nieuwe
    r:file wegschrijven
    fx:s:GERADNEW
    fo:GERADEN
*eind e:
```

8.8 Record-keeping

Is de file GERADEN vol, dat is dus na ongeveer 100 beurten, dan komt geen foutmelding. Om te voorkomen dat door het intikken van verkeerde getallen fouten ontstaan worden de getallen opgeslagen in de karakterstrings e$ en t$, dit in plaats van numerieke strings. Piet Pietersen, Jan Jansen en Klaas Klaassen en daarna nog een paar ll'n raden de getallen. Met de edit-mode kunnen we zien wie welke getallen ze hebben geraden, nou ja geraden? Roepen we GERADEN op en zetten we de file op het scherm dan ziet het begin van die file er zo uit:

```
239

Piet Pietersen

1
1

Jan Jansen
2
2

Klaas Klaassen
3
3
```

Toelichting: het getal 239 staat dus voor de aktuele file-lengte.

8/8.9

Geluid

In tegenstelling tot computerspelletjes speelt geluid in onderwijsprogramma's meestal maar een zeer ondergeschikte rol. Het is, zeker voor de meeste onderwijsgevenden, niet zo plezierig als in een klaslokaal een paar computers onophoudelijk staan te piepen, kreunen en steunen. Toch kunnen geluidseffecten in onderwijsprogramma's erg functioneel zijn. Daarom wordt in dit hoofdstuk alleen besproken wat beslist nodig is om eenvoudige geluidseffecten te genereren. Voor een gedetailleerde bespreking van de toch niet geringe mogelijkheden van de in de Commodore ingebouwde geluids-chip SID (de 6581 Sound Interface Device), verwijzen we naar hoofdstuk 11 van dit naslagwerk. In dat hoofdstuk wordt de techniek voor het aansturen van de SID uitvoerig besproken. Om dat vanuit PILOT te doen wordt de "v"-(voice)-opcode gebruikt. Die opcode kan vergeleken worden met een POKE-instructie, zoals we die in andere talen (bijvoorbeeld in BASIC) kennen. Met deze instructie kan een willekeurige geheugenlokatie voorzien worden van een bepaalde inhoud. Zo ook vanuit PILOT. Bepaalde geheugenlokaties, we spreken in dit geval liever over registers, krijgen met de "v"-opcode een inhoud waardoor de geluids-chip wordt aangestuurd. Als vanuit BASIC met de POKE-instructie die chip wordt aangestuurd, moet de precieze geheugenlokatie bekend zijn. De geheugenlokaties beginnen, voor geluid dus, op nr. 54272. Vanuit PILOT gaat het om de registers 0 t/m 28. Voor elk geluid dat we willen produceren moeten we er voor zorgen dat de 29 registers de juiste inhoud krijgen. Voor een gewone toon hoeven niet alle 29 registers een bepaalde inhoud te hebben maar kan het overgrote deel daarvan gewoon leeg, dus op "0", blijven.

In het kort iets over de SID.

De SID is een muziek-synthesizer en geluidsgenerator. In de SID bevinden zich 29 8-bit registers. De drie verschillende stemmen corresponderen met de register 0-6 voor de eerste, 7-13 voor de tweede en 14-20 voor de derde stem. Deze drie stemmen kunnen dus onafhankelijk van elkaar worden aangesproken. Daarnaast hebben we nog registers die zorgen voor de koppeling tussen de 3 afzonderlijke stemmen, dit zijn de registers 21,22 en 23. Met dit laatste wordt bedoeld dat de ene stem een bepaalde invloed kan hebben op een andere stem. Register 24 is voor de geluidsterkte van de drie stemmen. Op de registers 25 t/m 28 gaan we hier niet in (zie eventueel deel 11/4.1). Zoals gezegd is elk register 8-bit breed en kan dus een getal 0-255 bevatten.
Voor het vullen van een register is dus de "v"-opcode.

8.9 Geluid

Stel, we willen register 3 vullen met 233 dan gaat dat met:

```
v:3,233
```

De afzonderlijke bit (0-7) worden dan voorzien van: 11101001. (1x128 + 1x64 + 1x32 + 0x16 + 1x8 + 0x4 + 0x2 + 1x1 = 233)

Wat in hoofdstuk 11 wordt besproken over de 3 verschillende stemmen die tegelijkertijd kunnen worden gegenereerd en de begrippen ATTACK, DECAY, SUSTAIN en RELEASE geldt ook voor het aansturen van de geluid-chip vanuit PILOT. Ook wat in dat hoofdstuk gezegd over frequentie, omhullende, filter en resonantietechnieken geldt in beginsel ook voor geluid genereren vanuit PILOT. Voor het gebruik van al die technieken verwijzen we naar dat hoofdstuk. Voordat een bepaald geluid kan worden gegenereerd moeten alle geluidsregisters worden gereset, dus op "0" gezet. Dit gaat als volgt:

```
        r:zet de geluidsregisters op "0"
        c:x=0
*reset  v:x,0
        c:x=x+1
        j(x>30):reset
        e:
```

De "v"-opcode heeft steeds veelvouden van twee argumenten. Het eerste van het tweetal is het registernummer, dat is dus één van de 29 en het tweede geeft de inhoud die in het opgegeven register moet worden gestopt. Na het runnen van het programmaatje hebben alle geluidsregisters dus de inhoud "0". Na een "v"-opcode kunnen meerdere stellen argumenten komen. Elk wordt gescheiden door een ";"-teken. Dus:

```
r:zet de geluidsregisters op "0"
v;1,0;2,0;3,0;4,0;5,0
enz..........
e:
```

doet hetzelfde als het eerste programmaatje.

De SID kan 3 stemmen afzonderlijk weergeven. Voor het instellen van de klankkleur van die stem zijn 7 registers en voor het mengen van die klanken zijn nog eens 3 registers beschikbaar. Beperken we ons tot één enkele stem, de eerste, dan hebben we alleen te maken met de registers 0-6 voor het kiezen van de juiste toonhoogte en klankkleur. Het register 24 voor de geluidsterkte.

De geluidsterkte is het simpelste in te stellen. We kunnen kiezen van 0-15. Een 0 correspondeert met stilte, en 15 met de maximale geluidsterkte. Het geluidsniveau dat uiteindelijk de monitor, T.V. of andere versterker gaat produceren hangt af van de instelling van de zich in dat apparaat bevindende versterker. Zolang de inhoud van register 24 dus niet van inhoud verandert blijft het geluid even hard.
Als voorbeeld een gewone toon met een frequentie van ongeveer 440 Hz.

- de geluidsterkte zetten we op 10, dus:

```
v:24,10
```

- Het eerste bit (de gate) van register 4 moet op "ON" evenals het vierde bit van dit register. Daardoor krijgen we als golfvorm een driehoek (zie hoofdstuk 11/2.1 pag. 1 en 2) dus:

8.9 Geluid

v:4,17

- Voor de toonhoogte moeten de registers 0 en 1 voorzien zijn van 99 en 56 (hoe we aan deze getallen komen blijkt hierna) dus:

v:0,99;1,56

Zolang de ingestelde waarden van de geluidregisters niet veranderen blijft die toon klinken.

- We laten de toon ongeveer 2 sec klinken, dus:

w:30

- Laten we het hierbij dan blijven de registers hun waarden behouden en blijft de toon klinken. We zetten de troon uit door de gate weer op "OFF" te zetten dus:

v:4,16
e:

Voor het instellen van de toonhoogte moeten de registers 0 en 1 een waarde hebben. Van de toon zoeken we in de onderstaande tabel de waarden voor de low- en highbyte (zie eventueel hoofdstuk 9). We zien in die tabel dat het frequentiebereik niet minder dan 7 oktaven omvat. De in de tabel staande waarden van de tonen komt helaas niet precies overeen met de internationaal afgesproken stemming waarbij de "a", waarop gewoonlijk gestemd wordt, overeenkomt met 440 Hz. Ook wijkt de tabel af van de in hoofdstuk 11 staande tabel.

Onderstaande tabel is gebaseerd op de amerikaanse Commodore 64.

	oktaaf							
toon	0	1	2	3	4	5	6	7
c	12-1	24-2	48-4	97-8	195-16	135-33	15-67	30-134
c#	28-1	56-2	112-4	225-8	195-17	134-35	12-71	24-142
d	45-1	90-2	180-4	104-9	209-18	162-37	69-75	139-150
d#	62-1	125-2	251-4	247-9	239-19	223-39	192-79	126-159
e	81-1	163-2	71-5	143-10	31-21	62-42	125-84	250-168
f	102-1	204-2	152-5	48-11	96-22	193-44	131-89	6-179
f#	123-1	246-2	237-5	218-11	181-23	107-47	214-94	172-189
g	145-1	35-3	71-6	143-12	30-25	60-50	121-100	243-200
g#	169-1	83-3	167-6	78-13	156-26	57-53	115-106	230-212
a	195-1	134-3	212-7	24-14	49-28	99-56	199-112	143-225
a#	221-1	187-3	119-7	239-14	223-29	190-59	124-119	248-238
b	250-1	244-3	233-7	210-15	165-31	75-63	151-126	46-253

8.9 Geluid

Op het PILOT-schijfje staat een programma waarmee alle mogelijkheden op het gebied van geluid kunnen worden getest. Het programma zouden we een "sound editor" kunnen noemen. Het moet vanuit PILOT worden aangeroepen onder de naam "SOUND". Na indrukken van de r-toets (RUN) verschijnt het menu op het scherm. De cursor staat reeds op de plaats waar iets moet worden ingetikt. Begonnen wordt met intikken van het stemnummer waarvan de nieuwe registerwaarden worden ingevuld. We kiezen bijvoorbeeld "1". Meteen worden alle aktuele waarden van de geluidsregisters van stem "1" weergegeven. Eén voor één kunnen de registers, eventueel een bepaald bit van een register, worden ingevuld. Dit laatste geldt niet voor volume en resonantie, die waarden gelden steeds voor alle (3) stemmen.
Nadat alle waarden zijn ingevuld komt voor elke stem de vraag "welke stemmen moeten gespeeld worden?" (Voices to play, Y or N). Na het beantwoorden van deze drie vragen horen we een korte tijd het ingestelde geluid.
Meteen daarna staat de cursor weer op de plaats voor de keuze van het stemnummer. Wordt nu een andere stem gekozen dan verschijnen de ingestelde waarden van die nieuw gekozen stem. Om niet steeds na een kleine verandering alle vragen te hoeven afwerken kan de "@ "-toets gebruikt worden. Na indrukken en een RETURN, komen we meteen aan de vragen over de stemmen die moeten worden weergegeven.

Om prettig met de soundgenerator te kunnen werken moet men eigenlijk eerst de functies van de verschillende bit's van de geluidsregisters bestudeerd hebben. Zie hiervoor hoofdstuk 11/1, 11/2 en 11/3.

8/8.10

De routine SYSX

Een programma bestaat meestal uit een paar onderdelen. Dat kunnen afzonderlijke lessen, oefeningen of iets dergelijks zijn. We nemen als voorbeeld een programma dat bestaat uit 2 oefeningen. Met gehele getallen kan geoefend worden in optellen en aftrekken. Prettig is dan als de ll. zonder het programma te moeten afbreken van de ene oefening naar de andere kan springen. Voor dat springen van het ene naar het andere onderdeel kan SYSX-routine worden gebruikt.

Deze routine werkt als volgt: aan het begin van het programma wordt in het veld van een "p-opcode" (problem) door het plaatsen van een "e" de routine geaktiveerd. Het gevolg is dat elke keer dat door de ll. iets is ingetikt eerst wordt nagegaan of soms het "@ "-teken is ingetikt. Is dat inderdaad het geval dan wordt altijd naar de subroutine SYSX gesprongen. De SYSX is een gewone subroutine die begint met de label "*SYSX" en moet eindigen met "e". Het enige bijzondere aan deze subroutine is dat die wordt aangeroepen vanaf het toetsenbord door het intikken van het "@ "-teken. In die subroutine *die natuurlijk ergens in het programma moet voorkomen,* kunnen aanwijzingen staan en akties worden uitgevoerd waar de ll. op dat moment behoefte aan heeft. In die subroutine zou dus het indrukken, na het "@ "-teken, van een functietoets kunnen worden getest met als gevolg een bepaalde aktie. Dit laatste, nogmaals testen van het indrukken van een functietoets, is in ons geval niet nodig omdat we alleen maar naar de andere les willen springen of stoppen.

Het makkelijkst is het programma te laten starten met een (hoofd)menu. In dat menu staan 3 keuzemogelijkheden n.l. de 2 verschillende oefeningen en stoppen. We aktiveren de SYSX aan het begin van het programma. In de SYSX wordt steeds gesprongen naar het (hoofd) menu, althans dat moet ht resultaat zijn als een keer het "@ "-teken is ingetikt. In de routine SYSX kunnen de sprongopdrachten niet zonder meer worden gezet omdat het een **subroutine** is. Na het doorlopen van een subroutine wordt altijd gesprongen naar de regel volgend op de subroutine aanroep. We lossen dit hier als volgt op: in het begin van het programma zetten we een vlag op (c:d=0). In de subroutine zetten we die vlag op "1" en meteen na de ACCEPT testen we de stand van die vlag en springen al naar gelang de stand daarvan wel of niet naar het menu. Ergens op het scherm moet voor de ll. de verwijzing naar het menu staan. We werken deze situatie in het volgende programma uit.

8.10 De routine SYSX

```
A>
        r:voorbeeld van sysx-routine
        p:e
        c:x=10
        d:m$(2)
*menu   c:d=0
        ts:   REKENEN
        :   Je kunt kiezen uit:
        :    1. optellen
        :    2. aftrekken
        :
        :    3. stoppen
        : Wat kies je? Tik een 1, 2 of 3 in.
*kies   as:#p
        t(p=1 ! p=2):      Voor menu tik @ in!!
        j(p=1):opte
        j(p=2):aftr
        j(p=3):einde
        j:menu
*opte   c:a=rnd(x)
        c:b=rnd(x)
        c:c=a+b
        t: Reken uit:
        : #a + #b =
        a:
        j(d=1):menu
        t(%b=str(c)):dat is goed
        t(%b<>str(c)):dat is niet goed
        w:10
        j:opte
*aftr   c:a=rnd(x)
        c:b=rnd(x)
        c:c=a-b
        t: Reken uit:
        : #a - #b =
        a:
        j(d=1):menu
        t(%b=str(c)):dat is goed
        t(%b<>str(c)):dat is niet goed
        w:10
        j:aftr
*SYSX   c:c=1
        e:
*eind   t:tot ziens
        w:10
        e:
```

8.10

Opm. In het programma moet de subroutine SYSX in hoofdletters worden aangegeven.

Een andere situatie waarbij de SYSX bijzonder prettig werkt is het geven van hulp. In feite gaat dat ongeveer precies zo als bovenstaand programma. In de SYSX-subroutine wordt verwezen naar een subroutine waarin hulp wordt aangeboden voor het beantwoorden van een vraag. In dit geval kan de verwijzing naar een subroutine zonder meer in de SYSX worden geplaatst omdat we bij het geven van hulp het programma alleen maar even onderbreken.
Ook deze situatie werken we in een programma uit.

```
A>
        r: tweede voorbeeld met de SYSX
        r: les plaatsnamen
        p:ue
*vra1   t:Van welk land is Brussel de hoofdstad
        c:v=1
        a:
        m:BELGIE
*vra2   t:Van welk land is Amsterdam de hoofdstad
        c:v=2
        a:
        m:HOLLAND
*vra3   t:enz.
        c:v=3
        a:enz
        e:
*hulp1  t:De naam begint met de letters BEL, meer
        t:zeggen we niet. Tik je antwoord in of
        t:geef een RETURN.
        e:
*hulp2  t:Je hebt die naam, die begint met de letter
        t:"H" al vast eens gehoord. Tik je antwoor in
        t:of geef een RETURN.
        e:
*hulp3  t:hulp bij enz. is verder onbekend dus geef
        t: een RETURN
        e:
*SYSX   c:a$="u:hulp"
        c:b$=str(v)
        c:a$=a$ !! b$
        x:a$
        e:
```

8.10 De routine SYSX

In veel programma's is het handig als de ll. even gebruik kan maken van de rekenmogelijkheden van de computer. Ook in dat geval kan de SYSX-subroutine van dienst zijn. Met de SYSX, in samenwerking met de "x"-opcode (execute indirect), kan in het programma een rekenmachientje worden opgeroepen. De ll. kan na het indrukken van het "@ "-teken meteen zijn sommetje intikken.
Wel moeten we dan rekening houden met het feit dat in de antwoordbuffer behalve het sommetje dat moet worden uitgerekend het symbool "@ " zit.
We kunnen daar als volgt rekening mee houden:

```
A>
        r:rekenhulp
        p:e
        d:a$(25)
        d:b$(2)
        c:b$="@ "
        t:programma vraagt data? 1e keer
        a:$a$
        t:programma vraagt data  2e keer
        a:$a$
        w:30
        e:
*SYSX   c:a$="c:d="!!swp(b$)
        x:a$
        t:#d
        w:30
        e:@a
```

In het begin van het programma definiëren we een (hulp)string (b$) met de inhoud: "@ ". In de eerste regel van de SYSX-subroutine wordt met de "swp"-functie het "@ "-teken vervangen door een spatie (de "swp"-functie vervangt in een string het eerste letterteken door het tweede van die string. In dit geval begint de a dus met twee spaties). Op deze manier zijn we het "@ "-teken dat in de antwoordbuffer en daarna in de "a" terecht is gekomen kwijt. In a zit dus alleen het sommetje. In de volgende regel wordt het sommetje met een EXECUTE INDIRECT uitgerekend. Omdat de SYSX een gewone subroutine is wordt daarna in de laatste regel van de SYSX met "e:@ a" terug gesprongen naar de accept.
Aan de loop van het programma is dus niets veranderd.

8/8.11

Operatoren en functies

De volgende operatoren kunnen worden gebruikt:

rekenkundige bewerkingen:

+	optellen
-	aftrekken
*	vermenigvuldigen
/	delen

relationele operatoren:

<	kleiner dan
>	groter dan
=	is gelijk aan
<=	kleiner gelijk aan
>=	groter gelijk aan
< >	ongelijk aan

logische operatoren

&	en
!	of
not	niet

string samenstellen

!!	concatenation

De volgende functies kunnen worden gebruikt:

abs(x)	absolute waarde van x
asc(x$)	ASCII-waarde van het eerste karakter in de string x$
cap(x$)	zet de lettertekens van x$ in hoofdletters
chr(x)	karakter overeenkomstig de waarde in de ASCII-tabel
flo(x$)	numerieke representatie van x$ of, anders gezegd: het eerste getal dat in de x$ voorkomt
ins(x$)	berekent de positie in de antwoordbuffer waar het symbool in x$(1) staat
key(0)	"0" als geen toets is ingedrukt; als wel een toets is ingedrukt de ASCII-waarde van die toets
len(x$)	aktueel aantal posities in x$
rnd(x)	willekeurig getal van 0 tot x-1
rsp(x$)	verwijdert alle spaties uit x$
str(x)	stringrepresentatie van getal x of, anders gezegd: het getal x wordt beschouwd als een karakterstring
swp(x$)	elke keer dat het eerste karakter van x$ voorkomt in de antwoordbuffer wordt dit vervangen door het tweede karakter van x$.

Van een paar functies wordt in onderstaand programma een voorbeeld gegeven.

8.11 Operatoren en functies

```
A>
        r:een paar voorbeelden met functies
        d:a$(2)
        d:b$(20)
        d:c$(1)
        d:d$(1)
        d:e$(3)
        d:f$(12)
        d:g$(1)
*flo    t:          ****** flo-funktie
        c:f$="def345jklm2o"
        c:z=flo(f$)
        t:het eerste getal in f$:  #z
*str    t:          ****** str-funktie
        c:g$=str(z)
        t:g$
*ins    t:          ****** ins-funktie
        c:d$="#"
        t:tik in: &&&#&&&
        a:
        c:e=ins(d$)
        t:op de #e'e positie staat het #-teken
*key    t:          ****** key-funktie
        t:druk na ca. 1 sec een toets in
        c:x=0
*herha  c:x=x+1
        c:y=key(0)
        j(y<>0):verde
        j:herh
*verde  t:geteld is tot #x
        c:c$=chr(y)
        t:daarna is ingedrukt $c$
*swp    t:          ****** swp-funktie
        c:a$="20"
        t:tik in: 1231231230
        a:$b$
        t:in de antwoordbuffer zit:
        t:$b$
        c:b$=swp(a$)
        t:en na de swp(swap)-funktie:
        t:$b$
        a:
        e:
```

```
               ****** flo-funktie
het eerste getal in f$=  345
               ****** str-funktie
3
               ****** ins-funktie
tik in: &&&#&&&
&&&#&&&
op de 4'e positie staat het #-teken
               ****** key-funktie
druk na ca. 1 sec een toets in
geteld is tot: 79
daarna is ingedrukt W
               ****** swp-funktie
tik in: 1231231230
1231231230
in de antwoordbuffer zit:
1231231230
en na de swp(swap)-funktie zit er in:
1031031030
```

8/8.12

Vergelijkingen, voorwaarden

In de taal PASCAL komen if-, for-, repeat- en while-statements voor. Basic kent de if-then(-else) statements. Kent PILOT dergelijke statements niet? Nee, in ieder geval niet zoals we ze kennen in BASIC en PASCAL. In PILOT kan, zoals we dat al een groot aantal malen hebben laten zien, na elke opcode een vergelijking geplaatst worden. Die vergelijking wordt geëvalueerd en al naar gelang de uitslag van die evaluatie volgt wel/niet uitvoeren van de opdracht die door de opcode (met het veld) wordt bepaald. Zonder verder op de syntax in te gaan kunnen we wel vaststellen dat met deze vergelijkingen als voorwaarden even makkelijk als in BASIC en PASCAL voorwaardelijke opdrachten zijn uit te voeren.

In onderstaande programma's komen een paar vergelijkingen voor.

In het eerste voorbeeld wordt gevraagd de afmetingen van een praktikumlokaal te schatten. Het lokaal is 12 m lang en 8 m breed. De lengte moet op 2 meter- en de breedte op 1 m nauwkeurig geschat worden. Als beide schattingen goed zijn moet geantwoord worden: goed. Is een van beide schattingen goed dan moet geantwoord worden: redelijk, probeer het nog eens. Zijn beide antwoorden fout dan: lijkt er niet op, probeer het nog eens.

8.12 Vergelijkingen, voorwaarden

```
       r:schatting van de afmetingen van een lokaal
       r:l=lengte (12), b=breedte (8).
       c:l=12
       c:b=8
       c:t=0
*begin t:Hoe groot schat je lengte en breedte van dit lokaal?
       t:Geef eerst de lengte, dan de breedte.
*nog   th:lengte:
       a:#p
       th:breedte:
       a:#q
       c:t=t+1
       t(abs(l-p)>=2&abs(b-q)>=1):fout, nog een keer, #t'e keer
       j(abs(l-p)>=2&abs(b-q)>=1):nog
       t(not(abs(l-p)>=2!abs(b-q)<=1)):redelijk, nog een keer #t'e keer
       j(not(abs(l-p)>=2!abs(b-q)<=1)):nog
       t(not(abs(l-p)<=2!abs(b-q)>=1)):redelijk, nog een keer #t'e keer
       j(not(abs(l-p)<=2!abs(b-q)>=1)):nog
       t(abs(l-p)<=2&abs(b-q)<=1):goed, #t'e keer
       t:de werkelijke lengte is: 12 m lang en 8 m breed
       a:
       j:begin
       e:

A>
```

```
Hoe groot schat je lengte en breedte van dit lokaal?
Geef eerst de lengte, dan de breedte.
lengte: 15
breedte: 10
fout, nog een keer, 1'e keer
lengte: 13
breedte: 10
redelijk, nog een keer 2'e keer
lengte: 13
breedte: 9
goed, 3'e keer
de werkelijke lengte is: 12 m lang en 8 m breed
```

8.12 Vergelijkingen, voorwaarden

Opmerking: in bovenstaand voorbeeld slokken de vergelijkingen nog al wat van de 40 posities van de programmaregels op.
Is voor de tekst niet meer genoeg ruimte over dan kan even een variabele worden ingevoerd. Die variabele wordt dan op "1" gezet bij de evaluatie van de vergelijking. Vervolgens wordt allen de variabele getest.

Dus:

```
c:e-0
c<----------vergelijking----------->:e=1
t(e=1):tekst (bijna een hele regel)
```

Net zo makkelijk als numerieke variabelen met elkaar kunnen worden vergeleken kan dat ook met karakterstrings. In onderstaand programma komt in één enkele vergelijking, die we hier beter conditioner kunnen noemen, zowel een numerieke- als een karaktervariabele voor. De ll. wordt gevraagd een bepaalde naam en een daarbij behorende getalswaarde in te tikken. Denk bijvoorbeeld aan de naam van een provincie en het inwoneraantal.
Er zijn op elk vakgebied legio andere voorbeelden te bedenken bijvoorbeeld: naam-leeftijd, naam van element-soortelijke massa, naam van element-valentiewaarde. Ook is heel makkelijk in de vergelijking voor elk getal een foutenmarge in te bouwen zoals dat in onderstaand programma wordt gedaan. Door het invoeren van een teller kan de ll. zelf kiezen welke provincies, omdat daarvan het inwoneraantal bekend is, hij kiest. In het programma staat AA,BB,CC en DD dus voor de namen van de provincies, de getallen 800,700,600 en 500 voor inwonertallen en de getallen 100,80,60 en 40 staan voor de toegestane foutenmarge in de inwonertallen.

```
        r:provincies met inwonertallen
        p:us
        d:p$(20)
        c:t=0
        t:geef de naam van een provincie en daarna
        :het inwoneraantal
*nogeen th:naam provincie:
        a:$p$
        th:aantal inwoners in duizendtallen:
        a:
        t(p$="AA" & abs(%b-800)<100):goed, aa heeft 800 inwoners
        j(p$="AA" & abs(%b-800)<100):nogeen
        t(p$="BB" & abs(%b-700)<100):goed, bb heeft 700 inwoners
        j(p$="BB" & abs(%b-700)<100):nogeen
        t(p$="CC" & abs(%b-600)<100):goed, cc heeft 600 inwoners
        j(p$="CC" & abs(%b-600)<100):nogeen
        t(p$="DD" & abs(%b-500)<100):goed, dd heeft 500 inwoners
        j(p$="DD" & abs(%b-500)<100):nogeen
        r:enz
        c:t=t+1
        t(t<6):niet goed
        j(t<6):nogeen
*eind   e:
```

8/9

BASIC 7.0

Inhoud

8/9.1 Inleiding
8/9.2 Disk commando's
8/9.3 Geluid
8/9.4 Grafiek

8/9.1

Inleiding

De Commodore 128 bestaat uit drie computers in één behuizing. De C128 kan gebruikt worden als Commodore 64, als CP/M machine met Z80 processor en, uiteraard, ook als C 128. Elk van de drie computers heeft zijn eigen operating systeem en z'n eigen BASIC. Hieronder gaat het over de C 128 (dus niet de C64 of de CP/M machine) en het hierbij horende 16 K operating systeem en 32 K BASIC.
Als men de computer aan zet, verkeert deze altijd 'vanzelf' in de '128-modus'. Dat wil zeggen dat na het inschakelen de machine werkt als een C128. Om het apparaat te laten werken als een C64 of een CP/M apparaat, moet men nog een opdracht geven.
Hoeveel tekens op een regel op het beeldscherm passen, hangt ervan af of de toets op het toetsenbord met de aanduiding '40/80 display' is ingedrukt bij het aanzetten van de computer. De computer laat echter als alles in orde is zulks weten door op het beeldscherm te melden dat BASIC 7.0 beschikbaar is, met ongeveer 120 K geheugen. Die 120 K is een beetje opschepperij, want een 8-bits processor, die in de C128 zit, kan niet zoveel geheugen in een keer adresseren. Er is wel een geheugen van ongeveer 120 K aanwezig, maar dit bestaat uit een aantal stukken. Voor het werken met variabelen heeft men 63 K ter beschikking.
In dit hoofdstuk worden de instructies, commando's en functies die Commodore BASIC kent, kort besproken. De met een * gemerkte statements zijn BASIC 7.0 opdrachten, en kunnen dus alleen gebruikt worden als de computer in C 128-modus werkt. De niet gemerkte opdrachten behoren tot het repertoire van BASIC 2.0 en van BASIC 7.0. Deze opdrachten kunnen dus zowel in C64 modus als in C128-modus gebruikt worden (niet in CP/M).
Behalve al deze opdrachten zijn er ook nog instructies die speciaal aan BASIC 7.0 zijn toegevoegd voor het maken van grafische afbeeldingen. Deze grafische instructies, die alleen gebruikt kunnen worden in de modus met 40 tekens per regel, worden in een afzonderlijk hoofdstuk besproken.
Ook zijn er nog instructies voor het produceren van geluid en voor het bewaren, wissen en opvragen van gegevens op datasette of floppy disk. Ook deze worden beschreven in afzonderlijke hoofdstukken.

Functie ABS(X).

geeft, net als in BASIC 2.0, de absolute waarde van het getal X. Als X groter of gelijk aan nul is, is ABS(X) gelijk aan X. Is X kleiner dan nul, dan is ABS(X) gelijk aan X, maar dan zonder '−'-teken.

Functie AND

Wijkt ook niet af van dezelfde operator in BASIC 2.0 (zie hoofdstuk 7/12.2).

9.1 Inleiding

De syntax luidt:
<uitdrukking> AND <uitdrukking>. Het resultaat is WAAR als beide uitdrukkingen WAAR zijn. De beide uitdrukkingen kunnen we ook beschouwen als voorwaarden: zijn beide voorwaarden tegelijk WAAR, dan is de hele uitdrukking WAAR.

Functie ASC('%')

is een stringfunctie, om het codegetal waarmee de computer intern werkt te bepalen van de string X. Van een string X wordt het bij X horende codegetal bepaald. IS de string langer dan één teken, dan wordt slechts het codegetal van het eerste teken gegeven. IS de string 'leeg', dat wil zeggen dat er geen enkel teken aanwezig is dan verschijnt de foutmelding ?ILLEGAL QUANTITY ERROR'.

Functie ATN(%)

wijkt ook al niet af van de ATN-functie in BASIC 2.0. Van X wordt de arctangens berekend. De uitkomst is uitgedrukt in radialen, en ligt tussen −PI/2 en +PI/2.

* Commando AUTO X

schakelt automatische regelnummering in. X is de stapgrootte. Nogmaals AUTO intypen doet de regelnummering uitschakelen.

* Commando BANK X

selecteert één van de 64 K geheugenbanken.

* Instructie BEGIN <deel van programma> BEND

is een uitbreiding van de IF ...THEN-instructie. Achter THEN in de IF...THEN mag slechts één programmaregel instructies staan. Bij BEGIN...BEND zoveel als men wil.
Alles wat tussen BEGIN...BEND staat, wordt uitgevoerd alsof het een opdracht is.

* Commando BOOT'programmanaam',DX,UY

laadt het genoemde binair programma en voert het uit. Y is het nummer van het randapparaat vanwaar het programma gelezen moet worden (datasette of diskdrive), X het nummer van de drive (0 of 1).

Functie CHR$(X)

is de tegenhanger van de ASC−functie. X is een codegetal, waarmee de computer intern tekens van het toetsenbord codeert. X heeft een waarde van en met 0 tot en met 255, en CHR$(X) levert het teken dat hoort bij dit getal.

Instructie CLOSE X

Sluit de file met filenummer X. CLOSE is de tegenhanger van OPEN.

Instructie CLR.

is de opdracht om alle variabelen en arrays en de door u zelf gedefinieerde functies te wissen. Het BASIC-programma zelf wordt niet gewist.

Instructie CMD X

stuurt uitvoer naar een randapparaat. Normaal gaat uitvoer naar het beeldscherm. Om uitvoer naar bijvoorbeeld printer of diskdrive te sturen, dient de CMD-opdracht. X is het nummer van het kanaal naar randapparaat; dit dient met OPEN eerst geopend te worden.

9.1 Inleiding

Commando CONT

Doet een programma dat is gestopt doordat op de STOP-toets werd gedrukt, of omdat een STOP- of een END-instructie in het programma werd opgenomen, weer verder gaan. CONT werkt niet om een programma dat stuit op een fout, verder te doen gaan: na een foutmelding stopt het programma definitief, en kan dan slechts van voren af aan worden gestart.

Instructie DATA X,Y

Deel van de opdracht READ DATA. Zie READ.

Instructie DEF FN F(X)=<uitdrukking>

Met deze instructie kan men zelf een wiskundige functie definiëren. Later in het programma hoeft dan niet de hele uitdrukking opnieuw te worden opgenomen, maar kan worden volstaan met de functienaam F (en een waarde voor de variabele X).

* Functie DEC(X)

zet het hexadecimale getal X om in hetzelfde getal in het decimale stelsel.

* Commando DELETE <regelnummer 1> – <regelnummer 2>

wist regels in het BASIC-programma. Syntax is identiek aan die van LIST (zie aldaar).

Instructie DIM A(X)

definieert de grootte van een array. X is de index van het array. Meer dan een index is mogelijk, bijvoorbeeld A(X, Y, Z).

* Instructie DO/LOOP/WHILE/UNTIL/EXIT

is een BASIC-instructie waarmee men in hoge mate zijn programma structuur kan geven. DO geeft het begin van een lus, LOOP het eind. De lus kan beëindigd worden als een voorwaarde WAAR is gemaakt door de lus zelf (UNTIL), maar ook kan de lus zolang herhaald worden (WHILE) als een bepaalde voorwaarde WAAR is (en in de lus ONWAAR wordt gemaakt).
Met EXIT kan de lus worden verlaten; wordt EXIT uitgevoerd dan wordt de instructie direct achter LOOP staat, uitgevoerd.

Instructie END

breekt de uitvoering van het programma af. Op het beeldscherm verschijnt de melding 'READY'. Programma kan worden voortgezet met CONT

* Instructie ERR$(X)

geeft een beschrijving van een fout (Foutmelding). In combinatie met andere instructies zoals TRAP en RESUME kan zo een nauwkeurige foutmelding worden verkregen.

Voorbeeld:

```
10 FOR FOUT= 1 to 41
20      PRINT ERR$ (FOUT)
30 NEXT 1
```

levert alle mogelijke foutstrings.

Functie EXF(X)

berekent de X-de macht van het getal e. X mag niet groter dan 88.0296919 zijn.

9.1 Inleiding

Instructie FOR X = Y TO Z STEP S

maakt een lus, met X als teller. Beginwaarde van de lus is X, eindwaarde is Z, stapgrootte is S. Als het woord STEP niet wordt vermeld, is de stapgrootte 1. Het einde van de lus wordt aangeduid met NEXT.

Functie FRE(X)

geeft aan hoeveel bytes nog beschikbaar zijn in het geheugen voor BASIC. X is een willekeurig getal; het maakt niet uit wat men hier invult.

Instructie GET X$

leest tekens die via het toetsenbord worden medegedeeld aan de computer één voor één en kent ze toe aan de variabele X$. Het is ook mogelijk om met GET een getal van het toetsenbord te vragen, bijvoorbeeld via GET A of GET B%.
Als dan een string wordt ingetypt, geeft de computer een foutmelding: 'SYNTAX ERROR'. Het beste is het daarom bij de GET-instructie altijd een string-variabele te gebruiken.

Instructie GET# A, X$

is praktisch hetzelfde als de GET-instructie, maar in plaats dat er van het toetsenbord wordt gelezen, wordt nu van een randapparaat een teken gelezen. Het kanaalnummer van het randapparaat is A.

* Instructie GETKEY A$

doet uitvoering van het programma wachten totdat er een toets wordt ingedrukt. Er hoeft bij GETKEY$ dus niet een wachtlus te worden geprogrammeerd zoals bij GET.

* Commando GO 64

schakelt computer om van 128-modus naar 64-modus.

Instructie GOSUB XX

doet het programma een sprong maken naar de subroutine met regelnummer XX. Het eind van de subroutine is aangeduid met RETURN; zodra het programma bij deze instructie is aangekomen, wordt teruggekeerd naar het hoofdprogramma.

Instructie GOTO XX

doet het programma een sprong maken naar regelnummer XX. Programma voert de opdracht op regel XX uit, en gaat dan verder naar regel met nummer XX+1, XX+2 enzovoorts.

* Commando HELP

wordt gebruikt na het optreden van een foutmelding. Intypen van het commando heeft tot gevolg dat de programmaregel wordt getoond waarin fout is opgetreden plus de vermoedelijke plaats in de regel waar de fout huist.

* Instructie HEX$(X)

zet decimale getal X om in hetzelfde getal in het hexadecimale stelsel. X ligt tussen 0 tot en met 65535.

Instructie IF <voorwaarde> THEN <aopdracht>.

voert de opdracht achter THEN uit, als de voorwaarde waar is.

9.1 Inleiding

Is voorwaarde niet waar, dan gaat het programma gewoon verder met het volgende regelnummer.

* Instructie IF <voorwaarde> THEN <programmadeel 1> : ELSE <programmadeel 2>

is een uitbreiding van de IF...THEN... constructie uit BASIC 2.0, met een ELSE-gedeelte. Als de voorwaarde WAAR is wordt programmadeel1 uitgevoerd. Is de voorwaarde ONWAAR dan wordt programmadeel2 uitgevoerd. Aan het einde van programmadeel1 en voor ELSE moet een dubbele punt staan.

Instructie INPUT"tekst";X$

leest invoer van het toetsenbord.
Als het programma op een INPUT-opdracht stuit, verschijnt een vraagteken op het beeldscherm. Hiervan kan gebruik worden gemaakt door achter het woord INPUT een tekst (bijvoorbeeld een vraag) op te nemen.

Instructie INPUT# A,X$

Bijna hetzelfde als gewone INPUT-opdracht, maar hier worden gegevens ingevoerd vanaf een randapparaat. X$ is maximaal 80 tekens lang. Een komma of een RETURN wordt aangemerkt als het einde van een string.

* Functie INSTR(string1, string2,X)

geeft de positie aan van string 2 in string 1. Komt string 2 niet voor in string 1 dan is de uitkomst nul. Met X kan (hoeft niet) worden aangegeven vanaf het hoeveelste teken, van links af gerekend, string 1 moet worden doorzocht.

Functie INT(X)

rondt de waarde van getal X naar beneden af, op het dichtsbijzijnde gehele getal.

* functie JOY(X)

leest de toestand van de joystick-poort en wordt gebruikt bij het programmeren van spelletjes. X=1 doet het programma poort 1 lezen, X=2 de tweede poort.
De mogelijk toestanden zijn:

0 = neutraal	5 = onder
1 = boven	6 = linksonder
3 = rechts	8 = linksboven
4 = rechtsonder	128 = vuurknop

Waarden groter dan 128 geven aan dat de vuurknop wordt gebruikt in combinatie met een andere functie (bijvoorbeeld 132 = 128 + 4, dat wil zeggen dat vuurknop wordt gebruikt en joystick rechtsonder staat)

* Commando KEY
of KEY X, 'functie'

heeft twee verschillende mogelijkheden. Typt men alleen KEY, dan verschijnt op het beeldscherm een lijstje van de functies van de acht functietoetsen van de C128. Zet men achter KEY een nummer tussen 1 en 8 dan kan de corresponderende functietoets een andere functie gegeven worden.

Voorbeeld:

KEY 2, CHR$(27)+'X' heeft als resultaat dat onder functietoets 2 de functie 'ESC X' komt, zodat als men op F2 drukt het beeldscherm wordt overgezet van 40 naar 80 tekens of omgekeerd.

Functie LEFT$(X$,A)

9.1 Inleiding

is een stringfunctie. Levert het linkergedeelte van string X$, vanaf teken nummer een tot en met teken nummer A. A ligt tussen 1 en 255; is dit niet het geval, dan verkrijgt men de melding 'ILLEGAL QUANTITY ERROR'.

Functie LEN(X$)

telt het aantal tekens in string X$, inclusief eventuele spaties.

Instructie LET

dient om een waarde toe te kennen aan een variabele, bijvoorbeeld LET X= 5. Ook is toegestaan: X= 5, zonder LET.

Commando LIST <regelnummer1> – <regelnummer2>

doet het programma dat aanwezig is in het geheugen verschijnen op het beeldscherm, vanaf regelnummer1 tot en met regelnummer2. Worden geen regelnummers opgegeven, dan wordt het hele programma getoond. Indien voorafgegaan aan LIST een CMD-instructie wordt gegeven, wordt het programma 'gelist' op de printer of op floppydisk.
Om een listing op de printer te maken typt men het volgende in:
OPEN 4,4 : CMD 4 : LIST : CLOSE 4

Commando LOAD"programmanaam",A,B

laadt een programma met de aangegeven naam van datasette of diskdrive. Wordt geen naam opgegeven, dan wordt het eerstvolgende programma dat wordt aangetroffen op datasette, geladen. A is het nummer van het randapparaat
(8 als van diskdrive moet worden gelezen). B is het getal 1, dat slechts hoeft te worden vermeld als het beslist noodzakelijk is dat het programma weer op precies dezelfde plaats in het geheugen terecht komt als het eerder heeft gestaan.

Functie LOG (X)

geeft de logaritme met grondtal e van X. X moet groter zijn dan nul.

Functie MID$(X$,A,B)

levert een deel van de string X$, en wel het deel beginnend met het A-de teken en eindigend B plaatsen verderop. A en B moeten kleiner dan 255 zijn.

* Functie MID$ (X$,A,B)

levert nog steeds een deel van de string X$, en wel het deel beginnend met het A-de teken en eindigend B plaatsen verderop. A en B moeten kleiner dan 255 zijn. Maar in BASIC 7.0 is een uitbreiding aan deze functie gegeven: men kan nu ook tekens toekennen aan een string.

Voorbeeld:

```
10 X$="EEN NIEUWE FIETS"
20 MID$(X$,5,6)="OUDE  "
30 PRINT X$
```

levert als resultaat:

EEN OUDE FIETS

* Commando MONITOR

schakelt de in de C 128 ingebouwde monitor voor machinetaal in. Zit men hier in dan keert men terug naar BASIC 7.0 met het commando X.

Commando NEW

9.1 Inleiding

wist de variabelen in het geheugen en het aanwezige programma.

NEXT

is onderdeel van FOR-lus. Zie aldaar.

Functie NOT <uitdrukking>

keert de logische waarde van de uitdrukking om. Als de uitdrukking WAAR is, is NOT <uitdrukking> onwaar, en omgekeerd.

Instructie ON X GOTO A,B
of ON X GOSUB A,B

doet het programma een sprong maken naar een deel van de aangegeven regels (A of B) of subroutines. Als X=1 springt het programma naar A, als X=2 wordt naar B gesprongen, enzovoorts.

Instructie OPEN X,A,B

opent een kanaal naar een randapparaat. Dit kanaal dient voor het uitwisselen van data. X is het nummer van de file die wordt uitgewisseld en is een getal kleiner dan 127. A is het nummer van het randapparaat (1 = datasette, 3 = beeldscherm, 4 = printer, 8 = diskdrive). B is het secundair adres. De betekenis van dit getal verschilt per randapparaat. Bij een datasette staat een '0' voor lezen, '1' voor schrijven.

Functie <voorwaarde> OR <voorwaarde>

levert het resultaat WAAR als een van beide voorwaarden WAAR is.

Functie PEEK(X)

laat zien wat de inhoud is van geheugenplaats X. X is een getal van 0 tot en met 65535. PEEK(X) is een getal van 0 tot en met 255.

* Functie PEN(X)

levert de positie van de lichtpen. PEN(0) levert de horizontale positie van de lichtpen. PEN(1) de verticale.

Instructie POKE X,A

schrijft het getal A in geheugenplaats X. X is een getal van 0 tot en met 65535. A is een getal van 0 tot en met 255.

Functie POS(A)

geeft aan in welke kolom de cursor staat op het beeldscherm. A is een willekeurig getal; het maakt niet uit wat men hier invult.

* Functie POT(X)

geeft de waarde van de paddles. X geeft de positie van de X-de paddle aan.

Instructie PRINT A
of PRINT"tekst"
of PRINT<uitdrukking>

print variabelen, tekst of berekeningen op het beeldscherm. Wordt een variabele geprint, dan verschijnt op het beeldscherm de waarde die de variabele op dat moment heeft. Wordt tekst geprint die tussen aanhalingestekens moet worden geplaatst, dan verschijnt op het beeldscherm de tekst zonder aanhalingstekens.
Wordt een berekening geprint, dan verschijnt op het beeldscherm de uitkomst.

Instructie PRINT# X,<data>

9.1 Inleiding

schrijft gegevens naar een bestand met nummer X. Het bestand moet vooraf geopend zijn (Zie opdracht OPEN).

* Instructie PRINT USING"formaat";A

dient voor nette uitvoer van variabelen. Aanwijzingen voor het formaat staan tussen aanhalingstekens. A is de variabele die netjes geprint moet worden. Het formaat wordt bepaald met de volgende tekens:

#	= aantal tekens
+	= levert + voor een positief getal
-	= levert - voor een negatief getal
$	= wordt gebruikt voor valuta
.	= duidt positie van decimale punt in getal aan
↑ ↑ ↑ ↑	= uitvoer in wetenschappelijke notatie
=	= uitvoer centreren
>	= uitvoer rechts uitlijnen

* Instructie PUDEF"tekens"

verandert tekens die werden vastgelegd met een PRINT USING-instructie achteraf. Tussen aanhalingstekens staan de vier 4 posities die maximaal kunnen worden gewijzigd. Eerste definieert de spatie, de tweede de komma, de derde de decimale punt en de vierde het $-teken.

* Functie RCLR (X)

levert de kleur in het kleurengeheugen. Resultaat van RCLR(X) is een getal tussen 1 en 16 (de kleurcodes). X heeft als mogelijke waarden:
0 = achtergrondkleur in 40-tekens modus
1 = kleur van tekens in grafische modus
2 = 1ste kleur van tekens in grafische multicolor modus
3 = 2de kleur van tekens in grafische multicolor modus
4 = kleur van rand in 40-tekens modus
5 = schrijfkleur in 40-tekens of 80-tekens modus
6 = achtergrondkleur in 80-tekens modus

* RDOT(X)

levert de positie van de (onzichtbare) grafische cursor en de waarde van het kleurengeheugen op de plaats van deze cursor. RDDT(0) geeft de horizontale positie van de cursor.
RDOT(1) de verticale positie, RDOT(2) geeft de inhoud van het kleurengeheugen, waarbij 0 de achtergrondkleur aanduidt (bit niet gezet) en 1 de voorgrondkleur betekent (bit is gezet).

Instructie READ X

leest een element uit een DATA-regel en kent het toe aan variabele X.

Instructie REM <tekst>

is bedoeld om commentaar voor de menselijke lezer, en niet voor de computer, op te kunnen nemen in programma's. Als het programma is aangekomen bij de REM-instructie worde de hele tekst achter het woord REM genegeerd.

Commando RENUMBER X,Y,Z

doet de regelnummers van het programma opnieuw nummeren. X is het nieuwe regelnummer waarmee begonnen moet worden met hernummeren, Y is de stapgrootte waarmee genummerd moet worden (wordt geen waarde opgegeven dan is de stapgrootte 10), en Z is het oude nummer van de regel waarmee begonnen moet

9.1 Inleiding

worden met hernummer.

Instructie RESTORE

komt voor in combinatie met de READ-instructie. Normaal worden bij een READ-instructie de DATA-elementen gelezen in de volgorde zoals ze in de DATA-regels staan. Na een RESTORE-instructie begint het programma weer bij het eerste element; DATA-elementen kunnen zo meer dan één keer worden gelezen.

* Instructie RESUME
of RESUME X
of RESUME NEXT

dient voor foutafhandeling. De instructie kan alleen gebruikt worden in combinatie met de instructie TRAP.
RESUME doet uitvoering van een programma continueren nadat een foutmelding is verschenen. RESUME zonder regelnummer of NEXT is opdracht om de regel waarin de fout is geconstateerd opnieuw uit te voeren. RESUME met regelnummer zet het programma voort vanaf de regel met het aangegeven nummer. RESUME NEXT zet het programma voort vanaf de instructie direct na de opdracht die de foutmelding heeft veroorzaakt.
Aan het begin van het programmagedeelte waarin met een fout vermoedt zet men de instructie TRAP, gevolgd door een regelnummer. In de regel met dit nummer (veelal aan het eind van het programmadeel) zet men dan een RESUME-instructie; een foutmelding in het programmagedeelte zelf onderbreekt de uitvoering van het programma dan niet.

Instructie RETURN

geeft het einde van een subroutine aan. Na het aantreffen van RETURN, springt het programma terug naar de regel met GOSUB.

* Functie RGR(X)

levert een getal dat aangeeft in welke grafische modus (toestand) het computersysteem op dit moment verkeert.

* Instructie RREG(A,X,Y,S)

levert de waarden van de (A)ccu, het x- en hety-register en het (S)tatus-register. Met PRINT A,X,Y,S krijgt men deze waarden vervolgens op het beeldscherm te zien.

Instructie RIGHT$(X$,A)

geeft het rechtergedeelte van X$, waarbij A aangeeft hoeveel tekens van het rechtergedeelte worden weergegeven.
IS a groter dan het aantal tekens van X$ dan wordt X$ in zijn geheel weergegeven.

Functie RND(X)

genereert een willekeurig getal tussen 0 en 1. Is X kleiner dan nul, dan is RND(X) steeds hetzelfde getal. Is x groter of gelijk nul dan ontstaat telkens een ander getal.

Commando RUN

start de uitvoering van een programma. Achter RUN kan een regelnummer worden aangeduid; dan start het programma bij deze regel.

* Instructie RWINDOW(X)

levert breedte, hoogte en modus (40/80)

9.1 Inleiding

van een beeldscherm. Als X=0 wordt ingevuld krijgt men de breedte, voor X=1 de hoogte en X=2 de modus.

Commando SAVE"programmanaam",A

schrijft aangegeven programma weg naar datasette of diskdrive. A is het nummer van het randapparaat. Wordt geen waarde opgegeven dan wordt automatisch naar datasette geschreven.

Functie SGN(X)

geeft het teken van X weer. Als X groter dan nul is is SGN(X) gelijk aan 1. IS X nul, dan is SGN(X) ook nul. IS X kleiner dan nul dan is SGN(X)=-1.

Functie SIN(X)

geeft de sinus van X. X is in radialen, de uitkomst ligt tussen -1 en +1.

* Instructie SLEEP X

maakt een pauze, een wachtlus, in de uitvoering van het programma van X seconden. X is een getal tussen 1 en 65535.

Functie SPC(X)

doet de cursor X plaatsen overslaan vanaf de huidige positie. X ligt tussen 0 en 255.

Functie SQR(X)

berekent de (vierkants)wortel uit X. X moet groter of gelijk nul zijn.

Instructie STOP

onderbreekt de uitvoering van een programma. Op het scherm verschijnt de melding BREAK IN X, waarin X het regelnummer is waar het programma werd onderbroken. Met CONT kan uitvoering vervolgd worden.

Functie STR$(X)

maakt van getal X een string. Is X groter of gelijk nul dan is het eerste teken van de string een spatie.

* Instructie SYS B,A,X,Y,S

geeft vanaf adres B waarden A,X,Y,S door aan accu, X-, Y- en Statusregister.

Instructie SYS X

start een machinetal-programma dat begint op geheugenplaats X. X ligt tussen 0 en 65535.

Functie TAB(X)

wordt gebruikt in combinatie met PRINT en dient om getallen of tekst op een vaste plaats op het papier uit te laten printen. X is een getal tussen 0 en 255 dat de kolom aangeeft waar iets moet worden geprint, ten opzichte van de linkerkantlijn van het papier.

Functie TAN(X)

geeft de tangens van X. X moet worden opgegeven in radialen.

* Instructie TRAP X

wordt gebruikt in combinatie met instructie RESUME. X is het regelnummer waar de RESUME-instructie staat. Zie RESUME.

* Commando TRON

9.1 Inleiding

laat bij uitvoering van het programma zien welke regel op dit moment aan de beurt is, door het regelnummer te tonen op het beeldscherm.

* Commando TROFF

zet het commando TRON uit.

Functie USR(X)

start een machinetaal-programma vanaf een adres dat eerder werd geplaatst in de geheugenslocaties 785 en 786. X wordt doorgegeven aan het programma als parameter. Het resultaat wordt door het machinetaal-programma teruggegeven (aan BASIC).

Functie VAL(X$)

zet string X$, van links naar rechts, om in een getal.

Commando VERIFY"programmanaam",A

vergelijkt het genoemde programma dat zojuist werd weggeschreven op datasette of floppydisk met de inhoud van het computergeheugen. A is het nummer van het randapparaat.

Instructie WAIT X,Y

doet uitvoering van het programma pauzeren, totdat de inhoud van geheugenadres X de waarde Y heeft.

* Instructie WINDOW A,B,C,D,E

maakt op het beeldscherm een venster, met hoekpunten:
kolom linksboven A,
regel linksboven B,
kolom rechtsonder C,
regel rechtsboven D

E hoeft niet te worden opgegeven. Als E niet wordt ingevuld, wordt het venster niet automatisch gewist. Is E =1 dan wordt het venster automatisch gewist.

Functie XOR(A,B)

levert de logische waarde van A XOR B. A en B zijn getallen tussen 0 en 65535.

8/9.2

Disk commando's

De Commodore 128 beschikt over enkele zeer krachtige instructies voor het manipuleren van bestanden. Veel van deze opdrachten worden zeer dikwijls gebruikt, vooral als men een computer heeft met een diskdrive.

Instructie APPEND# X, "naam", Y ON Z
open een bestand met file-nummer X. Het bestand wordt echter niet zoals bij de instructie OPEN aan het begin geopend, maar aan het eind. APPEND kan daardoor gebruikt worden om data toe te voegen aan een reeds bestaand bestand. Y is een aanduiding (D0 of D1) dat aangeeft in welke diskdrive het bestand geopend moet worden. Heeft men maar één drive dan hoeft geen waarde voor Y opgegeven te worden.
Eventueel kan ook nog een apparaatnummer Z worden opgegeven, door het nummer van het apparaat te specificeren achter de letter U (bijvoorbeeld U8).

Commando BACKUP X TO Y, ON Z
kopieert de volledige inhoud van een floppydisk naar een tweede diskette. BACKUP kan alleen gebruikt worden bij apparaten met twee ingebouwde diskdrives.
Z is ook hier het apparaatnummer, dat begint met de letter U gevolgd door een getal. X en Y duiden de diskdrives aan (D0 of D1).

Commando BLOAD"naam" ON X,Y
dient voor het laden van een binair bestand vanaf floppy in het geheugen, op het startadres Y. X duidt aan in welke bank van het geheugen het bestand moet worden geladen. X is de letter B (van bank) gevolgd door het nummer van de bank. Y is de letter P gevolgd door een geheugenadres. Het woorde ON mag worden weggelaten; in plaats hiervan moet dan een komma worden geschreven.

Commando BSAVE"naam",X,Y TO Z
schrijft een programma als binair bestand op floppy, X is het bank-nummer (de letter B gevolgd door een 1), Y het beginadres van het programma in het geheugen (de letter P gevolgd door het geheugenadres) en Z is het eindadres (de letter P gevolgd door een getal).

Commando CATALOG X,Y,"filenaam"
laat de directory (inhoudsopgave) van een floppydisk op het beeldscherm zien. Als alleen het woord CATALOG wordt ingetypt krijgt men de hele directory. Bij een C128 met dubbele diskdrive krijgt men na het intypen van alleen maar CATALOG de inhoudsopgave van beide floppies. X specificeert het drive-nummer (letter D gevolgd door een nul of een).

9.2 Disk commando's

Y is het apparaatnummer (letter U gevolgd door nummer). Men kan ook de filenaam opgeven (eventueel met gebruik van "wild-cards" als * en ?). Bijvoorbeeld CATALOG"PROG*" laat alle files zien waarvan de naam begint met PROG.

Commando COLLECT X ON Y
wist alle beschreven blokken die niet tot een file behoren. Ook niet gesloten files worden gewist.
X is het drive-nummer (letter D plus getal: 0 of 1), Y is apparaat-nummer (letter U plus getal).

Commando CONCAT X,"naam1" TO Y, "naam2" ON Z
maakt van twee bestanden 1 bestand, door het bestand "naam1" achter het bestand dat de naam "naam2" heeft te plakken. X en Y zijn weer het drive-nummer (letter D gevolgd door 0 of 1). Z is het apparaat-nummer (letter U gevolgd door getal).

Commando COPY X,"naam1" TO Y, "naam2", Z
kopieert een bestand op de ene drive (drive X) naar een floppy in de andere drive (drive Y). Z is weer het apparaatnummer (letter U plus getal).
X en Y kunnen ook gelijk zijn: dan wordt een kopie van een file gemaakt op dezelfde floppy, maar met een andere naam.

Commando DCLEAR X ON Y
is de opdracht om alle geopende kanalen naar de drive te sluiten.
X is het drive-nummer (letter D plus getal 0 of 1), Y is apparaat-nummer (letter U plus getal).

Commando DCLOSE# X ON Y
sluit een file. X is het file-nummer, Y het apparaat-nummer (letter U plus getal).
DCLOSE# 1 sluit file nummer 1, DCLOSE zonder meer sluit alle files die op dit moment geopend zijn.

Commando DIRECTORY X,Y,"filenaam"
laat de directory (inhoudsopgave) van een floppy zien op het beeldscherm. X is het drive-nummer (letter D gevolgd door 0 of 1). Y is het apparaat-nummer (bijvoorbeeld U8). Bij het specificeren van een file-naam, kan gebruik worden gemaakt van "wildcards": * en ?.
DIRECTORY verschilt niet van CATALOG.

Commando DLOAD"naam", X,Y
laadt een programma van floppy in het geheugen van de computer. X is weer het drive-nummer, Y het apparaatnummer.
In plaats van een naam kan ook een variabele opgegeven worden, bijvoorbeeld DLOAD X$ laadt het programma met de naam die is toegekend aan variabele X$.

Commando DOPEN# X, "naam",Y,Z,A,B
opent file, om te lezen of om naar toe te schrijven. Gaat het om een relatieve file dan moet de lengte van het record worden gespecificeerd in A. Gaat het niet om een relatieve file, dan staat op deze plaats de letter W als de file moet worden gelezen, of niets als de file alleen maar wordt gelezen. X is het file-nummer. Y is het drivenummer (letter U plus getal), en Z het apparaat-nummer letternummer (letter U gevolgd door getal).

Commando DSAVE"naam",X,Y
dient om een programma weg te schrijven naar diskette. X en Y zijn respectievelijk

9.2 Disk commando's

het drive-nummer en het apparaatnummer. Om gegevens op cassette te schrijven kan DSAVE niet gebruikt worden; hiervoor is het commando SAVE beschikbaar. In plaats van "naam" mag ook een stringvariabele gebruikt worden.

Commando DVERIFY X,Y
wordt gebruikt direct na het DSAVE-commando en dient om de inhoud van het computergeheugen te vergelijken met de inhoud van de file "naam" op floppy. Zijn beide inhouden gelijk dan verschijnt op het beeldscherm de melding OK. Zijn de programma's niet gelijk dan verschijnt de boodschap VERIFY ERROR en moet het programma nogmaals geSAVEd worden.

Commando HEADER"naam",X,Y,Z
dient om een diskette te formatteren. Gegevens die al op de diskette staan gaan daarbij verloren. X is een ID-nummer, bestaande uit 2 tekens. "naam" is maximaal 16 tekens groot.
Y en Z zijn respectievelijk drive-nummer en apparaatnummer.

Commando RECORD# X,Y,Z
zet de wijzer van een relatieve file, met file-nummer X, op een willekeurig record Y, en binnen dat record op een willekeurig byte Z.

Commando RENAME"naam1" TO "naam2", X,Y
dient om een file een andere naam ("naam2") te geven. X en Y zijn respectievelijk drive-nummer en apparaatnummer.

Commando SCRATCH"naam",X,Y
wist een file van floppy. De computer stelt voor alle zekerheid de vraag ARE YOU SURE?. Door op "Y" te drukken wordt de file gewist.

8/9.3

Geluid

Met de Commodore 128 kan ook geluid worden gemaakt. Geluid dat vooraf geprogrammeerd dient te worden. Hiertoe staat een flink repetoire aan instructies ter beschikking. In dit hoofdstuk worden de opdrachten besproken die de muzikale programmeur ten dienste staan.

Commando SOUND A,B,C,D,E,F,G,H
is een eenvoudige opdracht om geluid te maken. A,B en C moeten worden opgegeven, de andere parameters mogen ook worden weggelaten.

A geeft aan welke stem gebruikt wordt (1, 2 of 3).
B geeft de frequentie van de toon weer (van 0 tot 65535).
C geeft de duur van de toon aan, in 1/60 secondes (van 0 tot 32767). Dus als C=60 dan duurt de toon 1 seconde, als C=720 duurt de toon 12 seconden enzovoorts.
D geeft aan of de frekwentie omhoog (D=0), omlaag (=1) of tussen 2 waarden (=2) beweegt
E is de minimumwaarde voor D (tussen 0 en 65535)
F is de stapgrootte van D (0-32676)
G geeft de golfvorm aan (0-3)
H is de pulsbreedte

Commando ENVELOPE A,B,C,D,E,F,G
dient om een toonsoort te selecteren. Er zijn tien toonsoorten (muziekinstrumenten).

A is het nummer van de toonsoort (0-9)
B is de attack(0-15)
C is decay (0-15)
D is sustain (0-15)
E is release (0-15)
F is de golfvorm, waarbij 0 een driehoek aanduidt, 1 een zaagtand, 2 een rechthoek, 3 ruis en 4 ringmodulatie.
G geeft de pusbreedte aan (0-4096), alleen voor rechthoekige golven.

Commando FILTER A,B,C,D,E
wordt gebruikt om verschillende filter in- of uit te schakelen. Er zijn drie filters. Geluid kan hiermee verfraaid worden.

A is de onderste frequentie van het filter
B is low-pass filter (0=uit, 1=aan).
C is band-pass filter (0=uit, 1=aan).
D is high-pass filter (0=uit, 1=aan).
E is resonantie (0-15)

Commando PLAY"A,B,C,D,E,F"
dient om tonen of melodieën te spelen. In de tekststring staan codes die dienen als instructies voor de geluidchip van de computer en letters, die de noten van de notenbalk aangeven.

A is de gebruikte stem (V1, V2 of V3)
B is het octaaf(O0-O6)
C is de toon (T0-T9)

9.3 Geluid

D is het volume (U0-U15)
E is het filter (X0 = uit, X1 = aan)
Voor F vult men één van de muzieknoten in: C,D,E,F,G,A of B.

Commando TEMPO X
geeft aan in welk tempo de noten worden gespeeld. X is een getal tussen 0 en 255.

De duur van het geluid kan berekend worden met de volgende formules:

duur = 19.22/x seconden

Commando VOL X
geeft aan met welk volume het geluid wordt afgespeeld. X loopt van 0 tot en met 15 (maximale geluidsterkte).

8/9.4

Grafiek

Behalve de veelgebruikte opdrachten voor BASIC-programma's en de geluidsinstructies kent de Commodore 128 ook nog een groot aantal grafische instructies. Met behulp hiervan kan men zelf de prachtigste tekeningen maken, maar ook 'sprites', bewegende figuren, definiëren. De instructies die BASIC 7.0 voor grafische doeleinden bevat, worden hieronder kort beschreven.

Functie BUMP(X)
wordt gebruikt om te kijken of twee of meer sprites tegen elkaar zijn gebotst. BUMP(1) geeft als resultaat welke sprites tegen elkaar botsten. BUMP(2) geeft als resultaat welke sprites botsten met de achtergrond. Het resultaat is in beide gevallen een getal, aan de hand waarvan men zelf moet berekenen welke botsingen hebben plaatsgevonden. De uitkomst moet men omzetten in het binair equivalent, een bitpatroon. Dit achtcijferige bitpatroon laat zien welke sprites botsten: de '1'-tjes in het patroon botsten, de nullen niet.
Als bijvoorbeeld BUMP(1) als resultaat het getal 129 levert dan hebben sprite 1 en sprite 8 gebotst: het bitpatroon van 129 is 10000001.
Voor BUMP(2) doet men een analoge bewerking: is bijvoorbeeld het resultaat van BUMP(2) 13 (bitpatroon 00001101) dan hebben sprites 5, 6 en 8 gebotst met de achtergrond.

Commando COLLISION X,Y
kan voor meerdere doeleinden gebruikt worden. Y is altijd een regelnummer van een programma. Als X gelijk is aan 1 springt in het geval er een botsing optreedt tussen twee sprites het programma naar dit regelnummer. Op dit regelnummer moet een subroutine beginnen, die eindigt met RETURN.
Als X gelijk is aan 2 gebeurt hetzelfde zodra er een botsing optreedt tussen een sprite en de achtergrond. Tenslotte kan X ook nog de waarde 3 hebben; dan reageert COLLISSION op een lichtpen.

Commando MOVSPR X,Y,Z
doet sprite nummer X bewegen naar positie Y,Z op het beeldscherm.

Commando MOVSPR X,Y,Z
doet sprite X constant bewegen, onder hoek Y met snelheid Z.

Functie RSPCOLOR (X)
geeft als resultaat welke kleur (welke multicolor-waarden) sprite X als laatste had.

Functie RSPPOS(X,Y)
geeft als resultaat de positie (in horizontale of verticale richting) of de snelheid van sprite X.

9.4 Grafiek

Als Y gelijk is aan 0 krijgt men de X-positie. Is Y gelijk aan 2 dan levert de functie de Y-positie en als Y gelijk is aan 3 krijgt men de snelheid van de sprite (een getal tussen 0 en 15).

Functie RSPRITE (X,Y)
levert een aantal kenmerken van de sprites op. X is het nummer van de sprite. Voor Y kan een van de volgende waarden worden ingevuld:

0 : staat sprite aan of uit
1 : kleur van de sprite
2 : heeft achtergrond prioriteit (0=nee, 1=ja)
3 : wordt in X-richting sprite vergroot (0=nee, 1=ja)
4 : wordt in Y-richting sprite vergroot (0=nee, 1=ja)
5 : is het een multi-color sprite (0=nee, 1=ja)

Commando SPRCOLOR X,Y
dient om de multicolor-waarden van alle sprites in te stellen. X bepaalt de eerste kleur en Y de tweede.

Commando SPRITE X,Y,Z,A,B,C,D
definieert een sprite.

X is het nummer van de sprite.
Y geeft aan of de sprite aan- of uit staat (1=aan, 0=uit).
Z is de kleur van de sprite (tussen 1 en 16).
A geeft de prioriteit aan (0=voorgrond, 1=achtergrond)
B geeft de vergroting aan de X-richting (0=normale formaat, 1=vergroot).
D geeft multicolor aan (1=aan, 0=uit)

Commando SPRDEF
schakelt de sprite-editor in, waarmee men sprites kan maken. Op een 'ruitjesvel' van 24 x 21 hokjes kan met de cursor getekend worden.

Commando BOX X,Y,Z,A,B,C,D
kan worden gebruikt om een rechthoek te tekenen.

X duidt het kleurengeheugen aan (0, 1, 2 of 3)
Y,Z is het coordinatenpaar dat de linkerbovenhoek aangeeft
A,B is het coordinatenpaar dat de rechterbenedenhoek aangeeft
C is de rotatie-hoek, waarmee de rechterhoek moet worden geroteerd.
D geeft aan of de rechthoek moet worden ingekleurd (1=ja, 0=nee)

Commando CHAR(X,Y,Z,"tekst",A)
wordt gebruikt om tekst op het scherm te plaatsen. Dit kan met deze instructie zowel in de tekstmodus als in de grafische toestand.

X geeft het kleurengeheugen aan
Y geeft het nummer aan van de kolom waarin het eerste teken van de tekst moet komen
Z geeft het nummer aan van de rij (regel) waarin de tekst moet komen
A geeft aan of de tekst in "inversed video" of normaal moet worden weergegeven (0=normaal, 1=reverse)

Commando CIRCLE (X,Y,Z,A,B,C,D,E,F)
dient, zoals het woord al zegt, om cirkels te tekenen. Maar kunnen er elipsen, vierkanten en andere figuren mee worden getekend, door de parameters te variëren.

9.4 Grafiek

X is het kleurengeheugen dat wordt gebruikt

Y,Z is het coordinatenpaar dat het middelpunt van de te tekenen figuur aanduidt

A is de grootte van de straal in horizontale richting

B is de grootte van de straal in verticale richting

C is de hoek waaronder het tekenen begint

D is de hoek waaronder het tekenen eindigt

E is de rotatie-hoek waaronder de figuur beweegt

F hoek voor cirkelsegmenten (standaardwaarde 2)

8/9.5

Foutmeldingen in Basic 7.0

BASIC 7.0 kent een uitgebreid repertoire aan foutmeldingen, mededelingen die op het scherm verschijnen zodra er iets mis gaat met een programma. Als de BASIC-interpreter op zo'n fout stuit, zet een speciale routine de uitvoering van het programma dat op dat moment 'draait' stop. Een mededeling over de soort fout die is opgetreden verschijnt op het beeldscherm, plus het nummer van de regel waarin de fout is opgetreden.
Er zijn 41 omstandigheden waarin uitvoering van een programma stopt. Dit stoppen is weliswaar vaak frustrerend voor de programmeur, maar het voorkomt wel ergere 'bugs' die zouden kunnen optreden als de uitvoering van een programma ongehinderd voortgang zou vinden.
De omstandigheden die in een foutmelding resulteren, worden hieronder stuk voor stuk beschreven. Ook die van het disk-operating systeem, dat immers een onderdeel is van de interpreter.

Elke foutmelding heeft een nummer, hetgeen opzoeken in de handleiding vergemakkelijkt. Bovendien verschijnt op het scherm een korte tekst, die ongeveer aangeeft wat er loos is. Niet precies, want verschillende fouten hebben soms dezelfde melding tot gevolg.

BAD DISK
(Error 36)
De floppydisk is beschadigd of verkeerd geformatteerd. Formatteer opnieuw of gebruik een nieuwe floppy.

BAD SUBSCRIPT
(Error 18)
Een array heeft een verkeerde DIMensie of de DIMensie is wel correct maar een subscript wordt gebruikt dat buiten het array ligt.

BEND NOT FOUND
(Error 37)
In het programma wordt een BEGIN-opdracht gebruikt zonder dat er een BEND bij staat.

BREAK
(Error 30)
Tijdens uitvoering van het programma stuit de computer op een STOP-instructie, hetzij in het programma zelf, hetzij vanaf het toetsenbord: er is op de STOP-toets gedrukt.

CAN't CONTINUE
(Error 26)
Treedt op na een CONT-instructie vanaf het toetsenbord, terwijl het programma niet kan worden voortgezet. Na sommige onderbrekingen van een programma kan het namelijk niet meer worden opgestart: als het programma al is beëindigd, als de programmeur wijzigingen in het pro-

9.5 Foutmeldingen in Basic 7.0

gramma heeft aangebracht, of als het programma stopte met een foutmelding.

CAN't RESUME
(Error 31)
In het programma staat een RESUME-instructie, maar de TRAP-opdracht staat er niet, of niet correct.

DEVICE NOT PRESENT
(Error 5)
Een randapparaat is niet of niet goed aangesloten, danwel het heeft geen stroom.

DIRECT MODE ONLY
(Error 34)
In een programma wordt een instructie gebruikt die alleen in 'direct mode' gebruikt mag worden, dat wil zeggen alleen via het toetsenbord mag worden ingevoerd, niet via een programmaregel. Voorbeelden van zulke instructies zijn AUTO en RENUMBER.

DIVISION BY ZERO
(Error 20)
Een poging wordt gedaan om een getal of een expressie te delen door nul.

FILE DATA
(Error 24)
Vanaf floppydisk of cassette wordt alleen maar zinloze informatie gelezen; de computer kan er geen betekenis aan toekennen.

FILE NOT FOUND
(Error 4)
De file met opgegeven naam kan niet worden gevonden.

FILE NOT OPEN
(Error 3)
De file die u gebruikt (om te lezen of om informatie in te schrijven) moet eerst worden geopend.

FILE OPEN
(Error 2)
De file die u probeert te openen (om te lezen of om informatie in te schrijven) is al open. Om hem te openen moet u hem nu eerst sluiten.

FILE READ
(Error 41)
Een floppydisk wordt gelezen en terwijl dit gebeurt, treedt er een storing op (bijvoorbeeld: de floppy wordt uit de drive gehaald).

FORMULA TOO COMPLEX
(Error 25)
Een expressie, bijvoorbeeld een berekening of een logische uitdrukking, is zo ingewikkeld dat deze niet kan worden berekend.

ILLEGAL DEVICE NUMBER
(Error 9)
Poging wordt gedaan om een randapparaat op incorrecte wijze te benaderen. Bijvoorbeeld wordt geprobeerd om gegevens weg te schrijven naar een read-only bestand.

ILLEGAL DIRECT
(Error 21)
Sommige BASIC-instructies kunnen alleen in programma's gebruikt worden en niet via het toetsenbord worden ingebracht in de computer. Wordt dit toch gedaan, dan resulteert dit in deze foutmelding. Voorbeelden van zulke instructies zijn INPUT, GET en READ.

ILLEGAL QUANTITY
(Error 14)

9.5 Foutmeldingen in Basic 7.0

Er klopt iets niet met de rekenkundige bewerking waarnaar deze foutmelding verwijst. Bijvoorbeeld een wiskundige functie is fout (zoals logaritme van nul trachten uit te rekenen).

LINENUMBER TOO LARGE
(Error 38)
Het grootste getal dat men in BASIC als programmaregelnummer mag gebruiken is 63999. Gebruikt men een groter getal als regelnummer dan wordt een SYNTAX Error gegenereerd. Gebruikt men echter de RENUMBER-instructie om regelnummers groter dan 63999 te creëren dan krijgt men een LINENUMBER TOO LARGE-melding.

LOAD
(Error 29)
Er gaat iets mis met het laden van een file. Probeer opnieuw te laden.

LOOP NOT FOUND
(Error 32)
Programma stuit op DO-instructie, maar er is geen bijbehorende LOOP-opdracht.

LOOP WITHOUT DO
(Error 33)
Programma stuit op een LOOP-instructie, maar er is geen bijbehorende DO-opdracht.

MISSING FILE NAME
(Error 8)
U poogt iets met een file te doen, maar u hebt vergeten de naam van de file op te geven.

NEXT WITHOUT FOR
(Error 10)
Interpreter stuit op een NEXT-instructie, maar er is geen bijbehorende FOR-opdracht.

NO GRAPHICS AREA
(Error 35)
Interpreter stuit op grafische opdrachten die niet voorafgegaan worden door een GRAPHIC-instructie.

NOT INPUT FILE
(Error 6)
U poogt gegevens te lezen uit een file die uitsluitend geschikt is voor Write-Only.

NOT OUTPUT FILE
(Error 7)
U poogt gegevens te schrijven naar een file die uitsluitend geschikt is voor Read-Only.

OUT OF DATA
(Error 13)
U poogt meer gegevens te lezen, met een READ-instructie, dan er staan in de DATA-regels.

OUT OF MEMORY
(Error 16)
Een programma-routine heeft meer geheugen nodig dan er normaal wordt toegewezen. Bijvoorbeeld is dit het geval als loops (FOR...NEXT, BEGIN...BEND) te diep worden genest, of als het programma in zijn geheel meer RAM nodig heeft dan beschikbaar is.

OVERFLOW
(Error 15)
De uitkomst van een berekening is te groot, groter dan 1.701411834E+38.

REDIM'D ARRAY
(Error 19)
Een array kan slechts één keer in een programma worden geDIMensioneerd. Probeert men het een tweede keer te doen, dan verschijnt deze foutmelding.

9.5 Foutmeldingen in Basic 7.0

RETURN WITHOUT GOSUB
(Error 12)
De interpreter stuit op een RETURN-opdracht waarbij geen corresponderende GOSUB-instructie gevonden kan worden.

STRING TOO LONG
(Error 23)
Een string mag hoogstens 255 tekens lang zijn. Elke poging om een langere string te maken resulteert in deze foutmelding.

SYNTAX
(Error 11)
De meest voorkomende foutmelding, heeft meestal betrekking op een typfout.
Deze foutmelding verschijnt ook als men een regelnummer poogt te creëren groter dan 63999, of als men een naam aan een variabele toekent en voor die naam een BASIC-woord gebruikt.

TOO MANY FILES
(Error 1)
Op een Commodore 128 kunnen slechts 10 files tegelijkertijd open zijn. Bij het openen van de elfde file verschijnt deze foutmelding.

TYPE MISMATCH
(Error 22)
U heeft geprobeerd om een getalswaarde toe te kennen aan een string-variabele, of een string toe te kennen aan een numerieke variabele.

UNDEF'D FUNCTION
(Error 27)
U probeert om een zelfgemaakte functie FN te gebruiken die niet of niet correct eerder is gedefinieerd.

UNDEF'D STATEMENT
(Error 17)
In het programma wordt verwezen, of een sprong gemaakt, naar een regel met een nummer dat niet bestaat.

UNIMPLEMENTED COMMAND
(Error 40)
U gebruikt een instructie die niet behoort tot BASIC 7.0. Het is mogelijk dat de instructie die u gebruikt wel in een andere versie van BASIC voorkomt, maar niet in BASIC 7.0.

UNRESOLVED REFERENCE
(Error 39)
U poogt regelnummers te hernummeren (met RENUMBER), maar RENUMBER kan alleen werken met regelnummers die bestaan. In geval van deze foutmelding heeft u regelnummers opgegeven die niet bestaan.

VERIFY
(Error 28)
Het verifiëren van gegevens op disk of cassette if mislukt: er zijn verschillen tussen de gegevens op floppy of cassette en de gegevens in het geheugen van de computer. Opnieuw SAVEn is het beste.

8/9.6

CP/M

Control Program for Microcomputers, kortweg CP/M is een eenvoudig besturingssysteem, met een grote schare gebruikers. Er is zeer veel programmatuur voor CP/M beschikbaar; dit is ongetwijfeld ook de reden dat de firma Commodore CP/M beschikbaar heeft gesteld aan C 128 eigenaren. Met een echte Commodore computer, of met het operating systeem van een Commodore, heeft CP/M echter in het geheel niets te maken. Het maakt ook geen onderdeel uit van de Commodore 128, op de wijze zoals BASIC dat bijvoorbeeld wel doet. CP/M zit wel ingebouwd in de C 128, maar in feite is het een extra computer; gewoon een afzonderlijke machine. Ingebouwd in dezelfde kast als de Commodore computer. Deze extra computer heeft ook een eigen processor, de Z80 (de Commodore maakt gebruik van de 8502 chip).

Het CP/M operating systeem zit niet "ingebakken" in de computer, maar moet worden geladen vanaf floppy disk. Dat gaat geheel automatisch (auto-boot) als men de CP/M System Disk in de drive plaatst, de computer aanzet en vervolgens op de reset-knop drukt. In plaats van het indrukken van de reset-knop kan men ook het commando BOOT intypen.
Na een korte tijd, waarin het scherm enkele keren van kleur verschiet, verschijnt op het scherm A>. De computer verkeert nu in de CP/M modus. Men gebruikt vanaf dit moment een heel andere computer, dan de vertrouwde Commodore. Functietoetsen hebben een andere werking, en ook de I/O is anders geregeld.

CP/M werkt met 80 tekens op een regel. Bij het starten van de computer moet dus op de toets 40/80 DISPLAY worden gedrukt. Doet men dat niet, dan ziet men slechts de helft van hetgeen er te zien is op het beeldscherm. Het ontbrekende gedeelte kan men wel zien in de 40-tekensmodus, door de CONTROL-toets ingedrukt te houden en tegelijkertijd de linker en rechter-toetsen (pijltjes-toetsen) in te drukken.

Om terug te keren naar de 128-modus dient de CP/M System Disk uit de drive te worden verwijderd, waarna op de resettoets wordt gedrukt.

CP/M zelf bevat een zeer uitgebreide handleiding over de commando's. Het is zaak deze handleiding goed te bestuderen, wil men effectief met CP/M om kunnen gaan. De beschrijving van de verschillende commando's is gemakkelijk te benaderen. Nadat de computer in "CP/M-modus" is gebracht, op de wijze zoals hiervoor werd beschreven, typt men achter A> het woord HELP.

9.6 CP/M

Onmiddellijk verschijnt een lijst van onderwerpen:

C128CP/M	COMMANDS	CNTRLCHARS	COPYSYS
DIR	DUMP	ED	ERASE
GET	HELP	HEXCOM	INITDIR
LINK	MAC	PATCH	PIP (COPY)
RMAC	SAVE	SET	SETDET
SUBMIT	TYPE	USER	XREF
DATE	DEVICE		
FILESPEC	GENCOM		
KEYFIG	LIB		
PUT	RENAME		
SHOW	SID		

Bovendien is men nu in de HELP-modus terecht gekomen, zoals blijkt uit de prompt op het scherm HELP>.

Informatie over de onderwerpen in de lijst kan men verkrijgen door eenvoudigweg het woord in te typen. Typt met achter HELP> nu C128-CP/M dan krijgt men een algemeen overzicht van CP/M zoals dat in de Commodore zit. Het liggende streepje tussen C128 en CP/M staat niet afgebeeld op een toets op het toetsenbord; men krijgt het door de witte toets met het pijltje naar links in te drukken (cursortoets).
Uit het algemene overzicht kan men op dezelfde wijze weer verdere informatie oproepen: door het juiste woord in te typen. Wil men bijvoorbeeld informatie over COMMAND_LINE, een onderwerp dat genoemd wordt in de lijst met de naam C128_CP/M dat typt men achter de prompt HELP> in:

C128_CP/M,COMMAND_LINE

Heeft men eerst C128_CP/M ingetypt, de lijst bekeken en daaruit een keuze gemaakt dan kan men ook direct typen:

.COMMAND_LINE

Het is verstandig om te beginnen met het bestuderen van de informatie onder C128_CP/M,CONVENTIONS. Hier treft men een beschrijving aan van de

9.6 CP/M

notatiewijze die op de HELP-schermen wordt gebruikt.

Hieronder worden alle HELP-onderdelen kort beschreven. Daarbij is er vanuit gegaan dat men reeds in de HELP-modus verkeert, zodat de prompt HELP> zichtbaar is op het beeldscherm.

C128_CP/M
geeft een beschrijving van CP/M op de Commodore 128.

C128_CP/M,COMMAND_LINE
geeft een beschrijving van het gebruik van de twee witte cursor-toetsen.

C128_CP/M,DISK_STATUS
geeft een beschrijving van de gegevens over de disk, die in de rechteronderhoek van het beeldscherm zichtbaar zijn.

C128_CP/M,KEYBRD_DEFS
geeft een beschrijving van het gebruik van de SHIFT-, CONTROL- en COMMODORE-toets.

C128_CP/M,KEYBRD_DEFS, ALPHNUM_KEYS
geeft een beschrijving van het gebruik van de alfanumerieke toetsen en de toetsen op het cijferblokje van het toetsenbord.

C128_CP/M,KEYBRD_DEFS, ARROW_KEYS
geeft een beschrijving van het gebruik van de vier grijze pijltjes-toetsen.

C128_CP/M,KEYBRD_DEFS, EXTRA_KEYS
geeft een beschrijving van het gebruik van de toets met het Britse Pond teken, de witte pijl omhoog en andere toetsen die in combinatie met de CONTROL-toets gebruikt worden.

C128_CP/M,RT_SHFT_FNCT
geeft een beschrijving van hoe functies kunnen worden toegekend aan toetsen.

C128_CP/M,RT_SHFT_FNCT, MODE_TOGGLE
geeft een beschrijving van hoe de ALT-toets gebruikt wordt.

C128_CP/M,RT_SHFT_FNCT, STRING_EDIT
geeft een beschrijving van hoe aan toetsen een ander gebruik kan worden toegekend, door middel van strings.

C128_CP/M,RT_SHFT_FNCT, HEX_EDIT
geeft een beschrijving van hoe de hexadecimale waarde die iedere toets heeft, kan worden gewijzigd.

C128_CP/M,SPECIAL_FNCT
geeft een beschrijving van het standaardgebruik van de toetsen NO SCROLL, de grijze pijl naar links en de grijze pijl naar rechts, ENTER en RUN/STOP.

C128_CP/M,VIRTUAL_DISK
geeft een beschrijving van het disk E, en hoe deze, niet bestaande, disk gebruikt kan worden.

C128_CP/M,MFM_FORMATS
geeft een beschrijving van de verschillende disk-formats.

CNTRLCHARS
geeft een beschrijving van alle toetsen die in combinatie met de CONTROL-toets een bijzondere functie hebben.

De CP/M SYSTEM DISK bevat aan beide zijden informatie. Elke zijde heeft zijn eigen directory, zoals hieronder weergegeven.

9.6 CP/M

Kant 1:

CPM+.SYS	KEYFIG.HLP
CCP.COM	FORMAT.COM
HELP.COM	PIP.COM
HELP.HLP	DIR.COM
KEYFIG.COM	COPYSYS.COM

Kant 2:

DATE.COM	PATCH.COM
DATEC.ASM	PIP.COM
DATEC.RSX	PUT.COM
DEVICE.COM	RENAME.COM
DIR.COM	SAVE.COM
DIRLBL.RSX	SET.COM
DUMP.COM	SETDET.COM
ED.COM	SHOW.COM
ERASE.COM	SUBMIT.COM
GENCOM.COM	TYPE.COM
GET.COM	
INITDIR.COM	

De namen op deze beide kanten van de schijf, zijn evenzovele commando's die gebruikt kunnen worden voor van alles en nog wat. Hier volgt een korte beschrijving:

COPYSYS
wordt gebruikt om systeemfiles te kopieëren naar een nieuwe disk.
In de HELP-mode is meer informatie te vinden, eenvoudigweg door COPYSYS in te typen.

DIR
laat de inhoudsopgave van een disk in de drive op het beeldscherm zien. DIR heeft vele mogelijkheden, als men achter het woordje DIR nog een tweede trefwoord intypt. Meer hierover is te vinden in de HELP-mode:

DIR
DIR,BUILT-IN
DIR,BUILT-IN,EXAMPLES
DIR,WITHOPTIONS
DIR,WITHOPTIONS,OPTIONS
DIR,WITHOPTIONS,EXAMPLES

DUMP
laat de ASCII-code zien van een file waarvan men de naam moet opgeven.
In de HELP-mode treft men meer informatie onder het kopje DUMP.

ED
activeert een eenvoudige regel-editor.
In de HELP-mode is hierover informatie te vinden onder:
ED
ED,COMMANDS
ED,EXAMPLES

ERASE
wist een file waarvan men de naam moet opgeven uit een directory van een disk.
In HELP-mode is meer informatie beschikbaar onder:
ERASE
ERASE,OPTIONS
ERASE, EXAMPLES

9.6 CP/M

GET
verandert de plek waarvandaan invoer komt. In plaats van het toetsenbord is dat bij GET een opgegeven file. In HELP-mode is meer informatie beschikbaar onder:
GET
GET,OPTIONS
GET,EXAMPLES

HELP
activeert de HELP-functie van het systeem. Informatie is te vinden onder het sleutelwoord HELP.

HEXCOM
conveert een hexadecimale file naar een COM-file zoals die onder CP/M gebruikt wordt. Meer informatie in de HELP-modus onder het trefwoord HEXCOM.

INITDIR
voegt aan een bestaande file datum en tijd toe. In de HELP-modus is meer informatie te vinden onder het trefwoord INITDIR.

PATCH
installeert een opdrachtenreeks die een file (bijvoorbeeld een programma) verandert.
In HELP-modus is meer informatie te vinden onder het sleutelwoord PATCH.

PIP
kopieert een file van een bepaalde plek naar een aangegeven bestemming. Wordt veel gebruikt voor het kopieëren van diskfiles.
In HELP-modus is meer informatie te vinden onder de sleutelwoorden.
PIP
PIP,OPTIONS
PIP,EXAMPLES

SAVE
schrijft een opgegeven gedeelte van het geheugen naar disk. In HELP-mode is meer informatie verkrijgbaar onder SAVE,EXAMPLE.

SET
dient voor het installeren van een groot aantal diskparameters.
In HELP-modus is meer informatie te vinden onder:

SET
SET,LABEL
SET,LABEL,EXAMPLES
SET,PASSWORDS
SET,PASSWORDS,MODES
SET,ATTRIBUTES
SET,ATTRIBUTES,EXAMPLES
SET,DEFAULT
SET,TIME-STAMPS
SET,TIME-STAMPS,OPTIONS
SET,TIME-STAMPS,EXAMPLES
SET,DRIVES

SETDEF
wordt vaak in combinatie met SUBMIT gebruikt. Bepaalt een volgorde waarin diskdrives worden bekeken door het systeem bij het zoeken naar een file.
In HELP-mode is uitleg te vinden onder:
SETDEF
SETDEF,EXAMPLES

SUBMIT
executeert een opgegeven CP/M batchfile.
In HELP-mode is uitleg te vinden onder:
SUBMIT
SUBMIT,SUBFILE
SUBMIT,EXECUTE
SUBMIT,PROFILE.SUB

9.6 CP/M

TYPE
laat de inhoud van een ASCII-file zien. In HELP-mode is informatie te vinden onder:
TYPE
TYPE,EXAMPLES

USER
kent een getal toe aan een computergebruiker. Dit getal wordt ook opgenomen in files, zodat deze beschermd zijn tegen gebruik door mensen die het getal niet kennen. Meer informatie is te vinden in de HELP-modus onder de trefwoorden:
USER,
USER,EXAMPLES

DATE
dient om datum en tijd in te stellen, danwel zichtbaar te doen worden. Meer informatie in de HELP-modus onder de trefwoorden:
DATE
DATE,EXAMPLES

DEVICE
dient voor het instellen van parameters als scherm-grootte en baudrate. In de HELP-modus is uitleg te vinden onder de trefwoorden:
DEVICE
DEVICE,OPTIONS
DEVICE,EXAMPLES
DEVICE,C128_DEVICES

FILESPEC
is alleen maar een tekst-bestand, waarin beschreven staat hoe files in CP/M-format worden gespecificeerd. In HELP-modus is hierover informatie te vinden onder het trefwoord FILESPEC

KEYFIG
kan gebruikt worden om willekeurig welke toets op het toetsenbord een andere definitie te geven. Meer informatie staat in HELP-modus onder de trefwoorden:
KEYFIG
KEYFIG,EDITING KEYS
KEYFIG,EDITING KEYS,
  EDIT COLORS
KEYFIG,EDITING KEYS,EDIT HEX
KEYFIG,EDITING KEYS,
  EDIT SPECIAL
KEYFIG,EDITING KEYS,
  EDIT STRINGS
KEYFIG,FINISHING UP
KEYFIG,FOR_EXPERTS
KEYFIG,KEY_VALUES
KEYFIG,LOG/PHY_CLRS
KEYFIG,SELECT_A_KEY
KEYFIG,SETTING_UP
KEYFIG,SETTING_UP,
  WHAT_TO_DO

PUT
verandert de bestemming van output die normaal naar de printer gaat. Het doet het tegenovergestelde van wat GET doet. Informatie in de HELP-modus is te vinden onder:
PUT
PUT,OPTIONS
PUT,EXAMPLES

SHOW
laat de toestand waarin de disk op dit moment verkeert op het beeldscherm zien. In de HELP-modus is meer informatie te vinden onder de trefwoorden:
SHOW
SHOW,EXAMPLES

Een aantal van bovenbeschreven commando's zijn ingebouwde commando's. Dat wil zeggen dat ze gelijk met de systeeminformatie in RAM worden gelezen bij het opstarten van het systeem.

9.6 CP/M

Andere commando's echter staan en blijven op disk, en worden slechts van gelezen als ze worden ingetypt.
Deze volgende commando's staan altijd in het geheugen (RAM):

DIR (alleen)

HELP

HEXCOM

INITDIR

PATCH

PIP

SAVE

TYPE voor beeldscherm

USER

Deze commando's staan altijd op de disk en worden van daaraf aangeroepen:

DIR + opties

DUMP

DATE

DEVICE

ED

ERASE

GET

KEYFIG

PUT

SET

SETDEF

SHOW

SUBMIT

TYPE voor printer en multiple files

Het kopiëren van de CP/M Systeem Disk

Om er zeker van te zijn dat men altijd een goed werkende systeemschijf bij de hand heeft, verdient het aanbeveling om deze disk direct na het uitpakken van de computer te kopiëren. Ook van belangrijke schijven die men zelf maakt, worden van tijd tot tijd kopieën gemaakt.
Dat gaat heel gemakkelijk. Het enige dat men nodig heeft, is een lege floppydisk, die aan beide zijden beschreven kan worden. Deze disk moet geformatteerd worden. Met het PIP-commando wordt vervolgens een kopie gemaakt.

Hoe het kopiëren precies gaat, hangt af van het feit of men één of twee diskdrives ter beschikking heeft. Hieronder worden de procedures voor beide systemen beschreven.

Kopiëren met één diskdrive.
Er wordt gebruik gemaakt van een niet-bestaande diskdrive: de virtuele diskdrive E. Ook al bestaat deze niet, dat maakt voor de computer niets uit: deze "denkt" dat hij wel bestaat. Er is geen enkel verschil tussen een drive A en een drive E.

De procedure is als volgt:

Eerst wordt de computer opgestart en in CP/M-modus gebracht.

9.6 CP/M

Als de prompt A> zichtbaar is, wordt het commando FORMAT ingetypt.

Een menuutje verschijnt dan, waaruit met de pijltjes-omlaag-toetsen de optie: C128 double sides wordt gekozen.
Druk op RETURN-toets om keuze definitief te maken.

De computer vraagt vervolgens om de CP/M System Disk uit de drive te verwijderen, en een nieuwe, nog te formatteren, disk erin te stoppen. Doe dit en typ vervolgens $.

De computer gaat vervolgens formatteren. Als dit is gebeurd, komt de vraag op het beeldscherm of u nog een floppy wilt formatteren. Typ N als u geen andere disk wilt formatteren, Y als u dit wel wilt.

Haal de zojuist geformatteerde disk uit de drive en stop de CP/M SYSTEM DISK er weer in.
Typ vervolgens de opdracht:
PIP E:=A:*.*

De SYSTEM DISK is Disk A, de nieuwe disk heet Disk E. Af en toe moeten disk A en E verwisseld worden in de drive. De computer geeft een melding als dit moet gebeuren. Iedere keer als u een disk in de drive heeft gedaan, moet u op de RETURN-toets drukken.

Op een gegeven moment is de kopieerprocedure afgelopen. U heeft dan echter pas één zijde van de disk gekopieerd. Stop de CP/M System Disk nu met de andere kant naar boven in de diskdrive en herhaal de hele procedure.

Kopiëren met twee diskdrives
Nu hoeft geen gebruik gemaakt te worden van een niet-bestaande diskdrive: er zijn twee diskdrives beschikbaar. De ene drive heet drive A, en de andere drive B.

De procedure is als volgt:
Eerst wordt de computer opgestart en in CP/M-modus gebracht.
De CP/M SYSTEM DISK wordt in drive A gedaan, de nieuwe nog te formatteren disk in drive B.
Als de prompt A> zichtbaar is, wordt ingetypt:

FORMAT B:

Een menuutje verschijnt dan, waaruit met de pijltjes-omlaag toetsen de optie: C128 double sides wordt gekozen. Druk op RETURN-toets om keuze definitief te maken.

De computer vraagt vervolgens om de CP/M System Disk uit drive A te verwijderen, en een nieuwe, nog te formatteren, disk in deze zelfde drive A te stoppen. Doet dit en typ vervolgens $.

De computer gaat vervolgens formatteren. Als dit is gebeurd, komt de vraag op het beeldscherm of u nog een floppy wilt formatteren. Typ N als u geen andere disk wilt formatteren, Y als u dit wel wilt.

Nu kan het kopiëren beginnen. Typ de opdracht:

PIP B:=A:*.*

Op een gegeven moment is de kopieerprocedure afgelopen. U heeft dan echter pas één zijde van de disk gekopieerd. Stop de CP/M System Disk nu met de andere kant naar boven in de diskdrive en herhaal de hele procedure.