

***Plenge · Szczepanowski***

***DAS TRAININGSBUCH  
ZUM  
SIMON's BASIC***

***EIN DATA BECKER BUCH***

**Plenge · Szczepanowski**

**DAS TRAININGSBUCH  
ZUM**

**SIMON's BASIC**

**EIN DATA BECKER BUCH**

ISBN 3-89011-009-6

3. überarbeitete Auflage

Copyright (C) 1985

DATA BECKER GmbH  
Merowingerstr. 30  
4000 Düsseldorf

Alle Rechte vorbehalten. Kein Teil dieses Buches darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren) ohne schriftliche Genehmigung der DATA BECKER GmbH reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

## Wichtiger Hinweis

Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technische Angaben und Programme in diesem Buch wurden von den Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler sind die Autoren jederzeit dankbar.

Das vorliegende Buch ist mit einer SIMON'S BASIC Diskettenversion vom November 1983 erstellt worden. Wir weisen Sie darauf hin, daß Disketten oder Modulversionen, die nach diesem Zeitpunkt erschienen sind, Korrekturen oder Änderungen in der Funktion der Befehle vorweisen können.

## VORWORT

SIMON'S BASIC ist eine außergewöhnliche Befehlserweiterung für den COMMODORE 64, die sich großer Beliebtheit erfreut. Mit gut 100 Befehlen aus den verschiedensten Bereichen erweitert sie das COMMODORE BASIC in eine umfassende, leistungsfähige Sprache. Wer den großen Programmierkomfort, den SIMON'S BASIC bietet, voll nutzen möchte, der muß mit den einzelnen Befehlen richtig umgehen können. Stolpersteine auf dem Weg zur Beherrschung von SIMON'S BASIC sind aber das nicht gerade umfangreiche Handbuch und die Macken, die SIMON'S BASIC aufweist.

Aufgrund der vielen Anfragen von COMMODORE 64 Anwendern entschlossen wir uns deshalb, ein umfangreiches Trainingshandbuch zu SIMON'S BASIC herauszugeben. Als Autoren konnten wir zwei Fachleute gewinnen, die für Sie sicher keine Unbekannten mehr sind. Axel Plenge, der die beliebte und erfolgreiche SUPERGRAPHIK erstellt hat, widmete sich in diesem Buch -wie kann es anders sein- den Graphik- und Soundmöglichkeiten des SIMON'S BASIC. Alle anderen Bereiche wurden von Norbert Szczepanowski erstellt, der schon mit dem FLOPPY-BUCH gezeigt hat, welche Fähigkeiten man den COMMODORE - Rechnern entlocken kann, wenn man sich nur intensiv genug damit beschäftigt.

Das Ergebnis der monatelangen Kleinarbeit von Axel Plenge und Norbert Szczepanowski kann sich sehen lassen. Auf weit über 300 Seiten werden alle SIMON'S BASIC Befehle und ihre Anwendung ausführliche und mit vielen Beispielen beschrieben. Jeder Befehl wurde einzeln behandelt, sodaß sich das vorliegende Trainingshandbuch auch als Nachschlagewerk eignet.

Viel Spaß bei der Lektüre dieses Buches und bei der Arbeit mit SIMON'S BASIC.

Dr. Achim Becker



# INHALTSVERZEICHNIS

## 1. KAPITEL

Einleitung .....	11
------------------	----

## 2. KAPITEL

Programmierhilfen .....	14
-------------------------	----

2.1 Belegung der Funktionstasten .....	14
--	----

2.1.1 KEY - Defintion der Tasten .....	16
--	----

2.1.2 DISPLAY - Anzeigen der Definitionen .....	19
---	----

2.1.3 Testaufgaben .....	21
--------------------------	----

2.2 Komfortable Programmerstellung .....	22
--	----

2.2.1 AUTO - automatische Zeilennumerierung .....	22
---	----

2.2.2 RENUMBER - Zeilen umnumerieren .....	24
--	----

2.2.3 MERGE - Programmverknüpfung .....	28
---	----

2.2.4 Testaufgaben .....	32
--------------------------	----

2.3 Listhilfen .....	33
----------------------	----

2.3.1 PAGE - Formatierte Listausgabe .....	34
--	----

2.3.2 OPTION - Hervorheben aller SIMON'S BASIC-Befehle .....	35
--	----

2.3.3 DELAY - Verändern der List-Geschwindigkeit .....	36
--	----

2.3.4 FIND - Aufsuchen von Zeichenketten .....	37
--	----

2.3.5 Testaufgaben .....	40
--------------------------	----

## 3. KAPITEL

Fehlerbehandlung .....	42
------------------------	----

3.1 Externe Fehlerbehandlung .....	43
------------------------------------	----

3.1.1 TRACE - Anzeigen der aktiven BASIC-Zeilen .....	43
---	----

3.1.2 RETRACE - Anzeige des letzten TRACE-Fensters .....	45
--	----

3.1.3 DUMP - Anzeigen der benutzten Variablen .....	47
---	----

3.1.4 COLD - Kaltstart .....	49
------------------------------	----

3.1.5 OLD - Anti-NEW .....	50
----------------------------	----

3.1.6 Testaufgaben .....	51
--------------------------	----

3.2	Programminterne Fehlerbehandlung .....	52
3.2.1	ON ERROR - Abfangen eines Programmfehlers .....	52
3.2.2	NO ERROR - Ausschalten der Fehlerbehandlung .....	55
3.2.3	ON ERROR / NO ERROR im Detail .....	55
3.2.4	OUT .....	62
3.2.5	Testaufgaben .....	63
<b>4. KAPITEL</b>		
	Programmschutz .....	64
4.1	DISAPA - Kennzeichen geschützter Zeilen .....	64
4.2	SECURE - Schützen der gekennzeichneten Zeilen .....	65
<b>5. KAPITEL</b>		
	Programmstruktur .....	66
5.1	Logische Operationen .....	67
5.1.1	IF ... THEN ... ELSE .....	67
5.1.2	RCOMP .....	68
5.1.3	REPEAT ... UNTIL .....	70
5.1.4	LOOP ... EXIT IF ... END LOOP .....	72
5.1.5	Testaufgaben .....	74
5.2	Sprungbehandlung .....	75
5.2.1	PROC - Symbolische Sprungadressen (Label) .....	75
5.2.2	END PROC - Routinenende festlegen .....	76
5.2.3	CALL - Sprung zu Label .....	77
5.2.4	EXEC - symbolischer Unterprogrammaufruf .....	78
5.2.5	CGOTO - berechneter Sprung .....	79
5.2.6	Testaufgaben .....	80
<b>6. KAPITEL</b>		
	Variablen .....	82
6.1	LOCAL - Festvariablendefinition .....	82
6.2	GLOBAL - Wiederherstellung der Festvariablen .....	83
6.3	Testaufgaben .....	84



## **7. KAPITEL**

Zahlenbehandlung .....	85
7.1 Arithmetische Operatoren .....	85
7.1.1 MOD - Rest einer Division .....	85
7.1.2 DIV - Vorkommazahl einer Division .....	87
7.1.3 FRAC - Bestimmen der Nachkommazahl .....	88
7.1.4 EXOR - Exklusiv-Oder Verknüpfung .....	89
7.1.5 Testaufgaben .....	91
7.2 Zahlenumwandlung .....	92
7.2.1 % - Umwandlung binär nach dezimal .....	92
7.2.2 \$ - Umwandlung hexadezimal nach dezimal .....	93
7.2.3 Testaufgaben .....	94

## **8. KAPITEL**

Stringoperationen .....	95
8.1 INSERT - Einfügen eines Strings in einen anderen .....	95
8.2 INST - Überschreiben eines Strings mit einem anderen .....	97
8.3 PLACE - Suchen eines Stringteils .....	99
8.4 DUP - Vervielfachen eines Strings .....	109
8.5 Testaufgaben .....	111

## 9. KAPITEL

Ausgabekontrolle .....	113
9.1 Formatierte Ausgabe .....	113
9.1.1 CENTRE- zentrierte Ausgabe .....	115
9.1.2 USE - dezimalpunktorientierte Ausgabe .....	117
9.1.3 AT - positionierte Ausgabe .....	119
9.1.4 LIN - Bestimmen der Cursorposition .....	122
9.1.5 PAUSE - Verzögern des Programmablaufs .....	123
9.1.5 Testaufgaben .....	126
9.2 Bildschirm blinken .....	127
9.2.1 FLASH - Blinken einer Bildschirmfarbe .....	127
9.2.2 OFF - Ausschalten des FLASH-Modus .....	129
9.2.3 BFLASH - Blinken des Bildschirmrahmens .....	131
9.2.4 BFLASH O - Ausschalten des BFLASH-Modus .....	133
9.2.5 Testaufgaben .....	135
9.3 Hintergrund- und Rahmenfarbe .....	136
9.3.1 COLOUR - Ändern der Rahmen- und Hintergrundfarbe .....	139
9.3.2 BCKGND - Einschalten des Extended-Color Modus .....	141
9.3.3 NRM - Ausschalten des Extended-Color-Modus .....	147
9.3.4 Testaufgaben .....	148
9.4 Steuerung : Bildschirmbereiche .....	149
9.4.1 FCHR - Bildschirmbereich mit Zeichen füllen .....	152
9.4.2 FCOL - Bildschirmbereich färben .....	156
9.4.3 FILL - Bildschirmbereich mit gefärbten Zeichen füllen .....	159
9.4.4 MOVE - Bildschirmbereich duplizieren .....	162
9.4.5 INV - Bildschirmbereich invertieren .....	163
9.4.6 Testaufgaben .....	166
9.5 Bildschirm scrollen .....	167
9.5.1 LEFT - Linksscrollen .....	168
9.5.2 RIGHT - Rechtsscrollen .....	171
9.5.3 UP - Aufwärtsscrollen .....	174
9.5.4 DOWN - Abwärtsscrollen .....	176
9.5.5 Testaufgaben .....	178

## 10. KAPITEL

Eingabekontrolle .....	179
10.1 FETCH - Selektion der eingegebenen Zeichen .....	180
10.2 INKEY - Abfrage der Funktionstasten .....	181
10.3 RESET - Setzen des DATA-Zeigers .....	182
10.3 Testaufgaben .....	184

## 11. KAPITEL

Ein/Ausgabe Peripheriebefehle .....	185
11.1 Diskettenbefehle .....	185
11.1.1 DISK - Befehlsübertragung .....	185
11.1.2 DIR - Anzeigen des Directorys .....	187
11.1.3 SCRSV - Abspeichern eines LOW-RES Bildschirms .....	189
11.1.4 SCRLD - Laden eines LOW-RES Bildschirms .....	191
11.1.5 Testaufgaben .....	192
11.2 Druckerbefehle .....	193
11.2.1 COPY - Hardcopy des HIGH-RES/MULTI COLOR Bildschirms	
11.2.2 HRDCPY - Hardcopy des LOW-RES Bildschirms .....	193
11.2.3 Testaufgaben .....	194

## 12. KAPITEL

Graphik .....	196
12.1 Hardwarevoraussetzungen .....	197
12.2 Graphikmodus und Farbbestimmung .....	202
12.2.1 HIRES - Bestimmen der Zeichen- und Hintergrundfarbe und Einschalten der HGR mit Löschen .....	204
12.2.2 MULTI - Bestimmen der Zeichenfarben und Umschalten in den MULTI-COLOR Modus .....	207
12.2.3 NRM - Ausschalten des Extended-Color-Modus .....	211
12.2.4 CSET 2 - Einschalten der Graphikanzeige ohne löschen	213
12.2.5 LOW COL - Farbwechsel und Zeichnen mit Farbsetzung ...	215

12.2.6 HI COL - Farbausgangszustand herstellen und Zeichnen ohne Farbsetzung .....	219
12.2.7 Testaufgaben .....	224
12.3 Graphikbefehle .....	225
12.3.1 PLOT - Setzen eines Punktes .....	226
12.3.2 TEST - Prüfen auf gesetztem Punkt .....	228
12.3.3 LINE - Zeichnen einer Linie .....	233
12.3.4 REC - Zeichnen eines Rechtecks .....	237
12.3.5 BLOCK - Zeichnen eines ausgefüllten Rechtecks .....	242
12.3.6 CIRCLE - Zeichnen eines Kreises .....	245
12.3.7 ARC - Zeichnen eines Bogens .....	250
12.3.8 ANGL - Radius einer Figur zeichnen .....	256
12.3.9 PAINT - Beliebige Fläche ausfüllen .....	261
12.3.10 Testaufgaben .....	265
12.4 Zeichnen eigener Figuren .....	267
12.4.1 DRAW - Erstellen einer Figur .....	267
12.4.2 ROT - Drehen und Vergrößern dieser Figur .....	273
12.4.3 Testaufgaben .....	278
12.5 Text innerhalb der Graphik .....	279
12.5.1 CHAR - einzelnes Zeichen in die Graphik setzen .....	279
12.5.2 TEXT - Zeichenkette in die Graphik setzen .....	283
12.5.3 Testaufgaben .....	287
<b>13. KAPITEL</b>	
Zeichensatzerstellung .....	288
13.1 MEM - Verlegen des Zeichensatzes von ROM in RAM .....	292
13.2 DESIGN 2 - Bestimmen des zu verändernden Zeichens .....	296
13.3 @ - Definition eines Zeichens .....	297
13.4 CSET 0/1 - Umschalten auf zweiten Zeichensatz .....	302
13.5 Testaufgaben .....	304

## 14. KAPITEL

Sprites (MOBs) .....	305
14.1 Spritedefinition .....	310
14.1.1 DESIGN O/1 - Speicherplatzzuteilung für Sprites .....	310
14.1.2 @ - Form eines Sprites eingeben .....	312
14.1.3 MOB SET - Eigenschaften des Sprites bestimmen .....	315
14.1.4 CMOB - Farbwahl eines MULTI-COLOR Sprites .....	319
14.1.5 Testaufgaben .....	320
14.2 Spritesteuerung .....	321
14.2.1 MMOB - Darstellen oder Bewegen eines Sprites .....	321
14.2.2 MOB OFF - Ausschalten eines Sprites .....	325
14.2.3 RLOCMOB - Darstellen und Weiterbewegen eines Sprites ..	329
14.2.4 DETECT - Sprite-Kollisions-Abfrage vorbereiten .....	333
14.2.5 CHECK - Abfrage auf Sprite-Kollision .....	335
14.2.6 Testaufgaben .....	340

## 15. KAPITEL

Musik .....	340
15.1 Hardwarevoraussetzungen .....	341
15.2 VOL - Einstellen der Lautstärke .....	345
15.3 WAVE - Wellenform bestimmen .....	348
15.4 ENVELOPE - Hüllkurve einstellen .....	353
15.5 MUSIC - Tonfolge festlegen .....	356
15.6 PLAY - Wiedergabe der Musik .....	360
15.7 Testaufgaben .....	363

## 16. KAPITEL

Steuernde Peripherie .....	364
16.1 LIGHTPEN .....	364
16.1.1 PENX - Bestimmen der X-Koordinate .....	366
16.1.2 PENY - Bestimmen der Y-Koordinate .....	367
16.2 PADDLE und JOYSTICK .....	369
16.2.1 POT - PADDLE-Wert abfragen .....	370
16.2.2 JOY - Funktion des Joysticks bestimmen .....	371

## **17. KAPITEL**

Anhang .....	373
17.1 Lösungen zu den Testaufgaben .....	373
17.2 Farbtabelle .....	374
17.3 Fehlermeldungen .....	375
17.3 Alphabetische Befehlsübersicht .....	377
17.4 SIMON'S BASIC Daten .....	378

# 1. KAPITEL

## EINLEITUNG

Sofort, nachdem wir zum ersten Male von Simon's Basic als komfortable Basicerweiterung hörten und uns näher mit Ihr beschäftigten, wußten wir, das ist die Erweiterung für den Commodore 64.

Sie haben sich von den vielen und überwältigenden Möglichkeiten dieses Gerätes überzeugen lassen und zugriffen. Doch zuhause bei der Lektüre Ihres CBM 64 - Benutzerhandbuchs mußten Sie feststellen, daß all die Fähigkeiten, die unter der Tastatur stecken, nur schwer und sehr träge vom originalen Basic V2 (das Basic, das stets in Ihrem Rechner vorhanden ist) bewältigt werden können. Graphik, Sound und Sprites müssen durch eine Unmasse an PEEKs und POKEs gesteuert werden, die fast undurchschaubar und gähnend langsam sind. Gleichzeitig fehlt jede Möglichkeit einer systematischen und komfortablen Programmierung, wie sie beispielsweise in höheren Programmiersprachen verwirklicht ist. So steht Ihnen zum Konstruieren von Schleifen nur ein Befehl (FOR...NEXT) zur Verfügung; ebenso bei Vergleichs- (IF) oder Sprungbefehlen (GOTO/GOSUB). Die Möglichkeiten zur Programmerstellung mit LIST als einzigen Befehl sind wirklich nicht sehr berauschend.

Kurz: wer wirklich ernsthaft mit seinem Rechner arbeiten will und nicht nur vorgefertigte Spiele oder andere Anwenderprogramme (Textverarbeitung, Dateiverwaltung) verwendet, hat eine Basicerweiterung nötig.

Simon's Basic wendet sich an den universellen Programmierer. Es beinhaltet also Befehle zu (fast) allen Bereichen, die den 64er zum Computer des Jahres 1983 gemacht haben. Selbstverständlich werden Spezialisten, die sich nur mit einem Themenbereich befassen wie z.B. Graphik, Sound usw., auf eine spezielle Erweiterung oder einer anderen Programmform angewiesen sein. Doch für den Einstieg und auch die fortschrittliche Basicprogrammierung ist Simon's Basic optimal.

Umfangreiche Programmierhilfen wie die Belegung der Funktionstasten, Listhilfen oder Umnummerierung von Basiczeilen ermöglichen Ihnen die schnelle und komfortable

Programmerstellung. Wo Sie vorher umständlich lange Befehle eintippen mußten, da genügt jetzt ein Tastendruck.

Viele Strukturbefehle wie REPEAT, symbolische Adressierung o.ä., die aus höheren Programmiersprachen stammen, erleichtern Ihnen das Erzeugen leistungsfähiger Programme.

Bildschirmformatierung und Graphikerzeugung mit allem, was dazu gehört (Sprites, Zeichensatzänderung, Multicolor und hochauflösende Graphik) eröffnen Ihnen die unzählbaren Möglichkeiten des 64ers. So unterstützen viele Graphikbefehle die Erstellung anspruchsvoller Graphiken auf Ihrem Bildschirm, die selbstverständlich auch auf einem Drucker ausgedruckt werden können.

Der Sound darf natürlich nicht fehlen, ebensowenig wie Hilfen bei der Joystick-, Lightpen- und Paddleabfrage.

Klar, kein Programm ist ohne Fehler oder Mängel, so auch Ihr Simon's Basic nicht. Es kommt vor, daß einzelne Funktionen nicht richtig ausgeführt werden oder Abstürze hervorgerufen werden. Wir werden Ihnen natürlich helfen, diese Klippen und Untiefen zu umschiffen und Ihr Programmschiff sicher in den Zielhafen zu steuern. Manche zusätzlichen Befehle hätte man sich natürlich noch wünschen können, so eine Graphikabspeicherung oder überhaupt mehr Diskettenbefehle, die hier nur recht unvollkommen unterstützt werden.

In diesem Buch werden wir versuchen, Ihnen die einzelnen Befehle möglichst gut verständlich und vor allem geordnet darzustellen. Jeweils auf die Mängel im ursprünglichen Simon's Basic - Handbuch zu verweisen schien uns (außer bei sehr gravierenden Fehlern) angesichts des chaotischen Zustandes dieses Heftchens unzweckmäßig und unnötig vom Wesentlichen ablenkend.

Somit haben wir den Befehlen eine völlig andere Ordnung gegeben, die auf einander aufbauend für ein Trainings- und Lernbuch geeignet ist:

Beginnend bei den Programmierhilfen und der Fehlerbehandlung zur komfortablen Programmierung über Befehle zur strukturierten Programmierung kommen wir zu dem großen Kapitel der Ein- und Ausgabekontrolle, dem die Graphik-, Sprite- und Soundprogrammierung angeschlossen sind. Wie Sie sehen, wurde mit den Befehlen begonnen, die zur allgemeinen Programmierung notwendig sind, um später auf die Anwendungen mit Graphik und



Sound vorbereitet zu sein.

Als Abschluß zu jedem Sinnabschnitt können Sie durch einen kleinen Test, dessen Lösungen am Ende des Buches stehen, Ihr Wissen überprüfen und gegebenenfalls bei einzelnen unklaren Punkten nachhaken.

Wir hoffen, Ihnen mit dieser Darstellung ein lehrreiches und informatives Buch in die Hände gegeben zu haben. Besonderes Augenmerk sollten Sie vielleicht auf den Abschnitt 9.3 werfen, der drei Befehle beschreibt, die im Handbuch nicht aufgeführt wurden und auch sonst vorher völlig unbekannt waren. Es gibt noch eine Reihe weiterer im Handbuch nicht aufgeführter Kommandos, die jedoch nicht oder nicht fehlerfrei funktionieren und oft in einem Absturz des Rechners enden. Aus diesem Grunde wurden sie hier nicht weiter aufgeführt.

## **2. KAPITEL**

### **PROGRAMMIERHILFEN**

#### **2.1 BELEGUNG DER FUNKTIONSTASTEN**

Die meisten Personal Computer besitzen Funktionstasten zum vereinfachten Eingeben oft benötigter Befehle. Man unterscheidet jedoch zwei verschiedene Arten von Funktionstasten:

- individuell belegbare Funktionstasten, die vom Benutzer mit beliebigen Zeichenfolgen oder Befehlen belegt werden können.
- ASCII-Funktionstasten, denen wie allen anderen Tasten ein ASCII-Wert zugeordnet ist. Die Betätigung dieser Funktionstasten kann im Programm abgefragt werden. Abhängig davon kann der weitere Programmablauf gesteuert werden.

Der COMMODORE 64 ist in der Standardversion nur mit ASCII-Funktionstasten ausgerüstet. Diese acht Funktionstasten haben folgende ASCII-Werte:

<b>F1 - 133</b>	<b>F3 - 134</b>	<b>F5 - 135</b>	<b>F7 - 136</b>
<b>F2 - 137</b>	<b>F4 - 138</b>	<b>F6 - 139</b>	<b>F8 - 140</b>

Im Programm können Sie die Betätigung dieser Tasten z.B. mit der folgenden Befehlsfolge abfragen:

```
100 GET X$:IF ASC(X$)<133 OR ASC(X$)>140 THEN 100  
110 PRINT"DER ASCII-WERT DIESER FUNKTIONSTASTE IST:"  
      :ASC(X$)  
120 GOTO 100
```

Dieses Programm akzeptiert nur die Betätigung der Funktionstasten und zeigt dann den entsprechenden ASCII-Wert. Mit der Taste RUN/STOP können Sie dieses Programm verlassen. Doch diese Handhabung ist nicht der eigentliche Sinn und Zweck von Funktionstasten. Mit SIMON'S BASIC haben Sie nun

die Möglichkeit, jede Funktionstaste mit beliebigen Befehlen oder Zeichenketten zu belegen. Doch nicht nur das, es wurden zusätzliche 8 Funktionstasten eingerichtet, die mit Hilfe der Commodore-Taste (C=) angesprochen werden können. Die folgende Übersicht zeigt, wie Sie die Funktionstasten erreichen können:

-----		
<b>: FUNKT.-TASTE :</b>		<b>DRÜCKEN :</b>
-----		
<b>: F1</b>	<b>:</b>	<b>F1</b>
<b>: F2</b>	<b>:</b>	<b>SHIFT + F1</b>
<b>: F3</b>	<b>:</b>	<b>F3</b>
<b>: F4</b>	<b>:</b>	<b>SHIFT + F3</b>
<b>: F5</b>	<b>:</b>	<b>F5</b>
<b>: F6</b>	<b>:</b>	<b>SHIFT + F5</b>
<b>: F7</b>	<b>:</b>	<b>F7</b>
<b>: F8</b>	<b>:</b>	<b>SHIFT + F7</b>
<b>: F9</b>	<b>:</b>	<b>C= + F1</b>
<b>: F10</b>	<b>:</b>	<b>C= + SHIFT + F1</b>
<b>: F11</b>	<b>:</b>	<b>C= + F3</b>
<b>: F12</b>	<b>:</b>	<b>C= + SHIFT + F3</b>
<b>: F13</b>	<b>:</b>	<b>C= + F5</b>
<b>: F14</b>	<b>:</b>	<b>C= + SHIFT + F5</b>
<b>: F15</b>	<b>:</b>	<b>C= + F7</b>
<b>: F16</b>	<b>:</b>	<b>C= + SHIFT + F7</b>
-----		

## 2.1.1 KEY

**FORMAT:** KEY N. "STRING"  
**PARAMETER:** N - NUMMER DER FUNKTIONSTASTE  
(1-16)  
STRING - ZEICHENFOLGE ODER STRING-VARIABLE (MAX. 15 ZEICHEN)  
**FUNKTION:** BELEGEN DER FUNKTIONSTASTEN MIT ZEICHENFOLGEN ODER BEFEHLEN  
**BEISPIEL:** KEY1. "LIST -100"  
BELEGT DIE TASTE F1 MIT DEM BEFEHL LIST -100. EIN RETURN MUß "VON HAND" GEGEBEN WERDEN.

Mit diesem Befehl können Sie nun Ihre eigene Funktionstasten definieren. Soll die Funktionstaste gleichzeitig das RETURN auslösen, so muß der Zeichenkette der entsprechende ASCII-Code des RETURNS (13) angehängt werden. Stellen Sie nun mit Hilfe der folgenden zwei Befehlen den Unterschied fest:

**KEY1. "PRINT 16\*15"**  
**KEY3. "PRINT 16\*16"+CHR\$(13)**

Nachdem Sie nun die zwei Funktionstasten definiert haben, betätigen Sie die Funktionstaste F1 und danach die Taste F3. Sie werden feststellen, daß die Funktionstaste F3 gleichzeitig das RETURN auslöst. Doch das ist nicht immer sinnvoll. Befehle, deren irrtümliche Betätigung schlimme Folgen haben kann (z.B. NEW) sollten ohne automatisches RETURN gespeichert werden. Sollten Sie nun einmal versehentlich die Funktionstaste für NEW betätigen, so wird der Befehl nur akzeptiert, wenn danach RETURN gedrückt wird. Sie können die Fehlbedienung also noch rechtzeitig feststellen.

Versuchen Sie nun einmal die Funktionstasten mit Befehlen zu belegen. Benutzen Sie auch das abschließende RETURN.

Damit Sie Ihre Standardbelegung nicht immer wieder mühevoll eingeben müssen, können die Funktionstasten auch in einem Programm belegt werden. Das folgende Programm belegt die Tasten F1, F3, F5 und F7:

**100 KEY1, "LIST"**  
**110 KEY3, "GOTO"**  
**120 KEY5, "PRINT"**  
**130 KEY6, "OPEN1.8.15."**

Wenn diese vier Funktionen immer wieder benötigt werden, so kann dieses Programm abgespeichert werden. Die Belegung der Funktionstasten mit diesen Befehlen ist nun recht einfach. Sie laden das Funktionstasten-Programm und starten es mit RUN. Schon stehen Ihnen die gewünschten Funktionstasten zur Verfügung.

Der KEY-Befehl hat eine Besonderheit: Befehle mit Anführungszeichen können nur auf die Funktionstasten gelegt werden, wenn für das Anführungszeichen ein CHR\$-Code benutzt wird. Eine kleine Knobelei: Versuchen Sie die Funktionstaste F1 mit dem Befehl 'LOAD"\$",8' zu belegen! Dieser Befehl lädt das Inhaltsverzeichnis der Diskette in den Speicher. Viele werden dabei sicher verzweifeln. Doch hier die Lösung:

**KEY1, "LOAD"+CHR\$(34)+"\$"+CHR\$(34)+",8"**

Dieser Befehl wirkt auf dem ersten Blick undurchsichtig, da das Anführungszeichen als CHR\$(34) eingesetzt wurde. Sie müssen also jedes gewünschte Anführungszeichen in der Zeichenkette gegen "+CHR\$(34)+" ersetzen.

Eine weitere Möglichkeit ist, die Funktionstasten mit Hilfe eines Programms direkt zu belegen. Die Zeichenfolge darf aber nicht die Zeichen die Zeichen ":" und "," enthalten, da diese vom INPUT nicht akzeptiert werden. Das folgende Programm ermöglicht die Belegung der Funktionstasten im Dialog:

```

100 INPUT"NUMMER DER FUNKTIONSTASTE: ";N
110 INPUT"ZU BELEGEN MIT: ";Z$
120 PRINT"RETURN ANHAENGEN (J/N)?"
130 GETX$:IFX$<>"N" AND X$<>"J" THEN 130
140 IFX$="J" THEN Z$=Z$+CHR$(13)
150 IF LEN(X$)<16 THEN 170
160 PRINT"MAXIMAL 15 ZEICHEN !":GOTO 180
170 KEY N.Z$
180 PRINT"WEITERE BELEGUNGEN (J/N)?"
190 GETX$:IFX$<>"N" AND X$<>"J" THEN 170
200 IF X$="J" THEN 100
210 END

```

Dieses Programm vereinfacht die Belegung der Funktionstasten nach dem Einschalten des Rechners. Es akzeptiert die Eingabe des Anführungszeichen, jedoch nicht Komma und Doppelpunkt. Ein RETURN wird auf Wunsch angehängen. Die Länge der Zeichenfolge darf inklusive RETURN 15 nicht überschreiten.

Ein Tip: Wenn Sie die Funktionstaste 1 mit 'KEY1,"?FRE(0)" +CHR\$(13)' belegen, erhalten Sie "auf Knopfdruck" den aktuellen, freien Speicherplatz.

#### **HINWEIS:**

Im Handbuch zu SIMON'S BASIC ist keine Möglichkeit beschrieben, eine Funktionstaste zu löschen. Gibt man z.B. den Befehl 'KEY1,""' ein, so wird nicht etwa die Funktionstaste gelöscht, sondern der bisherige Inhalt mit undefinierbaren Zeichen aufgefüllt! Doch verfallen Sie deshalb nicht in Wutausbrüche, Sie können eine Funktionstaste z.B. mit 'KEY1,CHR\$(0)' löschen!

## 2.1.2 DISPLAY

**FORMAT: DISPLAY**  
**PARAMETER: KEINE**  
**AUFGABE: BELEGUNG DER FUNKTIONSTASTEN ANZEIGEN**  
**BEHMERKUNG: ANSTATT 'DISPLAY' KANN AUCH 'KEY 0'**  
**EINGEGEBEN**  
**WERDEN!**

Die Belegung der Funktionstasten ist zwar eine große Erleichterung, jedoch kann der Überblick über deren Belegung leicht verloren gehen. Eine Situation wie "habe ich nun den LIST-Befehl auf Taste F1 oder auf Taste F3 gelegt?" ist bei 16 belegbaren Funktionstasten nicht abzuwenden. Damit Sie nicht mehrere Funktionstasten ausprobieren müssen, um den gewünschten Erfolg zu erzielen, bietet Ihnen SIMON'S BASIC eine enorme Erleichterung zum Auffinden von in Vergessenheit geratenen Funktionstasten an:

Der DISPLAY-Befehl zeigt Ihnen alle Funktionstastenbelegungen auf dem Bildschirm an. Dem Inhalt der Funktionstasten ist der entsprechende KEY-Befehl vorangestellt. Ein DISPLAY nach dem Laden von SIMON'S BASIC bewirkt die folgende Ausgabe:

```
KEY1. ""  
KEY2. ""  
KEY3. ""  
KEY4. ""  
KEY5. ""  
KEY6. ""  
KEY7. ""  
KEY8. ""  
KEY9. ""  
KEY10. ""  
KEY11. ""  
KEY12. ""  
KEY13. ""  
KEY14. ""  
KEY15. ""  
KEY16. ""
```

Nach dem Start von SIMON'S BASIC sind also alle Funktionstasten "leer". Die DISPLAY-Bildschirmausgabe kann natürlich editiert werden, d.h. möchten Sie die ein oder andere Funktionstaste umdefinieren, so bewegen Sie nach dem DISPLAY den Cursor auf die entsprechende Zeile, ändern den KEY-Befehl und drücken RETURN. Diese Eingabehilfe verwendeten Sie sicher auch beim Erfassen und Ändern von Programmen. Hier können Sie auch nach dem LIST die gewünschten Zeilen auf diese Art ändern. Man erspart sich also die erneute Eingabe der Zeile.

Natürlich können Sie die vom DISPLAY bewirkte Ausgabe auch auf den Drucker "lenken". Erfahrene CBM 64-Anwender werden nun natürlich schmunzeln und sagen "natürlich, das ist doch kein Problem". Aber auch der Unerfahrene soll in diesem Buch auf seine Kosten kommen.

Öffnen Sie zunächst den Druckerkanal mit 'OPEN 1,4'. Danach leiten Sie die folgende Bildschirmausgabe mit 'CMD1' auf diesen Kanal. Der anschließenden 'DISPLAY' druckt Ihnen die Liste der Funktionstastenbelegung aus. Nun schließen Sie den Druckerkanal mit CLOSE1.



### 2.1.3 TESTAUFGABEN

1) Wie bekannt, werden Funktionstasten mit KEY belegt. Einer der folgenden Befehle ist nicht korrekt:

- A) KEY1, "LIST"
  - B) KEY0, "RUN"
  - C) KEY B, A\$
- 

2) Der Funktionstaste 5 soll der Befehl 'LIST' zugeordnet werden. Es soll automatisch ein RETURN ausgelöst werden. Welcher der folgenden Befehle ist dazu richtig?

- A) KEY5, "LIST"+"RETURN"
  - B) KEY5, "LIST"+CHR\$(34)
  - C) KEY5, "LIST"+CHR\$(13)
- 

3) Der Befehl 'LOAD"\$",8' soll mit automatischem RETURN auf die Funktionstaste 1 gelegt werden. Welcher Befehl erfüllt diese Aufgabe?

- A) KEY1, "LOAD"\$",8"+CHR\$(13)
  - B) KEY1, "LOAD"\$",8"+CHR\$(34)
  - C) KEY1, "LOAD"+CHR\$(34)+"\$"+CHR\$(34)+"",8"+CHR\$(13)
  - D) KEY1, "LOAD"+CHR\$(13)+"\$"+CHR\$(13)+"",8"+CHR\$(34)
- 

4) Mit wieviel Zeichen kann eine Funktionstaste maximal belegt werden?

- A) 255
  - B) 15
  - C) 16
  - D) 256
-

## 2.2 KOMFORTABLE PROGRAMMERSTELLUNG

### 2.2.1 AUTO

**FORMAT:** AUTO SN.SW

**PARAMETER:** SN - STARTNUMMER DER NUMERIERUNG  
SW - SCHRITTWEITE DER NUMERIERUNG  
(1-255)

**FUNKTION:** AUTOMATISCHE ZEILENNUMERIERUNG MIT  
VARIABLEN  
STARTNUMMER UND SCHRITTWEITE

**BEISPIEL:** AUTO 100.10  
DIE ANFANGSZEILE DES FOLGENDEN PRO-  
GRAMMS 100 UND DIE SCHRITTWEITE 10  
BETRAGEN

Wie allseits bekannt, werden BASIC-Programmzeilen mit Zeilennummern versehen, die vom Interpreter sequentiell ausgeführt werden. Es ist zwar möglich, mit der Schrittweite 1 zu numerieren, jedoch können dann keine Zeilen mehr eingefügt werden. Deshalb sollte man eine angemessene Schrittweite wählen. Meistens wird mit einer Schrittweite von 10 gearbeitet. Wenn Ihnen die selbständige Eingabe der Nummer vor jeder Zeile mißfällt, ist dieser Befehl angebracht. Er gibt Ihnen jeweils die Zeilennummer vor, deren Inhalt Sie dann anschließen. Nach Betätigung von RETURN, mit dem Sie die Eingabe der aktuellen Zeile anschließen, wird die nächste Zeilennummer ausgegeben, usw.

Betätigen Sie nach der vorgegebenen Zeilennummer die Taste RETURN, wird der AUTO wieder aufgehoben und Sie können normal weiterarbeiten.

Beachten Sie jedoch, daß die AUTO-Funktion das Ende der Zeilennumerierung (63999) nicht erkennt. Testen Sie dies mit folgendem Beispiel:

**AUTO 63500. 100**  
**63500 PRINT**  
**63600 PRINT**  
**63700 PRINT**  
**63800 PRINT**  
**63900 PRINT**  
**64000 PRINT**

**?SYNTAX ERROR**  
**READY.**  
**64100**

**?SYNTAX ERROR**  
**READY.**

Das Erreichen der Numerierungsgrenze wird nicht ganz "sauber" abgefangen. Die Eingabe der Zeilennummer 64000 verursacht ein ?SYNTAX ERROR und die Zeilennummer 64100 wird vorgegeben. Wollen Sie nun die AUTO-Funktion mit RETURN verlassen, so wird ebenfalls vorher ein ?SYNTAX ERROR ausgegeben. Doch da diese Grenze in den seltensten Fällen erreicht wird und dies auch keine schlimmen Folgen hat, sollten Sie dies lediglich zur Kenntnis nehmen.

Weitere Schwierigkeiten können auftreten, wenn Sie bereits Zeilen eingegeben haben und einen AUTO-Befehl absetzen, der diese Zeilennummern vorgibt. Wenn Sie die AUTO-Funktion nun verlassen, so ist die vorgegebene Zeile gelöscht. Ein Beispiel soll dies verdeutlichen:

Angenommen, Sie haben das folgende Programm eingegeben:

```
100 FOR I=65 TO 90
110 PRINT"DEM ASCII-WERT ";
120 PRINT I;
130 PRINT" IST DAS ZEICHEN ";
140 PRINT CHR$(I);
150 PRINT" ZUGEORDNET."
160 NEXT I
```

Sie haben das Programm aufgelistet und möchten nun weitere Zeilen ab 200 erfassen. Der entsprechende Befehl wäre 'AUTO 200,10'. Nun machen Sie einen Eingabefehler und geben 'AUTO 100,10'. Der Rechner reagiert und gibt die Zeile 100 vor. Nun erkennen Sie den Eingabefehler und verlassen die Funktion mit RETURN. Die Zeile 100 ist nun gelöscht.

Der einzige Ausweg ist der "Notausgang", also die Betätigung der Tasten RUN/STOP und RESTORE gleichzeitig, wenn die Zeile 100 ausgegeben wird.

Haben Sie das Problem erkannt? Wenn ja, dann achten Sie stets auf korrekte Eingabe des AUTO-Befehls, wenn sich bereits ein Programm im Speicher befindet.

### 2.2.2 RENUMBER

**FORMAT:** RENUMBER SN,SW

**PARAMETER:** SN - ERSTE ZEILE DER NEUEN NUMERIERUNG  
SW - SCHRITTWEITE DER NEUEN NUMERIERUNG

**FUNKTION:** NEUE NUMERIERUNG VON PROGRAMMEN MIT  
VARIABLER ANFANGSZEILE UND SCHRITTWEITE

**BEISPIEL:** RENUMBER 100,10  
DAS IM SPEICHER BEFINDLICHE PROGRAMM  
WIRD AB DER ZEILE 100 IN 10ER SCHRITTEN  
NEU DURCHNUMERIERT

**BEMERKUNG:** SPRUNGADRESSEN DER GOTO- UND GOSUB-BE-  
FEHLE WERDEN DER NEUEN NUMERIERUNG NICHT  
ANGEPASST!

Wenn die bisherige Numerierung der BASIC-Zeilen unübersichtlich erscheint und an vielen Stellen durch Einfügung neuer Zeilen zu "eng" geraten ist, so hilft dieser Befehl weiter. Er numeriert das komplette Programm neu, wobei die Startzeile und die Schrittweite frei wählbar ist. "Fantastisch" werden viele von Ihnen denken, doch die Möglichkeiten, die Sie sich mit diesem Befehl ausrechnen, werden schnell gebremst. Was nutzt Ihnen eine komplette Neunumerierung, wenn die Sprungadressen der GOTO- und GOSUB-Befehle nicht berücksichtigt, also nicht geändert werden? Die mangelnde Strukturierungsmöglichkeiten des Standard-BASIC machen es unmöglich, ohne GOTO- und GOSUB zu arbeiten.

Wenn Sie jedoch die Möglichkeiten zur Strukturierung Ihrer Programme ausnutzen, die Ihnen SIMON'S BASIC bietet, werden Sie diesen Befehl effektiv einzusetzen wissen. Mit SIMON'S BASIC ist es z.B. möglich, symbolische Adressen zu vergeben. So kann ein herkömmlicher Befehl wie 'GOTO 100' zu 'GOTO AUSGABE' verwandelt werden. Die Vergabe von symbolischen Adressen bei sämtlichen GOTO- und GOSUB-Befehlen ist also nicht zu vermeiden, wenn Sie RENUMBER einsetzen. Näheres zur Vergabe von symbolischen Sprungadressen finden Sie in Kapitel 5.2.

Erproben Sie nun einmal diesen Befehl mit Hilfe des folgenden Programms:

```
10 PRINT"10"  
11 PRINT"11"  
12 PRINT"12"  
15 PRINT"15"  
20 PRINT"20"
```

Geben Sie diese wenigen Zeilen nun ein. Die Numerierung ist unregelmäßig, und zwischen den Zeilen 10 und 11 oder 11 und 12 kann nichts mehr eingefügt werden. In so einer Situation ist es sinnvoll, das Programm neu zu numerieren. Geben Sie dazu den Befehl 'RENUMBER 10,10' ein. Wenn Sie danach das Programm mit 'LIST' auf dem Bildschirm anzeigen, sieht die Numerierung folgendermaßen aus:

```
10 PRINT"10"  
20 PRINT"11"  
30 PRINT"12"  
40 PRINT"15"  
50 PRINT"20"
```

Nun ist die Numerierung wieder "sauber" und Sie können an allen Stellen Zeilen einfügen.

Da wir zur Erstellung dieses Buches jeden Befehl "auf Herz und Nieren" geprüft haben, stießen wir auch hier auf eine Besonderheit: Eine Schrittweite von 0 bewirkt, daß alle Zeilen mit der gleichen Nummer versehen werden. Ein derartig manipuliertes Programm ist sogar lauffähig. Um sich davon zu überzeugen, geben Sie nach Eingabe des Programms zum letzten Beispiel den Befehl 'RENUMBER 10,0' ein. Der anschließende Befehl 'LIST' wird Ihnen ein ungewohntes Listing auf dem Bildschirm bringen:

```
10 PRINT"10"  
10 PRINT"11"  
10 PRINT"12"  
10 PRINT"15"  
10 PRINT"20"
```

Überzeugen Sie sich von der Lauffähigkeit dieses Programm, indem Sie es mit 'RUN' starten. Wenn Sie diese Zeilen nun einzeln löschen wollen, geben Sie jeweils '10' mit abschließendem RETURN ein. Die Zeile 10 wird dann gelöscht. Aber welche? Die erste Zeile 10 (PRINT"10") wird gelöscht. Wenn Sie nun nochmals die Zeile 10 löschen, wird die Zeile 'PRINT"11"' gelöscht usw. Auf diese Weise können Programme gegen Eingriffe geschützt werden, da keine Zeilen mehr geändert oder eingefügt werden können. Wenn Sie z.B. die Zeile 'PRINT"15"' in 'PRINT"17"' ändern wollen, so wird diese Zeile nicht geändert, sondern die Zeile 'PRINT"10"' gegen die Zeile 'PRINT"17"' ausgetauscht. Das Programm ist also nicht mehr editierfähig.

### Noch eine Besonderheit:

Wenn Sie einen RENUMBER-Befehl eingeben, der dazu führt, daß die maximale Zeilennummer 63999 überschritten wird, so können diese Zeilen weder angesprungen, noch gelöscht werden. Der Beweis: Numerieren Sie das eben beschriebene Programm mit 'RENUMBER 63800,100'. Nach LIST finden Sie die folgenden Zeilen auf dem Bildschirm vor:

**63800 PRINT"10"**

**63900 PRINT"11"**

**64000 PRINT"12"**

**64100 PRINT"15"**

**64200 PRINT"20"**

Die erlaubte Zeilennummerierung darf aber 63999 nicht überschreiten! Der einfache Beweis: Versuchen Sie einmal eine Zeile 64000 einzugeben! Doch hier wird wieder gegen alle Regeln verstoßen. Das Programm ist zwar lauffähig, jedoch können die Zeilen ab 64000 nicht mehr gelöscht werden! Versuchen Sie es einmal mit der Eingabe '64000' und anschließendem RETURN. Es erscheint die Meldung '?SYNTAX ERROR'. Auch können die Zeilen ab 64000 nicht mit GOTO oder GOSUB angesprochen werden. Versuchen Sie einmal die Eingabe 'GOTO 64100' mit anschließendem RETURN. Der Interpreter meldet wiederum '?SYNTAX ERROR'. Dieses Programm kann nur noch ein "sauberer" RENUMBER in Ordnung bringen.

### 2.2.3 MERGE

**FORMAT:** MERGE "NAME".GN  
**PARAMETER:** NAME - NAME DES PROGRAMMS, DAS DAZU GELADEN  
WERDEN SOLL  
GN - NUMMER DES GERÄTS, VON DEM GELADEN  
WERDEN SOLL (1=DATASETTE; 8=FLOPPY)  
**FUNKTION:** HINZULADEN EINES EXTERN GESPEICHERTEN  
PROGRAMMS  
**BEISPIEL:** MERGE "EING-ROUT".8  
DAS PROGRAMM "EING-ROUT" SOLL VON DER FLOPPY  
GELADEN UND AN DAS BEREITS IM SPEICHER  
BEFINDLICHE PROGRAMM ANGEHÄNGT WERDEN

Wer viel in BASIC programmiert, entwickelt individuelle Routinen (Unterprogramme), die in mehreren Programmen Verwendung finden. Diese Routinen müssen jedoch verwaltet werden. Eine Möglichkeit ist, die Routinen auszudrucken und bei Gebrauch vom Papier abzuschreiben. Doch das ist sehr umständlich. Sinnvoller ist es, die Routinen auf Diskette zu speichern und mit dem Hauptprogramm zu verknüpfen. Doch dies ist leichter gesagt als getan. Das Betriebssystem des CBM 64 bietet keine Möglichkeit, Programme ohne aufwendige POKES zu verknüpfen.

Der MERGE-Befehl ermöglicht Ihnen nun, häufig benötigte Programmteile auf Diskette abzulegen und bei Bedarf dem im Speicher befindlichen Programm anzuhängen. Doch dabei müssen drei Dinge beachtet werden:

1. Zeilennummern des im Speicher befindlichen und des nachzuladenden Programms dürfen nicht identisch sein.
2. Die Zeilennummern des nachgeladenen Programmes müssen höher als die des im Speicher befindlichen Programmes sein.



3. Die Variablenbelegung muß unterschiedlich sein. Gleiche Variablen in unterschiedlichen Programmteilen können später die Ursache für viele, nur schwer zu lösende Fehler sein.

Der beste Weg, Probleme beim MERGEN von Programmen zu vermeiden, ist eine klare, einheitliche Strukturierung ALLER Ihrer Programme. Weisen Sie jedem häufiger benutzten Unterprogramm einen ein für alle Mal festgelegten Zeilennummernbereich zu, in dem Sie sonst grundsätzlich nicht programmieren. Ebenso verfahren Sie bitte bei den Variablen. Sicher fragen Sie sich, welche Auswirkungen es hat, wenn Sie diese drei Regeln zum MERGE-Befehl mißachten. Da Probieren bekanntlich über Studieren geht, wenden wir im folgenden Beispiel den MERGE-Befehl rücksichtslos an:

Geben Sie zunächst das nachzuladene Programm ein:

```
10 PRINT"ZEILE 10. NACHGELADEN"  
20 PRINT"ZEILE 20. NACHGELADEN"  
30 PRINT"ZEILE 30. NACHGELADEN"
```

Speichern Sie dieses nachzuladene Programm mit ´SAVE "TEST.MERGE",8´ auf der Floppy oder mit ´SAVE "TEST"´ auf Kassette ab. Nun löschen Sie dieses im Speicher befindliche Programm mit dem Befehl ´NEW´. Da wir das eben abgespeicherte Programm hinzuladen möchten, reizen wir die Karten aus und erfassen das Hauptprogramm mit den gleichen Zeilennummern:

```
10 PRINT"ZEILE 10. HAUPTPROGRAMM"  
20 PRINT"ZEILE 20. HAUPTPROGRAMM"  
30 PRINT"ZEILE 30. HAUPTPROGRAMM"
```

Nun wollen wir sehen, wie SIMON'S BASIC reagiert, wenn wir das eben abgespeicherte Programm, das bekanntlich die gleichen die gleichen Zeilennummern enthält, nachgeladen wird. Geben Sie dazu diesen Befehl ein:

**MERGE "TEST.MERGE".8**

oder

**MERGE "TEST.MERGE"**

Die Geräteadresse 8 müssen Sie nur angeben, wenn Sie von der Diskettenstation laden. Nachdem der Rechner mit 'READY' zurückkommt, zeigt Ihnen der Befehl 'LIST', was Sie soeben fabriziert haben:

```
10 PRINT"ZEILE 10. HAUPTPROGRAMM"  
20 PRINT"ZEILE 20. HAUPTPROGRAMM"  
30 PRINT"ZEILE 30. HAUPTPROGRAMM"  
10 PRINT"ZEILE 10. NACHGELADEN"  
20 PRINT"ZEILE 20. NACHGELADEN"  
30 PRINT"ZEILE 30. NACHGELADEN"
```

Dieses "Programm" ist ein seltener Anblick für alle, die nicht erst seit zwei Tagen programmieren. Alle Regeln zum Aufbau eines BASIC-Programmes sind mißachtet, jede Zeile ist doppelt vorhanden. Prüfen wir nun einmal ob der BASIC-Interpreter diese Unordnung verträgt. Starten Sie dazu das Programm mit 'RUN'. Der Erfolg ist verblüffend: Der Interpreter winkt nicht mit der weißen Fahne, sondern arbeitet das Programm ohne zu meckern ab. Um dies zu erklären, muß etwas weiter ausgeholt werden. Nach dem Befehl 'RUN' wird zunächst die Adresse (Position) der ersten BASIC-Zeile ermittelt. Diese Adresse befindet sich in der sogenannten ZERO-PAGE, die ersten 256 Bytes des Hauptspeichers. Hier werden Informationen abgespeichert, die das Betriebssystem oder der BASIC-Interpreter in seinen Routinen benötigt. Dies ist vergleichbar mit einem BASIC-Programm, das wichtige Informationen in Variablen ablegt. Die ZERO-PAGE ist also der "Variablenbereich" von Betriebssystem und Interpreter. Die Adresse der ersten BASIC-Zeile (beim CBM 64 stets \$0801 oder dezimal 2049) wird zuerst ermittelt und in der ZERO-PAGE als aktuelle Zeile abgelegt. Nachdem Zeile nun abgearbeitet wurde, wird die Adresse der nächsten Zeile als aktuelle Zeile gespeichert. Die Adresse der nächsten Zeile befindet sich am Anfang jeder BASIC-Zeile und wird beim Befehl 'LIST' nicht mit angezeigt.

Die Verkettung von Zeile zu Zeile wird also nicht durch die eigentliche Zeilennummer, sondern durch eine spezielle Folgeadresse hergestellt. Die Zeilennummer wird dann beim Programmablauf nur bei Sprungbefehlen (GOTO, GOSUB) benötigt. Hier wird die anzuspringende Zeile zur aktuellen Zeile erklärt. Das Ende des Programms wird an der Folgeadresse 0 erkannt. Ein Programm mit mehreren gleichen Zeilennummern steigt somit erst bei Sprungbefehlen aus.

Da jede Zeile doppelt existiert, ist es nun interessant festzustellen, wie auf ein 'GOTO 10' reagiert wird. Wir die erste Zeile 10 angesprochen, die Zweite oder ...? Überzeugen Sie sich selbst: Geben Sie den Befehl 'GOTO 10' ein und Sie werden feststellen, daß stets die erste Zeile angesprochen wird. Dies ist damit zu begründen, daß der Interpreter bei der Suche nach der angesprungenen Zeile stets am Programmumfang beginnt.

Ein derartiges, ungeordnetes Programm sollten Sie natürlich nicht einsetzen. Verwenden Sie den RENUMBER-Befehl, um hier Ordnung zu schaffen: 'RENUMBER 10,10'. Das Ergebnis, das Sie mit 'LIST' betrachten können, sieht nun folgendermaßen aus:

```
10 PRINT"ZEILE 10. HAUPTPROGRAMM"  
20 PRINT"ZEILE 20. HAUPTPROGRAMM"  
30 PRINT"ZEILE 30. HAUPTPROGRAMM"  
40 PRINT"ZEILE 10. NACHGELADEN"  
50 PRINT"ZEILE 20. NACHGELADEN"  
60 PRINT"ZEILE 30. NACHGELADEN"
```

#### Beachten Sie:

In der ZERO-PAGE wird nicht nur der Anfang des BASIC-Programms gespeichert, sondern auch das Ende. Wird das BASIC-Ende in einem Programm mit POKE-Befehlen hochgesetzt, so kann ein MERGE Probleme bereiten. Für Programmierer, die derartige Experimente nicht durchführen, trifft dies natürlich nicht zu. Sollten Sie aber durch Zufall diesen BASIC-Ende Zeiger manipulieren, sodaß ein MERGE wirkungslos ist, kann mit den Befehlen 'NEW' und 'OLD' das BASIC-Ende wieder angepasst werden.

## 2.2.4 TESTAUFGABEN

- 1) Die automatische Zeilennummerierung mit der Anfangszeile 100 und der Schrittweite 10 soll eingeschaltet werden. Welcher Befehl bewirkt dies?
- A) AUTO 100,10
  - B) AUTO 10,100
  - C) AUTO 100 BY 10
  - D) AUTO 10 BY 100
- 
- 2) Die Schrittweite des AUTO-Befehls ist begrenzt auf:
- A) 255
  - B) 256
  - C) unbeschränkt
- 
- 3) Welche Befehle werden vom RENUMBER-Befehl nicht ordnungsgemäß behandelt?
- A) FOR / NEXT - Schleifen
  - B) GOTO
  - C) GOSUB
  - D) INPUT
- 
- 4) Was muß bei der ordnungsgemäßen Anwendung des MERGE-Befehls beachtet werden?
- A) Die Zeilennummern müssen größer als 100 sein.
  - B) Es dürfen keine identischen Zeilennummern auftreten.
  - C) Die Zeilennummern des nachgeladenen Programms müssen größer sein als die letzte Zeile des vorhandenen Programms.
  - D) Vor Anwendung von MERGE muß das Programm mit dem Befehl RENUMBER neu durchnummeriert werden.
-

## 2.3 LISTHILFEN

Zum Bearbeiten eines BASIC-Programms ist der Einblick ins Listing unabwendbar. Die beste Übersichtlichkeit bietet ein ausgedrucktes BASIC-Listing. Doch nicht Jeder verfügt über einen Drucker. Eine Druckerausgabe ist auch nicht immer sinnvoll. Wenn Sie z.B. "mal eben" etwas ändern möchten, so ist es sehr zeitraubend, das gesamte Programm auf dem Drucker auszugeben.

Das BASIC-Programm soll also auch auf dem Bildschirm übersichtlich erscheinen. Bei der Ausgabe von BASIC-Listings auf dem Bildschirm ist Ihr CBM 64 recht unkomfortabel. Sie können lediglich mit dem Befehl 'LIST' das Programm auflisten, die LIST-Geschwindigkeit mit der Taste 'CTRL' steuern und an gegebener Stelle mit der Taste 'RUN/STOP' abbrechen. Möchten Sie nun ab der abgebrochenen Zeilennummer weiterlisten, so ist wieder ein 'LIST'-Befehl notwendig. Im Standard-BASIC gibt es folgende Möglichkeiten zum Einsatz des 'LIST'-Befehls:

LIST           - das gesamte Programm wird ausgegeben  
LIST n         - eine bestimmte Zeile wird ausgegeben (LIST 100)  
LIST -n        - es wird bis zur Zeile n ausgegeben (LIST -1000)  
LIST n-        - es wird ab der Zeile n ausgegeben (LIST 500-)  
LIST n1-n2     - Die Zeilen n1 bis n2 werden ausgegeben.  
              (LIST 500-1000)

Mit SIMON'S BASIC haben Sie nun weitere Möglichkeiten, die Auflistung Ihres Programms zu steuern.

### 2.3.1 PAGE

**FORMAT:** PAGE N  
**PARAMETER:** N - ANZAHL DER GEWÜNSCHTEN ZEILEN PRO  
ABSCHNITT MINUS 1 (1-23)  
**FUNKTION:** TEILEN DES PROGRAMMLISTINGS IN ABSCHNITTEN  
**BEISPIEL:** PAGE 10  
DIE FOLGENDE LIST-AUSGABE DES PROGRAMMS ER-  
FOLGT ZU ABSCHNITTEN VON 11 ZEILEN  
**BEMERKUNG:** DER BEFEHL PAGE 0 SETZT DIE PAGE-FUNKTION  
WIEDER ZURÜCK

Mit Hilfe dieses Befehls teilen Sie nun Ihr BASIC-Programm in Abschnitte auf. Die Größe der Abschnitte ist frei wählbar, darf aber eine Bildschirmseite nicht überschreiten. Zu beachten ist hier, daß es sich bei der Angabe der Zeilen um Bildschirmzeilen, nicht um Programmzeilen handelt.

Bei diesem Befehl haben wir folgendes zu bemängeln: Wählen Sie einen Wert über 23, so wird keine PAGE-Funktion ausgelöst. Sinnvoll wäre es hier, wenn die Fehlermeldung, die bei einem Wert von  $n > 255$  ausgegeben wird (ILLEGAL QUANTITY ERROR) auch bei einem Wert  $n > 23$  aktiv wird, der ohnehin wirkungslos ist.

Der Ablauf zum Einsatz dieses Befehl ist folgender:

1. Sie geben einen entsprechenden PAGE-Befehl mit der gewünschten Abschnittsgröße ein.
2. Sie starten die Ausgabe mit 'LIST'
3. Sie "blättern" mit der Taste RETURN
4. Sie unterbrechen die Ausgabe bei Bedarf mit der Taste 'RUN/STOP'

Die Unterbrechung der Ausgabe ist nicht ganz so einfach, wie es scheint. Zwar können Sie die LIST-Ausgabe mit der Taste 'RUN/STOP' unterbrechen, doch nur während der Ausgabe, nicht wenn die gesamten Zeilen des Abschnittes bereits auf dem Bildschirm enthalten sind und der Rechner auf das RETURN zur Ausgabe des nächsten Abschnittes wartet. Probieren Sie dies einmal aus. Laden Sie eines Ihrer Programme und geben Sie anschließend den Befehl 'PAGE 10' ein. Wenn Sie nun mit

'LIST' die Ausgabe des ersten Abschnittes einleiten, erscheint zunächst die erste Zeilennummer des Programmes auf dem Bildschirm. Drücken Sie nun RETURN, so wird der Bildschirm gelöscht und der erste Abschnitt des Programms angezeigt. Nun können Sie die Ausgabe aber nicht unterbrechen. Sie müssen dazu nochmals die RETURN-Taste drücken und noch während der folgenden Ausgabe die 'RUN/STOP'-Taste betätigen.

Mit der PAGE-Funktion kann natürlich auch jede der Variationen des LIST-Befehls eingesetzt werden, die zu Anfang beschrieben wurden.

### 2.3.2 OPTION

**FORMAT:**        **OPTION N**  
**PARAMETER:**   **N - 10 ZUM EINSCHALTEN; UNGLEICH 10. ABER**  
                  **KLEINER 256 ZUM AUSSCHALTEN**  
**FUNKTION:**     **BEIM LISTEN DES PROGRAMMS WERDEN ALLE**  
                  **SIMON'S BASIC-BEFEHLE REVERSE DARGE-**  
                  **STELLT. ALSO HERVORGEHOVEN**  
**BEISPIEL:**     **OPTION 10 - SCHALTET KENNZEICHNUNG EIN**  
                  **OPTION 0 - SCHALTET KENNZEICHNUNG AUS**

Das Hervorheben aller SIMON'S BASIC-Befehle kann durchaus sinnvoll sein. Nehmen wir z.B. einmal an, ein vorhandenes Programm, das mit Hilfe von SIMON'S BASIC geschrieben wurde, soll in STANDARD-BASIC umgeschrieben werden. Hier ist es sehr zeitaufwendig, das gesamte Programm nach diesen Befehlen zu durchsuchen. Wenn Sie diese Befehle jedoch mit 'OPTION 10' kenntlich machen, so wird das Auffinden der Befehle zum Kinderspiel.

Ein Beispiel: Laden Sie ein Programm, das mit SIMON'S-BASIC Befehlen bestückt ist. Vielleicht haben Sie das in Kapitel 2.1.1 enthaltene Programm zur Belegung der Funktionstasten eingegeben. Wenn Sie nun 'OPTION 10' und anschließend 'LIST' eingeben, sehen Sie die Auswirkung des OPTION-Befehls: Sämtliche Befehle des SIMON'S BASIC sind reverse dargestellt. Die Befehle werden auf allen Druckern, die einen Commodore-Modus besitzen, ebenfalls reverse ausgegeben.

### 2.3.3 DELAY

**FORMAT:** DELAY N  
**PARAMETER:** N - VERZÖGERUNGSWERT (1-255)  
**FUNKTION:** VERÄNDERN DER LIST-GESCHWINDIGKEIT MIT  
VARIABLEM VERZÖGERUNGSWERT  
**BEISPIEL:** DELAY 10  
DIE FOLGENDE LISTAUSGABE DES PROGRAMMS WIRD  
MIT SHIFT AUF CA. 50 ZEICHEN/SEK VERZÖGERT

Neben der Standard-Verzögerung des Listings mit der CTRL-Taste kann nun mit der SHIFT-Taste eine variable Verzögerung gesteuert werden. Der Verzögerungswert kann mit dem DELAY-Befehl eingestellt werden. Wenn Sie die Commodore-Taste (C=) gedrückt halten, wird die Listausgabe gestoppt.

Damit Sie ein Gefühl für den Verzögerungswert erhalten, folgt nun eine Übersicht:

VERZÖGERUNGSWERT	ZEICHEN/SEK.
1	290
5	100
10	50
20	25

Der Wert 1 entspricht etwa der normalen Listgeschwindigkeit und der Wert 10 der Verzögerung mit der CTRL-Taste.

Die PAGE- und die DELAY-Funktion können miteinander kombiniert werden. D.h. Sie können Ihr Listing in Abschnitte aufteilen, deren Anzeige Sie mit der SHIFT-Taste verzögern können. Sicher haben Sie bald Ihre persönlichen PAGE- und DELAY-Werte ermittelt, die Sie dann z.B. jedem Programm als erste Zeile voranstellen können. Diese Zeile sieht dann z.B. so aus:

**1 PAGE 10:DELAY 5**

Wenn Sie dann das Programm nach dem Laden Starten steht Ihnen nach einer Unterbrechung Ihre persönlich ermittelte Listhilfe zur Verfügung.



## 2.3.4 FIND

**FORMAT:** FIND CODE ODER FIND STRING  
**PARAMETER:** CODE - ALLES, WAS SICH AUßERHALB VON ANFÜHRUNGSZEICHEN BEFINDET  
STRING - STRINGS, ALSO ZEICHEN INNERHALB VON ANFÜHRUNGSZEICHEN  
**FUNKTION:** DAS GESAMTE BASIC-PROGRAMM WIRD NACH BESTIMMTEN ZEICHENFOLGEN DURCHSUCHT  
**BEISPIEL:** FINDPRINT  
GIBT ALLE ZEILEN AUS, IN DENEN EIN PRINT-BEFEHL ENTHALTEN IST  
**BEMERKUNG:** HINTER DEM FIND-BEFEHL DARF NUR EIN LEERZEICHEN STEHEN. WENN DIESES AUCH GESUCHT WERDEN SOLL

Diesen Befehl werden die wenigsten SIMON'S BASIC Anwender jemals effektiv eingesetzt haben. Der Grund dafür ist, daß der Befehl im mitgelieferten Handbuch auf 11 Zeilen beschrieben wird, für eine intensive Beschreibung aber fast 11 Seiten notwendig sind!

Was kann man nun mit diesem Befehl anfangen? Bevor nun alles eingehend beschrieben wird, beachten Sie die folgende wichtige Besonderheit dieses Befehls: Ihnen ist sicherlich bekannt, daß in BASIC Leerzeichen zwischen Befehl und Parameter keine Auswirkung haben. So haben z.B. die Befehle 'PRINT10' und 'PRINT 10' die selbe Auswirkung. Der FIND-Befehl bildet hier eine Ausnahme: Wenn Sie zwischen Befehl und Parameter ein Leerzeichen eingeben, so wird dieses Leerzeichen mitgesucht. Ein Beispiel: 'FIND10' würde z.B. eine Zeile mit dem Code 'PRINT10' anzeigen, nicht jedoch 'FIND 10'.

Wie Sie bereits der Parameterbeschreibung entnommen haben, gibt es zwei verschiedene Arten zu suchen: Die Suche nach BASIC-Code und die Suche nach Strings. Um den Unterschied zu erkennen, geben Sie nun das folgende Programmbeispiel ein:

```
10 PRINT "GEBEN DEN WERT A EIN"  
20 INPUT A  
30 PRINT "SIE HABEN":A;"EINGEGEBEN"
```

Es sollen alle Zeilen angezeigt werden, die ein 'A' enthalten. Der entsprechende Befehl ist

### **FINDA**

Nun die große Preisfrage: Welche Zeilen werden angezeigt? Damit Sie Ihren Kopf nicht zu sehr beanspruchen müssen, folgt nun die Antwort:

Die Zeilen 20 und 30 werden angezeigt

Zwar könnte man nun eine weitere Preisfrage nach dem WARUM starten, doch sollen Sie nicht die Befehle selbst erforschen. Das haben wir bereits für Sie gemacht. Die Zeile 10 wird nicht angezeigt, da hier zwar ein 'A' enthalten ist, aber in einem String (innerhalb von Anführungszeichen). Lassen Sie uns weiter experimentieren. Geben Sie den Befehl

### **FIND A**

ein und Sie werden feststellen, daß nur die Zeile 20 angezeigt wird, da dies die einzige Zeile ist, die im BASIC-Code ein Leerzeichen gefolgt von einem 'A' enthält. Das 'A' in der Zeile 30 hat kein führendes Leerzeichen.

Ein weiteres Beispiel: Der Befehl 'FINDPRINT' zeigt die Zeilen 10 und 30 an, da hier ein 'PRINT' enthalten ist. Der Befehl 'FIND PRINT' jedoch zeigt keine Zeile an, da das Leerzeichen zwischen Zeilennummer und Zeileninhalt vom BASIC-Interpreter selbstständig gesetzt wird und kein Leerzeichen im eigentlichen Sinne darstellt. Selbstverständlich können Sie zum Aufsuchen von BASIC-Befehlen auch deren Abkürzungen verwenden. So ist z.B. 'FINDPRINT' identisch mit 'FIND?'

Nun kann natürlich auch jedes beliebige Sonderzeichen gesucht werden. Der Befehl 'FIND;' z.B. sucht nach allen Semikolons. Dies sind alle wesentlichen Bemerkungen zur Suche von BASIC-Code.

Das Suchen nach Strings ist nicht so vielseitig wie die Suche nach BASIC-Code. Wenn Sie z.B. in unserem Beispiel

feststellen möchten, in welcher Zeile 'GEBEN SIE DEN WERT A EIN' enthalten ist, so ist dazu der Befehl 'FIND"GEBEN SIE DEN WERT A EIN"' erforderlich. Sie müssen den zu suchenden String also in Anführungszeichen einschließen.

Eine Vereinfachung bietet der Befehl FIND zum Suchen nach Strings: Sie können nach einem bestimmten Anfangsteil eines Strings suchen. So zeigt Ihnen z.B. der Befehl 'FIND"G' alle Zeilen in denen Strings enthalten sind, die mit 'G' anfangen. Sie können aber nicht Teile in der Mitte oder am Ende eines Strings aufsuchen.

Um mit dem Befehl 'FIND' vertraut zu werden, geben Sie das folgende Programm ein und beachten die anschließende Tabelle der unterschiedlichsten FIND-Möglichkeiten:

```

10 REM BEISPIEL ZUM BEFEHL FIND
20 PRINT "GEBEN SIE DEN WERT A EIN";
30 INPUT A
40 PRINT "GEBEN SIE DEN WERT B EIN";
50 INPUT B
60 C=A+B
70 PRINT "DIE SUMME DER BEIDEN WERTE"
80 PRINT "IST":C

```

<b>FIND-BEFEHL</b>	<b>GEFUNDENE ZEILEN</b>
-----	
<b>FINDREM</b>	10
<b>FINDFIND</b>	10
<b>FINDA</b>	30 60
<b>FIND A</b>	30
<b>FINDB</b>	10 50 60
<b>FIND B</b>	10 50
<b>FIND"GEBEN SIE</b>	20 40
<b>FINDC</b>	60 80
<b>FINDC=</b>	60
<b>FINDA+B</b>	60
<b>FINDPRINT</b>	20 40 70 80
<b>FINDINPUT</b>	30 50
<b>FIND;</b>	20 40 80

### 2.3.5 TESTAUFGABEN

1) Welcher der folgenden Befehle teilt das Programmlisting in Abschnitten zu 10 Bildschirmzeilen auf?

- A) PAGE 10
  - B) PAGE 11
  - C) PAGE 9
- 

2) Mit welcher Taste können Sie die Abschnitte des mit PAGE aufgeteilten Programmlistings "durchblättern"?

- A) mit RETURN
  - B) mit der Commodore-Taste (C=)
  - C) mit SHIFT
  - D) mit CTRL
- 

3) Welcher Befehl schaltet die Markierung der SIMON'S BASIC-Befehle ein?

- A) OPTION
  - B) OPTION 1
  - C) OPTION 10
  - D) OPTION 0
- 

4) Welcher Befehl verändert die Geschwindigkeit des Programmlistings?

- A) DECAY
  - B) DELAY
  - C) DELLAY
-

5) Welche Taste verzögert die Geschwindigkeit während dem  
listen des Programmes auf dem Bildschirm?

- A) RETURN
  - B) SHIFT
  - C) Commodore-Taste (C=)
- 

6) Es liegt das folgende Programm vor:

```
10 INPUT "GEBEN SIE EINE ZAHL EIN";A
```

```
20 PRINT "SIE HABEN";A;"EINGEGEBEN"
```

Welcher der folgenden FIND-Befehle zeigt 1keine0 Zeile  
als gefunden an?

- A) FINDA
- B) FIND A
- C) FIND"G

### 3. KAPITEL

#### FEHLERBEHANDLUNG

Auch die besten Programmierer können sich nicht vor einem intensiven Test und der sich daraus meistens ergebenden Fehlerbehandlung ihrer Programme drücken. Doch welche Möglichkeiten der Fehlerbehandlung bietet der CBM 64 in der Grundversion? Man unterscheidet hierbei zwei verschiedene Arten der Fehlersuche: Die externen und die internen Maßnahmen zum Aufspüren von Fehlern (Fehler nennt man in der Fachsprache BUGS und die Fehlersuche DEBUGGING).

Externe Maßnahmen sind z.B. die Anzeige von Variablen nach einer Programmunterbrechung oder ein gezielter LIST auf die vom Interpreter als fehlerhaft gemeldete Zeile.

Interne Maßnahmen sind Befehle, die zwar im Programm eingebaut sind, aber nicht zum eigentlichen Programm gehören. Sie werden nach abgeschlossenem Fehlertest wieder entfernt oder als Remarks (REM) unwirksam gemacht.

Der Commodore 64 bietet Ihnen einen Befehl, der ausschließlich zur internen Fehlererkennung bestimmt ist. Dies ist der Befehl STOP, dessen Funktion und Möglichkeiten den Wenigsten bewußt ist. Ein STOP wird in das Programm an einer Stelle eingebaut, wo eine Unterbrechung des Programms zur anschließenden Kontrolle von Variablen nötig ist. Der Rechner meldet beim Erreichen eines STOPS 'BREAK IN ....'. Danach können alle gewünschten Variablen angezeigt werden (z.B. PRINT A\$). Soll das Programm dann fortgesetzt werden, so geben Sie den Befehl CONT ein. Das Programm läuft nun weiter bis zum Ende oder bis zum nächsten STOP.

#### WICHTIG:

Wird nach einer Programmunterbrechung ein Eingriff in das Programm vorgenommen (Zeilen geändert) so ist keine Fortsetzung des Programms mit CONT möglich. Auch alle Variablen werden wieder gelöscht. Der Rechner meldet "?CAN'T CONTINUE ERROR". Variablen jedoch können nach einer Programmunterbrechung beliebig verwaltet werden. So kann durchaus einer Variablen ein Testwert zugewiesen und das Programm mit CONT fortgesetzt werden.

Weitere Befehle, die speziell der Fehlerbehandlung dienen,

weist der Commodore 64 nicht vor. Alle weiteren Maßnahmen zur Fehlersuche muß der Programmierer selbst treffen. So kann er z.B. beim Einlesen einer Datei in den Rechner mit einem eingesetzten PRINT alle Daten, die der Rechner übernimmt, anzeigen.

Diese ins Programm intergrierten Hilfen zur Fehlersuche müssen nach erfolgreicher Fehlerbeseitigung wieder entfernt werden. Was meinen Sie wie ein Anwender eines von Ihnen entwickelten Programms reagiert, wenn z.B. bei der Rechnungsasugabe die Meldung 'BREAK IN 1220' erscheint?. Sinnvoll ist es, die Zeilen, in denen Sie interne Maßnahmen zur Fehlersuche einbauen, zu notieren. So ist es am Ende des Programmtests einfach, alle Zeilen zu normalisieren. Wie schon gesagt, besitzt der Commodore 64 nur die Befehle STOP und CONT zur Unterstützung bei der Fehlersuche. SIMON'S BASIC bietet auch hier einige zusätzliche Befehle.

### **3.1 EXTERNE FEHLERBEHANDLUNG**

#### **3.1.1 TRACE**

**FORMAT: TRACE N**  
**PARAMETER: N - 10 ZUM EINSCHALTEN**  
**UNGLEICH 10 ZUM AUSSCHALTEN**  
**FUNKTION: ANZEIGE ALLE GERADE DURCHLAUFENDEN BASIC-**  
**ZEILEN RECHTS OBEN AM BILDSCHIRM**  
**BEISIEL: TRACE 10**  
**SCHALTET TRACE-MODUS EIN**

TRACE ist eine sehr beliebte Funktion zur Fehlerbehandlung. In komfortablen BASIC-Interpretern wie z.B. MBASIS von MICROSOFT ist eine solche Funktion enthalten. Nun brauchen auch die Commodore 64 Anwender nicht auf TRACE zu verzichten. Die Wirkungsweise dieses Befehls ist einfach zu erklären. Sie schalten den TRACE-Modus vor dem Starten des Programms mit 'TRACE 10' ein. Wenn Sie nun das Programm starten, erscheint in der oberen rechten Ecke des Bildschirms ein Fenster von 6 Zeilen mit jeweils 6 Zeichen. In diesem Fenster wird immer die zuletzt durchlaufende Zeile angezeigt. Natürlich wird das

Programm dadurch etwas langsamer, da der Rechner neben Ihrem Programm auch das TRACE-"Programm" steuern muß. Das folgende Beispielprogramm soll Ihnen die TRACE-Funktion verdeutlichen:

```
5 PRINT CHR$(147);
10 PRINT CHR$(19); "10":GETXS:IFX$="" THEN10
20 PRINT CHR$(19); "20":GETXS:IFX$="" THEN20
30 PRINT CHR$(19); "30":GETXS:IFX$="" THEN30
40 PRINT CHR$(19); "40":GETXS:IFX$="" THEN40
50 PRINT CHR$(19); "50":GETXS:IFX$="" THEN50
60 END
```

Die Sonderzeichen für 'CLR/HOME' und 'HOME' wurden mit CHR\$(147) und CHR\$(19) codiert, da die Sonderzeichen nicht jedem bekannt sind.

Geben Sie also dieses Programm ein. Nun kommt der große Augenblick: Schalten Sie den TRACE-Modus mit 'TRACE 10' ein und starten das Programm mit 'RUN'. Die letzte Zeile des Bildschirmfensters zeigt Ihnen nun die zuletzt bearbeitete Zeile (Zeile 5). Der PRINT in Zeile 10 ist schon ausgeführt, aber die Zeile 10 ist noch nicht vollständig bearbeitet. Es wird noch auf einen Tastendruck Ihrerseits gewartet. Drücken Sie nun irgendeine Taste, wird die Zeile 10 im TRACE-Fenster angezeigt und der PRINT in Zeile 20 zeigt Ihnen, daß das Programm sich gerade in Zeile 20 befindet. So geht das nun weiter, bis das Programm beendet ist. Am Ende des Programms finden Sie das folgende TRACE-Fenster vor:

```
#5
#10
#20
#30
#40
#50
```

Die Zeile 60 ist zwar auch ausgeführt worden, jedoch wird sie nicht mehr angezeigt, da das Programmende erreicht ist. Schalten Sie nun den TRACE-Modus mit 'TRACE 0' wieder aus. Überzeugen Sie sich mit 'RUN' davon, daß der TRACE auch wirklich ausgeschaltet ist.



Diese Demonstration hat Ihnen nun die Wirkungsweise des TRACE-Befehls verdeutlicht.

Aber auch bei diesem Befehl gibt es etwas zu beachten: Schalten Sie nochmals den TRACE-Modus mit 'TRACE 10' ein starten das Programm mit 'RUN'. Nun unterbrechen Sie den Programmablauf mit der Taste 'RUN/STOP'. Der Rechner meldet nun 'BREAKIN 10'. Wenn Sie nun in der aktuellen 5. Zeile den Befehl 'LIST' eingeben, erscheint die Meldung '?SYNTAX ERROR'. Nun werden Sie sich vielleicht fragen, warum der Rechner den Befehl nicht akzeptiert. Die Antwort jedoch ist einfach: In der Zeile 5, in der Sie den LIST-Befehl eingegeben haben, befindet sich am Ende der Zeile der Ausschnitt des TRACE-Fensters. Dieser Ausschnitt wird mit als Befehl interpretiert und dies hat die Fehlermeldung zur Folge. Sie müssen also entweder den Bildschirm löschen, oder den Cursor außerhalb der ersten 6 Zeilen bewegen, bevor Sie weitere Befehle eingeben.

Da der TRACE von SIMON'S BASIC keinen Single-Step, also Programmablauf Zeile für Zeile ermöglicht, ist die Programmkontrolle eingeschränkt. Das Scrollen der Programmzeilen im TRACE-Fenster sehr schnell. Es ist daher sinnvoll, zum Testen des Programms an verschiedenen Stellen Warteschleifen einzubauen. Natürlich müssen diese Zeilen schriftlich festgehalten werden, um Sie im endgültigem Programm wieder zu löschen. Eine Wartescheife von ca. 1 Sekunde kann mit dem SIMON'S BASIC-Befehl 'PAUSE 1' erzeugt.

### 3.1.2 RETRACE

<b>FORMAT:</b>	<b>RETRACE</b>
<b>PARAMETER:</b>	<b>KEINE</b>
<b>FUNKTION:</b>	<b>NACH LÖSCHEN DES BILDSCHIRMS WIRD DAS ZULETZT ANGEZEIGTE TRACE-FENSTER ZURÜCKGEHOLT</b>

Im letzten Kapitel wurde geschildert, daß zum Eingeben von Befehlen in den ersten 6 Zeilen des TRACE-Fensters der Bildschirm gelöscht werden muß. Wenn Sie aber nun nochmal Einblick in die letzten TRACE-Zeilen nehmen möchten, ist der Befehl RETRACE notwendig. Das folgende Programm demonstriert die RETRACE-Funktion:

```
5 PRINT CHR$(147);  
10 FOR I =1TO10  
20 PRINT I  
30 NEXT  
40 PRINT CHR$(147)
```

Die letzte Zeile dieses Programms löscht den Bildschirm. Es kann durchaus auch in einem anderen Program der Fall sein, daß der Bildschirm gelöscht wird und die letzten TRACE-Zeilen löscht. Diese Zeilen können aber gerade interessant sein. Lassen Sie dieses Programm nun laufen, nachdem Sie den TRACE-Modus mit 'TRACE 10' eingeschaltet haben. Nach Ablauf des Programms sind keine TRACE-Zeilen mehr auf dem Bildschirm. Geben Sie nun den Befehl 'RETRACE' ein und das folgende TRACE-Fenster erscheint auf dem Bildschirm:

```
#20  
#30  
#20  
#30  
#20  
#30
```

Das Program ist also immer zwischen den Zeilen 20 und 30 hin- und hergesprungen. Die eigentliche Schleife befindet sich also in Zeile 20 und 30. Man erkennt, daß die Zeilen 5 und 10 nur einmal durchlaufen werden.

### 3.1.3 DUMP

**FORMAT:** DUMP  
**PARAMETER:** KEINE  
**FUNKTION:** ANZEIGEN ALLER BENUTZTEN VARIABLEN  
UND IHRE MOMENTANEN WERTE  
**BEMERKUNG:** MACHT PROBLEME BEI NEGATIVEN ZAHLEN  
UND LEEREN STRINGS

Nach einer Programmunterbrechung ist es interessant, wichtige Variablen zu kontrollieren. Dazu verwendet man den PRINT-Befehl. Der Nachteil ist, dass jede anzuzeigende Variable einen PRINT-Befehl benötigt. Der Befehl DUMP erleichtert diese Kontrolle. Er zeigt alle im Programm benutzten Variablen mit den entsprechenden Werten auf dem Bildschirm an. Die Ausgabe des Befehls DUMP kann wie beim Befehl LIST mit der CTRL-Taste verzögert werden.

So komfortabel der DUMP-Befehl erscheinen mag, so sehr wird der positive Eindruck durch einige Mängel verdrängt:

1. es werden keine ARRAYS angezeigt (z.B. A(23) oder A\$(2))
2. negative Variablen werden stets positiv angezeigt
3. negative Integer-Variablen (z.B. A%) wird der Wert  $2^{16}$  aufaddiert
4. definierte Leerstrings (z.B. A\$="") werden mit 256 undefinierbaren Zeichen belegt

Dies sind sehr wesentliche Mängel des DUMP-Befehls. Lediglich Strings und positive Variablen werden korrekt angezeigt. Geben Sie z.B. einmal die folgende Zeile im Direktmodus (d.h. ohne Zeilennummer) ein:

```
A=10:B=-5:C%=-57:D$=" "
```

Bevor Sie nun den Befehl DUMP eingeben, sollten Sie sich hinsetzen, denn das Ergebnis ist verblüffend:

A=10

B=5

C%=65479

```
DS="????????????????????????????????????????????????????
????????????????????????????????????????????????????????
????????????????????????????????????????????????????????
????????????????????????????????????????????????????????
????????????????????????????????????????????????????????
????????????????????????????????????????????????????????
????????????????????????????????????????????????????????
????????????????????"
```

Lediglich die Variable A wird korrekt angezeigt! In diesem Fall ist es ratsam, die Variablen mit dem Befehl

**PRINT A.B.C%.DS**

abzufragen. Das Ergebnis ist dann auch vollkommen korrekt:

**10            -5            -57**

Die numerischen Variablen werden angezeigt, der Leerstring nicht.

Doch sollten Sie den Befehl nicht aus Ihrem Handbuch streichen. Wenn Sie diese "BUGS" beim DUMP berücksichtigen, ist er doch - wenn auch nur beschränkt - einsetzbar. Bei einer so umfangreichen Befehlserweiterung wie SIMON'S BASIC sie darstellt, sollte man derartige Fehler nicht zu sehr kritisieren.

### 3.1.4 COLD

**FORMAT:** COLD  
**PARAMETER:** KEINE  
**FUNKTION:** BRINGT SIMON'S BASIC IN DEN AUSGANGS-  
ZUSTAND

Wenn man den Commodore 64 ohne SIMON'S BASIC in den Zustand nach dem Einschalten versetzen möchte, setzt man den Befehl 'SYS 64738' ein. Gibt man diesen Befehl ein, wenn SIMON'S BASIC im Rechner vorhanden ist, so wird der Zustand nach dem Laden von SIMON'S BASIC erreicht. Die Adresse 64738 kann leicht in Vergessenheit geraten. Der Befehl COLD erfüllt den selben Zweck und ist auch einfacher zu behalten. Setzt man diesen Befehl ein, so muß man beachten, daß das im Speicher befindliche Programm wieder gelöscht wird. Auch alle Variablen werden zurückgesetzt. Das Programm kann man nach dem Kaltstart mit dem Befehl 'OLD' (siehe 3.1.5) wieder "zurückholen".

Der Kaltstart mit 'COLD' ist wie 'SYS 64738' kein Kaltstart im eigentlichen Sinne. Kaltstart bedeutet, daß alle Adressen im Hauptspeicher auf Null gesetzt werden. Das ist bei 'COLD' oder 'SYS 64738' nicht der Fall. Hier werden nur die BASIC-Zeiger, die auf Anfang und Ende des Programms zeigen, wieder in den Ausgangszustand versetzt. Da nicht der gesamte Hauptspeicher gelöscht wird, kann man hier von einem "Warmstart" sprechen. Ein "echter" Kaltstart ist beim Commodore 64 nur durch Aus- und wieder Einschalten des Rechners möglich.

### 3.1.4 OLD

**FORMAT: OLD**  
**PARAMETER: KEINE**  
**FUNKTION: REGENERIEREN EINES MIT NEW GELÖSCHTEN PROGRAMMS**

Wenn Sie versehentlich einen der Befehle NEW oder COLD eingegeben haben, die ja bekanntlich das BASIC-Programm löschen, ist dieser Befehl sehr hilfreich. Er holt das gelöschte BASIC-Programm wieder zurück. Probieren Sie dies nun aus, indem Sie zunächst die folgenden BASIC-Zeilen eingeben:

```
10 REM *****  
20 REM * TEST OLD *  
30 REM *****
```

Löschen Sie nun das Programm mit dem Befehl 'NEW'. Mit einem anschließenden 'LIST' können Sie sich davon überzeugen, daß das Programm auch wirklich verschwunden ist. Nutzen Sie nun den Befehl 'OLD', um das Programm zurückzuholen. Wenn Sie nun 'LIST' eingeben, sehen Sie den Erfolg.

Das gleiche Beispiel können Sie nun mit dem Befehl 'COLD' durchführen. Geben Sie diesen Befehl nun ein und holen Sie anschließend das Programm mit 'OLD' wieder zurück. Werden nach dem Befehl 'NEW' BASIC-Zeilen gelöscht, geändert oder hinzugefügt, so hat der anschließende Befehl 'OLD' keine Wirkung!.

### 3.1.6 TESTAUFGABEN

1) Welcher Befehl schaltet den TRACE-Modus von SIMON'S BASIC ein?

- A) TRACE
  - B) TRACE 0
  - C) TRACE 10
  - D) RETRACE
- 

2) Welcher Befehl holt das zuletzt vorhandene TRACE-Fenster auf dem Bildschirm zurück?

- A) TRACE
  - B) TRACE 0
  - C) TRACE 10
  - D) RETRACE
- 

3) Was zeigt der Befehl DUMP auf dem Bildschirm an?

- A) Die benutzten Variablen
  - B) Die Inhalte der benutzten Variablen
- 

4) Welcher Befehl bewirkt einen Kaltstart des CBM 64?

- A) COLD
  - B) OLD
  - C) NEW
- 

5) Welcher Befehl holt ein gelöschttes BASIC-Programm wieder zurück?

- A) COLD
  - B) OLD
  - C) NEW
-

## 3.2 PROGRAMMINTERNE FEHLERBEHANDLUNG

Die programminterne Fehlerbehandlung ist notwendig, um Fehler im Programm abzufangen und ein Abstürzen zu verhindern. SIMON'S BASIC bietet Ihnen nun diese sehr komfortable Möglichkeit, jegliches Abstürzen eines Programms zu verhindern. Sei es ein 'SYNTAX ERROR' oder ein 'DEVICE NOT PRESENT ERROR', sämtliche Fehler werden im Programm erkannt und individuelle Fehlermeldungen können ausgegeben werden. Die folgenden Kapitel werden diese umfangreichen Möglichkeiten ausführlich demonstrieren.

### 3.2.1 ON ERROR

**FORMAT:** ON ERROR:GOTO LN  
**PARAMETER:** LN - ZEILENNUMMER, DIE BEIM AUFTRETEN EINES FEHLERS ANGESPRUNGEN WERDEN SOLL  
**FUNKTION:** ABGEBEN DER FEHLERKONTROLLE AN DAS PROGRAMM  
**BEISPIEL:** ON ERROR:GOTO 1000  
BEI AUFTRETEN EINES FEHLERS WIRD AB SOFORT ZUR ZEILE 1000 VERZWEIGT  
**BEMERKUNG:** DER BEFEHL ÜBERGIBT DIE FEHLERNUMMER IN VARIABLE ERRN UND DIE FEHLERZEILE IN VARIABLE ERLN

Ein gut organisiertes und programmiertes Programm darf in keiner Situation "abstürzen". Dazu trägt einmal die intensive Kontrolle der vom Anwender eingegebenen Werte bei. Wünscht der Bediener eines Programms z.B. den 120. Datensatz eines Arrays, das nur bis 100 dimensioniert wurde, so muß das Programm dies erkennen und den Anwender auf die Fehleingabe hinweisen. Ist dies nicht der Fall, so steigt das Programm bei der Adressierung des Arrays mit der Fehlermeldung '? BAD SUBSCRIPT ERROR IN ...' aus.

Aber nicht jeder Fehler kann vom Programm abgefangen werden. Nehmen wir einmal an, es sollen Daten auf die Diskettenstation gespeichert werden. Der Anwender schließt



aber keine Diskettenstation an den Rechner an. Dies verursacht dann beim ersten Versuch, Daten zu übertragen, die Fehlermeldung 'DEVICE NOT PRESENT ERROR IN ...'. Es ist jedoch unmöglich, programmtechnisch festzustellen, ob das ein oder andere Gerät angeschlossen ist. Dieser Fehler und noch viele andere können mit SIMON'S BASIC erkannt werden.

Dazu wird die Fehlernummer in der Variablen ERRN und die Nummer der Zeile, in der der Fehler erkannt wurde, in ERRLN gespeichert. Die Fehlernummern können Sie der folgenden Tabelle der von SIMON'S BASIC erkennbaren Fehler entnehmen:

<b>FEHLERNUMMER</b>	<b>FEHLER</b>
1	TOO MANY FILES
2	FILE OPEN
3	FILE NOT OPEN
4	FILE NOT FOUND
5	DEVICE NOT PRESENT
10	NEXT WITHOUT FOR
11	SYNTAX
12	RETURN WITHOUT GOSUB
13	OUT OF DATA
14	ILLEGAL QUANTITY
15	OVERFLOW
16	OUT OF MEMORY
17	UNDEFINDED STATEMENT
18	BAD SUBSCRIPT
19	RE-DIMENSIONED ARRAY
20	DIVISION BY ZERO
21	ILLEGAL DIRECT
22	TYPE MISMATCH
23	STRING TOO LONG

Sämtliche Fehlermeldungen der Floppystation können jedoch nicht mit ON ERROR erkannt werden. Diese Fehler müssen an den gegebenen Stellen aus dem Fehlerkanal der Floppy gelesen und analysiert werden. Näheres hierzu finden Sie in dem DATA BECKER BUCH 'Das große Floppybuch'.

Doch nun zur Anwendung des Befehl 'ON ERROR'. Dieser Befehl

wird an der Stelle des Programms eingesetzt, ab der die Fehlerkontrolle dem Programm übergeben werden soll. Möchten Sie also grundsätzlich die Fehlerkontrolle übernehmen, so ist dies die erste Zeile des Programms. Ein Programm sollte aber nicht so unsicher sein, daß dies erforderlich ist. Nur ein Fehler, der programmtechnisch nicht vermieden werden kann, sollte an gegebener Stelle mit ON ERROR lokalisiert werden. Die Fehlermeldung "DEVICE NOT PRESENT" z.B. kann nicht vom Programmierer vermieden werden. Wird im Programm ein Gerät angesprochen, daß nicht angeschlossen ist, so erscheint diese Fehlermeldung. Der Programmierer kann hier nun die ON ERROR-Funktion einsetzen und den Anwender darauf hinweisen, das Gerät anzuschließen bzw. einzuschalten. Wenn Sie ON ERROR einsetzen, müssen Sie sich entscheiden, wo die Auswertungsroutine enthalten sein soll. Nehmen wir beispielsweise an, sie wollen die Fehlerkontrolle an die Routine ab Zeile 10000 abgeben. Der entsprechende Befehl wäre dann:

**10 ON ERROR:GOTO 10000**

Der ON ERROR-Befehl darf auf keinem Fall im Direkt-Modus, d.h. ohne Zeilennummer eingegeben werden, da sonst auch die Fehlerkontrolle der direkt eingegebenen Anweisungen abgegeben wird. Der Rechner hängt sich nach jeder Eingabe auf und kann nur mit der Taste 'RUN/STOP' und anschließendem NO ERROR wieder in den Ausgangszustand versetzt werden.

Da zum Einsatz von 'ON ERROR' auch der Befehl 'NO ERROR' notwendig ist, folgt nun zunächst die Beschreibung von 'NO ERROR'. Im Kapitel 3.2.3 werden dann beide Befehle eingesetzt.

### 3.2.2 NO ERROR

**FORMAT: NO ERROR**  
**PARAMETER: KEINE**  
**FUNKTION: ÜBERGIBT DIE FEHLERKONTROLLE WIEDER AN DAS STANDARD-BASIC**

Dieser Befehl unterdrückt nicht, wie im Handbuch von SIMON'S BASIC beschrieben, die Fehlermeldungen, sondern übergibt die mit 'ON ERROR' übernommene Fehlerkontrolle wieder an das Standard-BASIC. Nach einer Fehlerbehandlung mit 'ON ERROR' ist es unbedingt erforderlich, vor dem Verlassen des Programms oder nach dem Unterbrechen des Programms mit der Taste 'RUN/STOP', mit 'NO ERROR' zurückzuschalten. Wird dies ignoriert, so kann es überraschende Folgen haben, ja sogar SIMON'S BASIC zum Absturz bringen. Sinnvoller wäre es, den 'ON ERROR'-Modus außerhalb eines Programms grundsätzlich unwirksam zu machen. Es ist völlig sinnlos, wenn nach Eingabe eines Befehls wie z.B. 'LIST' zu der Fehleroutine Ihres Programms verzweigt wird, und nur die Taste 'RUN/STOP' das 'READY' des Betriebssystems erzwingt. Wie und wo der Befehl 'NO ERROR' eingegeben wird, erfahren Sie im folgenden Kapitel.

### 3.2.3 ON ERROR/NO ERROR IM DETAIL

Wie setzt man nun die Befehle 'ON ERROR' und 'NO ERROR' sowie die Variablen ERRN und ERRLN in einem Programm ein? Welche wichtigen Voraussetzungen muß der Anwende beachten, um schlimme Reaktionen auf falsche Anwendungen zu vermeiden? Zwar sollte die ON ERROR-Funktion immer nur an gegebener Stelle eingesetzt werden, doch zur Demonstration übernehmen wir in den folgenden Beispielen grundsätzlich die gesamte Fehleranalyse. Dazu wird der ON ERROR-Befehl am Anfang des Programms gesetzt. Die entsprechende Zeile sieht dann z.B. so aus:

```
10 ON ERROR:GOTO 10000
```

Diese Zeile bestimmt nun, daß bei Auftreten eines Fehlers zur Zeile 10000 verzweigt wird. Dieser Anfangszeile wird nun das eigentliche Programm angeschlossen. Geben Sie hier zur Demonstration die folgenden, zum Teil fehlerhaften Zeilen ein:

```
20 REM *****
30 REM   TEST ON ERROR
40 REM *****
50 PRINT"---TEST ANFANG---"
60 PRINT:"START"
70 PRINT CHR$(255)
80 A=10:A$="20"
90 PRINT A+A$
100 X$(11)="---TEST ENDE---"
110 PRINT X$(11)
120 NO ERROR:END
```

Dies ist nun das Programm, das mit einigen Fehlern bestückt ist. Es fehlt nun noch die Fehleranalyse, die wie in Zeile 10 bestimmt, ab Zeile 10000 beginnt. Eine einfache Fehleranalyse sieht etwa so aus:

```
1000 REM *****
10010 REM   FEHLERANALYSE
10020 REM *****
10030 PRINT"FEHLER NUMMER" ERRN "IN ZEILE" ERRLN
10040 PRINT"PROGRAMMABBRUCH"
10050 NO ERROR:END
```

Dies ist zunächst eine sehr einfache Möglichkeit, auf Fehler zu reagieren. Starten Sie nun dieses Programm mit 'RUN'. Folgendes wird auf dem Bildschirm ausgegeben:

```
---TEST ANFANG---
FEHLER NUMMER 11 IN ZEILE 60
PROGRAMMABBRUCH
```

Die Fehlernummer 11 weist, wie Sie der Tabelle in Kapitel 3.2.1 entnehmen können, auf den Fehler 'SYNTAX ERROR'. Diese Nummer wird in der Variablen ERRN gespeichert. Die fehler-

verursachende Zeile kann der Variablen ERRLN entnommen werden. Nach Ausgabe der Fehlernummer und -zeile wird hier das Programm abgebrochen. Doch dies ist nicht Sinn der Sache. Auf diese Weise wird der Fehler auch von dem Standard-BASIC behandelt. Das Programm soll aber lediglich die Fehlermeldung ausgeben und im Programm fortfahren. D.h. die Fehlerroutine muß nach dem Fehlerhinweis wieder zu der Zeile springen, die der Fehlerzeile folgt. Diese Zeile kann berechnet werden. Wenn das Programm kontinuierlich mit der Schrittweite 10 numeriert wurde, so ergibt sich die Nummer der Folgezeile aus der Nummer der Fehlerzeile (ERRLN) zuzüglich der Schrittweite 10. Dies erfordert jedoch einen berechneten Sprung. Der GOTO-Befehl des Standard-BASIC kann nur auf eine bestimmte Zeile springen. SIMON'S BASIC bietet einen Befehl zum berechneten Sprung. Man kann hier eine Variable oder einen mathematischen Ausdruck als Sprungadresse einsetzen. In unserem Fall würde dieser Sprung bei einer Schrittweite von 10 wie folgt codiert:

#### **CGOTO (ERRLN+10)**

Der Befehl zum berechneten Sprung lautet 'CGOTO' und wird im Kapitel 5.2.5 näher erläutert. Die Sprungadresse ergibt sich also aus (ERRLN+10). Fügen wir nun diesen Befehl ein, so ergibt sich folgende Fehlerroutine:

```
1000 REM *****
10010 REM   FEHLERANALYSE
10020 REM *****
10030 PRINT "FEHLER NUMMER" ERRN "IN ZEILE" ERRLN
10040 PRINT "SPRUNG NACH ZEILE" ERRLN+10
10050 CGOTO (ERRLN+10)
```

Geändert wurden hier die Zeilen 10040 und 10050. Die Zeile 10040 gibt die Zeile aus, in der das Programm fortgesetzt wird. In der Zeile 10050 wird dann zu dieser Zeile gesprungen. Nachdem Sie dieses Programm nun gestartet haben, wird das folgende Fehlerprotokoll auf dem Bildschirm ausgegeben:

**FEHLER NUMMER 11 IN ZEILE 60**  
**SPRUNG NACH ZEILE 70**  
**FEHLER NUMMER 14 IN ZEILE 70**  
**SPRUNG NACH ZEILE 80**  
**FEHLER NUMMER 22 IN ZEILE 90**  
**SPRUNG NACH ZEILE 100**  
**FEHLER NUMMER 18 IN ZEILE 100**  
**SPRUNG NACH ZEILE 110**  
**FEHLER IN ZEILE 110**  
**SPRUNG NACH ZEILE 120**

Dies ist eine Fehlerliste, in der noch die numererisch angegebenen Fehler mit Hilfe der Tabelle in Kapitel 3.2.1 übersetzt werden müssen. Dem Anwender Ihres Programms sind die Fehler des Betriebssystems jedoch nicht bekannt. Sämtliche Fehler, die der Anwender hervorrufen kann sollten vom Programm z.B. mit ON ERROR abgefangen werden. Wird ein Fehler erkannt, so muß eine Fehlermeldung ausgegeben werden, die den Anwender des Programms in Klartext darauf hinweist und gegebenenfalls weitere Anweisungen gibt.

Daß o.g. Programmbeispiel gibt nur die Fehlermeldungen aus. Um Ihnen zu demonstrieren, daß jede Fehlernummer durch einen beliebigen Text ersetzt werden kann, folgt nun eine Fehleroutine, die die Fehler in deutsch übersetzt. Anstatt der deutschen Fehlermeldung können Sie nun auch detaillierte Fehlerbeschreibungen ausgeben, die für den Anwender aufschlußreicher sind.

Für alle, die nicht ausgesprochen schreibfaul sind, folgt nun die entsprechende Programmänderung:

Wir rufen zu Anfang des Programm ein Unterprogramm auf, das die Fehlermeldungen in einem Array ablegt. Das Unterprogramm soll die Anfangszeile 20000 erhalten und in Zeile 5 aufgerufen werden. Schauen Sie soch zunächst das komplette, abgeändete Programm einmal an:

```

5 GOSUB 20000
10 ON ERROR:GOTO 10000
20 REM *****
30 REM TEST ON ERROR
40 REM *****
50 PRINT"---TEST ANFANG---"
60 PRINT:"START"
70 PRINT CHR$(256)
80 A=10:A$="20"
90 PRINTA+A$
100 X$(11)="---TEST ENDE---"
110 PRINT X$(11)
120 NO ERROR:END
10000 REM *****
10010 REM FEHLERANALYSE
10020 REM *****
10030 PRINTF$(ERRN)" IN ZEILE"ERRLN
10040 PRINT"SPRUNG NACH ZEILE"ERRLN+10
10050 CGOTO (ERRLN+10)
20000 REM *****
20010 REM LESEN FEHLERTABELLE
20020 REM *****
20030 DIM F$(23)
20040 READ I,I$
20050 IF I=0 THEN RETURN
20060 F$(I)=I$
20070 GOTO 20040
20080 DATA 1,ZU VIELE DATEIEN
20090 DATA 2,DATEI OFFEN
20100 DATA 3,DATEI NICHT OFFEN
20110 DATA 4,DATEI NICHT GEFUNDEN
20120 DATA 5,GERAET NICHT VORHANDEN
20130 DATA 10,'NEXT' OHNE 'FOR'
20140 DATA 11,SYNTAX FEHLER
20150 DATA 12,'RETURN' OHNE GOSUB
20160 DATA 13,ZU WENIG 'DATAS'
20170 DATA 14.UNZULAESSIGER WERT
20180 DATA 15,UEBERLAUF
20190 DATA 16,ZU WENIG SPEICHERPLATZ
20200 DATA 17.FEHLENDE ZEILE

```

20210 DATA 18. UNGUELTIGER INDEX  
20220 DATA 19. FELD BEREITS DIMENSIONIERT  
20230 DATA 20. DIVISION DURCH NULL  
20240 DATA 21. DIREKT-MODUS VERBOTEN  
20250 DATA 22. FALSCHER VARIABLENTYP  
20260 DATA 23. ZEICHENKETTE ZU LANG  
20270 DATA 0. ENDE

Ein Programm mit einer derartigen Fehlerbehandlung kann nun nicht mehr "abstürzen" wenn eine Fehler vorkommt. Es stellt also einen gewissen Programmschutz dar, da ein abgebrochenes Programm vom Anwender aufgelistet oder gespeichert werden kann.

Für diejenigen unserer Leser, die dieses Programm nicht ganz überblicken, folgt nun eine Beschreibung der Änderungen. Zunächst wird in Zeile 5 das Unterprogramm aufgerufen, das die deutschen Fehlermeldungen einliest. Diese Unterprogramm belegt die Zeilen 20000 bis 20270. In der Zeile 20030 wird das Array eingerichtet, das die Fehlermeldungen enthält. Hierbei soll der Index gleichzeitig die Fehlernummer sein. D.h. der Text zur Fehlernummer 11 befindet sich in dem String FE\$(11), usw. Zeile 20040 liest aus den DATA-Zeilen die Fehlernummer und den Klartext. Der Fehlertext wird dann in dem String mit dem Index der Fehlernummer abgespeichert. Dies wiederholt sich solange, bis das Ende der DATA-Zeilen erreicht ist, was durch Lesen einer Null erkannt wird. Ist dies der Fall, wird zum Hauptprogramm zurückgesprungen (RETURN).

Die Fehlertexte befinden sich also nun im Array. In der Fehleranalyse wird nun der Text des Fehlers ausgegeben. Die Fehlernummer (ERRN) ist nun Index des entsprechenden Strings. Zeile 10020 muß entsprechend geändert werden.

Starten Sie nun nochmals dieses mit einer sehr komfortablen Fehleranalyse versehene Programm mit 'RUN'. Es wird eine Fehlerliste ausgegeben, die wesentlich aussagefähiger ist als bisher:



---TEST ANFANG---

**SYNTAX FEHLER IN ZEILE 60**  
**SPRUNG NACH ZEILE 70**  
**UNZULAESSIGER WERT IN ZEILE 70**  
**SPRUNG NACH ZEILE 80**  
**FALSCHER VARIABLENTYP IN ZEILE 90**  
**SPRUNG NACH ZEILE 100**  
**UNGUELTIGER INDEX IN ZEILE 100**  
**SPRUNG NACH ZEILE 110**  
**UNGUELTIGER INDEX IN ZEILE 110**  
**SPRUNG NACH ZEILE 120**

In der Entwicklungsphase Ihres Programms ist so eine Fehleranalyse durchaus brauchbar. Die Fehlermeldungen werden aber auch auf dem Bildschirm ausgegeben, wenn es vielleicht nicht gewünscht ist, z.B. in eine Bildschirmmaske. Sinnvoller und professioneller ist es, diese Fehlerliste während des Programmablaufs auf dem Drucker oder in eine Datei zu übertragen. Der Bildschirm wird dann nicht beeinflusst. Die entsprechenden Änderungen sind leicht einzubauen:

Fehlerprotokoll auf dem Drucker:

- Druckerkanal am Anfang des Programms öffnen:  
**3 OPEN 1.4**
- Zeilen 10030 und 10040 ändern:  
**10030 PRINT#1.F\$(ERRN)" IN ZEILE"ERRLN**  
**10040 PRINT#1."SPRUNG NACH ZEILE"ERRLN+10**
- Druckerkanal am Ende des Programm schließen:  
**120 NO ERROR:CLOSE 1:END**

Fehlerprotokoll in Datei:

- Zeile 3 einfügen:  
**3 OPEN 1.8.2."FEHLERDATEI.S.W"**
- Zeilen 10030 und 10040 wie oben ändern
- Datei wie oben schließen

Nach Ablauf des Programms haben Sie nun die Fehlerliste entweder schwarz auf weiß, oder in Form einer Datei auf Diskette. Diese Datei können Sie dann bei Bedarf auf dem

Bildschirm ausgeben. Die folgenden BASIC-Zeilen übernehmen diese Aufgabe:

```
10 OPEN 1.8.2."FEHLERDATEI.S.R"  
20 GET#1,X$  
30 PRINTX$:  
40 IF ST<>64 THEN 20  
50 CLOSE1
```

Hier wird die sequentielle Fehlerdatei geöffnet und byteweise ausgelesen sowie auf dem Bildschirm angezeigt. Das Ende der Datei wird an der Statusvariablen ST erkannt, die dann den Wert 64 enthält.

Nun stehen Ihnen alle professionellen Wege zur Fehleranalyse Ihres Programms offen. Nutzen Sie diese komfortablen Möglichkeiten, die Ihnen SIMON'S BASIC bietet so oft als möglich aus.

### **3.2.4 OUT**

An dieser Stelle sollte der Befehl OUT beschrieben werden. Da dieser Befehl in der vorliegenden Disketten-Version von SIMON'S BASIC jedoch nicht funtionsfähig ist, wird im Interesse unserer Leser auf eine Beschreibung verzichtet. Die im Handbuch enthaltene Beschreibung dieses Befehl trifft ohnehin auf den Befehl 'NO ERROR' zu. Wird der Befehl im Programm eingesetzt, hat er die Wirkung des Befehls 'END'. Im Direkt-Modus zeigt er die zuletzt ausgegebene Meldung des Rechners. Geben Sie z.B. 'RUN' ein, was einen '?SYNTAX ERROR' verursacht, so wird diese Fehlermeldung mit einem anschließenden 'OUT' nochmals ausgegeben. Der Befehl ist also in keinster Weise einsetzbar.

### 3.2.5 TESTAUFGABEN

1) Es soll die Fehlerkontrolle an das Programm abgegeben werden. Die Routine zur Fehleranalyse beginnt ab Zeile 1000. Welcher Befehl erfüllt diese Aufgabe?

- A) ON ERROR:GOTO 1000
  - B) ON ERROR;GOTO 1000
  - C) ON ERROR,GOTO 1000
  - D) ON ERROR GOTO 1000
- 

2) In welcher Variablen wird von dem Befehl 'ON ERROR' die Fehlernummer bereitgestellt?

- A) ER
  - B) EN
  - C) ERN
  - D) ERRN
- 

3) In welcher Variablen wird die Fehlerzeile bereitgestellt?

- A) ERRLINE
  - B) ERLN
  - C) ERRLN
  - D) ERL
- 

4) Welcher Befehl gibt die Fehlerkontrolle an das Standard-BASIC zurück?

- A) OUT ERROR
  - B) NO ERROR
  - C) OFF ERROR
- 

5) Welcher der folgenden Fehlermeldungen kann nicht mit dem Befehl 'ON ERROR' abgefangen werden?

- A) OUT OF MEMORY ERROR
  - B) OUT OF DATA ERROR
  - C) LOAD ERROR
-

## 4. KAPITEL PROGRAMMSCHUTZ

### 4.1 DISAPA

**FORMAT: DISAPA**  
**PARAMETER: KEINE**  
**FUNKTION: MIT SECURE 0 ZU SCHÜTZENDE ZEILEN  
KENNZEICHNEN**

Diesen Befehl setzen Sie als ersten Befehl vor jeder Zeile, die dann anschließend mit SECURE 0 geschützt werden soll. Nehmen wir beispielsweise an, die Zeile 20 des folgenden Programms soll geschützt werden:

```
10 INPUT"GEBEN SIE DEN PROGRAMMCODE EIN":A$
20 IF A$="A12B13" THEN 40
30 PRINT"CODE NICHT KORREKT!":GOTO 10
40 PRINT"CODE KORREKT!"
```

Dazu setzen Sie den Befehl 'DISAPA' vor dem eigentlichen Code der Zeile 20:

```
20 DISAPA:IF A$="A12B13"THEN 40
```

Wenn Sie das Programm nun mit LIST auf dem Bildschirm anzeigen, werden Sie feststellen, daß hinter dem DISAPA-Befehl vier Doppelpunkte gesetzt wurden:

```
10 INPUT"GEBEN SIE DEN PROGRAMMCODE EIN":A$
20 DISAPA:::IF A$="A12B13" THEN 40
30 PRINT"CODE NICHT KORREKT!":GOTO 10
40 PRINT"COOE KORREKT!"
```

Soll das gesamte Programm geschützt werden, so muß JEDE Zeile mit einem DISAPA-Befehl belegt werden. Das ist natürlich nicht sehr komfortabel, läßt aber leider nicht vermeiden. Sinn des DISPA ist es jedoch, wichtige Zeilen, die z.B. einen Kopierschutz enthalten, zu schützen. Mit DISAPA gekennzeichnete Zeilen werden mit dem folgenden Befehl geschützt.

## 4.2 SECURE

**FORMAT: SECURE 0**  
**PARAMETER: KEINE**  
**FUNKTION: SCHÜTZEN DER MIT DISAPA GEKENNZEICHNETEN ZEILEN**

Der Befehl 'SECURE 0' wird stets im Direkt-Modus, also außerhalb eines Programms eingesetzt. Die zuvor mit 'DISAPA' gekennzeichneten Zeilen werden hier geschützt. Sie haben im Kapitel 4.1 die Zeile 20 des folgenden Programms mit DISAPA gekennzeichnet:

```
10 INPUT"GEBEN SIE DEN PROGRAMMCODE EIN":A$
20 DISAPA:::IF A$="12B13" THEN 40
30 PRINT"CODE NICHT KORREKT!":GOTO10
40 PRINT"CODE KORREKT!"
```

Geben Sie nun den Befehl 'SECURE 0' ein, und Sie werden feststellen, daß die Zeile 20 geschützt wurde:

**20**

Der Inhalt der Zeile 20 ist also unkenntlich gemacht worden. Der Nachteil hier ist, daß die Zeile 20 natürlich nicht mehr zurückgeholt werden kann. Das ist auch selbstverständlich, sonst wäre es auch kein Programmschutz, wenn er beliebig wieder aufgehoben werden kann. Die geschützten Zeilen können nur noch gelöscht werden. Es ist also ratsam, von jedem geschützten Programm zwei Versionen zu speichern:

- 1. DIE VERSION MIT DEN DISAPA-BEFEHLEN**
- 2. DIE VERSION NACH DEM SECURE 0**

Somit können Sie jederzeit das ungeschützte Programm laden, ändern und wieder sichern.

Vielleicht werden Sie sich fragen, wie eine Zeile unsichtbar wird und trotzdem wirksam ist. Der Trick besteht darin, daß einer Zeile BASIC-Zeile ein Byte mit dem Inhalt 0 sowie 4 Doppelpunkte vorangestellt werden. Diese Zeile wird aufgrund der Null nicht gelistet und wegen der vier Doppelpunkte trotzdem abgearbeitet. Wir könnten an dieser Stelle ein Programm zeigen, daß die mit SECURE 0 geschützten Zeilen wieder sichtbar macht, aber dann wäre dieser Befehl sinnlos. Diese Aufgabe überlassen wir denjenigen, die gerne harte Nüsse knacken.

## 5. KAPITEL

### PROGRAMMSTRUKTUR

Die Struktur eines Programms bestimmt gleichzeitig die Übersichtlichkeit. Schon in Standard-BASIC ist es sinnvoll, strukturiert zu programmieren. So ist es z.B. zweckmäßig, Unterprogramme am Ende des Programmes anzuordnen und mit REMARKS (REM) zu kennzeichnen. Weiterhin sollte eine Verschachtelung des Programms kenntlich gemacht werden. Dazu verschiebt man die Zeilen, die zur entsprechenden Schachtelung gehören, einige Stellen nach rechts. Dies kann man entweder ausschließlich mit Doppelpunkten oder mit einem Doppelpunkt und weiteren Leerzeichen erreichen. Bei einer Schleife z.B. wird die Verschachtelung z.B. wie folgt gekennzeichnet:

```
10 FOR I=1 TO 10
20   :::A=A+1
30   :::PRINT I,A
40 NEXT I
```

Eine strukturierte IF-Abfrage ist ebenfalls übersichtlicher:

```
10 IF A$="DRUCKEN" THEN 60
20 : PRINT "KEINE DRUCKAUSGABE"
30 : PRINT "AUSGABE AUF BILDSCHIRM (J/N)?"
40 : GET X$:IFX$<> "J" AND X$<> "N" THEN 40
50 : IF X$="J" THEN A$="ANZEIGEN"
60 GOSUB 10000
```

Es ist nun leichter zu erkennen, was im NEIN-Zweig der IF-Abfrage unternommen wird.

Die Möglichkeiten der Strukturierung sind mit dem Standard-BASIC jedoch gering. SIMON'S BASIC bietet hier nun umfangreiche Möglichkeiten, das Programm sehr übersichtlich zu gestalten.

## 5.1 LOGISCHE OPERATIONEN

### 5.1.1 IF...THEN...ELSE

**FORMAT:** IF BEDINGUNG THEN JA-ZWEIG : ELSE :  
NEIN-ZWEIG

**PARAMETER:** BEDINGUNG - LOGISCHER VERGLEICH  
JA-ZWEIG - ANWEISUNGEN FÜR WAHR  
NEIN-ZWEIG- ANWEISUNGEN FÜR UNWAHR

**BEISPIEL:** IF X\$="N" THEN PRINT"ENDE":END:ELSE:  
GOTO 1000

Diese IF-Abfrage ermöglicht es, sowohl den Ja-Zweig als auch den Nein-Zweig einer Bedingung in einer Zeile abzuhandeln. Mit der herkömmlichen IF-Abfrage werden nur die Anweisungen für den Ja-Zweig in der IF-Zeile aufgenommen. Vergleichen wir die beiden Abfragemöglichkeiten mit folgendem Beispiel: Ein Zeichen soll von der Tastatur eingelesen werden. Bei betätigter Taste 'D' soll das Unterprogramm ab Zeile 1000 und bei 'B' das Unterprogramm ab Zeile 2000 aufgerufen werden. Mit dem Standard-BASIC lösen wir das Problem mit folgenden Programmzeilen:

```
10 GET X$:IF X$<>"D" AND X$<>"B" THEN 10
20 IF X$="D" THEN GOSUB 1000
30 IF X$="B" THEN GOSUB 2000
40 ....
```

Hier wurden zwei IF-Zeilen für die Verzweigung benötigt. Das Problem kann aber auch mit einer IF-Abfrage gelöst werden:

```
10 GET X$:IF X$<>"D" AND X$<>"B" THEN 10
20 IF X$="D" THEN GOSUB 1000:GOTO 40
30 GOSUB 2000
40 ....
```

Auch hier werden 4 Zeilen benötigt. Mit SIMON'S BASIC bietet sich nun die folgende Lösungsmöglichkeit an:

```

10 GET XS:IF XS<>"D" AND XS<>"B" THEN 10
20 IF XS="D" THEN GOSUB 1000:ELSE:GOSUB 2000
30 ....

```

Diese Lösung ist eleganter und sollte soweit möglich, immer eingesetzt werden.

### 5.1.2 RCOMP

```

FORMAT:      RCOMP: JA-ZWEIG : ELSE : NEIN-ZWEIG
PARAMETER:  JA-ZWEIG  - ANWEISUNGEN FÜR WAHR
                NEIN-ZWEIG - ANWEISUNGEN FÜR UNWAHR
FUNKTION:   ABFRAGE NACH DER BEDINGUNG DES LETZEN
                IF...THEN...ELSE
BEISPIEL:   RCOMP END:ELSE:GOTO 1000

```

In einem Programm kann es durchaus vorkommen, daß nach einer IF...THEN...ELSE-Abfrage und einigen Folgebefehlen die gleiche Bedingung nochmals abgefragt werden muß. Hier bietet sich dann der etwas kürzere Befehl RCOMP an, der als verkürzte Form des jeweils letzten IF...THEN...ELSE Befehls die gleiche Abfrage beinhaltet. Ein Beispiel: Die Nummer jedes Artikels der Datei "ARTDAT" wird in die Variable AN übernommen. Ist die Artikelnummer größer als 999, so ist die Lieferantenummer (LN) des Artikel in die Datei "LIEF.1000-", sonst in die Datei "LIEF.-999" aufzunehmen. Weiterhin wird unter gleichen Bedingungen der Preis (PR) in den Dateien "PREIS.1000-" und "PREIS.1000" übernommen. Das folgende Programm löst dieses Problem:

```

100 REM *****
105 REM      HAUPTPROGRAMM
110 REM *****
120 OPEN 1,8,2,"ARTDAT.S.W"
130 INPUT#1,AN,LN,PR
140 IF AN>999 THEN GOSUB 1000:ELSE:GOSUB 2000
150 RCOMP GOSUB 3000:ELSE:GOSUB 4000
170 IF ST<>64 THEN 130

```



```

180 CLOSE 1:END
1000 REM *****
1005 REM SCHREIBEN LIEF.1000-
1010 REM *****
    ...
    ...
1040 RETURN
2000 REM *****
2005 REM SCHREIBEN LIEF.-999
2010 REM *****
2020 ...
2030 ...
2040 RETURN
3000 REM *****
3005 REM SCHREIBEN PREIS.1000-
3010 REM *****
3020 ...
3030 ...
3040 RETURN
4000 REM *****
4005 REM SCHREIBEN PREIS.-999
4010 REM *****
4020 ...
4030 ...
4040 RETURN

```

Ohne Einsatz des Befehls 'RCOMP' müsste die Abfrage in Zeile 140 wiederholt werden. Die Anweisung 'RCOMP' kann solange wiederholt eingesetzt werden, bis eine weitere IF...THEN...ELSE-Verknüpfung erfolgt. Da der Befehl RCOMP nur die Bedingung des letzten IF...THEN...ELSE Befehls ersetzt, können die Anweisungen für den Ja- und NEIN-Zweig natürlich unterschiedlich sein. Die Übersichtlichkeit des Programms kann verloren gehen, wenn die letzte IF...THEN...ELSE Abfrage eines RCOMP-Befehl entweder in einem anderen Programmteil oder weit zuvor erfolgte. Sinnvoll ist der Einsatz von RCOMP also nur in nächster Nähe der entsprechenden IF...THEN...ELSE Abfrage.

### 5.1.3 REPEAT...UNTIL

**FORMAT:** REPEAT: SCHLEIFE: UNTIL: BEDINGUNG  
**PARAMETER:** SCHLEIFE - ANWEISUNGEN IN DER SCHLEIFE  
BEDINGUNG - BEDINGUNG ZUM SCHLEIFENAB-  
SCHLUß  
**FUNKTION:** SCHLEIFENSTEUERUNG MIT BEDINGUNG  
**BEISPIEL:** REPEAT: A=A+1: UNTIL A=10000

Dies ist eine alternative Schleifensteuerung gegenüber den herkömmlichen FOR...NEXT-Schleifen. Durch Anwendung dieser Alternative kann eine übersichtliche Schleife aufgebaut werden. Das folgende Beispiel wird dies demonstrieren: Es soll die Summe der Zahlen 10-100 gebildet werden.

```
10 A=10
20 REPEAT
30 :::S=S+A
40 :::A=A+1
50 UNTIL A>100
60 PRINT S
```

Nach Ablauf dieses Programm wird das Ergebnis (5005) angezeigt. Ihnen ist vielleicht aufgefallen, daß auf 'A>100' und nicht auf 'A=100' abgefragt wurde. Diese Bedingung ist erforderlich, da der Zähler (A) nach der Addition erhöht wird. Die zuletzt erreichte 100 muß also noch aufaddiert werden.

Dieses Beispiel ist jedoch viel besser und schneller mit der herkömmlichen FOR...NEXT-Methode zu lösen:

```
10 FOR A=10 TO 1000
20 ::S=S+A
30 NEXT
40 PRINT S
```

Es ist eigentlich ein ungünstiges Beispiel. Der FOR...NEXT-Befehl schneidet wesentlich besser ab. Der wesentliche Vorteil des REPEAT...UNTIL liegt in der Steuerung von Schleifen mit ungewissem Ende. D.h. Der FOR...NEXT-Befehl

enthält im zweiten Parameter (TO ...) stets eine konstante oder auch variable Zahl. Das Ende einer FOR...NEXT Schleife ist also vorher bestimmt und nicht von Ereignissen innerhalb der Schleife abhängig. Bei der REPEAT-UNTIL-Steuerung jedoch ist das Endekriterium eine beliebige Bedingung, die innerhalb der Schleife auftritt. Somit ist REPEAT...UNTIL nicht mit FOR...NEXT vergleichbar. REPEAT...UNTIL ist eigentlich eine Alternative zu "von Hand" aufgebauten Schleifen. An einem einfachen Beispiel läßt sich dies verdeutlichen: Eine sequentielle Datei soll ausgelesen werden. Das Ende der Datei wird dadurch erkannt, daß die Statusvariable ST vom Rechner mit dem Wert 64 belegt wird. Dieses Ende der Datei ist ungewiß, sodaß es nicht mit einer FOR...NEXT-Schleife abgefangen werden kann. Mit herkömmlichem BASIC würde man die Datei wie folgt auslesen:

```

10 OPEN 1,8,2,"DATEI.S.R"
20 I=0
30 :::I=I+1
40 :::INPUT#1,AS(I)
50 IF ST<>64 THEN GOTO 30
60 CLOSE 1:END

```

Hier wird eine Schleife "von Hand" aufgebaut, die aber mit einer IF-Abfrage verlassen wird. Mit REPEAT...UNTIL läßt sich dieses Problem eleganter lösen:

```

10 OPEN 1,8,2,"DATEI.S.R"
20 I=0
30 REPEAT
40 :::I=I+1
50 :::INPUT#1,AS(I)
60 UNTIL ST<>64
70 CLOSE 1

```

Hier wird zwar eine Zeile mehr benötigt, doch ein wesentlicher Beitrag zur strukturierten Programmierung geleistet: Ein GOTO wurde vermieden.

#### 5.1.4 LOOP...EXIT IF...END LOOP

```
FORMAT:      LOOP: SCHLEIFE: EXIT IF BEDINGUNG: END LOOP  
PARAMETER:  SCHLEIFE - ANWEISUNGEN IN DEN SCHLEIFE  
            BEDINGUNG - BEDINGUNG ZUM SCHLEIFENAB-  
                SCHLUB  
FUNKTION:   SCHLEIFENSTEUERUNG MIT BEDINGUNG  
BEISPIEL:  LOOP:A=A+1:EXIT IF A=1000:END LOOP
```

Auf dem ersten Blick ähnelnd diese Schleifenlogik sehr dem REPEAT...UNTIL. Die wesentlichen Vorteile dieser Steuerung jedoch sind:

- die Bedingung zum Schleifenabbruch kann mitten in den Schleifenanweisungen enthalten sein. Beispiel:

```
100 LOOP  
110 A=A+1  
120 EXIT IF A>20  
130 B=B+A  
140 END LOOP
```

- Es können mehrere Bedingungen zum Schleifenabbruch an beliebiger Stelle gesetzt werden. Beispiel:

```
100 LOOP  
110 A=A+1  
120 EXIT IF A>20  
130 B=B+A  
140 EXIT IF B>100  
150 END LOOP
```

**KOMMEN WIR NOCHMAL AUF DIE STRUKTURIERTE PROGRAMMIERUNG ZURÜCK. UM DAS LETZTE BEISPIEL MIT DEM STANDARD-BASIC ZU LÖSEN, WÄREN FOLGENDE ZEILEN ERFORDERLICH:**

```
110 A=A+1  
120 IF A>20 THEN 150  
130 B=B+A  
140 IF A>100 THEN 150  
150 ....
```

Es wurden hier wieder zwei GOTO-Befehle benötigt, was nicht Sinn der strukturierten Programmierung ist.

Ein weiteres Beispiel zur Anwendung von LOOP: Von der Tastatur soll abgefragt werden, ob eine Druckausgabe gewünscht ist. Wenn Ja, soll das Unterprogramm 1000 aufgerufen werden, sonst soll das Programm fortgesetzt werden. Zunächst die Lösung mit Standard-BASIC:

```
100 PRINT "DRUCKAUSGABE (J/N)?"
110 GET X$:IF X$<>"J" AND X$<>"N" THEN 110
120 IF X$="J" THEN GOSUB 1000
130 .....
```

Nun die Lösung mit SIMON'S BASIC:

```
100 PRINT "DRUCKAUSGABE (J/N)?"
110 LOOP
120 GET X$
130 EXIT IF X$="J"
140 EXIT IF X$="N"
150 END LOOP
160 IF X$="J" THEN GOSUB 1000
170 ...
```

Hier sind zur Lösung des Problemes zwar mehrere Zeilen notwendig, jedoch ist dieses Programm wesentlich übersichtlicher. Auch entfällt wiederum ein GOTO, was der Übersichtlichkeit des Programmes zu Gute kommt. "Wilde Sprünge" von der einen zur anderen Zeile sind nicht Sinn der strukturierten Programmierung.

### 5.1.5 TESTAUFGABEN

1) Welcher der folgenden IF...THEN...ELSE-Anwendungen ist korrekt?

- A) IF A=1 THEN 100:ELSE GOTO 200
  - B) IF A=1 THEN 100:ELSE:GOTO 200
  - C) IF A=1 THEN 100 ELSE GOTO 200
  - D) IF A=1 THEN 100 ESLE:GOTO 200
- 

2) Worauf bezieht sich der Befehl RCOMP?

- A) Auf die letzte IF-Abfrage
  - B) Auf die letzte IF...THEN...ELSE-Abfrage
  - C) Auf die erste IF-Abfrage
  - D) Auf die erste IF...THEN...ELSE-Abfrage
- 

3) Welche REPEAT-Anwendung ist richtig?

- A) REPEAT:GETX\$:UNTIL X\$<>"
  - B) REPEAT GETX\$ UNTIL X\$<>"
  - C) REPEAT GETX\$:UNTIL X\$<>"
- 

4) Welche LOOP-Anwendung ist falsch?

- A) LOOP:GET X\$:EXIT IF X\$="J":EXIT IF X\$="N" END LOOP
  - B) LOOP:A=A+1:B=B+A:EXIT IF A>10 OR B>20:END LOOP
  - C) LOOP:EXIT IF I=10:I=I+1:READ A(I):END LOOP
- 

5) Welche Zahlen werden mit folgenden Programm angezeigt?

```
A=10:LOOP:EXIT IF A>20:A=A+1:PRINT A:END LOOP
```

- A) 10 bis 20
- B) 11 bis 20
- C) 10 bis 21
- D) 11 bis 21

## 5.2 SPRUNGBEHANDLUNG

### 5.2.1 PROC

**FORMAT:** PROC LABEL  
**PARAMETER:** LABEL - SYMBOLISCHE ADRESSE  
**FUNKTION:** SYMBOLISCHE SPRUNGADRESSEN VERGEBEN  
**BEISPIEL:** 100 PROC EINGABE-ROUTINE

Bei der Beschreibung des Befehls RENUMBER wurde auf dieses Kapitel verwiesen. Der Befehl RENUMBER kann keine GOTO- oder GOSUB-Zeilennummern parallel zur Programmnummerierung berücksichtigen. Dies kann man nur umgehen, wenn diese beiden Befehle nicht verwendet werden. Ein Programm ohne GOTO und GOSUB ist unmöglich werden Sie vielleicht denken. Doch dem ist nicht so. Für beide Befehle gibt es nun Alternativen, die keine bestimmte Zeilennummer, sondern symbolische Adressen verwenden. Die Zeilen, die mit diesen Befehlen angesprochen werden sollen, müssen dazu vorher mit einer symbolischen Adresse (einem Label) versehen werden. Dies übernimmt der Befehl PROC, dessen Anwendung unproblematisch ist. Vor jeder Zeile, die symbolisch angesprochen wird, setzt man eine PROC-Zeile. Ein Beispiel:

```
100 PROC EINGABE NAME  
110 INPUT"GEBEN SIE DEN NAMEN EIN:":XS
```

Wenn nun die Zeile 110 angesprungen werden soll erfolgt dies nicht wie gewöhnlich mit 'GOTO 110', sondern mit dem Befehl 'CALL EINGABE NAME', der später beschrieben wird.

#### Beachten Sie:

In der PROC-Zeile dürfen keine weiteren Anweisungen enthalten sein!

## 5.2.2 END PROC

**FORMAT:** END PROC  
**PARAMETER:** KEINE  
**FUNKTION:** DAS ROUTINENENDE FESTLEGEN  
**BEMERKUNG:** VERGLEICHBAR MIT RETURN

Wie bereits geschildert, werden mit PROC symbolische Sprungadressen vergeben. Zusätzlich kann zusammen mit dem Befehl END PROC eine komplette Routine definiert werden. Das Anfang der Routine enthält dann die PROC-Zeile und das Ende der Routine die END PROC-Zeile. Ein Beispiel:

```
100 PROC AUSGABE ADRESSE
110 ::::PRINT"NAME: "N$
120 ::::PRINT"VORNAME: "V$
130 ::::PRINT"STRASSE: "S$
140 ::::PRINT"PLZ/ORT: "O$
150 END PROC
```

Dies ist eine übersichtlich angeordnete Routine, die mit Befehl 'EXEC AUSGABE ADRESSE' aufgerufen werden kann. Der Befehl END PROC entspricht dem bisherigen Befehl RETURN.



### 5.2.3 CALL

<b>FORMAT:</b>	<b>CALL LABEL</b>
<b>PARAMETER:</b>	<b>LABEL - SYMBOLISCHE SPRUNGADRESSE</b>
<b>FUNKTION:</b>	<b>SPRUNG ZU EINEM MIT PROC DEFINIERTEN LABEL</b>
<b>BEISPIEL:</b>	<b>CALL EINGABE</b>
<b>BEMERKUNG:</b>	<b>VERGLEICHBAR MIT GOTO</b>

Dies ist ein Befehl, der die mit PROC definierten Label anspricht. Aus einem 'GOTO 100' wird nun z.B. ein 'CALL ADR1'. Ein mit Label versehenes Programm kann nun beispielsweise so aussehen:

```
100 PROC GET
110 ::::GET X$:IF X$="" THEN CALL GET
120 ::::IF X$="A" THEN CALL INPUTA:ELSE:CALL GET
130 PROC INPUTA
    ...
```

Und wieder wurden alle GOTOs vermieden. Der Nachteil des RENUMBER von SIMON'S BASIC wird immer mehr in den Hintergrund gedrängt.

## 5.2.4 EXEC

**FORMAT:** EXEC LABEL  
**PARAMETER:** LABEL - SYMBOLISCHE SPRUNGADRESSE  
**FUNKTION:** AUFRUF EINES MIT PROC UND END PROC DEFINIERTEN  
UNTERPROGRAMMS  
**BEISPIEL:** EXEC EINGABE-ROUTINE  
**BEMERKUNG:** VERGLEICHBAR MIT GOSUB

Routinen (Unterprogramme) werden in jedem größeren Programm benötigt. Strukturierte Programmiersprachen wie z.B. COBOL oder PASCAL bestehen fast nur aus Routinen, die vom Hauptprogramm gesteuert werden. Die Möglichkeit bietet sich nun auch mit SIMON'S BASIC. Ein gut organisiertes und strukturiertes Programm sieht etwa so aus:

```
110 EXEC OPEN-DATEI
120 PROC START
130 :::EXEC ANZEIGEN EINGABEMASKE
140 :::EXEC FUELLEN EINGABEMASKE
150 :::EXEC SCHREIBEN DATENSATZ
160 :::PRINT "WEITERE DATEN ERFASSEN (J/N)?"
170 :::EXEC PROC GET J/N
180 :::IF AN$="J" THEN CALL START
190 CALL CLOSE-DATEI
200 EXEC PROGRAMMENDE
```

Dies ist ein Datenerfassungsprogramm, bei dem nur noch die entsprechenden Routinen entwickelt werden müssen. Ein für Sie sicher ungewöhnlicher Programmierstil, dessen Nachahmung jedoch lohnenswert ist. Man entwickelt zunächst das Hauptprogramm und dann die einzelnen Unterprogramme. Dieser strukturierte Programmierstil erleichtert auch die Fehlersuche.

Bei der symbolischen Programmierung gibt es zwei zusätzliche Fehlermeldungen, die Sie beachten sollten:

PROC NOT FOUND - Es wurde mit CALL oder EXEC ein Label angegeben, daß nicht definiert wurde.  
END PROC WITHOUT EXEC- Das Programm trifft auf END PROC, obwohl kein EXEC ausgeführt wurde.

## 5.2.5 CGOTO

**FORMAT:** CGOTO BERECHNUNG  
**PARAMETER:** BERECHNUNG - ARITHMETISCHER AUSDRUCK  
**FUNKTION:** SPRUNG ZU EINER BERECHNETEN ZEILENNUMMER  
**BEISPIEL:** CGOTO (N+I\*2)  
SPRING ZU DER ZEILE, DIE SICH BEI ABLAUF  
DES BEFEHLS DURCH DIE BERECHNUNG ERGIBT.

Bei der Befehlsbeschreibung zu ON ERROR wurden Sie bereits mit diesem Befehl konfrontiert. Dort wurde der Rücksprung von der Fehlerroutine berechnet aus Fehlerzeile plus Schrittweite. Ohne den Befehl CGOTO wäre dieser berechnete Rücksprung nicht möglich.

Wo kann man diesen Befehl nun einsetzen? Eine Möglichkeit ist, nach Ausgabe eines Menüs zum entsprechenden Programmteil zu verzweigen:

```
10 PRINT"PROGRAMMAUSWAHL:"
20 PRINT"-----"
30 PRINT" 1 - ARTIKEL ERFASSEN"
40 PRINT" 2 - ARTIKEL PFLEGEN"
50 PRINT" 3 - KUNDEN ERFASSEN"
60 PRINT" 4 - KUNDEN PFLEGEN"
70 PRINT" 5 - BESTELLUNG"
80 PRINT" 6 - DRUCKEN RECHNUNG"
90 PRINT" 7 - DRUCKEN LIEFERSCHEIN"
100 PROC GET
110 ::::GET X$:IF X$<"1" OR X$>"7" THEN CALL GET
120 Z=VAL(X$)
130 CGOTO Z*10+130
140 CALL ARTIKEL ERFASSEN
150 CALL ARTIKEL PFLEGEN
160 CALL KUNDEN ERFASSEN
170 CALL KUNDEN PFLEGEN
180 CALL BESTELLUNG
190 CALL DRUCKEN RECHNUNG
200 CALL DRUCKEN LIEFERSCHEIN
```

Dies ist eine sogenannte Sprungtabelle. Abhängig von der Menüauswahl, die nach dem GET-Befehl den Wert 1 bis 9 enthält, wird zu dem entsprechenden Unterprogramm verzweigt. Es ist nur eine der vielen möglichen Anwendungen mit CGOTO. Sicherlich haben Sie dies bereits erkannt und eigene Einsatzmöglichkeiten entdeckt.

### 5.2.6 TESTAUFGABEN

1) Welche Vorteile bieten symbolische Sprungadressen?

- A) Es werden die Befehle GOTO und GOSUB vermieden
  - B) Das Programm wird übersichtlicher
  - C) Unterprogramme werden vermieden
- 

2) Welcher Befehl ersetzt den standardmäßigen GOTO?

- A) CALL
  - B) EXEC
- 

3) Welcher Befehl ersetzt den standardmäßigen GOSUB?

- A) CALL
  - B) EXEC
-

4) Darf in der PROC-Zeile eine weitere Anweisung enthalten sein?

- A) ja
  - B) nein
- 

5) Welcher Befehl beendet ein mit EXEC aufgerufenes Unterprogramm?

- A) RETURN
  - B) END PROC
  - C) PROC
- 

6) Zu welcher Zeile verzweigt der folgende CGOTO-Befehl?

```
10 A=9  
20 CGOTO (A+1)*10+130
```

- A) Zeile 1310
- B) Zeile 210
- C) Zeile 230
- D) Zeile 220

## 6. KAPITEL VARIABLEN

Variablen nehmen im Standard-BASIC immer einen bestimmten Wert an. Wird dieser Wert verändert und nicht zwischengespeichert, so kann der ursprüngliche Wert nicht zurückgeholt werden. Mit SIMON'S BASIC haben Sie nun die Möglichkeit dem globalen Wert von Variablen (wie er normal definiert wird) an gewünschter Stelle einen lokalen Wert zuzuweisen (einen Wert, der nur für einen bestimmten Programmteil gültig ist). Der ursprüngliche Wert geht nicht verloren, sondern kann mit einem Befehl wieder hergestellt werden. Jede Variable kann also zwei Werte annehmen: den globalen Wert und den lokalen Wert.

### 6.1 LOCAL

**FORMAT:** LOCAL VARIABLE 1, VARIABLE 2,  
VARIABLE 3, .....

**PARAMETER:** VARIABLE - BELIEBIGE VARIABLE  
(Z.B. A, A\$, A%)

**FUNKTION:** FESTLEGEN DER LOKALEN VARIABLEN

**BEISPIEL:** LOCAL A\$, X\$, B  
DIE VARIABLEN A\$, X\$ UND B WERDEN ALS  
LOKALE VARIABLEN BESTIMMT

Die Anzahl der Variablen in umfangreichen Programmen ist meistens derartig groß, daß der Überblick leicht verloren geht. Mit dem Befehl LOCAL kann nun diese Anzahl stark reduziert werden. Sie können nun in Ihren Programmen ein und dieselbe Variable an verschiedenen Stellen verwenden. Bevor die Variablen verändert, also auf Ihren lokalen Wert gebracht werden, müssen Sie mit dem Befehl LOCAL als lokale Variablen bestimmt werden. Ein einfaches Beispiel:

```

100 REM *****
110 REM GLOBALE WERTE FESTLEGEN
120 REM *****
130 A=120:B%=850:XS="GLOBAL"
140 PRINT"GLOBALE WERTE: ";A;B%;XS
150 REM *****
160 REM LOKALE WERTE FESTLEGEN
170 REM *****
180 LOCAL A.B%.XS
190 A=240:B%=1700:XS="LOCAL"
200 PRINT"LOKALE WERTE: ";A;B%;XS

```

In Zeile 130 werden die Variablen mit den globalen Werten belegt. Die Zeile 180 bestimmt dann diese Variablen als lokale Variablen, denen in Zeile 190 neue Werte zugewiesen werden. Der Befehl LOCAL speichert die globalen Werte zwischen, bevor die lokalen Werte gesetzt werden. Mit dem folgenden Befehl kann der globale Wert wieder zurückgeholt werden.

## 6.2 GLOBAL

```

FORMAT:    GLOBAL
PARAMETER: KEINE
FUNKTION:  URSPRÜNGLICHE WERTE WIEDER HERSTELLEN

```

Dieser Befehl weist 1allen0 Variablen wieder den ursprünglichen Wert zu. Um die globalen Werte des vorherigen Programms wieder herzustellen, müssen folgende Zeilen angehangen werden:

```

210 *****
220 GLOBALE WERTE WIEDER HERSTELLEN
230 *****
240 GLOBAL
250 PRINT"GLOBALE WERTE: ";A;B%;XS

```

In der Zeile 240 werden also die alten Werte wieder übernommen. Die lokalen Werte sind jedoch nicht verloren gegangen. Mit einem erneuten LOCAL können die Variablen die lokalen Werte wieder annehmen. Geben Sie z.B. den Befehl 'LOCAL A' ein. Betrachten Sie nun den Wert der Variablen mit 'PRINT A'. Sie sehen, daß der lokale Wert (240) zurückgeholt wurde. Auf diese Weise können Sie beliebig zwischen globalen und lokalen Werten hin- und herschalten.

### 6.3 TESTAUFGABEN

1) Welche Aufgabe hat der Befehl LOCAL?

- A) globale Werte löschen und lokale Variablen bestimmen
  - B) globale Werte zwischenspeichern und lokale Variablen bestimmen
  - C) globale Werte zwischenspeichern und lokale Variablen Werte zuweisen
  - D) globale Werte löschen und lokale Variablen Werte zuweisen
- 

2) Welche Aufgabe hat der Befehl GLOBAL?

- A) lokale Werte löschen und globale Werte bestimmen
  - B) lokale Werte zwischenspeichern und globale Werte wiederherstellen
  - C) lokale Werte zwischenspeichern und globale Werte bestimmen
  - D) lokale Werte löschen und globale Werte wiederherstellen
- 

3) Welche Variablen können mit lokalen Werten belegt werden?

- A) Integervariablen (z.B. A%)
  - B) Stringvariablen (z.B. A\$)
  - C) Fließkommavariablen (z.B. A)
-



## 7. KAPITEL ZAHLENBEHANDLUNG

### 7.1 ARITHMETISCHE OPERATOREN

#### 7.1.1 MOD

```
FORMAT:      MOD (X,Y)  
PARAMETER:  X - DIVIDEND  
               Y - DIVISOR  
FUNKTION:   REST EINER INTEGER-DIVISION ERMITTELN  
BEISPIEL:  PRINT MOD(14.3)  
               ZEIGT DEN REST DER DIVISION (2)
```

Dieser Befehl ermittelt den ganzzahligen Rest einer Integer-Division. Was sind aber nun Integer-Zahlen? Integer-Zahlen sind ganzzahlig, d.h. ohne Nachkommawert. Beim CBM 64 gibt es nur 2-Byte Integerzahlen. Dies erspart viel Speicherplatz und erhöht die Rechengenauigkeit. Eine 2-Byte-Binärzahl kann maximal den Wert 65535 (2 hoch 16 minus 1) annehmen. Auch jede Hauptspeicheradresse Ihres Commodore 64 ist eine 2-Byte-Zahl.

Die Integervariablen des BASIC können jedoch auch negative Werte annehmen. Da das Bit 15 (ganz links) das Vorzeichen bestimmt, bleiben nur noch 15 Bits zur Speicherung des Wertes übrig. Eine Integervariable kann demnach die Werte -32768 bis +32767 annehmen. Wird ein anderer Wert zugewiesen (z.B. A%=33000), so folgt die Fehlermeldung "ILLEGAL QUANTITY ERROR".

Der MOD-Befehl akzeptiert nur positive Integerzahlen zwischen 0 und 65535. Jeder andere Wert verursacht auch hier die Fehlermeldung "ILLEGAL QUANTITY ERROR", also auch jede negative Zahl.

Das folgende Programm ermittelt den Rest einer Division mit Hilfe des MOD-Befehls:

```
10 INPUT "DIVIDENT" : ";X
20 INPUT "DIVISOR" : ";Y
30 PRINT "REST" : ";MOD (X,Y)
```

Der Befehl MOD kann also mit Variablen Parametern innerhalb eines Programms und direkt eingesetzt werden. Sie können z.B. direkt den Befehl 'PRINT MOD(144,7)' eingeben.

Da die Integervariablen des Commodore 64 (z.B. A%) bei einer Division nur den Vorkomawert speichern, muß der Rest mit dem Befehl MOD ermittelt werden. Ein Beispiel:

```
10 A%=50/12
20 A=50/12
30 PRINT "ERGEBNIS A%:"A%
40 PRINT "ERGEBNIS A : "A
```

Nach Ablauf des Programms sehen Sie das Ergebnis: Die Integervariable A% beinhaltet nur den Vorkomawert der Division. Die Gleitkommavariable A jedoch beinhaltet das korrekte Ergebnis. Der Befehl MOD ermöglicht nun auch die Ermittlung des Divisionsrestes:

```
10 A%=50/12
20 B%=MOD(50,12)
30 PRINT "ERGEBNIS : "A%"REST"B%
```

Somit kann nun auch eine Division mit Integervariablen durchgeführt werden.

### 7.1.2 DIV

```
FORMAT:      DIV (X.Y)
PARAMETER:   X - DIVIDEND
              Y - DIVISOR
FUNKTION:    VORKOMMAZAHL DER DIVISION ERMITTELN
BEISPIEL:    PRINT DIV (14.3)
              ZEIGT DIE VORKOMMAZAHL DER DIVISION (3)
```

Mit den beiden Befehlen MOD und DIV kann nun sowohl der ganzzahlige Quotient, als auch der Rest einer Integerdivision ermittelt werden. Das folgende Programm errechnet beides:

```
100 INPUT "DIVIDEND : ";X
110 INPUT "DIVISOR : ";Y
120 D=DIV(X,Y):M=MOD(X,Y)
130 PRINT"DAS ERGEBNIS IST"D"REST"M
```

Die Anwendung der beiden Befehle ist also recht einfach. Wie komfortabel die beiden Befehle sind, zeigt das folgende Beispiel:

In Kapitel 7.2 wird ein Befehl beschrieben, mit dem Sie Binärzahlen in Dezimalzahlen umwandeln können. Doch die Umwandlung einer Dezimalzahl in eine Binärzahl kann nur mit einem Programm gelöst werden. Das folgende Programm führt diese Umwandlung mit Hilfe der Befehle MOD und DIV auf der Grundlage der Restwertmethode durch:

```
100 PROC EINGABE
110 ::INPUT"DEZIMALZAHL (0-65535)";D
120 ::IF D>=0 AND D<=65535 THEN CALL AUSGABE
130 ::PRINTCHR$(145)::CALL EINGABE
140 PROC AUSGABE
150 ::PRINT"BINAERZAHL:"
160 ::FORI=1TO16
170 ::::R=MOD (D,2):D=DIV (D,2)
180 ::::PRINTTAB(16-I);R:CHR$(145)
190 ::NEXTI
```

Nachdem eine korrekte Zahl eingegeben wurde, wird aus dieser Dezimalzahl eine 16-Bit-Binärzahl gebildet.

### 7.1.3 FRAC

```
FORMAT:      FRAC(N)
PARAMETER:   N - ZAHL ODER AUSDRUCK (GLEITKOMMA)
FUNKTION:    BESTIMMEN DES NACHKOMMAWERTES EINER
              GLEITKOMMAZAHL
BEISPIEL:    PRINT FRAC(10/3)
              ZEIGT DEN NACHKOMMAWERT DES QUOTIENTEN
              (.33333333)
```

Wer schon einmal den Nachkommawert einer Zahl mit dem Standard-BASIC ermittelt hat, weiß das dies nicht sehr elegant gelöst werden kann. Es muß nämlich der Integerwert der Zahl von der Zahl abgezogen werden. Anstatt dem einfachen Befehl `PRINT FRAC(10/3)` muß `PRINT 10/3-INT(10/3)` eingegeben werden.

Bei den vorherigen Befehlen wurde ein Programm zur Umrechnung von Dezimalzahlen in Binärzahlen vorgestellt. Nun folgt ein Programm, mit dem Sie Dezimalzahlen in Hexadezimalzahlen umwandeln können:

```
100 DIMA$(15)
110 FORI=0TO15
120 ::READA$(I)
130 NEXTI
140 DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
150 PROC EINGABE
160 ::INPUT"DEZIMALZAHL (0-65535)":D
170 ::IFD>=0ANDD<=65535THENCALL AUSGABE
180 ::PRINTCHR$(145)::CALL EINGABE
190 PROC AUSGABE
200 ::PRINT"HEX-ZAHL:"
210 ::FORI=1TO4
220 :::R=16*FRAC(D/16):D=INT(D/16)
230 :::PRINTTAB(4-I):A$(R):CHR$(145)
240 ::NEXTI
```

Auch hier geben Sie eine Zahl zwischen 0 und 65535 ein, die anschließend in ein 4-stellige Hexadezimalzahl umgewandelt wird.

#### 7.1.4 EXOR

**FORMAT:** EXOR (N,N1)  
**PARAMETER:** N,N1 - ZU VERKNÜPFENDE ZAHLEN (0-65535)  
**FUNKTION:** ZWEI ZAHLEN IM EXCLUSIVE-ODER VERKNÜPFEN  
**BEISPIEL:** PRINT EXOR (255,1)  
 SCHALTET BIT 0 DES WERTES 255 UM, ALSO AUF NULL

Für den ein oder anderen ist die gesamte Bool'sche Algebra Neuland. Drei Verknüpfungen besitzt das Standard-BASIC bereits: OR, AND und NOT. Diese logischen Verknüpfungen werden z.B. in IF-Abfragen verwendet. Weiterhin kann mit diesen Verknüpfungen eine bestimmte Speicherstelle manipuliert werden. Die Verknüpfungen haben hier folgende Wirkung:

**OR - BITS EINSCHALTEN**  
**AND - BITS AUSSCHALTEN**

Da im Commodore 64 zahlreiche Register für die verschiedensten Zwecke enthalten sind (Grafik, Sound), werden diese Befehle für die Programmierung der Register benötigt. Jedes dieser Register umfaßt ein Byte (8 Bit). Diese Bits werden mit Hilfe der logischen Verknüpfungen ein- und ausgeschaltet. Doch wie schaltet man nun diese Bits? Dazu muß das Format eines Bytes bekannt sein:

	0	0	0	0	0	0	0	0
	-----							
<b>BIT</b>	7	6	5	4	3	2	1	0
	-----							
<b>WERTIGKEIT</b>	128	64	32	16	8	4	2	1

Ein Byte besteht also aus 8 Bits, die numeriert sind und eine bestimmte Wertigkeit haben. Soll nun eines dieser Bits

eingeschaltet werden, so muß mit der Wertigkeit dieses Bytes im ODER (OR) verknüpft werden. Ein Beispiel: Das Bit 4 der Speicherstelle 53248 soll eingeschaltet werden. Die entsprechende Operation wäre dann:

***POKE 53248.PEEK(53248) OR 16***

Es wurde der Inhalt der Speicherstelle mit der Wertigkeit des Bits 4 (16) verknüpft. Ein weiteres Beispiel: Es sollen die Bits 0,1,2 und 3 auf 1 gesetzt werden. Dazu müssen die einzelnen Wertigkeiten aufaddiert werden. Der entsprechende Befehl ist dann:

***POKE 53248.PEEK(53248) OR 15***

Schwieriger ist das Ausschalten der Bits, da dazu nicht mit der Wertigkeit der Bits im UND (AND) verknüpft werden muß, sondern mit 255 minus der Wertigkeit. Soll das Bit 0 ausgeschaltet, also auf 0 gesetzt werden, so muß mit 255-1 (254) verknüpft werden. Die entsprechenden Befehle zum Ausschalten der zuvor eingeschalteten Bits sehen dann wie folgt aus:

***POKE 53248.PEEK(53248) AND (255-16)***

***POKE 53248.PEEK(53248) AND (255-15)***

***ODER MIT EINEM BEFEHL***

***POKE 53248.PEEK(53248) AND (255-31)***

SIMON'S BASIC bietet nun eine zusätzliche Verknüpfung, das EXOR an. Mit EXOR werden Bits umgeschaltet. D.h., war ein Bit vor der EXOR-Verknüpfung gesetzt, so wird es gelöscht und umgekehrt. Ein Beispiel: Das Bit 7 der Speicherstelle 53250 soll umgeschaltet werden. Der entsprechende Befehl dazu ist:

***POKE 53250.EXOR(PEEK(53250),128)***

Die Schreibweise ist etwas abgewandelt, da das Format des EXOR nicht wie bei OR und AND 'n EXOR n1' sondern

'EXOR(n,n1)' ist. Vielleicht hätte SIMON'S BASIC sich hier anpassen sollen.

### 7.1.5 TESTAUFGABEN

1) Welches Ergebnis zeigt der Befehl PRINT MOD (33,7)?

- A) 4
  - B) 5
  - C) 0
  - D) 28
- 

2) Welches Ergebnis zeigt der Befehl PRINT DIV (33,7)?

- A) 4
  - B) 5
  - C) 0
  - D) 28
- 

3) Welchen Höchstwert dürfen die MOD- und DIV-Parameter nicht überschreiten?

- A) 32767
  - B) 65535
  - C) 1E+38
- 

4) Welchen Wert ermittelt der Befehl FRAC?

- A) den Nachkommawert einer Integerzahl
  - B) den Vorkommawert einer Integerzahl
  - C) den Nachkommawert einer Gleitkommazahl
  - D) den Vorkommawert einer Gleitkommazahl
- 

5) Die Bits 0 und 1 der Speicherstelle 53300 sollen umgeschaltet werden. Welcher Befehl ist richtig?

- A) POKE 53300,PEEK(53300) EXOR 3
  - B) POKE 53300,EXOR(PEEK(53300),3))
-

## 7.2 ZAHLENUMWANDLUNG

### 7.2.1 %

```
FORMAT:      % BINÄRWERT
PARAMETER:   BINÄRWERT - ACHSTELLIGE BINÄRZAHL
FUNKTION:    UMRECHNEN BINÄR - DEZIMAL
BEISPIEL:    PRINT %11111111
              ZEIGT DEN DEZIMALWERT 255 AN
```

Mit dem Zeichen '%' werden 8-stellige Binärzahlen gekennzeichnet, die dann angezeigt oder abgespeichert werden können. Die 8-stellige Binärzahl darf nur die Ziffern 0 und 1 enthalten, da sonst die Fehlermeldung "NOT BINARY CHAR" ausgegeben wird.

Nicht nur die Schreibweise

```
PRINT %00001111
```

ist möglich, sondern auch

```
POKE 53280,%00011100
ODER A=%00001110
```

Die Binärzahl wird jeweils in eine Dezimalzahl umgewandelt. Bei der logischen Verknüpfung bietet diese Schreibweise wesentliche Vorteile. Um einzelne Bits zu manipulieren, muß die Verknüpfung nicht erst vom Programmierer umgerechnet werden. Der Befehl zum Einschalten der Bits 0 und 2 der Speicherstelle 49152 kann dadurch einfacher dargestellt werden:

```
POKE 49152.PEEK (49152) OR %00000101
```

Die %-Schreibweise vereinfacht das Arbeiten mit Bits und Bytes.

Im Kapitel 7.1.2 finden Sie ein Programm, mit dem Sie Dezimalzahlen in 16-Bit-Binärzahlen umwandeln können.



## 7.2.2 §

**FORMAT:** \$ HEX-WERT  
**PARAMETER:** HEX-WERT - VIERSTELLIGE HEXADEZIMALZAHL  
**FUNKTION:** UMWANDLUNG HEXADEZIMAL - DEZIMAL  
**BEISPIEL:** PRINT \$00FF  
ZEIGT DEN DEZIMALWERT 255 AN

Hier wird eine vierstellige Hexadezimalzahl in eine Dezimalzahl umgewandelt. Wird keine vierstellige Zahl mit gültigen Hexadezimalziffern angegeben, so wird die Fehlermeldung "NOT HEX CHAR" ausgegeben.

Diese Schreibweise kann sehr hilfreich sein. Auch BASIC-Programmierer setzen Routinen des Betriebssystems ein. Interessante Routinen entnimmt man dem ROM-Listing des Commodore 64, das im DATA BECKER BUCH "64 Intern" enthalten ist. Die Adressen dieser Routinen sind zumeist hexadezimal angegeben. Will man nun eine Routine benutzen, so muß die Adresse erst dezimal umgerechnet werden. Das entfällt mit Anwendung der \$-Kennzeichnung. Wollen Sie z.B. die Betriebssystem-Adresse \$FCE2 (Kaltstart) dezimal umrechnen so geben Sie lediglich den folgenden Befehl ein:

**PRINT \$FCE2**

Der Rechner zeigt dann die dezimale Adresse dieser Routine an (64738). Mit SYS 64738 können Sie diese Routine, die den Rechner in den Zustand nach dem Einschalten versetzt starten. Einfacher ist es jedoch, die Routine direkt mit folgendem Befehl aufzurufen:

**SYS \$FCE2**

Die \$-Kennzeichnung kann also auch hier überall eingesetzt werden, wo sonst dezimale Werte eingesetzt wurden.

### 7.2.3 TESTAUFGABEN

1) Welcher der folgenden Befehle führt zur Fehlermeldung?

- A) PRINT %10101000
  - B) POKE 53000,EXOR (PEEK(53000),%00000011)
  - C) PRINT \$0011
- 

2) Was muß beim Einsatz der %-Kennzeichnung beachtet werden?

- A) Die Binärzahl darf max. 8-stellig sein
  - B) Die Binärzahl darf max. 16-stellig sein
  - C) Es dürfen ausschließlich die Binärziffern 0 und 1 verwendet werden
  - D) Es darf nur die Binärziffer 1 verwendet werden
- 

3) Welcher der folgenden Befehle führt zu Fehlermeldung?

- A) PRINT \$FF
  - B) POKE 1024,\$0000
  - C) A%=\$00FF+\$000F
- 

4) Was muß beim Einsatz der \$-Kennzeichnung beachtet werden?

- A) Die Hex-Zahl darf max. 2-stellig sein
  - B) Die Hex-Zahl darf max. 4-stellig sein
  - C) Es dürfen nur Dezimalziffern (0-9) verwendet werden
  - D) Es dürfen nur Hexadezimalziffern (0-9 und A-F) verwendet werden
-

## 8. KAPITEL STRINGOPERATIONEN

### 8.1 INSERT

```
FORMAT:      INSERT ("STRING1", "STRING2", POS)
PARAMETER:   STRING1 - EINZUSETZENDER STRING
              STRING2 - AUßERER STRING
              POS    - POSITION, NACH DER EINGE-
                      FÜGT WERDEN SOLL
FUNKTION:    EINSETZEN EINES STRINGS IN EINEN ANDEREN
BEISPIEL:    10 X$=INSERT(" CBM V2", "EXPANDED BASIC", 8)
              20 PRINT X$
              FÜGT DEN 1. STRING AB DER 8. STELLE IM
              2. STRING EIN: "EXPANDED CBM V2 BASIC"
              WIRD ANGEZEIGT.
```

Mit diesem Befehl können Sie einen String in einen anderen an beliebiger Position einsetzen. Neben Stringkonstanten können auch Stringvariablen angegeben werden:

```
10 A$=" CBM V2"
20 B$="EXPANDED BASIC"
30 X$=INSERT (A$,B$,8)
40 PRINT X$
```

Zwei Stringvariablen wurden hier eingesetzt. Aber auch der dritte Parameter, die Position, kann als Variable angegeben werden. Somit sind nun alle 3 Parameter des INSERT als Variablen eingesetzt:

```
10 A$=" CBM V2"
20 B$="EXPANDED BASIC"
30 C=8
40 X$=INSERT (A$,B$,C)
50 PRINT X$
```

Beim Einsatz des INSERT-Befehls müssen folgende Punkte beachtet werden:

1. Der einzusetzende String darf nicht größer als der äußere String sein.
2. Der resultierende String darf 255 Zeichen nicht überschreiten.
3. Die Positionsangabe darf höchstens die Länge des äußeren Strings minus 1 betragen. Es darf also nicht angefügt, sondern nur eingefügt werden.
4. Wird der Befehl im Direkt-Modus eingegeben (nicht in einer Programmzeile), so dürfen nur Stringvariablen, keine Stringkonstanten (z.B. "XYZ") verwendet werden.

Bei Nichtbeachtung der Punkte 1 und 3 wird die Fehlermeldung "INSERT TOO LARGE" ausgegeben. Da die Länge eines Strings auf keinen Fall 255 Zeichen überschreiten darf, wird bei Mißachtung von Punkt 2 die Fehlermeldung "STRING TOO LARGE" ausgegeben.

Der Punkt 4 erscheint sicher unwahrscheinlich. Zwar ist der Einsatz von INSERT im Direkt-Modus nicht unbedingt erforderlich, doch die Reaktion von SIMON'S BASIC gibt zu denken. Nach Ablauf des zuletzt genannten Programms bleiben die Stringvariablen A\$ und B\$ erhalten, so daß der folgende Befehl im Direkt-Modus einen Erfolg vorweist:

***PRINT INSERT(A\$,B\$,8)***

Es wird der korrekte String "EXPANDED CBM V2 BASIC" ausgegeben. Auch wenn der erste Parameter gegen eine Stringkonstante ausgetauscht wird, ist das Ergebnis korrekt:

***PRINT INSERT(" CBM V2",B\$,8)***

Doch nun aufgepasst: Wird auch der zweite Parameter als Stringkonstante angegeben, so hat dies erstaunliche Folgen:

***PRINT INSERT(" CBM V2",EXPANDED BASIC",8)***

Ein völlig durcheinander geratener String erscheint auf dem Bildschirm:

## EXPANDEDD BASIC BASIC

Doch wie gesagt, wird dieser Befehl kaum im Direkt-Modus eingegeben. Diese SIMON'S BASIC-Macke kann demnach ignoriert werden. Im Programm arbeitet der Befehl problemlos.

### 8.2 INST

```
FORMAT:      INST ("STRING1", "STRING2", POS)
PARAMETER:   STRING1 - ÜBERSCHREIBENDER STRING
              STRING2 - ALTER STRING
              POS    - POSITION, NACH DER ÜBERSCHRIEBEN
                    WERDEN SOLL
FUNKTION:    ÜBERSCHREIBEN EINES STRINGS MIT EINEM
              ANDEREN
BEISPIEL:    10 X$=INST ("130", "MENGE 500 STCK", 6)
              20 PRINT X$
              ÜBERSCHREIBT DEN 2. STRING MIT DEM 1.
              AB DER POSITION NACH 6; DER STRING
              "MENGE 130 STCK" WIRD AUSGEGEBEN
```

Die Parameter dieses Befehls sind indentisch mit denen des zuvor beschriebenen INSERT-Befehls. Der Befehl INST jedoch fügt nicht ein, sondern überschreibt den 2. String. Der 1. String darf nicht größer sein als der 2. String. Schlimme Folgen hat es, wenn der 2. Parameter eine Stringvariable ist, die nichts beinhaltet. Dann steigt SIMON'S BASIC aus und läßt keine Stringverarbeitung mehr zu.

Die Positionsangabe darf den 2. String nicht überschreiten. Umfaßt der 2. String z.B. 20 Zeichen, so darf auch die Positionsangabe 20 nicht überschreiten. Wird dies mißachtet, so wird zwar keine Fehlermeldung ausgegeben, doch ist der entstehende String nicht ordnungsgemäß aufgebaut. Natürlich darf der entstehende String auch nicht 255 Zeichen überschreiten.

Ein interessantes Einsatzgebiet findet der INST-Befehl z.B. beim Aufbau eines Datensatzes zur relativen Datenspeicherung. Ein Datensatz besteht meist aus mehreren Feldern, die sich immer an ein und derselben Position befinden müssen. Ein Datensatz einer Adressendatei z.B. kann folgendermaßen aufgebaut sein:

<b>FELD</b>	<b>POSITION IM DATENSATZ</b>	<b>VARIABLE</b>
-----		
<b>NAME</b>	<b>0</b>	<b>N\$</b>
<b>VORNAME</b>	<b>25</b>	<b>V\$</b>
<b>STRASSE</b>	<b>45</b>	<b>S\$</b>
<b>PLZ/ORT</b>	<b>75</b>	<b>O\$</b>

Diese vier Felder sollen nun zu einem String (R\$) verknüpft werden, dessen Aufbau der oben genannten Struktur entspricht. Der String R\$ soll 100 Zeichen umfassen und muß zuvor mit Leerzeichen gefüllt werden. Der folgende Programmausschnitt zeigt, wie der Datensatz im String R\$ aufgebaut wird:

```

100 R$=DUP(" ",100)
110 R$=INST(N$,R$,0)
120 R$=INST(V$,R$,25)
130 R$=INST(S$,R$,50)
140 R$=INST(O$,R$,75)

```

Der 100 Leerzeichen umfassende String R\$ wird in der Zeile 100 mit dem Befehl DUP erzeugt (DUP siehe Kapitel 8.4). Die folgenden 4 INST-Befehle setzen die vier Variablen an den entsprechenden Platz im Datensatz R\$. Die gesamte relativ aufgebaute Adressendatei ist dann einheitlich strukturiert, so daß nach dem Einlesen eines Datensatzes jedes beliebige Feld an der richtigen Stelle aufgefunden wird. Allen Lesern, die mehr über relative Datenspeicherung mit dem Floppylaufwerk VC-1541 wissen möchten, empfehlen wir das DATA BECKER-Buch "Das große Floppybuch".

Weiterhin eignet sich der INST-Befehl auch sehr gut zur Ausgabe von Tabellen auf dem Drucker. Die einzelnen Felder der Druckzeile werden vor dem Drucken an die entsprechende

Stelle eines Strings gesetzt. Nach der Aufbereitung wird dieser String dann gedruckt. Im Prinzip ist dies vergleichbar mit der zuvor beschriebenen Datensatzaufbereitung.

Wer diese Stringaufbereitung zuvor mit dem Standard-BASIC gelöst hat, weiß, welche Erleichterung der INST-Befehl bietet.

### 8.3 PLACE

```
FORMAT: PLACE ("STRING1".STRINGV)
PARAMETER: STRING1 - GESUCHTER STRING
                STRINGV - STRINGVARIABLE, IN DER GESUCHT
                    WERDEN SOLL
FUNKTION: SUCHEN EINER ZEICHENFOLGE IN EINEM STRING
BEISPIEL: A$="123*4567"
                PRINT PLACE ("*",A$)
                ZEIGT DIE POSITION DES STERNS IM STRING A$
```

Mit PLACE steht Ihnen ein sehr effektiver Befehl zur Verfügung, der mit dem Standard-BASIC nur schwer zu ersetzen ist. Der Befehl PLACE läßt sich auch gut mit INST kombinieren. Ein Beispiel: Der Dezimalpunkt ist beim Commodore 64 wie auch bei allen anderen Rechnern gemäß der amerikanischen Norm ein Punkt, der deutsche Dezimalpunkt ist jedoch ein Komma. Mit PLACE können nun Beträge nach deutscher Norm ausgegeben werden. Halten Sie sich zunächst einmal das folgende Beispiel vor Augen:

```
10 INPUT"BETRAG: ";B
20 B$=STR$(B)
30 P=PLACE(".",B$)
40 B$=INST(".",B$,P-1)
50 PRINT"BETRAG: ";B$
```

In Zeile 10 wird ein Betrag eingelesen und in eine numerische Variable gespeichert, da mit diesem Betrag evtl. gerechnet wird. Zeile 20 wandelt die numerische Variable in eine Stringvariable um. Nun wird die Position des Punktes in Zeile 30 ermittelt und in der Variablen P gespeichert. Danach

ersetzt die Zeile 40 den Punkt, dessen Position in P-1 enthalten ist, gegen ein Komma. Die Position im INST-Befehl muß um 1 verringert werden, da die Befehle INSERT und INST beim Zählen mit 0 beginnen. Die Position 1 des PLACE-Befehls entspricht z.B. der Position 0 beim INSERT oder INST. Die Lösung dieses Problems mit dem Standard-BASIC würde ca. die dreifache Zeilenzahl erfordern und wäre natürlich auch relativ unübersichtlich.

Nicht nur ein Zeichen, sondern eine beliebige Zeichenfolge kann mit dem PLACE-Befehl aufgesucht werden. Das folgende Beispiel bestätigt dies:

```
10 A$="DATA BECKER BUECHER MACHEN SCHLAU"  
20 PRINT PLACE("BUECHER",A$)
```

Geben Sie dieses Programm ein und starten es mit RUN. Am Bildschirm wird die Zahl 13 ausgegeben, was der Position der Zeichenfolge "BUECHER" entspricht.

Dieser komfortable Befehl reizte uns so sehr, daß wir ein Programm zum Speichern und Suchen von unformatierten Informationen entwickelten. Jede Information besteht aus maximal 2 Bildschirmzeilen und wird auf eine Diskette gespeichert. Die gesamte Informationsdatei wird im Rechner als Tabelle verwaltet, was in Verbindung mit dem Befehl PLACE ein sehr schnelles Auffinden von Stichworten ermöglicht. Nun zunächst das Listing dieses Programms:



```

1000 REM *****
1010 REM   VORBEREITUNGEN
1020 REM *****
1030 POKE 53280,2:POKE53281,2
1040 PRINTCHR$(158)
1050 DIM I$(4000)
1060 PROC MENUE
1070 ::REM *****
1080 ::REM   MENUE
1090 ::REM *****
1100 ::EXEC PROGRAMMKOPF
1110 ::PRINT" WAEHLEN SIE DIE GEWUENSCHTE FUNKTION:"
1120 ::PRINT" ";DUP("-",37):PRINT
1130 ::PRINTTAB(8);" -1- INFODATEI LADEN"
1140 ::PRINTTAB(8);" -2- INFODATEI SICHERN"
1150 ::PRINTTAB(8);" -3- INFOS EINGEBEN"
1160 ::PRINTTAB(8);" -4- INFOS LOESCHEN"
1170 ::PRINTTAB(8);" -5- INFOS SUCHEN":PRINT
1180 ::PRINTTAB(8);" -6- PROGRAMMENDE"
1190 ::LOOP
1200 ::::GETX$
1210 ::::X=VAL(X$)
1220 ::::EXIT IF X>0 AND X<7
1230 ::END LOOP
1240 ::IF X=1THEN EXEC INFODATEI LADEN
1250 ::IF X=2THEN EXEC INFODATEI SICHERN
1260 ::IF X=3THEN EXEC INFOS EINGEBEN
1270 ::IF X=4THEN EXEC INFOS LOESCHEN
1280 ::IF X=5THEN EXEC INFOS SUCHEN
1290 ::IF X=6THEN EXEC PROGRAMMENDE
1300 ::CALL MENUE
1310 PROC INFODATEI LADEN
1320 ::REM *****
1330 ::REM   INFODATEI LADEN
1340 ::REM *****
1350 ::EXEC PROGRAMMKOPF
1360 ::INPUT"NAME DER INFODATEI: ";DNS$
1370 ::OPEN1.8.2.DNS$:CLOSE1
1380 ::OPEN15.8.15
1390 ::INPUT#15:F1$,F2$,F3$,F4$

```

```

1400 ::IF F1$="00"THEN CLOSE 15:CALL LESEN
1410 ::::PRINT:PRINT"DISKETTENFEHLER!"
1420 ::::PRINT"-----"
1430 ::::PRINTF1$,"F2$","F3$","F4$
1440 ::::CLOSE15
1450 ::::CALL RUECKSPRUNG1
1460 :::PROC LESEN
1470 ::::OPEN1.8.2.DNS
1480 ::::PRINT"NEU- ODER DAZULADEN (N/D)?"
1490 ::::REPEAT
1500 ::::::GET X$
1510 ::::UNTIL X$="N" OR X$="D"
1520 ::::IF X$="N" THEN Y=0
1530 ::::LOOP
1540 ::::::Y=Y+1
1550 ::::::INPUT#1,I$(Y)
1560 ::::::EXIT IF ST=64
1570 ::::END LOOP
1580 ::::CLOSE1
1590 ::::PRINT"RECHNER BEINHALTET"Y"INFORMATIONEN"
1600 :::PROC RUECKSPRUNG1
1610 ::::PRINT:PRINT">>RETURN<<"
1620 ::::REPEAT
1630 ::::::GETX$
1640 ::::UNTIL X$=CHR$(13)
1650 END PROC
1660 PROC INFODATEI SICHERN
1670 :::REM *****
1680 :::REM  INFODATEI SICHERN
1690 :::REM *****
1700 :::EXEC PROGRAMMKOPF
1710 :::IF Y=0THENPRINT"KEINE DATEN!!":CALL RUECKSPRUNG2
1720 :::INPUT"DATEINAME :":DNS$
1730 :::PRINT"DATEI "DNS" WIRD GESICHERT!"
1740 :::OPEN 1.8.2.CHR$(64)+": "+DNS$+",S.W"
1750 :::FOR I=1TOY
1760 ::::PRINT#1,I$(I)
1770 :::NEXT I
1780 :::CLOSE 1
1790 :::PROC RUECKSPRUNG2

```

```

1800 ::::PRINT:PRINT">>RETURN<<"
1810 ::::REPEAT
1820 ::::::GET X$
1830 ::::UNTIL X$=CHR$(13)
1840 END PROC
1850 PROC INFOS EINGEBEN
1860 ::REM *****
1870 ::REM   INFOS EINGEBEN
1880 ::REM *****
1890 ::EXEC PROGRAMMKOPF
1900 ::IF Y>0 THEN CALL EINGABE
1910 ::::INPUT"DATEINAME: ";DNS
1920 ::PROC EINGABE
1930 ::EXEC PROGRAMMKOPF
1940 ::::Y=Y+1
1950 ::::PRINT"GEBEN SIE DIE INFORMATION NR."Y"EIN:"
1960 ::::PRINT"(MAXIMAL 2 ZEILEN)"
1970 ::::PRINTDUP("-",39)
1980 ::::INPUT I$(Y)
1990 ::::PRINTDUP("-",40);
2000 ::::PRINT"RICHTIG (J/N)?"
2010 ::::REPEAT
2020 ::::::GETX$
2030 ::::UNTIL X$="J" OR X$="N"
2040 ::::IF X$="N" THEN Y=Y-1:CALL EINGABE
2050 ::::PRINT"WEITERE EINGABEN (J/N)?"
2060 ::::REPEAT
2070 ::::::GET X$
2080 ::::UNTIL X$="J" OR X$="N"
2090 ::::IF X$="J"THEN CALL EINGABE
2100 END PROC
2110 PROC INFOS LOESCHEN
2120 ::REM *****
2130 ::REM   INFOS LOESCHEN
2140 ::REM *****
2150 ::EXEC PROGRAMMKOPF
2160 ::IF Y=0THENPRINT"KEINE DATEN!!":CALL RUECKSPRUNG3
2170 ::INPUT"NUMMER DES INFOS: ";N
2180 ::PRINTDUP("-",39)
2190 ::PRINT" ";I$(N)

```

```

2200 ::PRINTDUP("-".39)
2210 ::PRINT"LOESCHEN (J/N)?"
2220 ::REPEAT
2230 ::::GET X$
2240 ::UNTIL X$="J" OR X$="N"
2250 ::IF X$="N" THEN CALL RUECKSPRUNG3
2260 ::FORI=N TO Y
2270 ::::I$(I)=I$(I+1)
2280 ::NEXT I
2290 ::Y=Y-1
2300 ::PRINT"INFO IST GELOESCHT!"
2310 ::PROC RUECKSPRUNG3
2320 ::::PRINT">>RETURN<<"
2330 ::::REPEAT
2340 ::::::GET X$
2350 :::::UNTIL X$=CHR$(13)
2360 END PROC
2370 PROC INFOS SUCHEN
2380 ::REM *****
2390 ::REM      INFOS SUCHEN
2400 ::REM *****
2410 ::EXEC PROGRAMMKOPF
2420 ::IF Y=0THENPRINT"KEINE DATEN!!":CALL RUECKSPRUNG4
2430 ::PRINT"WIEVIEL STICHWORTE (1-5)?"
2440 ::REPEAT
2450 ::::GET X$
2460 ::UNTIL X$>"0" AND X$<"6"
2470 ::S=VAL(X$):VK$="0"
2480 ::EXEC PROGRAMMKOPF
2490 ::IFS=1THENCALL WEITER
2500 ::PRINT"UND/ODER VERKNUEPFUNG (U/O)?"
2510 ::REPEAT
2520 ::::GET X$
2530 ::UNTIL X$="U" OR X$="O"
2540 ::VK$=X$
2550 ::PROC WEITER
2560 ::PRINT"BILDSCHIRM ODER DRUCKER (B/D)?"
2570 ::REPEAT
2580 ::::GET X$
2590 ::UNTIL X$="B" OR X$="D"

```

```

2600 ::IF X$="B"THEN OPEN1.3:ELSE:OPEN1.4
2610 ::AU$=X$:PRINTDUP("-",20)
2620 ::FOR I=1TOS
2630 ::::PRINT"STICHWORT"I": ";
2640 ::::INPUTS$(I)
2650 ::NEXTI
2660 ::FOR S1=1TOY
2670 ::::U1=0:FL=0
2680 ::::FORS2=1 TO S
2690 :::::Z=PLACE (S$(S2),I$(S1))
2700 :::::IF Z<>0 AND VK$="0"THEN FL=1:EXEC AUSGABE
2710 :::::IF Z=0 AND VK$="U"THEN CALL NEXT S1
2720 :::::IF Z<>0 AND VK$="U"THENU1=U1+1
2730 :::::IFFL=1THENCALL NEXT S1
2740 ::::NEXT S2
2750 ::::IF U1=S THEN EXEC AUSGABE
2760 ::PROC NEXT S1
2770 ::NEXT S1
2780 ::PRINT"SUCHE BEENDET!"
2790 ::CALL RUECKSPRUNG4
2800 ::PROC AUSGABE
2810 ::::IF AU$="B"THEN EXEC PROGRAMMKOPF
2820 ::::PRINT#1,"INFO NR.",S1
2830 ::::PRINT#1,DUP("-",39)
2840 ::::PRINT#1,I$(S1)
2850 ::::PRINT#1,DUP("-",39)
2860 ::::PRINT#1
2870 ::::IF AU$="D"THEN CALL RUECK1
2880 :::::PRINT">>RETURN<<"
2890 :::::REPEAT
2900 :::::GET X$
2910 :::::UNTIL X$=CHR$(13)
2920 ::::PROC RUECK1
2930 ::END PROC
2940 ::PROC RUECKSPRUNG4
2950 ::::PRINT">>RETURN<<"
2960 ::::REPEAT
2970 :::::GETX$
2980 :::::UNTIL X$=CHR$(13)
2990 ::::CLOSE1

```

```

3000 END PROC
3010 PROC PROGRAMMENDE
3020 ::REM *****
3030 ::REM      PROGRAMMENDE
3040 ::REM *****
3050 ::EXEC PROGRAMMKOPF
3060 ::PRINT"SIND SIE SICHER (J/N)"
3070 ::REPEAT
3080 ::::GET X$
3090 ::UNTIL X$="J" OR X$="N"
3100 ::IF X$="N" THEN END PROC
3110 ::EXEC PROGRAMMKOPF
3120 ::PRINT"      DAS PROGRAMM KANN MIT DEM BEFEHL":PRINT
3130 ::PRINT"              'CALL MENUE':PRINT
3140 ::PRINT"      WIEDER GESTARTET WERDEN, OHNE DASS":PRINT
3150 ::PRINT"              DATEN VERLOREN GEHEN !!"
3160 ::END
3170 PROC PROGRAMMKOPF
3180 ::REM *****
3190 ::REM      PROGRAMMKOPF
3200 ::REM *****
3210 ::PRINTCHR$(147);
3220 ::PRINTDUP("=",40);
3230 ::PRINTCHR$(18);"      UNFORMATIERTES INFORMATIONS";
3235 ::PRINT"SYSTEM      ";CHR$(146);
3240 ::PRINTDUP("=",40);
3250 PRINT"INFOS:"Y"      DATEINAME: "DN$
3260 ::PRINTDUP("=",40);
3270 ::PRINT:PRINT
3280 END PROC

```

Es lohnt sich wirklich, dieses Programm einzugeben. Ihnen steht damit ein Informationssystem zur Verfügung, das Sie zu den verschiedensten Zwecken einsetzen können.

Nach dem Starten meldet sich das Programm mit einem 6 Funktionen enthaltenden Menü:

### **-1- INFODATEI LADEN**

-----

Nach Auswahl dieses Menüs wird nach dem Namen der Infodatei gefragt. Dies ist der Name, den Sie zuvor beim Speichern definiert haben. Nun bestimmen Sie, ob die Datei neu- oder dazugeladen werden soll. Wenn Sie z.B. bereits eine Datei im Rechner gespeichert haben, so kann eine weitere Datei hinzugeladen werden. Eine auf diese Weise zusammengesetzte Datei kann dann unter einem anderen Namen wieder abgespeichert werden.

Wenn das Programm die im Speicher befindlichen Informationssätze gemeldet hat, kommen Sie mit Betätigung der RETURN-Taste wieder zurück ins Menü.

### **-2- INFODATEI SPEICHERN**

-----

Dieses Unterprogramm meldet "KEINE DATEN!!" wenn es aufgerufen wird, obwohl keine Daten im Rechner gespeichert sind.

Wenn Sie nun den Dateinamen angeben, wird die Datei auf Diskette gesichert. Doch Vorsicht: Eine evtl. unter diesem Namen bereits vorhandene Datei wird überschrieben! Auch hier kommen Sie mit der RETURN-Taste wieder zurück ins Menü.

### **-3- INFOS EINGEBEN**

-----

Diese Funktion wählen Sie aus, wenn Sie erstmals Daten erfassen oder wenn Sie die bereits gespeicherte Datei erweitern möchten. Sind keine Informationen gespeichert, so fragt das Programm nach dem Dateinamen, der dann wie auch die Anzahl der Informationen im Rechner, in der Kopfzeile angezeigt wird.

Haben Sie diesen Namen eingegeben, erscheint die Eingabemaske auf dem Bildschirm. Sie können nun einen Informationssatz eingeben, der maximal 2 Bildschirmzeilen umfaßt. Nun wird

gefragt, ob die Eingabe korrekt ist. Geben Sie hier 'N' für nein ein, so können Sie die Eingabe wiederholen. Bei 'J' fragt das Programm, ob Sie weitere Informationen erfassen möchten. Geben Sie hier 'N' ein, so erscheint wieder das Menü.

#### **-4- INFOS LOESCHEN**

-----

Natürlich werden manche Daten mit der Zeit nicht mehr benötigt. Z.B. Termine, die bereits verstrichen sind. In diesem Unterprogramm können Sie nun gezielt Datensätze löschen. Dazu geben Sie die Nummer des Infos ein. Ist Ihnen diese Nummer nicht bekannt, so muß Sie mit Hilfe von Menü -6- ermittelt werden.

Bevor Sie endgültig entscheiden, ob der Datensatz gelöscht wird, erscheint der komplette Informationssatz noch einmal auf dem Bildschirm. Wenn Sie hier die Taste 'N' betätigen, meldet sich wieder das Menü. Beachten Sie, daß nach Löschen eines Datensatzes die Datei im Rechner umorganisiert wird. Alle nachfolgenden Informationssätze rücken um eine Stelle auf.

#### **-5- INFOS SUCHEN**

-----

Dies ist der Kern des Programms. Die größte Datei nutzt Ihnen wenig, wenn Sie nicht innerhalb kurzer Zeit auf wichtige Informationen zugreifen können. Das ist wie ein Telefonbuch, das nach Telefonnummer sortiert ist. In diesem Programmteil haben Sie nun die Möglichkeit, Ihre Datei komfortabel auszuwerten und die Auswertung auf Drucker oder Bildschirm sichtbar zu machen.

Nach Anwahl dieser Funktion geben Sie die Anzahl der Stichworte (1-5) an, nach denen in jedem Datensatz gesucht werden soll. Wenn Sie mehr als ein Stichwort auswählen, so folgt eine wichtige Abfrage: Wollen Sie im UND oder im ODER verknüpfen. Der ein oder andere unserer Leser versteht vielleicht etwas vom Teppich knüpfen, doch die logischen Verknüpfungen sind nicht immer bekannt. Angenommen Sie wollen nach 2 Stichworten suchen. Nun gibt es zwei Möglichkeiten, die beiden Stichworte zur Suche einzusetzen. Soll der gesuchte Datensatz beide Stichworte enthalten, verknüpfen Sie



diese mit UND, sie drücken also die Taste 'U'. Soll der Informationssatz jedoch nur mindestens eines der beiden Stichworte enthalten ("entweder oder"), so verknüpfen Sie mit ODER, geben also 'O' ein. Es ist also gar nicht so schwer. Von diesen Stichworten können Sie also bis zu 5 eingeben. Nach der Frage zur Verknüpfung bestimmen Sie, ob die gefundenen Informationen auf Drucker oder Bildschirm ausgegeben werden sollen. Danach geben Sie Ihre Stichworte ein. Wenn das Programm am Ende der Datei angelangt ist, wird dies gemeldet. Sie kommen nun wieder mit RETURN zurück ins Menü.

#### **-6- PROGRAMMENDE**

Natürlich hat so ein vorbildlich entwickeltes Programm auch einen ordnungsgemäßen Ausgang. Der Notausgang 'RUN/STOP-Taste' ist hier nicht angebracht. Wählen Sie diese Funktion aus, so wird nochmals gefragt, ob das Programm auch wirklich beendet werden soll. Bei 'N' erscheint das Menü und bei 'J' die Mitteilung, wie Sie das Programm starten können, ohne daß Ihre Daten verloren gehen.

Nun nutzen Sie den enormen Hauptspeicher des CBM 64 aus und füllen Sie ihn mit Ihrer persönlichen Datenbank.

#### **8.4 DUP**

```
FORMAT:    DUP ("STRING".N)
PARAMETER: STRING - STRING ODER STRINGVARIABLE
              N      - DUPLIKATIONSFAKTOR
FUNKTION:  VERVIELFACHEN EINER ZEICHENKETTE
BEISPIEL: PRINT DUP("-".40)
              FÜLLT EINE BILDSCHIRMZEILE MIT DEM
              MINUSZEICHEN
```

Sicher haben Sie diesen Befehl bereits in dem letzten Programm entdeckt. Mit DUP entfallen nun Befehle wie 'PRINT"-----"'. Aber nicht nur einzelne Zeichen können dupliziert werden. Auch mehrstellige

Zeichenketten können nun beliebig vervielfacht werden. Ein Beispiel:

```
10 A$=DUP ("SOS ",10)
20 PRINT A$
```

Die beiden Parameter sind voll variabel. Die folgende Befehlsfolge bestätigt es:

```
10 A$="SOS "
20 FOR I=1 TO 10
30 PRINT DUP (A$,I)
40 NEXT
```

Auch Steuerzeichen wie z.B. Cursorsteuerzeichen können dupliziert werden. Der folgende Programmausschnitt verwendet dem ASCII-Wert von 'CUROR NACH OBEN' (CHR\$(145)):

```
10 PRINT"ZEILE 10"
20 PRINT DUP(CHR$(145),10)
30 PRINT"ZEILE 30"
```

Der Aufbau einer Schleife zum Positionieren des Cursors um mehrere Stellen entfällt also.

Selbstverständlich können Sie mit diesem Befehl auch Ihre Überschriftenzeile auf dem Druckerpapier unterstreichen:

```
10 OPEN1,4
20 PRINT#1.DUP("-",80)
30 CLOSE 1
```

DUP ist letztendlich ein Befehl, auf den Sie schon lange gewartet haben.

## 8.5 TESTAUFGABEN

1) Welches Ergebnis liefern die folgenden Zeilen?

```
10 A$="AAA BBB CCC"  
20 PRINT INSERT (" XXX",A$,4)
```

- A) AAAXXX BBB CCC
  - B) AAA XXX BBB CCC
  - C) AAA XXXBBB CCC
- 

2) Was muß beim Einsatz des Befehls INSERT beachtet werden?

- A) Der einzusetzende String muß kleiner sein als der äußere String
  - B) Der einzusetzende String muß größer sein als der äußere String
  - C) Der resultierende String darf 255 Zeichen nicht überschreiten
- 

3) Welcher der folgenden INSERT-Befehle liefert die folgende Zeichenfolge?

KLAGENFURT

- A) 10 PRINT INSERT("LAGENF", "KURT", 1)
  - B) 10 PRINT INSERT("LAGENF", "KURT", 0)
  - C) 10 PRINT INSERT("LAGENF", "KURT", 2)
  - D) keiner der Befehle
- 

4) Welches Ergebnis resultiert aus den folgenden Zeilen?

```
10 A$="AAABBBCCDDDD"  
20 PRINT INST (" ",A$,4)
```

- A) AAA BBCCDDDD
  - B) AAAB BBCCDDDD
  - C) AAAB BCCDDDD
  - D) AAA BBCCDDDD
-

5) Welche Position wird mit folgender Befehlsfolge angezeigt?

```
10 A$="AA BB CC"
```

```
20 PRINT PLACE (" ",A$)
```

- A) 1
  - B) 2
  - C) 3
- 

6) Welcher der folgenden Befehle gibt nicht den Wert 12 aus?

```
A) 10 PRINT PLACE(" ", "AAA.BBB.CCC DDD.EEE")
```

```
B) 10 PRINT PLACE("XY", "YX X Y XX YXY XX")
```

```
C) 10 PRINT PLACE("100", "1000100010100100")
```

---

7) Welcher der folgenden Befehle zeigt nicht diese Zeichenfolge : "-----" ?

```
A) PRINT DUP("-", 12)
```

```
B) PRINT DUP "--", 4)
```

```
C) PRINT DUP "----", 3)
```

---

8) Kann der Befehl DUP auch Steuerzeichen (z.B. Cursor Home) duplizieren?

A) ja

B) nein

---

9) Wie lang dürfen mit DUP produzierte Zeichenketten höchstens sein?

A) 256 Zeichen

B) 1000 Zeichen

C) 255 Zeichen

D) 1024 Zeichen

---

## **9. KAPITEL**

### **AUSGABEKONTROLLE**

Simon's Basic besitzt die Möglichkeit, sehr komfortabel die Ausgabe auf Ihrem Bildschirm zu gestalten. Mit den folgenden Befehlen werden Ihnen Mittel in die Hand gegeben, die die äußere Gestalt Ihres zukünftigen Programmes bis zum professionellen verfeinern.

So können Sie Ihre Ausgabe formatieren, blinken lassen oder sehr einfach ganze Bildschirmbereiche behandeln, was Ihrem Programm maximalen Komfort und Geschwindigkeit verleiht.

Zunächst werden die Formatierungs-, dann die Blink-, "Füll-" und schließlich die Scroll-Befehle besprochen. Probieren Sie wie immer sofort alles aus, um die neuen Befehle möglichst schnell kennenzulernen, zu verstehen und schließlich handhaben zu können.

#### **9.1 FORMATIERTE AUSGABE**

Unter formatierter Ausgabe versteht man die geordnete Positionierung Ihrer Kommentare, Rechnungen etc., die Sie auf dem Bildschirm sichtbar machen möchten, nach bestimmten Kriterien. Das originale CBM-Basic bietet schon einige Möglichkeiten, Bildschirminhalte gegliedert darzustellen. Alle Befehle, die zu diesem Zweck in das CBM-Basic aufgenommen wurden, können in Zusammenhang mit dem PRINT-Befehl verwendet werden (außer POS; s. Handbuch). Da der Programmierer vielfach nichts Rechtes mit diesen Kommandos anzufangen weiß, sollen Ihnen ihre Funktion und die sich daraus ergebenden Möglichkeiten hier kurz dargestellt werden. In diesem Zusammenhang muß man zunächst einen wichtigen Begriff erläutern:

#### **DER CURSOR:**

Der Cursor zeigt die Position auf dem Bildschirm an, an die das nächste Zeichen gesetzt wird. Er existiert in zwei Zuständen: Im Eingabe-Modus stellt er das gut sichtbare kleine, blinkende Rechteck auf Ihrem Schirm dar. Im Programmmodus ist er nicht sichtbar und nur "virtuell" vorhanden, d.h. er stellt lediglich die aktuelle

Bildschirmposition dar. Wenn wir also von Cursorposition sprechen, so ist stets die Position des nächsten zu setzenden Zeichens gemeint.

**- SPC**

Diese Funktion legt einen Zwischenraum zwischen der momentanen Cursorposition und dem anschließend zu schreibendem Text fest. Sie dient also der relativen Positionierung der Bildschirmausgabe.

**- TAB**

TAB spezifiziert die Spalte einer Zeile, von der ab die nächste Ausgabe stattfinden soll. Hier wird also eine absolute Positionierung vorgenommen (s. auch AT).

**- POS**

Um die momentane Cursorposition innerhalb einer Zeile zu erfahren, verwenden Sie diesen Befehl. Er ermöglicht Ihnen eine Orientierung auf dem Bildschirm und ist sehr effektiv in Zusammenhang mit dem LIN-Befehl des Simon's Basic.

**- ./;**

Wie Ihnen vielleicht bekannt ist, wird nach jedem PRINT-Statement ein sogenanntes carriage-return ausgeführt, d.h. es wird an den Anfang der nächsten Zeile gesprungen. Dies kann durch Hintenanfügen eines "," (Komma) oder ";" (Semikolon) unterbunden werden. Während das Semikolon rein unterdrückende Funktion hat, um an eine PRINT-Ausgabe direkt ohne Zwischenraum eine weitere anzuhängen, wird die Ausgabe bei Verwendung des Kommas zusätzlich automatisch tabellarisch vorgenommen (s. Handbuch).

Nun aber zu den zusätzlichen Simon's Basic-Befehlen:

### 9.1.1 CENTRE

```
FORMAT      :CENTRE STR
PARAMETER   : - STR: "ZEICHENKETTE" ODER
              STRINGVARIABLE MIT EINER
              LANGE < 40 !
BEISPIEL    :CENTRE AS
ODER        CENTRE "SIMON'S BASIC"
FUNKTION     :ZENTRIERTE AUSGABE EINES STRINGS
```

#### ERLÄUTERUNGEN:

Mit dem Befehl CENTRE haben Sie die Möglichkeit, Ihre Ausgabe auf dem Bildschirm zentriert zu gestalten. Unter zentrierter Ausgabe versteht man Folgendes:

Angenommen, Sie wollen die Überschrift 'Commodore 64' gestalten. Diese Überschrift soll nun aber in der Mitte der ersten Zeile erscheinen.

Normalerweise müßte man erst umständlich die Anzahl der Buchstaben der Überschrift zählen, von der Anzahl der Zeichen pro Zeile (40) abziehen und das Ergebnis durch 2 teilen, um die Anzahl der Leerzeichen vor der Schrift zu berechnen.

Diese Umstände können Sie sich sparen. Der Befehl CENTRE setzt automatisch die notwendigen Leerstellen (nicht Leerzeichen!), um den Schriftzug in die Mitte zu bringen.

Der Ausdruck darf nicht länger als 39 Zeichen sein, andernfalls erscheint die Ausgabe recht willkürlich auf dem Bildschirm. Der Cursor steht natürlich auf der Position direkt hinter der Schrift, d.h. wenn Sie als nächstes in die folgende Zeile schreiben wollen, so müssen Sie erst ein carriage return in Form eines PRINT oder PRINT CHR\$(13); senden.

Sie können dies auch für Ihre Zwecke ausnutzen, indem Sie z.B. mit CENTRE "" nur den Cursor positionieren, ohne etwas zu schreiben, um später von dieser Position aus Zeichen zu setzen.

Fügen Sie direkt hinter einen CENTRE-Befehl einen zweiten, so wird die Anzahl der Leerstellen vor der ersten Ausgabe wieder aufgegriffen und in dem gleichen Abstand vor die zweite etc. gesetzt. Sie haben also bei geschickter Ausnutzung dieses wahrscheinlich unbeabsichtigten Phänomens eine Möglichkeit,

verschiedene Ausgabeteile mit konstantem Abstand auszugeben.

**BEISPIELE:**

```
100 REM #####
110 REM ##                               ##
120 REM ##  CENTRE-BEISPIEL  ##
130 REM ##                               ##
140 REM #####
150 REM
160 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
170 A$ = "SIMON'S BASIC"
180 B$ = "FUER CBM 64"
190 CENTRE A$ : PRINT
200 CENTRE B$ : PRINT
210 CENTRE "*****"
220 REM
230 REM #####
240 REM ##  ERGAENZUNG  ##
250 REM #####
260 REM
270 PRINT
280 FOR X = 17 TO 1 STEP -2
290 CENTRE DUP ("*",X) : PRINT
300 NEXT X
```

In dem ersten Teil dieses Beispiels wird lediglich eine Überschrift zentriert ausgegeben. Der zweite Teil, der einfach nur an das Programm angehängt wird, demonstriert Ihnen einen schönen Effekt.



## 9.1.2 USE

```
FORMAT      :USE STR1 . STR2
PARAMETER   : - STR1: "ZEICHENKETTE" ODER
                STRINGVARIABLE DER FORM
                "###.###" ODER
                "## TEXT . ## TEXT"
            - STR2: "ZEICHENKETTE" ODER
                STRINGVARIABLE MIT
                NUMERISCHEM WERT
BEISPIEL    :USE "###.##" ,"129.345"
FUNKTION     :DEZIMALPUNKTORIENTIERTE AUSGABE
                NUMERISCHER WERTE
```

### ERLÄUTERUNGEN:

Mit dem Befehl USE wird Ihnen die Möglichkeit einer tabellarischen Ausgabe von Dezimalzahlen in die Hände gelegt. Das heißt, Sie können zum Beispiel Abrechnungen schön Dezimalpunkt (-komma) unter Dezimalpunkt auf Ihrem Bildschirm darstellen, ohne erst umständlich festzustellen, wo im Einzelnen die jeweiligen Dezimalstellen eingeordnet werden müssen.

Das dabei zu verwendende Format geben Sie mit str2 an. Jedes Ziffernkreuz (#) stellt dabei eine Dezimalstelle der späteren Ausgabezahl dar. Bis zum Punkt in str2 wird bei der Ausgabe für jedes Ziffernkreuz eine Leerstelle freigehalten, falls diese nicht durch eine Ziffer besetzt wird. Entsprechendes gilt für die Stellen nach dem Komma (Punkt). Ist eine Dezimalzahl länger als die angegebene Zahl an Ziffernkreuzen, so wird sie an der letzten Stelle einfach abgeschnitten. Bitte beachten Sie dies, und runden Sie im Bedarfsfall vorher die entsprechende Zahl.

Sie können selbstverständlich an den oben unter "Parameter" angegebenen Stellen in der Formatanweisung entsprechenden Text einfügen (unter Berücksichtigung der Rolle des Punktes (.) und der Ziffernkreuze (#) für das Format).

Diese Formatanweisung kann natürlich als einfache "Zeichenkette" oder per vorher präparierter Stringvariable in den Befehl eingefügt werden.

str2 enthält die zu formatierende Dezimalzahl, die vorher durch den Befehl STR\$( ) in einen String umgewandelt wurde, oder einfach als "Zeichenkette" angegeben wird.

Probleme gibt es bei gleichzeitiger Verwendung von Text und Formatierung in einer Zeile:

In einem solchen Falle entstehen aufgrund einer Programmierfehler in der Formatierung, falls die letzte(n) Dezimalstelle(n) hinter dem Komma (Punkt) gleich Null wird. Tauchen hier also Unstimmigkeiten auf, so ist das meist nicht Ihre Schuld. Schreiben Sie in einem solchen Falle den Text zunächst mit PRINT und fügen dann die Formatierung mit USE hintenan.

### BEISPIEL:

```
100 REM #####
110 REM ##                ##
120 REM ##  USE-BEISPIEL  ##
130 REM ##                ##
140 REM #####
150 REM
160 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
170 A$ = "##.##"
180 A = 12.34
190 B = 230.12
200 USE "BUEROBEDARF: " + A$.STR$(A) : PRINT " DM"
210 USE "BUECHER      : " + A$.STR$(B) : PRINT " DM"
220 PRINT "-----"
230 USE "SUMME        : " + A$.STR$(A+B) : PRINT " DM"
```

Dieses Beispiel zeigt Ihnen eine interessante Anwendung des eben beschriebenen Befehls. Ich bin sicher, Ihnen fallen noch weit mehr ein.

### 9.1.3 AT

```
FORMAT      :PRINT STR1 AT(S,Z) STR2
PARAMETER   : - STR1/ : ZWEI AUSZU-
              STR2   GEBENDE "ZEICHENKETTEN" ODER
                   STRINGVARIABLEN. WOBEI STR1
                   OPTIONAL IST. D.H. AUCH
                   WEGGELASSEN WERDEN KANN.
              - S     : SPALTE
              - Z     : ZEILE (S.U.)
BEISPIEL    :PRINT AT(10,12) "CBM 64"
FUNKTION    :POSITIONIERTE AUSGABE EINES
              STRINGS
```

#### ERLÄUTERUNGEN:

Mit AT in einem PRINT-Statement haben Sie die Möglichkeit, einen unter str2 angegebenen String ("Zeichenkette" oder Stringvariable) beliebig auf dem Bildschirm zu positionieren. Dabei geben die Parameter s und z jeweils die Spalte und die Zeile des Ausgabeanfangs an. D.h. s und z bestimmen die Cursorposition vor der Ausgabe (Cursorkoordinaten). Es gilt dabei wieder, daß weitere Bildschirmausgaben direkt an das letzte Zeichen der positionierten Ausgabe angefügt werden. Auch hier ließe sich mit str2 = "" (Leerstring) eine einfache Cursorpositionierung ohne Bildschirmausgabe erreichen. Wenn Sie wollen, können Sie mit str1 natürlich Text noch ab der alten Position schreiben.

Eine besonders interessante Anwendung des AT-Befehls ist die Erstellung von Low-Resolution-Graphiken (LGR = niedrig auflösende Graphik). Dabei sind als "Punkte" natürlich alle Zeichen des Commodore 64 zugelassen (s. Beispiele).

Man sieht, der AT-Befehl ist ein äußerst nützliches und vielseitiges Kommando, dessen Reichweite erst beim Probieren langsam klar wird (und probieren sollten Sie ständig; nur so lernen Sie die Vielzahl der Befehle gut kennen!)

Noch eine Anmerkung zur Syntax: Die erste Klammer nach AT gehört mit zum Befehl. Sie dürfen also kein Leerzeichen zwischen AT und ( setzen!

**BEISPIELE:**

```
100 REM #####
110 REM ##           ##
120 REM ## AT-BEISPIEL-1 ##
130 REM ##           ##
140 REM #####
150 REM
160 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
170 PRINT AT(10.5) "DIE ´AT´-"
180 PRINT AT(13.10) "FUNKTION"
190 PRINT AT(16.15) "IN AKTION"
```



```
100 REM #####
110 REM ##           ##
120 REM ## AT-BEISPIEL-2 ##
130 REM ##           ##
140 REM #####
150 REM
160 PRINT CHR$(247) : REM BILDSCHIRM LOESCHEN
170 FOR X=0 TO 24
180 PRINT AT(X,X) "SIMON´S BASIC";
190 NEXT X
200 PRINT AT(0,20) ""
```



```
100 REM #####
110 REM ##           ##
120 REM ## AT-BEISPIEL-3 ##
130 REM ##           ##
140 REM #####
150 REM
160 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
170 FOR X=0 TO 39
180 PRINT AT(X , 12 * SIN(X/6) + 12) "***"
190 NEXT X
```

Diese Beispiele demonstrieren Ihnen einige Anwendungsmöglichkeiten dieses Befehls von der einfachen positionierten und damit evt. formatierten Ausgabe von Texten bis hin zur Graphikverwendung. Da durch den AT-Befehl keinerlei Zeichen auf den Bildschirm gebracht werden, die den Cursor an die von Ihnen bestimmte Stelle setzen (z.B. Leerzeichen o.ä.), hat der Befehl natürlich in Zusammenhang mit dem Betrieb eines Druckers nur in dem Fall eine Funktion, wenn der Befehl HRDCPY (Bildschirmhardcopy; s. # 11.2.2) verwendet wird. Beim direkten Ausdruck nach Eingabe von CMD oder PRINT# wird er ignoriert und str2 einfach an die alte Druckposition angehängt.

Probieren Sie alles aus und verändern Sie ruhig das Eine oder Andere, Sie werden Ihre helle Freude haben!

### 9.1.4 LIN

<b>FORMAT</b>	<b>:LIN (ALS FUNKTION)</b>
<b>PARAMETER</b>	<b>: ---</b>
<b>BEISPIEL</b>	<b>:A = LIN</b>
<b>FUNKTION</b>	<b>:BESTIMMEN DER CURSORZEILEN- POSITION</b>

#### ERLÄUTERUNGEN:

LIN ist einer der Befehle, die nicht eigenständig stehen, sondern wie z.B. SIN oder SQR eine Funktion darstellen, mit der weiter gerechnet wird, oder die direkt in einen Speicher (etwa der Form: A = LIN (s.u.)) gegeben werden kann.

LIN gibt ihnen ein Instrumentarium in die Hand, um auf einfachste Weise die Bildschirmzeile zu bestimmen, in der sich der Cursor befindet.

Wie Sie wissen, haben Sie mit dem originalen Commodore-Basic die Möglichkeit, die Cursorposition innerhalb einer Zeile mittels POS (wie oben erläutert) zu erfahren. Die jeweilige Zeile jedoch bleibt unbekannt. Dieser Mangel ist hiermit behoben. Mit POS und LIN können Sie nun die exakte Cursor'koordinate' erkennen und für Ihre Rechnungen oder Ausgabe verwenden.

Beachten Sie die unterschiedliche Syntax dieser beiden Befehle!

#### BEISPIEL:

```
100 REM #####
110 REM ##          ##
120 REM ##  LIN-BEISPIEL  ##
130 REM ##          ##
140 REM #####
150 REM
160 PRINT CHR$(147); : REM BILDSCHIRM LOESCHEN
170 A = LIN
180 FOR X=1 TO 20
190 PRINT "DIES IST DIE" LIN CHR$(20) ". ZEILE"
200 NEXT X
210 PRINT
```

220 PRINT "NUN SIND DIE ZEILEN" A "BIS" LIN  
"BESCHRIEBEN";

Dieses Beispiel dient lediglich dazu, Ihnen die Wirkungsweise des LIN-Befehls zu demonstrieren. Anwendungen fallen Ihnen sicher im Laufe des Probierens genügend ein. Versuchen Sie es doch einmal!

## 9.1.5 PAUSE

```
FORMAT      : PAUSE STR . S
ODER        : PAUSE S
PARAMETER   : - STR: "ZEICHENKETTE"
              ODER STRINGVARIABLE, DIE
              EINEN KOMMENTAR ENTHALTEN
              - S : ANZAHL DER WARTESEKUNDEN
BEISPIEL    : PAUSE "VERZOEGERUNG".3
FUNKTION    : VERZOGERN DES PROGRAMMABLAUFS
              UM S SEKUNDEN
```

### ERLÄUTERUNGEN:

Was Sie sonst umständlich mit Warteschleifen erledigen müssen, können Sie nun in einen Befehl fassen: PAUSE  
PAUSE dient dazu, den Programmablauf um einen gewissen Zeitwert, den Sie mit s (Einheit: 1 Sekunde) angeben, zu verzögern. Gleichzeitig haben Sie optional (d.h. wahlweise) noch die Möglichkeit, diese Wartezeit mit str zu kommentieren.

Der Programmbenutzer kann die Warte'schleife' mit <RETURN> unterbrechen, um das Programm fortzuführen. Sollte Ihr Programm gerade "in diesem Befehl stecken", so ist es nicht durch <RUN/STOP> zu unterbrechen. Verwenden Sie hierzu entweder <RUN/STOP><RESTORE> oder <RETURN> und dann <RUN/STOP>.

Dieser Befehl kann natürlich sehr effektiv zur Fehlerkorrektur von Programmen eingesetzt werden, indem mittels des PAUSE-Statements z.B. Zwischenwerte angezeigt oder schnelle, unüberschaubare Vorgänge verlangsamt werden.

### BEISPIEL:

```
100 REM #####
110 REM ##           ##
120 REM ## PAUSE-BEISPIEL ##
130 REM ##           ##
140 REM #####
150 REM
```



```
160 PRINT CHR$(5) : REM WEISSE SCHRIFT
170 FOR X=0 TO 15
180 POKE 53248 + 33 , X : REM HINTERGRUNDFARBE
190 PRINT CHR$(147) "<RETURN> FUER WEITER!"
200 PRINT
210 PAUSE "SIE SEHEN FARBNUMMER" + STR$(X) . 4
220 NEXT X
```

In dem oben angeführten Beispiel wird in einem 4-Sekundenabstand die Hintergrundfarbe 16 mal gewechselt. Sie sind jeweils in der Lage, die Pause, d.h. die Wartezeit mit <RETURN> zu unterbrechen.

Viel Spaß beim Programmieren!

### 9.1.6 TESTAUFGABEN

Und nun wie immer unser kleiner Test (Zutreffendes bitte ankreuzen; es können auch mehrere Antworten richtig sein):

1.) Was haben Sie bei dem Befehl CENTRE zu beachten?

- a) hinter CENTRE darf kein PRINT-Statement stehen
- b) zwei CENTRE-Befehle sollten durch ein PRINT getrennt sein
- c) der zu zentrierende String darf maximal 40 Zeichen lang sein

2.) Welche Syntax der einzelnen Befehle ist richtig?

- a) PRINT CENTRE "ABC"
- b) PRINT AT(2,4) "CBM 64"
- c) PRINT AT (2,4) "CBM 64"
- d) A = LIN \* 2
- e) PAUSE 5, "BITTE RETURN DRUECKEN"
- f) USE "### DM . ## PF", STR\$(123.34)

3.) Welche Befehle senden Sie, wenn Sie genau in die Mitte des Bildschirms zwei Sterne setzen wollen?

- a) PRINT "\*\*\*"
- b) CENTRE "\*\*\*"
- c) PRINT AT(19,12) "\*\*\*"
- d) A\$ = "\*\*\*" : PRINT AT(0,12) " " : CENTRE A\$

Die Lösungen finden Sie wie immer am Ende des Buches.

## 9.2 BILDSCHIRMBLINKEN

Der Entwickler von Simon's Basic hat auch an die Erzeugung schöner Effekte gedacht: Die blinkende Ausgabedarstellung. Sie kann zum Kenntlichmachen bestimmter Bildschirminhalte oder ganzer Sachverhalte oder einfach als "Spielzeug" verwendet werden. Die Anwendungen reichen von "ernsten" Programmen bis hin zu schönen Spielen und "Privatprogrammen".

### 9.2.1 FLASH

```
FORMAT      :FLASH F.S
PARAMETER   : - F: FARBE, DIE BLINKT
              MIT F VON 0 BIS 15
              - S: BLINKGESCHWINDIGKEIT
              MIT S VON 0 BIS 255
BEISPIEL    :FLASH 4.10
FUNKTION     :BLINKEN EINER BILDSCHIRMFARBE
```

#### ERLÄUTERUNGEN:

Mit FLASH können Sie alle Zeichen, die in einer bestimmten Farbe f auf dem Bildschirm stehen, kontinuierlich mit der Geschwindigkeit s (speed) blinken lassen. D.h. alle Zeichen, die diese Farbe besitzen, wechseln ständig von Normal nach Invers (Invers bedeutet, das Zeichen sieht so aus, als hätten Sie es gesetzt, nachdem Sie die Tasten <ctrl> <RVS ON> gleichzeitig gedrückt hätten).

f bestimmt wie gesagt die Farbe der Zeichen, die vom inversen zum normalen Zustand wechseln (und umgekehrt). Dabei sind für f Werte zwischen 0 (= schwarz) und 15 (= grau3) zugelassen. Die einzelnen Werte sind wie folgt den verschiedenen Farben zugeordnet:

0 = SCHWARZ	8 = ORANGE
1 = WEIß	9 = BRAUN
2 = ROT	10 = HELLROT
3 = TÜRKIS	11 = GRAU 1
4 = VIOLETT	12 = GRAU 2
5 = GRÜN	13 = HELLGRÜN
6 = BLAU	14 = HELLBLAU
7 = GELB	15 = GRAU 3

Die Geschwindigkeit  $s$  wird dabei etwa (für 1 Sekunde ist  $s=58$ ) in der Einheit  $1/60$  Sekunde angegeben ( $s=0$  entspricht dabei  $s=256$ ) - mit einer kleinen Einschränkung:

Wird  $s$  sehr klein ( $<10$ ), wird also sehr schnell blinken gelassen, so hat Ihr Rechner stets sehr viel zu tun, denn er läßt schließlich den gesamten Bildschirm blinken. Deshalb wird Ihr Basic-Programm zunehmend langsamer. Gleichzeitig ändert sich aber auch die Einheit von  $s$  zu  $1/30$ ,  $1/15$  etc. Sekunden. Ihr Bildschirm wird also nie jede  $1/60$  Sekunde blinken können.

Gleichfalls treten Schwierigkeiten mit dem Fernseher oder Monitor auf, da Ihr Computer alle  $1/20$  Sekunde ein neues Bild herstellt (wie bei einer Filmkamera). Die auftretenden Effekte (Streifen und Unterbrechungen) können Sie sich einmal ansehen (s. auch BFLASH).

Noch eine wichtige Kleinigkeit ist beim Gebrauch von FLASH zu beachten:

Haben Sie bereits FLASH eingeschaltet und bringen nun zusätzlichen Text in der Blinkfarbe auf den Bildschirm, so kann es vorkommen, daß Teile oder Ihr ganzer Zusatz genau "entgegengesetzt" blinkt, sich also im inversen Zustand befindet, wenn der übrige Text normal ist und umgekehrt. Dies liegt daran, daß Ihr Rechner im Normalmodus schreibt (oder invers, wenn Sie RVS ON aktiviert haben), unabhängig vom gegenwärtigen Blinkzustand des Bildes. Die Zeichen aber werden einfach nur "umgedreht". Aus dem gleichen Grunde blinkt inverse Schrift nach dem Einschalten von FLASH genau in der entgegengesetzten Phase.

## 9.2.2 OFF

```
FORMAT      :OFF
PARAMETER   : ---
BEISPIEL    :OFF
FUNKTION    :AUSSCHALTEN DES FLASH-MODUS
```

### ERLAUTERUNGEN:

Mit OFF ist es Ihnen möglich, das mit FLASH eingeschaltete Blinken wieder rückgängig zu machen. Dabei wird jeweils der momentane Blinkzustand beibehalten, d.h. sind die in der blinkenden Farbe gezeichneten Zeichen im Moment invers dargestellt, so bleiben sie nach OFF gleichfalls invers und umgekehrt. Es empfiehlt sich also, wenn Sie genau bestimmen wollen, in welchem Zustand Ihr Textfenster nach dem Ausschalten durch OFF ist, das gesamte Fenster mit PRINT CHR\$(147) zu löschen und neu zu erstellen.

Grundsätzlich sollte nach jedem FLASH in Ihrem Programm irgendwann einmal ein OFF folgen, spätestens am Ende des Programms, da der Bildschirm sonst auch nach Beendigung Ihres Programmes weiterblinkt.

### BEISPIEL ZU FLASH UND OFF:

```
100 REM #####
110 REM ##                ##
120 REM ## FLASH-BEISPIEL ##
130 REM ##                ##
140 REM #####
150 REM
160 POKE 53248 + 33 ,0 : POKE 53248 + 32 ,0 : REM
HINTERGRUND- + RAHMENFARBE
170 PRINT CHR$(147) CHR$(158) : REM BILDSCHIRM LOESCHEN +
FARBE = GELB
180 CENTRE "BITTE WARTEN, BIS READY. ERSCHEINT" : PRINT
190 CENTRE "ODER <RETURN> GEDRUECKT HALTEN"
200 PRINT AT(0,8) "" : REM CURSOR SETZEN
210 CENTRE "#####" : PRINT
220 CENTRE "#####" : PRINT
230 CENTRE "##                ##" : PRINT
```

```

240 CENTRE "###" : PRINT
250 CENTRE "###" : PRINT
260 CENTRE "###" : PRINT
270 CENTRE "#####": PRINT
280 CENTRE "#####": PRINT
290 PRINT AT(0,10) CHR$(31) : REM BLAU
300 CENTRE " " : PRINT
310 CENTRE " COMMODORE 64 " : PRINT
320 CENTRE " SIMONS BASIC " : PRINT
330 CENTRE " " : PRINT
340 FOR X=10 TO 1 STEP -1
350 FLASH 6.X : REM FLASH BLAU
360 PAUSE 2
370 NEXT X
380 OFF

```

Dieses Beispiel soll Ihnen eine kleine Demonstration zur Verwendung und dem Zusammenspiel sowohl der Befehle FLASH und OFF als auch verschiedener anderer Ausgabebefehle geben. Nachdem der Bildschirm wie der Rahmen schwarz getüncht wurden, erscheint ein gelbes Ziffernkreuzrähmchen mit einem blauen Schriftzug auf der Mitte des Schirms. Alsdann fängt die blaue Schrift zu blinken an. Dieses Blinken wird mit der Zeit stetig schneller, bis es nach Einstellen der höchsten Blinkgeschwindigkeit stoppt. Recht amüsant, oder?

### 9.2.3 BFLASH

```
FORMAT      :BFLASH S.F11.F12
PARAMETER   : - S: BLINKGESCHWINDIGKEIT
              - F11/ : ZWEI FARBEN.
              F12  ZWISCHEN DENEN
                  JEWELTS GEWECHSELT WIRD.
BEISPIEL    :BFLASH 30.12.14
FUNKTION    :BLINKEN DES BILDSCHIRMRAHMENS
              ZWISCHEN DEN ZWEI ANGEGEBENEN
              RAHMENFARBEN
```

#### ERLÄUTERUNGEN:

BFLASH ermöglicht Ihnen, diverse Blitz- bzw. Blinkeffekte zu erzeugen, die dazu dienen, z.B. in Spielen besondere Situationen (Treffer, Sieg etc.) hervorzuheben oder allgemein die Aufmerksamkeit auf die bestehende Bildschirmanzeige zu lenken; kurz: dieser Befehl dient -wie auch FLASH- dazu, Ihr Programm auf einfache Weise anschaulicher zu gestalten.

Nun, mit BFLASH können Sie die Farbe des äußeren Bildschirmrahmens Ihres Fernsehgerätes (Monitor) ständig in einem bestimmten Zeitintervall zwischen zwei Farben hin und herschalten, ohne jedoch diese Schaltung selbst ständig vornehmen zu müssen (s. FLASH).

Wie auch bei FLASH können Sie die Geschwindigkeit des Blinkens eigens angeben. Hierzu dient der Parameter s. Seine Berechnung erfolgt analog zu FLASH, d.h. s wird in 1/60 Sekunde angegeben. s=0 steht ebenfalls für s=256. Um also jede Sekunde einmal die Farbe zu wechseln, geben Sie für s 60 ein (Anmerkung: dies ist nur ein Richtwert, in Wirklichkeit wird schon bei s=58 jede Sekunde gewechselt.).

Da ein Farbwechsel des Bildschirmrahmens sehr schnell durch einen POKE, d.h. durch das Ändern einer Speicherstelle vonstatten geht, ist es möglich, auch ohne großen Zeitverlust einen raschen ( $0 < s < 10$ ) Farbwechsel vorzunehmen. D.h. anders als bei FLASH können Sie tatsächlich alle 1/60 Sekunde die Farbe wechseln. Da das Bildschirmbild jedoch nur etwa alle 1/20 Sekunde erneuert wird, gibt es bei sehr raschen Wechseln besonders interessante Streifeneffekte. Probieren Sie es einmal aus!

Die zwei weiteren anzugebenden Variablen bestimmen die

jeweiligen Farben, zwischen denen hin und her gewechselt wird und können Werte von 0 (=schwarz) bis 15 (=grau3) annehmen (s. Farbtabelle bei FLASH). Hierbei ist Folgendes zu beachten:

Wird zwischen zwei Farben mit starkem Kontrastunterschied (Extrem: schwarz-weiß) gewechselt, so hat das ebenfalls Auswirkungen auf das übrige Fernsehbild: Die helle Farbe nimmt dem inneren Bildschirmfenster die Farbtintensität bzw. die Leuchtkraft, während die dunkle Farbe dafür sorgt, daß der innere Teil besonders herauskommt. Damit ist ein gleichzeitiges, ungewolltes "Mitblinken" des Textfensters zu beobachten (besonders bei einer leuchtarmen Hintergrundfarbe (Extrem: schwarz)). Dieser Effekt ist sehr gut bei Beispiel 1 unter BFLASH 0 zu erkennen.



## 9.2.4 BFLASH 0

<b>FORMAT</b>	<b>:BFLASH 0</b>
<b>PARAMETER</b>	<b>: ---</b>
<b>BEISPIEL</b>	<b>:BFLASH 0</b>
<b>FUNKTION</b>	<b>:AUSSCHALTEN DES BFLASH-MODUS</b>

### ERLÄUTERUNGEN:

BFLASH 0 besitzt in etwa die gleiche Funktion, wie OFF lediglich auf BFLASH bezogen. D.h. BFLASH 0 schaltet das Blinken des Bildschirmrahmens aus. Dabei verbleibt der Rahmen relativ unkontrollierbar in der zuletzt angezeigten Farbe. Es empfiehlt sich also, den Rahmen grundsätzlich neu mit POKE 53248 + 32,f (f bedeutet hier die gewünschte Farbnummer) zu definieren, um sicherzugehen, welche Rahmenfarbe man jetzt tatsächlich anzeigt.

BFLASH 0 sollte grundsätzlich irgendwann im Laufe Ihres Programms, spätestens aber am Ende gesetzt werden, sofern man von BFLASH Gebrauch gemacht hat, da sonst auch noch nach Beendigung des Programms BFLASH aktiv ist.

### BEISPIELE ZU BFLASH UND BFLASH 0:

#### BEISPIEL 1 :

ERSETZEN SIE IN DEM FLASH-BEISPIEL DIE ZEILEN 350 UND 380 DURCH DIE FOLGENDEN:

```
350 BFLASH X.6.7 : REM BLAU-GELB WECHSEL
380 BFLASH 0
```

-----

#### BEISPIEL 2 :

```
100 REM #####
110 REM ##                               ##
120 REM ## BFLASH - BEISPIEL 2 ##
130 REM ##                               ##
140 REM #####
150 REM
160 POKE 53248 + 33.2 : REM HINTERGRUND ROT
```

```

170 BFLASH 1.7.6 : REM BLINKFARBEN = GELB - BLAU
180 PRINT CHR$(5) CHR$(147) : REM SCHRIFT WEISS: BILDSCHIRM
LOESCHEN
190 FOR X=0 TO 39
200 Y = X/2
210 PRINT AT( X , Y ) "*" : REM DIAGONALE 1
220 PRINT AT( 39-X , Y ) "*" : REM DIAGONALE 2
230 PRINT AT( X , 0 ) "*" : REM HORIZONTALE OBEN
240 PRINT AT( X , 20 ) "*" : REM HORIZONTALE UNTEN
250 PRINT AT( 0 , Y ) "*" : REM SENKRECHTE LINKS
260 PRINT AT( 39 , Y ) "*" : REM SENKRECHTE RECHTS
270 NEXT X : REM NAECHSTER PUNKT
280 FLASH 1.15
290 PRINT AT(3.0) CHR$(150) : REM POSITIONIEREN UND TEXT:
HELLROT
300 CENTRE "SIE STAUNEN?" : PRINT
310 CENTRE "KLEINIGKEIT!"
320 PRINT AT(0.20) "" : REM READY. POSITIONIEREN
330 PAUSE 6 : REM 6 SEKUNDEN WARTEN
340 BFLASH 0
350 OFF : REM BLINKEN AUS

```

Dieses wohl recht eindrucksvolle Beispiel demonstriert das mögliche Zusammenspiel zwischen FLASH, BFLASH, OFF, BFLASH 0 und einiger weiterer Befehle des Simon's Basic. Wenn Sie wollen, können Sie einmal versuchen, nachzuvollziehen, was in den einzelnen Teilen des Programms geschieht, wobei die Zeilen 190 - 270 nicht unbedingt für das Verständnis der verschiedenen Befehle notwendig sind.

Trotzdem lassen sich einige Veränderungen leicht vornehmen, um mit dem Programm zu "spielen". Probieren Sie ruhig einmal!

### 9.2.5 TESTAUFGABEN

Zum Schluß dieses Kapitels wieder unsere altbeliebten Testaufgaben:

1.) Welchen Befehl tippen Sie ein, wenn Sie den Rahmen mit den Farben weiß und blau mit einer Geschwindigkeit von etwa einem Wechsel pro Sekunde blinken lassen wollen?

- a) BFLASH 0
- b) FLASH 0,1
- c) BFLASH 60,1,0
- d) BFLASH 0,1,1

2.) Sie wollen das Blinken der Zeichenfarbe Rot beenden, das Sie vorweg eingestellt hatten. Welche Befehle wählen Sie?

- a) BFLASH 0,0,0
- b) BFLASH 0
- c) OFF
- d) FLASH 0

3.) Was machen Sie, wenn Sie nur einen Teil des Textes mittels des FLASH-Befehls blinken lassen wollen?

- a) Ich schreibe zunächst den Text, der blinken soll, schalte das Blinken an, und anschließend setzte ich den restlichen nicht blinkenden Text.
- b) Ich gehe in der genau umgekehrten Reihenfolge wie unter a) beschrieben vor.
- c) Ich schreibe den Blinktext in einer anderen Farbe als den nicht blinkenden Text und lasse die Farbe des ersten Textes blinken.
- d) Ich schreibe beide Texte in der gleichen Farbe, lasse aber mit FLASH 12,40 nur die obere Hälfte des Bildschirms, in der der erste Text steht, blinken.

### 9.3 HINTERGRUND- UND RAHMENFARBE

Nun passen Sie auf! Wir sind im Begriff, eine sensationelle Entdeckung zu machen: Simon's Basic besitzt nicht nur die Befehle des originalen Handbuchs, nein, nimmt man sich diese Basicerweiterung genauer unter die Lupe, so findet man einige weitere Befehle, die überhaupt nicht im Handbuch vermerkt sind (s. auch Anhang). Insgesamt gibt es 7 weitere, zumindest bei der Grundkonzeption geplante Kommandos. Von diesen 7 Befehlen funktionieren jedoch nur 3. Die anderen 4 konnten wohl (wahrscheinlich aus Zeitnot - die Erstellung solcher Programme ist ja oft Terminalsache) nicht früh genug lauffähig gemacht werden. In wie weit diese Befehlsworte jedoch in der angekündigten Modulversion Berücksichtigung finden, ist natürlich noch unbekannt. Fest steht, daß Sie 3 weitere Befehle zu Ihrer freien Verfügung haben, die sich mit der Ausgabe auf dem Bildschirm befassen. Bevor wir uns jedoch damit beschäftigen, muß zunächst Einiges erläutert werden:

Der extended Colour - Modus:

Ihr Commodore 64 besitzt neben dem bekannten normalen Textmodus mit einer Hintergrundfarbe für alle Zeichen und 15 Zeichenfarben einen weiteren Textmodus, in dem Sie für jedes Zeichen eine andere Hintergrundfarbe (jeweils 4 Hintergrundfarbregister, also 4 frei wählbare Hintergrundfarben stehen Ihnen zur Verfügung) wählen können. Diese Darstellungsart heißt: extended Colour - Modus.

Wie gesagt, stehen Ihnen hier für jedes Zeichen eine von 4 Hintergrundfarben zur Verfügung, die Sie mit den entsprechenden Befehlen oder direkt durch EinPOKEN der jeweiligen Werte in die folgenden Register verändern können:

**HINTERGRUNDFARBE 0 ..... POKE 53248 + 33.HF0**

**HINTERGRUNDFARBE 1 ..... POKE 53248 + 34.HF1**

**HINTERGRUNDFARBE 2 ..... POKE 53248 + 35.HF2**

**HINTERGRUNDFARBE 3 ..... POKE 53248 + 36.HF3**

Wie Sie sehen, entspricht das Farbregister 0 dem normalen Register zur Festlegung der Hintergrundfarbe. hf ist der jeweilige Farbcode, den Sie aus dem Anhang entnehmen

können. Wie legen Sie nun fest, welche dieser Hintergrundfarben die jeweiligen Zeichen schließlich besitzen?

Die oberen 2 Bits jedes Bytes aus dem Videoram, also dem Bildschirmspeicher, der eigentlich nur die Bildschirmcodes für die einzelnen Zeichen in dem Textfenster speichert (s.o.), legen dies jeweils für jedes Zeichen fest. Da aber diese beiden Bits normalerweise ebenfalls dazu verwendet werden, um verschiedene Zeichen zu codieren, stehen Ihnen im extended Colour - Modus lediglich 64 Zeichen zur Verfügung.

Alle Graphikzeichen und im Kleinschrift/Großschrift - Modus ebenfalls die Großbuchstaben, sowie alle inversen Zeichen fallen dem zum Opfer. Steuern Sie diese trotzdem an, so wird eines der erlaubten Zeichen mit einer anderen Hintergrundfarbe erscheinen. Welche Zeichen davon wie betroffen sind, können Sie am besten der Tabelle der Bildschirmcodes (CBM 64-Benutzerhandbuch Seiten 133 - 134) entnehmen. Dabei gilt die folgende Zuordnung:

<b>MSBS</b>	<b>.....</b>	<b>BILDSCHIRMCODES</b>	<b>.....</b>	<b>HINTERGRUNDFARBE</b>
<b>00</b>	<b>.....</b>	<b>000 - 063</b>	<b>.....</b>	<b>HF 0</b>
<b>01</b>	<b>.....</b>	<b>064 - 127</b>	<b>.....</b>	<b>HF 1</b>
<b>10</b>	<b>.....</b>	<b>128 - 191</b>	<b>.....</b>	<b>HF 2</b>
<b>11</b>	<b>.....</b>	<b>192 - 255</b>	<b>.....</b>	<b>HF 3</b>

Unter MSBs verstehen wir hier die beiden obersten Bits des Bildschirmspeichers (Bits 6 und 7). Die Bildschirmcodes sind jeweils dezimale Werte. Vergleichen Sie diese Tabelle mit der angegebenen Bildschirmcodetabelle im Handbuch!

Wollen Sie also ein B mit der Hintergrundfarbe 7 = gelb auf den Bildschirm bringen, so belegen Sie das gewünschte Hintergrundfarbregister (hier 1) mit dem Wert 7 für gelb und POKEn eine 2 für B zuzüglich 64 für die Ansteuerung des Registers 1 an die entsprechende Stelle im Bildschirmspeicher oder geben mittels PRINT-Statement das Zeichen <shift><B>, also CHR\$(98) auf dem Bildschirm aus. Im Programm sähe dies dann so aus:

**10 BCKGNDS 2.3.4.5 : REM EXTENDED COLOUR - MODUS EIN (S.U.)**

**20 POKE 53248 + 34.7 : REM HINTERGRUNDFARBE 1 = GELB**

**30 POKE 1024.2 + 64 : REM ZEICHEN OBEN LINKS IN DIE ECKE**

**ODER:**

**10 BCKGNDS 2.3.4.5 : REM EXTENDED COLOUR - MODUS EIN (S.U.)**

**20 POKE 53248 + 34.7 : REM HINTERGRUNDFARBE 1 = GELB**

**30 PRINT CHR\$(98) : REM ZEICHEN AN DIE CURSORPOSITION**

Nach Ablauf dieser Programme befinden Sie sich weiterhin in diesem Modus und können so ein wenig herumprobieren. Doch nun zu den Befehlen, die Sie sicher schon ungeduldig erwartet haben.

### 9.3.1 COLOUR

<b>FORMAT</b>	: COLOUR RF.HF
<b>PARAMETER</b>	: - RF: RAHMENFARBE (0-15) - HF: HINTERGRUNDFARBE (0-15)
<b>BEISPIEL</b>	: COLOUR 6 7
<b>FUNKTION</b>	: BESTIMMEN DER RAHMEN- UND DER HINTERGRUNDFARBE

#### **ERLAUTERUNGEN:**

Der erste der drei neuen Befehle ermöglicht Ihnen das Ändern der Bildschirm - Rahmen- und - Hintergrundfarbe auf die einfachste Weise, ohne dafür umständliche POKEs einzugeben, deren Funktion nach einiger Zeit (außer, wenn durch REM erklärt) in einem alten Programm recht unüberschaubar wird.

Mit rf, dem ersten Parameter, der die nähere Funktion von COLOUR definiert, bestimmen Sie die Farbe des äußeren Rahmens Ihres Bildschirms. Dabei geben Sie für rf den Farbcode ein, mit dessen zugeordneter Farbe Sie den Rahmen belegen wollen. Dieser Wert wird dann automatisch in das entsprechende Register eingespeichert. Bei der Festlegung des Bildschirmrahmens sollten Sie beachten, daß sehr helle Farben ähnlich dem BFLASH-Befehl dem Fensterinneren die Leuchtkraft nehmen, was bis zur Unlesbarkeit eines Textes führen kann. Das Beispiel stellt dies anschaulich dar. Dieser Teil der Funktion des COLOUR-Befehls entspricht also dem Befehl:

#### **POKE 53248 + 32.RF**

hf bedient, wie Sie sich sicher schon denken können, die Hintergrundfarbe des Textes (Hintergrundfarbe 0). Damit ist natürlich lediglich die Hintergrundfarbe im Textmodus gemeint. Für die Graphik gelten andere Befehle (in Multicolor (s.u.) allerdings verändert er ebenfalls die Hintergrundfarbe und ist dort auch die einzige Möglichkeit diese (außer durch POKEn) zu verändern!), wogegen die Rahmenfarbe ebenfalls im Graphikmodus verändert werden kann. Wenn Sie die Hintergrundfarbe festlegen, so achten Sie darauf, daß Sie Farben wählen, die einen möglichst großen Kontrast zu der Textfarbe besitzen, da es ansonsten zu grauenhaften Effekten kommt (Beispiel: rot auf blau). Dieser Teil des Befehls ersetzt also den Befehl:

#### **POKE 53248 + 33.HF**

Insgesamt könnte man COLOUR durch  
POKE 53248 + 32.RF : POKE 53248 + 33.HF  
ersetzen. Der Nutzen dieses Kommandos dürfte damit klar sein.

**BEISPIEL :**

```
100 REM #####
110 REM ##                ##
120 REM ## COLOUR-BEISPIEL ##
130 REM ##                ##
140 REM #####
150 REM
160 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
170 PRINT CHR$(31) : REM TEXTFARBE = BLAU
180 PRINT AT(0,8) "" : REM CURSOR POSITIONIEREN
190 CENTRE "#####" : PRINT
200 CENTRE "##                ##" : PRINT
210 CENTRE "## WIR AENDERN DIE ##" : PRINT
220 CENTRE "## RAHMENFARBE ##" : PRINT
230 CENTRE "##                ##" : PRINT
240 CENTRE "#####" : PRINT
250 REM
260 FOR RF=0 TO 15
270 COLOUR RF,0 : REM HINTERGRUNDFARBE = SCHWARZ/RAHMENFARBE
WECHSELT
280 PAUSE 1
290 NEXT RF
300 REM
310 PRINT AT(0,11) "" : REM CURSOR POSITIONIEREN
320 CENTRE "HINTERGRUNDFARBE" : PRINT
330 REM
340 FOR HF=0 TO 15
350 COLOUR 0,HF : REM RAHMENFARBE = SCHWARZ/HINTERGRUNDFARBE
WECHSELT
360 PAUSE 1
370 NEXT HF
```



### 9.3.2 BCKGNDS

```
FORMAT      :BCKGNDS HF0.HF1.HF2.HF3
PARAMETER   : - HF0: HINTERGRUNDFARBRE-
              GISTER 0 BELEGEN (0-15)
              - HF1: HINTERGRUNDFARBRE-
              GISTER 1 BELEGEN (0-15)
              - HF2: HINTERGRUNDFARBRE-
              GISTER 2 BELEGEN (0-15)
              - HF3: HINTERGRUNDFARBRE-
              GISTER 3 BELEGEN (0-15)
BEISPIEL    :BCKGNDS 3.4.5.6
FUNKTION     :EINSCHALTEN DES EXTENDED COLOR-
              MODES UND BESTIMMEN DER 4
              HINTERGRUNDFARBEN
```

#### ERLÄUTERUNGEN:

Bevor Sie sich mit diesem Befehl beschäftigen, sollten Sie unbedingt die Einleitung zu diesem Abschnitt (9.3) gelesen haben, da Ihnen ansonsten höchstwahrscheinlich die gesamte Grundlage zum Verständnis der folgenden Zusammenhänge fehlt. Dieses Kommando bietet Ihnen eine Möglichkeit, die Sie wohl bisher noch nicht von Ihrem Rechner kannten, den extended Color - Mode. Der CBM 64 hat noch eine ganze Menge Weiteres auf dem Kasten, was bisher in den wenigsten Programmen allgemeinzugänglich nutzbar dargeboten wird. Was der extended Color - Mode ist, und welche Eigenschaften er besitzt, wurde ausführlich in der Einleitung zu diesem Abschnitt dargelegt. Wie Sie dort erfuhren, können Sie in dieser Betriebsart jedem Bildschirmzeichen eine eigene Hintergrundfarbe zuweisen. Dabei wählen Sie jeweils für jedes Zeichen, aus welchem Hintergrundfarbregister die Farbe stammen soll. Gleichzeitig beschränkt sich Ihr Zeichenvorrat auf 64 Zeichen.

Mit BCKGNDS schalten Sie nun diesen Zustand ein, der natürlich nur im Textmodus sichtbar ist, und bestimmen die Farben, die den vier Registern zugeordnet werden sollen. Diese vier Farben wählen Sie mit den Parametern hf0, hf1, hf2 und hf3, die die Farbcodes der jeweiligen Farben darstellen. Ist hf0 beispielsweise gleich 7, so erhält das Hintergrundfarbregister 0 die Farbe gelb --- als Hintergrundfarbe aller Zeichen, die ganz normal eingegeben

wurden. Einige unten erläuterte Beispiele mögen dies erläutern:

### BEISPIEL 1:

```
100 REM #####
110 REM ##                               ##
120 REM ##  BCKGNDS-BEISPIEL-1  ##
130 REM ##                               ##
140 REM #####
150 REM
160 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
170 PRINT CHR$(5) : REM TEXTFARBE = WEISS
180 COLOUR 0.0 : REM RAHMENFARBE = SCHWARZ
190 BCKGNDS 2.3.6.7 : REM EXTENDED COLOUR - MODUS EINSCHALTEN
200 REM UND HINTERGRUNDFARBEN BESTIMMEN
210 FOR X=0 TO 255
220 POKE 2*X + 1024.CODE : REM ZEICHEN SETZEN
230 POKE 2*X + 55296.0 : REM ZEICHENFARBE IN FARBRAM
  (SCHWARZ)
240 CODE = CODE + 1 : REM NAECHSTER BILDSCHIRMCODE
250 NEXT X
260 PRINT AT(0.15) "" : REM CURSOR FUER READY.
POSITIONIEREN
```

Dieses Beispiel stellt sämtliche möglichen Zeichen in allen 4 Hintergrundfarben, die vorher mittels BCKGNDS definiert wurden, auf dem Bildschirm dar. Dabei sollten Ihnen die Zeilen 220 und 230 nicht unerklärlich sein, wenn Sie die Ausführungen über den Befehl FCHR gelesen haben. Hier wird zunächst der in Zeile 240 stets erhöhte Bildschirmzeichencode an eine Stelle in den Bildschirmspeicher gelegt; anschließend setzt man die Farbe in die entsprechende Farbram - Speicherzelle, um das Zeichen auch sichtbar zu machen. Ändern Sie doch einmal mit BCKGNDS die einzelnen Hintergrundfarben!

**BEISPIEL 2:**

```
100 REM #####
110 REM ##                               ##
120 REM ## BCKGNDS-BEISPIEL-2 ##
130 REM ##                               ##
140 REM #####
150 REM
160 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
170 PRINT CHR$(5) : REM TEXTFARBE = WEISS
180 COLOUR 0.0 : REM RAHMENFARBE = SCHWARZ
190 BCKGNDS 2.5.6.0 : REM EXTENDED COLOUR - MODUS EINSCHALTEN
200 REM UND HINTERGRUNDFARBEN BESTIMMEN
210 REM ES FOLGEN NORMALE ZEICHEN:
220 CENTRE "WIE SIE SEHEN. KANN MAN IM" : PRINT
230 CENTRE "EXTENDED COLOUR - MODUS" : PRINT
240 CENTRE "VOELLIG NORMAL SCHREIBEN." : PRINT
250 PAUSE 9
260 REM
270 REM DIE FOLGENDEN ZEICHEN SIND ALLE GE<SHIFT>ET (SICHTBAR
MIT <SHIFT>< C < > )
280 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
290 PRINT : REM LEERZEILE
300 CENTRE "ABER SIE BESITZEN BEZUEGLICH" : PRINT
310 CENTRE " HINTERGRUNDFARBE EINE " : PRINT
320 CENTRE "GANZE PALETTE MEHR MOEGLICHKEITEN" : PRINT
330 PAUSE 9
340 REM
350 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
360 PRINT : REM LEERZEILE
370 RV$ = CHR$(18) : REM RVS ON FUER HINTERGRUNDFARBE
380 REM DIE FOLGENDEN ZEICHEN SIND OHNE <SHIFT> ABER INVERS
GEZEICHNET
390 CENTRE RV$ + "SIE KOENNEN FUER JEDES ZEICHEN" : PRINT
400 CENTRE RV$ + "EINE SEPARATE HINTERGRUND-" : PRINT
410 CENTRE RV$ + "FARBE VON 4 MOEGLICHEN AUSWAEHLEN" : PRINT
420 PAUSE 9
430 REM
440 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
450 PRINT : REM LEERZEILE
460 REM DIE FOLGENDEN ZEICHEN SIND MIT <SHIFT> UND INVERS
```

## GEZEICHNET

```
470 KOS = CHR$(172) : REM KOMMACODE + 64 FUER HF3 ERGIBT DAS  
ZEICHEN < C< > D  
480 CENTRE RV$ + "ALLERDINGS MUESSEN SIE SICH" : PRINT  
490 CENTRE RV$ + "DARAUF BESCHRAENKEN" + KOS +  
" KEINE GRAPHIK" : PRINT  
500 CENTRE RV$ + "ZEICHEN IN IHRER AUSGABE ZU VERWENDEN" :  
PRINT  
510 PAUSE 9  
520 REM  
530 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN  
540 PRINT : REM LEERZEILE  
550 CENTRE "NATUERLICH LASSEN " + RV$ + "SICH ALLE" : PRINT  
560 PRINT : REM LEERZEILE  
570 ROS = CHR$(146) : REM RVS OFF  
580 HS = "HI" + RV$ + "NT" + ROS + "ER" + RV$ + "GR" + ROS +  
"UN" + RV$ + "D "  
590 HS = HS + ROS + "FA" + RV$ + "RB" + ROS + "EN" : REM WORT  
ZUSAMMENSETZEN  
600 PRINT AT(10,4) HS  
610 PRINT : REM LEERZEILE  
620 CENTRE "GLEICHZEITIG " + RV$ + "DARSTELLEN"  
630 PAUSE 15  
640 REM  
650 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN  
660 PRINT : REM LEERZEILE  
670 CENTRE "SCHAUEN SIE SICH DOCH EINMAL DAS" : PRINT  
680 CENTRE "FOLGENDE PROGRAMMLISTING IN DIESEM" : PRINT  
690 CENTRE "MODUS AN:KOENNEN SIE ES SICH ERKLAEREN?" : PRINT  
700 PAUSE 9  
710 REM  
720 LIST
```

Wie Sie sicher gemerkt haben, wurde dieses Listing in Kleinbuchstaben ausgedruckt. Dies hat eine besondere Bedeutung:

Es wurde der alternative Zeichensatz des Commodore 64 verwendet, d.h. alles, was sonst groß geschrieben steht wird nun klein ausgegeben und alle ge<shift>eten Graphikzeichen erscheinen nun als Großbuchstaben. Da wir aber diese ge<shift>eten Zeichen benötigen, um alle Hintergrundfarben an-

zusprechen, haben wir diese Ausgabe gewählt. Schalten Sie also Ihr Gerät vor der Eingabe dieses Programms unter gleichzeitigem Drücken der Commodore- und der <shift>-Tasten (Erstere wird in den REM-Zeilen des Programms mit c< ausgedrückt) auf den Groß-/Kleinschrift-Modus um. Im extended Color-Modus sieht das Ganze dann auch wieder anders aus.

Nun zur Erklärung: Alle normal geschriebenen Zeichen werden auch normal mit der üblichen Hintergrundfarbe 0 ausgegeben, dies sind im Programm die nicht geschifteten kleinen Buchstaben (CHR\$(32) bis CHR\$(95)). Alle Zeichen, die mit <shift> oder der Commodore-Taste eingegeben wurden (CHR\$(96) bis CHR\$(191), ohne CHR\$(128) bis CHR\$(159)), werden dagegen mit der Hintergrundfarbe 1 gezeichnet.

Welche Tasten nun welchen Zeichen entsprechen, erkunden Sie folgendermaßen:

1.) Sie suchen sich den Code des gewünschten Zeichens in der Bildschirmcodetabelle (s.Handbuch). Das Zeichen darf keinen Code größer als 63 besitzen, sonst kann es nicht dargestellt werden.

2.) Addieren Sie 64 zu dem erhaltenen Code und sehen Sie in der selben Tabelle nach, welches Zeichen diesem neuen Code entspricht.

3.) Suchen Sie nun in der ASCII-Tabelle Ihres Handbuches das gefundene Zeichen heraus, und schon besitzen Sie den ASCII-Wert, den Sie ausgeben müssen, um das gewünschte Zeichen in Hintergrundfarbe 1 (oder 3) zu erhalten.

Alle Buchstaben können Sie ohne dieses komplizierte Nachschlageverfahren einfach durch gleichzeitiges Drücken der <shift> - Taste in der gewünschten Hintergrundfarbe erzeugen. Bei den Ziffern oder Satzzeichen dagegen müssen Sie leider in oben aufgeführter Weise vorgehen (s. Zeile 470 des zweiten Beispiels).

Wollen Sie nun ein Zeichen mit der Hintergrundfarbe 2 darstellen, so gehen Sie genauso vor, als wollten Sie mit der Farbe 0 zeichnen, nur müssen Sie hier alle Zeichen im Invers-Modus (nach Eingabe von RVS ON) ausgeben. Die Hintergrundfarbe 3 stellen Sie durch inverses Ausgeben der

Zeichen dar, die Sie sonst für die Farbe 1 verwenden.

**BEISPIEL: DIE ZIFFER 9 UND DER BUCHSTABE C DARGESTELLT  
IN DEN 4 HINTERGRUNDFARBEN:**

**HINTERGRUNDFARBE 0 : PRINT "9"  
PRINT "C"**

**HINTERGRUNDFARBE 1 : PRINT CHR\$(185)  
PRINT "C"**

**HINTERGRUNDFARBE 2 : PRINT CHR\$(18) "9"  
PRINT CHR\$(18) "C"**

**HINTERGRUNDFARBE 3 : PRINT CHR\$(18) CHR\$(185)  
PRINT CHR\$(18) "C"**

Sollten Sie trotz der vorgehenden Ausführungen weiterhin Verständnisschwierigkeiten bezüglich dieser Art der Zeichendarstellung haben, so werfen Sie dieses Buch nicht "frustriert" in den nächsten Papierkorb. Überschlagen Sie zunächst diese Kapitel und kommen später (nach der Graphik - Lektüre zum Beispiel oder erst beim zweiten Lesen dieses Buches) noch einmal hierauf zurück, wenn Sie Ihren Rechner etwas genauer kennen. Allerdings: ohne probieren und werkeln geht es nicht. Sie müssen schon jedes Beispiel durchgehen, da die weiteren Erklärungen auf diesen Beispielen fußen. Also frisch ans Werk!

### 9.3.3 NRM

<b>FORMAT</b>	<b>:NRM</b>
<b>PARAMETER</b>	<b>:-----</b>
<b>BEISPIEL</b>	<b>:NRM</b>
<b>FUNKTION</b>	<b>:AUSSCHALTEN DES EXTENDED COLOR - MODES</b>

#### **ERLÄUTERUNGEN:**

Nun, nachdem Sie alles durchgearbeitet haben, was Sie über den extended Color - Modus wissen sollten, fehlt nur noch ein Teil im Mosaik dieses Themenkomplexes: das Ausschalten des extended Color - Modus.

Diese Funktion übernimmt der Befehl NRM (für Normal), der gleichzeitig den letzten funktionstüchtigen, bisher unbekanntem Befehl darstellt. Mit NRM wird jedoch lediglich der besagte Modus ausgeschaltet. Die Hintergrundfarbe 0, die nun wieder die einzige Hintergrundfarbe ist, bleibt dabei erhalten. Sie kann nun wieder mit COLOUR verändert werden. NRM wird uns noch einige Male begegnen, da er einige Aufgaben hat, die im Folgenden kurz dargestellt sind:

- 1.) AUSSCHALTEN DES EXTENDED COLOR - MODUS**
- 2.) AUSSCHALTEN DES GRAPHIKMODUS**
- 3.) AUSSCHALTEN DES MEM-MODUS (D.H. RÜCKSETZEN AUF DEN ALTEN ZEICHENSATZ)**
- 4.) EINSCHALTEN DES GROBSCHRIFT-/GRAPHIKZEICHEN-MODUS**

#### **BEISPIEL:**

**GEBEN SIE NACH ABLAUF DES 2. BEISPIELS UNTER 9.3.2 NRM EIN.**

### 9.3.4 TESTAUFGABEN

1.) Sie wollen die Hintergrundfarbe weiß und die Rahmenfarbe gelb gestalten. Welche Befehle geben Sie ein?

- a) COLOUR 1,7
- b) COLOUR 7,1
- c) BCKGNDS 1,7
- d) POKE 53248+33,1 : POKE 53248+32,7

2.) Sie wollen das Zeichen "A" mit der Hintergrundfarbe 2 darstellen. Wie steuern Sie die Ausgabe?

- a) PRINT CHR\$(65)
- b) PRINT CHR\$(97)
- c) PRINT CHR\$(18) CHR\$(65)
- d) PRINT CHR\$(18) CHR\$(97)

Die (auch hier nicht einfachen) Lösungen stehen wie immer im Anhang.



## 9.4 STEUERUNG: BILDSCHIRMBEREICHE

In diesem Abschnitt sollen eine Reihe von sehr nützlichen Befehlen angesprochen werden, die sich mit der Handhabung großer Bildschirmbereiche beschäftigen. Hiermit stehen Ihnen schnelle und effektive Möglichkeiten zur Verfügung, Ihren Bildschirminhalt zu manipulieren. Allerdings treten hier schon einige Verständnisprobleme auf, wenn man nichts Genaues über den Aufbau des Bildschirms, also wie der Rechner Ihr Bild mit den Farben etc. überhaupt herstellt, weiß. Aus diesem Grunde wird Ihnen nun erst einmal eine kleine Einführung in die "Hardware" des sogenannten Textfensters gegeben:

Damit sich der Rechner alle Ausgaben, die auf dem Bildschirm stehen, merken kann (er muß dieses Bild auf Ihrem Fernseher schließlich alleine alle 1/20 Sekunden selbst erstellen, damit Sie es ständig beobachten können), legt er alle Zeichen in den sogenannten Bildschirmspeicher ab. Dieser Bildschirmspeicher umfaßt etwa 1 K (ca. 1000 Byte = 1000 Zeichen) und geht von der Speicherstelle \$400 (Hexadezimal), d.i. 1024 (Dezimal) bis hin zu \$7E7 hexadezimal oder 2023 dezimal.

Den einzelnen Zeichen werden jeweils bestimmte Codes zugeordnet. Dies kennen Sie sicher bereits von der sogenannten ASCII-Codierung. Die Bildschirmcodes entsprechen jedoch nicht der geläufigen ASCII-Codierung, sondern haben für jedes Zeichen einen eigenen Code. Dabei wird natürlich ebenfalls zwischen inversen und normalen Zeichen unterschieden, denn wie sollte der Rechner anhand eines Codes anders wissen, ob er nun das Zeichen invers oder normal auf den Fernseher bringen soll?

Eine Tabelle der Bildschirmcodes finden Sie in Ihrem deutschen Benutzerhandbuch auf den Seiten 133 und 134. Wenn Sie zu den einzelnen Werten jeweils 128 hinzuaddieren, so erhalten Sie das gleiche Zeichen in inverser Form.

Beispiel:

```
10 PRINT CHR$(147) CHR$(158); : REM BILDSCHIRM LOESCHEN
- FARBE = GELB
20 FOR X=1 TO 100
30 PRINT " "; : REM FARBE IN BETREFFENDEM BEREICH SETZTEN
40 NEXT X
50 FOR X=1024 TO 1124
60 POKE X,1 : CODE FUER A IN BILDSCHIRMSPEICHER POKEN
70 NEXT X
```

In obigem Programm werden von oben links in der Ecke 100 "A" in den Bildschirmspeicher geschrieben. In den Zeilen 10 bis 40 wird zunächst mittels Leerzeichen die Farbe gesetzt (ein Problem, das ein Stück weiter unten erläutert wird), dann wird in die ersten hundert Speicherzellen des Bildschirmspeichers der Wert 1 gespeichert als Code für das Zeichen "A".

Neben den Zeichen muß aber für jedes Zeichen gleichfalls die Zeichenfarbe gespeichert werden, da ja bekanntlich rein theoretisch jedes Zeichen eine andere Farbe besitzen kann. Hierfür existiert ein weiterer sogenannter Farbspeicher oder Farbram mit der gleichen Größe wie der Bildschirmspeicher, der die notwendigen Informationen enthält. Dieser Farbram geht von hexadezimal \$D800 BIS ca. \$DFFF bzw. Dezimal von 55296 bis 56295 und kann natürlich wie der Bildschirmspeicher genauso mit POKE angesprochen werden. Dabei bestimmt jedes Byte des Farbrams die Farbe des dazugehörigen Zeichens aus dem Bildschirmspeicher.

Die Hintergrundfarbe des Textbildes wird dagegen durch ein einziges Register des sogenannten Videochips angesprochen (s. unter Graphik; der Videochip oder auch VIC ist der IC (integrierter Schaltkreis) in Ihrem Computer, der das gesamte Bild auf dem Fernseher oder Monitor herstellt.). Dieses Register liegt in der Speicherstelle 53248+33 und kann z.B. mit

```
POKE 53248+33,0 : REM HINTERGRUNDFARBE = SCHWARZ
verändert werden. Doch dazu mehr in dem Kapitel Graphik.
```

Ein wichtiges Problem bei einigen der in diesem Abschnitt

angesprochenen Befehlen ist das Folgende:

Wird am Anfang eines Programmes oder auch zwischendurch der Bildschirm gelöscht, so wird neben dem Bildschirmspeicher auch der Farbram gelöscht. Letzterer, indem jedes Byte die Farbe des Hintergrundes bekommt. Aus diesem Grunde sehen Sie zunächst nichts, wenn Sie lediglich etwas in den Bildschirmspeicher hineinschreiben, da das Zeichen, das eigentlich auf dem Bildschirm erscheinen sollte ja die gleiche Farbe besitzt, wie der Hintergrund. Wenn Sie allerdings mit dem Cursor über die Stelle gehen, wo Sie das Zeichen vermuten oder die Hintergrundfarbe ändern, dann kommt es zum Vorschein. Erst, wenn Sie die korrespondierende Speicherstelle mit der Farbe laden, die das Zeichen erhalten soll, bekommt es die richtige Farbe.

#### **BEISPIEL:**

```
10 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
20 POKE 1024.2 : REM EIN B IN BILDSCHIRMSPEICHER
30 STOP : REM KANN AUCH WEGGELASSEN WERDEN
40 POKE 55296.0 : REM ZEICHEN ERHAELT DIE FARBE SCHWARZ
```

Lassen Sie zunächst Zeile 30 in diesem Programm und bewegen Ihren Cursor nach Ablauf des Programms in die linke obere Ecke. Dort befindet sich ein B. Nun löschen Sie Zeile 30 und starten das Programm ein weiteres Mal. Jetzt wird das zugehörige Farbbyte im Farbram gesetzt und schon erscheint Ihr B schwarz.

So, nun besitzen Sie das nötige Rüstzeug, um sich mit den folgenden Befehlen zu befassen.

## 9.4.1 FCHR

```
FORMAT      :FCHR Z.S.B.H.C
PARAMETER   : - Z: ZEILE (0-24) UND
              - S: SPALTE (0-39) DER OBEREN
                LINKEN ECKE DES FELDES
              - H: HÖHE UND
              - B: BREITE DES FELDES
              - C: DEFINIERT DEN ZU SETZENDEN
                BILDSCHIRMCODE (0-255)
BEISPIEL    :FCHR 3.5.7.9.1
FUNKTION    :FÜLLEN EINES BILDSCHIRMBEREICHES
              MIT EINEM ZEICHEN
```

### ERLÄUTERUNGEN:

Mit FCHR (von FILL CHARACTERS) wird Ihnen ermöglicht, einen ganzen Bildschirmbereich mit einem Zeichen zu füllen, dessen Bildschirmcode (s. Einführung zu diesem Abschnitt) Sie mit c angeben. Es wird immer ein Rechteck, das durch die einzelnen Parameter bestimmt wird, mit diesem einen Zeichen beschrieben. Dabei geben die ersten beiden Parameter die absolute Lage des Rechtecks auf dem Bildschirm an:

z stellt hier die Zeile, s die Spalte der oberen linken Ecke Ihres Rechtecks an. z und s sind somit praktisch die Anfangskoordinaten.

Mit h und b nun legen Sie die Höhe bzw. die Breite des Feldes fest, wobei stets in Einheiten eines Zeichens gemessen wird, d.h. h oder b geben die relativen Ausmaße (relativ zu dem mit z und s festgelegten "Ursprungspunkt") Ihres Zeichenrechtecks an. h und b dürfen also nicht größer werden, als die Anzahl der Zeilen oder Spalten vom Ursprungspunkt bis zum Ende der Spalte bzw. Zeile. Beginnen Sie also mit Ihrem Rechteck in Zeile 10 / Spalte 10, so darf h nur noch maximal den Wert von 14 annehmen, da nur noch 14 Zeilen bis zum unteren Rand des Bildschirms frei sind. Umgekehrt darf b höchstes nur noch 29 werden, da dies die Zahl der freien Zeichen bis zum rechten Rand ist. Werden diese Werte überschritten, ragt nach Ihrer Eingabe also das Rechteck über den Bildschirmrand hinaus. Geben Sie für einen der Parameter den Wert 0 ein, so wird die Fehlermeldung BAD MODE, die uns noch weiter beschäftigen wird, ausgegeben.

Wie man aus den obigen Angaben erkennt, ersetzt der Befehl FCHR folgende Basicroutine:

```
170 FOR Y=Z TO H : REM ZEILENSCHLEIFE
180 FOR X=S TO B : REM SPALTENSCHLEIFE
190 POKE 1024 + 40*Y + X.C : REM ZEICHEN SETZEN
210 NEXTX,Y : NAECHSTE SPALTE/ZEILE
```

Hier bedeuten Z,H,S,B,C das Gleiche wie in der obigen Nomenklatur (also Zeile, Höhe, Spalte, Breite, Code). Mit Zeile 190 wird die Adresse aus der aktuellen (d.h. im Moment behandelten) Zeile (Y) und Spalte (X) berechnet: 1024 ist die Anfangsadresse des Bildspeichers. Dazu werden für jede Zeile 40 Zeichen hinzugezählt (jede Zeile enthält ja 40 Zeichen). Zuletzt wird noch die Anzahl der Zeichen der angebrochenen Zeile addiert und die Adresse des Bildschirmzeichens ist berechnet. Diese Formel (Adresse = 1024 + 40\*Zeilen + Spalte) können Sie bei Ihrer Adressenberechnung stets anwenden, wenn Sie direkt mit dem Bildschirmspeicher arbeiten. Wenn Sie einmal einen Geschwindigkeitsvergleich anstellen wollen, so sei hier ein vollständiges Beispiel eines FCHR-Ersatzes gegeben:

```
100 REM #####
110 REM ##                ##
120 REM ## FCHR-ERSATZ ##
130 REM ##                ##
140 REM #####
150 REM
160 Z=1 : H=23 : S=2 : B=35 : C=1 : REM PARAMETER
170 FOR Y=Z TO H : REM ZEILENSCHLEIFE
180 FOR X=S TO B : REM SPALTENSCHLEIFE
190 POKE 1024 + 40*Y + X.C : REM ZEICHEN SETZEN
200 REM POKE 55296 + 40*Y + X.O : REM FARBE SETZEN
210 NEXTX,Y : NAECHSTE SPALTE/ZEILE
```

Wenn Sie dieses kleine Programm ausprobiert haben, so haben Sie sicher bemerkt, daß sich bei gelöschtem Bildschirm recht wenig tut. Dies hat mit der in der Einleitung beschriebenen Trennung von Zeichen- und Farbspeicher zu tun. Aus diesem Grunde wurde in Zeile 200 ein (zur Zeit noch inaktiver - weil

in einer REM-Zeile stehender) Zusatz eingeführt: die Farbgebung.

Streichen Sie das erste REM in Zeile 200 und lassen Ihr Programm nun laufen, so haben Sie mehr Erfolg. Dieser Befehl ist eigentlich ein kleiner Vorgriff auf den Befehl FCOL, der im nächsten Abschnitt besprochen werden soll: Sie haben hier praktisch eine Kombination der beiden Befehle FCHR und FCOL, die eigentlich eng zusammengehören, vorliegen. Gleichzeitig wurde hier eine Formel vorgestellt, die Sie zur Berechnung des einem Zeichen zugehörigen Farbspeichers verwenden können. Sie ähnelt zwangsläufig der oben beschriebenen Bildspeicherformel. Diese beiden unterscheiden sich lediglich in Ihrer Basisadresse. Während der Bildspeicher wie erwähnt bei Adresse 1024 beginnt, liegt der Farbram (s.o.) ab 55296.

In diesem Beispiel wurden willkürlich die dem Befehl FCHR angehörigen Parameter in der Zeile 160 gesetzt - verändern Sie doch einmal und probieren Sie im Gegensatz dazu einmal den FCHR-Befehl selbst aus. Den Bildschirmcode eines Zeichens erfahren Sie, wie oben in der Einleitung erwähnt, aus Ihrem CBM64 - Benutzerhandbuch auf den Seiten 133/134.

#### **BEISPIEL:**

**DOCH NUN EIN BEISPIEL. DAS IHNEN DIE WIRKUNGSWEISE DES FCHR-BEFEHLS DARLEGEN SOLL:**

```
100 REM #####
110 REM ##                ##
120 REM ## FCHR-BEISPIEL ##
130 REM ##                ##
140 REM #####
150 REM
160 POKE 53248 + 33.1 : REM HINTERGRUNDFARBE = WEISS
170 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
180 CENTRE "IM MOMENT IST DER BILDSCHIRM LEER" : PRINT
190 PAUSE 3
200 FCHR 3.0.40.22.1 : REM ZEICHEN (A) OHNE FARBE SETZEN:
210 REM AB REIHE 3. SPALTE 0 MIT DER BREITE 40 UND DER HOEHE
22
220 PRINT AT(0.0) "": : REM CURSOR OBEN POSITIONIEREN
230 CENTRE "NUN WURDEN ZEILEN 3 BIS 24 MIT ZEICHEN" : PRINT
240 CENTRE "OHNE FARBE GEFUELLT. D.H. SIE SEHEN:" : PRINT :
```

```
PRINT
250 REM LEERZEILE
260 CENTRE "NICHTS!"
270 PAUSE 6
280 PRINT AT(0,0) " " ; REM CURSOR POSITIONIEREN
290 CENTRE "NUN WERDEN WIR EINMAL DIE HINTERGRUND-" : PRINT
300 CENTRE " FARBE AENDERN UND SCHON WIRD ALLES " : PRINT :
PRINT
310 REM LEERZEILE
320 CENTRE "DURCHSICHTIGER"
330 PAUSE 6
340 POKE 53248 + 33,14 : REM HINTERGRUNDFARBE = HELLBLAU
350 PAUSE 6
360 POKE 53248 + 33,1 : REM HINTERGRUNDFARBE WIEDER WEISS
370 PRINT AT(0,0) " " ; REM CURSOR POSITIONIEREN
380 CENTRE " ODER , WENN SIE MIT DEM CURSOR UEBER " : PRINT
390 CENTRE " DEN BILDSCHIRM GEHEN, ERKENNEN SIE " : PRINT :
PRINT
400 REM LEERZEILE
410 CENTRE "WAS SACHE IST!"
```

Wir hoffen, dieses Beispiel dokumentiert sich selbst in  
ausreichendem Maße.

## 9.4.2 FCOL

**FORMAT** :FCOL Z.S.B.H.F  
**PARAMETER** : - Z: ZEILE (0-24) UND  
- S: SPALTE (0-39) DER OBEREN  
LINKEN ECKE DES FELDES  
- H: HOHE UND  
- B: BREITE DES FELDES  
- F: DEFINIERT DIE ZU SETZENDE  
ZEICHENFARBE (0-15)  
**BEISPIEL** :FCOL 5.4.5.3.8  
**FUNKTION** :FÜLLEN EINES BILDSCHIRMBE-  
REICHES MIT EINER ZEICHENFARBE

### ERLÄUTERUNGEN:

Bevor Sie diesen Abschnitt bearbeiten, der sich mit dem FCOL-Befehl auseinandersetzt, ist es für Ihr Verständnis unbedingt notwendig, vorher die Einleitung dieses Abschnittes 9.4 und (!) die obigen Ausführungen über den FCHR-Befehl gelesen und durchgearbeitet zu haben, da die beiden Befehle FCHR und FCOL sehr eng miteinander verknüpft sind.

Mit den ersten 4 Parametern des FCOL-Befehls, die im Übrigen in der Funktionsweise genau den Angaben im FCHR-Befehl (s.o.) entsprechen, hier also nicht extra wieder besprochen werden, geben Sie die Lage und die Größe eines Rechtecks an, das mit einer bestimmten Zeichenfarbe gefüllt werden soll. D.h. der Farbram, der ja bekanntlich die Farbe jedes einzelnen Zeichens auf Ihrem Bildschirm definiert, wird an dieser Stelle mit der durch f angegebenen Farbe aufgefüllt (f ist der Farbcode einer bestimmten Farbe und geht von 0 bis 15; s. Anhang). Jedes Zeichen in diesem Bereich bekommt also nun die angegebene Farbe f. Wollen Sie jedoch in diesen Bereich neuen Text hineinschreiben, so wird durch den PRINT-Befehl die Zeichenfarbe im Farbram für die betreffenden Zeichen neu gesetzt, d.h. neuer Text bekommt auch eine neue Farbe. Wollen Sie jedoch für diesen neuen Text (Zeichen) die alte Farbe aus dem Farbram übernehmen, so müssen Sie direkt den Bildschirmspeicher auf oben beschriebene Weise mit POKE (oder z.B. mit FCHR) ansprechen. Die Konsequenz aus diesen Darlegungen lautet:

Die Auswirkungen des FCOL-Befehles sind nur dann sichtbar,



wenn auch Text (Zeichen) in dem angesprochenen Bildschirmbereich vorhanden ist (genauso, wie FCHR nur wirkt, wenn die Farbe im Farbram bereits gesetzt ist oder direkt gesetzt wird).

Natürlich können wir den FCOL-Befehl genauso wie auch FCHR durch eine kleine Basic-Routine ersetzen, die zum besseren Verständnis hier angebracht wird. Die Erläuterungen hierzu entsprechen den unter FCHR gebrachten:

```
160 FOR Y=Z TO H : REM ZEILENSCHLEIFE
170 FOR X=S TO B : REM SPALTENSCHLEIFE
190 POKE 55296 + 40*Y + X.F : REM FARBE SETZEN
200 NEXT X.Y : NAECHSTE SPALTE/ZEILE
```

Bitte beachten Sie auch hier, daß Sie eventuell recht wenig von diesem Befehl sehen, wenn Sie keinen Text auf dem Bildschirm haben.

#### BEISPIELE:

ALS BEISPIEL SOLL DAS FOLGENDE PROGRAMM DIENEN. DAS IHNEN EINEN EINBLICK IN DIE WIRKUNGSWEISE DES FCOL-BEFEHLS GIBT:

```
100 REM #####
110 REM ##          ##
120 REM ## FCOL-BEISPIEL 1 ##
130 REM ##          ##
140 REM #####
150 REM
160 POKE 53248 + 33,4 : REM HINTERGRUNDFARBE = LILA
170 PRINT CHR$(147); : REM BILDSCHIRM LOESCHEN
180 PRINT CHR$(5) : REM ZEICHENFARBE = WEISS
190 CENTRE "IM MOMENT IST DER BILDSCHIRM LILA" : PRINT
200 PAUSE 3
210 PRINT AT(0,0) " "; : REM CURSOR OBEN POSITIONIEREN
220 CENTRE "NUN WERDEN DIE ZEILEN 01 BIS 24 MIT" : PRINT
230 CENTRE "DER FARBE BLAU GEFUELLT. D.H. SIE" : PRINT
240 CENTRE "SEHEN NUR DIE BESTEHENDEN" : PRINT
250 CENTRE "ZEICHEN VERAENDERT." : PRINT : PRINT
260 REM LEERZEILE
270 PAUSE 8
280 PRINT CHR$(159); : REM ZEICHENFARBE = TUERKIS
```

```

290 CENTRE "ACHTUNG!!!" : PRINT : PRINT
300 REM LEERZEILE
310 PAUSE 1
320 FCOL 1.0.40.24.6 : REM FARBE (BLAU) OHNE ZEICHEN SETZEN:
330 REM AB REIHE 0, SPALTE 0 MIT DER BREITE 40 UND DER HOEHE
340
350 PRINT CHR$(5) : REM ZEICHENFARBE = WEISS
360 CENTRE "ALLES, WAS NUN GESCHRIEBEN WIRD" : PRINT
370 CENTRE "WIRD WIEDER IN DER URSPRUNGSFARBE" : PRINT
380 CENTRE "GESETZT."

```

ALS NÄCHSTES WIRD DER EINSATZ BEIDER BEFEHLE (FCOL, FCHR) IM ZUSAMMENSPIEL DEMONSTRIERT:

```

100 REM #####
110 REM ##                               ##
120 REM ## FCHR/FCOL-BEISPIEL ##
130 REM ##                               ##
140 REM #####
150 REM
160 POKE 53248 + 33.0 : REM HINTERGRUNDFARBE = SCHWARZ
170 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
180 FOR X=0 TO 35 STEP 5 : REM BLOCKLINIE
190 Y = Y+2 : REM ZEILENZAEHLER
200 FCHR Y,X,5,5,Y
210 REM BLOCK VON ZEICHEN MIT DEM BILDSCHIRMCODE Y SETZEN
220 FCOL Y,X,5,5,Y/2
230 REM FARBBLOCK DER FARBE Y/2 SETZEN
240 NEXT X
250 PAUSE 6
260 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
270 Y = 0 : ANFANGSFARBE = SCHWARZ
280 FCHR 0.0.40.25,128+32
290 FCOL 0.0.40.25.6
300 REM BILDSCHIRM MIT INVERSEN BLAUEN LEERZEICHEN FUELLEN
310 FOR X=4 TO 34 STEP 2
320 FCOL 5.X,2,15,Y : REM EINE ZEILE FAERBEN
330 Y = Y+1 : NAECHSTE FARBE
340 NEXT X

```

### 9.4.3 FILL

**FORMAT** : FILL Z.S.B.H.C.F

**PARAMETER** : - Z: BESTIMMT DIE ZEILE (0-24),  
- S: DIE SPALTE (0-39) DER OBEREN LINKEN ECKE DES FELDES  
- H: GIBT DIE HOHE,  
- B: DIE BREITE DES FELDES AN.  
- C: DEFINIERT DEN ZU SETZENDEN ZEICHENCODE (0-255).  
- F: DIE ZU SETZENDE ZEICHENFARBE (0-15)

**BEISPIEL** : FILL 3.5.7.9.2.11

**FUNKTION** : FÜLLEN EINES BILDSCHIRMBEREICHES MIT EINEM GEFÄRBTEN ZEICHEN

#### ERLÄUTERUNGEN:

Vor dem Studium dieses Befehles sollten Sie den gesamten Abschnitt 9.4 bis hierhin gelesen haben.

Haben Sie dies getan, werden Sie staunen. Wie Sie sahen, benötigten wir stets zwei Befehle, wenn wir in einen bestimmten Bildschirmbereich ein Zeichen mit Farbe setzen wollten. Dies bleibt uns nun erspart: Der FILL-Befehl ermöglicht uns nun das Füllen eines Blockes mit einem Zeichen, das wir sofort sehen, da gleichzeitig die Farbe mitgesetzt wird. Wir stehen also nicht mehr vor dem in den beiden vorherigen Befehlsbeschreibungen geschilderten Problem.

Der FILL-Befehl ersetzt praktisch einen FCHR- und FCOL-Befehl hintereinandergesetzt (mit den gleichen Parametern). Dabei kann man die Bedeutung der einzelnen Parameter unter FCHR bzw. FCOL nachlesen.

Statt

**100 FCHR Z.S.B.H.C : FCOL Z.S.B.H.F**

kann man jetzt einfacher sagen:

**100 FILL Z.S.B.H.C.F**

Die beiden obigen Zeilen erfüllen exakt die selbe Funktion. Ebenfalls gelten die selben Aussagen, die bereits über die beiden bekannten Befehle gemacht wurden.

**BEISPIELE:**

**ERSETZEN SIE IM 2. BEISPIEL UNTER FCOL (FCHR/FCOL-BEISPIEL)  
DIE ZEILEN 280 UND 290 DURCH:  
285 FILL 0.0.40.25.128+32.6  
UND DIE ZEILEN 200 UND 220 DURCH:  
200 FILL Y.X.5.5.Y.Y/2**

```
100 REM #####
110 REM ##                               ##
120 REM ## FILL-BEISPIEL-2 ##
130 REM ##                               ##
140 REM #####
150 REM
160 REM #####
170 REM ##                               ##
180 REM ## GRIDRUNNER ##
190 REM ##                               ##
200 REM #####
210 REM
220 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
230 POKE 53248 + 33.6 : REM HINTERGRUNDFARBE = BLAU
240 POKE 53248 + 32.0 : REM RAHMENFARBE = SCHWARZ
250 FOR X=1 TO 100
260 FILL 0.10.20.25.119.7 : REM CHR$(183)
270 FILL 0.10.20.25. 69.7 : REM CHR$(100)
280 FILL 0.10.20.25. 64.7 : REM CHR$( 96)
290 FILL 0.10.20.25. 70.7 : REM CHR$(102)
300 FILL 0.10.20.25.111.7 : REM CHR$(175)
310 NEXT X
```

Na, wenn das kein eindrucksvolles Beispiel ist! Zur Erklärung: In den Zeilen 260 bis 300 werden nacheinander die Zeichen gesetzt, die Striche an verschiedenen Positionen eines Zeichens darstellen. Nacheinander werden immer weiter unten liegende Striche in einem jeweiligen Zeichen gesetzt. Dadurch kommt es zu dem Laufeffekt.

#### 9.4.4 MOVE

**FORMAT** : MOVE Z,S,B,H,ZD,SD  
**PARAMETER** : - Z : BESTIMMT DIE ZEILE (0-24),  
- S : DIE SPALTE (0-39) DER OBEREN LINKEN ECKE DES FELDES  
- H : GIBT DIE HOHE,  
- B : DIE BREITE DES FELDES AN.  
- ZD : NENNT DIE ZEILE (0-24).  
- SD : DIE SPALTE DER NEUEN POSITION  
**BEISPIEL** : MOVE 4.5.2.3.10.11  
**FUNKTION** : DUPLIZIEREN EINES BILDSCHIRMBEREICHES

#### ERLÄUTERUNGEN:

Nun kommen wir zu einem Befehl, der Ihnen rein theoretisch ungeheure Möglichkeiten eröffnet. Der MOVE-Befehl gibt Ihnen die Fähigkeit, Teile des Bildschirms zu kopieren, d.h. verschoben zu duplizieren. Sie können also einen Teil des Bildschirms präparieren und anschließend ohne Aufwand auf andere Teile des Textfensters übertragen.

Die Parameter z,s,b und h kennen Sie bereits von den FCHR- / FCOL- und FILL-Befehlen. Sie bestimmen hier in bekannter Weise den zu duplizierenden Bereich. Dieser Bereich ist also hier wieder ein beliebig geformtes Rechteck. Mit zd und sd nun determinieren Sie die Zeile und Spalte der linken oberen Ecke des Bereiches, in den das Duplikat eingefügt werden soll (also ähnlich den Parametern z und s). paßt jedoch der Teil nicht mehr in den Bildschirm hinein, d.h. träte das Duplikat aus dem Bildschirm heraus, so wird die Fehlermeldung BAD MODE, die uns sicher schon bekannt ist (s.o.), produziert.

An dieser Stelle muß leider eine einschränkende Bemerkung gemacht werden: Die Disketten-/Kassettenversion (bis V2) des Simon's Basic weist hier einen kleinen, aber leider unangenehmen Fehler auf:

Sie können den oben beschriebenen Befehl zwar uneingeschränkt verwenden, ohne irgendeine (falsche) Fehlermeldung zu erhalten. Wenn Sie jedoch für zd oder sd Variablen oder mathematische Ausdrücke einsetzen, so setzt die richtige Funktion des Befehles aus, d.h. er liefert sinnlose Ergebnisse (es

werden nur Teile oder gar nichts dupliziert). Somit können Sie diese Parameter lediglich in Ziffernschreibweise direkt eingeben. Dieser Fehler im Simon's Basic ist natürlich ärgerlich, aber Fehler lassen sich bei der Komplexität solcher Programme kaum vermeiden. Nichtsdestoweniger ist dieser Befehl eine sehr nützliche Hilfe bei der Bildschirmgenerierung.

**BEISPIEL:**

**UM IHNEN DIE WIRKUNGSWEISE DES MOVE-BEFEHLS DARZULEGEN. SEI AN DIESER STELLE EIN KLEINES BEISPIEL ANGEBRACHT:**

```
100 REM #####
110 REM ##          ##
120 REM ## MOVE-BEISPIEL ##
130 REM ##          ##
140 REM #####
150 REM
160 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
170 PRINT "DIESER TEIL "
180 PRINT "DES BILDES "
190 PRINT "WIRD JETZT "
200 PRINT "DUPLIZIERT!"
210 PAUSE4
220 MOVE 1.0.11.4.1.28
230 PAUSE1
240 MOVE 1.0.11.4.5.14
250 PAUSE1
260 MOVE 1.0.11.4.9.0
270 PAUSE1
280 MOVE 1.0.11.4.9.28
290 PAUSE1
300 MOVE 1.0.11.4.13.14
310 PAUSE1
320 MOVE 1.0.11.4.17.0
330 PAUSE1
340 MOVE 1.0.11.4.17.28
350 PAUSE1
```

## 9.4.5 INV

<b>FORMAT</b>	: INV Z.S.B.H
<b>PARAMETER</b>	: - Z: ZEILE (0-24) UND - S: SPALTE (0-39) DER OBEREN LINKEN ECKE DES FELDES - H: HOHE UND - B: BREITE DES FELDES
<b>BEISPIEL</b>	: INV 5.6.3.4
<b>FUNKTION</b>	: INVERTIEREN EINES BILDSCHIRMBEREICHES

### ERLÄUTERUNGEN:

Mit INV wurde Ihnen ein weiterer "bärenstarker" Befehl zur Bildschirmsteuerung in die Hände gegeben. INV ermöglicht es Ihnen, einen ganzen Bildschirmbereich zu invertieren. Die Syntax der dem INV-Kommando folgenden Parameter sollte Ihnen inzwischen von allen anderen Blockbefehlen her bekannt sein (FCHR, FCOL, FILL, und MOVE). Sie definieren bekanntlich den Bildschirmausschnitt, der behandelt werden soll.

Nun, wie wirkt INV denn jetzt? Die Funktion dieses Befehles ist recht einfach erläutert. Sie ähnelt dem FLASH-Befehl, der ja bekanntlich alle Bildschirmzeichen einer Farbe stets mit einer definierten Geschwindigkeit vom inversen zum normalen Zustand blinken läßt und umgekehrt. Mit INV also werden alle Zeichen, die Sie vorher normal (also mit RVS OFF) auf den Bildschirm geschrieben haben von nun an invers auf Ihrem Bild zu sehen sein, gerade so, als hätten Sie sie mit RVS ON durch PRINT gesetzt. Umgekehrt werden alle Zeichen, die vorher invers gezeichnet waren "umgedreht" in den normalen Zustand. Der Unterschied zum FLASH-Befehl liegt also zum Einen darin, daß INV nur einen einmaligen Wechsel (nämlich dann, wenn Sie den Befehl INV senden) des Zeichenzustandes bewirkt, während FLASH ja ständig in einem gegebenen Zeitabstand blinkt. Zum Zweiten können Sie mit INV einen ganz bestimmten Bildausschnitt wählen, wohingegen bei FLASH der gesamte Bildschirm blinkt. Weiterhin läßt FLASH nur eine bestimmte Zeichenfarbe blinken. Im Gegensatz hierzu differenziert INV hier nicht.

Sie können natürlich mit INV Ihr eigenes FLASH basteln, das sich z.B. nur auf einen bestimmten Bildschirmbereich bezieht

und auch nicht vor der Farbe halt macht. Eine kleine Anmerkung: Durch einen kleinen Fehler im Simon's Basic kann es vorkommen, daß, falls Sie zwischen zwei INV-Befehlen, die den gleichen Bereich (oder gleiche Teile) adressieren, einmal die Hintergrundfarbe wechseln, irgendwo in diesem Bereich einmal die alte Hintergrundfarbe auftritt. Dies sollten Sie zur Kenntnis nehmen. Wann und ob dieser Fehler korrigiert wird, ist noch unbekannt.

Selbstverständlich haben Sie ansonsten alle Möglichkeiten des Einsatzes. Einige Anwendungen zeigen Ihnen die folgenden Beispiele:

### BEISPIELE:

#### BEISPIEL 1:

FÜGEN SIE AN DAS MOVE-BEISPIEL VON PARAGRAPH 9.4.4 DIE FOLGENDEN BASICZEILEN HINTEN AN:

```
360 INV 1. 0.11. 4
370 INV 1.28.11. 4
380 INV 17. 0.11. 4
390 INV 17.28.11. 4
400 PAUSE 1
410 GOTO 360
```

```
-----
100 REM #####
110 REM ##          ##
120 REM ## INV-BEISPIEL-2 ##
130 REM ##          ##
140 REM #####
150 REM
160 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
170 POKE 53248 + 33,12 : REM HINTERGRUNDFARBE = GRAU
180 FILL 2.10.10.10.127. 7 : REM = CHR$(191)
190 FILL 14.10.10.10.102.14 : REM = CHR$(166)
200 FILL 2.25.10.10.105.13 : REM = CHR$(169)
210 FILL 14.25.10.10. 97. 2 : REM = CHR$(161)
220 INV 2.10.10.22 : REM ERSTE ZWEI
230 INV 2.25.10.22 : REM ZWEITE ZWEI
```



```
240 FOR X=1 TO 80 : NEXT X : REM KURZE WARTESCHLEIFE  
250 GOTO 220 : REM BLINKEN
```

Zum zweiten Beispiel eine kleine Anmerkung: In Zeile 240 wurde eine programmierte Warteschleife eingefügt. Dies hätte natürlich auch mit PAUSE geschehen können. PAUSE ist aber eher für recht lange Wartezeiten zu gebrauchen, da diese nur in 1 Sekunde-Einheiten bestimmt werden können. Natürlich können Sie die Warteschleife ersetzen oder ihre Dauer variieren. Wenn Sie Beispiel 2 beenden wollen, so drücken Sie RUN/STOP und schon steht der Rechner wieder unter Ihrer Kontrolle.

#### 9.4.6 TESTAUFGABEN

Zum Abschluß dieses Abschnittes wieder ein paar Testaufgaben, um Ihre Sicherheit zu überprüfen:

1.) Sie wollen den Bereich begrenzt durch Zeile 5 und 10 bzw. Spalte 4 und 9 kopieren auf den Bereich begrenzt durch die Zeile 8 und Spalte 11 (jeweils einschließlich). Welche Syntax wählen Sie?

- a) MOVE 5,10,4,9,8,11
- b) MOVE 5,4,10,9,8,11
- c) MOVE 5,4,6,6,8,11
- d) MOVE 4,5,6,6,11,8

2.) Wie füllen Sie einen Teilbereich des Bildschirms mit dem Zeichen A in roter Farbe?

- a) FCHR 5,5,10,10,1
- b) FILL 5,5,10,10,1,2
- c) FCHR 5,5,10,10,1 : FCOL 5,5,10,10,2
- d) FILL 5,5,10,10,65,2

3.) Was geschieht bei dem folgenden Befehl: INV 0,0,0,0 ?

- a) Der Befehl wird ignoriert
- b) Es wird das Zeichen in der oberen linken Ecke invertiert
- c) Die Fehlermeldung BAD MODE wird gesendet
- d) Die Fehlermeldung ILLEGAL QUANTITY ERROR erscheint

Nun, haben Sie den Test bestanden? Wenn ja, dann können Sie weiterlesen (es war diesmal gar nicht einfach).

## 9.5 BILDSCHIRM SCROLLEN

Als Nächstes werden Ihnen insgesamt acht weitere sehr schöne und nützliche Befehle vorgestellt, die sich mit dem Verschieben (Scrollen) von Bildschirmteilen befassen. Sie werden, das versprechen wir Ihnen, zu Ihrem Vergnügen kommen. Die Effekte, die sich mit diesen Kommandos bewerkstelligen lassen, sind so schön, daß man sich tagelang mit Ihnen beschäftigen könnte (Soweit unsere persönliche Meinung).

Nun zum System dieser Befehle:

Die acht Kommandos ähneln sich zum Teil sehr. Deswegen wurde hier die Technik der Sekundärbefehle angewandt. Darunter ist folgendes zu verstehen: Es existieren insgesamt 4 Grundbefehle (LEFT, RIGHT, UP, DOWN), die jeweils die Richtung angeben, in die gerollt werden soll (entsprechend: links, rechts, aufwärts, abwärts). Zu diesen sogenannten Primärbefehlen setzen Sie nun einen weiteren Buchstaben als Sekundärbefehl, der die Art des Scrollens bestimmt. Dabei können Sie aus jeweils 2 möglichen Sekundärbefehlen einen auswählen:

- **W = ZYCLISCHES BILDSCHIRMROLLEN**
- **B = ROLLEN OHNE BILDUMLAUF**

(Ein Befehl könnte also etwa so aussehen (s. auch unten):

### **LEFTW 0.0.10.10**

Wichtig in diesem Zusammenhang ist es, daß zwischen Primär- (z.B. LEFT) und Sekundärbefehl (hier W) kein Leerzeichen eingeschoben wird.)

Unter zyclischem Bildschirmrollen versteht man dabei, daß diejenigen Teile, die aus dem definierten Bereich hinausgeschoben werden, nicht verloren gehen, sondern an der anderen Seite des Blockes wieder auftauchen, sodaß sich das Bild stets im Kreise dreht. Sicherlich kennen Sie diesen Effekt bereits aus vielen Actionspielen, die bereits für den CBM 64 erhältlich sind.

Rollen ohne Bildumlauf dagegen ist dadurch gekennzeichnet, daß hier nicht die Zeichen, die auf der einen Seite hinaus-

geschoben wurden, auf der anderen Seite wieder eingesetzt werden, sondern hier werden vielmehr lediglich Leerzeichen nachgeschoben, sodaß nach ein paar Malen Rollen irgendwann einmal das ursprüngliche Bild verschwunden ist.

Wenn Sie später einmal diese Befehle ausprobieren, um damit ebenfalls Graphiken zu verschieben, werden Sie merken, daß diese Befehle nur für den Textbetrieb gelten. Wollen Sie auch bei der graphischen Darstellung diesen Komfort haben, so müssen Sie sich eine spezielle Graphikerweiterung anschaffen (z.B. Supergraphik 64), die sich auf die Erstellung von Graphiken konzentriert und neben diesen Dingen noch eine Reihe weiterer Möglichkeiten besitzt.

Noch eine Bemerkung in technischer Hinsicht: Probieren Sie ruhig die Beispiele aus, es lohnt sich!

### 9.5.1 LEFT

```
FORMAT      :LEFTW Z.S.B.H
PARAMETER   : - Z: ZEILE (0-24) UND
              - S: SPALTE (0-39) DER OBEREN
                LINKEN ECKE DES FELDES
              - H: HÖHE UND
              - B: BREITE DES FELDES
BEISPIEL    :LEFTW 5.7.3.4
FUNKTION     :LINKS SCROLLEN EINES BILD-
              SCHIRMBEREICHES
```

#### ERLÄUTERUNGEN:

Bei dem Befehl LEFT haben wir es mit dem Komando zu tun, das uns - wie der Name schon sagt - unseren Bildschirm nach links verschiebt. Dabei kann jeweils ein bestimmter rechteckiger Bereich ausgewählt werden, für den die Verschiebung ausgeführt wird. Dabei geben die ersten beiden Parameter die absolute Lage des Rechtecks auf dem Bildschirm an:

z stellt hier die Zeile, s die Spalte der oberen linken Ecke Ihres Rechtecks an. z und s sind somit praktisch die Anfangskordinaten.

Mit h und b nun legen Sie die Höhe bzw. die Breite des Feldes

fest, wobei stets in Einheiten eines Zeichens gemessen wird, d.h. h oder b geben die relativen Ausmaße (relativ zu dem mit z und s festgelegten "Ursprungspunkt") Ihres Zeichenrechtecks an. h und b dürfen also nicht größer werden, als die Anzahl der Zeilen oder Spalten vom Ursprungspunkt bis zum Ende der Spalte bzw. Zeile. Beginnen Sie demnach mit Ihrem Rechteck in Zeile 10 / Spalte 10, so darf h nur noch maximal den Wert von 14 annehmen, da nur noch 14 Zeilen bis zum unteren Rand des Bildschirms frei sind. Umgekehrt darf b höchstes nur noch 29 werden, da dies die Zahl der freien Zeichen bis zum rechten Rand ist. Werden diese Werte überschritten, ragt nach Ihrer Eingabe das Rechteck über den Bildschirmrand hinaus, oder geben Sie für einen der Parameter den Wert 0 ein, so wird die Fehlermeldung BAD MODE, ausgegeben.

Wie Sie sehen stimmt (entgegen den Aussagen des Simon's Basic Handbuchs, das an dieser Stelle einen Fehler aufweist) die Parametersyntax mit der der Bildschirmbereichsbefehle überein, was ja auch zweckmäßig ist.

```

100 REM #####
110 REM ##          ##
120 REM ## LEFT-BEISPIEL ##
130 REM ##          ##
140 REM #####
150 REM
160 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
170   AS = "CBM      64 --- SIMON'S BASIC://" : REM SCHRIFT
DEFINIEREN
180 PRINT AT(7.12) AS : REM POSITIONIERT SCHREIBEN
190 FOR X=1 TO 216 : REM AUFZAEHLEN
200 LEFTW 12.7.LEN(AS).1 : REM LEN(AS) IST DIE LAENGE VON AS
210 FOR Y=X TO 100 : NEXT Y : REM VARIABLE
WARTESCHLEIFE/SCHNELLER WERDEND
220 NEXT X
230 FOR X=216 TO 1 STEP -1 : REM ABZAEHLEN
240 LEFTW 12.7.LEN(AS).1 : REM LEN(AS) IST DIE LAENGE VON AS
250 FOR Y=X TO 100 : NEXT Y : REM VARIABLE
WARTESCHLEIFE/LANGSAMER WERDEND
260 NEXT X
270 PRINT : REM LEERZEILE
280 CENTRE "UEBERZEUGT?"

```

Jetzt wissen Sie, weshalb es Leute gibt, die von Scrolling-Befehlen nur so schwärmen. In dem obigen Beispiel wird eine Laufschrift organisiert, die zunächst zunehmend schneller läuft, dann aber wieder in ihrer Geschwindigkeit abnimmt. Die hierzu zentralen Zeilen sind die Warteschleifen in Zeilen 210 und 250. Die Dauer einer Warteschleife ist hier abhängig von dem momentanen Wert für x, der, wie man sieht, durch die FOR...NEXT - Schleife ständig hoch (Zeilen 190-220) bzw. in der zweiten Schleife ab Zeile 230 wieder ´runter gezählt wird. Damit nichts von der Laufschrift verloren geht, wurde hier der W-Sekundärbefehl verwendet, durch den der nach links hinausgeschobene Teil der Laufschrift rechts wieder zum Vorschein kommt.

## 9.5.2 RIGHT

```
FORMAT      :RIGHTW Z.S.B.H
PARAMETER   : - Z: ZEILE (0-24) UND
              - S: SPALTE (0-39) DER OBEREN
                LINKEN ECKE DES FELDES
              - H: HOHE UND
              - B: BREITE DES FELDES
BEISPIEL    :RIGHTW 4.5.6.7
FUNKTION    :RECHTS SCROLLEN EINES BILD-
              SCHIRMBEREICHES
```

### ERLÄUTERUNGEN:

RIGHT ist der analoge Gegenspieler des LEFT-Befehls. Er verschiebt, wie der Name schon sagt, den mit den bekannten Parametern (s. LEFT) definierten Bildschirmbereich genau in die entgegengesetzte Richtung, nämlich nach rechts. Ansonsten funktioniert RIGHT (rein theoretisch) vollkommen gleich. Ich sage rein theoretisch, da auch hier wieder bei der Arbeit mit diesem Befehl eine kleine (aber glücklicherweise nicht allzu einschränkende (wie z.B. beim MOVE-Befehl)) "Unregelmäßigkeit" aufgetreten ist:

Der Fehler ist besonders gut bei einem ununterbrochenen Rechts-Scrolling zu beobachten (s. Beispiele). Während des Durchlaufs eines Befehls kann es zu einem kurzen Aufblinken verschiedener Zeichen am linken Blockrand des zu verschiebenden Bildschirmausschnitts kommen. Diese Zeichen verschwinden jedoch nach Beendigung des RIGHT-Befehles wieder von Schirm. Dies ist also lediglich ein kleiner Schönheitsfehler, der vielleicht in einer späteren Version schon behoben ist. Doch halten wir uns nicht weiter bei den Erklärungen auf, sondern gehen direkt über zu den Beispielprogrammen, da der Rest ja vom LEFT-Befehl bekannt sein sollte.

### BEISPIELE:

```
100 REM #####
110 REM ##                ##
120 REM ## RIGHT-BEISPIEL ##
```

```

130 REM ##                      ##
140 REM #####
150 REM
160 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
170 FOR X=0 TO 39
180 Y = 12 * SIN(X/6) + 12
190 PRINT AT(X,Y) "*" : REM SINUSKURVE ZEICHNEN
200 NEXT X : REM 40 WERTE
210 AS$ = "----- NA. WIE GEAELLT IHNEN DAS? -----" : REM
SCHRIFT DEFINIEREN
220 FOR X=40 TO 1 STEP -1
230 RIGHTB 0,0,40,25 : REM RECHTS OHNE UMLAUF VERSCHIEBEN
240 PRINT AT(0, 9) MID$(AS$,X,1) : REM SCHRIFT EINFAEDELN
250 REM MID$(AS$,X,1) SCHREIBT DEN X-TEN BUCHSTABEN DES
STRINGS AS$
260 PRINT AT(0,11) ">"
270 PRINT AT(0,12) "<"
280 PRINT AT(0,13) ">" : REM ZEICHENEINSCHUB
290 NEXT X : REM 40 MAL

```

In diesem schönen Beispiel wird gezeigt, wie Sie in ein Scrolling Text einfädeln können. Sie können also z.B. auch Laufschriften generieren, die sehr viel länger sind als nur eine Bildschirmzeile. In obigem Programm wird das Einfädeln von Text dadurch erreicht, daß nach jedem rechts-Verschieben ein Buchstabe des zu schreibenden Textes an die entsprechende Zeilenposition vor den gerade verschobenen Textteil gesetzt wird. In unserem Beispiel wird der in einem Stringspeicher gelagerte Text Buchstabe für Buchstabe nachgeschoben. Dies wird durch den MID\$-Befehl des originalen Basic erreicht. Dieser Befehl (s. CBM 64-Handbuch) holt bekanntlich einen Teil mitten aus dem vollständigen String heraus. Der zu behandelnde String oder Stringspeicher (str\$) steht an erster Stelle der dem Befehl folgenden Klammer (Befehlssyntax: MID\$(str\$,a,l)). Der Anfang des herauszuholenden Teils wird durch a definiert (Startposition innerhalb des Strings), die Länge des Ausschnitts bestimmt l. Nähere Auskünfte erteilt Ihr Handbuch und das Kapitel 8 mit den Stringoperationen. Ein weiteres recht amüsantes Beispiel finden Sie unter dem Befehl DOWN, das sich ebenfalls der Technik des Einfädels bedient.



Nun aber zu einem kombinierten RIGHT-LEFT-Beispiel, das Ihnen ebenfalls schöne Effekt zeigt:

```
100 REM #####
110 REM ##                               ##
120 REM ## RIGHT/LEFT-BEISPIEL ##
130 REM ##                               ##
140 REM #####
150 REM
160 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
170 FOR X=0 TO 39
180 Y = 12 * SIN(X/3) + 12 : REM WERTE BERECHNEN
190 PRINT AT(X,Y) "*" : REM SINUSKURVE ZEICHNEN
200 NEXT X : REM 40 WERTE
210 FOR Y=1 TO 5
220 FOR X=1 TO 40
230 LEFTW 0, 0,20,25 : REM LINKER TEIL
240 RIGHTW 0,20,20,25 : REM RECHTER TEIL
250 NEXT X : REM 40 MAL GEGENEINANDER VERSCHIEBEN
260 FOR X=1 TO 40
270 RIGHTW 0, 0,20,25 : REM LINKER TEIL
280 LEFTW 0,20,20,25 : REM RECHTER TEIL
290 NEXT X : REM 40 MAL AUSEINANDER VERSCHIEBEN
300 NEXT Y : REM 5 * HIN UND HER
310 WAIT 198,255 : REM AUF TASTENDRUCK WARTEN
```

Eine Anmerkung zu diesem Beispiel: In Zeile 310 taucht ein neuer etwas undurchsichtiger Befehl auf. Dieser wird im CBM 64-Handbuch (deutsche Ausgabe auf Seiten 124/125) erläutert. Die weitere Erklärung würde hier den Rahmen sprengen, deshalb merken Sie sich bitte Folgendes: Der Ausdruck `WAIT 198,255` wartet auf einen Tastendruck Ihrerseits, der daraufhin mit `GET A$` abgefragt werden kann. Die Zeile

```
10 WAIT 198,255 : GET A$
```

ersetzt also die Zeile

```
10 GET A$ : IF A$ = "" THEN GOTO 10
```

Mit `POKE 198,0` kann ein zufällig im Tastaturbuffer stehendes Zeichen gelöscht werden (Die Speicherstelle 198 gibt die Anzahl der gedrückten Tasten, also die Anzahl der Zeichen wieder, die sich im Tastaturbuffer befinden (maximal 10)).

### 9.5.3 UP

```
FORMAT      :UPW Z.S.B.H
ODER        UPB Z.S.B.H
PARAMETER   : - Z: ZEILE (0-24) UND
              - S: SPALTE (0-39) DER OBEREN
                LINKEN ECKE DES FELDES
              - H: HOHE UND
              - B: BREITE DES FELDES
BEISPIEL    :UPW 10.10.5.6
ODER        UPB 10.10.5.6
FUNKTION    :AUFWÄRTS SCROLLEN EINES BILD-
              SCHIRMBEREICHES
```

#### ERLÄUTERUNGEN:

Was Ihnen anhand der 2 obigen Befehle erläutert wurde, sollte Ihnen den UP-Befehl eigentlich schon weitestgehendst erläutern. UP ermöglicht es, Teile des Bildschirms, die Sie in alt gewohnter Weise definieren, nach oben zu schieben. Dies natürlich wieder, wie alle Befehle dieses Abschnittes auf zwei verschiedenen Arten (s. Einleitung zu 9.5; W und B). Es bedarf eigentlich keiner weiteren Erläuterungen, wenn Sie sich bereits mit RIGHT und LEFT beschäftigt haben. Also wollen wir lediglich ein kleines Beispiel bringen.

#### BEISPIEL:

```
100 REM #####
110 REM ##          ##
120 REM ##  UP-BEISPIEL  ##
130 REM ##          ##
140 REM #####
150 REM
160 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
170 PRINT "MIT RUN/STOP"
180 PRINT "UNTERBRECHEN"
190 FOR X=0 TO 11
200 PRINT AT(X+15,X) CHR$(109) : REM 1. DIAGONALE ZEICHNEN
210 NEXT X
220 FOR X=12 TO 23
```

```
230 PRINT AT(23-X+15.X) CHR$(110) :REM 2. DIAGONALE ZEICHNEN
240 NEXT X
250 PAUSE 2
260 UPW 0.15.12.24 : REM NACH OBEN ROLLEN
270 GOTO 260
```

Es ist nicht schlimm, wenn Sie im obigen Beispiel die Errechnung der Cursorposition in dem AT(-Statement nicht verstehen. Für die Interessierten aber sei es ganz kurz erläutert: In der ersten Schleife (Zeilen 190 - 210) wird die erste, von links oben nach rechts unten gehende Diagonale Punkt für Punkt berechnet. Dabei entsteht der Spaltenwert (erster Wert in der AT-Klammer) einfach durch das von Schleifendurchlauf zu Schleifendurchlauf stetige Ansteigen des X-Wertes zuzüglich einer Konstanten 15, die die Punkte einfach um 15 Spalten in die Mitte des Bildschirms schiebt. Der Zeilenwert steigt lediglich mit den X-Werten an, d.h. in jede Zeile kommt genau ein Punkt.

Auf ähnliche Weise wird nun die zurücklaufende Diagonale (Zeilen 220 - 240) berechnet: hier allerdings fällt der Spaltenwert mit steigendem X, da mit steigendem X stetig mehr von 23 abgezogen wird. Der Rest ist analog aus dem oben Gesagten herzuleiten. Entwickeln Sie doch auch einmal solche Zusammenhänge!

## 9.5.4 DOWN

**FORMAT** : DOWNW Z.S.B.H  
**PARAMETER** : - Z: ZEILE (0-24) UND  
- S: SPALTE (0-39) DER OBEREN  
LINKEN ECKE DES FELDES  
- H: HOHE UND  
- B: BREITE DES FELDES  
**BEISPIEL** : DOWNW 1.0.10.15  
**FUNKTION** : ABWÄRTS SCROLLEN EINES BILD-  
SCHIRMBEREICHES

### ERLÄUTERUNGEN:

Nun sind wir bereits beim letzten Vertreter dieser Gruppe der "Bildschirmscroller" angelangt: DOWN. Sie können sich wahrscheinlich schon denken, wovon die Rede ist - es bleibt ja nur noch eine Möglichkeit offen: Das Abwärtsscrollen.

Machen wir es kurz: Wenn Sie nicht wissen, was Sie mit den Parametern anfangen sollen, so lesen Sie sich am besten die Erläuterung unter LEFT durch und fangen dann wieder hier an. DOWN schließt das Vierergespann der Scroller-Befehle und gibt Ihnen hiermit jede Form der Variation in die Hand.

Doch neben allem Optimismus doch ein kleiner Dämpfer. Wie Sie sich erinnern, tauchte beim RIGHT-Befehl ein kleines Problem auf (s.o.). Dort blitzten ab und zu verschiedene Zeichen am Blockrand auf. Ähnliches passiert auch bei DOWN. Am oberen Rand (dem Einschieberand) wird das Bild manchmal durch eine aufblinkende Reihe von Zeichen (z.B. Klammeraffen) gestört (s. Beispiel Motocross unten). Wie auch bei RIGHT sind diese nach der Beendigung des Befehls nicht mehr zu sehen. Man sollte nicht unbedingt auf das Programm schimpfen. Es ist durchaus möglich, daß dieser Effekt auch mit dem Videocontroller Ihres Rechners (also ein Hardwarefehler) zu tun hat. Nichtsdestotrotz zeigen wir ihnen nachfolgend ein außerordentlich interessantes Beispiel der Möglichkeiten des DOWN - Scroll - Kommandos. Beachten Sie die Kürze des Programms. Man könnte dieses Programm, wie auch viele andere Beispielprogramme, natürlich mit einigen Tricks auf ein paar Zeilen zusammenschweißen, doch das würde die Verständlichkeit beeinträchtigen. Es sei Ihnen überlassen, die Programme zu

optimieren und zu verändern. Das ist ja auch der Sinn der Beispiele.

**BEISPIEL:**

```
100 REM #####
110 REM ##          ##
120 REM ##  DOWN-BEISPIEL  ##
130 REM ##          ##
140 REM #####
150 REM
160 REM #####
170 REM ##          ##
180 REM ##  MOTOCROSS  ##
190 REM ##          ##
200 REM #####
210 REM
220 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
230 POKE 53248 + 33.13 : REM HINTERGRUNDFARBE = HELLGRUEN
(RASEN)
240 X = 10 : REM STRASSENPOSITION ANFANG
250 RV$ = CHR$( 18) : REM RVS ON
260 SC$ = CHR$(144) : REM SCHWARZ
270 GR$ = CHR$(152) : REM GRAU
280 ST$ = RV$ + SC$ + "!" + GR$ + "  !    " + SC$ + "!"
290 REM STRASSE: SCHWARZER RAND - GRAUER BELAG (10
LEERZEICHEN) - SCHWARZER RAND
300 X = X + RND(1)*2 - 1 : REM 0.1,-1 HINZUAEHLN FUER
ZUFAELLIGE KURVEN
310 IF X < 0 THEN X=0
320 IF X > 38-LEN(ST$) THEN X = 38-LEN(ST$)
330 REM ILLEGALE WERTE VERHINDERN
340 DOWNB 0.0.40.25 : REM STRASSE HINUNTERSCHIEBEN
350 PRINT AT(X,0) ST$ : REM STRASSENSTREIFEN ZEICHNEN
360 GOTO 300
```

Dieses Programm kann nur mit RUN/STOP angehalten werden.

### 9.5.5 TESTAUFGABEN

Zum Abschluß dieses Abschnittes selbstverständlich wieder unser Standardtest zur Wissenskontrolle:

1.) Sie wollen den Bereich von Zeilen 6 bis 10 und Spalten 5 bis 13 (jeweils einschließlich) um ein Zeichen nach oben schieben, ohne Bildumlauf. Welche Syntax wählen Sie?

- a) UPW 6,10,5,13
- b) UPD 6,10,5,13
- c) UP D 6,5,9,5
- d) UPD 6,5,9,5
- e) UPW 6,5,9,5

2.) Sie wollen den gesamten Bildschirms diagonal nach rechts unten verschieben (mit Bildumlauf), welche Befehlskombination wählen Sie?

- a) RIGHTW 0,0,40,25 : DOWNW 0,0,40,25
- b) LEFTW 0,0,40,25 : UPW 0,0,40,25
- c) ist nicht möglich
- d) DOWNW 0,0,40,25 : RIGHTW 0,0,40,25
- e) keine der hier angebotenen Möglichkeiten

# 10. KAPITEL

## EINGABEKONTROLLE

### 10.1 FETCH

**FORMAT:**     **FETCH "KONTR".LN.VAR**

**PARAMETER:** **KONTR - KONTROLLZEICHEN**  
              **LN    - MAXIMALLÄNGE DER EINGABE**  
                          **(DARF UNTER- ABER NICHT**  
                          **ÜBERSCHRITTEN WERDEN)**  
              **VAR   - BELIEBER VARIABLENTYP**

**FUNKTION:**   **KONTROLLIERTE EINGABE**

**BEISPIEL:**   **10 FETCH"12345".1.X\$**  
                  **ERWARTET DIE EINGABE EINER DER ZAHLEN**  
                  **'12345' UND SPEICHERT SIE IN X\$**

**BEMERKUNG:**   **DIE EINGABE MUß MIT RETURN ABGESCHLOSSEN**  
                  **WERDEN**

Mit FETCH steht Ihnen nun ein Befehl zur Verfügung, der alle Nachteile des INPUT-Befehls vergessen läßt. Die Kontrolle der Eingabe eines Programms ist wichtig, da der Benutzer sonst durch Falscheingaben das Programm zum Absturz bringen kann. Soll z.B. ein 5 stelliger Wert in einen String eingelesen werden, so ist der Befehl 'INPUT A' nicht ratsam. Zum einen kann der Benutzer mehr als 5 Zeichen eingeben, zum anderen kann er sogar durch Eingabe eines Buchstabens eine Fehlermeldung verursachen. Mit dem Befehl FETCH wird die Eingabe des 5-stelligen Wertes wie folgt durchgeführt:

**FETCH"0123456789".5.A\$**

Es werden nur die Zeichen 0 bis 9 akzeptiert und in A\$ gespeichert.

Anstatt der tolerierten Zeichen können auch Kontrollzeichen angegeben werden, die dann einen bestimmten Bereich abdecken:

CHR\$(19) - Nur Großbuchstaben (CHR\$(65) bis CHR\$(90))  
CHR\$(17) - Zahlen und Sonderzeichen (CHR\$(32) bis CHR\$(64))  
CHR\$(29) - nur Großbuchstaben mit und ohne SHIFT

Soll z.B. ein 12-stelliger String (A\$) eingelesen werden, der nur Großbuchstaben enthalten darf, so gilt folgender FETCH-Befehl:

***FETCH CHR\$(19), 12, A\$***

Während der Eingabe kann das letzte Zeichen mit der Taste INST/DEL gelöscht werden. Sind jedoch alle vorgesehenen Zeichen eingegeben worden, so wird nur noch RETURN akzeptiert und INST/DEL blockiert.

Ein weiteres Beispiel der Anwendungsmöglichkeiten von FETCH: Sehr häufig kommt es vor, daß in einem Programm Ja/Nein abgefragt wird. Mit dem herkömmlichen GET-Befehl wurde dies bisher wie folgt gelöst:

```
10 PRINT "PROGRAMM BEENDEN (J/N)?"  
20 GET X$:IF X$<>"J" AND X$<>"N" THEN 20  
30 IF X$="J" THEN END  
40 .....
```

Mit dem Befehl FETCH läßt sich dieses Problem viel eleganter lösen:

```
10 PRINT "PROGRAMM BEENDEN (J/N)?"  
20 FETCH "JN", 1, X$  
30 IF X$="J" THEN END  
40 .....
```

Der Unterschied zwischen diesen beiden Möglichkeiten ist, daß der Eingabe eines Zeichens mit FETCH ein RETURN folgen muß.



## 10.2 INKEY

**FORMAT:** INKEY  
**PARAMETER:** KEINE  
**FUNKTION:** ABFRAGE NACH BETÄTIGTER FUNKTIONSTASTE  
**BEISPIEL:** A=INKEY  
SPEICHERT DIE BETÄTIGTE FUNKTIONSTASTE  
IN DER VARIABLEN A

Mit INKEY kann die Nummer der betätigten Funktionstasten ermittelt werden (1-16). Da der Commodore 64 standardmäßig mit 8 Funktionstasten versehen ist, werden die restlichen Tasten mit der Commodore-Taste angesteuert (näheres siehe Kapitel 2).

INKEY speichert also die Nummer der betätigten Funktionstaste in eine vorher bestimmte Variable. Wird keine Taste gedrückt, so wird der Variablen der Wert 0 zugewiesen. Das folgende Programm wartet auf die Betätigung einer Funktionstaste und zeigt Sie dann an:

```
10 X=INKEY:IF X=0 THEN 10
20 PRINT "FUNKTIONSTASTE NR."X
30 GOTO 10
```

Dieses Programm können Sie nur mit der Taste 'RUN/STOP' verlassen. Es entfällt die im Standard-BASIC übliche Abfrage auf den ASCII-Wert. Da ohnehin nur den normalen 8 Funktionstasten ein ASCII-Wert zugeteilt wurde (133-140), können die 8 zusätzlichen Tasten nur mit INKEY abgefragt werden.

Die Abfrage nach einer betätigten Funktionstaste ist z.B. bei einer Menüsteuerung notwendig, die wie folgt aufgebaut ist:

**WAHLEN SIE DIE GEWUNSCHTE FUNKTION:**

-----  
F1 - ADRESSEN EINGEBEN  
F2 - ADRESSEN LOESCHEN  
F3 - ADRESSEN ANZEIGEN  
F4 - ADRESSEN LADEN  
F5 - ADRESSEN SICHERN

Die Zeilen, die diese Auswahl steuern sehen dann z.B. folgendermaßen aus:

```
100 A=INKEY:IF A=0 THEN 100
110 CGOTO (A*10 + 110)
120 CALL EINGEBEN
130 CALL LOESCHEN
140 CALL ANZEIGEN
150 CALL LADEN
160 CALL SICHERN
```

Weiterhin können der Anwender eines Programms mit Hilfe der Funktionstasten entscheidungen treffen wie z.B.:

### **AUSGABE AUF DRUCKER (F1) ODER BILDSCHIRM (F3)**

Die steuernden Programmzeilen hierzu sind:

```
100 A=INKEY:IF A=0 THEN 100
110 IF A=1 THEN CALL DRUCKER
```

### **10.3 RESET**

<b>FORMAT:</b>	<b>RESET ZN</b>
<b>PARAMETER:</b>	<b>ZN - NUMMER DER DATA-ZEILE, AUF DIE DER DATA-ZEIGER GESETZT WERDEN SOLL</b>
<b>FUNKTION:</b>	<b>SETZEN DES DATA-ZEIGERS AUF DIE MIT READ ZU LESENDEN DATEN</b>
<b>BEISPIEL:</b>	<b>RESET 100 DIE DATA-ZEILEN AB 100 WERDEN MIT DEM NÄCHSTEN READ GELESEN</b>

Zum Speichern einer großen Anzahl von Konstanten benutzt man in BASIC DATA-Zeilen und den Befehl READ. Nach jedem Lesen eines DATA-Wertes wird der Zeiger auf den nächsten DATA-Wert gesetzt, sodaß ein weiterer READ den zweiten Wert liest, usw. Versucht man jedoch, mehr DATA-Werte zu lesen, als im Programm enthalten, so erscheint die Fehlermeldung "OUT OF

DATA ERROR". Mit dem Befehl RESTORE des Standard-BASIC wird wieder auf den ersten DATA-Wert eines Programmes positioniert.

Der folgende Programmausschnitt soll weitere Erklärungen unterstützen:

```
200 DATA EDGAR.MANFRED.KARL.OTTO.WILHELM.*
300 DATA 0.1.2.3.4.5.6.7.8.9.0.A.B.C.O.E.F.*
400 DATA 102.103.105.108.123.120.142.189.*
500 DATA IRLAND. ENGLAND. DEUTSCHLAND. FRANKREICH.*
```

Dies sind DATA-Zeilen mit voneinander abweichenden Inhalten. Diese Zeilen werden von einem Programm gelesen und in Variablen abgespeichert. Sollen aber nun mit dem Standard-BASIC an einer bestimmten Stelle des Programms die Länder eingelesen werden, so muß mit RESTORE wieder auf den DATA-Anfang positioniert werden. Dann müssen die ersten DATA-Werte überlesen werden. Deren Anzahl muß also bekannt sein.

Mit SIMON'S BASIC ist das nun viel einfacher: Sie geben lediglich den Befehl RESET 130 ein, und der DATA-Zeiger wird auf den ersten Wert in Zeile 130 positioniert. Ist die mit RESET angegebene Zeile nicht vorhanden oder nicht mit DATA-Werten belegt, so wird automatisch auf die folgende Zeile positioniert.

Ein Beispiel zum individuellen Positionieren mit RESET:

```
10 PRINT "BITTE WAEHLN SIE:"
20 PRINT "-----"
30 PRINT "1 - NAMEN"
40 PRINT "2 - HEX-ZIFFERN"
50 PRINT "3 - ZAHLEN"
60 PRINT "4 - LÄNDER"
70 FETCH"1234".1.A$
80 IF A$="1" THEN RESET 200
90 IF A$="2" THEN RESET 300
100 IF A$="3" THEN RESET 400
110 IF A$="4" THEN RESET 500
120 READ A$
130 IF A$="*" THEN END
140 PRINT A$
150 GOTO 120
```

Geben Sie nun noch die zuvor aufgeführten DATA-Zeilen ein. Das Programm liefert Ihnen nach dem Start die gewünschten Daten.

### 10.3 TESTAUFGABEN

1) Welche Zeichenfolge wurde nicht mit dem Befehl  
`FETCH"0123456789ABCDEF\$",3,A\$` eingelesen?

- A) \$87
  - B) \$FF
  - C) A\$A
  - D) C \$
- 

2) Welche Zeichen werden von dem folgenden Befehl akzeptiert?

FETCH CHR\$(19),3,X\$

- A) nur Ziffern und Sonderzeichen
  - B) nur Großbuchstaben mit und ohne SHIFT
  - C) nur Großbuchstaben
- 

3) Wieviel Funktionstasten können mit dem Befehl INKEY kontrolliert werden?

- A) 8
  - B) 16
  - C) 15
  - D) 7
- 

4) Welcher Befehl kann den DATA-Zeiger auf beliebige Zeilen positionieren?

- A) RESTORE
  - B) RESET
  - C) CRESTORE
-

## 11. KAPITEL EIN/AUSGABE BEFEHLE

### 11.1 DISKETTENBEFEHLE

#### 11.1.1 DISK

**FORMAT:** DISK "ANWEISUNG"  
**PARAMETER:** ANWEISUNG - FLOPPY-SYSTEMBEFEHL  
**FUNKTION:** VEREINFACHTE KOMMUNIKATION MIT DEM FLOPPY-LAUFWERK  
**BEISPIEL:** DISK "N:TESTDISK.MN"  
FORMATIERT EINE DISKETTE

Der übliche Umweg zur Übertragung eines Floppy-Systembefehls entfällt nun. Bisher mußten Sie z.B. zum Formatieren einer Diskette die folgenden Befehle eingeben:

```
OPEN1.8.15."N:TESTDISK.MN"  
CLOSE1
```

Mit dem DISK-Befehl ist dies nun wesentlich einfacher. Der Floppy-Befehlskanal braucht nicht mehr geöffnet und geschlossen zu werden. Diese Aufgabe übernimmt der Befehl DISK. Für alle diejenigen, die nicht so sehr mit den Floppy-Befehlen vertraut sind, folgt nun eine Beschreibung der wichtigsten Befehle:

#### **NEW - DISKETTE FORMATIEREN**

-----  
**FORMAT:** DISK "N:name,id"  
**PARAMETER:** name - Name der Diskette (max. 16 Zeichen)  
id - Kennzeichen der Diskette (2 Zeichen)  
**BEISPIEL:** DISK "N:SIMON'S BASIC,01"

## **SCRATCH - FILES LÖSCHEN**

-----  
FORMAT:       DISK "S:filename1, filename2,....."  
PARAMETER:   filename1 - 1. zu löschendes File  
              filename2 - 2. zu löschendes File  
BEISPIEL:     DISK "S:TESTPROGRAMM"

## **RENAME - FILES UMBENENNEN**

-----  
FORMAT:       DISK "R:neufilename=altfilename"  
PARAMETER:   neufilename - neuer Filename  
              altfilename - bisheriger Filename  
BEISPIEL:     DISK "R:INFO=INFO.TEST"

## **INITIALISE - INITIALISIEREN DER DISKETTE**

-----  
FORMAT:       DISK "I"  
PARAMETER:   keine

## **VALIDATE - ÜBERFLÜSSIGE BLÖCKE LÖSCHEN**

-----  
FORMAT:       DISK "V"  
PARAMETER:   keine

Allen, die mehr von dem vielseitigem Floppylaufwerk wissen möchten, empfehlen wir das DATA-BECKER BUCH "Das große Floppbuch".

## 11.1.2 DIR

**FORMAT:** DIR "\$"  
DIR "\$:STRING"  
DIR "\$:??STRING"  
DIR "\$:\*=FT"

**PARAMETER:** STRING - FILENAME ODER TEIL EINES FILE-  
NAMENS  
? \* - "JOKER": SIEHE TEXT  
FT - FILETYP (P=PRG, S=SEQ, R=REL,  
U=USR)

**FUNKTION:** AUFLISTEN DES DISKETTENINHALTES, OHNE DAB  
EIN BASIC-PROGRAMM GELOSCHT WIRD

**BEISPIEL:** DIR "\$"  
ZEIGT DEN GESAMTEN DISKETTENINHALT AN

Dieser Befehl ist, wie Sie sehen, sehr vielseitig. Vier verschiedene Schreibarten, die sogar noch gemischt werden können. Das erste Format ist das einfachste. Zum Anzeigen des Directorys der Diskette benutzen Sie den Befehl

### DIR "\$"

Entgegen dem Standard-BASIC bleibt hier das BASIC-Programm erhalten. Doch nun zu den etwas verzwickten Formaten.

Mit einem Stern kann man das Directory selektiert auflisten. Geben Sie z.B. den Befehl 'DIR "\$:A\*"' an, so werden alle Files der Diskette angezeigt, die mit dem Buchstaben A beginnen. Der Stern ignoriert also den Rest des Filenamens. Noch ein Beispiel: 'DIR "\$:TEST\*"' zeigt alle Files, die mit den Zeichen 'TEST' beginnen. (TEST.1, TEST.NEU, TESTPROGRAMM usw.).

Das Fragezeichen ersetzt die Stellen des Filenamens, die zum Auflisten ignoriert werden sollen. Sie dürfen auch vorne oder mitten im Filenamens enthalten sein. Ein Beispiel: 'DIR "\$:???01"' zeigt alle 6-stellige Filenamens an, deren letzten drei Zeichen '.01' sind. Oder 'DIR "\$:TEST.??"' zeigt alle 7-stellige Filenamens an, deren ersten fünf Zeichen 'TEST.' sind.

Nun kann noch das Fragezeichen mit dem Stern kombiniert

werden. So zeigt z.B. der Befehl 'DIR "\$:???01\*"' alle  
Dateinamen der Diskette, deren 4., 5. und 6. Zeichen '.01'  
entsprechen.

Weiterhin können die Files auch Ihrem Typ entsprechend  
angezeigt werden. 'DIR \$:\*=P"' z.B. gibt alle Programme aus,  
die sich auf der eingelegten Diskette befinden. Auch dieses  
Format kann beliebig mit anderen Formaten kombiniert werden.  
Die Frage, was der Befehl 'DIR"\$:???18\*=P' nun ausgibt,  
überlasse ich denjenigen Lesern, die mir bis zu dieser Stelle  
ohne Probleme folgen konnten.

Probieren geht über jedes Studieren. Testen Sie diese  
verschiedenen Schreibweisen des DIR-Befehls daher solange  
aus, bis Sie Ihnen eindeutig klar sind.

Den Befehl DIR können Sie durchaus auch in einem Programm  
benutzen. Sinnvoll ist es dann, vorher den Bildschirm zu  
löschen und das Programm nach Ausgabe des Directorys "auf  
Tastendruck" fortzusetzen. Das folgende Programm ist ein  
Beispiel zur Lösung dieses Problems:

```
100 PRINT "DIRECTORY ANZEIGEN (J/N)?"
110 REPEAT
120 ::GET X$
130 UNTIL X$="J" OR X$="N"
140 IF X$="N" THEN CALL WEITER
150 ::PRINT CHR$(147):REM BILDSCHIRM LOESCHEN
160 ::DIR:REM DIRECTORY AUFLISTEN
170 ::PRINT"WEITER MIT RETURN"
180 ::REPEAT
190 ::::GET X$
200 ::UNTIL X$=CHR$(13)
210 PROC WEITER
```

Interessant ist die Tatsache, daß ein bereits vorhandenes  
Programm wie beim normalen 'LOAD"\$",8' gelöscht wird. Wie ist  
dies zu erklären? Da der Befehl 'LOAD' ohne Sekundäradresse 1  
grundsätzlich ab dem Anfang des BASIC-Programmes (\$0801)  
lädt, wird das vorhandene BASIC-Programm natürlich gelöscht.  
Der Befehl DIR jedoch läßt das Directory wie eine Datei Byte  
für Byte und stellt sie auf dem Bildschirm dar. Das  
eingeleseene Directory wird also nicht im Hauptspeicher des  
CBM 64 abgelegt, sondern direkt auf dem Bildschirm angezeigt.



### 11.1.3 SCRSV

**FORMAT:** SCRSV LFN.GA.SA."NAME.S.W"  
**PARAMETER:** LFN - LOGISCHE FILENUMMER  
GA - GERÄTEADRESSE (1=KASS. 0=FLOPPY)  
SA - SEKUNDÄRADRESSE  
NAME - DATEINAME DES BILDSCHIRMS  
**FUNKTION:** ABSPEICHERN EINES LOW-RESOLUTION-  
BILDSCHIRMS  
**BEISPIEL:** SCRSV 1.8.2."TEST.SCRSV.S.W"

Der LO-RESOLUTION-Bildschirm, also der Bildschirm, der nach dem Einschalten aktiv ist, kann mit diesem Befehl auf Diskette oder Kassette abgespeichert werden. Es kann sowohl die niedrig auflösende Graphik, als auch der normale Text-Bildschirm mit dem Befehl SCRSV auf Diskette als sequentielle Datei gespeichert werden. Dies ist z.B. beim Arbeiten mit Bildschirmmasken sinnvoll. Bildschirmmasken sind eine Art Karteikarte, die mit der Tastatur gefüllt werden muß, also eine Technik zur Datenerfassung. Sie brauchen diese Bildschirmmaske nicht immer wieder im Programm aufzubauen, sondern nur einmalig und dann speichern. Bei Bedarf wird die Bildschirmmaske dann wieder geladen und auf dem Bildschirm sichtbar.

Die Parameter dieses Befehls entsprechen den Parametern zum Öffnen einer sequentiellen Datei. D.h. es darf z.B. kein File mit der entsprechenden logischen Filenummer bereits offen sein, da sonst die Fehlermeldung "FILE OPEN ERROR" ausgegeben wird. Nun ein Beispiel:

```
100 REM *****  
110 REM      BEISPIEL SCRSV  
120 REM *****  
130 PRINT CHR$(147);  
140 PRINT "NAME      : ..... "  
150 PRINT "VORNAME   : ..... "  
160 PRINT "STRASSE    : ..... "  
170 PRINT "PLZ/ORT    : ..... "  
180 PRINT "TELEFON    : ..... "  
190 SCRSV 1.8.2."TEST.SCRSV.S.W"  
200 GOTO 200
```

Der Befehl SCRSV kann sowohl im Programm- als auch im Direkt-Modus angewendet werden. Im Direkt-Modus jedoch wird dann natürlich auch der eingegebene SCRSV-Befehl, der sich noch auf dem Bildschirm befindet, als Bestandteil des Text-Bildschirms abgespeichert.

Die Speicherung des LOW-RESOLUTION Bildschirms ist ein zweckmäßiger Befehl um wichtige Bildschirminformationen "mal eben" festzuhalten. Bedauerndwert ist die Tatsache, daß der HIGH-RESOLUTION Bildschirm nicht gespeichert werden kann. Diese Funktion, die SUPERGRAPHIK 64 z.B. ermöglicht hat folgende Vorteile:

1. Eine Grafik, deren Erstellung u.U. sehr viel Zeit in Anspruch nehmen kann (z.B. eine dreidimensionale Grafik mit umfangreichen mathematischen Berechnungen) kann in wenigen Sekunden von der Diskette geladen und auf dem Bildschirm angezeigt werden.
2. Die abgespeicherten HIGH-RESOLUTION Grafiken der verschiedensten Grafik-Programme für den CBM 64 sind untereinander kompatibel.

#### 11.1.4 SCRLD

**FORMAT:** SCRLD LFN.GA.SA."NAME"  
**PARAMETER:** LFN - LOGISCHE FILENUMMER  
GA - GERÄTEADRESSE (1=KASS 2=DISK)  
SA - SEKUNDÄRADRESSE  
NAME - DATEINAME DES BILDSCHIRMS  
**FUNKTION:** LADEN DES MIT SCRSV GESPEICHERTEN LOW-  
RESOLUTION-BILDSCHIRMS  
**BEISPIEL:** SCRLD 1.8.2."TEST.SCRLD"  
LÄDT DEN BILDSCHIRM MIT DEM NAMEN  
"TEST.SCRLD"

Ein Bildschirminhalt, der mit SCRSV gespeichert wurde, wird nun wieder geladen. Der Bildschirm enthält die Daten, die vor dem Speichern vorhanden waren. Sie haben also nun die Möglichkeit, bestimmte Bildschirminhalte mit einem externen Programm zu erstellen und auf der Diskette (oder Kassette) abzulegen. Da der Ladevorgang von Diskette relativ schnell verläuft, erhalten Sie im endgültigen Programm die gewünschten Masken mit einem Befehl.

Das Programm zum Laden des zuvor gespeicherten Bildschirms sieht folgendermaßen aus:

```
100 REM *****  
110 REM          BEISPIEL SCRLD  
120 REM *****  
130 SCRLD 1.8.2."TEST.SCRSV"  
140 GOTO 140
```

Dieses Programm verlassen Sie mit der Taste 'RUN/STOP'. Natürlich können Sie diesen Bildschirm auch laden, indem Sie den SVRSC-Befehl im Direkt-Modus eingeben. Der entsprechende Befehl ist dann:

```
SCRLD"1.8.2."TEST.SCRSV"
```

### 11.1.5 TESTAUFGABEN

1) Die Datei "TEST.SCRSV" soll gelöscht werden. Welcher Befehl ist richtig?

- A) DISK "S:TEST.SCRSV"
  - B) DISK "N:TEST.SCRSV"
  - C) DISK "R:TEST.SCRSV"
- 

2) Welches FILE wird nicht von dem folgenden Befehl angezeigt?

DIR "\$:????.TE"

- A) VARB.TEST
  - B) KAO1.TE
  - C) SCRL.TE
- 

3) Welches File wird nicht von dem folgenden Befehl angezeigt?

DIR "\$:??FILE\*"

- A) MNFILES
  - B) MAFILE.PROGRAMM
  - C) FILEMANAGER
- 

4) Ein LOW-RESOLUTION-Bildschirm soll auf Diskette gespeichert werden. Welcher Befehl ist richtig?

- A) SCRSAVE 1,8,2,"LOWRES01,S,W"
  - B) SCRSV 8,8,3,"LOWRES01",S,W
  - C) SCRSV 2,8,2,"LOWRES01"
  - D) SCRSV 5,8,3,"LOWRES01,S,W"
-

## 11.2 DRUCKERBEFEHLE

### 11.2.1 COPY

**FORMAT:** COPY  
**PARAMETER:** KEINE  
**FUNKTION:** AUSGEBEN DER HARDCOPY EINER HI-RES ODER  
MULTI-COLOUR-GRAFIK

Der Befehl COPY öffnet und schließt selbständig den Druckerkanal. Ein OPEN 1,4 vor dem Befehl ist also nicht erforderlich.

Die Hardcopy wird auf folgenden Druckern ausgegeben:

- **CBM VC-1525**
- **SEIKOSHA GP 100 VC**
- **EPSON RX-80, FX-80 MIT DATA-BECKER-INTERFACE**
- **CBM MPS-801**
- **CBM MPS-803**

Da der Graphikdruck der Drucker CBM VC-1526 und MPS-802 vollkommen anders organisiert ist (8-Nadeldruck), ist der Befehl COPY hier nicht einsetzbar. Eine spezielle Hardcopy-Routine für diese beiden Drucker ist jedoch in dem DATA BECKER Programm "SUPERGRAPHIK 64" enthalten.

Diesen Befehl können Sie sowohl im Programm- als auch im Direkt-Modus verwenden. Wenn Sie z.B. nach der Erstellung einer Grafik mit Hilfe eines Programms anschließend ein Hardcopy ausgeben möchten, so hängen Sie diesem Programm die Zeile mit dem Befehl COPY an.

Sie können aber auch nach Ablauf des Programms, wenn der Rechner sich mit 'READY' meldet, den Befehl COPY einsetzen, obwohl der LOW-RESOLUTION- also Textmodus aktiv ist. Die Grafikseite muß also nicht sichtbar sein, um Sie als Hardcopy auf dem Drucker auszugeben.

## 11.2.2 HRDCPY

**FORMAT:** HRDCPY  
**PARAMETER:** KEINE  
**FUNKTION:** AUSGEBEN DER HARDCOPY EINES LOW-RESOLUTION-BILDSCHIRMS

Mit diesem Befehl können Sie den normalen Textbildschirm, den Sie auch auf Diskette speichern können, auf dem Drucker ausgeben. In manchen Programmen ist es sinnvoll, "auf Knopfdruck" einen Ausdruck vom momentanen Bildschirminhalt zu erstellen. Der dafür zuständige Programmausschnitt könnte etwa so aussehen:

```
100 PRINT "HARDCOPY GEWUENSCHT (J/N)?"  
110 FETCH "JN".1.X$  
120 IF X$="J" THEN HRDCPY  
130 .....
```

Natürlich können Sie den Befehl auch im Direkt-Modus (also im Außer-Programm-Betrieb) eingeben.

Dieser Befehl ist auch mit dem neuen CBM Drucker VC-1526 einsetzbar.

### 11.2.3 TESTAUFGABEN

1) Welche Ausgabe übernimmt der Befehl COPY?

- A) HIGH-RESOLUTION
  - B) MULTI-COLOUR
  - C) LOW-RESOLUTION
- 

2) Welche Ausgabe übernimmt der Befehl HRDCPY?

- A) HIGH-RESOLUTION
  - B) MULTI-COLOUR
  - C) LOW-RESOLUTION
- 

3) Ist es erforderlich, vor Anwendung der Befehle COPY und HRDCPY den Druckerkanal zu öffnen (OPEN1,4)?

- A) ja
  - B) nein
-

## 12. KAPITEL GRAPHIK

**WIR SIND IM BEGRIFF ZU DEM HÖHEPUNKT DES PROGRAMMIERENS. DEM  
TRAUM JEDES POTENTIELLEN UND TATSÄCHLICHEN COMPUTERBESITZERS.  
KURZ DEM IDEAL DER PROGRAMMIERKUNST ZU STOßEN: DER  
GRAPHIK !**

Erst durch die Fähigkeit der Graphikgestaltung wird der Computer zum Computer, der Fähigkeit, Informationen so ansprechend und naturgetreu wie möglich zu vermitteln, der Fähigkeit, auch dem nicht Eingeweihten die ungeheure Vielfalt und den Nutzen der modernen Informationsverarbeitung und nicht zu vergessen, der Freizeitgestaltung zu vermitteln. Zusammen mit dem akustischen Schauspiel, der Synthesizer - Klangberauschung liegt hier das Hauptaugenmerk der Computerkunst. Nicht umsonst haben die Techniker bei der Konzeption Ihres Commodore 64 keine Mühe gescheut, um diesem Spitzengerät all diese Möglichkeiten zu eröffnen. Es liegt an den Softwarespezialisten, diese Möglichkeiten leicht zugänglich zu gestalten. Aus diesem Grunde durfte die Graphik selbstverständlich in Ihrem Simon's Basic nicht fehlen, auch wenn es die ungeheuren Möglichkeiten natürlich zwangsweise (Simon's Basic soll ja eine allgemeine Basic-Erweiterung darstellen) nicht vollständig ausnutzt. Wollen Sie sich intensiv mit der Graphik und den Soundmöglichkeiten Ihres Rechners beschäftigen, so sollten Sie sich eine spezielle Graphikerweiterung zulegen, die es schon für recht wenig Geld zu haben gibt (ich kann hier nur die Supergraphik 64 mit ihren insgesamt 183 Befehlen und Befehlskombinationen, die genau auf Ihre Graphik- und Soundbedürfnisse ausgerichtet ist, empfehlen - mehr sage ich nicht).

Doch nun zu der Besprechung der Graphikbefehle unseres Simon's Basic. Bevor wir mit der genauen Darlegung der einzelnen Befehle beginnen, müssen wir uns zum besseren Verständnis und aus dem Grunde einer einheitlichen Terminologie zunächst einmal die Grundlagen und die Graphikmöglichkeiten Ihres CBM 64 vor Augen führen. Dabei werden hier nur die Möglichkeiten dargelegt, die auch vom Simon's Basic ausgeschöpft werden. Wollen Sie sich genauer mit der Graphikbearbeitung Ihres Rechners beschäftigen, so



gibt es hierzu einschlägige Literatur (z.B. 64 intern etc.).  
Nun denn, auf ins Gefecht!

## **12.1 HARDWAREVORAUSSETZUNGEN**

Hier sollen Ihnen nun einige Grundkenntnisse der Realisierung der Graphik in Ihrem Computer vermittelt werden. Sprites, Zeichensatz und Synthesizer werden in späteren Kapiteln abgehandelt.

### **FARBEN:**

Ihr Commodore 64 verfügt über die Möglichkeit, 16 Farben sowohl im Graphikbetrieb als auch (wie sicher schon bekannt) für die Textgestaltung zu verwenden. Diese 16 Farben besitzen jeweils einen sogenannten Farbcode, der als Binärzahl in die verschiedenen Register gespeichert wird, die dem Gerät zur Zeichendarstellung verhelfen. Wird z.B. in das 32. Register des Video Interface Chips der Wert 0 gePOKEt, so nimmt der äußere Bildschirmrahmen die Farbe schwarz an. Im Folgenden sind die den einzelnen Farben zugeordneten Codes aufgelistet:

<b>0 - SCHWARZ</b>	<b>8 - ORANGE</b>
<b>1 - WEIß</b>	<b>9 - BRAUN</b>
<b>2 - ROT</b>	<b>10 - HELLROT</b>
<b>3 - TÜRKIS</b>	<b>11 - GRAU 1</b>
<b>4 - VIOLETT</b>	<b>12 - GRAU 2</b>
<b>5 - GRÜN</b>	<b>13 - HELLGRÜN</b>
<b>6 - BLAU</b>	<b>14 - HELLBLAU</b>
<b>7 - GELB</b>	<b>15 - GRAU 3</b>

### **HOCHAUFLOSENDE GRAPHIK:**

Ihr Rechner hat die Möglichkeit, von Haus aus zwei verschiedene Graphikarten zu bedienen: Die Hochauflösende Graphik (HGR) und die Multicolorgraphik (MC).

Erstere bietet ein Graphikfeld von 320 Punkten in x-Richtung und 200 Punkten in y-Richtung (320x200). Dies verschafft Ihnen ein Reservoir von insgesamt 64000 Punkten jeweils in

gleicher Dichte verteilt auf Ihrem Bildschirmfenster. Natürlich muß die Graphik genauso wie der Text (s. Kapitel 9) gespeichert sein. Dies geschieht in dem sogenannten Graphikspeicher. Jeder Punkt ist einzeln ansprechbar und ist in diesem Graphikspeicher durch ein Bit repräsentiert. Ein Bit ist eine Informationseinheit und kann die Werte 1 oder 0 annehmen. Jeweils 8 Bits hintereinander bezeichnet man als ein Zeichen (Wort) bzw. als ein Byte. Ein Byte kann also  $2 \text{ hoch } 8 = 256$  verschiedene Werte annehmen. Diese kommen durch die verschiedenen Kombinationen von gesetzten (1) und nicht gesetzten (0) Bits zustande. Ein Byte repräsentiert also 8 Punkte auf dem Bildschirm. Folglich bedarf es  $64000/8 = 8000$  Bytes (etwa 8 Kilobyte (8 K)), um den gesamten Bildschirminhalt der HGR zu speichern. Diese 8 K werden im Simon's Basic platzsparend in dem RAM unter dem ROM des Betriebssystems von hexadezimal \$E000 bis \$FF40 (oder dezimal von Speicherstelle 57344 bis 65344) abgelegt. Wenn Sie also direkt in den Bildschirmspeicher POKEn wollen, so können Sie sich dieses Speicherbereiches bedienen.

Um derartige direkte Manipulationen vornehmen zu können, müssen Sie wissen, wie nun der Aufbau des Graphikbildes aus den Speicherinformationen vonstatten geht:

Grundlage des Bildschirmspeichers ist eine 8x8-Matrix, die jeweils durch 8 Bytes des Graphikspeichers dargestellt wird. Diese gleicht der Matrix der Zeichendarstellung für den normalen Textbetrieb (s. Schema unten). Da eine solche Matrix also 8 Punkte hoch und breit ist, passen insgesamt 40 solcher Päckchen in eine Zeile und 25 in eine Spalte. Dies ist sich folgendermaßen vorzustellen:

<b>BIT</b>	<b>7 6 5 4 3 2 1 0</b>	<b>7 6 5 4 3 2 1 0</b>	<b>....</b>
<b>BYTE 0</b>	. . . . .	. . . . .	
<b>BYTE 1</b>	. . . . .	. . . . .	
<b>BYTE 2</b>	. . . . .	. . . . .	
<b>BYTE 3</b>	. . . . .	. . . . .	
<b>BYTE 4</b>	. . . . .	. . . . .	
<b>BYTE 5</b>	. . . . .	. . . . .	
<b>BYTE 6</b>	. . . . .	. . . . .	
<b>BYTE 7</b>	. . . . .	. . . . .	
<b>BYTE 320</b>	. . . . .	. . . . .	
<b>BYTE 321</b>	. . . . .	. . . . .	
<b>BYTE 322</b>	. . . . .	. . . . .	
<b>...</b>			

Es wird also Zeile für Zeile aus diesen Päckchen generiert. Wichtig ist diese Anordnung gleichfalls für die Farbgebung:

Sie haben 16 Farben zur Auswahl, mit denen Sie den Hintergrund (nicht gesetzte Punkte) und auch die gesetzten Punkte darstellen können. Da aber nicht eine derartige Masse an Speicherplatz zur Verfügung steht, um jedem der 64000 Punkte eine eigene Farbe zu geben (Sie bräuchten dafür  $64.000 \cdot 4 = 256.000$  Bits, also 32 K RAM), wurde stets ein 8x8-Päckchen als eine Farbeinheit gewählt. D.h. Sie können jeweils für ein solches Päckchen die Farbe für alle nicht gesetzten (Hintergrundfarbe) und alle gesetzten Punkte bestimmen. Hierfür ist lediglich ein 1 K - Speicher notwendig, der sogenannte Videoram. Dieser liegt, ebenfalls keinen Basic-Speicherplatz verbrauchend, bei \$C000 - \$C3E7 (dezimal: 49152 - 50151). Er ist genauso organisiert, wie der Bildschirmspeicher der Textausgabe. In jedem Byte dieses Videorams wird in den oberen 4 Bits die Punktefarbe und in den unteren 4 Bits die Hintergrundfarbe eines 8x8-Blocks gespeichert.

Wenn Sie von Basic aus einen bestimmten Punkt ansteuern wollen, sei es, Sie wollen ihn setzen (was schneller mit den entsprechenden Befehlen des Simon's Basic geht) oder nur testen, ob er gesetzt ist, dann bedienen Sie sich am besten der folgenden Formel zur Adressenberechnung:

```

10 REM X UND Y SIND DIE BEIDEN KDDRDINATEN DES PUNKTES
20 AD = 57344 : REM BASISADRESSE DES GRAPHIKSPEICHERS
30 DY = 320 * INT(Y/8) + (Y AND 7)
40 OX = 8 * INT(X/8)
50 MA = 2 ^ (7 - (X AND 7))
60 AV = AD + DY + OX
70 REM BEISPIEL: SETZEN EINES PUNKTES:
80 POKE AV, PEEK(AV) OR MA

```

Vielleicht versuchen Sie einmal, dieses Programm zu verstehen. Aber Sie können auch ohne diese Formel leben, da es ja bereits Befehle zum Setzen eines Punktes im Simon's Basic gibt. Als nächstes ein einfaches Beispiel des direkten EinPOKENS in den Graphikbildschirm zur Erzeugung eines Streifenmusters:

```

10 HIRES 2.1 : REM GRAPHIK EINSCHALTEN (S.U.)
20 FOR X=57344 TO 65344
30 POKE X, %11110000 : REM ERSTE 4 BITS EINES BYTES SETZEN
40 NEXT X
50 WAIT 198.255 : REM AUF TASTENDRUCK WARTEN

```

Sie sehen, wie langsam natürlich ein solches Basicprogramm ist.

### **MULTICOLORGRAPHIK:**

Wie Sie gesehen haben, besitzt ihr Rechner im HGR-Modus nur eine sehr beschränkte Farbauflösung. Um dieses speicherplatzbedingte Manko auszugleichen, wurde zusätzlich zum HGR-Modus der sogenannte Multicolormodus geschaffen. Er ermöglicht es, in einem 8x8-Block statt zwei, insgesamt 4 Farben gleichzeitig zu verwenden. Dies muß allerdings zwangsläufig auf Kosten der Punktauflösung gehen.

Im Multicolormodus stehen jeweils 2 Bit für einen doppelt breiten Punkt auf dem Bildschirm. Die Auflösung beträgt demnach 160 doppelt breite Punkte in x-Richtung und 200 Punkte in y-Richtung (160x200). Die beiden Bits des Graphikspeichers, die zusammen einen Punkt darstellen, geben nun die Nummer eines der 4 möglichen Farbregister eines Blockes an (unter Nummer ist hier nicht der Farbcode (0-15), sondern die Zuordnung des Registers gemeint, das den Farbcode

enthält). Dabei steht in jedem der vier Farbregister der eigentliche Farbcode des Punktes. Den 4 Farbregister sind die folgenden Speicher zugeordnet:

<b>FARBREGISTER</b>	<b>BIT-CODE</b>	<b>SPEICHER</b>
<b>0</b>	<b>00</b>	<b>REGISTER 33 DES VIC</b>
<b>1</b>	<b>01</b>	<b>UNTERE 4 BITS DES VIDEORAM</b>
<b>2</b>	<b>10</b>	<b>OBERE 4 BITS DES VIDEORAM</b>
<b>3</b>	<b>11</b>	<b>FARBRAM</b>

Steht also z.B. in dem entsprechenden Byte des Graphikspeichers ein Bitpaar 10, so hat der Punkt diejenige Farbe, deren Code in den oberen 4 Bits des zugehörigen Bytes des Videorams (Farbspeicher) abgelegt ist. Das 33. Register des VIC (= Videocontroller) ist Ihnen bereits als das Register bekannt, das auch im Textmodus die Hintergrundfarbe enthält und durch POKE 53248+33,f angesteuert wird, wobei f den Farbcode darstellt. Der Farbram nun ist der Speicherbereich, der im Textmodus die Zeichenfarbe liefert. Sie können also rein theoretisch für jeden 8x8-Block die Farben 1-3 variieren. Das folgende Beispiel mag Ihnen die Situation erläutern.

```

10 HIRES 2.1 : MULTI 12.8.6 : REM MULTICOLORGRAPHIK
EINSCHALTEN (S.U.)
20 FOR X=57344 TO 65344
30 POKE X, %1100100001 : REM 3 EIN-PUNKT-STREIFEN MIT 3
FARBEN ZEICHNEN
40 NEXT X
50 WAIT 198.255 : REM AUF TASTENDRUCK WARTEN

```

Das obige Beispiel zeichnet, stets durch eine Lücke (Hintergrundfarbe = 00) getrennt, ein Punkt breite Streifen auf den Bildschirm, deren 3 Farben sich ständig abwechseln.

Soweit zu den Hardware-Grundlagen, die Sie sich nicht unbedingt merken brauchen, die Ihnen aber einen Überblick über die recht komplizierte Graphikgestaltung des Commodore 64 bieten sollte, um die folgenden Ausführungen zu verstehen, und eine gute Hilfe darstellen, um später schöne Graphiken darzustellen.

## 12.2 GRAPHIKMODUS/FARBBESTIMMUNG

### KOORDINATENSYSTEM:

Simon's Basic unterteilt Ihren Graphikbildschirm in 320x200 Punkte (von 0 bis 319) in der hochauflösenden Graphik (HGR; s.o.) und 160x200 Punkte im Multicolormodus (0-159); in Multicolor ist jeder Punkt also doppelt so breit, wie in HGR. Dies hat spezielle Verzerreffekte, wenn man x- und y-Koordinaten gleichwertig behandelt. Für jeden Punkt müssen Sie also eine x- und eine y-Koordinate angeben, um ihn eindeutig zu bestimmen. Dabei wurde der Koordinatenursprung (Punkt 0,0) in die linke obere Ecke des Bildschirms verlegt, d.h. die y-Koordinaten zählen von oben nach unten und die x-Koordinaten von links nach rechts (Der Punkt 319,199 liegt in HGR beispielsweise in der unteren rechten Ecke des Bildes). Sollten Sie einmal eine nicht existierende Koordinate (z.B. 400,600) eingeben, so entsteht ein ILLEGAL QUANTITY ERROR.

Die Farben und die Farbauflösung in HGR und Multicolor wurden bereits in dem Kapitel "Grundlagen" ausgiebig behandelt (s.o.).

Sie haben die Möglichkeit, direkt in eine Graphik zu zeichnen, während Sie vom Betrachter auf der Mattscheibe beobachtet werden kann. Sie können jedoch auch jeden der untenstehenden Graphikbefehle korrekt anwenden, auch wenn Sie sich im Textmodus befinden, d.h. Sie können eine Graphik verdeckt zeichnen und später sichtbar machen.

Um mit einer kleinen Zahl von Befehlen möglichst viele Möglichkeiten auszuschöpfen, wurde der sogenannte Zeichenmodus (Zeichentyp) für jeden Zeichenbefehl (LIN, REC, ...) eingeführt. Er bestimmt die Art und Weise, wie ein Punkt, eine Linie, ein Rechteck usw. auf den Graphikbildschirm gezeichnet wird. D.h. Ein solches Gebilde kann mittels des Zeichenmodus unterschiedliches Aussehen erlangen. Der jeweilige Zeichenmodus wird stets in dem zu bearbeitenden Befehl angegeben (s. hierzu die einzelnen Befehle). Man unterscheidet dabei die Variationen des Zeichenmodus in Multicolor und diejenigen in HGR. Das folgende Schema verdeutlicht die einzelnen Funktionen:

**- HOCHAUFLÖSENDE GRAPHIK (HGR):**

**ZEICHENMODUS.....FUNKTION**

- 0.....LÖSCHT EIN GEBILDE**
- 1.....ZEICHNET EIN GEBILDE**
- 2.....INVERTIERT EIN GEBILDE**  
**(S.U.)**

**- MULTICOLORMODUS (MC):**

**ZEICHENMODUS.....FUNKTION**

- 0.....LÖSCHT EIN GEBILDE**
- 1.....ZEICHNET EIN GEBILDE**  
**MIT DER FARBE DES**  
**FARBREGISTERS 1**
- 2.....ZEICHNET EIN GEBILDE**  
**MIT DER FARBE DES**  
**FARBREGISTERS 2**
- 3.....ZEICHNET EIN GEBILDE**  
**MIT DER FARBE DES**  
**FARBREGISTERS 3**
- 4.....INVERTIERT EIN GEBILDE**  
**= FARBREGISTERWECHSEL:**  
**FARBREG. 0 ZU FARBREG. 3**  
**FARBREG. 1 ZU FARBREG. 2**  
**FARBREG. 2 ZU FARBREG. 1**  
**FARBREG. 3 ZU FARBREG. 0**

Diese Tabelle bedarf natürlich einiger Erklärungen:

Zunächst einmal wird unter Gebilde eine beliebige Figur verstanden, also Punkt, Linie, Kreis, etc, die Sie direkt mit einem Befehl des Simon's Basic erzeugen können. Invertieren eines Gebildes bedeutet hierbei das Löschen eines Punktes an der Stelle der zu zeichnenden Figur, an der bereits ein Punkt gesetzt war und entsprechend umgekehrt das Setzen dort, wo noch kein Punkt stand.

Unter Farbbregister sind die Speichereinheiten gemeint, die für die Festlegung einer der 4 möglichen Farben eines 8x8-Blocks bestimmt sind (s. Grundlagen zu diesem Kapitel). In ein Farbbregister wird also der entsprechende Farbcode (0-15) gelegt, den alle Punkte, die dieses Register ansteuern

(mittels den 2 für jeden Punkt im Graphikspeicher reservierten Bits, s.o.), erhalten sollen. Mit Farbbregister 0 ist dabei die normale Hintergrundfarbe gemeint, die ja im Multicolormodus aus dem Hintergrundfarbbregister 0 stammt (s. 9.3).

Der Zeichenmodus erst macht Ihre Befehle zu dem, wovon alle bewundernd sprechen, einem flexiblen und variationsreichen Graphikbefehlskomplex. Doch nun wollen wir uns mit den eigentlichen Befehlen zur Erzeugung von Graphiken erster Wahl befassen:

### 12.2.1 HIRES

```
FORMAT      :HIRES PF.HF
PARAMETER   : - PF: PUNKTFARBE (0-15)
              - HF: HINTERGRUNDFARBE (0-15)
BEISPIEL    :HIRES 7.8
FUNKTION     :EINSCHALTEN UND LÖSCHEN DER
              HOCHAUFLÖSENDEN GRAPHIK (HGR)
              UND BESTIMMEN DER
              PUNKT- UND HINTERGRUNDFARBEA
```

#### ERLÄUTERUNGEN:

Der Befehl, der Ihnen das Reich der Graphik eröffnet, ist der erste einer Reihe von Befehlen, die zur Initialisierung der Graphikbearbeitung, der Graphikmodus - Steuerung und der Farbwahl dienen. Sie stellen die Basis aller Graphikprogrammierung dar und sollten vollständig durchgearbeitet werden.

Dieser erste Befehl, der HIRES - Befehl, dient dazu, zunächst einmal vom Text- zum Graphikmodus umzuschalten. Gleichzeitig löscht er den Graphikspeicher, sodaß Sie nach der Eingabe dieses Befehls einen leeren Bildschirm vor sich sehen. Dies ist wichtig zu wissen, da man oft während eines Programmes einmal in den Textmodus zurückspringt. Will man nun wieder dieselbe Graphik unverändert zeigen, so muß mit einem anderen Befehl gearbeitet werden, der die Graphik ohne Löschvorgang einschaltet (s.u.). Natürlich können Sie den HIRES - Befehl auch verwenden, während Sie die Graphik anzeigen. In diesem



Falle wird nur gelöscht, und die Farben gesetzt.

Bei dem Löschen des Graphikbildes werden alle Bits des Graphikspeichers auf 0 zurückgesetzt und nur die Hintergrundfarbe aus dem Videoram ist sichtbar. Diese und die später verwendete Punktfarbe können Sie mit den beiden, dem HIRES-Befehl folgenden Parametern festlegen:

Der erste Wert definiert die Farbe (Farbcode) aller gesetzten Punkte des gesamten Graphikbildschirms (falls nicht durch einen anderen Farbbefehl verändert). D.h. alle Punkte, die im Folgenden gesetzt werden, erhalten diese eine Farbe, falls sie nicht durch einen extra Befehl (LOW COL) andersfarbig gezeichnet werden (Die Ausführungen über die Punktfarbe spielen natürlich nur in HGR eine Rolle. Die Farben der Punkte im Multicolor - Modus werden extra gesetzt, sodaß in MC die im HIRES-Befehl gesetzte Punktfarbe ignoriert wird, wie dies auch für den zweiten Parameter gilt.).

Im zweiten Wert geben Sie den Farbcode der Hintergrundfarbe an, d.h. der Farbe aller "nicht gesetzten" Punkte in dem Graphikfenster. Dies ist gleichfalls die Farbe, die Sie sehen, wenn Sie z.B. nach einem HIRES-Befehl ein gelöscht Graphikbild vor sich haben. Die hier gesetzte Hintergrundfarbe hat allerdings nichts mit der durch COLOUR (s. Kapitel 9.3) veränderbaren Hintergrundfarbe des Textmodus zu tun. Erst im Multicolor - Modus sind wieder Hintergrundfarbe des Textes und der Graphik aufgrund gleicher Register identisch.

Eine Bemerkung noch zum Ende: Der Graphikbildschirm ist nur im Programmmodus sichtbar, d.h. führen Sie den HIRES oder ähnliche Befehle direkt per Hand aus, so wird deren Funktion zwar ausgeführt, jedoch sofort wieder in den Textmodus zurückgeschaltet. Dasselbe passiert natürlich nach dem Beenden des Programms (auch etwa durch einen Fehler o.ä.). Wollen Sie den Graphikmodus im Programm ausschalten, so bedienen Sie sich entweder des NRM-Befehls (s. 9. Kapitel) oder des CSET 0/1 - Befehls (s. 13. Kapitel). Durch einen kleinen Trick jedoch ist es möglich, auch außerhalb des Programms die Graphik sichtbar zu halten und mit ihr zu arbeiten. Hierzu ist Ihnen im Folgenden ein 3-zeiliges Programm angegeben, das diesen Effekt (der eigentlich gar nicht vorgesehen ist) erzeugt. Dabei werden zwei Ihnen noch unbekannte Befehle verwendet:

1.) MEM, DER IM KAPITEL 13 ERLÄUTERT IST

2.) BLOCK, EIN IM FOLGENDEN ERKLÄRTER GRAPHIKBEFEHL.

Hier sollten Sie sie zunächst einfach hinnehmen, ohne sie zu verstehen.

10 HIRES 6.7 : REM GRAPHIK EINSCHALTEN

20 MEM : REM ZEICHENSATZ VERLEGEN (S. 13.1)

30 BLOCK 0.0,319,105.0 : REM OBEREN GRAPHIKTEIL LOESCHEN

Eine kleine Einschränkung muß gemacht werden: Sie können keinerlei Farbensetzung vornehmen, außer durch den Effekt, daß nun geschriebener Text als kleine Farbquadrate sichtbar wird.

BEISPIEL:

100 REM #####

110 REM ## . ##

120 REM ## HIRES-BEISPIEL ##

130 REM ## ##

140 REM #####

150 REM

160 FOR HF=0 TO 14 : REM HINTERGRUNDFARBE

170 HIRES 15, HF : REM GRAPHIK EINSCHALTEN. LOESCHEN UND  
FARBEN SETZEN

180 FOR X=59264 TO 59583

190 POKE X, %11110000 : REM BILDSCHIRMAUSSCHNITT: STREIFEN

200 NEXT X

210 NEXT HF : REM NAECHSTE HINTERGRUNDFARBE

220 REM NACH BEENDIGUNG: RUECKKEHR IN TEXT

Dieses Programm besteht aus zwei ineinander geschachtelten FOR ... NEXT - Schleifen. Die Äußere von beiden läßt die Hintergrundfarbe von Mal zu Mal insgesamt 15 Male wechseln, während die Innere eine Zeile lange, senkrechte Streifen durch EinPOKEn der entsprechenden Werte in die zuständigen Speicherzellen zeichnet (s. Einleitung zu diesem Kapitel). Dieses EinPOKEn geschieht natürlich zwangsweise ohne Farbgebung. Daran kann man sehen, daß die Farbe der einzelnen Punkte (hier: grau 3 --- Farbcode 15) bereits durch den HIRES - Befehl vorgegeben ist. Wechseln Sie doch einmal statt der

Hintergrundfarbe die Punktfarbe in der äußeren Schleife!

## 12.2.2 MULTI

```
FORMAT      :MULTI F1.F2.F3
PARAMETER   : - F1: PUNKTFARBE DES
              FARBREGISTERS 1 (0-15)
              - F2: PUNKTFARBE DES
              FARBREGISTERS 2 (0-15)
              - F3: PUNKTFARBE DES
              FARBREGISTERS 3 (0-15)
BEISPIEL    :MULTI 4,5,6
FUNKTION     :UMSCHALTEN DES GRAPHIKBETRIEBS
              AUF MULTICOLORMODUS (MC) UND
              BESTIMMEN DER 3 PUNKTFARBEN
```

### ERLÄUTERUNGEN:

Wie Sie bereits wissen, falls Sie die Einleitung gelesen haben, besitzt Ihr Rechner und damit auch Simon's Basic den sogenannten Multicolormodus, der aufgrund der extrem niedrigen Farbauflösung im HGR-Betrieb gewählt wurde. Mit Multicolor (MC) stehen Ihnen insgesamt 4 Farben pro 8x8- (bzw. 4x8-, da die Graphikauflösung in x-Richtung ja geringer ist) Block zur Verfügung.

Der MULTI-Befehl wirkt nicht analog zum HIRES-Befehl! Während letzterer eine ganze Reihe von Funktionen übernimmt (Einschalten der Graphik, Einschalten des HGR-Betriebs, Bestimmen der Hintergrund und Punktfarbe), schaltet MULTI lediglich von HGR- auf MC-Betrieb um und definiert die entsprechenden Farben. D.h. soll die Graphik während der Erstellung sichtbar sein (die andere Möglichkeit des verdeckt Zeichnens ist natürlich auch erlaubt), so muß sie zunächst durch HIRES (oder CSET 2 (s.u.)) eingeschaltet worden sein.

In den dreien dem MULTI-Kommando folgenden Parametern belegen Sie drei der vier in der Einleitung besprochenen Farbregister, die diejenige Farbe (als Farbcode) enthalten, die ein Punkt besitzt. Welches Register für welchen Punkt ausgewählt wird, wird ja bekanntlich (s.o.) durch die 2-Bit-Information eines Punktes im Graphikspeicher fest-

gelegt. Das bei diesen Parametern fehlende Farbbregister ist das Hintergrundfarbbregister, das entweder durch EinPOKE (POKE 53248 + 33,f0) oder durch Senden des Befehls COLOUR (s. Kapitel 9) verändert werden kann.

Über die Adressen bzw. die Lage der einzelnen Farbbregister informiert Sie die Einleitung ausführlich (f1/2 im Videoram / f3 im Farbram). Dort erfahren Sie auch, daß das Farbbregister 3 (oder besser die Farbbregister 3), das Farbe Nummer 3 für jeden 8x8 (bzw. 4x8) - Block angibt, im sogenannten Farbram, also dem Farbspeicher des Textmodus, liegt. Diese Überschneidung bringt nun auch gleich einige Probleme mit sich: Wurde der Multicolormodus eingeschaltet, so wurde zwangsläufig auch das Farbbregister 3, also der Farbspeicher, mit dem entsprechenden Wert belegt. D.h. alle Zeichen, die zur Zeit im Textfenster stehen, egal ob gerade Graphik oder Text sichtbar sind, bekommen die Farbe, die Sie als f3 im MULTI-Befehl eingegeben haben. Probieren Sie es aus! Schreiben Sie einmal den Bildschirm voller Zeichen und geben Sie dann ein:

### **MULTI 1.2.3**

Sie werden den Effekt sogleich bemerken. Verändern Sie ruhig einmal den 3. Parameter in diesem Beispiel. Sie sehen also: haben Sie mühevoll einen farbigen Text konstruiert, so wird alle Arbeit wieder durch einen Befehl zunichte gemacht.

Gleichzeitig geschieht natürlich umgekehrt Einiges, wenn Sie, während Sie eine Graphik im Multicolormodus erstellen, Text in das Textfenster verdeckt (das ist ja auch möglich) schreiben. Sofort ändert sich an diesen Stellen in der Graphik die Farbe 3. Man muß damit leben und entsprechend vorsichtig agieren.

Natürlich können Sie diese Effekte auch für Ihre Zwecke ausnutzen. Ihrer Phantasie sind dabei keine Grenzen gesetzt. Im Anschluß an diese Darlegungen sei ein kleines Beispiel angeführt, das ihnen einige Auswirkungen des MULTI-Befehls veranschaulicht:

**BEISPIEL :**

```
100 REM #####
110 REM ##          ##
120 REM ##  MULTI-BEISPIEL  ##
130 REM ##          ##
140 REM #####
150 REM
160 COLOUR 0.6
170 HIRES 7.6 : REM GRAPHIK EINSCHALTEN, LOESCHEN UND
HINTERGRUNDFARBE = BLAU
180 AD = 60000 : REM BASISADRESSE AUSGANGSPUNKT
190 FOR X=0 TO 7
200 POKE X+AD , %00000000 : REM 1 STREIFEN MIT DER FARBE 0
(HINTERGRUND) ZEICHNE
210 NEXT X : REM NAECHSTES BYTE
220 FOR X=8 TO 15
230 POKE X+AD , %01010101 : REM 1 STREIFEN MIT DER FARBE 1
ZEICHNEN
240 NEXT X : REM NAECHSTES BYTE
250 FOR X=16 TO 23
260 POKE X+AD , %10101010 : REM 1 STREIFEN MIT DER FARBE 2
ZEICHNEN
270 NEXT X : REM NAECHSTES BYTE
280 FOR X=24 TO 31
290 POKE X+AD , %11111111 : REM 1 STREIFEN MIT DER FARBE 3
ZEICHNEN
300 NEXT X : REM NAECHSTES BYTE
310 REM
320 PAUSE 5 : REM NOCH IST NUR HGR EINGESCHALTET!
330 FOR X=0 TO 15
340 MULTI X.7.5 : REM MULTICOLOR EINSCHALTEN UND 3 FARBEN
WAEHLEN:
350 REM FARBE 1: VARIABEL
360 REM FARBE 2: GELB
370 REM FARBE 3: GRUEN
380 PAUSE 1
390 NEXT X
400 REM
410 REM NAECHSTE FARBE BLINKEN LASSEN:
420 FOR X=0 TO 15
```

```

430 MULTI 7.X.5 : REM MULTICOLOR EINSCHALTEN UND 3 FARBEN
WAEHLN:
440 REM FARBE 1: GELB
450 REM FARBE 2: VARIABEL
460 REM FARBE 3: GRUEN
470 PAUSE 1
480 NEXT X
490 REM
500 REM NAECHSTE FARBE BLINKEN LASSEN:
510 FOR X=0 TO 15
520 MULTI 7.5.X : REM MULTICOLOR EINSCHALTEN UND 3 FARBEN
WAEHLN:
530 REM FARBE 1: GELB
540 REM FARBE 2: GRUEN
550 REM FARBE 3: VARIABEL
560 PAUSE 1
570 NEXT X

```

Wie Sie sehen, lassen sich Teile der Graphik, die in verschiedenen Farben (Farbregisterzuordnungen) gespeichert wurden, farblich getrennt ansprechen.

Jeweils in den Zeilen 200, 230, 260 und 290 wurden ganz bestimmte Werte (hier in Dualform angegeben (s. Kapitel 7)) in den Graphikspeicher eingeschrieben (wir hätten auch in den entsprechenden Farben mit den Befehlen PLOT etc. zeichnen können). Jeweils zwei Bits (hier zwei Ziffern) jedes Wertes bestimmen die Zuordnung des Punktes zu einem der 4 Farbregister (s. Einleitung). Verändern Sie ruhig einmal z.B. an diesen Stellen das Programm. Dies wird das Verständnis sehr erleichtern!

### 12.2.3 NRM

```
FORMAT      :NRM
PARAMETER   : ----
BEISPIEL    :NRM
FUNKTION    :AUSSCHALTEN DES GRAPHIKMODUS
             UND EINSCHALTEN DES GROSS-
             SCHRIFT / GRAPHIKZEICHEN-
             ZEICHENSATZES
```

#### ERLÄUTERUNGEN:

Dieser Befehl sei an dieser Stelle nur der Vollständigkeit halber erwähnt, da er schon unter # 9.3.3 ausgiebig erläutert wurde und dessen genaue Funktion dort nachzulesen ist. Im gegenwärtigen Zusammenhang ist seine Funktion zum Ausschalten des Graphikmodus interessant. Dies wird im folgenden Beispiel dargestellt:

#### BEISPIEL:

```
100 REM #####
110 REM ##                ##
120 REM ##  NRM-BEISPIEL  ##
130 REM ##                ##
140 REM #####
150 REM
160 PRINT CHR$(14) : REM AUF GROSS-/KLEINSCHRIFT UMSCHALTEN
170 PRINT CHR$(147) : REM TEXT-BILOSCHIRM LOESCHEN
180 CENTRE "IM MOMENT IST DER KLEIN-/GROSSCHRIFT-" : PRINT
190 CENTRE "MODUS AKTIVIERT." : PRINT
200 PRINT : REM LEERZEILE
210 CENTRE "ACHTEN SIE EINMAL AUF DIE FOLGENDE" : PRINT
220 CENTRE "MELDUNG!"
230 PAUSE 9
240 PRINT CHR$(147) : REM TEXT-BILDSCHIRM LOESCHEN
250 CENTRE "DIES IST DIE TEXTSEITE" : REM MELDUNG DER
TEXTSEITE
260 REM
270 FOR Y=1 TO 5
280 PAUSE 2
290 HIRES 7.6 : REM HGR EIN + LOESCHEN / PUNKTFARBE=GELB /
```

## HINTERGRUND=BLAU

```
300 FDR X=59264 TD 59583 : REM UMFASST EINE ZEILE
310 PDKE X. %11110000 : REM STREIFEN ZEICHNEN
320 NEXT X
330 PAUSE 2
340 NRM : REM TEXTDDUS EINSCHALTEN (AUF GRDSSBUCHSTABEN)
350 NEXT Y : REM 5 MAL WECHSELN
360 PRINT : PRINT : REM 2 LEERZEILEN
370 CENTRE "WIE SIE SEHEN. WURDE DURCH NRM AUF" : PRINT
380 CENTRE "GRDSSBUCHSTABEN UMGESCHALTET"
390 REM VERAENDERN SIE DOCH EINMAL DEN WERT %11110000 IN
ZEILE 220
400 REM Z.B. IN %11000000 DDER 11001100 DDER ...
```

### 12.2.4 CSET 2

```
FORMAT      :CSET 2
PARAMETER   : ----
BEISPIEL    :CSET 2
FUNKTION    :EINSCHALTEN DER GRAPHIKANZEIGE
              OHNE LÖSCHEN DES
              GRAPHIKBILDSCHIRMS
```

#### ERLÄUTERUNGEN:

Mit CSET wird Ihnen ein Befehl in die Hand gegeben, der eine Reihe verschiedener und auf den ersten Blick unzusammenhängender Funktionen ausführt. Seine verschiedenen Erscheinungsformen werden später noch einmal Mittelpunkt unseres Interesses stehen; nämlich dann, wenn wir es mit den verschiedenen Zeichensätzen unseres CBM 64 zu tun haben werden (Kapitel 13). Dort taucht er in Form von CSET 0 und CSET 1 auf. Diese beiden Befehlsvariationen schalten auch unter anderem den sichtbaren Graphikbetrieb aus (ähnlich NRM), aber davon mehr in dem entsprechenden Kapitel. Hier und jetzt interessiert uns seine Erscheinungsform als CSET 2, die uns eine weitere Möglichkeit der Graphikmanipulation in die Hände gibt: CSET 2 schaltet nämlich - wie uns schon vom HIRES-Befehl bekannt - den Graphikbildschirm ein. Hier aber weist er einen entscheidenden Unterschied im



Vergleich zu seinem großen Bruder auf: CSET 2 schaltet ein, ohne jedoch den Graphikbildschirm zu löschen! D.h. haben Sie eine Graphik gezeichnet, so können Sie zwischendurch einmal z.B. mit NRM oder CSET 0/1 (s. Kapitel 13) in den Textmodus umschalten, um Kommentare oder Ähnliches abzugeben, und haben nun die Möglichkeit, die alte Graphik sofort wieder voll sichtbar zu machen, ohne sie erneut zeichnen zu müssen. Auch die Farbgebung bleibt voll erhalten, d.h. auch diese brauchen Sie nicht erneut zu setzen. Sie besitzen sogar die Möglichkeit, und das ist unter Anderem sehr nützlich, eine Graphik zu zeichnen, während Sie einen Text anzeigen. Später schalten Sie dann, wenn es Ihnen gefällt, mit CSET 2 diese Graphik ein, um sie auch nach außen sichtbar zur Schau zu stellen. Welche Möglichkeiten sich damit für Sie auftun, brauche ich Ihnen nicht erst zu sagen. Überlegen Sie sich doch einmal Einiges!

Eine der vielen möglichen Anwendungen sei Ihnen anhand eines Beispiels gezeigt:

#### BEISPIEL:

```

100 REM #####
110 REM ##                               ##
120 REM ## CSET 2-BEISPIEL ##
130 REM ##                               ##
140 REM #####
150 REM
160 PRINT CHR$(147) : REM TEXT-BILDSCHIRM LOESCHEN
170 CENTRE "DIES IST DIE TEXTSEITE" : REM MELDUNG DER
TEXTSEITE
180 PAUSE 2
190 HIRES 7.2 : REM HGR EIN + LOESCHEN / PUNKTFARBE=GELB /
HINTERGRUND=ROT
200 REM
210 FOR X=59264 TO 59583 : REM UMFASST EINE ZEILE
220 POKE X, %11110000 : REM STREIFEN ZEICHNEN
230 NEXT X
240 FOR X=1 TO 5
250 PAUSE 2
260 NRM : REM TEXTMODUS EINSCHALTEN (AUF GROSSBUCHSTABEN)
270 PAUSE 2

```

**280 CSET 2 : REM GRAPHIKMODUS EINSCHALTEN OHNE LOESCHEN UND  
FARBSETZUNG**

**290 NEXT X**

Wie Sie sehen ist ein gleichzeitiger Betrieb von Graphik und Text möglich. Wie immer können Sie natürlich an diesem Programm basteln, wie es Ihnen gefällt. Besonderes Augenmerk sollten Sie vielleicht auf den Dualwert der Zeile 220 legen. Den können Sie eventuell (wie auch bei den vorigen Beispielen), falls Sie den Graphikaufbau noch nicht recht verstanden haben, ruhig einmal verändern.

## 12.2.5 LOW COL

```
FORMAT      :LOW COL F1.F2.F3
PARAMETER   : - F1: HGR: PUNKTFARBE BEIM
              ZEICHNEN
              MC: FARBBELEGUNG DES
              FARBREGISTERS 1 (0-15)
              BEIM ZEICHNEN
              - F2: HGR: KEINE FUNKTION
              MC: FARBBELEGUNG DES
              FARBREGISTERS 2 (0-15)
              BEIM ZEICHNEN
              - F3: HGR: KEINE FUNKTION
              MC: FARBBELEGUNG DES
              FARBREGISTERS 3 (0-15)
              BEIM ZEICHNEN
BEISPIEL    :LOW COL 3.6.2
FUNKTION     :BESTIMMEN DER PUNKTFARBE(N) ZUR
              FARBSETZUNG BEIM ZEICHNEN UND
              SETZEN DIESER FARBE(N) FÜR
              JEDEN GEZEICHNETEN PUNKT
```

### ERLÄUTERUNGEN:

Sicher haben Sie sich bei der Lektüre der bisherigen Befehle gefragt, ob man denn alle Punkte nur in einer (HGR) bzw. drei (MC) Farben darstellen kann, da man diese bisher ja nur durch die Befehle HIRES oder MULTI einmal am Anfang und dann nur für die gesamte Graphik setzen kann. Nun, dem kann abgeholfen werden. Es gibt einen Befehl, um alle im Folgenden gesetzten Punkte mit einer neuen Farbe zu belegen: LOW COL.

Mit den dem Befehlsword folgenden Parametern bestimmen Sie jeweils die neue Farbkombination. In HGR werden von den drei erforderlichen Werten f1,f2 und f3 lediglich nur die beiden ersten benötigt, da wir hier ja nur eine Punktfarbe pro 8x8-Block festlegen können. Der zweite Wert dient zur Festlegung der Hintergrundfarbe des 8x8-Feldes (wie aus 12.1 bekannt, kann die Hintergrundfarbe in HGR ebenfalls in jedem solchen Felde einen anderen Wert annehmen, da sie gleichfalls aus dem Videoram (Graphikfarbspeicher) entstammt). Dieser Effekt wird in dem zweiten unter HI COL aufgeführten Beispiel erläutert. Trotzdem müssen Sie aber alle drei Parameter

eingeben, um keinen SYNTAX ERROR zu erhalten. Der überflüssige f3 wird dabei ignoriert und kann somit beliebig gewählt werden. In Multicolor dagegen bestimmen die beiden letzten Farbcodes die zwei weiteren Punktfarben (die Hintergrundfarbe wird ja bekanntlich für das gesamte Bild durch COLOUR einmal bestimmt), die in einem 8x8- bzw. 4x8-Block vorkommen können.

Doch die Handhabung dieses Befehls ist nicht so einfach, wie es auf dem ersten Blick erscheint. Es gibt nämlich Unterschiede bei der Zeichnung eines Punktes oder einer Figur, bevor LOW COL gegeben wurde und danach:

Nach dem Einschalten der Graphik durch den HIRES-Befehl (und der eventuellen Umschaltung durch MULTI auf MC) liegen - wie bekannt - die Punktfarben für das gesamte Graphikbild bereits fest und sind in den einzelnen Speicherzellen der verschiedenen Farbreghister schon gespeichert. Infolge dessen braucht die Farbe bei der Zeichnung eines Punktes nicht extra noch mitgesetzt zu werden, sondern es wird einfach ein Punkt in den Graphikspeicher gesetzt. Dieser bekommt ja automatisch die entsprechende Farbe, die bereits im Farbspeicher bereitsteht.

Haben Sie jedoch den LOW COL-Befehl eingegeben, so muß die Farbe eines Punktes extra noch nach den Farbinformationen, die Sie an dieses Kommando anhängen, in die entsprechenden Speicherzellen der verschiedenen Farbreghister eingespeichert werden. D.h. zusätzlich zu dem Setzen (oder Speichern) eines Punktes in den Graphikspeicher wird noch die Farbe in die korrespondierenden Speicherzellen des Punktspeichers (in HGR) bzw. der 3 Farbreghister (in MC) eingelegt.

Da aber nun eine Speicherzelle eines Farbreghisters für jeweils 8x8 (HGR) bzw. 4x8 (MC) Punkte zuständig ist, bekommen alle anderen Punkte, die zufällig bereits in dem Block gesetzt waren, dieselbe Farbe wie der neue unter LOW COL gesetzte Punkt, auch wenn Sie ursprünglich eine andere Farbe besaßen.

Wird nun später einmal wieder ohne LOW COL (dieser Befehl kann durch HI COL (s.u.) wieder aufgehoben werden), also ohne gleichzeitige Farbsetzung in den besagten 8x8 (4x8) - Block gezeichnet, so erhält der gesetzte Punkt dementsprechend die Farbe, die die bereits gesetzten Punkte an dieser Stelle schon besitzen, es wird also die Farbe übernommen.

Diese Effekte werden anschaulich durch das Beispiel unter HI COL demonstriert. Achten Sie dort besonders auf die Einzelheiten!

Wenn Sie nun beispielsweise in HGR 2 Figuren (Punkte, Linien, Kreise, ...) unterschiedlicher Farbe zeichnen wollen, die sich wahrscheinlich sehr nahe kommen oder sogar berühren, so sollten Sie darauf achten, bei welcher Figur es Ihnen wichtiger ist, daß sie vollständig die Farbe erhält, die ihr zugedacht ist, da es vorkommen kann, da sich Teile dieser Figuren in Ihrer Farbgebung überschneiden. Nun gibt es zwei Wege, wie Sie dieses Problem lösen können:

Entweder Sie zeichnen die Figur, auf deren Farbe Sie nicht so großen Wert legen, zuerst, oder Sie zeichnen diese ohne Farbgebung, also ohne LOW COL, während Sie die andere mit LOW COL konstruieren. Im ersten Fall muß die Reihenfolge des Zeichnens strikt eingehalten werden, im zweiten Falle ist sie egal.

Sie sehen also, Ihr CBM 64 hat eine recht komplizierte Graphik (besonders was die Farbgebung betrifft), was aber ob seiner Möglichkeiten nicht verwundert. Wenn man einmal das System verstanden hat, dann ist die Programmierung jedoch nicht mehr allzu schwierig. Deshalb gilt besonders hier ein eiserner Grundsatz: probieren, probieren und nochmals probieren!!!

Hier ein

#### **BEISPIEL:**

```
100 REM #####
110 REM ##                               ##
120 REM ##  LOW COL-BEISPIEL  ##
130 REM ##                               ##
140 REM #####
150 REM
160 COLOUR 0.1 : REM RAHMENFARBE = SCHWARZ
170 AB = 8 : REM ABSTAND ZWEIER LINIEN
180 REM
190 FOR Z=1 TO 2 : REM ZWEI DURCHLAEUFE
200 HIRES 7.0 : REM HGR EIN + LOESCHEN / PUNKTFARBE=GELB /
HINTERGRUND=SCHWARZ
210 FOR Y=1 TO 15
```

```

220 FOR X=0 TO 320
230 PLOT X.Y*AB.1 : REM EINE REIHE PUNKTE ZEICHNEN (S.
PLOT-BEFEHL)
240 NEXT X : REM 320 PUNKTE (EINE ZEILE) ZEICHNEN
250 LOW COL Y.O.O : REM 2. FARBE SETZEN-2 LETZTEN PARAMETER
IN HGR UNWESENTLICH
260 NEXT Y : REM FARBWECHSEL
270 PAUSE 5
280 NRM : REM AUF TEXTMODUS SCHALTEN
290 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
300 CENTRE "BITTE SEHEN SIE SICH DEN EFFEKT AN." : PRINT
310 CENTRE "WENN DIE LINIEN ZU NAHE ANEINANDER" : PRINT
320 CENTRE "LIEGEN." : PRINT
330 PRINT : REM LEERZEILE
340 CENTRE "AUSGEOEST DURCH DIE NIEDRIGE" : PRINT
350 CENTRE "FARBAUFLÖSUNG!"
360 PAUSE 9
370 AB = 2 : REM LINIENABSTAND
380 NEXT Z : REM GRAPHIK NEU ZEICHNEN - MIT KLEINEREM
LINIENABSTAND

```

In dem obigen Programm taucht in der Zeile 230 ein bisher unbekannter Befehl auf: PLOT. PLOT dient zum Zeichnen eines Punktes mit den Koordinaten x,y und wird später im Abschnitt 12.3.1 näher besprochen. Hier sollten Sie Ihn einfach kritiklos hinnehmen.

Dieses Beispiel demonstriert Ihnen die Problematik der niedrigen Farbauflösung (in HGR). Bei dem zweiten Zeichnen der waagerechten Linien (durch die äußere FOR ... NEXT - Schleife) werden diese so nahe beieinander gezeichnet, daß es zu Überschneidungen mit den vorher gezeichneten Linien in der Farbgebung kommt.

## 12.2.6 HI COL

<b>FORMAT</b>	<b>:HI COL</b>
<b>PARAMETER</b>	<b>: ----</b>
<b>BEISPIEL</b>	<b>:HI COL</b>
<b>FUNKTION</b>	<b>:AUFHEBEN DES LOW COL- MODUS = ZEICHNEN OHNE FARBSETZUNG</b>

### **ERLÄUTERUNGEN:**

Vor der Lektüre dieses Befehls sollten Sie die Funktionsweise des LOW COL-Befehls durchgearbeitet und verstanden haben, da dies im Folgenden vorausgesetzt wird.

Wie Sie unter 12.2.5 (LOW COL) gelesen haben, wird normalerweise nach dem Initialisieren der Graphik durch HIRES jeder gesetzte Punkt ohne Farbsetzung gezeichnet, da bereits alle Punkte eine, in diesem Befehl definierte Farbe besitzen. Erst nach der Eingabe von LOW COL wird unter Farbsetzung gezeichnet, also mit jedem gesetztem Punkt die angegebene Farbe in das Videoram (Graphikfarbspeicher) bzw. bei Multicolor zusätzlich in das Farbram (Textfarbspeicher) abgelegt.

Diese Farbbehandlung brachte, wie wir unter LOW COL gesehen haben, einige Probleme mit sich, scheint aber die einzig sinnvolle zu sein. Aus Gründen der uneingeschränkten Verfügbarkeit über die Farbmöglichkeiten des Commodore 64 hatten wir schon in Abschnitt 12.2.5 die Notwendigkeit eines Befehls erkannt, der die Wirkungsweise des LOW COL-Kommandos (Farbsetzung) wieder aufhebt, sodaß wie zu Beginn der Graphikaktivitäten Figuren gezeichnet werden können, ohne daß gleich die Farbe mit in den Farbspeicher gelegt wird. Diese Funktion übernimmt nun der parameterlose HI COL-Befehl. Welche Wichtigkeit diesem Befehl im Rahmen der Farbbehandlung zukommt, wird Ihnen vielleicht durch das unten aufgeführte Beispiel einer Kombination von HI COL und LOW COL, einem unbedingt zusammengehörenden Befehlspaar, verdeutlicht.

**BEISPIEL :**

```
100 REM #####
110 REM ##          ##
120 REM ##  HI COL-BEISPIEL  ##
130 REM ##          ##
140 REM #####
150 REM
160 COLOUR 0.1 : REM RAHMENFARBE = SCHWARZ
170 REM
180 HIRES 6.0 : REM HGR EIN + LOESCHEN /  PUNKTFARBE=BLAU /
HINTERGRUND=SCHWARZ
190 FOR Y=0 TO 11
200 FOR X=0 TO 319
210 PLOT X,Y*16.1 : REM EINE REIHE PUNKTE ZEICHNEN (S.
PLOT-BEFEHL)
220 NEXT X : REM 320 PUNKTE (EINE ZEILE) ZEICHNEN
230 LOW COL Y+1.0.0 : REM 2. FARBE SETZEN-2 LETZTEN PARAMETER
IN HGR UNWESENTLICH
240 FOR X=0 TO 319
250 PLOT X,Y*16+8.1 : REM EINE REIHE PUNKTE ZEICHNEN (S.
PLOT-BEFEHL)
260 NEXT X : REM 320 PUNKTE (EINE ZEILE) ZEICHNEN
270 HI COL : REM MIT ALTER FARBE = OHNE FARBSETZUNG ZEICHNEN
(BLAU)
280 NEXT Y : REM FARBWECHSEL
290 PAUSE 5
300 NRM : REM AUF TEXTMODUS SCHALTEN
310 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
320 CENTRE "ACHTEN SIE EINMAL DARAUF, WIE UNTER" : PRINT
330 CENTRE "LOW COL MIT FARBSETZUNG, UNTER" : PRINT
340 CENTRE "HI COL DAGEGEN OHNE FARBSETZUNG" : PRINT
350 CENTRE "GEZEICHNET WIRD"
360 PAUSE 10
370 CSET 2 : REM GRAPHIK EINSCHALTEN OHNE LOESCHEN
380 FOR Y=0 TO 9
390 FOR X=0 TO 199
400 PLOT Y*32.X.1 : PLOT Y*32+1.X.1 : REM DOPPELT DICK
410 REM EINE SPALTE PUNKTE ZEICHNEN (S. PLOT-BEFEHL)
420 NEXT X : REM 200 PUNKTE (EINE SPALTE) ZEICHNEN
430 LOW COL Y.0.0 : REM 2. FARBE SETZEN-2 LETZTEN PARAMETER
```



**IN HGR UNWESENTLICH**

**440 FOR X=0 TO 199**

**450 PLOT Y\*32+16.X,1 : PLOT Y\*32+17.X,1 : REM DOPPELT DICK**

**460 REM EINE SPALTE PUNKTE ZEICHNEN (S. PLOT-BEFEHL)**

**470 NEXT X : REM 200 PUNKTE (EINE SPALTE) ZEICHNEN**

**480 HI COL : REM MIT ALTER FARBE = OHNE FARBSETZUNG ZEICHNEN  
(BLAU)**

**490 NEXT Y : REM FARBWECHSEL**

**500 PAUSE 20**

Wie Sie vielleicht bemerkt haben, lehnt sich dieses Beispiel eng an das Programm für den Befehl LOW COL an. Wir haben dies derart gemacht, um Ihnen das Verständnis etwas zu erleichtern.

Zunächst eine kleine Programmbeschreibung: In diesem Beispiel eines kombinierten LOW COL - HI COL Einsatzes werden zu Anfang waagerechte Striche mit konstantem Abstand von 8 Punkten stets abwechselnd mit und ohne Farbsetzung gezeichnet. Dabei wird die allgemeine (in HIRES angegebene) Punktfarbe blau gehalten. Die mit LOW COL gesetzten Linien dagegen wechseln stets in Ihrer Farbe ab.

Dies ist an sich nichts Besonderes, da dieser Vorgang Ihnen schon in etwa vom vorherigen Beispiel her bekannt sein sollte. Nun kommt aber ein zweites Element hinein, das Ihnen den Zusammenhang, bzw. die Unterschiede zwischen Punkten mit und ohne Farbsetzung darlegt. Es werden (ab Zeile 370) senkrechte Linien mit einem Abstand von 16 Punkten ebenfalls jeweils mit und ohne Farbsetzung gezogen. Dabei sollte Ihnen auffallen, wie die Farbsetzung der unter LOW COL gezeichneten Senkrechten die Farbe der umliegenden Punkte beeinflusst, während im Gegensatz hierzu die unter HI COL gesetzten Linien von den verschiedenen Farben der bereits gesetzten Punkte (Linien) beeinflusst werden (Die Senkrechten werden jeweils doppelt dick gezeichnet, um die Farbe der einzelnen Linien erkennen zu können.).

-----

Das nächste Beispiel ist eigentlich ein Beispiel zu LOW COL, wird aber aufgrund des auftauchenden HI COL-Befehls erst hier angeführt. Dieses Beispiel erläutert einen völlig anderen

Aspekt des LOW COL-Kommandos, nämlich den des Setzens der Hintergrundfarbe nach LOW COL:

**BEISPIEL :**

```
100 REM #####
110 REM ##                      ##
120 REM ##  LOW COL-BEISPIEL-2  ##
130 REM ##                      ##
140 REM #####
150 REM
160 HIRES 5.6 : REM GRAPHIK EIN/HINTERGRUND = BLAU/PUNKTE =
GRUEN
170 LOW COL 1.5.0 : REM  FARBWECHSEL:  HINTERGRUND  =
GRUEN/PUNKTE = WEISS
180 REM
190 REM WAAGERECHTE ZEICHNEN (MIT FARBE):
200 FOR X=0 TO 319
210 PLOT X,66.1 : REM PUNKT ZEICHNEN
220 NEXT X
230 PAUSE 2
240 LOW COL 0.1.0 : REM  FARBWECHSEL:  HINTERGRUND  =
WEISS/PUNKTE = SCHWARZ
250 REM
260 REM WAAGERECHTE LOESCHEN (MIT FARBE):
270 FOR X=0 TO 319
280 PLOT X,133.0 : REM PUNKT LOESCHEN
290 NEXT X
300 PAUSE 2
310 HI COL : REM RUECKSCHALTEN AUF ZEICHNEN OHNE FARBSETZUNG
320 REM
330 REM SENKRECHTE ZEICHNEN (OHNE FARBE):
340 FOR Y=0 TO 199
350 PLOT 160.Y,1 : REM PUNKT ZEICHNEN
360 NEXT Y
370 WAIT 198.255 : REM AUF TASTENDRUCK WARTEN
```

Wie Sie sehen, wird neben der Punktfarbe gleichfalls die Hintergrundfarbe unter LOW COL gesetzt. Dies ist möglich, da die Hintergrundfarbe - wie aus der Erläuterung der Grundlagen in Abschnitt 9.1 bekannt - genauso wie die Punktfarbe aus dem

Videoram (= Farbspeicher der Graphik) stammt, somit also auch für jeden 8x8-Block verschieden sein kann.

Der zweite Punkt, der durch dieses Beispiel veranschaulicht wird, ist das Setzen der Farbe, gleichgültig, ob der angesprochene Punkt gesetzt oder gelöscht wird, was ja bekanntlich durch den Zeichenmodus angegeben wird (s. PLOT). Obwohl wir also einen Punkt löschen, wird die Farbe in den Farbregistern verändert. Soll also die Farbe ebenfalls wieder zurückgesetzt werden, so müssen Sie vor dem Löschen die LOW COL-Farbe mit demjenigen Wert belegen, der Ihrer normalen HI COL-Farbe (also der Farbe, die im HIRES-Befehl gesetzt wurde) entspricht.

## 12.2.7 TESTAUFGABEN

Diesmal wird unser Test zwangsläufig etwas schwieriger, da wir es auch mit einem recht komplizierten Thema zu tun haben. Verzweifeln Sie also nicht gleich, wenn nicht sofort alles klappt. Lesen Sie sich in diesem Falle die entsprechenden Stellen noch einmal in Ruhe durch.

1.) Wie initialisieren Sie zum Anfang Ihrer Graphikarbeit die Multicolorgraphik? durch:

- a) HIRES ...
- b) HIRES ... : MULTI ...
- c) CSET 2 : MULTI ...
- d) CSET 2

2.) Was bewirkt der Befehl NRM?

- a) nur ein Ausschalten der Graphik
- b) Ausschalten der Graphik und Einstellen des Groß- / Klein - Zeichensatzes
- c) Ausschalten der Graphik und Einstellen des Großschrift / Graphik - Zeichensatzes
- d) Zurücksetzen der Farbe auf die Ausgangswerte

3.) Sie zeichnen einen Punkt, nachdem Sie LOW COL eingegeben haben. Was ist zu beachten?

- a) Es findet nur ein Farbwechsel statt
- b) Alle Punkte in dem gleichen 8x8-Feld erhalten die gleiche Farbe
- c) Die Hintergrundfarbe für diesen Punkt wird gesetzt
- d) Die Farbnummer der Punktfarbe wird um eins erniedrigt.

### 12.3 GRAPHIKBEFEHLE

Alle recht trockene Arbeit in den Abschnitten zuvor wird nun durch die eigentlichen Graphikbefehle belohnt. Nun können Sie allein die Graphik betreiben mit dem Zeichnen von Punkten, Strichen, Kreisen, Feldern und Vieles mehr. Alles Befehle, die direkt mit der Graphik zu tun haben, und mit denen wir unsere eigenen, schönen Graphiken herstellen können. Schauen Sie zu und probieren Sie, denn nun beginnt das Niemandsland. Nun haben Sie die Möglichkeit, Dinge zu erschaffen, die nie ein anderer Mensch vor Ihnen auf den Bildschirm gebracht hat. Hier fängt die Kunst, das Design an. Oder Sie gebrauchen die Graphik nur zum Erstellen von Spielen, wozu Ihnen später noch die Spritebefehle sehr nützlich erscheinen werden. Kurz: nun kommen wir zum Kernthema unseres Graphikkapitels, den Zeichenbefehlen.

Zu jedem der folgenden Graphikbefehle können Sie neben den anderen Parametern, die von Fall zu Fall verschieden sind und an Ort und Stelle erläutert werden, den Ihnen schon aus dem vorherigen Abschnitt bekannten Zeichenmodus (in der Parameterschreibweise als zm gekennzeichnet) wählen. Damit können Sie je nach Belieben bestimmen, ob Sie die angesprochene Figur zeichnen, löschen oder invertieren wollen, wie in Abschnitt 12.2 erklärt. Somit stehen Ihnen pro Befehl eine ganze Reihe von Möglichkeiten der Figurdarstellung offen, da jeder Befehl natürlich gleichfalls in Multicolor funktioniert. Unter x- bzw. y-Koordinate wird natürlich die Einteilung verstanden, die Simon's Basic wie unter 12.2 beschrieben vornimmt.

Und noch einen dringenden Tip möchte ich Ihnen auf den Weg geben: Probieren Sie möglichst alle Beispiele aus, Sie werden es nicht bereuen! Nur so lernen Sie die faszinierenden Möglichkeiten der einzelnen Befehle recht imposant kennen. Wichtig ist, daß Sie versuchen, möglichst viel an den angegebenen Programmen zu verändern, um die resultierenden Phänomene kennenzulernen.

Nun denn, viel Spaß!

### 12.3.1 PLOT

**FORMAT** : PLOT X.Y.ZM  
**PARAMETER** : - X : X-KOORDINATE DES PUNKTES  
                  MC: 0-159 / HGR: 0-319  
              - Y : Y-KOORDINATE DES PUNKTES  
                  MC: 0-199 / HGR: 0-199  
              - ZM: ZEICHENMODUS  
**BEISPIELE** : PLOT 160.100.1  
**FUNKTION** : SETZEN EINES PUNKTES BEI X.Y

#### ERLÄUTERUNGEN:

Mit PLOT, dem einfachsten Zeichenbefehl, werden Sie in das Reich der Graphik eingeführt. PLOT setzt lediglich einen einzigen Punkt auf den Bildschirm, dessen Koordinaten Sie mit x und y angeben. Beachten Sie, daß der Nullpunkt des imaginären Koordinatensystems in der oberen linken Ecke des Bildschirms liegt. Mit zm geben Sie - wie beschrieben - den Zeichenmodus an, d.h. bei z=0 wird ein eventuell gesetzter Punkt an der angegebenen Stelle gelöscht, bei z=1 wird er gesetzt, und bei z=2 erscheint an der besagten Koordinate ein Punkt, wenn vorher keiner dort stand, und umgekehrt wird hier ein Punkt gelöscht, falls bereits ein Punkt gesetzt war.

An diesem Befehl können Sie den Umgang mit den zwei Koordinaten und dem Zeichenmodus noch recht einfach üben und die verschiedenen Effekte ausprobieren.

Im Anschluß hieran werden Ihnen zwei Beispiele (in einem Programm) vorgeführt, die Ihnen einfach einmal die Funktionsweise des PLOT-Befehls darstellen sollen.

#### BEISPIEL:

```
100 REM #####
110 REM ##                ##
120 REM ##  DAS HERZLICHE  ##
130 REM ##  PLOT-BEISPIEL  ##
140 REM ##                ##
150 REM #####
160 REM
170 HIRES 8.9 : REM GRAPHIK EIN/PUNKTFARBE =
ORANGE/HINTERGRUND = BRAUN
```

```

180 FOR X=1 TO 319 STEP .5
190 PLOT X.60 * SIN(X/25) + 100 .1
200 NEXT X
210 PAUSE 5
220 HIRES 8.9 : REM GRAPHIK LOESCHEN
230 FOR Y=1 TO 0 STEP -1 : REM 1.) ZEICHNEN / 2.) LOESCHEN
240 FOR X=103 TO 290 STEP .5
250 PLOT 40 * SIN(X/30) + 160 . 50 * SIN(X/25) + 100 .Y
260 NEXT X
270 PAUSE 1
280 NEXT Y : REM WIEDER LOESCHEN
290 WAIT 198.255 : REM AUF TASTE WARTEN

```

Im ersten Teil des Beispiels wird Ihnen an einer einfachen Sinuskurve die Funktion des PLOT-Befehls erläutert. Verändern Sie ruhig einmal die Parameter!

Im zweiten Teil des Programms (ab Zeile 220) werden Sie mit einer interessanten Anwendung konfrontiert, die Ihnen gleichzeitig die Funktion des Zeichenmodus schildert (hier im Speicher y enthalten).

Zeile 290 wurde bereits an anderer Stelle beschrieben und beinhaltet den Befehl, der auf einen einfachen Tastendruck Ihrerseits wartet. Er wird im Folgenden häufiger gebraucht, da - wie erwähnt - der Rechner nach Beendigung des Programms stets in den Textmodus zurückkehrt und Ihnen damit die weitere Ansicht auf die Graphik verwehrt wird.

## 12.3.2 TEST

**FORMAT** : TEST (X.Y)  
**PARAMETER** : - X: X-KOORDINATE DES PUNKTES  
MC: 0-159 / HGR: 0-319  
- Y: Y-KOORDINATE DES PUNKTES  
MC: 0-199 / HGR: 0-199  
**BEISPIEL** : TEST (100.150)  
**FUNKTION** : TESTEN DER ANGEgebenEN BILD-  
SCHIRMKOORDINATE AUF EINEN  
GESETZTEN PUNKT

### ERLÄUTERUNGEN:

Angenommen, Sie schreiben ein Programm, das irgendwelche Punkte, sei es durch den eben beschriebenen PLOT-Befehl, sei es durch andere Graphikbefehle, auf den Bildschirm bringt. Im Laufe des Programms jedoch wollen Sie wissen, ob an einer ganz bestimmten Stelle auf dem Graphikbildschirm irgendwann einmal ein Punkt gesetzt wurde. Dies kann z.B. geschehen, um Schnittpunkte, Kollisionen (bei Spielen) oder abgegrenzte Bildschirmbereiche festzustellen.

Nun, die geforderten Aufgaben übernimmt der Befehl TEST. Hierbei bestimmen die in den Klammern gleichsam als Argument stehenden Parameter x und y in bekannter Art und Weise die Koordinaten des zu überprüfenden Punktes.

TEST wird wie eine arithmetische Funktion gehandhabt. Arithmetische Funktionen sind z.B. SIN, COS, SQR oder auch der Simon's Basic-Befehl LIN. Alle diese Befehle erscheinen nicht alleine, sondern in arithmetischen Ausdrücken, d.h. mit Hilfe dieser Funktionen wird ein Wert errechnet, der zu weiteren Rechnungen herangezogen oder direkt z.B. in eine Speicherzelle abgelegt werden kann:

**A = TEST(0.0)**  
**B = 3\*5 + TEST(0.10)**

Der Befehl TEST nun liefert nur einen von zwei möglichen Werten: 0 oder 1. Welchen Wert das Ergebnis letztendlich liefert, hängt davon ab, ob an der angesprochenen Stelle ein Punkt gesetzt ist oder nicht. Ist an der Stelle mit den



Koordinaten x,y ein Punkt gelöscht, so nimmt die Funktion den Wert 0 an. Im umgekehrten Falle jedoch, d.h. wenn dort ein Punkt gesetzt ist, erscheint der Wert 1. Mit diesen Werten wird dann, falls gewünscht in den folgenden Rechnungen weitergerechnet.

Mit diesem Befehl lassen sich vor allem schöne Spiele erzeugen. Aus diesem Grunde wurde im Anschluß an das Beispiel, das die Funktion dieses Kommandos erläutert ein kleines aber recht amüsantes Spielchen, das lediglich die vorhandenen Graphikbefehle ausnutzt, hintenangefügt. Probieren Sie es aus, es macht Spaß!

### BEISPIELE:

```
100 REM #####
110 REM ##                ##
120 REM ## TEST-BEISPIEL-1 ##
130 REM ##                ##
140 REM #####
150 REM
160 HIRES 6.7 : REM GRAPHIK EIN + LDESCHEN
170 T1 = TEST(100,100) : REM TESTEN VDR DEM SETZEN
180 PAUSE 1
190 PLDT 100,100,1 : REM PUNKT SETZEN
200 T2 = TEST(100,100) : REM TESTEN NACH DEM SETZEN
210 PAUSE 1
220 NRM : REM GRAPHIK AUS
230 PRINT CHR$(147) : REM BILDSCHIRM LDESCHEN
240 PRINT "TEST VDR DEM SETZEN:" T1
250 PRINT "TEST NACH DEM SETZEN:" T2
```

-----

```
100 REM #####
110 REM ##                ##
120 REM ## TEST-BEISPIEL-2 ##
130 REM ##                ##
140 REM #####
150 REM
```

```

160 REM #####
170 REM ##          ##
180 REM ##  SUPER-SNAKE  ##
190 REM ##          ##
200 REM #####
210 REM
220 COLOUR 10.1 : REM RAHMEN HELLROT
230 HIRES 1.2 : REM GRAPHIK EIN/PUNKTFARBE = WEISS /
HINTERGRUND = ROT
240 G = 4 : REM GESCHWINDIGKEIT
*****
250 X = -G : Y = 0 : REM STARTRICHTUNG: LINKS
260 A = 160 : B = 100 : REM STARTKOORDINATEN
270 GET AS$ : REM TASTE HOLEN
280 IF AS$ = "I" THEN X = 0 : Y = -G : REM HOCH
290 IF AS$ = "M" THEN X = 0 : Y = G : REM RUNTER
300 IF AS$ = "J" THEN X = -G : Y = 0 : REM LINKS
310 IF AS$ = "K" THEN X = G : Y = 0 : REM RECHTS
320 A = A+X : B = B+Y : REM WEITERBEWEGEN
330 IF A<0 OR A>319 OR B<0 OR B>199 THEN GOTO 370
340 IF TEST (A.B) = 1 THEN GOTO 370
350 PLOT A.B.1 : REM PUNKT SETZEN
360 GOTO 270
370 REM
380 REM #####
390 REM ##          ##
400 REM ##  TREFFERROUTINE  ##
410 REM ##          ##
420 REM #####
430 REM
440 FOR C=1 TO 15
450 FOR D=0 TO 15
460 COLOUR D.1 : REM RAHMEN BLINKEN
470 NEXT D
480 NEXT C
490 A = INT( RND(1) * 319/G ) * G
500 B = INT( RND(1) * 199/G ) * G
510 COLOUR 10.1
520 Z = Z+1 : REM ZAEHLER
530 IF Z = 20 THEN GOTO 550
540 GOTO 270

```

```

550 REM
560 REM      #####
570 REM      ##          ##
580 REM      ## ENDE ##
590 REM      ##          ##
600 REM      #####
610 REM
620 NRM : REM GRAPHIK AUS
630 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
640 PRINT AT(0,10) "" : REM CURSOR POSITIONIEREN
650 CENTRE "KEINE PANIK !!!"
660 POKE 198.0 : REM TASTEN LOESCHEN
670 WAIT 198.255 : REM AUF TASTE WARTEN

```

Das vorstehende Spiel funktioniert folgendermaßen:  
Auf dem Graphikbildschirm bewegt sich stets ununterbrochen eine immer länger werdende Schlange. Sie haben nun die Aufgabe mittels der Tasten

I für hoch  
M für runter  
J für links  
K für rechts

entsprechend der Tastenanordnung:



auf der Schreibmaschinentastatur die Bewegungsrichtung dieser Schlange so zu steuern, daß sie sich weder aus dem Bildschirmfenster hinaus bewegt, noch gegen einen anderen Teil Ihres Schwanzes stößt. In beiden Fällen bekommen Sie Punktabzug. Anschließend erscheint die Schlange wieder irgendwo auf dem Bildschirm. Bei 20 solchen Karambolagen ist das Spiel zuende. Sie haben natürlich die Möglichkeit, das Spiel nach Ihren Vorstellungen auszufeilen und zu verändern. Sie können es z.B. für zwei gegeneinander spielende Spieler umschreiben, Punkte zählen, statt der Tastatursteuerung die Joysticks zuhilfe nehmen und so weiter und so fort. Das Spiel

ist extra so eingerichtet, daß Sie Änderungen gut und bequem vornehmen können. Die Geschwindigkeit der Schlange beispielsweise können Sie in Zeile 240 leicht verändern (G enthält die Geschwindigkeit = Schrittweite pro Durchlauf). In den Zeilen 280 - 310 werden die vier Tasten geprüft und dementsprechend die Richtung geändert. Zeile 320 dann errechnet die neuen Koordinaten. Die Speicher X und Y bestimmen jeweils die Anzahl der Schritte pro Durchlauf in x- bzw. y-Richtung. A und B beinhalten die aktuellen Koordinaten des Punktes, die in Zeile 330 daraufhin geprüft werden, ob sie noch innerhalb des Graphikfensters liegen.

Das Interessante für uns an diesem Beispiel ist aber der Einsatz des TEST-Befehls in Zeile 340. Hier wird jeweils der aktuelle Punkt geprüft. Ist er gesetzt (TEST = 1), so wurde ein Teil des Schwanzes der Schlange getroffen. Ansonsten (TEST = 0) wird nun an dieser Stelle ein neuer Punkt gezeichnet.

Vielleicht regt dieses Beispiel Ihre Phantasie zu Hochflügen an, die bei Ihren Angehörigen oder Freunden hellen Anklang finden werden. Viel Spaß!

### 12.3.3 LINE

```
FORMAT      :LINE X1.Y1.X2.Y2.ZM
PARAMETER   : - X1: X-KOORDINATE DES
              STARTPUNKTES DER LINIE
              MC: 0-159 / HGR: 0-319
              - Y1: Y-KOORDINATE DES
              STARTPUNKTES DER LINIE
              MC: 0-199 / HGR: 0-199
              - X2: X-KOORDINATE DES
              ENDPUNKTES DER LINIE
              MC: 0-159 / HGR: 0-319
              - Y2: Y-KOORDINATE DES
              ENDPUNKTES DER LINIE
              MC: 0-199 / HGR: 0-199
              - ZM: ZEICHENMODUS
BEISPIEL    :LINE 0.0.310.199.1
FUNKTION     :ZEICHNEN EINER LINIE
```

#### ERLÄUTERUNGEN:

Nun, mit dem PLOT-Befehl könnten Sie nun rein theoretisch sämtliche Figuren zeichnen, die denkbar sind. Aber stellen Sie sich einmal vor, welchen Aufwand es darstellte und welche Zeit es in Anspruch nähme, wenn Sie z.B. allein mit dem PLOT-Befehl eine Linie zeichnen wollten. Sie müßten umständlich jeden einzelnen Punkt der Linie (nach der 2-Punktegleichung) in Basic berechnen und dann setzen. Wie lange das dauert, können Sie sich vorstellen!

Um diesem Hindernis Abhilfe zu verschaffen, wurden in Simon's Basic eine ganze Reihe von Befehlen entwickelt, die gleich fertige Figuren zeichnen. Einer davon ist der sogenannte LINE-Befehl. Er ermöglicht es Ihnen in einfachster Weise, eine Linie darzustellen, deren zwei Begrenzungspunkte von Ihnen angegeben werden können. Sie brauchen sich um nichts weiter zu kümmern, der Rest wird von Simon's Basic erledigt (Sie werden zwar sehen, daß alle Befehle zur Erzeugung von Figuren im Simon's Basic nicht besonders schnell sind, dafür aber gibt es ja schließlich - wie schon erwähnt - spezielle Graphikprogramme, die hier den Anspruchsvollen befriedigen.). So können Sie leichter und schneller über die verschiedensten Funktionen verfügen, die im Mittelpunkt der Graphikgestaltung

stehen.

Die ersten zwei Parameter (x1 und y1) stellen bei LINE nun die Koordinaten dar, von denen aus eine Linie gezeichnet werden soll (= 1. Punkt), gefolgt von x2 und y2, die weiterhin den Endpunkt der Linie festlegen (= 2. Punkt). Dabei ist es nicht immer egal, welcher Punkt nun Anfangs- oder Endpunkt ist. Haben Sie beispielsweise von den Koordinaten 100,110 eine Linie nach 200,150 gezeichnet und wollen diese nun löschen, so müssen Sie diese Linie in der gleichen Richtung löschen, wie Sie sie gezeichnet haben. Ansonsten wird Ihre Linie nur gepunktet gelöscht (nur Teile der Linie werden gelöscht), was dann wohl wahrscheinlich nicht in Ihrem Interesse liegt, zumal dieses gepunktete Löschen nicht einmal für alle Linien gilt, da der Grad des Löschens von dem Winkel der Linie mit der horizontalen abhängt. Also:

10 HIRES 2.3

20 LINE 100.110.200.150.1

30 PAUSE 1

40 LINE 200.150.100.110.0 : REM FALSCHI

50 WAIT 198.255

DAGEGEN:

10 HIRES 2.3

20 LINE 100.110.200.150.1

30 PAUSE 1

40 LINE 100.110.200.150.0 : REM RICHTIG!

50 WAIT 198.255

Ähnliches gilt natürlich für das Invertieren.

Mit LINE lassen sich schon recht schöne Effekte erzeugen, wofür in den folgenden Beispielen einige Anregungen gegeben werden.

**BEISPIELE:**

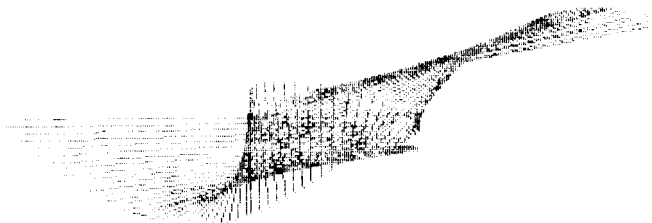
```
100 REM #####
110 REM ##                ##
120 REM ##  LINE-BEISPIEL  ##
130 REM ##                ##
140 REM #####
150 REM
160 REM  #####
170 REM  ##                ##
180 REM  ##  FIGUR 1  ##
190 REM  ##                ##
200 REM  #####
210 REM
220 HIRES 6.7
230 FOR X=1 TO 319 STEP 4
240 LINE 0,0,X,40 * SIN(X/20) + 100,1
250 NEXT X
260 PAUSE 10
270 HIRES 6.7
280 REM
290 REM  #####
300 REM  ##                ##
310 REM  ##  FIGUR 2  ##
320 REM  ##                ##
330 REM  #####
340 REM
350 FOR X=1 TO 350 STEP 1
360 A = 50 * SIN(X/10) + 160
370 B = 60 * SIN(X/20) + 100
380 C = 70 * SIN(X/30) + 160
390 D = 80 * SIN(X/40) + 100
400 LINE A,B,C,D,1
410 NEXT X
420 PAUSE 10
430 HIRES 6.7
440 REM
```

```

450 REM      #####
460 REM      ##           ##
470 REM      ##  FIGUR 3  ##
480 REM      ##           ##
490 REM      #####
500 REM
510 FOR X=1 TO 319 STEP 4
520 A = X
530 B = 60 * SIN(X/60) + 100
540 C = 40 * SIN(X/25) + 160
550 D = 20 * SIN(X/30) + 100
560 LINE A,B,C,D,1
570 NEXT X
580 WAIT 198.255 : REM AUF TASTE WARTEN

```

Wie Sie sehen, lassen sich alle Parameter nicht nur einfach angeben, sondern können auch durch mehr oder weniger komplizierte Rechnungen errechnet werden. Sie brauchen die einzelnen oben verwendeten Formeln nicht unbedingt zu verstehen. Trotzdem können Sie nach Herzenslust an Ihnen Änderungen vornehmen. Schon kleine Änderungen können das ganze Bild völlig verändern. Mit der Zeit bekommen Sie sicher (besonders, wenn Sie sich etwas in der Mathematik auskennen) einige Tricks heraus, wie Sie eine Figur auch zielgerichtet verändern oder entwerfen können, und wie sich welche Veränderungen höchstwahrscheinlich auswirken (Im ersten Beispiel wurde z.B. jeder Punkt einer einfachen Sinuskurve mit dem Punkt 0,0 verbunden.).





## 12.3.4 REC

**FORMAT** : REC X.Y.B.H.ZM  
**PARAMETER** : - X : X-KOORDINATE DER OBEREN  
LINKEN ECKE DES RECHTECKS  
MC: 0-159 / HGR: 0-319  
- Y : Y-KOORDINATE DER OBEREN  
LINKEN ECKE DES RECHTECKS  
MC: 0-199 / HGR: 0-199  
- B : BREITE DES RECHTECKS  
MC: 0-159 / HGR: 0-319  
- H : HOHE DES RECHTECKS  
MC: 0-199 / HGR: 0-199  
- ZM: ZEICHENMODUS  
**BEISPIEL** : REC 20.40.100.50.1  
**FUNKTION** : ZEICHNEN EINES RECHTECKS

### ERLÄUTERUNGEN:

Ein weiterer, fertiger Befehl zum Erzeugen von Figuren ist das REC-Kommando. Mit Hilfe dieses Kommandos ist es Ihnen möglich, ein Rechteck variabler Form, Größe und Position auf den Graphikbildschirm zu bringen. Sie besitzen dadurch ein Werkzeug, das Ihnen hilft, z.B. Rahmen um bestimmte Graphiken (Text in der Graphik (s.u.) etc.) zu ziehen, effektvolle Rechtecksgraphiken oder einfach Rechteckfiguren zu creieren. Die Position des gewünschten Vierecks geben Sie einfach durch die Koordinaten der oberen linken Ecke unserer Figur an. Diese Ecke definiert gleichzeitig zwangsläufig die Ausdehnung nach oben und nach links. Um nun die Größe des Rechtecks und damit seine Ausdehnung nach unten und rechts zu bestimmen, hängen wir an die gerade besprochenen ersten zwei Parameter des REC-Befehls zwei weitere an, die dem Rechner die Breite (b) und die Höhe (h) unserer Figur angeben. Dabei werden jeweils alle Kanten des Rechtecks mitgezählt. Selbstverständlich können diese Werte die gleiche Größe annehmen, wie die vorstehenden Koordinaten. Das ist z.B. notwendig, wenn Sie etwa das gesamte Graphikfenster mit

**REC 0.0.319.199.1**

umrahmen wollen.

Auf eines sollten Sie jedoch achten, falls Sie den Effekt nicht absichtlich provozieren wollen: Geben Sie keine Höhe

oder Breite eines Rechtecks an, das rein theoretisch außerhalb des Graphikfensters läge. Wie z.B.:

### **REC 100.150.50.80.1**

Wie Sie sehen, falls Sie dieses Beispiel ausprobiert haben (Natürlich müssen Sie zunächst mit HIRES auf Graphik umschalten und dann z.B. mit WAIT 198,255 im Programmmodus warten, um das Bild betrachten zu können.), wird das Rechteck an der einen (rechten) Seite korrekt nur bis zum Bildschirmrand gezogen. An der anderen (linken) Seite jedoch wird die volle Höhe gezeichnet, die dann natürlich über den Ursprungspunkt des Rechtecks (obere linke Ecke) herausragt. Ähnliches gilt dann ebenfalls für den Fall einer zu groß gewählten Breite.

Die folgenden Beispiele geben Ihnen wie immer Tips und Tricks zur REC-Anwendung:

### **BEISPIELE:**

Sie haben natürlich die Möglichkeit, einen Rahmen mit einer größeren Dicke zu zeichnen, als die konstante Dicke von 1 Punkt. Die folgende Routine soll Ihnen diese Variation des REC-Befehls veranschaulichen. Die gewünschte Dicke wird zuvor (hier in Zeile 170) - neben den anderen notwendigen Parametern - in dem Speicher DI abgelegt. Der dazu erforderliche Algorithmus (Rechenvorschrift), der als Unterprogramm in jedes Ihrer Programme eingebaut werden kann, sähe dann wie folgt aus (ab Zeile 280):

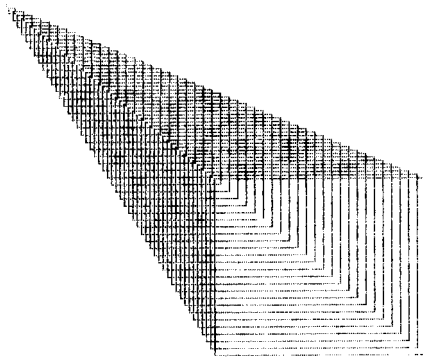
```
100 REM #####
110 REM ##                                     ##
120 REM ## VARIABLE RAHMENDICKE ##
130 REM ##                                     ##
140 REM #####
150 REM
160 HIRES1.0
170 DI = 9 : X = 100 : Y = 50 : B = 90 : H = 80 : REM
EINGABEPARAMETER-BEISPIEL
180 GOSUB 280 : REM ROUTINENAUFRAF
190 WAIT 198.255
200 END
210 REM
```

```

220 REM #####
230 REM ##          ##
240 REM ##  UNTERPROGRAMMSTART  ##
250 REM ##          ##
260 REM #####
270 REM
280 FOR Z=0 TO 01-1
290  REC X+Z, Y+Z, B - 2*Z, 80 - 2*Z, 1
300 NEXT Z
310 RETURN : REM UNTERPROGRAMMENDE

```

Sie sehen, das Ergebnis kann sich sehen lassen. Wie immer sollten Sie die einzelnen Parameter ändern und abwandeln (z.B. DI).



Im Anschluß hieran einige Effekte, die sich mit dem Rechteckbefehl erreichen lassen:

```

100 REM #####
110 REM ##          ##
120 REM ##  REC-BEISPIEL  ##
130 REM ##          ##
140 REM #####
150 REM
160 REM #####
170 REM ##          ##
180 REM ##  FIGUR 1  ##

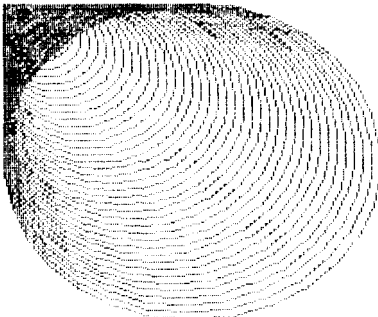
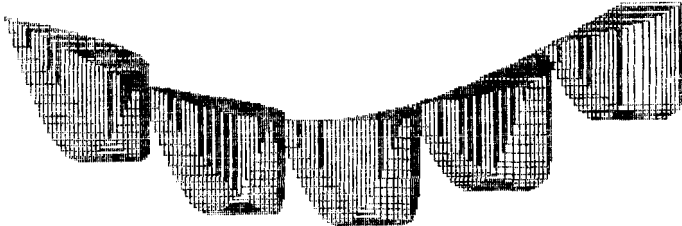
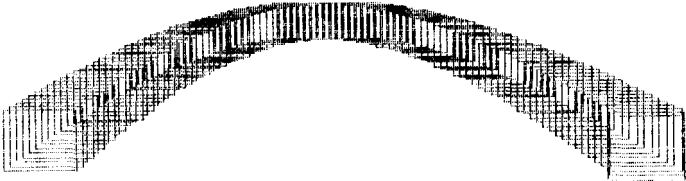
```

```

190 REM  ##          ##
200 REM  #####
210 REM
220 HIRES 3.4
230 FOR X=1 TO 99 STEP 2
240 REC X.X.X.X.1
250 NEXT X
260 PAUSE 10
270 HIRES 3.4
280 REM
290 REM  #####
300 REM  ##          ##
310 REM  ##  FIGUR 2  ##
320 REM  ##          ##
330 REM  #####
340 REM
350 FOR X=1 TO 290 STEP 3
360 A = X
370 B = 60 * SIN(X/90+3.14) + 100
380 C = ABS(B/4)+10
390 O = C
400 REC A.B.C.O.1
410 NEXT X
420 PAUSE 10
430 HIRES 3.4
440 REM
450 REM  #####
460 REM  ##          ##
470 REM  ##  FIGUR 3  ##
480 REM  ##          ##
490 REM  #####
500 REM
510 FOR X=1 TO 290 STEP 2
520 A = X
530 B = 60 * SIN(X/90) + 50
540 C = ABS(30 * SIN(X/20))
550 O = C*2
560 REC A.B.C.O.1
570 NEXT X
580 WAIT 198.255

```

Lassen Sie sich nicht von den vielen komplizierten Rechnungen abschrecken. Sie brauchen das nicht alles zu verstehen. Ich hoffe, die Programme sprechen für sich.



## 12.3.5 BLOCK

**FORMAT** :BLOCK X1.Y1.X2.Y2.ZM  
**PARAMETER** : - X1: X-KOORDINATE DER OBEREN  
LINKEN ECKE DES FELDES  
MC: 0-159 / HGR: 0-319  
- Y1: Y-KOORDINATE DER OBEREN  
LINKEN ECKE DES FELDES  
MC: 0-199 / HGR: 0-199  
- X2: X-KOORDINATE DER UNTEREN  
RECHTEN ECKE DES FELDES  
MC: 0-159 / HGR: 0-319  
- Y2: Y-KOORDINATE DER UNTEREN  
RECHTEN ECKE DES FELDES  
MC: 0-199 / HGR: 0-199  
- ZM: ZEICHENMODUS  
**BEISPIEL** :BLOCK 100,150,160,200,1  
**FUNKTION** :ZEICHNEN EINES AUSGEFÜLLTEN  
RECHTECKS (FELD)

### ERLÄUTERUNGEN:

Nehmen Sie einmal an, Sie wollten ein Statistikprogramm erstellen und dazu einige Balkendiagramme anfertigen können. Nun, bisher ständen Sie auf dem Schlauch. Sie haben zwar die Möglichkeit, mittels der unter REC als erstes Beispiel angegebenen Routine einen solchen Balken zu erstellen, indem Sie einfach den dort erstellten Rahmen so dick gestalten, daß sich zwei Rahmenseiten berühren. Wie Sie sich jedoch denken können, wird hier sehr viel Zeit vergeudet. Zudem benötigen Sie in Basic hierzu ein ganzes Unterprogramm, was Sie wiederum einigen Speicherplatz kostet. Kurz, wir fordern einen Befehl, der ein solches ausgefülltes Rechteck (Feld) leicht und bequem zeichnet. Diese Forderung wird durch das Einführen des sogenannten BLOCK-Befehls erfüllt (Sie werden später (noch in diesem Kapitel) sehen, daß es noch eine dritte Möglichkeit gibt, indem man ein Rechteck mit REC zeichnet und anschließend mit PAINT ausfüllt, was jedoch ebenfalls recht zeitintensiv und natürlich komplizierter ist.).

Wie ist nun dieses BLOCK-Kommando anzuwenden?

Dies ist wirklich nicht allzu schwierig: Man nehme die obere

linke Ecke des gewünschten Feldes und setze ihre Koordinaten an die erste Stelle der insgesamt fünf Parameter dieses Befehls als Begrenzung nach links und nach oben (bekannt von dem vorherigen REC). Anschließend einige man sich über die Größe und Gestalt unseres Feldes und setze nun die Koordinaten der unteren linken Ecke der Figur als dritte und vierte Parameter hintenan.

Achtung! Hier besteht (leider) ein gravierender Unterschied zum REC-Befehl. In letzterem geben Sie nicht, wie hier, die Koordinaten der eben dargestellten zweiten Ecke, sondern die Breite und Höhe des Rechtecks direkt an, ein Manko, das hätte vermieden werden können. Nun wird man Probleme bei dem gemeinsamen Gebrauch von REC und BLOCK haben, die Sie vielleicht jedoch mit ein wenig Überlegung meistern könnten. Wie immer gibt der bereits erläuterte Zeichenmodus die Art der Darstellung an.

Eine Besonderheit des BLOCK-Befehls jedoch muß hier noch erwähnt werden. Wenn Sie versuchen, die Farbe, mit der Sie einen Block zeichnen wollen, durch LOW COL zu verändern, so werden Sie wenig Erfolg haben. Dieser Modus, in dem bekanntlich mit Farbsetzung gezeichnet und der mit diesem Befehl (LOW COL) eingeschaltet wird, wird von dem BLOCK-Befehl ignoriert, d.h. Sie können mit BLOCK lediglich in den Farben zeichnen, die Sie im Anfang mit HIRES gewählt oder durch ein Zeichnen unter LOW COL verändert haben. Falls Sie Wert auf diese Farbgebung legen, so müssen Sie wohl oder übel doch wieder auf die oben erwähnte Routine mittels REC zurückgreifen. Nichtsdestotrotz besitzen Sie durch BLOCK ein schönes Mittel, um schnell und elegant Felder auf Ihren Bildschirm zu bringen. Anwendungen folgen auf dem Fuße:

#### **BEISPIELE:**

**ALS ERSTES FINDEN SIE EINE WEITERE MÖGLICHKEIT, EINEN RAHMEN DEFINIERTER DICKE AUF DEM BILDSCHIRM DARZUSTELLEN. DANN FOLGEN EINIGE WEITERE ANREGUNGEN.**

```
100 REM #####
110 REM ##                ##
120 REM ##  BLOCK-BEISPIEL  ##
130 REM ##                ##
140 REM #####
```

```

150 REM
160 REM      #####
170 REM      ##           ##
180 REM      ##  RAHMEN  ##
190 REM      ##           ##
200 REM      #####
210 REM
220 HIRES 7.0 : REM GRAPHIK EIN
230 REM RAHMEN
240 BLOCK 110,60,220,120,1 : REM AEUSSERES FELD ZEICHNEN
250 BLOCK 120,70,210,110,0 : REM INNERES FELD LOESCHEN
260 PAUSE 10
270 REM
280 REM #####
290 REM ##           ##
300 REM ##  INVERTIERUNG  ##
310 REM `##           ##
320 REM #####
330 REM
340 HIRES 7.0 : REM GRAPHIK LOESCHEN
350 FOR X=1 TO 280 STEP 20
360 BLOCK X, 50, X*1.2, 100 * SIN(X/80) + 50, 2 : REM
INVERTIERUNG DER BLOECKE
370 NEXT X : REM NAECHSTER BLOCK
380 PAUSE 10
390 REM
400 REM #####
410 REM ##           ##
420 REM ##  MULTICOLOR  ##
430 REM ##           ##
440 REM #####
450 REM
460 HIRES 7.0
470 COLOUR 2.0 : REM RAHMEN ROT / HINTERGRUNDFARBE = SCHWARZ
480 MULTI 3,4,5 : REM MULTICOLOR EINSCHALTEN
490 BLOCK 5, 20,150,190,1 : REM FARBE 1
500 BLOCK 45, 70,150,190,2 : REM FARBE 2
510 BLOCK 85,120,150,190,3 : REM FARBE 3
520 BLOCK 20, 50,100,150,4 : REM INVERTIEREN
530 WAIT198,255 : REM AUF TASTE WARTEN

```



## 12.3.6 CIRCLE

**FORMAT** :CIRCLE MX.MY.RX.RY.ZM

**PARAMETER** : - MX: X-KOORDINATE DES MITTEL-  
PUNKTES DER ELLIPSE  
(KREIS)  
MC: 0-159 / HGR: 0-319

- MY: Y-KOORDINATE DES MITTEL-  
PUNKTES DER ELLIPSE  
(KREIS)  
MC: 0-199 / HGR: 0-199

- RX: RADIUS DER ELLIPSE  
(KREIS) IN X-RICHTUNG  
(SENKRECHT)  
MC: 0-159 / HGR: 0-319

- RY: RADIUS DER ELLIPSE  
(KREIS) IN Y-RICHTUNG  
(WAAGERECHT)  
MC: 0-199 / HGR: 0-199

- ZM: ZEICHENMODUS

**BEISPIEL** :CIRCLE 150.80.60.30.1

**FUNKTION** :ZEICHNEN EINER ELLIPSE  
(EINES KREISES)

### ERLÄUTERUNGEN:

Ja, hört denn das überhaupt nicht mehr auf? Man schreibt sich hier die Finger wund und kommt und kommt nicht zum Ziel. Ja, Sie werden staunen, Simon's Basic kann eine ganze Menge. Hier z.B. haben wir es mit einem Befehl zu tun, der Ihnen eine Menge Arbeit abnimmt und es Ihnen ermöglicht, anspruchsvolle Graphiken zu erstellen. Mit diesem Befehl werden Sie in Zukunft Kreise und Ellipsen in Massen auf Ihren Schirm zaubern. Einfach, indem Sie dem Rechner einige Parameter übergeben, die die Ellipse näher bestimmen.

Zunächst einmal seien hier die geometrischen Grundlagen dargestellt, die Sie in diesem Zusammenhang über Kreise und Ellipsen wissen sollten:

Als allererstes gibt es verschiedene Arten, eine Ellipse zu definieren. Im Simon's Basic wurde einfachheitshalber die Mittelpunktsdarstellung der Ellipse verwendet, d.h. eine Ellipse besitzt genau einen Mittelpunkt. Dieser Mittelpunkt

besitzt jeweils zu gegenüberliegenden Punkten auf der Ellipse den gleichen Abstand (Radius). Wichtig für die Definition unserer Ellipse sind nun die zwei besonderen Radien in senkrechter und in waagerechter Richtung. D.h. ziehen wir einen senkrechten Strich vom Mittelpunkt der Ellipse zu Ihrem Rand, so erhalten wir den ersten (senkrechten) Radius. Ziehen wir dagegen einen waagerechten Strich vom Mittelpunkt zum Rand, so erhalten wir den zweiten (waagerechten) Radius. Diese beiden Radien sind bei einer echten Ellipse stets verschieden. Nun wissen wir auch etwas mit dem Kreis anzufangen. Der Kreis ist einfach nur ein Sonderfall einer Ellipse: Wenn beide Radien gleich sind, so haben wir es nicht mehr mit einem abgeplatteten Gebilde zu tun, wie es eine Ellipse darstellt. Jetzt haben wir das, was wir "den" Radius eines Kreises nennen.

Nun, auf ähnliche Weise wird im CIRCLE-Befehl die Ellipsendarstellung verwirklicht. Sie geben zu allererst die Koordinaten des besagten Mittelpunktes Ihrer gewünschten Ellipse an (mx,my). Jetzt bestimmen wir die Größe und das Maß der "Abplattung" unserer Ellipse, indem wir den x- (= senkrechten) und y- (= waagerechten) Radius übergeben (rx,ry). Haben wir dies getan, so fügen wir noch den Zeichenmodus hinten an, und schon geht die Sache los.

Wollen Sie einen Kreis zeichnen, so sollten Sie sich erst einmal vergegenwärtigen, ob Sie sich im Multicolormodus oder in HGR befinden. In HGR brauchen Sie hierzu nur die beiden Radien gleich groß werden lassen. In MC dagegen ist ja jeder Punkt bekanntlich doppelt so breit wie hoch. Hier sollten Sie also den x-Radius halb so groß wählen wie den y-Radius (es können hier von Fernseher zu Fernseher kleine Differenzen bzw. Verzerrungen auftreten, die Sie entweder vernachlässigen können, oder Sie errechnen sich die richtigen Korrekturfaktoren durch einfache Messung am Fernseher mittels eines Zentimetermaßes; die diesbezüglichen Angaben im offiziellen Simon's Basic Handbuch dürfen Sie deshalb vergessen).

Sollte Ihr Kreis einmal über den Bildschirm hinausragen, so wird bei kleinen Überschreitungen einfach ein Strich vom zuletzt ins Bild passenden Punkt zum zuerst wieder auftauchenden gezogen. Bei großen Überträgen entsteht mehr oder weniger Chaos.

## BEISPIELE:

Das erste Beispiel ist ein Übungsbeispiel, das Ihnen helfen soll, den Befehl CIRCLE besser zu verstehen. Dies scheint aufgrund der vielen Parameter, die Sie zu diesem Befehl angeben müssen, angezeigt. Übertragen Sie dieses Beispiel ruhig vollständig mit REM - Zeilen. Sie können so leicht die einzelnen Funktionen der Parameter überblicken. Nun ändern Sie fleißig und schauen Sie, was herauskommt.

```
100 REM #####
110 REM ##                ##
120 REM ## CIRCLE-BEISPIEL-1 ##
130 REM ##                ##
140 REM #####
150 REM
160 HIRES 7.8 : REM GRAPHIK EIN
170 REM TESTBEISPIEL:
180 REM BITTE AENDERN SIE DIE PARAMETER!
190 REM
200 REM      MX! MY! RX! RY!ZM
210 REM  ----+----+----+----+----
220 REM      !  !  !  !
230 CIRCLE 160,100, 70, 90, 1
240 REM  ----+----+----+----+----
250 WAIT 198,255 : REM AUF TASTE WARTEN
```

-----

```
100 REM #####
110 REM ##      NOCH EIN      ##
120 REM ## CIRCLE-BEISPIEL ##
130 REM ##                ##
140 REM #####
150 REM
160 REM      #####
170 REM      ##                ##
180 REM      ## TRICHTER ##
190 REM      ##                ##
200 REM      #####
210 REM
220 HIRES 3.2 : REM GRAPHIK EIN
```

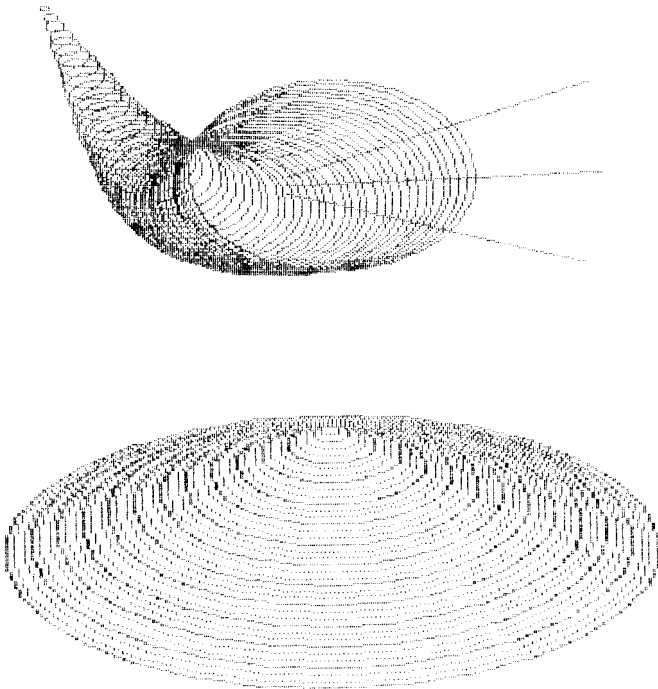
```

230 COLOUR 2.2 : REM RAHMEN = ROT
240 FOR X=1 TO 90 STEP 2
250 CIRCLE X+10.X+10.X.X.1
260 NEXT X
270 PAUSE 20
280 REM
290 REM #####
300 REM ##          ##
310 REM ## ALPHORN ##
320 REM ##          ##
330 REM #####
340 REM
350 HIRES 3.2
360 FOR X=1 TO 110 STEP 2
370 CIRCLE 1.3*X + 10. 50 * SIN(X/30 -.2) + X + 10. X/2 +
X/8. X/2. 1
380 NEXT X
390 LINE 130.105.270. 50.1
400 LINE 130.110.280.100.1
410 LINE 130.115.270.150.1
420 PAUSE 20
430 REM
440 REM #####
450 REM ##          ##
460 REM ## MUSCHEL ##
470 REM ##          ##
480 REM #####
490 REM
500 HIRES 3.2
510 COLOUR 12.12 : REM HINTERGRUND + RAHMEN GRAU 2
520 MULTI 6.7.7 : REM MULTICOLOR AN
530 FOR X=1 TO 78 STEP 2
540 T = T+1 : REM FARBREGISTER WECHSELN
550 LOW COL F.F+1.F+2 : REM FARBWECHSEL
560 CIRCLE 80. 0.92*X+10. X. X. T
570 IF T=3 THEN T = 0 : F = F+3 : IF F=16 THEN F = 0 : REM
FARBE ERHOEHEN
580 NEXT X
590 PAUSE 5
600 REM #####
610 REM BLINKEN:

```

```
620 REM !!!!!!!!!!!!!
630 FOR X=0 TO 15
640 COLOUR X.X : REM HINTERGRUNDFARBE + RAHMEN WECHSELN
650 PAUSE 4
660 NEXT X
670 COLOUR 9.9
680 WAIT 198.255 : REM AUF TASTE WARTEN
```

Bitte stören Sie sich nicht daran, wenn die Ausführung dieser Beispiele etwas dauert. Der im Simon's Basic eingebaute CIRCLE-Befehl ist zugegebenermaßen nicht so schnell wie in anderen Graphikprogrammen (siehe Supergraphik). Trotzdem werden Sie gerade mit Ihm sehr viel Freude haben.



## 12.3.7 ARC

**FORMAT** : ARC MX,MY,SW,EW,A,RX,RY,ZM

**PARAMETER** :

- MX: X-KOORDINATE DES MITTEL-  
PUNKTES DER ELLIPSE,  
DEREN BOGEN GEZEICHNET  
WERDEN SOLL  
MC: 0-159 / HGR: 0-319
- MY: Y-KOORDINATE DES MITTEL-  
PUNKTES DER ELLIPSE,  
DEREN BOGEN GEZEICHNET  
WERDEN SOLL  
MC: 0-199 / HGR: 0-199
- SW: STARTWINKEL DES BOGENS  
(0-360 ALTGRAD)
- EW: ENDWINKEL DES BOGENS  
(0-360 ALTGRAD)
- A : WINKELABSTAND ZWISCHEN  
ZWEI ZU BERECHNENDEN  
BOGENPUNKTEN (S. TEXT)  
(0-360 ALTGRAD)
- RX: RADIUS DER ELLIPSE  
(KREIS) IN X-RICHTUNG  
(SENKRECHT), DEREN BOGEN  
GEZEICHNET WERDEN SOLL  
MC: 0-159 / HGR: 0-319
- RY: RADIUS DER ELLIPSE  
(KREIS) IN Y-RICHTUNG  
(WAAGERECHT), DEREN BOGEN  
GEZEICHNET WERDEN SOLL  
MC: 0-199 / HGR: 0-199
- ZM: ZEICHENMODUS

**BEISPIEL** : ARC 160,100,20,120,20,50,40,1

**FUNKTION** : ZEICHNEN DES BOGENS EINER  
ELLIPSE (EINES KREISES) JEWEILS  
IN EFINIERTEN SCHRITTEN

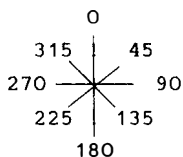
### ERLÄUTERUNGEN:

Neben CIRCLE, mit dem Sie sich vor der Lektüre dieses Abschnittes beschäftigen sollten, gibt es noch zwei weitere Befehle, die sich mit verschiedenen Aspekten der Ellipsen

beschäftigen. Der erste dieser beiden zeichnet nur Teile einer Ellipse, sogenannte Ellipsenbögen. Ähnlich, wie uns der Begriff Halb- oder Viertelkreis bekannt sein dürfte, gibt es hier z.B. Halb-, Viertel-, Achtelellipsen und so weiter. Gleichzeitig besitzt dieser Befehl jedoch noch eine weitere Besonderheit, auf die wir später zurückkommen wollen.

Zunächst einmal: Was ist nötig, um nun einen Bogen zu definieren? Nun, als Allererstes muß natürlich einmal bekannt sein, zu welcher Ellipse (Kreis) wir einen Bogen zeichnen wollen. Damit stehen wir vor der Aufgabe, auf jeden Fall zunächst sämtliche Parameter anzugeben, die diese Ellipse (von der ja nur ein Teil gezeichnet werden soll) eindeutig bestimmen, d.h. es werden alle Werte von uns verlangt, die uns schon von CIRCLE her bekannt sind (Mittelpunkt - Koordinaten (mx,my) und x- und y-Radius (rx,ry)). Die Anordnung dieser Parameter wird Ihnen in der obigen Formatbeschreibung dargelegt.

Daneben jedoch müssen wir nun auch noch die Bogengröße eingeben. Wie Sie wissen, kann man jeden Kreis (und natürlich auch jede Ellipse) in 360 Abschnitte oder Grad unterteilen. Im Simon's Basic liegt hierbei der Nullpunkt (also 0 Grad) an dem obersten Punkt unserer Ellipse. Von hier aus werden nun im Uhrzeigersinn 360 Grad abgetragen, sodaß wir letztendlich wieder am Ausgangspunkt ankommen. Wollen wir nun den Bogen bestimmen, so geben wir die Gradzahl an, bei der dieser Bogen beginnen soll, um seine Lage zu definieren (sw), und dann senden wir die Gradzahl, bei der er enden soll (ew) (Eine Halbellipse z.B. mit Ihrem Startpunkt am obersten Punkt der Figur können wir also mit den Parametern sw=0 und ew=180 zeichnen). Das folgende Schema soll Ihnen die Gradeinteilung verdeutlichen:



Damit hätten wir eigentlich alles angegeben, was zum Zeichnen eines Ellipsenbogens notwendig ist. Simon's Basic jedoch geht hier weiter. Neben all diesen Dingen können Sie noch einen

weiteren Parameter (a) angeben. Es ist ja logisch, daß zum Zeichnen einer Ellipse eine ganze Menge Punkte berechnet werden müssen. Normalerweise geht Simon's Basic so vor, daß es nacheinander die notwendigen Punkte in einer bestimmten Schrittlänge errechnet und diese dann durch einen Strich miteinander verbindet, sodaß ein zusammenhängendes Gebilde resultiert. Diese Schrittlänge ist bei CIRCLE so gewählt, daß ein annehmbarer Kompromiss zwischen Auflösung (je weiter die errechneten Punkte auseinander liegen, desto zackiger wird die Ellipse) und Rechengeschwindigkeit (je mehr Punkte errechnet werden, desto langsamer wird gezeichnet) getroffen wird. Es wird also in Wirklichkeit ein Vieleck gezeichnet, das uns als Ellipse erscheint.

Der ARC-Befehl nun erlaubt Ihnen, diesen Punktabstand frei zu wählen, d.h. Sie bestimmen den Grad der Auflösung. Gleichzeitig aber erhalten Sie die Möglichkeit, ein beliebiges gleichseitiges Vieleck oder Teile dieses zu zeichnen. Dabei gibt der hierfür zuständige Parameter a (für Abstand) jeweils den Abstand zweier Punkte in Grad (entsprechend obiger Ausführungen) an. Eine normale, mit CIRCLE gezeichnete Ellipse wird dabei mit a=12 gezeichnet, d.h. wenn der erste Punkt bei 0 Grad liegt, wird der nächste für 12, dann für 24 Grad und so weiter errechnet. Die beiden folgenden Befehle entsprechen sich in Ihrem Ergebnis also:

- ARC 160.100.0.360.12.50.70.1
- CIRCLE 160.100. 50.70.1

Dementsprechend haben sie die Möglichkeit z.B. verschiedene Mehrecke zu zeichnen. Die folgende Tabelle gibt Ihnen dabei einige Hilfen:



A	!	FIGUR
====!	=====	
120	!	3-ECK
90	!	4-ECK
72	!	5-ECK
60	!	6-ECK
51	!	7-ECK
45	!	8-ECK
40	!	9-ECK
36	!	10-ECK

Aus dieser Tabelle können Sie die Werte für a entnehmen, die für die einzelnen angegebenen Figuren zu wählen sind. Wollen Sie ein vollständiges n-eck zeichnen, so müssen Sie gleichzeitig sw=0 und ew=360 setzen. Doch das alles sollten Sie an dem Übungsbeispiel ausprobieren!

**BEISPIELE:**

Das folgende Programm ist wieder ein Übungsbeispiel, an dem Sie die Auswirkungen der einzelnen Parameter testen können (verwenden Sie ruhig einmal die Werte aus der obigen Figurentabelle).

```

100 REM #####
110 REM ##          ##
120 REM ##  ARC-BEISPIEL-1  ##
130 REM ##          ##
140 REM #####
150 REM
160 HIRES 7.8 : REM GRAPHIK EIN
170 REM TESTBEISPIEL:
180 REM BITTE AENDERN SIE DIE PARAMETER!
190 REM
200 REM MXI MYI SWI EWI AI RXI RYIZM
210 REM ---+---+---+---+---+---+---+---+---+---+
220 REM  !  !  !  !  !  !  !  !
230 ARC 160,100, 0.360, 45, 70, 90, 1
240 REM ---+---+---+---+---+---+---+---+---+
250 WAIT 198.255 : REM AUF TASTE WARTEN

```

```

100 REM #####
110 REM ##          ##
120 REM ##  ARC-BEISPIEL-2  ##
130 REM ##          ##
140 REM #####
150 REM
160 REM  #####
170 REM  ##          ##
180 REM  ##  FIGUR 1  ##
190 REM  ##          ##
200 REM  #####
210 REM
220 HIRES 3.9 : REM GRAPHIK EIN
230 COLOUR 9.9 : REM RAHMEN = BRAUN
240 FOR X=1 TO 90 STEP 2
250 ARC X+10, X+10, X, 2*X + 90, X, X, X, 1
260 NEXT X
270 PAUSE 20
280 REM
290 REM  #####
300 REM  ##          ##
310 REM  ##  FIGUR 2  ##
320 REM  ##          ##
330 REM  #####
340 REM
350 HIRES 3.9
360 FOR X=1 TO 160 STEP 2
370 ARC 1.3*X + 10, 50 * SIN(X/30 -.2) + X+10, X, 190+X,
30+X, X/2 + X/8, X/2, 1
380 NEXT X
390 PAUSE 20
400 REM
410 REM  #####
420 REM  ##          ##
430 REM  ##  FIGUR 3  ##
440 REM  ##          ##
450 REM  #####
460 REM
470 HIRES 3.9
480 COLOUR 12.12 : REM HINTERGRUND + RAHMEN GRAU 2
490 MULTI 6.7.7 : REM MULTICOLOR AN

```

```

500 FOR X=1 TO 78 STEP 2
510 T = T+1 : REM FARBREGISTER WECHSELN
520 LOW COL F.F+1.F+2 : REM FARBWECHSEL
530 ARC 80, 0.92*X+10, 90+X, 270+X, 60, X, X, T
540 IF T=3 THEN T = 0 : F = F+3 : IF F=16 THEN F = 0 : REM
FARBE ERHOEHEN
550 NEXT X
560 PAUSE 5
570 REM #####
580 REM BLINKEN:
590 REM #####
600 FOR X=0 TO 15
610 COLOUR X.X : REM HINTERGRUNDFARBE + RAHMEN WECHSELN
620 PAUSE 4
630 NEXT X
640 COLOUR 9.9
650 WAIT 198.255 : REM AUF TASTE WARTEN

```

Es ist doch erstaunlich, was man alles mit so einem kleinen Befehl anstellen kann, oder?



**FORMAT** : ANGL MX,MY,W,RX,RY,ZM  
**PARAMETER** : - MX: X-KOORDINATE DES MITTEL-  
 PUNKTES DER ELLIPSE,  
 DEREN RADIUS  
 GEZEICHNET WERDEN SOLL  
 MC: 0-159 / HGR: 0-319  
 - MY: Y-KOORDINATE DES MITTEL-  
 PUNKTES DER ELLIPSE,  
 DEREN RADIUS  
 GEZEICHNET WERDEN SOLL  
 MC: 0-199 / HGR: 0-199  
 - W : WINKELPOSITION DES RADIUS  
 0-360 ALTGRAD  
 - RX: RADIUS DER ELLIPSE (KREIS)  
 IN X-RICHTUNG (SENKRECHT).  
 DEREN RADIUS GEZEICHNET  
 WERDEN SOLL  
 MC: 0-159 / HGR: 0-319  
 - RY: RADIUS DER ELLIPSE (KREIS)  
 IN Y-RICHTUNG (WAAGERECHT).  
 DEREN RADIUS GEZEICHNET  
 WERDEN SOLL  
 MC: 0-199 / HGR: 0-199  
 - ZM: ZEICHENMODUS  
**BEISPIEL** : ANGL 160,100,20,60,70,1  
**FUNKTION** : ZEICHNEN EINES RADIUS EINER ELLIPSE  
 (EINES KREISES) AN EINER DEFINIERTEN  
 WINKELPOSITION

**ERLAUTERUNGEN:**

Nun zu dem angekündigten weiteren Befehl, der sich mit Variationen zum Thema Ellipse beschäftigt. Wiederum sollten Sie die beiden vorstehenden Befehle zumindest gelesen haben, um hier "mitreden" zu können.

Der Befehl ANGL ermöglicht Ihnen, einen beliebigen Radius an einer ebenso beliebigen Stelle einer (imaginären) Ellipse zu zeichnen, d.h. nicht die Ellipse oder Teile von ihr werden gezeichnet, sondern lediglich der Radius an der angegebenen Stelle. Wie auch bei ARC muß hier zunächst einmal die Ellipse

definiert werden, auf die sich das Kommando beziehen soll. Dies geschieht natürlich wieder auf altbekannte Weise mit Mittelpunkts- und Radienangabe (mx,my,...,rx,ry). Aber das sind ja schon alte Kamellen für Sie (oder?).

Wichtig ist nun die Angabe, an welcher Stelle der Ellipse dieser Radius gezeichnet werden soll. Hier bietet sich natürlich wieder die vom ARC-Befehl her bekannte Definition per Winkelgrade an (s. ARC). Sie senden dem Computer in Ihrem Befehl also einfach die Gradzahl (die ja einer Stelle auf der Ellipse entspricht), von der aus Sie den Radius zur Ellipsenmitte gezeichnet haben wollen. Das ist schon alles, was Sie zu tun haben.

Wie kein anderer Befehl ist dieser zum Erzeugen schöner Effekte nützlich. Sie können z.B. die dazugehörige Ellipse oder ein entsprechendes n-eck (s. ARC) darüber zeichnen und so weiter. Dieser Befehl ist wirklich nur durch Probieren zu erschließen. Probieren Sie alles aus!

#### **BEISPIELE:**

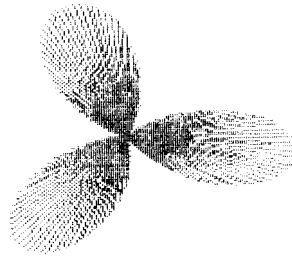
Wie mittlerweile bekannt, haben Sie hier die Möglichkeit mit Hilfe des folgenden Übungsbeispiels die vielen Parameter auf ihre Auswirkungen zu untersuchen. Seien Sie aber gewahr, daß besonders der ANGL-Befehl auf das Zeichnen vieler Figuren (Figurenkomplexe), wie in den dann folgenden Beispielen gezeigt, angewiesen ist, wenn er richtig zur Geltung kommen soll. Aus diesem Grunde ist z.B. eine FOR...NEXT - Schleife hier einfach nicht wegzudenken. Versuchen Sie sich auch auf diesem Gebiete!

```
100 REM #####
110 REM ## ##
120 REM ## ANGL-BEISPIEL-1 ##
130 REM ## ##
140 REM #####
150 REM
160 HIRES 7.8 : REM GRAPHIK EIN
170 REM TESTBEISPIEL:
180 REM BITTE AENDERN SIE DIE PARAMETER!
190 REM
```

```

200 REM  MXI MYI  WI RXI RYIZM
210 REM  ----+----+----+----+----+
220 REM      !  !  !  !  !
230 ANGL 160,100, 45, 70, 90, 1
240 REM  ----+----+----+----+----+
250 WAIT 198,255 : REM AUF TASTE WARTEN

```



```

100 REM #####
110 REM ##                               ##
120 REM ## ANGL-BEISPIEL-2 ##
130 REM ##                               ##
140 REM #####
150 REM
160 REM
170 REM      #####
180 REM      ##                               ##
190 REM      ## FIGUR 1 ##
200 REM      ##                               ##
210 REM      #####
220 REM
230 HIRES 7.8 : REM GRAPHIK EIN
240 FOR X=1 TO 140 STEP 2
250 ANGL X.X.X.X.X,1
260 NEXT X
270 REM
280 REM      #####
290 REM      ##                               ##
300 REM      ## KLEEBLATT ##
310 REM      ##                               ##
320 REM      #####

```

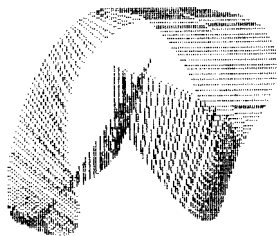
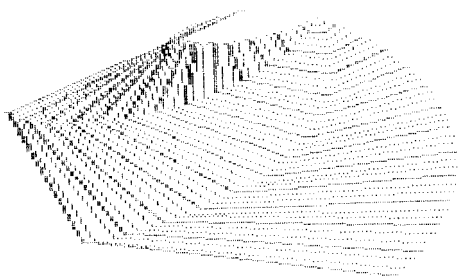
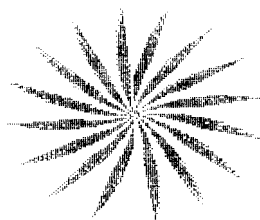
```

330 REM
340 HIRES 7.8 : REM GRAPHIK LOESCHEN
350 FOR X=1 TO 360 STEP 2
360 A = 40 * SIN(X/19 -.5) + 40
370 ANGL 160, 100, X, A, A, 1
380 NEXT X
390 PAUSE 20
400 REM
410 REM      #####
420 REM      ##                ##
430 REM      ##  FIGUR 3  ##
440 REM      ##                ##
450 REM      #####
460 REM
470 HIRES 7.8 : REM GRAPHIK LOESCHEN
480 FOR X=1 TO 400 STEP 2
490 MX = 50 * SIN(X/60) + 100
500 MY = 60 * SIN(X/40) + 70
510 W  = X/2
520 RX = 20 * SIN(X/20) + 30
530 RY = 30 * SIN(X/30) + 30
540 ANGL MX, MY, W, RX, RY, 1
550 NEXT X
560 PAUSE 20
570 REM
580 REM      #####
590 REM      ##                ##
600 REM      ##  MARGERITE  ##
610 REM      ##                ##
620 REM      #####
630 REM
640 HIRES 7.8 : REM GRAPHIK LOESCHEN
650 FOR Y=0 TO 31
660 T = ABS(T-1)
670 FOR X=1 TO 20 STEP 1
680 ANGL 160, 100, X+Y*11, X*3, X*3, T
690 NEXT X
700 NEXT Y
710 WAIT 198.255

```

Wie Sie sehen, können hier außerordentlich schöne Effekte mit

einfachsten Mitteln erzeugt werden. Kein Zeichenbefehl - so glaube ich - wirft so Vieles ab wie ANGL (außer vielleicht der folgende). Überzeugen Sie sich selbst!





### 12.3.9 PAINT

**FORMAT** : PAINT X.Y.ZM  
**PARAMETER** : - X : X-KOORDINATE DES  
TESTPUNKTES  
MC: 0-159 / HGR: 0-319  
- Y : Y-KOORDINATE DES  
TESTPUNKTES  
MC: 0-199 / HGR: 0-199  
- ZM: ZEICHENMODUS  
**BEISPIEL** : PAINT 100.120.1  
**FUNKTION** : AUSFÜLLEN EINER BELIEBIGEN  
UMSCHLOSSENEN FLÄCHE

#### ERLÄUTERUNGEN:

Der letzte in diesem Abschnitt zu besprechende Graphikbefehl ist der PAINT-Befehl. Setzen Sie sich erst einmal hin, bevor Sie weiterlesen und halten Sie sich fest, das könnte Sie glatt vom Hocker hauen. Ich darf Sie mit einem Herrn bekannt machen, der Ihnen viel, sehr viel Arbeit abnimmt. Sie geben Ihm auf Ihrem Bildschirm eine beliebige Fläche vor, irgendetwas, was Sie wollen. Dann tritt er in Aktion und Ihre Fläche beginnt zu strahlen.

Nun, bleiben wir auf dem Boden! Der Entwickler des Simon's Basic hat sich hier etwas sehr Hübsches ausgedacht:

Angenommen, Sie wollen nicht nur mit dem inzwischen bekannten CIRCLE-Befehl einen einfachen Kreis (oder Ellipse) zeichnen, sondern diesen auch ausgefüllt wissen. Oder Sie haben sich eine schöne abgeschlossene Figur ausgedacht und haben das Gleiche damit vor. In diesem Falle arbeiten Sie folgendermaßen: Sie zeichnen zunächst einmal die Umrandung der Ihnen vorschwebenden Figur auf den Bildschirm. Dabei achten Sie darauf, daß es keine Lücke in dieser Umrandung gibt. Nun bestimmen Sie einen einzigen Punkt, der innerhalb dieser Figur liegt und übergeben dessen Koordinaten in altbekannter Weise dem PAINT-Befehl als sogenannten Testpunkt. Das Übrige wird von selbst erledigt. PAINT findet Ihre Umrandung und füllt die Figur mit Punkten vollständig auf. Der umgekehrte Fall gilt beim Löschen: hier erkennt PAINT alle gelöschten Punkte als Rand und löscht gleichfalls alle innerhalb liegenden Punkte (samt ehemaliger Umrandung).

Natürlich können Sie auch einen Punkt außerhalb der Figur angeben. In diesem Falle wird der gesamte Bildschirm außer allen abgeschlossenen Figuren in ihm ausgefüllt (gelöscht). Lediglich das Invertieren (zm=2) zeitigt aus einem unerfindlichen Grunde merkwürdige Effekte und sollte vermieden werden (außer, wenn Ihnen das gefällt - probieren Sie es aus).

Es kann schon einmal vorkommen, daß bestimmte Teile von Figuren (besonders in spitzen Ecken) nicht ausgefüllt werden. Das jedoch liegt nicht am PAINT-Befehl, sondern vielmehr an der Art und Weise, wie Linien und Ellipsen etc. auf den Bildschirm gebracht werden.

Anders als im BLOCK-Befehl ist bei PAINT der LOW COL - Modus voll wirksam. Sie können also - falls erwünscht - als BLOCK-Ersatz ein Rechteck (mit REC gezeichnet) mit PAINT ausfüllen und haben so die Möglichkeit, die Farbe zu ändern (s. auch REC und BLOCK). Zur Demonstration wieder einige

#### BEISPIELE:

```

100 REM #####
110 REM ##                ##
120 REM ## PAINT-BEISPIEL ##
130 REM ##                ##
140 REM #####
150 REM
160 REM
170 REM      #####
180 REM      ##          ##
190 REM      ## OVAL  ##
200 REM      ##          ##
210 REM      #####
220 REM
230 HIRES 8.9 : REM GRAPHIK EIN
240 CIRCLE 160.100.50.80.1 : REM ELLIPSE ALS FIGURBEGRENZUNG
250 PAUSE 1
260 PAINT 160.100.1 : REM AUSFUELLEN
270 PAUSE 5
280 REC 10.10.60.70.1 : REM RECHTECK ALS FIGURBEGRENZUNG
290 PAUSE 1
300 PAINT 11.11.1

```

```

310 PAUSE 15
320 REM
330 REM #####
340 REM ##          ##
350 REM ##  FIGUR 2  ##
360 REM ##          ##
370 REM #####
380 REM
390 HIRES 8.9 : REM GRAPHIK LOESCHEN
400 REM
410 REM *****
420 REM FELD UMREISSEN:
430 REM *****
440 REM
450 LINE 100,100, 50, 70,1
460 LINE 50, 70,150,120,1
470 LINE 150,120,120, 20,1
480 LINE 120, 20,300,199,1
490 LINE 300,199,160,100,1
500 LINE 160,100,160,170,1
510 LINE 160,170, 60,130,1
520 LINE 60,130,100,100,1
530 PAUSE 1
540 PAINT 101,101,1 : REM FELD AUSMAHLEN
550 PAUSE 20
560 PAINT 101,101,0 : REM FELD LOESCHEN
570 WAIT 198,255 : REM AUF TASTE WARTEN

```

-----

```

100 REM #####
110 REM ##          ##
120 REM ##  PAINT-BEISPIEL  ##
130 REM ##          ##
140 REM #####
150 REM
160 HIRES 3.2 : REM GRAPHIK EIN
170 CIRCLE 160,100,30,40,1 : REM INNEN
180 CIRCLE 160,100,50,60,1 : REM MITTE
190 CIRCLE 160,100,70,80,1 : REM AUSSEN
200 PAINT 160+29,100+39,1 : REM MITTLEREN STREIFEN AUSFUELLEN

```

210 PAINT 0. 0.1 : REM AUESSERES FELD AUSFUELLEN  
220 PAUSE 15  
230 REM  
240 REM #####  
250 REM ## ##  
260 REM ## BLUENCHEN ##  
270 REM ## ##  
280 REM #####  
290 REM  
300 HIRES 3.2  
310 CIRCLE 160.100.10.10.1  
320 CIRCLE 160.100.30.40.1  
330 CIRCLE 160.100.50.20.1  
340 PAINT 160. 100.1  
350 PAINT 160.100+38.1  
360 PAINT 160.100-38.1  
370 PAINT 160+48. 100.1  
380 PAINT 160-48. 100.1  
390 WAIT 198.255

### 12.3.10 TESTAUFGABEN

Nun endlich sind wir am Ende dieses Abschnittes und wollen unser Wissen anhand einiger Testaufgaben wieder einmal überprüfen.

1.) Welche Syntax ist richtig?

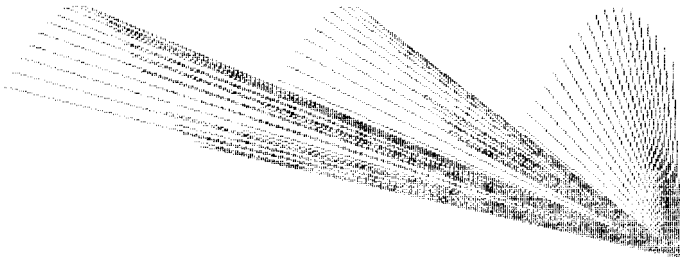
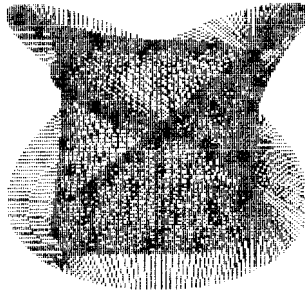
- a) PLOT 1,0,0 :  
zeichnet einen Punkt bei 0,0
- b) TEST (100,100) :  
testet, ob ein Punkt bei den Koordinaten 100,100 gesetzt ist
- c) IF TEST (100,100) = 0 THEN 90 :  
Falls kein Punkt bei 100,100 gesetzt ist, wird nach Zeile 90 verzweigt.
- d) REC 10,20,100,150 :  
Zeichnet ein Rechteck mit den zwei Eckpunkten 10,20 und 100,150
- e) BLOCK 10,20,100,150 :  
Zeichnet ein Feld mit den zwei Eckpunkten 10,20 und 100,150
- f) ARC 160,100,0,360,12,50,70,1  
Zeichnet eine vollständige Ellipse

2.) Sie wollen die mit CIRCLE 160,100,40,30,1 gezeichnete Ellipse ausfüllen. Welche Befehle wenden Sie an?

- a) BLOCK 160,100,40,30,1
- b) PAINT 160,100,1
- c) PAINT 0,0,1
- d) BLOCK 160,100,1

3.) Sie wollen einen Rahmen mit einer Dicke von 10 Punkten zeichnen. Welche Befehlskombination wählen Sie?

- a) FOR X=0 TO 9 : REC 100+X,100+X,50-2\*X,60-2\*X,1 : NEXT X
- b) BLOCK 100,100,150,160,1 : BLOCK 110,110,140,160,0
- c) REC 100,100,50,60,1 : REC 110,110,30,40,1 : PAINT 105,105,1
- d) REC 100,100,50,60,1



## 12.4 ZEICHNEN EIGENER FIGUREN

### 12.4.1 DRAW

```
FORMAT      : DRAW STR.X.Y.ZM
PARAMETER   : - STR: STRINGVARIABLE ODER
              "ZEICHENKETTE"
              BESTEHEND AUS DER
              FIGURDEFINITION (S.U.)
              - X : X-KOORDINATE DES START-
                  PUNKTES DER DRAW-FIGUR
                  (HGR: 0-319/MC: 0-159)
              - Y : Y-KOORDINATE DES START-
                  PUNKTES DER DRAW-FIGUR
                  (HGR: 0-199/MC: 0-199)
              - ZM : ZEICHENMODUS
BEISPIEL    : DRAW "5678".160.100.1 ODER
              DRAW AS.160.100.1
FUNKTION    : ZEICHNEN EINES BELIEBIGEN
              GEBILDES
```

#### ERLÄUTERUNGEN:

Bisher beschäftigten wir uns lediglich mit fest-programmierten, durch eine bestimmte Rechenvorschrift (Algorithmus) vorbestimmten Figuren, die wir lediglich vergrößern oder verzerren (z.B. CIRCLE) konnten. Die Grundform dieser Gebilde jedoch war uns stets vorgegeben.

Nun könnte man natürlich hingehen und mittels des PLOT- und LINE-Befehls aus vordefinierten Punkten eine eigene, komplexe Figur gestalten. Dieses Vorgehen ist natürlich sehr speicherplatzaufwendig und zeitintensiv. Wollten Sie gleichfalls noch eine beliebige Verschiebung Ihres Gebildes oder gar eine Rotation ermöglichen, so wäre die Programmiersprache Basic total überfordert.

Dieser Vorgang wird nun durch den Simon's Basic - Befehl DRAW überflüssig. Auf einfache und vor allem schnellste Weise ist nun die Erstellung einer eigenen beliebig gestalteten Figur möglich. Diese Figur kann weiterhin beliebig auf dem Graphikbildschirm positioniert, ja sogar gedreht und vergrößert werden.

Die zu erstellende Figur wird durch den Inhalt des mit str bezeichneten Stringparameters festgelegt. D.h. Sie erstellen auf sofort erläuterte Art und Weise einen String und hängen ihn (in Form einer Stringvariablen oder einfach als "Zeichenkette") an das DRAW-Befehlswort hintenan.

Wie ist nun eine solche Definition zu bewerkstelligen?

Nun, stellen Sie sich einmal vor, der Computer zeichnete mit einem Stift diese Figur auf den Bildschirm. Sie geben nun an, in welche der vier möglichen Richtungen (auf, ab, rechts, links) Ihr Gerät den Stift (oder auch Graphikcursor) bewegen soll. Dies geschieht jeweils um eine Längeneinheit, die Sie mit ROT (s.u.) bestimmen können. Gleichzeitig geben Sie Ihm an, ob der Stift während der Bewegung auf dem Zeichenpapier (Bildschirm) liegen oder über Ihm schweben soll. Im ersten Falle käme eine Linie zustande, im zweiten würde der Stift nur bewegt, ohne zu zeichnen. Diese zwei mal vier Funktionen werden dem Computer nun durch die folgenden Ziffern mitgeteilt:

**ZIFFER.....BEWEGUNGSRICHTUNG.....ZEICHNEN?**

0.....	RECHTS.....	NEIN
1.....	AUF.....	NEIN
2.....	AB.....	NEIN
3.....	LINKS.....	NEIN
4.....	AB.....	NEIN
5.....	RECHTS.....	JA
6.....	AUF.....	JA
7.....	AB.....	JA
8.....	LINKS.....	JA
9.....	A.B.B.R.U.C.H.....	

**ALSO:**

$$\begin{array}{c}
 1/6 \\
 3/8 \quad \perp \quad 0/5 \\
 2/7 \\
 (4)
 \end{array}$$

**DER ZIFFER 9 IST KEINE ZEICHENFUNKTION ZUGEORDNET, UND SIE FÜHRT ZUM ABRUCH DER ZEICHNUNG. AUCH WENN NOCH WEITERE ZIFFERN FOLGEN SOLLTEN. SIE MUß JEDOCH NICHT, WIE DAS**



**HANDBUCH ERSCHEINEN LÄßT, ALS ENDE EINER DEFINITION HINTER JEDEM STRING STEHEN.**

**IHRE FIGUR WIRD ALSO DURCH EIN ANEINANDERHÄNGEN DIESER RICHTUNGSANWEISUNGEN ZUSAMMENGESETZT. DIE LÄNGE IST LEDIGLICH DURCH DIE MAXIMALE STRINGLÄNGE (256 ZEICHEN) BEGRENZT. DA SIE JEDOCH NUR MAXIMAL 80 ZEICHEN IN EINE BILDSCHIRMZEILE (BASICZEILE) HINEINBEKOMMEN, MÜSSEN SIE LÄNGERE DEFINITIONEN DURCH FOLGENDEN TRICK AUS MEHREREN ZEILEN ZUSAMMENSETZEN:**

**BEISPIEL:**

**10 A\$ = "1234567890123456789012345678901**

**234567890123456789012345678901234567890"**

**20 A\$ = A\$ + "12345241367844343278"**

Der Speicher A\$ kann nun als Definitionsstring in den DRAW-Befehl eingesetzt werden.

Neben str sind noch drei weitere Parameter anzugeben, um eine vollständige Syntax zu erhalten. Den letzten (zm) brauchen wir hier nicht weiter zu behandeln, da Ihnen dessen Funktion bereits aus dem Abschnitt 12.2 bekannt sein sollte (Zeichenmodus). x und y jedoch geben uns die Möglichkeit, die Position auf dem Graphikbildschirm zu bestimmen, an der zum ersten Male der Zeichenstift angesetzt werden soll, d.h. die Startposition des Zeichenstiftes (Graphikcursors). Von hier aus beginnt der Computer nun mit der Definitionsabarbeitung. Eine schöne Anwendung des DRAW-Befehls ist die Ergänzung der Sprites durch Figuren dieser Art (sogenannte Shapes), die durch schnelles Löschen und versetztes Neuzeichnen einer Figur über den Bildschirm bewegt werden können (s. Beispiel 2). Mit dieser Technik sollten Sie sich unbedingt vertraut machen. Es lohnt sich!

Die Stärke dieses Befehls liegt unter Anderem in der Möglichkeit der Rotation und Vergrößerung mittels des ROT-Befehls (s. 12.4.2).

**BEISPIELE:**

```

100 REM #####
110 REM ##          ##
120 REM ## DRAW-BEISPIEL-1 ##
130 REM ##          ##
140 REM #####
150 REM
160 HIRES 6.7 : REM GRAPHIK EIN
170 ROT 0.40 : REM KEINE ROTATION / GROESSE 40
180 DRAW "65789".160.100.1 : REM QUADRAT ZEICHNEN
190 REM HOCH + PLOT / RECHTS + PLOT
200 REM RUNTER + PLOT / LINKS + PLOT
210 REM ENDE
220 WAIT 198.255 : REM AUF TASTE WARTEN

```

Das obige Beispielprogramm soll Ihnen als Experimentier- oder Arbeitsprogramm dienen. Sie können hier Änderungen vornehmen und die vorgegebene Figur (ein Quadrat) so verändern, wie es Ihnen gefällt. Die Zeilen 190 bis 200 beschreiben die Funktion der einzelnen Ziffern in dem Definitionsstring. Dabei könnten Sie die 9 selbstverständlich - wie oben beschrieben - weglassen.

-----

```

100 REM #####
110 REM ##          ##
120 REM ## DRAW-BEISPIEL-2 ##
130 REM ##          ##
140 REM #####
150 REM
160 REM
170 REM #####
180 REM ##          ##
190 REM ## BUCHSTABENDEFINITIONEN: ##
200 REM ##          ##
210 REM #####
220 REM
230 DATA "666666555575777787888" : REM D
240 DATA "666666555577788880000777" : REM A

```

```

250 DATA "000666666888000555" : REM T
260 DATA "666666555577788880000777" : REM A
270 DATA "" : REM SPACE
280 DATA "6666665555777888800007778888" : REM B
290 DATA "55555333336665555333366655555" : REM E
300 DATA "555533336666665555" : REM C
310 DATA "666575757333111565656333777" : REM K
320 DATA "55555333336665555333366655555" : REM E
330 DATA "666575757333111666555777888" : REM R
340 REM
350 REM #####
360 REM ## ##
370 REM ## ZEICHENTEIL ##
380 REM ## ##
390 REM #####
400 REM
410 HIRES2.3
420 ROT 0.4 : REM KEINE ROTATION / GROESSE 4 (S.U.)
430 FOR X=1 TO 11
440 READ A$: REM NACHEINANDER DATA LESEN
450 DRAW A$, X*30 - 28, 50 + 10*X, 1 : REM BUCHSTABEN
ZEICHNEN
460 NEXT X
470 PAUSE 4
480 REM
490 REM #####
500 REM ## BUCHSTABEN VERBREITERN: ##
510 REM #####
520 REM
530 RESTORE : REM DATAZEIGER RUECKSETZEN
540 FOR X=1 TO 11 : REM BUCHSTABENSCHLEIFE
550 READ A$: REM NACHEINANDER DATA LESEN
560 FOR Z=0 TO 7 STEP 1 : REM BREITE-SCHLEIFE
570 DRAW A$, X*30 - 28 + Z, 50 + 10*X + Z, 1 : REM
BUCHSTABEN ZEICHNEN
580 NEXT Z
590 NEXT X
600 WAIT 198.255 : REM AUF TASTE WARTEN

```

Das obige Programm stellt einen Schriftzug mittels des DRAW-Befehls dar, dessen einzelne Buchstaben in den Zeilen

230 bis 330 als Stringdefinition in sogenannten DATA-Zeilen abgelegt sind. Aus diesen Zeilen können die einzelnen Elemente (hier: jeder durch die Anführungszeichen ("") begrenzte String) später nacheinander durch den Befehl READ (Zeilen 440 und 550) in beliebige Stringspeicher eingelesen werden. Jede DATA-Zeile liefert hier die Definition für ein bestimmtes Zeichen (jeweils in den REM-Ausdrücken beschrieben).

Alsdann werden die einzelnen Buchstaben auf den Bildschirm gebracht und im nächsten Schritt durch versetztes Zeichnen verdickt (ein Tip: ändern Sie doch einmal den STEP - Parameter in Zeile 560).

**DATA  
BECKER**

## 12.4.2 ROT

**FORMAT** : ROT R,G  
**PARAMETER** : - R: ROTATIONSWINKEL DER FIGUR  
UM DEN AUSGANGSPUNKT (0-7)  
- G: VERGRÖßERUNGSFAKTOR (0-255)  
**BEISPIEL** : ROT 0,20  
**FUNKTION** : ROTIEREN UND VERGRÖßERN DER DRAW-FIGUR

### ERLÄUTERUNGEN:

DRAW alleine schon birgt eine ganze Menge von Möglichkeiten des Einsatzes zum leichten Erstellen von Graphiken Ihrer Wahl. ROT setzt dem Ganzen die Krone auf.

Mit ROT können Sie die zu erstellende Figur (Shape) um den Ausgangspunkt als Drehangel verdrehen. Sie legen Ihre Figur also schräg oder stellen alles auf den Kopf, ganz wie es Ihnen in den Sinn kommt. Ihre Shapes können Kapriolen drehen oder gekonnt abschmieren. Kurz, es liegt ganz in Ihrer Hand, was Sie mit Ihrem Computer anstellen.

Es stehen Ihnen 8 festgelegte Drehwinkel zur Verfügung, die wie folgt festgelegt sind und jeweils eine viertel Drehung darstellen.

### R.....ROTATIONSWINKEL

-----  
0.....0 GRAD  
1.....45 GRAD  
2.....90 GRAD  
3.....135 GRAD  
4.....180 GRAD  
5.....225 GRAD  
6.....270 GRAD  
7.....315 GRAD

Wählen Sie also r gleich 0, so wird Ihr in DRAW definiertes Gebilde ohne Drehung so, wie Sie es eingegeben haben gezeichnet. Wird r jedoch z.B. 1, so wird es um 45 Grad (Neugrad) im Uhrzeigersinn gedreht (Hier wird also nicht mit der mathematischen Winkelmessung gearbeitet). Geben Sie für r eine Zahl größer als 7 an, so erscheint die Meldung BAD MODE, wie wir sie schon von verschiedenen anderen Befehlen her kennen.

Bei der Verwendung von ROT zur Rotation ist noch einiges anzumerken. Eine Drehung um den Startpunkt durch den Befehl ROT wahrt nicht unbedingt die Proportionen der gedrehten Figur im Vergleich zu anderen oder allgemein dem Bildschirmfenster. Bei ungeraden Drehpositionen ( $r=1,3,5,7$ ) erscheint die gesamte Figur vergrößert. Dies liegt daran, daß bei der Drehung ein Punkt nicht etwa die Bahn eines Kreises zurücklegt, sondern die eines Quadrates. Dieser Effekt wird am besten in dem folgenden Beispiel deutlich, in dem langsam lediglich ein Strich gedreht wird:

```
10 HIRES 2.3 : REM GRAPHIK EIN
20 FOR R=0 TO 7
30 PAUSE 2
40 ROT R.20 : REM ROTIEREN LASSEN
50 DRAW "6",160,100.1 : REM STRICH ZEICHNEN
60 NEXT R
70 WAIT 198,255 : REM AUF TASTE WARTEN
```

Neben  $r$  können Sie - wie Sie sehen - noch einen zweiten Parameter angeben:  $g$

Mit  $g$  bestimmen Sie den Faktor, mit dem Sie Ihre Figur vergrößern wollen. Normalerweise entspricht jedem Schritt nach rechts, links, oben oder unten (s. DRAW) bei der Definition des Shapes ein Schritt um einen Punkt in die angegebene Richtung. Die Schrittweite beträgt also einen Punkt. Diese Schrittweite und damit die Größe der gesamten Figur können Sie nun mit  $g$  verändern. Ist somit  $g$  gleich 1, so wird die zu zeichnende Figur in Originalgröße (also unvergrößert) gezeichnet. Dies wird oftmals derart winzig, daß eine Vergrößerung schon allein aus optischen Gründen zwingend ist.  $g$  kann Werte von 0 bis 255 annehmen, wobei 0 einer Vergrößerung um 256 Einheiten entspricht. Nur wenig über den Bildschirmrand ragende Teile der Zeichnung werden abgeschnitten. Ansonsten entsteht an einer anderen Stelle des Bildschirms ein Übertrag (wie auch bei allen anderen Graphikbefehlen).

Selbstverständlich ist hierbei auf die veränderten Größenverhältnisse und Verzerrungen im Multicolormodus zu achten, die in früheren Kapiteln ausgiebig behandelt wurden.

Nun einige Beispiele, die Ihnen die Anwendung der Befehle DRAW und ROT veranschaulichen sollten. Dabei zeigt Ihnen der erste Teil den Umgang mit beweglichen Shapes:

#### BEISPIELE:

```
100 REM #####
110 REM ##          ##
120 REM ##  ROT-BEISPIEL  ##
130 REM ##          ##
140 REM #####
150 REM
160 REM
170 REM #####
180 REM ##          ##
190 REM ##  UHRZEIGER  ##
200 REM ##          ##
210 REM #####
220 REM
230 HIRES 7.6 : REM GRAPHIK EIN
240 FOR T=1 TO 20 : REM RUNDENSCHLEIFE
250 FOR X=0 TO 7 : REM ROTATIONSSCHLEIFE
260 ROT L.40 : REM LETZTEN ROTATIONSGRAD EINSTELLEN
270 DRAW "6".160.100.0 : REM LETZTEN STRICH LOESCHEN
280 ROT X.40 : REM NEUER ROTATIONSGRAD
290 DRAW "6".160.100.1 : REM STRICH ZEICHNEN
300 L = X : REM LETZTEN ROTATIONSGRAD MERKEN
310 NEXT X
320 NEXT T
330 PAUSE 3
340 REM
350 REM #####
360 REM ##          ##
370 REM ##  BILD 2  ##
380 REM ##          ##
390 REM #####
400 REM
410 HIRES 7.6 : REM GRAPHIK LOESCHEN
420 FOR X=1 TO 69
430 ROT X/10. X : REM ROTATION UND GROESSE
440 DRAW "365577886".160.100.1
```

```

450 NEXT X
460 PAUSE 10
470 REM
480 REM #####
490 REM ##          ##
500 REM ## STERN ##
510 REM ##          ##
520 REM #####
530 REM
540 HIRES 7.6 : REM GRAPHIK LOESCHEN
550 FOR X=1 TO 69
560 ROT X/10. X : REM ROTATION UND GROESSE
570 DRAW "5885766".160.100.1
580 NEXT X
590 WAIT 198.255 : REM AUF TASTE WARTEN

```

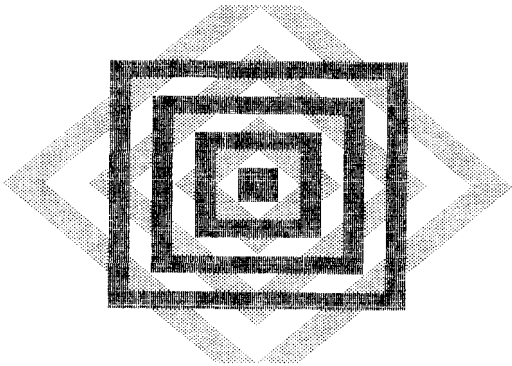
Im ersten Teil wird Ihnen, wie gesagt, der Umgang mit Bewegungen dargelegt. Wichtig hierbei ist, daß zunächst das Objekt an der nächsten Position gezeichnet und dann das bereits gezeichnete, alte Objekt an der letzten Position gelöscht wird, um eine möglichst fließende Bewegung vorzutauschen. Manchmal muß man auch den umgekehrten Weg gehen, um, besonders bei großen Figuren, störende Linien zu eliminieren. So können Sie nicht nur rotieren lassen, sondern gleichfalls in verschiedene Richtungen bewegen, indem Sie stets an neuen Koordinaten zeichnen. Dies mag das nächste Beispiel erläutern:

```

100 REM #####
110 REM ##          ##
120 REM ## ROT-BEISPIEL-2 ##
130 REM ##          ##
140 REM #####
150 REM
160 HIRES 2.3 : REM GRAPHIK EIN
170 ROT 0.20 : REM KEINE ROTATION
180 FOR X=1 TO 319 STEP 4
190 DRAW "6578".A.100.0 : REM AUF ALTER POSITION LOESCHEN
200 DRAW "6578".X.100.1 : REM AUF NEUER POSITION ZEICHNEN
210 A = X : REM ALTE POSITION MERKEN
220 NEXT X

```





### 12.4.3 TESTAUFGABEN

Zur Wissenskontrolle wieder unser schöner, kleiner Test:

1.) Welche Befehle verwenden Sie, wenn Sie ein Quadrat zeichnen wollen?

- a) REC 160,100,70,70,1
- b) DRAW "0132",160,100,1
- c) DRAW "5687",160,100,1
- d) ROT 4,5

2.) Sie wollen Ihre Figur um 90 Grad gegen den Uhrzeigersinn drehen und fünffach vergrößern. Welche Befehle verwenden Sie?

- a) ROT 90,5
- b) ROT 2,5
- c) ROT 6,5
- d) ROT 5,2

3.) Welche Ziffernzuordnungen für die Shapedefinition sind richtig?

- a) 0 : rechts ohne Zeichnen
- b) 8 : rechts mit Zeichnen
- c) 4 : links ohne Zeichnen
- d) 6 : oben mit Zeichnen
- e) 9 : unten mit Zeichnen

## 12.5 TEXT IN DER GRAPHIK

### 12.5.1 CHAR

```
FORMAT      : CHAR X.Y.C.ZM.G
PARAMETER   : - X : X-KOORDINATE DER LINKEN
                OBEREN ECKE DES ZEICHENS
                HGR: 0-319 / MC: 0-159
              - Y : Y-KOORDINATE DER LINKEN
                OBEREN ECKE DES ZEICHENS
                HGR: 0-199 / MC: 0-199
              - C : BILDSCHIRMCODE DES
                ZEICHENS (0-255)
              - ZM: ZEICHENMODUS
              - G : VERTIKALE GROBE DES
                ZEICHENS (0-255)
BEISPIEL    : CHAR 160.100.3.1.2
FUNKTION     : ZEICHNEN EINES EINZELNEN ZEICHENS
```

#### ERLÄUTERUNGEN:

Ihre Graphiken stehen von nun an nicht mehr anonym im Raum. Sie können sie jetzt beschriften oder Erklärungen abgeben. Reden an Ihr Vaterland oder einfache, knappe Kommentare gehören punktuell zu Ihrem täglichen Repertoire. Lassen Sie sich von dieser ganz heißen Sache überraschen, das geht Ihnen bis unter die Haut!

Nun, dieser Befehl ermöglicht es Ihnen - um wieder zum Thema zurückzukehren - einen einzelnen Buchstaben oder ein beliebiges Graphikzeichen normal oder invers, je nach Wunsch, auf den Bildschirm zu bringen. Dieses Zeichen definieren Sie mit dem Parameter c (für Bildschirmcode). Wie Sie aus dem 9. Kapitel vielleicht noch wissen (s. FCHR oder FILL), werden die einzelnen Buchstaben oder Graphikzeichen Ihres Commodore 64 in dem Bildschirmspeicher unter einem anderen Code als dem ASCII-Code abgespeichert. Dies war notwendig, um auch die inversen Zeichen codieren zu können. Wie schon erwähnt finden Sie eine vollständige Bildschirmcode-tabelle in Ihrem deutschen CBM 64 - Handbuch auf den Seiten 133/134. Um hier die Codes der inversen Zeichen zu erhalten, müssen Sie jeweils 128 zu dem Code der normalen Zeichen hinzuaddieren.

Mit Hilfe dieser Tabelle geben Sie nun den Parameter `c` an, der das von Ihnen gewählte Zeichen angibt (z.B. `c=5` für ein normales großes E; oder `c=128+10` für ein inverses J). Wie Sie sehen, gibt es keine Möglichkeit, den anzusprechenden Zeichensatz auszuwählen. Es werden somit nur die Zeichen des Großschrift- / Graphikzeichen - Modus durch diesen Befehl auf den Bildschirm gebracht. Um die Zeichen des alternativen Zeichensatzes auszuwählen, müssen Sie sich also zwangsläufig des demnächst zu besprechenden Befehls `TEXT` bedienen (s.u.). Nun können Sie weiterhin angeben, an welche Stelle des Bildschirms Sie das einzelne Zeichen setzen wollen. Dies geschieht mit Hilfe der Parameter `x` und `y`, die die Koordinaten der oberen linken Ecke des Zeichens auf dem Bildschirm bestimmen. Wollen Sie also die obere linke Ecke Ihres Zeichens im Nullpunkt positionieren, so wählen Sie `x=0` und `y=0`.

Neben dem bekannten Zeichenmodus können Sie weiterhin noch einen Parameter `g` am Schluß des Befehls angeben. Dieser Wert hat die Funktion, wie bei `ROT` (s.o.) den Vergrößerungsfaktor der einzelnen Zeichen festzulegen. Im Unterschied zu `ROT` jedoch wird hier jedes Zeichen nur in vertikaler Richtung vergrößert, die Zeichen werden also lediglich länger (und nicht breiter wie bei `ROT`). `g` kann hier Werte zwischen 0 und 255 annehmen, wobei `g=0` `g=1` entspricht. Natürlich sind hier nur Werten von 0 bis höchstens 30 sinnvoll, da hier bereits die Zeichen vollkommen unleserlich werden und eventuell schon über den Bildschirmrand hinausragen.

Wie Sie sehen steht Ihnen hier ein schönes Instrument zur Verfügung, um auf leichte Weise nicht nur Ihre Graphiken zu beschriften, sondern auch einfach die zur Verfügung stehenden Graphikzeichen als Graphikelemente wie bei der Erstellung von Bildern im Textmodus zu verwenden. Doch seien Sie gefaßt auf Weiteres!

### **BEISPIELE:**

Wie schon bei früheren Befehlen, die sehr viele Parameter besaßen, wird Ihnen auch hier ein Arbeitsbeispiel angegeben, das Sie möglichst vollständig (auch mit `REM`-Zeilen) übertragen sollten, um die einzelnen Parameter möglichst schnell und richtig zu verstehen und zu behalten.

```

100 REM #####
110 REM ##          ##
120 REM ## CHAR-BEISPIEL-1 ##
130 REM ##          ##
140 REM #####
150 REM
160 HIRES 1.0 : REM GRAPHIK EIN
170 REM TESTBEISPIEL:
180 REM BITTE AENDERN SIE DIE PARAMETER!
190 REM
200 REM   XI  YI  CIZMI G
210 REM ----+----+----+----+----
220 REM   !   !   !   !
230 CHAR 160,100, 2, 1, 3
240 REM ----+----+----+----+----
250 WAIT 198,255 : REM AUF TASTE WARTEN

```

-----

```

100 REM #####
110 REM ##          ##
120 REM ## CHAR-BEISPIEL-2 ##
130 REM ##          ##
140 REM #####
150 REM
160 REM
170 REM #####
180 REM ##          ##
190 REM ## ZEICHENVORRAT ##
200 REM ##          ##
210 REM #####
220 REM
230 HIRES 5.6
240 FOR X=0 TO 255
250 T = X/40
260 CHAR FRAC(T) * 40 * 8, X/2, X, 1, 2
270 NEXT X
280 PAUSE 20
290 REM
300 REM   #####
310 REM   ##          ##

```

```
320 REM      ## GROESSE ##
330 REM      ##           ##
340 REM      #####
350 REM
360 HIRES 5.6
370 FOR X=1 TO 26
380 CHAR X*8, X, X. 1, X
390 NEXT X
400 WAIT 198.255 : REM AUF TASTE WARTEN
```

Der erste Teil des zweiten Beispiels demonstriert Ihnen den gesamten, Ihnen zur Verfügung stehenden Zeichensatz. Der zweite Teil erklärt Ihnen die Größenbehandlung. Viel Spaß!

## 12.5.2 TEXT

**FORMAT** : TEXT X.Y.STR.ZM.G.A

**PARAMETER** :

- X : X-KOORDINATE DER LINKEN OBEREN ECKE DES ERSTEN ZEICHENS DES STRINGS  
HGR: 0-319 / MC: 0-159
- Y : Y-KOORDINATE DER LINKEN OBEREN ECKE DES ERSTEN ZEICHENS DES STRINGS  
HGR: 0-199 / MC: 0-199
- STR: STRINGSPEICHER ODER "ZEICHENKETTE" MIT DEN AUSZUGEBENEN ZEICHEN
- ZM : ZEICHENMODUS
- G : VERTIKALE GRÖÖE DER ZEICHEN (0-255)
- A : ABSTAND DER LINKEN PUNKTE ZWEIER ZEICHEN

**BEISPIEL** : TEXT 50,100,"TEST".1,3,15

**FUNKTION** : SCHREIBEN EINES TEXTES

### ERLÄUTERUNGEN:

Wie wir gesehen haben, können Sie mit dem soeben beschriebenen CHAR lediglich ein Zeichen auf einmal in die Graphik schreiben. Dies bietet sich an bei der groß angelegten Verwendung eines Zeichens zum Erstellen bestimmter Muster o.ä. Wollen Sie jedoch einen längeren Text (oder Kleinbuchstaben) in Ihre Graphik zaubern, so erweist sich dieser Befehl als zu umständlich und aufwendig. Hier bietet sich der TEXT-Befehl an. Er ermöglicht es Ihnen, ganze Zeichenketten in die Graphik zu befördern.

Dies geschieht auf etwas unterschiedliche Art und Weise, wie bei CHAR. Sie geben hier mit x und y die Koordinaten der oberen linken Ecke des ersten Buchstabens der Zeichenkette (String) an, um die Position des Schriftzuges zu definieren. Als Nächstes wird der auszugebende String verlangt; entweder in der Form eines Stringspeichers, in den Sie vorher den Text (oder die Zeichen) eingegeben haben, oder direkt als "Zeichenkette". Sie können zuvor oder auch mitten im String angeben, aus welchem Zeichensatz die nachfolgenden Zeichen

dieses Strings stammen sollen. Mit <ctrl> A schalten Sie auf den ersten, also den Großschrift / Graphikzeichen - Modus um, d.h. alle Zeichen nach dieser Eingabe (die im String als ein inverses A erscheint) stammen aus dem ersten Zeichensatz. Geben Sie nun ein <ctrl> B ein (erscheint als inverses B), so unterliegt die Ausgabe dem zweiten oder Klein- / Großschrift - Modus. Wir fassen also zusammen:

<b>ZEICHEN.....FUNKTION.....</b>	<b>ERSCHEINT ALS</b>
<b>&lt;CTRL&gt; A....GROB/GRAPHIK.....</b>	<b>INVERSES A</b>
<b>&lt;CTRL&gt; B....KLEIN/GROB.....</b>	<b>INVERSES B</b>

Sie können somit auch innerhalb eines Strings zwischen den einzelnen Zeichensätzen umschalten, ohne daß davon die bereits gezeichneten Zeichen betroffen wären. Geben Sie weder <ctrl> A noch <ctrl> B an, so werden die Buchstaben automatisch wie unter <ctrl> A gesetzt.

Probieren Sie doch einmal dieses Beispiel, das Ihnen diese Beziehungen darlegt:

**10 HIRES 6,7**

**20 TEXT 50,100,"(<CTRL> A)TEXT - (<CTRL> B)TEXT",1,1,8**

**30 WAIT 198,255**

Unter (<ctrl> A) verstehen wir hier natürlich die Tastendruckkombination aus der Control - Taste und dem Zeichen A.

Nun können Sie aber neben den bereits besprochenen Parametern noch einige weitere Werte an den Befehl hintenanfügen (zm als Zeichenmodus sollte Ihnen nun wirklich geläufig sein).

Mit g bezeichnen wir wieder (wie bei CHAR) die vertikale Größe. Hierzu gilt alles unter CHAR Gesagte und sollte dort nachgelesen werden.

Ein weiterer Parameter (a) gibt Ihnen nun die Wahl, mit welchem Abstand Sie die einzelnen Zeichen des Strings auf den Bildschirm bringen wollen. Unter Abstand verstehen wir stets die Zahl der Punkte zwischen zwei Stellen. Dabei ist jeweils der Abstand der linken Kante eines Zeichens von der linken Kante des folgenden Zeichens gemeint. Wählen Sie also den Abstand zu klein (d.h. kleiner als 7), so werden sich die einzelnen Zeichen überschneiden, da jedes Zeichen eine Breite von 7 Punkten besitzt. Mit Hilfe dieses Parameters können Sie



Ihre Schriftzüge beliebig weit auseinanderziehen und schöne Effekte erzeugen.

Überzeugen Sie sich selbst. TEXT bietet Ihnen mannigfaltige Möglichkeiten der Variation. Probieren Sie es aus.

Einen kleinen Wermutstropfen kann ich Ihnen jedoch nicht ersparen. In der geläufigen Version V2 des Simon's Basic ist es leider nicht möglich (wie schon vom MOVE-Befehl aus 9.4.3 her bekannt), im TEXT-Befehl die Parameter g und a variabel durch Rechnungen oder Speicheraufruf zu bestimmen, da ansonsten Teile des auszugebenden Textes verschwinden oder weitere Zeichen angehängt werden. Sie dürfen also lediglich Ziffern für diese Werte einsetzen. Ansonsten erfüllen sie natürlich ihren Zweck.

#### BEISPIELE:

**NUN SOLLTEN SIE SICH ZUNÄCHST EINMAL WIEDER MIT DEN PARAMETERN UND IHREN AUSWIRKUNGEN BESCHÄFTIGEN. OAZU WIE IMMER EIN ÜBUNGSPROGRAMM:**

```
100 REM #####
110 REM ##                               ##
120 REM ## TEXT-BEISPIEL-1 ##
130 REM ##                               ##
140 REM #####
150 REM
160 HIRES 1.0 : REM GRAPHIK EIN
170 REM TESTBEISPIEL:
180 REM BITTE AENDERN SIE DIE PARAMETER!
190 REM
200 REM   X!  Y!   STR      IZM! G!  A
210 REM  ----+----+-----+---+---+--
220 REM      !  !           !  !  !
230 TEXT 50,100,"ADEMO-TEXT", 1, 2, 30
240 REM  ----+----+-----+---+---+--
250 WAIT 198,255 : REM AUF TASTE WARTEN
```

-----

```
100 REM #####
110 REM ##          ##
120 REM ##  TEXT-BEISPIEL-2  ##
130 REM ##          ##
140 REM #####
150 REM
160 HIRES 9,10 : REM GRAPHIK EIN
170 FOR X=1 TO 8
180 TEXT 100, 25*X-20, "CBM 64",1,3,20
190 NEXT X
200 PAUSE 5
210 REM
220 REM
230 REM
240 HIRES 9,10 : REM GRAPHIK LOESCHEN
250 REC      4, 4,311,191,1
260 LINE     0, 0,319,199,1
270 LINE    319, 0, 0,199,1
280 BLOCK   90,80,220,120,1
290 BLOCK  100,90,210,110,0
300 TEXT   103,97, "SIMON'S BASIC",1,1,8
310 WAIT  198,255 : REM AUF TASTE WARTEN
```

### 12.5.3 TESTAUFGABEN

Wie Sie wahrscheinlich schon begierig erwartet haben, kommen nun wieder unsere heißbegehrten Testaufgaben zur Wissenskontrolle. Na denn, auf in den Kampf!

1.) Es soll ein großes normales A auf den Bildschirm an die Stelle mit den Koordinaten 160,100 in Originalgröße gebracht werden. Welche Befehle verwenden Sie?

- a) CHAR 160,100,128+1,1,1
- b) TEXT 160,100,"A",1,1,1
- c) CHAR 160,100,1,1,1
- d) TEXT 160,100,"A",1,2,2

2.) Was passiert mit den zu zeichnenden Zeichen, wenn Sie in CHAR oder in TEXT den Parameter, den wir g genannt haben vergrößern?

- a) Die Zeichen werden vertikal verkleinert
- b) Die Zeichen werden in x-Richtung vergrößert
- c) Die Zeichen werden in y-Richtung vergrößert
- d) Die Zeichen werden in beide Richtungen vergrößert
- e) Die Zeichen werden horizontal verkleinert
- f) keine dieser Möglichkeiten

3.) Wie haben Sie den Parameter a bei TEXT zu wählen, wenn Sie die zu zeichnenden Buchstaben mit einem Zwischenraum von jeweils einem Punkt auf den Bildschirm bringen wollen?

- a) a=0
- b) a=1
- c) a=7
- d) a=8
- e) a=9

## 13. KAPITEL

### ZEICHENSATZERSTELLUNG

Neben den ungewöhnlich variationsreichen Graphikmöglichkeiten bietet Ihnen Ihr Commodore 64 noch weitere Kostbarkeiten. Eine dieser Fähigkeiten ist die Softwaremäßige Veränderung des Zeichensatzes, eine Eigenschaft, die Sie noch zu schätzen lernen werden. Sie ist die Grundlage fast aller Spiele und ist dasjenige Mittel (neben den Sprites (=MOBs)), das alle Spiele auf dem CBM 64 so unheimlich schnell und trotzdem graphik- und effektreich werden läßt. Ohne diese Möglichkeit ist eine vernünftige Spielprogrammierung undenkbar geworden. Wo sich andere Computer mit riesigen Graphikspeichern herumquälen, dort schnippst Ihr Commodore 64 einmal mit dem kleinen Finger. Doch stellen wir uns erst einmal die Frage:

Was ist denn eigentlich ein "Zeichensatz"?

Nun, das ist oberflächlich betrachtet recht einfach zu erklären. Unter Zeichensatz verstehen wir die Gesamtheit aller Zeichen (Buchstaben und Graphikzeichen), die Sie im Textmodus durch Dücken verschiedener Tasten und Tastenkombinationen (siehe <shift> und <C=> (Commodore - Taste) auf den Bildschirm bringen können.

Die Form und das Aussehen dieser Zeichen muß dem Computer natürlich bekannt, also irgendwo und irgendwie gespeichert sein. Gleichzeitig sollten Sie auch nach irgendwelchen Kriterien geordnet sein, damit Ihr Rechner weiß, daß er beispielsweise ein A auf den Bildschirm bringen soll, wenn Sie die Taste A drücken. Diese Informationen sind natürlich in allen Computern gespeichert.

Beim CBM 64 ist dieser Speicher so angelegt, daß er von irgendwelchen Programmen aus erreichbar ist, d.h. sein Inhalt kann ausgelesen und beispielsweise irgendwo in einen anderen Speicherbereich kopiert (übertragen) werden. Dies alleine nützt uns natürlich noch nicht viel. Wir könnten lediglich sehen, wie unser Computer die einzelnen Zeichen erstellt, da wir den Inhalt dieses Speichers nicht verändern können (wir nennen einen solchen Speicher bekanntlich ROM (= Read Only Memory = Lesespeicher), der nur von uns gelesen werden kann, dafür aber erhalten bleibt, auch wenn der Rechner

ausgeschaltet wird; im Gegensatz hierzu wird der RAM (= Random Access Memory = Schreib-/Lesespeicher), in dem alle Ihre eigenen Programme und Speicher stehen, beim Ausschalten des Gerätes gelöscht. Das normale Basic-Betriebssystem liegt ja - wie Sie wissen - ebenfalls in einem ROM.).

Nun aber besitzt dieser uns bekannte Rechner die Fähigkeit, die Speicheradresse, aus der er die Form der einzelnen Zeichen abliest, zu verändern. Sie haben also die Möglichkeit, Ihrem CBM 64 zu sagen, daß er sich die Zeichengestalt von nun an z.B. aus dem Speicherbereich ab \$2000 (= 8192) also aus dem RAM holen soll. Diesen Speicherbereich können wir natürlich nun selbst verändern. Haben wir nun vorher den Zeichensatz aus dem ursprünglichen ROM in diesen Bereich kopiert, so bemerken wir zunächst keine Änderung, da ja alle Information erhalten geblieben ist. Verändern wir nun jedoch Teile dieses Speicherbereiches, so ändern wir damit gleichzeitig die Form eines bestimmten Zeichens. Damit stellt sich gleich die nächste Frage:

Wie wird die Form eines Zeichens gespeichert?

Nun, zunächst einmal wissen Sie, daß Ihr Computer insgesamt 4 Zeichensätze mit je 128 Zeichen besitzt, von denen jeweils nur 2 gleichzeitig auf dem Textbildschirm erscheinen. Wir wollen im Folgenden diese vier Zeichensätze kurz benennen:

- SATZ A/1 - NORMAL - GROBSCHRIFT/GRAPHIKZEICHEN**
- SATZ A/2 - INVERS - GROBSCHRIFT/GRAPHIKZEICHEN**
- SATZ B/1 - NORMAL - GROB-/KLEINSCHRIFT**
- SATZ B/2 - INVERS - GROB-/KLEINSCHRIFT**

Bekanntlich können Sie die beiden Zeichensätze A und B durch die gleichzeitige Betätigung der Tasten <C=> und <shift> von Hand aus wechseln. Vom Programm aus dienen hierzu die ASCII-Werte 14 und 142 (Anmerkung: 142 = 128+14), d.h. Sie können mit

**PRINT CHR\$(14);**

auf Satz B und mit

**PRINT CHR\$(142);**

auf Satz A umschalten (s. hierzu auch die Befehle CSET 0 und CSET 1). Mit

### **PRINT CHR\$(8)**

blockieren Sie dabei die Möglichkeit der Umschaltung über die Tastatur, die ja auch während des Laufens eines Programmes möglich ist, und mit

### **PRINT CHR\$(9)**

heben Sie diese Blockade wieder auf (s. hierzu auch das CBM 64 - Benutzerhandbuch auf den Seiten 135-137).

Die Umschaltung zwischen Sätzen 1 und 2 bewerkstelligen Sie durch die Verwendung von <RVS ON> und <RVS OFF>.

In dem Zeichensatzspeicher müssen natürlich alle diese 4 Zeichensätze getrennt aufgelistet sein. Sie haben also die Möglichkeit  $4 \times 128 = 512$  Zeichen zu verändern (Wie gesagt, können davon jedoch nur jeweils 256 verschiedene Zeichen gleichzeitig angezeigt werden.).

Jedes Zeichen besteht auf dem Bildschirm aus einer Matrix von  $8 \times 8$  Punkten, wie Sie vielleicht schon wissen. Entsprechend müssen also im Zeichensatzspeicher diese insgesamt 64 Punkte repräsentiert sein. Dies wird erreicht, indem jeder Punkt des Zeichens auf dem Bildschirm ähnlich wie in HGR durch ein Bit im Speicher vertreten ist. Somit setzt sich ein Zeichen - Bit - Muster aus 8 Byte zu je 8 Bit zusammen. Jedes Byte repräsentiert eine der 8 Zeilen des Zeichens. Ein gesetztes Bit bedeutet also einen gesetzten Punkt des Zeichens. Wir können uns die Speicherung eines Zeichens wie folgt vorstellen:

<b>BIT</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
<b>BYTE 0</b>	.	.	.	.	.	.	.	.
<b>BYTE 1</b>	.	.	.	.	.	.	.	.
<b>BYTE 2</b>	.	.	.	.	.	.	.	.
<b>BYTE 3</b>	.	.	.	.	.	.	.	.
<b>BYTE 4</b>	.	.	.	.	.	.	.	.
<b>BYTE 5</b>	.	.	.	.	.	.	.	.
<b>BYTE 6</b>	.	.	.	.	.	.	.	.
<b>BYTE 7</b>	.	.	.	.	.	.	.	.

Der Zeichensatzspeicher ist also aus insgesamt 512 hintereinanderliegender Definitionen dieser Art zu je 8 Bytes zusammengesetzt. Er benötigt also einen Speicherbereich von 4 K (= 4096 Bytes), der normalerweise im ROM von \$D000 - \$DFFF (dezimal: 53248 - 57344) liegt. Dieser Bereich ist jedoch von

Basic aus nicht auszulesen. Zur Demonstration sei an dieser Stelle gezeigt, wie ein normales, großes A im Zeichensatzspeicher definiert wird:

```

      BIT   7 6 5 4 3 2 1 0
  BYTE 0   . . . * * . . .
  BYTE 1   . . * * * * . .
  BYTE 2   . * * . . * * .
  BYTE 3   . * * * * * * .
  BYTE 4   . * * . . * * .
  BYTE 5   . * * . . * * .
  BYTE 6   . * * . . * * .
  BYTE 7   . . . . . . . .
  
```

**WIR ERHALTEN ALSO FOLGENDE 8 BYTES:**

```

  BYTE 0: 00011000 = $18 = 024
  BYTE 1: 00111100 = $3C = 060
  BYTE 2: 01100110 = $66 = 102
  BYTE 3: 01111110 = $7E = 126
  BYTE 4: 01100110 = $66 = 102
  BYTE 5: 01100110 = $66 = 102
  BYTE 6: 01100110 = $66 = 102
  BYTE 7: 00000000 = $00 = 000
  
```

Diese acht Werte stehen nun an der Stelle in dem Zeichensatzspeicher, die für das große, normale A reserviert ist. Wie erhält man denn nun diese Position der Definition eines Zeichens? Nun, Ausgangspunkt aller Berechnung sind die Bildschirmcodes der einzelnen Zeichen, der Codes also, die zur Bestimmung eines Zeichens im Bildschirmspeicher stehen (s. CBM 64 - Benutzerhandbuch). Der Rest ist relativ einfach: Da jedes Zeichen 8 Byte benötigt, müssen wir nur den Wert des Bildschirmcodes mal 8 nehmen und die Basisadresse des Zeichenspeichers, also die Anfangsadresse, bei der unser Zeichenspeicher beginnt (normal: \$D000), hinzuaddieren. Es ergibt sich die Formel:

**ADRESSE = BASISADRESSE + 8 \* BILDSCHIRMCODE**

Für das Zeichen A im normalen Zeichenspeicher wäre dieses:

**ADRESSE = \$D000 + 8 \* 1 = \$D008 = 53248**

Jetzt haben Sie einen kleinen Einblick, wie die Zeichensatzerstellung vor sich geht. Weiteres bezüglich der Verwirklichung im Simon's Basic lesen Sie bitte unter den einzelnen Befehlen nach.

### **13.1 MEM**

**FORMAT** :MEM  
**PARAMETER** : ----  
**BEISPIEL** :MEM  
**FUNKTION** :VERLEGEN DES ZEICHENSATZES VON  
\$D000 (ROM) NACH \$E000 (RAM)

#### **ERLÄUTERUNGEN:**

Wie wir in der Einleitung zu diesem Kapitel gesehen haben, liegt der Zeichensatz im Ursprungszustand in 4 K ROM von \$D000 bis \$DFFF. Da es unser Ziel ist, die Definition einiger Zeichen zu ändern, müssen wir ihn zwangsläufig in einen RAM-Bereich verlegen. Diese Funktion (und noch einige andere) führt der Befehl MEM aus. MEM verlegt den Zeichensatz dabei unter den Speicherbereich des Betriebssystems, also von \$E000 - \$EFFF (dezimal: 57344 - 61439). Wie sie sehen, liegt hier gleichfalls ein Teil des Graphikspeichers (hexadezimal: \$E000 - \$FFFF / dezimal:57344 - 65535), was dazu führt, daß nicht gleichzeitig Graphik verwendet und der Zeichensatz verändert werden kann. Tun Sie dies trotzdem, so kommt es zu merkwürdigen Effekten. Zur Veranschaulichung sei das folgende Programm gegeben:

**10 HIRES 6.7 : REM GRAPHIK EINSCHALTEN**  
**20 MEM : REM ZEICHENSATZ VERSCHIEBEN UND COPIEREN**  
**30 WAIT 198.255 : REM AUF TASTE WARTEN**

Wie sie sehen, wird zunächst auf Graphik umgeschaltet und dann der Zeichensatz nach \$E000 verschoben (also auch hinüber copiert). Der gesamte Zeichensatz erscheint damit im oberen Graphikbild, da sich - wie gesagt - die beiden Bereiche überschneiden. Da sich der Aufbau der Zeichen und der Graphik ähneln, können Sie die Zeichen richtig original erkennen. Ein



weiterer Effekt kommt hinzu: nach dem Verlassen des Programms kommen Sie nicht - wie gewohnt - wieder in den Textmodus zurück, sondern verbleiben im Graphikmodus (s. auch Kapitel 12). Gleichzeitig erscheint Text, den Sie (eigentlich unsichtbar) schreiben, als kleine Farbquadrate auf dem Graphikbildschirm, da sich der Videoram (also der Farbspeicher und zugleich Bildschirmspeicher) durch den MEM-Befehl nun bei \$CC00 (52224) befindet. Da jedoch Simon's Basic im Graphikmodus annimmt, daß (wie üblich) der Farbspeicher bei \$C000 (49152) liegt, können Sie die Farbe durch keinen Graphikbefehl ändern, obwohl diese voll funktionieren (s. hierzu auch Kapitel 12). Erst durch CSET oder NRM kommen Sie wieder zurück in den Text (doch wundern Sie sich auch hier nicht über einige Ungereimtheiten).

Zusammenfassend können wir also sagen: Der Befehl MEM führt folgende Funktionen aus:

- A) VERSCHIEBEN DES ZEICHENSATZ-ROM-INHALTES NACH \$E000 (57344).**
- B) DEM COMPUTER WIRD MITGETEILT, DAß SICH DER ZEICHENSATZ NUN DORT BEFINDET.**
- C) VERSCHIEBEN DES VIDEORAMS (BILDSCHIRM- ODER TEXTSPEICHER) NACH \$CC00 (52224).**
- D) BILDSCHIRM LÖSCHEN**
- E) VERLEGEN DES SPRITE-BEREICHES NACH \$C000 (S.KAPITEL 14).**

Damit können wir nun Änderungen an den einzelnen Zeichen vornehmen.

**BEISPIEL:**

```

100 REM #####
110 REM ##                ##
120 REM ## MEM-BEISPIEL ##
130 REM ##                ##
140 REM #####
150 REM
160 REM
170 REM#####
180 REM*****##

```

```

190 MEM : REM ZEICHENSATZ INS RAM *##
200 REM*****##
210 REM#####
220 REM
230 BA = 14*4096 : REM BASISADRESSE ($E000)
240 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
250 REM
260 REM #####
270 REM ## ##
280 REM ## ALLE ZEICHEN AUFLISTEN ##
290 REM ## ##
300 REM #####
310 REM
320 FOR Y=0 TO 24
330 FOR X=0 TO 39 STEP 4
340 FILL Y.X.4.1.C.0
350 C = C+1 : REM CODE ERHOEHEN
360 NEXT X
370 NEXT Y
380 PAUSE 2
390 REM
400 REM #####
410 REM ## BITTE VERAENDERN: ##
420 REM### ##
430 REM***** ##
440 C = 0 : REM BILDSCHIRMCODE *##
450 REM*****##
460 REM#####
470 REM
480 REM #####
490 REM ## ##
500 REM ## ZEICHEN BLINKEN ##
510 REM ## ##
520 REM #####
530 REM
540 FOR X=0 TO 40
550 T = ABS(T-255) : REM ZWISCHEN 0 UND 255 WECHSELN
560 FOR Y=0 TO 7
570 POKE BA + C*8 + Y.T : REM BLINKEN LASSEN
580 NEXT Y
590 NEXT X

```

```

600 REM
610 REM #####
620 REM ##          ##
630 REM ##  ZEICHEN AENDERN  ##
640 REM ##          ##
650 REM #####
660 REM
670 REM          %%%%%%%%%%
680 POKE BA + C*8 + 0. % 01100000 :
690 POKE BA + C*8 + 1. % 01101100 :
700 POKE BA + C*8 + 2. % 01111110 :
710 POKE BA + C*8 + 3. % 00110110 :
720 POKE BA + C*8 + 4. % 00000110 :
730 POKE BA + C*8 + 5. % 00001111 :
740 POKE BA + C*8 + 6. % 00000110 :
750 POKE BA + C*8 + 7. % 00000000 :
760 REM          %%%%%%%%%%
770 REM
780 WAIT 198.255 : REM AUF TASTE WARTEN
790 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
800 LIST

```

In diesem Programm wird zunächst einmal in Zeile 190 der Zeichensatz verschoben, um ihn verändern zu können. Als nächstes werden (fast) alle Zeichen der Zeichensätze A/1 und A/2 aufgelistet, um jeweilige Veränderungen feststellen zu können. Das zu verändernde Zeichen wird kurz aufblinken gelassen, und anschließend schreibt das Programm die neu entworfenen 8 Bytes (Zeilen 680-750; hier sollen Sie Ihre eigenen Zeichen erstellen) mittels der in der Einleitung entwickelten Adressenformel an diejenige Stelle des Zeichensatzspeichers, die für das betreffende Zeichen (der Speicher C enthält den Bildschirmcode des Zeichens) zuständig ist. Nun können Sie in den Zeilen 680 bis 750 Eigencreationen editieren (vergessen Sie nicht, nach jeder Zeile ein <return> einzugeben.). Zur Zeichendefinition sehen Sie bitte auch unter # 13.3 nach.

## 13.2 DESIGN 2

**FORMAT** : DESIGN 2.ZA  
**PARAMETER** : - ZA: ADRESSE DER DEFINITION  
DES ZU ÄNDERNDEN ZEICHENS  
BERECHENBAR NACH DER  
FORMEL:  $ZA = \$E000 + B * C$   
WOBEI C = BILDSCHIRMCODE  
DES ZEICHENS  
**BEISPIEL** : DESIGN 2.  $\$E000 + B * 1$   
**FUNKTION** : UMDEFINIEREN EINES ZEICHENS

### ERLÄUTERUNGEN:

Wie Sie unter MEM gesehen haben, könnten wir bereits ohne Weiteres ein Zeichen undefinieren. Etwas komfortabler geschieht dies jedoch durch zwei weitere Befehle des Simon's Basic, die eng zusammengehören und von denen der erste nun erläutert werden soll.

DESIGN kann erst nach dem Befehl MEM eingegeben werden und legt nun fest, welches Zeichen undefiniert werden soll und bestimmt durch seine Position im Programm, welche Zeichendefinition für dieses Zeichen verwendet werden soll. Dies geschieht folgendermaßen:

Mit za übergeben Sie die Information, für welches Zeichen nun die neue Creation gelten soll. za stellt dabei die Adresse der Zeichendefinition (Zeichenmuster) im Zeichensatzspeicher dar und wird nach der uns bekannten - da in der Einleitung angeführten - Formel

**ADRESSE = STARTADRESSE + B\*BILDSCHIRMCODE**

bestimmt. Da durch den zuvor eingegebenen MEM-Befehl der Zeichensatz nach  $\$E000$  (57344) verschoben wurde (s.o.), lautet unsere Startadresse natürlich  $\$E000$ . Der Bildschirmcode hängt nun von dem Zeichen ab, das Sie verändern wollen und ist bekanntlich im CBM 64 - Benutzerhandbuch auf den Seiten 133/134 angegeben. Für den Buchstaben A lautet die Formel also:

**$ZA = \$E000 + B * 1 = \$E00B$**

Statt  $\$E000$  kann natürlich auch die entsprechende Dezimalzahl angegeben werden. Um nun festzulegen, welches neue Zeichenmuster, das in sogenannten Musterzeilen (s. 13.3) angegeben ist, statt des Buchstaben A zukünftig auf dem

Bildschirm erscheinen soll, müssen Sie den DESIGN-Befehl direkt vor die Zeichendefinition setzen (s. # 13.3). Ansonsten erhalten Sie die Fehlermeldung BAD CHAR FOR A MOB. Alles Weitere erfahren Sie unter # 13.3.

### 13.3

```
FORMAT      : @...ZEICHENMUSTER...  
PARAMETER   : - ZEICHENMUSTER: S.U.  
BEISPIEL    : S.U.  
FUNKTION    : SPEICHERN DES ZEICHENMUSTERS
```

#### ERLÄUTERUNGEN:

Nun kommen wir zu dem zentralen Befehl der Zeichenumdefinierung, dem Klammeraffen. Der Klammeraffe hat ähnliche Funktion wie das DATA-Statement des normalen Basic. Er steht jeweils am Anfang einer Zeile (im Folgenden "Musterzeile" genannt) und ist das Kennzeichen für eine Reihe von Daten. In diesem Falle ist dies die Definition des Zeichens. Diese wird auf folgende Art und Weise bewerkstelligt:

Wie Sie aus der Einleitung zu diesem Kapitel wissen, setzt sich ein Zeichen aus einer 8x8-Punktematrix zusammen. Entsprechend sieht auch die Zeichendefinition aus. Jede der 8 für ein Zeichen notwendigen Zeilen wird in einer separaten Basiczeile eingegeben. Jede Zeile enthält dabei das Muster für die 8 Punkte einer Zeichenzeile in der folgenden Form (hier für das Beispiel eines großen, normalen A):

```
100 @ ...BB...  
110 @ ..BBBB..  
120 @ .BB..BB.  
130 @ .BBBBBB.  
140 @ .BB..BB.  
150 @ .BB..BB.  
160 @ .BB..BB.  
170 @ .....
```

WIE SIE SEHEN, SETZEN SIE FÜR JEDEN IM ZEICHEN GESETZTEN

**PUNKT EIN B IN DAS ZEICHENMUSTER. EINEN PUNKT DAGEGEN FÜR DIE PUNKTE. DIE NICHT GESETZT WERDEN. SIE ERHALTEN SO EIN GROßES ABBILD DES ZUKÜNFTIGEN ZEICHENS SCHON IM BASIC - LISTING (SIE WERDEN UNTER KAPITEL 14 SEHEN. DAB SPRITES ÄHNLICH DEFINIERT WERDEN).**

**DIREKT EINE ZEILE VOR DIESER DEFINITION MUß NUN DER ENTSPRECHENDE DESIGN-BEFEHL STEHEN. UM ZU BESTIMMEN, WELCHES ZEICHEN NUN DAS NEUE MUSTER ERHALTEN SOLL. IN ZEILE 90 DES VORSTEHENDEN BEISPIELS MÜßTEN SIE NUN EINGEBEN:**

**90 DESIGN 2, \$E000 + C\*8**

wobei C wie bekannt der Bildschirmcode ist. Um die Sache vollständig zu machen, setzen Sie noch

**80 MEM**

davor, um den Zeichensatz zunächst einmal zu verschieben und schon ist Ihr zeichen umdefiniert (Wählen Sie in diesem Beispiel C=1, so ändert sich natürlich nichts, da dieses Zeichen (A) ja schon genauso aussieht).

Statt der vielen Punkte in einer Musterzeile können Sie auch Leerzeichen setzen, wobei jedoch gewährleistet sein muß, daß mindestens eines der zwei letzten Zeichen einer Zeile kein Leerzeichen ist, da sonst ein BAD CHAR FOR A MOB als Fehlermeldung ausgegeben wird, was gleichfalls passiert, wenn ein falsches oder zuviele Zeichen in einer Zeile stehen.

Mit Hilfe dieser Befehle können Sie nun Ihre eigenen Sonderzeichen oder gar einen eigenen ganzen Zeichensatz erstellen. Da dies natürlich sehr umständlich und speicherplatzaufwendig ist, sollten Sie hierfür spezielle Zeichensatzerstellungsprogramme verwenden und somit direkt einen ganzen Zeichensatz in den Speicher laden, oder mit Hilfe der sogenannten Overlay - Technik zunächst das Programm laden, das den neuen Zeichensatz erstellt. Nach dem Ablauf läd dieses Programm selbsttätig das neue oder eigentliche Programm ein. Dafür müssen Sie jedoch wissen, wie lang dieses zweite Basic-Programm sein soll. Ist es kürzer als das Zeichendefinierungsprogramm, so haben Sie keine Probleme und können einfach durch ein

**200 LOAD "PROGRAMMNAME".8**

(z.B. angehängt an unser kleines Beispielprogramm) das zweite Programm einladen, welches automatisch gestartet wird. Ist nun das nachzuladende Programm länger als das erste, so müssen Sie vorher durch ein

**190 POKE 45,AD - INT(AD/256)\*256**

**195 POKE 46, INT(AD/256)**

mitteilen, wie lang das nachzuladende Programm sein wird, wobei AD die Adresse des Basic - Speichers beinhaltet, bis zu der Ihr Programm gehen wird. Diese wiederum können Sie im Simon's Basic durch die Formel

**AD = 30720 - FRE(0) + \$801**

errechnen (natürlich nur wenn Sie das Programm im Speicher haben). Diese von nun an feste Zahl fügen Sie dann in das erste Programm ein. Für Interessierte: 30720 ist die Zahl der frei zur Verfügung stehenden Bytes im Simon's Basic, wenn kein Programm im Speicher ist; FRE(0) ergibt die Anzahl der frei zur Verfügung stehenden Bytes, wenn Ihr nachzuladendes Programm im Speicher steht. Diese Differenz 30720 - FRE(0) errechnet also die Anzahl der Bytes, die Ihr Programm einnimmt. Hierzu addieren Sie nun die Startadresse des Basicprogramms (normalerweise \$801, also 2049 dezimal) und schon haben Sie die Adresse des Basicprogrammendes. Also haben Sie in das Beispielprogramm die Zeile

**180 AD = 100**

einzufügen, wenn Ihr Programm 100 Bytes lang ist. Soweit eine "Ultrakurzeinführung" in die sogenannte Overlay - Technik, bei der Sie natürlich noch einige Dinge mehr zu beachten haben, die jedoch faszinierende Möglichkeiten eröffnet, da Sie mehrere Basicprogramme hintereinander ausführen können. Wenn Sie nicht alles verstanden haben, so ist das für den Anfang nicht weiter schlimm. Sie sollten sich aber unbedingt ein Handbuch verschaffen, das solche Profi - Programmierung ausgiebig erläutert, wie z.B. das Buch "64 für Profis" von DATA BECKER.

Viel Spaß beim Ausprobieren!

### **BEISPIELE:**

**100 REM #####**

**110 REM ## ##**

**120 REM ## @ /DESIGN 2-BEISPIEL ##**

**130 REM ## ##**

**140 REM #####**

**150 REM**

**160 MEM : REM ZEICHENSATZ VERSCHIEBEN**

```

170 PRINT : REM LEERZEILE
180 CENTRE "VIELLEICHT SCHREIBEN SIE DEMNAECHST" : PRINT
190 CENTRE "ALLE IHRE TEXTE MIT EINEM ANDEREN" : PRINT
200 CENTRE "ZEICHENSATZ!" : PRINT
210 PRINT : REM LEERZEILE
220 CENTRE "A B C D E"
230 PAUSE 10
240 REM
250 REM #####
260 REM ##                               ##
270 REM ## ZEICHENDEFINITION ##
280 REM ##                               ##
290 REM #####
300 REM
310 DESIGN 2, $E000 + 1*8 : REM A
320 @ .BBBBBBB
330 @ ..BB..BB
340 @ .BBBBBBB
350 @ .BBBBBBB
360 @ ..BB..BB
370 @ ..BB..BB
380 @ .BBB..BB
390 @ .....
400 DESIGN 2, $E000 + 2*8 : REM B
410 @ .BBBBBBB
420 @ ..BB..BB
430 @ ..BB..BB
440 @ .BBBBBBB
450 @ ..BB..BB
460 @ ..BB..BB
470 @ .BBBBBBB
480 @ .....
490 DESIGN 2, $E000 + 3*8 : REM C
500 @ .BBBBBBB
510 @ ..BB..BB
520 @ ..BB....
530 @ ..BB....
540 @ ..BB....
550 @ ..BB..BB
560 @ .BBBBBBB
570 @ .....

```



580 DESIGN 2. \$E000 + 4\*8 : REM D  
590 @ .BBBBBBB  
600 @ ..BB..BB  
610 @ ..BB..BB  
620 @ ..BB..BB  
630 @ ..BB..BB  
640 @ ..BB..BB  
650 @ .BBBBBBB  
660 @ .....  
670 DESIGN 2. \$E000 + 5\*8 : REM E  
680 @ .BBBBBBB  
690 @ ..BB..BB  
700 @ ..BB....  
710 @ ..BBBBB.  
720 @ ..BB....  
730 @ ..BB..BB  
740 @ .BBBBBBB  
750 @ .....  
760 PAUSE 15  
770 PRINT CHR\$(147) : REM BILDSCHIRM LOESCHEN  
780 CENTRE "MIT MEM KOENNEN SIE NATUERLICH" : PRINT  
790 CENTRE "ALLES WIEDER RUECKGAENGIG MACHEN"  
800 PAUSE 10  
810 MEM  
820 PRINT : REM LEERZEILE  
830 CENTRE "SEHEN SIE?" : PRINT  
840 PRINT : REM LEERZEILE  
850 CENTRE "A B C D E "  
860 WAIT 198.255 : REM AUF TASTE WARTEN

### 13.4 CSET 0/1

```
FORMAT      :CSET 0
ODER        CSET 1
PARAMETER   :-----
BEISPIEL    :CSET 0
FUNKTION    :UMSCHALTEN AUF ZWEITEN
              ZEICHENSATZ
```

#### ERLÄUTERUNGEN:

CSET haben wir bereits in der Form CSET 2 unter Abschnitt 12.2.4 kennengelernt. Dort hatte es die Funktion, die Graphik einzuschalten, ohne sie zu löschen. Nun lernen wir zwei weitere Arten des CSET-Befehls kennen. Ihre Funktion ist eigentlich recht leicht umrissen. CSET 0 entspricht 100%ig dem Befehl NRM (dessen Funktionen Sie unter 9.3.3 nachlesen können) und schaltet somit neben einigen anderen Dingen den Großschrift-/Graphikzeichenmodus ein. Diese Teilfunktion können Sie, wie Ihnen sicher bekannt ist auch mit

#### PRINT CHR\$(142):

ersetzen (natürlich nur diese Teilfunktion).

Mit CSET 1 nun besitzen Sie die Möglichkeit den alternativen Zeichensatz, also die Klein-/Großschrift einzuschalten, während die übrigen Funktionen identisch erhalten bleiben (Ausschalten der Graphik, des extended Colour - Modus und des MEM-Modus). Die besagte Teilfunktion ist bekanntlich ebenfalls durch

#### PRINT CHR\$(14):

erreichen. Zum besseren Verständnis sei an dieser Stelle ein Beispiel angefügt:

#### BEISPIEL:

```
100 REM #####
110 REM ##                               ##
120 REM ## CSET 0/1-BEISPIEL ##
130 REM ##                               ##
140 REM #####
150 REM
160 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
170 CSET 0 : REM GROSS-/GRAPHIKZEICHEN
```

180 CENTRE "IM MOMENT BEFINDEN WIR UNS IM NORMALEN" : PRINT  
190 CENTRE "GROSSCHRIFT-/GRAPHIKZEICHEN-MODUS" : PRINT  
200 PRINT : REM LEERZEILE  
210 PAUSE 7  
220 CENTRE "WAS SICH JEDOCH SOFORT AENDERN WIRD!" : PRINT  
230 PRINT : PRINT : REM 2 LEERZEILEN  
240 PAUSE 5  
250 CENTRE "ACHTUNG!"  
260 PAUSE 2  
270 CSET 1 : REM KLEIN-/GROSSCHRIFT  
280 WAIT 198.255 : REM AUF TASTE WARTEN

### 13.5 TESTAUFGABEN

Was halten Sie an dieser Stelle von einer kleinen Klausur? Spickzettel herausholen, gute Kontakte zum Nachbarn sind Trumpf!

1.) Was ist bei der Verwendung von MEM zu beachten?

- a) MEM schaltet die Graphik aus
- b) MEM copiert den Zeichensatz nach \$E000
- c) Graphik und eigener Zeichensatz sind nicht zugleich möglich
- d) Der MEM-Zustand wird durch NRM und CSET 0 ausgeschaltet
- e) Der MEM-Zustand wird durch CSET 2 und CSET 1 ausgeschaltet

2.) Was ist bei DESIGN 2 zu beachten?

- a) DESIGN 2 schaltet die Graphik aus
- b) Mit DESIGN 2 wird ein Zeichenmuster übertragen
- c) Der Parameter hinter DESIGN 2 gibt die Adresse der Zeichendefinition an
- d) DESIGN 2 ist unter Graphikbedingungen nicht anzuwenden.

## 14. KAPITEL

### SPRITES (MOBS)

Eines der hervorstechendsten Merkmale Ihres Commodore 64 sind natürlich die Sprites (oder MOBs = Movable Objekt Blocks = bewegliche Objekte). Sprites sind eigenständige kleine Graphiken, die unabhängig voneinander und von dem übrigen Bildschirminhalt in dem Text- oder Graphikfenster bewegt werden können. Insgesamt haben Sie die Möglichkeit, 8 Sprites gleichzeitig auf den Bildschirm zu bringen.

Sprites können bezüglich Ihrer Farbe, Ihrer Größe und der Priorität vor den Hintergrundzeichen und auch gegeneinander variiert werden. Sie können Kollisionen zwischen Sprites untereinander und mit dem Hintergrund feststellen.

All diese Funktionen können sehr leicht mithilfe des VIC (Videocontroller 6567) und seinen Registern realisiert werden. Simon's Basic erleichtert zudem die Handhabung dieser Sprites noch enorm, sodaß Sie sämtliche Funktionen leicht und überschaubar auf wenige Befehle komprimiert vor sich liegen sehen.

Zunächst einmal wollen wir uns mit dem Aufbau der Sprites befassen:

Jedes Sprite besteht aus 504 Punkten, die Sie einzeln setzen können. Verwendet wird dabei eine 24x21-Punktematrix, d.h. ein Sprite ist 24 Punkte breit und 21 Punkte hoch. Innerhalb dieses Bereiches können Sie nun die unterschiedlichsten Graphiken oder Figuren erstellen. Wir können diesen Sachverhalt in einer kleinen Skizze darlegen:

	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3
0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
4	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
USW.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
20	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
21	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Wir wollen uns gar nicht mit dem genauen Aufbau der Sprites

im Speicher befassen, da dies im Simon's Basic gar keine Rolle mehr spielt. Hier können Sprites auf recht einfache Art und Weise creiert werden. Innerhalb des skizzierten Bereiches können Sie nun nach Lust und Laune Ihre Figur entwerfen. Sie haben die Möglichkeit, die Farbe der gesetzten Punkte getrennt von dieser Definition aus 16 möglichen zu bestimmen (s.u.).

Nicht jedes Sprite jedoch besitzt diesen 24x21-Punkte - Aufbau. Sie können jeweils zwischen hochauflösenden und sogenannten Multicolor - Sprites wählen. Erstere besitzen den gerade geschilderten Aufbau. Die Multicolor - Sprites hingegen werden ähnlich der Multicolor - Graphik gebildet. Aus diesem Grunde besitzt ein solches Sprite in x-Richtung die halbe Auflösung. Hier befinden sich also nur 12, jedoch doppelt breite Punkte in einer Zeile. Dafür aber kann ein Gebilde aus insgesamt vier verschiedenen Farben (mit der Hintergrundfarbe) zusammengesetzt sein, während, wie gesagt, ein hochauflösendes (HGR) Sprite nur zwei Farben beinhaltet. Diese vier Farben sind in verschiedenen Registern des VIC untergebracht (s. CBM 64 - Benutzerhandbuch auf den Seiten 153/154). Einschränkend muß jedoch gesagt werden, daß lediglich die Farbe 2 für alle Sprites unterschiedlich sein kann. Die anderen Farben (Farben 1 und 3 neben der Hintergrundfarbe) sind jeweils für alle Sprites gleich, da sie aus identischen Registern gewonnen werden. Entgegen den Angaben des CBM 64 Benutzerhandbuchs können auch in Multicolor sämtliche 16 Farben zur Erstellung Ihrer Figuren verwendet werden.

Neben der Farbe und der Definition der Sprites können noch einige Dinge verändert werden: Sie haben die Möglichkeit, Ihre Gebilde in x-, in y- oder in beide Richtungen um den Faktor 2 zu vergrößern. D.h. Ihr Sprite wird doppelt breit, doppelt hoch oder beides.

Zudem geben Sie an, welche Priorität das jeweilige Sprite vor dem Hintergrund besitzt, d.h. ob Ihr Sprite nun vor den Zeichen oder der übrigen Graphik steht (oder fliegt), sie also verdeckt, oder ob es sich dahinter befindet, durch diese also verdeckt wird. Gleichzeitig bestimmen Sie durch die Nummer eines Sprites (jedes der acht möglichen Sprites besitzt eine Nummer von 0 bis 7) die Priorität der Sprites

untereinander. Diejenigen Sprites mit der höheren Nummer besitzen die höhere Priorität, verdecken bei Überschneidungen also stets die Sprites mit einer niedrigeren Nummer. So können Sie recht einfach 3-dimensionale Effekte erzeugen.

Doch damit nicht genug. Ihr Commodore 64 bietet Ihnen noch eine Reihe weiterer Bonbons, die Ihnen das Leben versüßen und auf die Sie ganz schön scharf sein werden, wenn Sie wissen, um was es geht! Ihr Computer registriert nämlich jegliche Berührung (oder Kollision) eines Sprites mit einem Hintergrundzeichen (oder überhaupt mit einem Punkte auf dem Bildschirm) oder einem anderen Sprite. Dieses Ereignis wird ebenfalls in verschiedenen Registern des VIC abgelegt, was Sie jedoch nicht weiter zu interessieren braucht, da Ihnen Simon's Basic diese Arbeit weitgehend abnimmt. Hierbei sind nur einige Dinge zu beachten, die unter den jeweiligen Befehlen (DETECT und CHECK) aufgeführt sind. Wichtig ist lediglich die Tatsache, daß zwischen Kollisionen Hintergrund - Sprite und Sprite - Sprite differenziert werden kann. Im letzteren Fall kann noch entschieden werden, welche der acht Sprites zusammengestoßen sind.

Eine Sache sollte hier noch kurz erläutert werden: das Ablegen eines Sprites (MOBs) im Speicher Ihres CBM 64. Bei der Abspeicherung eines Sprites wird der gesamte Schreib- / Lesespeicher des Rechners in sogenannte Blöcke zu je 64 Bytes unterteilt. Jeder Block erhält eine spezifische Nummer. So besitzen die ersten 64 Bytes des Speichers (Adressen 0-63) die Blocknummer 0 usw. Wollen Sie nun eine Spritedefinition in den Speicher übertragen (was, wie Sie sehen werden, mit dem Befehl DESIGN realisiert wird), so müssen Sie diese zunächst einmal innerhalb eines solchen Blockes ablegen. Später geben Sie dann bei der Darstellung eines Sprites auf dem Bildschirm an, aus welchem Block der Computer die Definition (oder das Zeichenmuster) entnehmen soll. Auf diese Weise ist es möglich z.B. zwei Sprites (bis auf die Größe oder Farbe etwa) identisch aussehen zu lassen, einfach indem Sie zwei verschiedenen Spritenummern dem selben Speicherblock zuweisen. Bei der Nummerierung eines Blockes jedoch sind Sie auf die Werte 0-255 beschränkt. Aus diesem Grunde können Sie Ihre verschiedenen Figuren nur innerhalb von  $255 \cdot 64 = 16384$  Bytes (16 K) unterbringen. Wo diese 16 K im Speicher Ihres Computers liegen, bestimmt die Lage des Videoram

(Bildschirmspeichers). Diese ist im Simon's Basic folgendermaßen bestimmt:

ZUSTAND	BLOCKLAGE
NORMALER TEXTMODUS:	0 - 16383 (\$0000 - \$3FFF)
NEUER ZEICHENSATZ :	49152 - 65536 (\$C000 - \$FFFF)
GRAPHIKMODUS :	49152 - 65536 (\$C000 - \$FFFF)

D.h. wenn Sie gleichzeitig in Graphik und in Text arbeiten, liegen die Spritedefinitionen an unterschiedlichen Speicherorten, obwohl die Blocknummern identisch sind. Wenn Sie also abwechselnd in Graphik und Text arbeiten, müssen Sie diesen Sachverhalt berücksichtigen: Ein Sprite, das für den Textmodus in den Speicher übertragen wurde, wird in der Graphik nicht oder nur mit vollkommen unsinnigem Aussehen auftauchen, da der Speicherbereich der nun für den betreffenden Block zuständig ist, die Spritedefinition natürlich nicht enthält. Umgekehrt gilt entsprechendes. Dies können Sie lediglich dadurch umgehen, indem Sie das selbe Sprite ein zweites Mal in den Graphik - Block - Bereich übertragen, oder es beim Umschalten auf Graphik ausschalten. Noch eine Sache müssen Sie bei der Wahl des Blockes für eine Spritedefinition beachten: Vermeiden Sie diejenigen Blöcke, die Speicherbereiche bezeichnen, die bereits durch andere Dinge belegt sind (z.B. Nullseite etc.). Im folgenden seien hier einige mögliche Blöcke aufgestellt:

#### TEXTMODUS:

BLOCKNUMMER	ADRESSEN	BEMERKUNGEN
13 - 15	832- 1023 (\$0340-\$03FF)	OHNE EINSCHRÄNKUNG VERWENDBAR
64 -255	4096-16383 (\$1000-\$3FFF)	VORSICHT BEI LANGEN BASICPROGRAMMEN!

#### GRAPHIKMODUS:

BLOCKNUMMER	ADRESSEN	BEMERKUNGEN
16 - 63	50176-53247 (\$C400-\$CFFF)	MIT IEC-BUS V. DATA BECKER (S-VERSION) NUR BIS BLOCK 47

#### NEUER ZEICHENSATZ:



<b>BLOCKNUMMER</b>	<b>ADRESSEN</b>	<b>BEMERKUNGEN</b>
<b>16 - 47</b>	<b>50176-52223 (\$C400-\$CBFF)</b>	<b>EINGESCHRÄNK VERWENDBAR (S.U.)</b>
<b>192 -255</b>	<b>61440-65535 (\$F000-\$FFFF)</b>	<b>UNEINGESCHRÄNK VERWENDBAR</b>

Hierzu ist Folgendes zu sagen: bei der Verwendung der C-Seite (Blöcke 16-63 im Graphikmodus und unter Verwendung eines neuen Zeichensatzes) ist Vorsicht geboten. Irgendwo in diesem Bereich liegen einige Zwischenspeicher des Simon's Basic (unter Anderem die Belegung der Funktionstasten (Blöcke 25-30)). Es wäre möglich, daß in diesem Bereich noch weitere Zwischenspeicher liegen, die zur Ausübung bestimmter Funktionen notwendig sein könnten. Sollten Sie also mit diesem Bereich arbeiten, so speichern Sie zunächst einmal Ihr Programm ab, bevor Sie den ersten Probestart unternehmen (das sollten Sie übrigens immer tun, da Simon's Basic ab und zu zum Absturz neigt).

Sie sehen also, mit den Sprites werden Ihnen Möglichkeiten in die Hand gegeben, die Sie erst bei der Erprobung und Anwendung völlig ausschöpfen können. Sehr geeignet sind diese Figuren, die ja hardwaremäßig erzeugt werden und deswegen sehr schnell zu handhaben sind, zur Produktion von schönen Spielen oder auch Laufschriften. Lassen Sie sich etwas einfallen. Die einzelnen Beispielprogramme nach der Erläuterung der verschiedenen Befehle sollen Ihnen hier Ansätze und Ideen vermitteln, die bei Ihrem nächsten Programm vielleicht Anwendung finden.

Zunächst werden die Befehle erläutert, die zum Anfang einer Spritedefinition notwendig sind, bevor Sie dieses überhaupt auf dem Bildschirm zu sehen bekommen. Selbstverständlich können diese Dinge im Laufe des Programms wieder geändert werden, sollten jedoch einmal vor dem Sichtbarwerden definiert werden. Alsdann sehen Sie sich mit den Befehlen konfrontiert, die die fertigen Sprites endlich steuern. Lassen Sie sich überraschen. Viel Spaß!

## 14.1 SPRITEDEFINITIONEN

### 14.1.1 DESIGN 0/1

**FORMAT** : DESIGN 0.BA + B\*64  
**ODER** : DESIGN 1.BA + B\*64  
**PARAMETER** : - 0 : HOCHAUFLOSENDES SPRITE  
- 1 : MULTICOLOR - SPRITE  
- BA: BASISADRESSE DES VIDEO-  
BEREICHES:  
TEXTMODUS : BA=0  
GRAPHIK : BA=\$C000  
NEUER  
ZEICHENSATZ : BA=\$C000  
- B : NUMMER DES SPEICHERBLOCKES,  
BLOCKES, IN DEN DIE  
SPRITEDEFINITION UBERTRA-  
GEN WERDEN SOLL (0-255)  
**BEISPIEL** : DESIGN 0.\$C000 + 13 \* 64  
**FUNKTION** : UBERTRAGEN EINER SPRITEDEFI-  
NITION IN EINEN SPEICHERBEREICH

#### ERLÄUTERUNGEN:

Sicher erinnern Sie sich an den Befehl DESIGN 2 aus Abschnitt 13.2, der bei der Definition neuer Zeichen für den Textmodus eine Rolle spielte. Falls Sie diesen Teil des Buches noch nicht gelesen haben sollten, können Sie dies ruhig einmal tun. DESIGN 0 oder DESIGN 1 funktionieren vollkommen analog zu dem im erwähnten Kapitel dargestellten Befehl.

Mit DESIGN 0/1 übertragen Sie die Definition eines Sprites (MOBs), die (wie auch bei der Zeichensatzänderung) in sogenannten Musterzeilen (Klammeraffenzeilen; s. 14.1.2) abgelegt wird, in den dafür zuständigen Speicher. Der Befehl DESIGN (egal ob 0,1 oder 2) muß stets in der direkt vor den Musterzeilen befindlichen Basiczeile stehen, um die Fehlermeldung BAD CHAR FOR A MOB zu vermeiden (s. 14.1.2).

Die Ziffern 0 bzw 1 determinieren, welche Art von Spritedefinition Sie in den Speicher geben wollen. Dabei haben Sie, wie in der Einleitung zu diesem Kapitel dargelegt, die Möglichkeit, zwischen hochauflösenden (HGR-) Sprites mit

nur einer Punktfarbe (neben der Hintergrundfarbe) und sogenannten Multicolor - Sprites zu entscheiden, bei denen maximal 3 Farben (unter Beachtung der erwähnten Beschränkungen) zur Darstellung einer Figur Verwendung finden können. Wollen Sie nun die Definition eines HGR - Sprites in den Speicher übertragen, so wählen Sie die Ziffer 0 direkt hinter dem DESIGN - Wort (also: DESIGN 0), im anderen Falle eines Multicolor Sprites sollte es demnach heißen: DESIGN 1. Der zweite und dritte Parameter dienen wie bei 13.2 zur Bestimmung des Speicherbereiches, in den die Definition übertragen werden soll. Dabei sollten Sie das in der Einleitung hierzu Gesagte beherzigen. *ba* gibt hier also die Adresse des Blockes 0 (Basisadresse des Videoadressbereiches) und *b* die Nummer des anzusprechenden Blocks an. Alles Weitere entnehmen Sie bitte der Einleitung zu dem Kapitel 14 oder § 14.1.2

## 14.1.2

```
FORMAT      : e...MUSTERZEILE...  
PARAMETER   : - MUSTERZEILE: SPRITEDEFINITION  
                PUNKT (.): LEERPUNKT  
                B       : PUNKT MIT FARBE 1  
                C       : PUNKT MIT FARBE 2  
                D       : PUNKT MIT FARBE 3  
BEISPIEL    : e...B..BBBB....B...B.B.B.  
                e.. USW. (21 ZEILEN)  
FUNKTION    :HERSTELLEN EINER SPRITEDEFINITION
```

### ERLÄUTERUNGEN:

Auch diesen Befehl kennen Sie bereits aus dem 13. Kapitel als Kennzeichnung einer sogenannten Musterzeile, die dort die Aufgabe hatte, ein neues Zeichen zu creieren, das im Textmodus verwendet werden konnte. Hier nun dient der Klammeraffe dazu, eine Musterzeile einzuleiten, die die Definition einer Zeile eines Sprites enthält. Wie Sprites aufgebaut sind, sollte Ihnen aus Ihrem Benutzerhandbuch oder aus der Einleitung zu diesem Kapitel, die Sie unbedingt gelesen haben sollten, her bekannt sein. Wie dort beschrieben werden HGR- und Multicolor - Sprites unterschieden, die gleichfalls unterschiedlich definiert werden müssen. Bei der Verwendung von hochauflösenden (HGR-) Sprites besitzen Sie eine 24x21 - Punktematrix, in der angegeben wird, ob ein Punkt gesetzt oder nicht gesetzt wird. Im Simon's Basic ist ein HGR - Sprite somit aus insgesamt 21 Musterzeilen mit je 24 Punktanweisungen zusammengesetzt. Ein gesetzter Punkt wird durch ein B in dieser Definition symbolisiert; ein nicht gesetzter Punkt (nimmt die Hintergrundfarbe an) durch einen einfachen Punkt (.). In Multicolor nun haben Sie neben der Farbe, die Sie für jedes Sprite extra wählen können noch zwei weitere, die alle Sprites gemeinsam haben. Aus diesem Grunde stehen Ihnen hier die Buchstaben C und D zur Verfügung, um die entsprechenden Farben für einen Punkt zu symbolisieren. Gleichzeitig können nur noch 12 Punktanweisungen in einer Zeile auftauchen, da ja die Auflösung von Multicolor - Sprites bekanntlich nur halb so groß ist. Die Form einer Definition entnehmen Sie bitte dem Beispielprogramm. Noch ein Ding ist bei der Verwendung der Buchstaben zu

beachten: HGR - Sprites entnehmen die Farbe der mit B gesetzten Punkte der Farbinformation des MOB SET - Befehls (s.u.), Multicolor - Sprites ordnen die in MOB SET angegebene Farbe dem Buchstaben C zu. Wir erhalten also folgende Zuordnung:

<b>HOCHAUFLOSENDE SPRITES:</b>		
<b>BUCHSTABE</b>	<b>FARBNUMMER</b>	<b>ZUSTÄNDIGER FARBBEFEHL</b>
.	0	(TRANSPARENT)
<b>B</b>	<b>1</b>	<b>MOB SET</b>

<b>MULTICOLOR - SPRITES:</b>		
<b>BUCHSTABE</b>	<b>FARBNUMMER</b>	<b>ZUSTÄNDIGER FARBBEFEHL</b>
.	0	(TRANSPARENT)
<b>B</b>	<b>1</b>	<b>CMB</b>
<b>C</b>	<b>2</b>	<b>MOB SET</b>
<b>D</b>	<b>3</b>	<b>CMB</b>

Sollten Sie auf die Idee kommen, irgendwann einmal andere Buchstaben oder Zeichen in die Spritedefinition aufzunehmen (außer Leerzeichen unter gewissen Bedingungen (s. Kapitel 13), die Zeilen länger zu gestalten als erlaubt oder zwischen den Befehl DESIGN und den Musterzeilen eine weitere Basiczeile einzufügen, so muß ich Ihnen leider sagen: Unter diesen Umständen ist Ihr Rechner nicht mehr bereit, mit Ihnen zusammenzuarbeiten und empört sich mit einem BAD CHAR FOR A MOB als Fehlermeldung. Auch Computer kommen an den Punkt, an dem sie sich schamlos hintergangen fühlen. Sorgen Sie also dafür, daß solche Zwischenfälle im beiderseitigen Interesse vermieden werden, Ihr Computer wird es Ihnen danken - auf eine lange und gute Freundschaft.

#### BEISPIELE:

```

100 REM #####
110 REM ##          ##
120 REM ## /DESIGN-BEISPIEL ##
130 REM ##          ##
140 REM #####
150 REM
160 DESIGN 1.14*64 : REM DEFINITION IN BLOCK 14 UEBERNEHMEN

```

```

170 @ ...BBBBB....
180 @ .BBB...BBB..
190 @ BB.CC.CC.BB.
200 @ BB...C...BB.
210 @ BB...C...BB.
220 @ .BBBCCCBBB..
230 @ ...BBBBB....
240 @ B...000.....
250 @ BB..000.....
260 @ BBBBDDOBB...
270 @ ..BBDDOBB...
280 @ ....000.BB..
290 @ ....000..BB.
300 @ ....DDD...B.
310 @ ....000.....
320 @ ..CCC.CCC...
330 @ .CCC...CCC..
340 @ .CC.....CC..
350 @ CCC.....CCC.
360 @ CCC.....CCC.
370 @ BBBB BBBB BBBB
380 REM
390 REM #####
400 REM #####          #####
410 REM ##### ZUSATZ #####
420 REM #####          #####
430 REM #####
440 REM
450 COLOUR 9.9
460 MOB SET 1.14.8.0.1
470 CMOB 7.6
480 MMOB 1.0.0.300.205.2.200

```

Dieses Beispiel soll lediglich dazu dienen, Ihnen die Definition eines Sprites mithilfe der zwei bekannten Befehle näher zu bringen. Um dieses Sprite dann aber auf den Bildschirm zu bringen, sind noch eine Reihe weiterer Befehle notwendig, die hier erst einmal ab Zeile 460 ohne Kommentar hinten angefügt worden sind, damit Sie überhaupt einmal Ihr Produkt in natura betrachten können. Nehmen Sie diese Kommandos einfach so hin und kommen später vielleicht noch

einmal darauf zurück.

### 14.1.3 MOB SET

```
FORMAT      : MOB SET N.B.F.P.A
PARAMETER   : - N: NUMMER DES SPRITES (0-7)
              - B: NUMMER DES BLOCKES AUS
                  DEM DIE SPRITEDEFINITION
                  GEWONNEN UND DER SPRITE-
                  NUMMER ZUGEORDNET WIRD
                  (0-255: S. EINLEITUNG)
              - F: FARBE DES SPRITE (0-15)
                  HGR: BUCHSTABE B
                  MC : BUCHSTABE C
              - P: PRIORITÄT DES SPRITE VOR
                  DEN HINTERGRUNDZEICHEN:
                  P=0 : SPRITE VOR
                       HINTERGRUND
                  P=1 : SPRITE HINTER
                       HINTERGRUND
              - A: AUFLÖSUNG DES SPRITE:
                  A=0 : HOCHAUFLOSENDES
                       SPRITE
                  A=1 : MULTICOLOR - SPRITE
BEISPIEL    : MOB SET 0.13.7.1.0
FUNKTION     : EIGENSCHAFTEN DES SPRITE
              BESTIMMEN
```

#### ERLÄUTERUNGEN:

Wie Sie in der Einleitung zu diesem Kapitel hoffentlich gelesen haben, können Sie verschiedene Eigenschaften der Sprites (Farbe, Größe, Priorität, ...) getrennt von der eigentlichen Definition nachträglich (oder vorher) festlegen oder ändern. Ein Teil dieser Funktionen wird mit dem Befehl MOB SET erreicht. Wie Sie sehen, können Sie bei MOB SET eine ganze Reihe von Parametern verwenden, die jeweils ganz spezifische Aufgaben erfüllen. Eine Funktion jedoch führt dieser Befehl nicht aus, das muß gleich im Anfang gesagt werden: MOB SET schaltet kein Sprite an, d.h. kein Sprite

kann nur mit Hilfe dieses Befehls auf dem Bildschirm sichtbar gemacht werden.

Nun aber zu den einzelnen Parametern: Zunächst einmal geben Sie mit  $n$  an, um welches Sprite es sich überhaupt handeln soll. Wie Sie wissen, können Sie maximal 8 Sprites gleichzeitig auf dem Bildschirm darstellen. Jedes Sprite besitzt dabei eine ganz spezifische Nummer (0-7), die übrigens auch die Priorität der einzelnen Sprites unter sich regelt (s. Einleitung). Diese Nummer müssen Sie bei allen Spritebefehlen angeben, um darzulegen, auf welches Sprite sich der jeweilige Befehl bezieht.

Der zweite Parameter  $b$  bestimmt den Block, aus dem die Musterdefinition stammen soll, die vorher mit den inzwischen bekannten Befehlen DESIGN und Klammeraffe in den Speicher übertragen wurde. Diese Definition wird nun in unserem MOB SET - Befehl der angegebenen Spritenummer zugeordnet, sodaß später stets diese eine Figur bei dem Aufruf dieser Nummer angesprochen wird. Auf diese Art und Weise ist es beispielsweise ebenfalls möglich, mehrere Sprites gleichen Aussehens auf dem Bildschirm darzustellen, indem dem selben Block verschiedene Spritenummern zugeordnet werden. Bei der Blockauswahl sollten Sie die in der Einleitung beschriebenen Dinge beachten.

Um nun die Farbe des Sprites zu bestimmen, ist es notwendig, dem nächsten Parameter ( $f$ ) einen Wert zuzuordnen. Dieser Wert stellt den Farbcode (0-15) dar und besitzt in HGR- und Multicolorsprites unterschiedliche Bedeutung:

In hochauflösenden Figuren werden alle Punkte der in den Musterzeilen angegebenen Matrix, die mit einem B gekennzeichnet wurden in der angegebenen Farbe gezeichnet. In Multicolor - Sprites dagegen besitzen diese Aufgabe alle mit C gezeichneten Punkte.

Der nächste Wert legt die Priorität des Objektes vor den übrigen Hintergrundzeichen des Bildschirms fest.  $p$  kann lediglich 2 Werte annehmen: 0 oder 1. Im ersten Falle bewegt sich die Figur stets vor den übrigen Zeichen oder Punkten (sie verdeckt also den Hintergrund). Im zweiten Fall für  $p=1$  wird umgekehrt das Sprite verdeckt.

Zum guten Schluß legen Sie fest, ob die Sprite - Definition in hochauflösender ( $a=0$ ) oder Multicolor - Graphik ( $a=1$ ) dargestellt werden soll. Sie können also auch eine eigentlich



für Multicolor vorgesehene Definition unter HGR und umgekehrt  
laufen lassen.

Bei den beiden letzten Parametern werden größere Werte als 1  
als 0 angesehen.

Nun zu einem

### BEISPIEL:

```
100 REM #####
110 REM ##                                     ##
120 REM ## MOB SET-BEISPIEL ##
130 REM ##                                     ##
140 REM #####
150 REM
160 COLOUR 0.0 : REM RAHMEN UND HINTERGRUND = SCHWARZ
170 DESIGN 0.13*64 : SPRITEDEFINITION IN BLOCK 13
190 @ .....BB.....
200 @ .....BBBB.....
210 @ .....BBBB.....
220 @ .....BB.....
230 @ .....BB...BB...BB.....
240 @ .....BB...BB...BB.....
250 @ .....BB..BB..BB.....
260 @ .....BB.BB.BB.....
270 @ ..BB.....BBBBBB.....BB.
280 @ .BBBBBBBBBBBBBBBBBBBBBBBB
290 @ .BBBBBBBBBBBBBBBBBBBBBBBB
300 @ ..BB.....BBBBBB.....BB.
310 @ .....BB.BB.BB.....
320 @ .....BB..BB..BB.....
330 @ .....BB...BB...BB.....
340 @ .....BB...BB...BB.....
350 @ .....BB.....
360 @ .....BBBB.....
370 @ .....BBBB.....
380 @ .....BB.....
390 @ .....
400 REM
410 MMOB 0.100.100.100.100.3.0 : REM SPRITE AUF DEN
BILDSCHIRM BRINGEN
420 FOR X=0 TO 15
```

```
430 MOB SET 0.13.X.0.0 : REM EIGENSCHAFTEN BESTIMMEN UND  
FARBE WECHSELN  
440 PAUSE 2  
450 NEXT X  
460 COLOUR 8,8  
470 REM  
480 FOR X=1 TO 6  
490 MOB SET 0.13.7.1.0 : REM HINTER DEN BILDSCHIRMZEICHEN  
500 PAUSE 1  
510 MOB SET 0.13.7.0.0 : REM VOR DEN BILDSCHIRMZEICHEN  
520 PAUSE 1  
530 NEXT X
```

Auch hier wurde mit dem Befehl MMOB (s.u.) vorgegriffen, um das Sprite überhaupt sichtbar zu machen. Stören Sie sich nicht weiter daran.

#### 14.1.4 CMOB

```
FORMAT      :CMOB F1.F3
PARAMETER   : - F1: FARBNUMMER 1 FÜR MULTI-
              COLOR-SPRITES
              (BUCHSTABE B) (0-15)
              - F3: FARBNUMMER 3 FÜR MULTI-
              COLOR-SPRITES
              (BUCHSTABE D) (0-15)
BEISPIEL    :CMOB 2.7
FUNKTION    :ZUSÄTZLICHE FARBEN FÜR MULTI-
              COLOR-SPRITES WÄHLEN
```

#### ERLÄUTERUNGEN:

Der im folgenden beschriebene Befehl stellt eine Ergänzung des MOB SET - Kommandos dar, speziell für Multicolor - Sprites. Wie Sie bereits gelernt haben, können in einem Multicolor - Sprite drei Punktfarben bei der Erstellung einer Figur beteiligt sein (im Gegensatz zu einer in HGR - Sprites). Diese zwei zusätzlichen Farben können Sie nun mithilfe des CMOB-Befehls festlegen. Dabei steht f1 für die Farbe, die Sie bei der Definition der Minigraphik in den Musterzeilen mit einem B symbolisieren. Alle Punkte, die mit B bezeichnet wurden, erhalten nun die Farbe f1. Diejenigen jedoch, die Sie mit einem D gekennzeichnet haben, werden nun in der Farbnummer 3 (f3) auf dem Bildschirm dargestellt. Die Farbangabe des Befehls SET MOB legt bekanntlich die Farbe 2 der Sprites, also alle mit C gezeichneten Punkte der Spritedefinition, fest.

Wichtig in diesem Zusammenhang ist ferner, daß die beiden mit CMOB gewählten Farben für alle (maximal 8) Sprites gleichzeitig gelten, d.h. alle Sprites sind in diesen Farben identisch. Lediglich die mit C bezeichneten Punkte können, wie Sie sicher bereits wissen, für jedes Sprite unterschiedlich gewählt werden. Zur Anwendung dieses Befehls stehen Ihnen unter den verschiedenen Abschnitten diverse Beispiele zur Verfügung.

#### 14.1.5 TESTAUFGABEN

Zum Abschluß ein paar Testaufgaben, die Sie hoffentlich mit Links erledigen können (muß aber nicht unbedingt sein):

1.) Welche Syntax ist richtig?

- a) DESIGN 0
- b) MOB SET 1,15,3,0,0
- c) DESIGN 0,832
- d) CMOB 2,3

2.) Was ist bei der Verwendung von DESIGN zu beachten?

- a) DESIGN ist nur für die Spritedarstellung zu gebrauchen
- b) Der Block 13 bezeichnet im Graphikmodus einen anderen Speicherbereich als im Textmodus
- c) Der erste Parameter gibt an, ob es sich um ein kleines oder vergrößertes Sprite handelt
- d) Zwischen DESIGN und der ersten Musterzeile darf keine andere Basiczeile stehen
- e) DESIGN schließt Multicolor - Sprites aus

3.) Was bewirken die ersten zwei Parameter des MOB SET - Befehls?

- a) Sie geben an, ob es sich um ein Multicolor - oder HGR-Sprite handelt und bestimmen die Priorität
- b) Die Definition aus dem genannten Block wird dem Sprite mit der angegebenen Nummer zugeordnet
- c) Der erste Parameter stellt das Sprite auf dem Bildschirm dar
- d) Der zweite Parameter gibt an, welches Aussehen das Sprite haben soll

## 14.2 SPRITESTEUERUNG

### 14.2.1 MMOB

```
FORMAT      :MMOB N.X1.Y1.X2.Y2.GR.GE
PARAMETER   : - N : NUMMER DES ZU BEWEGENDEN
                SPRITES (0-7)
                - X1: X-KOORDINATE DES START-
                PUNKTES (0-511)
                - Y1: Y-KOORDINATE DES START-
                PUNKTES (0-255)
                - X2: X-KOORDINATE DES ZIEL-
                PUNKTES (0-511)
                - Y2: Y-KOORDINATE DES ZIEL-
                PUNKTES (0-255)
                - GR: GRÖÖE DES SPRITES (0-3)
                - GE: GESCHWINDIGKEIT, MIT DER
                SICH DAS SPRITE ZWISCHEN
                DEN ZWEI PUNKTEN BEWEGEN
                SOLL (0-255)

BEISPIEL    :MMOB 1.100.50.200.160.0.10
FUNKTION     :DARSTELLEN UND BEWEGEN EINES
                SPRITES AUF DEM BILDSCHIRM
```

#### ERLÄUTERUNGEN:

Endlich ist es soweit! Lang ersehnt, viel versprochen, oft verwiesen - kurz: MMOB.

Nun endlich ist es uns (offiziell) möglich, ein Sprite auf dem Bildschirm erscheinen zu lassen. Welche Freude! Wir können es nicht nur einfach hinstellen, nein, sogar quer über den Bildschirm fahren, trudeln oder rasen lassen.

MMOB ermöglicht uns einiges: Zunächst einmal schaltet er ein Sprite ein, sodaß es auf dem Bildschirm zu sehen ist. Gleichzeitig werden aber noch einige weitere Funktionen ermöglicht.

Zum einen können Sie die Position angeben, an der Ihre Figur zum ersten Male auf dem Bildschirm erscheint. Diese Position geben Sie mit x1 und y1 als Koordinaten an. Zum anderen können Sie zwei weitere Koordinaten x2 und y2 benennen, die bestimmen, zu welchem Punkt sich das Sprite direkt nach

seinem Auftauchen begeben soll. Zunächst einiges zu den Koordinaten: Grundsätzlich stimmen die Koordinaten, die Sie als Spritekoordinaten angeben nicht mit den Ihnen geläufigen Bildschirm - Graphikkoordinaten über ein. Angegeben wird stets die Koordinate, bei der sich die untere linke Ecke des Sprites befindet. Befindet sich dieser Punkt im uns bekannten Nullpunkt der Graphik (obere linke Ecke des Bildschirmfensters), so lauten die Spritekoordinaten dort bereits  $x=20$  und  $y=30$ . Entsprechend gilt diese Nullpunkt - Verschiebung für den gesamten Bildschirm. Weiterhin können Sie weit über das Fenster hinaus Werte angeben ( $x: 0-511$  ;  $y: 0-255$ ). Alle diese Dinge wurden unternommen, damit Sie ein Sprite auch aus dem Bildschirm hinaus bewegen können, müssen aber an entsprechender Stelle beachtet werden (s. Kollisionen).

Wollen Sie das Sprite lediglich an eine bestimmte Stelle auf dem Bildschirm setzen, ohne es zu bewegen, so wählen Sie einfach die beiden Koordinatenpaare  $x_1,y_1$  und  $x_2,y_2$  gleich. Nun ist der Start- gleich dem Zielpunkt, und Ihr Sprite bleibt direkt stehen.

Wie gesagt, wird sich nun Ihr Sprite vom Startpunkt ( $x_1,y_1$ ) bis hin zum Zielpunkt ( $x_2,y_2$ ) quer über den Text oder die Graphik bewegen. Dies passiert mit der Geschwindigkeit, die Sie als letzten Parameter  $ge$  angeben.  $ge$  kann Werte von 0 bis 255 annehmen. Dabei wird das Sprite umso schneller, je kleiner  $ge$  gesetzt wird. Mit  $ge=255$  also tuckert Ihre Schnecke gemächlich durch das Gebüsch, während der mit  $ge=0$  fliegende Starfighter über sie hinwegdonnert.

Aber halt, wir haben ja noch etwas vergessen. Wenn man es auch nicht hier bei diesem Befehl erwartet hätte: Mit dem Parameter  $gr$  können Sie gleichzeitig bei der Ausführung dieses Befehls die Größe der Figur bestimmen. Wie Sie wahrscheinlich schon wissen, können Sie jedes Sprite in zwei Richtungen vergrößern. Die Möglichkeiten, die Ihnen  $gr$  verschafft seien kurz in einer kleinen Tabelle dargelegt:

GR	VERGRÖßERUNG	VERGRÖßERUNGSFAKTOR	PUNKTMATRIX
0	KEINE	1X1	24X21
1	X-RICHTUNG	2X1	48X21
2	Y-RICHTUNG	1X2	24X42
3	X/Y-RICHTUNG	2X2	48X42

Unter Punktmatrix ist hierbei natürlich die Matrix gemeint, die auf dem Bildschirm erscheint (die ist ja auch bei Multicolor - Sprites identisch), also die Anzahl der Punkte des Bildschirms, die von einem Sprite maximal überdeckt werden.

Um das Ganze etwas klarer zu machen, sei hier wieder ein kleines Beispiel gegeben:

**BEISPIEL :**

```

100 REM #####
110 REM ##          ##
120 REM ##  MMOB-BEISPIEL  ##
130 REM ##          ##
140 REM #####
150 REM
160 COLOUR 0.0 : REM RAHMEN UND HINTERGRUND = SCHWARZ
170 DESIGN 0.13*64 : SPRITEDEFINITION IN BLOCK 13
180 @ .....
190 @ .....
200 @ .....
210 @ .....
220 @ .....
230 @ .....
240 @ .....
250 @ ...BB.....
260 @ ...BBB.....
270 @ ...BBBB.....
280 @ ...BBBBB.....BBBB.....
290 @ ...BBBBBB.....BBB.BB....
300 @ ...BBBBBBBBBBBBBBBBBB...
310 @ .BBBBB.BB.BB.BB.BB.BBBBB
320 @ ...BBBBBBBBBBBBBBBBBB...
330 @ .....
340 @ .....
350 @ .....

```

```

360 @ .....
370 @ .....
380 @ .....
390 REM
400 MOB SET 0.13.7.0.0 : REM SPRITE 0 = GELB
410 MOB SET 1.13.2.0.0 : REM SPRITE 0 = ROT
420 REM
430 REM #####
440 REM ##           ##
450 REM ## BEWEGUNG ##
460 REM ##           ##
470 REM #####
480 REM
490 MMOB 0.0.0.200.200.1.50 : REM IN X-RICHTUNG GEDEHNT
500 MMOB 1.0.100.220.220.0.70 : REM NORMAL
510 PAUSE 5
520 COLOUR 9.9 : REM HINTERGRUND UND RAHMEN = BRAUN

```

Wie Sie sehen, werden hier zwei Sprites gleichen Aussehens aber unterschiedlicher Farbe und Größe verschieden schnell über den Bildschirm bewegt. Achten Sie auf die Zeilen 400 und 410, in denen Sprite 0 und 1 dem gleichen Block zugeordnet werden.



## 14.2.2 MOB OFF

```
FORMAT      :MOB OFF N
PARAMETER   : - N : NUMMER DES AUSZUSCHAL-
              TENDEN SPRITES (0-7)
BEISPIEL    :MOB OFF 0
FUNKTION    :AUSSCHALTEN EINES SPRITES
```

### ERLÄUTERUNGEN:

Nachdem Sie nun also gelernt haben, wie Sie ein Sprite auf den Bildschirm bringen können (MMOB), müssen wir uns natürlich auch mit dem Befehl beschäftigen, der es uns ermöglicht dieses Objekt wieder zu löschen. Diese Aufgabe zu erfüllen, hat sich das Kommando MOB OFF bereiterklärt. Sie geben als einzigen Parameter einfach die Nummer des auszuschaltenden Sprites an und schon sehen Sie nichts mehr auf Ihrem Schirm.

MOB OFF löscht jedoch sämtliche Spritezuordnungen zu bestimmten Definitionsblöcken (Der Inhalt der Blöcke bleibt natürlich erhalten und kann auch weiter verwendet werden). D.h. Sie müssen, wenn Sie dieses Sprite wieder einschalten, also wieder auf dem Bildschirm darstellen wollen, der entsprechenden Spritenummer wieder mittels MOB SET einen bestimmten Block zuordnen. Die mit CMOB gewählten zwei zusätzlichen Multicolor - Farben bleiben dagegen erhalten. Im Anschluß hieran wird das Objekt natürlich wieder mit MMOB auf den Bildschirm gezaubert. Mit diesem Befehl ist es Ihnen möglich, z.B. getroffene Raumschiffe zu eliminieren oder eine bewegliche Figur darzustellen, indem Sie stets (wie im Beispiel anschaulich demonstriert) zwischen zwei oder mehreren Spritedefinitionen hin und her schalten.

Ein Tip: Sie können MOB OFF selbstverständlich auch dadurch ersetzen, daß Sie das betreffende Sprite aus dem Bildschirm heraus positionieren (mit MMOB), so daß es ebenfalls nicht mehr sichtbar ist. Der Vorteil: Die Blockzuordnung bleibt erhalten, d.h. Sie können Ihr Sprite irgendwann einmal wieder auftauchen lassen, ohne umständlich den Befehl MOB SET eintippen zu müssen.

Wollen Sie dagegen nur die Definition wechseln, also das Sprite verändern, so brauchen Sie vor dem MOB SET - Befehl nicht unbedingt ein MOB OFF einzugeben.

**BEISPIEL :**

```
100 REM #####  
110 REM ##                               ##  
120 REM ## MOB OFF-BEISPIEL ##  
130 REM ##                               ##  
140 REM #####  
150 REM  
160 COLOUR 0,0 : REM RAHMEN UND HINTERGRUND = SCHWARZ  
170 DESIGN 0,13*64 : REM SPRITEDEFINITION IN BLOCK 13  
180 @ .....  
190 @ .....  
200 @ ..... B . B .....  
210 @ ..... B . B .....  
220 @ ..... B . B .....  
230 @ ..... B . B .....  
240 @ ..... BBBB .....  
250 @ ..... BBBBBB .....  
260 @ ..... BB . . BB .....  
270 @ . . B . . BB . . BB . BB . . B . .  
280 @ . . B . BB . B . BB . BB . B . .  
290 @ . . BBBBBBBBBBBBBBBBBB . .  
300 @ . . BBBBBB . B . B . B . B . BBBBB . .  
310 @ . BBBBBB . B . B . B . B . BBBBBB .  
320 @ BBBBBBBBBBBBBBBBBBBBBBBBBB  
330 @ BB . . . . BBBBBBBBBB . . . . BB  
340 @ BB . . . . BB . . . . BB . . . . BB  
350 @ . . . . BB . . . . BB . . . .  
360 @ . . . . BBBB . . . . BBBB . . . .  
370 @ . . . . BBBB . . . . BBBB . . . .  
380 @ .....  
390 REM  
400 REM  
410 DESIGN 0,14*64 : REM SPRITEDEFINITION IN BLOCK 14  
420 @ .....  
430 @ .....  
440 @ ..... BB . . . . BB .....  
450 @ ..... BB . . . . BB .....  
460 @ ..... BB . . . . BB .....  
470 @ ..... BB . BB .....  
480 @ ..... BBBB .....  

```

```

490 @ .....BBBBBB.....
500 @ .....BB...BB.....
510 @ .....BB...BB.BB.....
520 @ .....BB..B.BB..BB.....
530 @ ...BBBBBBBBBBBBBBBBBB...
540 @ ..BBBB.B.B.B.B.B.BBBBB..
550 @ .BBBBBB.B.B.B.B.BBBBBBB.
560 @ BBBBBBBBBBBBBBBBBBBBBBBB
570 @ .....BBBBBBBBBB.....
580 @ .....BB..BB..BB.....
590 @ .....BB.....BB.....
600 @ ...BBBB.....BBBB....
610 @ ...BBBB.....BBBB....
620 @ .....
630 REM
640 REM
650 REM #####
660 REM ##          ##
670 REM ##  BLINKEN  ##
680 REM ##          ##
690 REM #####
700 REM
710 FOR X=1 TO 100
720 REM -----
730 MOB SET 0.13.6.0.0
740 MMOB 0.200.100.200.100.3.0 : REM ERSTES  SPRITE
EINSCHALTEN
750 FOR Y=1 TO 60 : NEXT Y
760 MOB OFF 0 : REM ZWEITES SPRITE AUSSCHALTEN
770 REM -----
780 MOB SET 0.14.6.0.0
790 MMOB 0.200.100.200.100.3.0 : REM ZWEITES  SPRITE
EINSCHALTEN
800 FOR Y=1 TO 60 : NEXT Y
810 MOB OFF 0 : REM ZWEITES SPRITE AUSSCHALTEN
820 REM -----
830 NEXT X
840 COLOUR 9.9

```

In diesem Beispiel wird durch den steten Wechsel zwischen zwei ähnlichen Spritedefinitionen ein Blinken bzw. Bewegung

simuliert. Die Zeilen 760 und 810 schalten das alte Sprite jeweils aus und können - wie oben erläutert - eigentlich auch weggelassen werden. Auf diese Weise ist es möglich, in sich bewegliche Figuren herzustellen. Wenn sich diese Figur auch noch aus mehreren Sprites zusammensetzt, so erhalten Sie wahrhaftig tolle Effekte. Probieren Sie doch etwas herum oder sehen sich einmal den nächsten Befehl (und das dazugehörige Beispiel) an. Sie werden sich wundern! Übrigens, wie gefallen Ihnen die Sprites? Wollen Sie sich nicht auch einmal an die Konstruktion eines solchen Objektes begeben? Sie werden sehen, es ist gar nicht so schwierig!

### 14.2.3 RLOCMOB

**FORMAT** : RLOCMOB N.X2.Y2.GR.GE  
**PARAMETER** : - N : NUMMER DES ZU BEWEGENDEN  
SPRITES (0-7)  
- X2: X-KOORDINATE DES ZIEL-  
PUNKTES (0-511)  
- Y2: Y-KOORDINATE DES ZIEL-  
PUNKTES (0-255)  
- GR: GRÖÙE DES SPRITES (0-3)  
- GE: GESCHWINDIGKEIT, MIT DER  
SICH DAS SPRITE ZU DEM  
PUNKT BEWEGEN SOLL (0-255)  
**BEISPIEL** : RLOCMOB 1.200.160.0.10  
**FUNKTION** : DARSTELLEN ODER WEITERBEWEGEN  
EINES SPRITES AUF DEM BILDSCHIRM

#### ERLÄUTERUNGEN:

Es folgt ein weiterer sehr nützlicher Befehl, der die Fähigkeit besitzt, ein Sprite von seiner ursprünglichen (z.B. mit MMOB eingestellten) Position aus zu einem zweiten Zielpunkt zu befördern. Somit brauchen Sie nicht - wie im MMOB-Befehl - ständig den Startpunkt mit anzugeben und sich zu merken. Ansonsten funktioniert dieses Kommando hundertprozentig analog zu MMOB (Sie sollten also die Ausführungen zu diesem in Abschnitt 14.2.1 dargestellten Befehl gut gelesen und verstanden haben.). RLOCMOB schaltet also gleichfalls ein angesprochenes Sprite ein. Mit n bestimmen Sie die Spritenummer, gr definiert die Größe und ge die Geschwindigkeit, mit der es sich fortbewegen soll.

Verwenden Sie jedoch RLOCMOB möglichst nicht, um ein Sprite einzuschalten, da es sonst quer über den Bildschirm zu der angegebenen Position fliegt, da die Ursprungskoordinate natürlich noch unbestimmt ist, was höchstwahrscheinlich nicht in Ihrem Interesse liegt.

Mit diesem Befehl ist es Ihnen nun sehr einfach möglich, kontrollierte Bewegungen auch mehrerer Sprite gleichzeitig zu unternehmen. Das Simon's Basic Handbuch bietet hier ein recht schönes Beispiel auf den Seiten 51-53, das Sie sich vielleicht auch einmal anschauen sollten.

Da zu diesem Befehl alles gesagt ist, wollen wir uns auch

nicht weiter bei der Vorrede aufhalten und direkt in medias res gehen. Unser

**BEISPIEL :**

```
100 REM #####
110 REM ##                                     ##
120 REM ##  RLOCMOB-BEISPIEL  ##
130 REM ##                                     ##
140 REM #####
150 REM
160 COLOUR 0,0 : REM RAHMEN UND HINTERGRUND = SCHWARZ
170 DESIGN 0,13*64 : REM SPRITEDEFINITION IN BLOCK 13
180 @ .....BBB.....
190 @ ..BBB.....BB.BB.....
200 @ ...BB..B...BBBBBBBB.....
210 @ .BBBBBB...BBBBBB..BB..
220 @ ...BB..B...BB..BBBBBB..
230 @ ..BBB.....BBB.....
240 @ .....BBB.....
250 @ .....BBB.....
260 @ .....BBBBB.....
270 @ .....BBB.BB.....
280 @ .....BBBB..BB.....
290 @ .....BBBBBB..BB.....
300 @ .....B.BBB.....
310 @ .....B..BBB.....
320 @ .....B..BB.BBB.....
330 @ .....B..BB..BB.....
340 @ .....B...BB..BB.....
350 @ .....BB..BB..BB.....
360 @ .BBBBBBB..BB...BBBB.....
370 @ B.BBBB...BB.....
380 @ ..B..B...BBBB.....
390 REM
400 REM
410 DESIGN 0,14*64 : REM SPRITEDEFINITION IN BLOCK 14
420 @ .....BBB...B.B.
430 @ .....BB.BB...B..
440 @ ..BBB.B...BBBBBBBB.....
450 @ .BBBBB...BBBBBB..BB..
```

```

460 @ ..BBB.B.....BBBBBBBBBB..
470 @ .....BBB.....
480 @ .....BBB.....
490 @ .....BBB.....
500 @ .....BBB.....
510 @ .....BBBB.....
520 @ .....BBBBB.....
530 @ .....BBB.BB.....
540 @ .....BBBB.....
550 @ .....B.BBBB.....
560 @ .....B..BB.BB.....
570 @ .....B..BB.BB.....
580 @ .....B..BBB..BB.....
590 @ .....BB.BB...BB.....
600 @ .BBBBBBB.BBB...BB.....
610 @ B.BBBB.....BB..BB.....
620 @ ..B..B.....BBBB.....
630 PRINT CHR$( 30) CHR$(147) : REM GRUEN + BILOSCHIRM
LOESCHEN
640 PRINT AT(0,9) DUP(CHR$(1B) + " ",40) : REM RASEN ZEICHNEN
650 REM
660 REM
670 REM #HHHHHHHHHHHHHH#
680 REM #H#           #H#
690 REM #H# LAUFEN #H#
700 REM #H#           #H#
710 REM #HHHHHHHHHHHHHH#
720 REM
730 MMOV 0,0,100,0,100,1,0 : REM SPRITE POSITIONIEREN
740 G = 10 : REM GESCHWINDIGKEIT
750 F = 1 : REM FARBSTART
760 FOR X=1 TO 300 STEP G
770 F = F+.3: REM FARBE WECHSELN
780 IF F=16 THEN F=1
790 REM -----
800 MOV SET 0,13,F,0,0
810 RLOCMB 0,X,101,1,0 : REM ERSTES SPRITE BEWEGEN
820 FOR Y=1 TO 20 : NEXT Y
830 MOV OFF 0 : REM ERSTES SPRITE AUSSCHALTEN
840 REM -----
850 MOV SET 0,14,F,0,0

```

```
860 RLOCMOB 0.X+G/2.101.1.0 : REM ZWEITES SPRITE BEWEGEN
870 FOR Y=1 TO 20 : NEXT Y
880 MOB OFF 0 : REM ZWEITES SPRITE AUSSCHALTEN
890 REM -----
900 NEXT X
910 COLOUR 9.9
```

Dieses Beispiel demonstriert Ihnen die Arbeit mit mehreren Spritedefinitionen simultan. Wie auch schon im vorigen Kapitel gesagt, können hier natürlich die Zeilen 830 und 880 wieder weggelassen werden. Sie dienen nur zum besseren Verständnis. In der Zeile 740 können Sie die Geschwindigkeit des Männchens verändern. Probieren Sie alles mögliche aus. Lassen Sie doch einmal einen Bienenschwarm nebenher laufen oder setzen Sie Bäume an den Wegesrand. Vielleicht steuern Sie den Mann einmal per Tastatur oder Joystick, wer weiß?



## 14.2.4 DETECT

```
FORMAT      : DETECT K
PARAMETER   : - K: ART DER KOLLISIONS-
              VORBEREITUNG:
              K=0: SPRITE-SPRITE
              K=1: SPRITE-HINTER-
                  GRUNDZEICHEN
BEISPIEL    : DETECT 1
FUNKTION    : VORBEREITEN DER
              KOLLISIONSABFRAGE
```

### ERLÄUTERUNGEN:

Mit diesem Befehl - kaum zu glauben - werden Welten erschlossen. Jetzt geht es richtig zur Sache. Da macht das Programmieren Spaß. Jung und Alt finden sich zusammen - Staunen. Sie werden Augen machen!

Es fängt zunächst ganz harmlos an: Dieser Befehl bereitet die Abfrage auf eine Kollision vor. Nun, das klingt zunächst einmal etwas kompliziert und ist es auch. Wie Sie aus der Einleitung wissen, gibt es im Registersatz des VIC zwei Speicherstellen, die sogenannte Kollisionen registrieren. Die eine von beiden ist für Berührungen von Sprites untereinander, die andere für solche gedacht, die zwischen einem Sprite und Hintergrundzeichen (im Textmodus z.B. Buchstaben) geschehen. Diese Register werden entsprechend gesetzt, wenn eine Kollision stattgefunden hat. Der Inhalt dieser Register jedoch bleibt solange erhalten, bis es wieder von Ihnen gelöscht wird, d.h. wenn Sie dieses Register nach einer Abfrage, ob eine Kollision stattgefunden hat nicht löschen, scheint es so, als wäre dieses alte Ereignis noch aktuell, die Kollision fände also noch statt.

Um dieses zu vermeiden ist der DETECT-Befehl notwendig. Er löscht das entsprechende Kollisionsregister und bereitet somit eine erneute Abfrage vor. Dieser Befehl sollte somit nach jeder Abfrage auf Kollision stehen (oder eventuell schon vorher), damit gleich die nächste Abfrage vorbereitet ist. Hat eine Kollision stattgefunden, so genügt es nicht, diesen Befehl einmal zu geben, Sie müssen im Laufe des Programms (auch wenn zwischendurch das Programm zuende war - s. Beispiel) bis zur nächsten Abfrage dieses Kommando vielmehr

mindestens zweimal gesendet haben, um nicht fälschlicherweise eine zweite Kollision zu registrieren. Dies beruht auf bestimmten Hardware - Eigenschaften und würde hier den Rahmen des Buches sprengen.

Sie sehen, die Anwendung ist nicht ganz so einfach, wie man es sich wünschen könnte, mit etwas Übung aber läßt sich auch dieses bewerkstelligen, und es wird überhaupt kein Problem mehr für Sie sein.

Der einzige Parameter in diesem Befehl ist dafür zuständig, festzulegen, ob eine Kollisionsabfrage für eine Sprite - Sprite - oder eine Sprite - Hintergrundzeichen - Kollision vorbereitet werden soll (s. Einleitung).

Mehr hierzu lesen Sie unter dem nächsten Befehl.

## 14.2.5 CHECK

**FORMAT** :CHECK(N1.N2)  
**ODER** CHECK(N)  
**PARAMETER** : - N1: NUMMER DES ERSTEN SPRITE,  
DAS AUF EINE KOLLISION  
MIT EINEM ANDEREN SPRITE  
GEPÜFT WERDEN SOLL (0-7)  
- N2: NUMMER DES ZWEITEN SPRITE,  
DAS AUF EINE KOLLISION  
MIT EINEM ANDEREN SPRITE  
GEPÜFT WERDEN SOLL (0-7)  
- N : NUMMER DES SPRITE, DAS  
AUF EINE KOLLISION MIT  
EINEM HINTERGRUNDZEICHEN  
GEPÜFT WERDEN SOLL (0-7)  
**BEISPIEL** :CHECK(0.7)  
**ODER** CHECK(1)  
**FUNKTION** :ABFRAGEN EINER KOLLISION

### ERLÄUTERUNGEN:

Endlich sind wir beim allerletzten Spritebefehl angekommen, den uns Simon's Basic bietet: CHECK

Der Befehl CHECK besitzt wieder Funktionscharakter, wie schon die Befehle SIN, SQR, LIN usw. Er wird ähnlich verwendet, wie der schon bekannte Befehl TEST, der nachprüft, ob an einer Stelle in einer Graphik ein Punkt gesetzt ist.

CHECK dient dazu festzustellen, ob zur Zeit eine Kollision eines Sprites mit einem anderen Objekt (Sprite oder Hintergrundzeichen) erfolgt ist. Er wird eingeleitet mit dem Befehl DETECT, den Sie sich vorher durchgelesen haben sollten. Ist zum Zeitpunkt der Ausführung keine Kollision registriert, so erhält CHECK den Wert 1. Diesen Wert können Sie natürlich beliebig einer Variablen zuweisen oder mit ihm Rechnungen durchführen. Z.B.:

$$A = 2 * CHECK(0) + 1$$

Wurde dagegen eine Kollision vermerkt, so wird der Funktion CHECK der Wert 0 zugewiesen.

In dem vorgehenden DETECT hatten Sie bereits angegeben, ob es sich um eine Sprite - Sprite - oder eine Sprite - Hintergrundzeichen - Kollisionsabfrage handelt. Diese Infor-

mation wird bei der Verwendung des CHECK-Befehls wieder aufgegriffen. Wurde somit der erste Fall vorbereitet, so muß die Syntax des nachfolgenden CHECK lauten:

### **CHECK(N1.N2)**

Mit diesen Parametern geben Sie an, zwischen welchen beiden Sprites Sie eine Kollision nachprüfen wollen. Andere Sprites, die eventuell ebenfalls kollidiert sind, werden dabei nicht berücksichtigt. Nur wenn sich diese beiden Objekte, deren Nummern Sie mit n1 und n2 determiniert haben, berührt haben, wird CHECK 0. Wollen Sie lediglich prüfen, ob ein Sprite mit einem beliebigen anderen Sprite kollidiert ist, so wählen Sie n1=n2. In diesem Falle wird jede Kollision dieses Sprites mit einem anderen Sprite registriert.

Analoges gilt, wenn Sie den zweiten Fall mit DETECT 1 vorbereitet haben. Hier lautet die Syntax dann:

### **CHECK(N)**

wobei n die Nummer des Sprites darstellt, dessen Kollision mit dem Hintergrund überprüft werden soll (die Information, die das Handbuch hierüber gibt ist natürlich falsch).

Wichtig in diesem Zusammenhang ist die Tatsache, daß (wie beim Befehl AT) die erste Klammer hinter dem Wort CHECK eigentlich noch zum Befehl gehört und nicht durch ein Leerzeichen abgetrennt werden darf. Also:

### **CHECK(2,2)**

und nicht

### **CHECK (2,2)**

Soweit hierzu. Ich hoffe Sie werden viel Spaß mit diesem Befehl haben (zumal Sie jetzt noch einige Möglichkeiten der Anwendung mehr, als der im Handbuch aufgeführten, haben). Zu Ihrem Amüsement ein kleines

### **BEISPIEL :**

100 REM #####

110 REM ## ##

120 REM ## CHECK-BEISPIEL ##

130 REM ## ##

140 REM #####

150 REM

160 COLOUR 0,0 : REM RAHMEN UND HINTERGRUND = SCHWARZ

170 DESIGN 1,13\*64 : REM SPRITEDEFINITION IN BLOCK 13

```

180 @ .....
190 @ .....
200 @ .....
210 @ .....
220 @ .....
230 @ .C.....
240 @ CCC.....
250 @ CCCBB.....
260 @ CCBDBDDDB..
270 @ CCCBDBDBDBB
280 @ CCBD.....
290 @ CCC.....
300 @ .C.....
310 @ .....
320 @ .....
330 @ .....
340 @ .....
350 @ .....
360 @ .....
370 @ .....
380 @ .....
390 REM
400 REM
410 DESIGN 0,14*64 : REM SPRITEDEFINITION IN BLOCK 14
420 @ .....
430 @ .....
440 @ .....
450 @ .....
460 @ .....
470 @ .....BBB.....
480 @ .....B.....
490 @ .....B.....
500 @ .....B.....
510 @ .....BBBBB.....
520 @ .....BB..BBBBBB.....
530 @ .....BBB..BBB..BB.....
540 @ .....BBBBB..BBB..BBB.....
550 @ ..BBBBBBBBBBBB..BBBB.....
560 @ ..BBBBBBBBBBBBBBBBBB.....
570 @ ..BBBBBBBBBBBBBBBBBB.....
580 @ ...BB..BBBBBBB..BB.....

```

```

590 @ BBBB.BB.BBBBBB.BB.BB....
600 @ .....BB.....BB.....
610 @ .....
620 @ .....
630 REM
640 MMOB 0,0,100,0,100,0,0 : REM ERSTES SPRITE POSITIONIEREN
650 MMOB 1,300,100,300,100,1,0 : REM ZWEITES SPRITE
POSITIONIEREN
660 DETECT 0 : REM KOLLISIONSREGISTER LOESCHEN
670 REM
680 REM #####
690 REM ##          ##
700 REM ## FAHREN ##
710 REM ##          ##
720 REM #####
730 REM
740 FOR X=1 TO 300 STEP 4
750 CMOB 2,7 : REM FARBEN FUER MULTICOLOR-SPRITE
760 REM -----
770 MOB SET 0,14,9,0,0
780 RLOCMOB 0,X,100,0,0 : REM ERSTES SPRITE BEWEGEN
790 REM -----
800 CMOB 7,2 : REM FARBEN FUER MULTICOLOR-SPRITE
810 MOB SET 1,13,11,0,1
820 RLOCMOB 1,320-X,40*SIN(X/30+1,3)+100,1,0 : REM ZWEITES
SPRITE BEWEGEN
830 REM -----
840 DETECT 0 : REM ABFRAGE AUF SPRITE-SPRITE-KOLLISION
VORBEREITEN
850 IF CHECK(0,1)=0 THEN GOTO 870
860 NEXT X
870 FOR Z=1 TO 15
880 COLOUR Z,Z-1
890 FOR S=1 TO 30 : NEXT S : REM WARTESCHLEIFE
900 NEXT Z
910 COLOUR 13,14

```

### 14.2.6 TESTAUFGABEN

Zum Abschluß dieses sicher hochinteressanten Kapitels über die Bedienung der Sprites im Simon's Basic sollten Sie hier und jetzt einmal überprüfen, was Sie behalten und verstanden haben.

1.) Was ist beim Gebrauch von DETECT und CHECK zu beachten?

- a) DETECT muß nach einer erfolgten Kollision einmal gegeben werden, um das Kollisionsregister zu löschen
- b) Durch den Parameter des DETECT-Befehls wird die Syntax von CHECK bestimmt
- c) CHECK(1,1) ist unzulässig
- d) CHECK(9) ist unzulässig
- e) Ist eine Kollision erfolgt, so erhält CHECK den Wert 1

2.) Welche Syntax ist richtig?

- a) MMOB 1,30,40,50,60,2,70
- b) MOB OFF
- c) RLOCMOB 3,70,3
- d) CHECK (2,3)

## 15. KAPITEL

### MUSIK

Die Zeit ist gekommen. Halten Sie sich bereit. In wenigen Augenblicken tauchen wir unter in das geheimnisvolle Reich der Klänge. Lassen Sie sich berauschen von der unendlichen Harmonie der synthetischen Klangerzeugung. Kommen Sie ruhig näher. Kommen Sie! Schauen Sie, nein lauschen Sie Ihrem guten, bisher stummen Freund, dem COMMODORE 64. Sie werden sehen, er wird ein begabter, hoffnungsvoller Nachwuchsmusiker unter Ihren Händen. Nehmen Sie sich seiner an und er wird Ihr bester Kamerad.

Vor Ihnen, ja, Sie werden es kaum glauben, vor Ihnen liegt ein kompletter, gut ausgerüsteter und dreistimmiger Musiksynthesizer. Glauben Sie nicht, die Ihnen bekannte Radiomusik stamme aus anderen Quellen. Mit ein wenig Geschick werden Sie die schönsten Melodien und knarrendsten Geräusche aus Ihrem Computer herauslocken. Selbstverständlich ist Simon's Basic kein Ersatz für ein richtiges Synthesizerprogramm, das alle, aber auch alle Möglichkeiten ausschöpft (ich empfehle hier immer das in der Tat gute SYNTHIMAT, das Ihnen auf einfachste Weise ermöglicht, alles aus Ihrem Rechner herauszuholen). Es bietet Ihnen trotzdem einige sehr schöne Einzelheiten. Lassen Sie sich überraschen!



## 15.1 HARDWAREVORAUSSETZUNGEN

Um die einzelnen Befehle zur Klangerzeugung zu verstehen, müssen wir uns zunächst einmal klar machen, auf welche Weise überhaupt ein einzelner Ton aufgebaut ist. Natürlich gibt es sehr komplexe Geräusche und Tonqualitäten. Um jedoch einen eigenen Ton zu erzeugen, müssen wir wissen, aus welchen Komponenten sich ein solcher Ton zusammensetzt.

Da haben wir zunächst einmal die Tonhöhe. Schon hier tauchen Probleme auf: Was bewirkt eigentlich, daß ein Ton eine bestimmte "Tonhöhe" besitzt. Nun, wie Sie aus der Physik vielleicht noch wissen, ist Schall einfach eine sehr schnelle Luftbewegung, die wir mit unseren Ohren wahrnehmen. Die einfachste Welle aber, die wir uns vorstellen können ist eine sinusförmige, völlig gleichmäßige Welle.



Es ist möglich, eine solche Schallwelle zu erzeugen, durch die die Luft tatsächlich sinusförmig schwingt. Viele haben ihn sicher schon einmal aus irgendwelchen Geräten gehört. Die unterschiedliche Tonhöhe kommt nun dadurch zustande, daß sich die einzelnen Berge und Täler einer solchen Welle immer näher kommen. D.h. sie kommen in einem schnelleren (oder langsameren) Rythmus auf uns zu. Die Anzahl der pro Sekunde auf uns zu kommenden Wellen nennen wir Frequenz. Sie nehmen wir als die Höhe eines Tones wahr.

Neben der Frequenz ist weiterhin noch die Lautstärke Teil eines Tones. Sie kommt durch die unterschiedliche Höhe von Bergen und Tälern einer Welle (Amplitude) zustande.

Der dritte Faktor, der die Tonqualität bestimmt, ist die sogenannte Wellenform. Die Wellenform ist, wie der Name schon sagt, die Form der Schallwelle, die unser Ohr erreicht. Hier gibt es natürlich die vielfältigsten Möglichkeiten. Ihr Commodore 64 bietet Ihnen vier Möglichkeiten, die Welle eines Tones zu bestimmen:

### 1.) DREIECKSSCHWINGUNG:



### 2.) SÄGEZAHNSCHWINGUNG:



### 3.) RECHTECKSCHWINGUNG:



### 4.) RAUSCHEN:



Dazu können diese Wellenformen noch untereinander logisch UNdiert werden.

Eine wesentliche Eigenschaft der Töne ist Ihr sogenanntes Obertonspektrum. Kein Ton (außer der Sinuston) folgt exakt diesem angegebenen Verlauf. Die einzelnen Wellenzüge schwingen vielmehr noch hin und her, als wären Sie mit einer zittrigen Hand gezeichnet. Diese auf der Welle liegenden Frequenzen hören wir ebenfalls, und sie sind die eigentlichen Kennzeichen eines Tones. Nur anhand dieser Obertöne können wir z.B. eine Geige von einer Flöte unterscheiden.

Die einzelnen Schwingungsarten, die Ihr Computer anbietet, besitzen nun unterschiedlich große Obertonspektren, weswegen wir diese unterscheiden können.

Nun ist es möglich, Teile dieser aufgelagerten Frequenzen ab- oder herauszuschneiden. Dies wird mit sogenannten Filtern erreicht. Filter kennen Sie von jedem modernen Radio: Der Hell-Dunkel - Knopf (bzw. Höhen und Baßregler). Mit Ihm

lassen sich z.B. einige hohe Töne wegfiltern, was dazu führt, daß die Musik aus dem Radio irgendwie belegt klingt.

Alle diese Möglichkeiten sind in Ihrem Computer verwirklicht worden. Sie bestimmen die Frequenz (Tonhöhe) eines Tones, die Wellenform, Lautstärke und die Frequenzen, die herausgefiltert werden sollen.

Doch eine Sache fehlt noch, die das ganze Bild abrundet: Der Tonverlauf.

Kein Ton ist plötzlich da und sofort wieder verschwunden. Es gibt Töne, die langsam lauter werden und wieder abklingen, oder Schüsse, die peitschenartig ertönen und ein kleinwenig nachhallen usw. Ihr Rechner ermöglicht es Ihnen nun, diesen Verlauf eines Tones zu programmieren. Dieser wird dabei in vier Phasen geteilt:

- 1.) ANSCHWELLPHASE
- 2.) ABSCHWELLPHASE
- 3.) HALTEPHASE
- 4.) AUSKLINGPHASE

Graphisch läßt sich das wie folgt darstellen:



Die englischen Ausdrücke hierfür sind: Attack, Decay, Sustain, Release

Der Ablauf ist dabei folgender: Zunächst klingt der Ton an. Sie geben die Zeit ein, die der Ton benötigt, um auf die volle (ebenfalls angegebene) Lautstärke anzuschwellen. Hernach wurde die Zeit bestimmt, die benötigt wird, um auf den bei Sustain angegebenen Lautstärkepegel abzuschwellen. Auf dieser Lautstärke bleibt der Ton stehen, bis er wieder abgeschaltet wird. Nun beginnt die Ausklingphase, deren Dauer Sie wiederum angegeben haben.

Die gesamte Verlaufsdefinition (Hüllkurve) wird natürlich zu

Anfang eines Musikstückes vorgegeben und bleibt bis zu ihrer Änderung erhalten.

Ihr Computer besitzt wie gesagt drei unabhängige Stimmen, die auch gleichzeitig spielen können. Für alle drei Stimmen können Sie somit die Frequenz, die Wellenform und die Hüllkurve angeben. Die Lautstärke und die Filter gelten jedoch für alle drei Stimmen gemeinsam. Natürlich gibt es noch eine Unmenge einzelner anderer Funktionen, die hier jedoch nicht im einzelnen dargelegt werden können, da sie Simon's Basic gar nicht ermöglicht. Zur näheren Information schauen Sie bitte in Ihrem Benutzerhandbuch oder einem sich speziell mit den Soundmöglichkeiten des CBM 64 beschäftigenden Buches nach. Nun wollen wir uns jedoch mit den einzelnen Befehlen zur Klangerzeugung beschäftigen.

## 15.2 VOL

```
FORMAT      :VOL L
PARAMETER   : - L: LAUTSTARKE (0-15)
BEISPIEL    :VOL 15
FUNKTION    :EINSTELLEN DER LAUTSTARKE DER
              ZU SPIELENDE TONE
```

### ERLAUTERUNGEN:

Der erste der insgesamt fünf Tonbefehle ist VOL (für VOLume). Mit Ihm haben Sie die Möglichkeit, die Lautstärke aller drei Tongeneratoren gleichzeitig festzulegen oder zu verändern. Zweckmäßigerweise geschieht dies am Anfang eines Programms, das sich mit der Musikgestaltung beschäftigt. Sie können Sie natürlich auch während des Programmes oder sogar während eines Tones (s. Beispiel) ändern, was spezielle Effekte mit sich bringt. Leider ist es hardwaremäßig nicht möglich, jeder Stimme eine eigene Lautstärke zu geben. Deshalb gilt 1 für alle 3 Stimmen gleichzeitig. Beim Betätigen dieses Befehles kann es zu einem kleinen Knacken kommen, was wohl in der Hardware begründet liegt. Lassen Sie l=0 werden, so sind alle drei Stimmen stumm, während l=15 die größte Lautstärke darstellt.

### BEISPIEL:

```
100 REM #####
110 REM ##                ##
120 REM ## VOL-BEISPIEL ##
130 REM ##                ##
140 REM #####
150 REM
160 REM
170 REM #####
180 REM ##                ##
190 REM ## TON          ##
200 REM ##                ##
210 REM #####
220 REM
230 VOL 15
240 ENVELOPE 1.1.8.10.10
```

```

250 WAVE 1.00100000
260 MUSIC 150,"S1c2I"
270 PLAY 2
280 REM
290 REM #####
300 REM ##           ##
310 REM ##  SCHWELLEN  ##
320 REM ##           ##
330 REM #####
340 REM
350 FOR X=0 TO 15
360 VOL X : REM LAUTSTAERKE
370 FOR W=1 TO 80 : NEXT W
380 NEXT X : REM LAUTSTAERKE ANSCHWELLEN LASSEN
390 PAUSE 2
400 VOL 0 : REM TON AUSSTELLEN
410 PAUSE 2
420 REM
430 REM #####
440 REM ##           ##
450 REM ##  ECHO  ##
460 REM ##           ##
470 REM #####
480 REM
490 REM
500 FOR X=15 TO 0 STEP -1
510 FOR Y=0 TO X
520 VOL Y : REM LAUTSTAERKE
530 FOR W=X TO 35 : NEXT W : REM WARTESCHLEIFE
540 NEXT Y : REM LAUTSTAERKE ANSCHWELLEN LASSEN
550 NEXT X : REM STETS LEISER WERDEND

```

Die Zeilen 240 bis 270 werden Ihnen sicher noch spanisch vorkommen, Sie sollten sie jedoch zunächst einmal hinnehmen als Befehle zur Erzeugung eines langen Tones. Wir werden die einzelnen Kommandos Schritt für Schritt durchgehen und erklären. Eines müssen Sie zum Eintippen des Programmes jedoch wissen: In der Zeile 260 werden einige seltsame Zeichen in Anführungsstrichen angegeben. Dies soll Ihnen eine Hilfe beim Übertragen des Programmes sein. Die großen Buchstaben bedeuten bei dem Befehl MUSIC von nun an

Kontrollzeichen. Diese Kontrollzeichen werden auch bei Ihnen auf dem Bildschirm als große, inverse Buchstaben gekennzeichnet, falls Sie den alternativen Zeichensatz eingeschaltet haben (also nach dem gleichzeitigen Drücken der Tasten <shift> und <c=>). Dabei sind jeweils die folgenden Kontrollzeichen gemeint:

S: <SHIFT><CLR/HOME>  
I: <F2>  
J: <F4>  
K: <F6>  
L: <F8>  
E: <F1>  
F: <F3>  
G: <F5>  
H: <F7>

Diese Tabelle besitzt bei allen Soundbeispielen in diesem Buch Gültigkeit. Wenn Sie also einmal nicht wissen, was gemeint ist, so schlagen Sie hier nach.

Alle Kleinbuchstaben in diesem MUSIC-Befehl sind ganz normale Tasten. Man könnte die Zeile 260 in diesem Programm also auch schreiben:

```
260 MUSIC 150. CHR$(147) + "1c2" +  
CHR$(137)
```

### 15.3 WAVE

**FORMAT** : WAVE ST. RRS DTRSG  
**PARAMETER** : - ST: STIMME. FÜR DIE DIE  
WELLENFORM ANGEGEBEN  
WIRD (1-3)  
- RRS DTRSG: FOLGE VON ACHT MAL  
1 ODER 0 ZUM EIN- (1)  
UND AUSCHALTEN (0)  
BESTIMMTER FUNKTIONEN  
**BEISPIEL** : WAVE 2.10000101  
**FUNKTION** : EINSTELLEN DER WELLENFORM DER  
ZU SPIELENDEN TÖNE

#### ERLÄUTERUNGEN:

Wir sind bereits mitten in der Klangsynthese! WAVE gibt Ihnen die Möglichkeit, die einzelnen vom Computer bereitgestellten Wellenformen (s. Einleitung zu diesem Kapitel) jeweils für eine der drei möglichen Stimmen festzulegen. Neben dieser Aufgabe können Sie noch eine Reihe weiterer Funktionen wahrnehmen.

Zunächst jedoch einmal etwas zu den anzugebenden Parametern: Der erste der zwei anzugebenden Werte bestimmt die Stimme, für die Ihre Wellendefinition gelten soll. Bekanntlich können Sie diese Tonqualität jeweils für alle drei Stimmen unterschiedlich auswählen.

Der zweite Parameter mag Ihnen auf den ersten Blick ein wenig diffus erscheinen. Diese Verwirrung lichtet sich aber gleich wieder. Sie können mit dem Befehl WAVE acht verschiedene Funktionen auswählen, die weiter unten beschrieben sind. Alle diese Funktionen besitzen nur zwei Möglichkeiten der Realisierung: an oder aus. Sie haben nichts weiter zu tun, als nacheinander jeweils eine 1 für eine eingeschaltete Funktion (z.B. Wellenform Dreieck) oder eine 0 für eine ausgeschaltete einzutippen. Bei den acht möglichen Funktionen ergeben sich acht Einsen oder Nullen ohne ein Zeichen voneinander getrennt, die oben in der Befehlskurzbeschreibung mit rrsdtrsg abgekürzt wurden.

Mit diesem Befehl programmieren Sie direkt ein Register des SID (Soundprozessor) (Register 4, 11 und 18 für die drei Stimmen - siehe Benutzerhandbuch auf den Seiten 155-157). Die



acht einzelnen Werte, die Sie als zweiten Parameter einzugeben haben, stellen die acht Bits dieses Registers dar, die die verschiedenen Aufgaben übernehmen. Diese sind im folgenden erläutert:

Zunächst stellen wir einmal die Funktionen der einzelnen Bits übersichtlich dar:

<b>BIT</b>	<b>...FUNKTION</b>
<b>7</b>	<b>....RAUSCHEN</b>
<b>6</b>	<b>....RECHTECK</b>
<b>5</b>	<b>....SÄGEZAHN</b>
<b>4</b>	<b>....DREIECK</b>
<b>3</b>	<b>....TESTBIT</b>
<b>2</b>	<b>....MODULATION</b>
<b>1</b>	<b>....SYNCHRONISATION</b>
<b>0</b>	<b>....GATEBIT</b>

Hier nun die ausführlichere Erklärung (vergl. auch CBM 64 Benutzerhandbuch):

#### **BIT 0 - GATEBIT:**

Dieses Bit ist zur Initialisierung und zum Beenden eines Tones gedacht. Wird es auf 1 gesetzt, so beginnt der mit den anderen Registern eingestellte Ton gemäß seiner vorbestimmten Hüllkurve (s. Einleitung) zu erklingen. Er wird solange in der Sustain - Phase gehalten (also der Haltelautstärke), bis dieses Bit wieder auf Null zurückgesetzt wird. Ist dies geschehen, klingt er - wie durch Release definiert - aus. Normalerweise wird dieses Bit automatisch durch Simon's Basic gesteuert. Es muß jedoch in einer WAVE - Anweisung gesetzt werden, wenn zwischendurch, während ein Ton klingt ein Parameter durch WAVE geändert werden soll, da der Ton ansonsten ausgeschaltet wird. Umgekehrt können Sie dieses Bit natürlich zum Ausschalten eines Tones gebrauchen.

#### **BIT 1 - SYNCHRONISATION:**

Lassen Sie gleichzeitig Töne auf mehreren Stimmen, also auf mehreren Oszillatoren spielen, so werden Sie sehen, daß je nach Ton verschiedene Schwebungen zu vernehmen sind. Diese Schwebungen kommen dadurch zustande, daß z.B. zu einem Zeitpunkt ein Schwingungstal (s. Einleitung) der einen Stimme

mit einem Schwingungsberg der anderen zusammentrifft. In diesem Falle heben sich die beiden Wellen natürlich auf, und Sie hören nichts. Bald darauf aber wird einmal ein Berg auf einen anderen Schwingungsberg stoßen, was zu einer gegenseitigen Verstärkung führt. Der Ton wird lauter. Diese beiden Extremen wechseln ständig. Aus diesem Grunde hören Sie ein Schwingen des Tones.

Manchmal ist dieser Effekt aber auch störend. Dann setzen Sie einfach das Synchronisationsbit und schon schwingen die beiden Töne in gleicher Phase. Das Bit 1 besitzt in den drei Stimmen unterschiedliche Aufgaben:

<i>STIMME</i>	<i>SYNCHRONISATION</i>
1.....	<i>STIMME 1 MIT 3</i>
2.....	<i>STIMME 2 MIT 1</i>
3.....	<i>STIMME 3 MIT 2</i>

#### ***BIT 2 - RINGMODULATION:***

Lesen Sie hierzu bitte die Ausführungen in Ihrem Simon's Basic Handbuch

#### ***BIT 3 - TESTBIT:***

Wenn zusammen mit dem Rauschgenerator noch eine weitere Schwingungsform derselben Stimme ausgewählt wurde, kann es vorkommen, daß der Rauschgenerator blockiert. Die Blockade kann durch dieses Bit wieder aufgehoben werden.

#### ***BIT 4 - DREIECKSCHWINGUNG:***

Wird dieses Bit gesetzt, so haben Sie die Dreieckschwingung ausgewählt. Sie hat Ähnlichkeit mit der reinen Sinusschwingung, die bekanntlich keine harmonischen Obertöne aufweist. Sie ähnelt dem Ton einer Flöte.

#### ***BIT 5 - SÄGEZAHNSCHWINGUNG:***

Die Sägezahnschwingung hat viele ungerade und gerade Obertöne und klingt recht hart (ähnlich einer Geige).

#### ***BIT 6 - RECHTECKSCHWINGUNG:***

Bei der Rechteckschwingung können Sie zusätzlich das Verhältnis der Dauer der Täler und Berge (s. Einleitung) variieren, das sogenannte Pulsverhältnis. Simon's Basic

bietet keine Möglichkeit dies innerhalb eines Befehls zu erledigen, obwohl das Pulsverhältnis vor der Verwendung der Rechteckschwingung einmal gesetzt werden muß. Somit müssen wir uns mit zwei POKEs behelfen:

**POKE 54272 - 5 + 7\*ST, LB : POKE 54272 - 4 + 7\*ST,  
HB**

Dabei beinhaltet die Variable ST die Nummer der Stimme, deren Pulsverhältnis verändert werden soll. HB stellt das High-Byte und LB das Low-Byte des einzugebenden Wertes dar.

#### **BIT 7 - RAUSCHEN:**

Mit dem Setzen dieses Bits wird der Rauschgenerator der betreffenden Stimme eingeschaltet. Dieser Modus ist sehr vielseitig. Sie können hiermit Meeresrauschen, Schüsse, Zischen und vieles mehr erzeugen.

Eine Anmerkung zu den Bits 4-7: Es ist praktisch möglich, mehrere Schwingungsformen gleichzeitig auszuwählen. Zu beachten ist jedoch, außer dem zu Bit 3 Gesagten, daß das Ergebnis nicht etwa die Summe aller Formen darstellt, sondern vielmehr eine logische UND - Verknüpfung der Komponenten (s. Beispiel).

#### **BEISPIEL:**

```
100 REM #####  
110 REM ##          ##  
120 REM ## VOL-BEISPIEL ##  
130 REM ##          ##  
140 REM #####  
150 REM  
160 REM  
170 REM      #####  
180 REM      ##          ##  
190 REM      ## TON ##  
200 REM      ##          ##  
210 REM      #####  
220 REM  
230 VOL 15  
240 ENVELOPE 1. 10. 10. 10. 10  
250 MUSIC 250, "S1C2I"
```

260 PLAY 2  
270 REM  
280 REM #####  
290 REM ##                   ##  
300 REM ## WELLENFORM ##  
310 REM ##                   ##  
320 REM #####  
330 REM  
340 REM SAEGEZAHN:  
350 WAVE 1.00100001  
360 PAUSE 2  
370 REM RAUSCHEN:  
380 WAVE 1.10000001  
390 PAUSE 2  
400 REM DREIECK/SAEGEZAHN:  
410 WAVE 1.00110001  
420 PAUSE 2  
430 REM DREIECK:  
440 WAVE 1.00010001  
450 PAUSE 2  
460 REM DREIECK/RECHTECK:  
470 WAVE 1.01010001  
480 PAUSE 2  
490 REM RECHTECK/SAEGEZAHN:  
500 WAVE 1.01100001  
510 PAUSE 2  
520 REM DREIECK/RECHTECK/SAEGEZAHN:  
530 WAVE 1.01110001  
540 PAUSE 2

Für das Verständnis der Zeilen 230-260 schauen Sie bitte unter der Anmerkung zum VOL - Beispiel nach.

## 15.4 ENVELOPE

**FORMAT** : ENVELOPE ST.A.D.S.R  
**PARAMETER** : - ST: STIMME. FÜR DIE DIE  
HÜLLKURVE ANGEGEBEN  
WIRD (1-3)  
- A : ATTACK - DAUER DER  
ANSCHWELLPHASE (0-15)  
- D : DECAY - DAUER DER  
ABSCHWELLPHASE (0-15)  
- S : SUSTAIN - LAUTSTARKE IN  
DER HALTEPHASE (0-15)  
- R : RELEASE - DAUER DER  
AUSKLINGPHASE (0-15)  
**BEISPIEL** : ENVELOPE 3.9.6.10.15  
**FUNKTION** : EINSTELLEN DER HÜLLKURVE DER  
ZU SPIELENDE TONE

### ERLÄUTERUNGEN:

Nun kommen wir zum wirklich kreativen Teil der Klangproduktion, der Hüllkurvenbestimmung. Wir hatten bereits in der Einleitung zu diesem Kapitel beschrieben, was unter einer Hüllkurve zu verstehen ist, nämlich das Festlegen des Ablaufs eines Tones bezüglich seiner Lautstärke. Diese Eigenschaft von Tönen kann, wie bereits erläutert, ebenfalls für alle drei Stimmen getrennt definiert werden. Mit Hilfe der vier Parameter, die jeweils für die einzelnen Tonphasen zuständig sind, haben Sie ein derart großes Spektrum an Möglichkeiten, daß es Ihnen schwer fallen wird, einen richtigen Überblick zu bekommen. Bei der Überlegung, wie ein Ton dargestellt werden soll, gibt es zwei Möglichkeiten, um auf den Ihnen gefallenden Sound zu kommen:

1.) Sie probieren solange herum und ändern die einzelnen Parameter, bis Sie einen (vorher natürlich noch nicht bekannten) schönen Ton zustandegebracht haben. Dies sollten Sie sowieso stets tun, um überhaupt einmal eine Vorstellung darüber zu bekommen, wie sich die einzelnen Parameter auswirken und welche Möglichkeiten es gibt. Die besagte Methode ist jedoch vor allem bei der Erzeugung einzelner Töne zur Effektherstellung anzuwenden.

2.) Wollen Sie jedoch einen vorgegebenen Klang (z.B. eine Flöte) nachahmen, so müssen Sie gedanklich (oder auf dem Papier) erst einmal analysieren, wie denn eigentlich der Tonverlauf des natürlichen Tones ist, um dann die entsprechenden Parameter zu setzen. Dies ist natürlich ein sehr viel zielbewußterer Weg. Ein Flötenton hat z.B. meist eine lange Anschwellphase (Attack wird groß) und wird auf voller Lautstärke gehalten, ohne daß der Ton vorher merklich unter die maximale Lautstärke fiel (Sustain wird groß) und wenn, dann nur sehr langsam (Decay groß). Die Ausklingphase dagegen ist kaum hörbar, da der Ton meist recht abrupt endet (Release klein).

### **BEISPIEL :**

Als Beispiel wird Ihnen hier nun ein kleines Programm gebracht, in dem Sie in der Zeile 290 die einzelnen Parameter des ENVELOPE - Befehls verändern sollten, um auf diese Weise einen kleinen Überblick über die einzelnen Möglichkeiten der HüllkurvenEinstellung zu erhalten. Ändern Sie doch auch einmal die Wellenform in der Zeile 410. Vielleicht wählen Sie einmal das Rauschen, bei dem die Hüllkurve besonders gut zu hören ist.

```
100 REM #####
110 REM ##                               ##
120 REM ## ENVELOPE-BEISPIEL ##
130 REM ##                               ##
140 REM #####
150 REM
160 REM
170 REM #####
180 REM ##                               ##
190 REM ## HUELLKURVE ##
200 REM ##                               ##
210 REM #####
220 REM
230 REM TESTBEISPIEL :
240 REM BITTE AENDERN SIE DIE PARAMETER
250 REM
```

```

260 REM      ST! AI DI SI R
270 REM -----+--+--+--+-----
280 REM      ! ! ! !
290 ENVELOPE 1, 10, 10, 5, 9
300 REM -----+--+--+--+-----
310 REM
320 REM
330 REM      #####
340 REM      ##      ##
350 REM      ## TON ##
360 REM      ##      ##
370 REM      #####
380 REM
390 VOL 15
400 REM SAEGEZAHN:
410 WAVE 1.00100001
420 MUSIC 50, "S1c2I"
430 PLAY 2
440 PAUSE 1 : REM AUSSCHALTEN:
450 WAVE 1.00100000

```

Zu den Befehlen in den Zeilen 420 und 430 und besonders der Schreibweise des Strings in Zeile 420 schauen Sie bitte unter den Bemerkungen zu den VOL-Beispiel in # 15.2 nach.

## 15.5 MUSIC

**FORMAT** :MUSIC TD,STR  
**PARAMETER** : - TD : TONDAUER ODER ZEITMAß,  
MIT DEM DAS STÜCK  
GESPIELT WERDEN  
SOLL (0-255)  
- STR: STRINGSPEICHER ODER  
"ZEICHENKETTE". DIE DIE  
KLANGABFOLGE FESTLEGEN  
**BEISPIEL** :MUSIC 8.A\$  
**ODER** MUSIC 10."...TÖNE..."  
**FUNKTION** :FESTLEGEN DER TONABFOLGE  
EINES MUSIKSTÜCKES

### ERLÄUTERUNGEN:

Wir kommen nun zu einem Befehl, der es Ihnen ermöglicht, eine bestimmte Tonfolge, die Sie spielen möchten (etwa ein Musikstück - s. Beispiel) festzulegen. Diese Tonfolge merkt sich der Computer und spielt sie automatisch hintereinander ab, wenn Sie ihm dies (mit einem PLAY-Kommando) befehlen. Mit td geben Sie ihm dabei an, mit welcher Geschwindigkeit er die Noten hintereinander spielen soll. Damit sind nun nicht die Notenwerte der einzelnen Töne gemeint (die geben Sie mit der Tonhöhe ein), sondern vielmehr das Zeitmaß, also wie schnell ein Stück als Gesamtheit sein soll (In der Musik gibt es dafür verschiedene Tempi - Angaben; z.B.: Allegro, Andante oder Presto). Dabei ist die Grundeinheit 1/60 Sekunde. Bei td=0 besitzt eine ganze Note (1/1) die Länge von 2/60 Sekunden. Eine Halbe dauert dann genau 1/60 Sekunde. Noch kleinere Notenwerte dauern dabei nicht kürzer. Bei td=1 liegt die Notendauer einer 1/1 Note schon bei 5/60 = 1/12 Sekunde. Bei dem Versuch eine 1/4 Note zu spielen, werden Sie einige Ungenauigkeiten zu hören bekommen, da 5 bekanntlich nicht ohne Rest durch 4 teilbar ist. Allgemein läßt sich die zeitliche Dauer z einer ganzen Note etwa durch folgende Formel bestimmen (in Sekunden):

$$Z = TO/12 \quad \text{ODER} \quad Z = 5*TO/60$$

Diese Formel liefert jeweils nur ungefähre Werte, die von Mal



zu Mal auch schwanken können (td=0 fällt vollkommen aus der Rolle). Die einzelnen anderen Notenwerte sind Bruchteile oder Vielfache des Ergebnisses. Um nun eine möglichst genaue Bruchteilbildung, also möglichst genau gespielte Viertel, Achtel usw. zu erhalten, müssen Sie Werte für td einsetzen, die durch 4 bzw. 8 ohne Rest teilbar sind.

Nun aber zu der eigentlichen Notendefinition, die in dem nachfolgenden String angegeben werden kann:

Am besten schalten Sie zu diesem Zwecke stets Ihr Gerät um auf den alternativen Zeichensatz (Klein- / Großschrift), indem Sie die <shift>- und die <C=>- Taste gleichzeitig drücken. Sollte dies nicht funktionieren (was beim Simon's Basic schon einmal vorkommt), so geben Sie vorher ein:

### **PRINT CHR\$(9)**

(s. Zeichensatzkapitel). Durch den alternativen Zeichensatz sind die einzelnen einzugebenden Zeichen besser zu unterscheiden und entsprechen gleichfalls den Zeichen, die in den einzelnen Beispielprogrammen angegeben sind.

Ganz am Anfang des Strings müssen Sie bestimmen, für welche Stimme Sie die folgenden Noten zu spielen beabsichtigen. Dies geben Sie ein, indem Sie (nach Eingabe der Anführungszeichen) <shift> <clr/home> drücken. Es erscheint ein inverses Herzchen (Zeichensatz A) oder ein inverses S (Zeichensatz B = alternativ). direkt dahinter folgt nun die Nummer der anzusprechenden Stimme (1-3). Diesen Ausdruck können Sie beliebig oft in Ihrem Musikstück verwenden, um auf eine andere Stimme umzuschalten. Dies kann z.B. nützlich sein, wenn Sie in den 3 Stimmen andere Wellenformen oder Hüllkurven etc. ausgewählt haben und mitten im Stück den Klang wechseln wollen. Ein mehrstimmiges Spiel (mehrere Stimmen gleichzeitig) ist aber soweit ersichtlich nicht möglich (außer, daß der letzte Ton der vorherigen Stimme in die Töne der nächsten Stimme hineinklingt).

Nun beginnt die eigentliche Klangprogrammierung: Als nächsten Parameter geben Sie zunächst die zu spielende Note ein (nicht, wie das Handbuch darlegt, zuerst den Notenwert!). Diese bestimmen Sie aus zwei Teilen. Erstens den Notennamen, zweitens die Oktave. Für den Notennamen werden die geläufigen Buchstaben der Tonleiter in der amerikanischen Schreibweise eingegeben:

(bitte beachten Sie den letzten Buchstaben. Er entspricht dem deutschen H (nicht dem deutschen b (= erniedrigtes H) !), was im Handbuch fehlerhaft angegeben ist) Wollen Sie einen Ton um einen Halbton erhöhen (also mit einem # versehen), was ja der Erniedrigung des darüberliegenden Tones entspricht, so drücken Sie den entsprechenden Buchstaben zusammen mit <shift> ein. Im alternativen Zeichensatz erscheint dann ein großer (statt ein kleiner) Buchstabe.

Im Anschluß hieran geben Sie nun eine Ziffer von 0-7 für die entsprechende Oktave an, in der der Ton liegen soll (Zur Orientierung: der Kammerton A mit einer Frequenz von 440 Herz liegt in der 4. Oktave. Sie haben also einzugeben: a4). Der höchste Ton, den Sie eingeben können, ist das erhöhte A der 7. Oktave: A7.

Um die Angaben für eine Note abzuschließen, geben Sie nun noch den Notenwert an, den Sie für diese Note vorgesehen haben. Dieser wird mittels der Funktionstasten eingegeben. Dabei erhalten die einzelnen Funktionstasten folgende Notenwerte:

TASTE	NOTENWERT	BILDSCHIRMZEICHEN
F1	1/16	INVERS-GROßES E
F3	1/8	INVERS-GROßES F
F5	1/4	INVERS-GROßES G
F7	1/2	INVERS-GROßES H
F2	1/1	INVERS-GROßES I
F4	2/1	INVERS-GROßES J
F6	4/1	INVERS-GROßES K
F8	8/1	INVERS-GROßES L

Mit Bildschirmzeichen ist das Zeichen gemeint, das auf dem Bildschirm erscheint, wenn Sie (nachdem Sie ein "Anführungszeichen" eingegeben haben) die entsprechende Funktionstaste drücken (vorausgesetzt, Sie befinden sich im alternativen Zeichensatz mit Groß- / Kleinschreibung).

Damit haben Sie eine Note definiert. Die nächste, die wieder aus drei Zeichen besteht, wird nun direkt hintenangefügt. Wollen Sie z.B. eine 3/2 Note (1 1/2) spielen, so können Sie auch mehrere Funktionstasten als Längenangaben hintereinander

eingeben, die dann jeweils in Ihren Notenwerten addiert werden. In unserem Fall einer 3/2 währe das: f7 f7 f7 oder f2 f7 (angegeben ist hier natürlich die Tastenfolge, nicht das auf dem Bildschirm erscheinende Zeichen).

Normalerweise werden die einzelnen Noten gebunden (Legato) gespielt. Wollen Sie einen kleinen Absatz, so geben Sie ein: <shift> <clr/home> mit einem nachfolgenden g. Dies sollten Sie gleichfalls tun, wenn Sie die Stimme wechseln und die vorherige Stimme nicht weiterklingen lassen wollen. Dieses Eingabe schaltet den Ton aus. Fügen Sie hinter dieses Kommando eine (oder mehrere) Notenwertangabe(n), so entsteht eine Pause mit der angegebenen Länge.

Auf diese Weise können Sie recht einfach ein kleines Liedchen programmieren. Größere Lieder definieren Sie (wie im Beispiel) durch Zusammenfügen mehrerer Strings. So können Sie auch einfach Wiederholungen einführen (s.u.). Das Beispiel hierzu finden Sie unter PLAY (# 15.6).

## 15.6 PLAY

```
FORMAT      :PLAY M
PARAMETER   : - M: SPIELMODUS:
              M=1: AUF SPIELLENDE WARTEN
              M=2: SPIELEN UND RECHNEN
BEISPIEL    :PLAY 1
FUNKTION    :SPIELEN DER MIT MUSIC
              FESTGELEGTE TONFOLGE
```

### ERLÄUTERUNGEN:

Endlich ist es soweit, das viele theoretische Wissen in der Praxis anzuwenden. Endlich können wir etwas Eigenes produzieren. Mit dem PLAY-Befehl nun ist es uns möglich, die Tonfolge, die wir in dem MUSIC-Befehl festgelegt haben, samt den Eigenschaften, die die übrigen Befehle determinieren, abspielen und uns damit berauschen zu lassen.

Der PLAY-Befehl kennt zwei Modifikationen, die mit dem einzigen Parameter m ausgewählt werden. Setzen Sie m=1, so wartet das Programm solange, bis die Musik ausgespielt hat, d.h. alle Befehle, die in Ihrem Programm hinter PLAY 1 kommen, werden erst nach Beendigung der Tonfolge ausgeführt. In der Zeit ist das Programm auch nur durch <run/stop> <restore> abubrechen.

Geben Sie statt dessen PLAY 2 ein, so werden sofort nach dem Start der Musik die folgenden Befehle ausgeführt, d.h. Ihr Programm läuft weiter, während die Musik spielt! Dies ist anschaulich in dem unten stehenden Beispielprogramm demonstriert. Endet Ihr Programm jedoch vor Ablauf der Musik, so wird das Spiel ebenfalls abgebrochen, wobei der gerade gespielte Ton stehenbleibt und erst einmal abgeschaltet werden muß (mit VOL 0 oder WAVE st,00000000). Der im Handbuch beschriebene Befehl PLAY 0 hat keine Wirkung. PLAY 2 kann auch sehr gut angewendet werden, wenn Sie während der Musik verschiedene Klangparameter verändern wollen, wie dies in den Beispielen unter VOL und WAVE anschaulich dargestellt wurde. Leider haben Sie keine Kontrolle darüber, wann im Laufe des Programms die Musik geendet ist. Aus diesem Grunde müssen Sie das Timing Ihres Programmes durch Ausprobieren bestimmen. Dann sehen wir uns doch einfach einmal das Beispiel an:

**BEISPIEL :**

```
100 REM #####
110 REM ##                                     ##
120 REM ## MUSIC/PLAY-BEISPIEL ##
130 REM ##                                     ##
140 REM #####
150 REM
160 REM
170 REM #####
180 REM ##                                     ##
190 REM ## ADE ZUR GUTEN NACHT ##
200 REM ##                                     ##
210 REM #####
220 REM
230 VOL 15 : REM LAUTSTAERKE
240 REM
250 REM NOTEN :
260 REM
270 A$ = "S1c3Gf3Hg3G%A3Ga3Ga3Fg3Ff3Gg
        3Ga3H%A3Gd4Gc4Gc4F%A3Fa3G"
280 A$ = A$ + "c4Gc4G%A3Fa3F%A3Gc4Ga3HSgH"
290 B$ = "c4Ga3Hc4Gf4Gd4Gd4Fc
        4F%A3G%A3Gg3H%A3Gd4Gc4G"
300 B$ = B$ + "c4F%A3Fa3Gc4Gc4G%
        A3Fa3F%A3Gc4Ga3HGSgE"
310 REM
320 REM
330 REM DREIECK:
340 WAVE 1,00010001
350 ENVELOPE 1.4.5. 5. 9
360 REM #####
370 MUSIC 10.A$ + B$ + B$
380 PLAY 2
390 REM #####
400 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
410 PRINT"ADE ZUR GU-TEN NACHT. "
420 FOR W=1 TO 2600 : NEXT W
430 PRINT"JETZT WIRD DER SCHLUSS GEMACHT. "
440 REM
450 FOR W=1 TO 2600 : NEXT W
```

```

460 PRINT "DASS ICH MUSS SCHEI-DEN."
470 FOR W=1 TO 2600 : NEXT W
480 FOR X=1 TO 2
490 PRINT
500 PRINT "IM SOMMER. DA WAECHST DER KLEE."
510 FOR W=1 TO 2600 : NEXT W
520 PRINT "IM WINTER. DA SCHNEIT'S DEN SCHNEE."
530 FOR W=1 TO 2600 : NEXT W
540 PRINT "DA KOMM ICH WIE-DER."
550 FOR W=1 TO 2600 : NEXT W
560 NEXT X

```

Ich hoffe, es fällt Ihnen nicht allzu schwer, die einzelnen Buchstaben in den Strings zur Tonfolgendefinition zu verstehen. Allgemein sei hier auf die Ausführungen unter VOL (# 15.2) verwiesen. Hier seien noch einmal kurz die wichtigsten Dinge wiederholt (wichtig ist jetzt, daß Sie sich im alternativen Zeichensatz befinden!):

Kleine Buchstaben sind normale Noten und werden durch einfachen Tastendruck auf die entsprechende Buchstabentaste eingegeben. Sie erscheinen ebenfalls als kleine Buchstaben auf dem Bildschirm.

Große Buchstaben in diesem Listing bedeuten Kontrollzeichen. Dies sind die 8 Funktionstasten (E-L) und die <shift> <clr/home> - Eingabe (S). Sie erscheinen auf der Mattscheibe als inverse Großbuchstaben.

Großbuchstaben, die dem Zeichen "%" folgen, bedeuten erhöhte Noten und werden durch das Drücken der entsprechenden Buchstabentaste zusammen mit <shift> eingegeben. Sie erscheinen als normale Großbuchstaben in dem Textfenster.

Zahlen werden ganz normal eingegeben und bestimmen die Stimme bzw. die Oktaven. Viel Spaß !

## 15.6 TESTAUFGABEN

Zum Abschluß wieder unser Minitest:

1.) Was wird durch den Befehl VOL 2 erreicht?

- a) Die Lautstärke für alle Stimmen wird auf Stufe 2 gesetzt
- b) Die Lautstärke wird für Stimme 2 bestimmt
- c) Die Syntax ist nicht richtig
- d) VOL 2 ist die drittlauteste Einstellung

2.) Was ist bei WAVE zu beachten?

- a) WAVE stellt die Wellenform für alle Stimmen ein
- b) Rauschen sollte nicht mit einer anderen Wellenform kombiniert werden
- c) Die Syntax lautet: WAVE 2,%00010001

3.) Wie ist der Befehl MUSIC zu bedienen?

- a) gibt es gar nicht
- b) Zunächst wird das Zeitmaß eingegeben, dann die Notenfolge
- c) Die umgekehrte Reihenfolge von b)
- d) Eine Note wird definiert, indem man den Notennamen, die Oktave und dann den Notenwert eingibt
- e) Erniedrigte Töne werden eingegeben, indem man den Notennamen <shift>et.

## **16. KAPITEL**

### **STEUERENDE PERIPHERIEN**

Ihr Commodore 64 besitzt verschiedene Möglichkeiten, um steuernde Geräte als Peripheriebausteine anzuschließen. Zu diesem Zwecke befinden sich an der rechten Seite Ihres Computers zwei sogenannte Controlports. Dies sind Steckbuchsen für den Anschluß von Joystick, Paddle, Lightpen oder gar selbstgebaute Steuereinheiten. Die Steckerbelegung (d.i. die Funktion der einzelnen Pinne) wird in Ihrem CBM 64 - Benutzerhandbuch auf der Seite 141 beschrieben. Sie brauchen diese nicht unbedingt zu kennen, um z.B. einen Joystick an Ihr Gerät anzuschließen und ihn richtig zu gebrauchen. Aus diesem Grunde verzichten wir hier auf eine nähere Erläuterung.

Die einzelnen Gerätschaften, die Ihr Rechner und auch Simon's Basic unterstützt, seien hier kurz beschrieben und die entsprechenden Befehle erläutert.

#### **16.1 LIGHTPEN**

Unter Lightpen (oder Lichtgriffel) versteht man einen handlichen Stift, der zur Eingabe oder Bestimmung eines Punktes auf dem Bildschirm dient und den direkten Kontakt zwischen Ihnen und dem Fernseher gestattet. Mit dem Lichtgriffel ist es also möglich, durch ein simples Auflegen der Stiftspitze auf den Bildschirm dem Computer eine Bildschirmposition einzugeben.

Wie geht das nun vonstatten? Sie zeigen mit Ihrem in Controlport 1 gesteckten Lichtgriffel auf einen Punkt des Bildschirms. Dabei ist es egal, ob sich dieser Punkt innerhalb oder außerhalb des eigentlichen Textfensters befindet. Der Computer ist alsdann in der Lage, diesen Punkt zu identifizieren, er kennt also die Koordinaten dieses Punktes. Wenn Sie diese in Ihrem Programm abfragen, können Sie beispielsweise feststellen, ob sich an der Stelle ein bestimmtes Objekt (Buchstabe oder eine Graphik) befindet.



Oder Sie zeichnen genau an dieser Stelle einen Punkt in die Graphik, sodaß Sie per Hand auf den Bildschirm zeichnen können! Eine andere Idee wäre, den Lightpen als komfortable Cursorsteuerung einzusetzen. Es gibt eine Menge Möglichkeiten der Verwendung.

Zunächst muß noch Einiges zu dem Koordinatensystem gesagt werden, da bei der Lightpenverwendung Unterschiede zu der uns bekannten Graphikeinteilung auftauchen. Gleichzeitig bestehen oft von Fernseher zu Fernseher einige Differenzen bezüglich der Randwerte. Die hier angegebenen Richtwerte können und sollten also von Ihnen korrigiert werden. Zunächst einmal wird, wie gesagt, jeweils vom Bildschirmrand und nicht vom Rand des Textfensters aus gemessen. Dabei liegt der linke Rand etwa bei  $x=30$ . Die  $x$ -Koordinate des linken Textfensterandes lautet  $x=40$ , die des rechten 200. Das Ende des Bildschirms zur rechten Seite liegt bei  $x=210$ . In  $y$ -Richtung, also gemessen vom oberen Bildschirmrand ist die Angelegenheit noch etwas komplizierter. Ebenso wie bei der  $x$ -Einteilung starten die  $y$ -Koordinaten bei  $y=30$ . Der obere Rand des Textfensters liegt bei  $y=40$ , der untere bei 240. Nun jedoch wird der untere Rand des Bildschirms (Rahmen) etwa in zwei Hälften geteilt. Bis zu  $y=255$  geht der Koordinatenverlauf noch regelmäßig. Dann aber beginnt die Zählung in der Mitte des Rahmens wieder bei  $y=0$  und endet schließlich am "Boden" mit  $y=25$ .

Um also die Lightpenkoordinaten auf die uns bekannte Einteilung umzurechnen, müssen wir durch folgende Formeln die Koordinaten transformieren:

$$X = (XP-40)*2$$

$$Y = YP-40$$

wobei  $x$  und  $y$  die Graphik- und  $x_p, y_p$  die Lightpenkoordinaten darstellen. Sie besitzen also in  $x$ -Richtung etwa die halbe Auflösung, was jedoch nicht viel ausmacht, da der Lichtgriffel sowieso von Natur aus etwas in  $x$ -Richtung streut. Um nun in der obigen Formel keine negativen oder zu große Werte zu erhalten, müssen Sie in einer folgenden Basiczeile auf die Richtigkeit dieser Koordinaten prüfen, sie dies im Beispiel gezeigt wurde (s.u.).

## 16. 1. 1 PENX

<b>FORMAT</b>	<b>:PENX</b>
<b>PARAMETER</b>	<b>: ----</b>
<b>BEISPIEL</b>	<b>:X = PENX</b>
<b>FUNKTION</b>	<b>:BESTIMMEN DER X-KOORDINATE DES LIGHTPEN</b>

### ERLÄUTERUNGEN:

Um die Koordinaten der aktuellen Position eines Lightpen auf dem Bildschirm festzustellen, wären eine ganze Reihe komplizierter Schritte mit PEEKs und POKEs zu tun, die mit den verschiedenen Registern des Videocontrollers zusammenhängen. Simon's Basic macht es Ihnen dabei jedoch sehr leicht:

Der Befehl PENX gestattet es Ihnen, direkt, ohne Umschweife die x-Koordinate des Lightpen zu erfahren. PENX wird analog zu TEST, LIN oder auch SQR wie eine Funktion verwendet. Sie können also PENX in Variablen abspeichern oder damit rechnen:

**X = 5 \* PENX + 4**

Dabei besitzt dieser Befehl stets die letzte Position des Stiftes als Wert. Im Unterschied zu der altgewohnten Koordinatenbestimmung mit dem Nullpunkt in der oberen linken Ecke des Textfensters, wird bei PENX der Abstand des Punktes von der linken Kante des Bildschirms, also Textfenster plus Rahmen angegeben. Trotzdem ist die Auflösung, also die maximale Punktezahl in einer Zeile weitaus kleiner als wir von der hochauflösenden Graphik gewohnt sind. Die Punkteinteilung ist etwas kompliziert und wird in der Einleitung zu diesem Abschnitt erläutert.

## 16.1.2 PENY

```
FORMAT      :PENY
PARAMETER   : ----
BEISPIEL    :Y = PENY
FUNKTION    :BESTIMMEN DER Y-KOORDINATE
             DES LIGHTPEN
```

### ERLÄUTERUNGEN:

PENY gibt Ihnen in der gleichen Weise wie oben unter # 16.1.1 beschrieben die y-Koordinate der aktuellen (bzw. letzten) Bildschirmposition des Lightpen an. Wieder wird direkt vom Bildschirmrand aus gemessen. Diesmal startet die Zählung bei  $y=30$  an der oberen Kante. Von oben nach unten gibt es insgesamt etwa 250 Punkte. Wollen Sie die entsprechenden Koordinaten in der Graphik des Simon's Basic wissen, so müssen Sie sie zunächst einmal umrechnen, wie dies in der Einleitung beschrieben wurde und im nachfolgenden Beispiel gehandhabt ist.

### BEISPIEL:

Dieses Beispiel ermöglicht Ihnen, direkt mit dem Lightpen quasi als Zeichenstift auf den Graphikbildschirm zu zeichnen. Probieren Sie es aus. Sie können also jetzt direkt Ihre Unterschrift eingeben oder Zeichnungen wie auf dem Papier erstellen.

```
100 REM #####
110 REM ##                               ##
120 REM ## PENX/PENY-BEISPIEL ##
130 REM ##                               ##
140 REM #####
150 REM
160 HIRES 6.7
170 PROC PLOT
180 X = (PENX - 40)*2 : REM X-KOORDINATE ERECHNEN
190 Y = PENY - 40 : REM Y-KOORDINATE ERRECHNEN
200 IF X<0 OR X>319 OR Y<0 OR Y>199 THEN CALL PLOT
210 LINE XA.YA.X.Y,1 : REM LINIE ZEICHNEN VON ALTEN NACH
NEUEN KOORDINATEN
```

220 XA = X

230 YA = Y : REM WERTE MERKEN

240 GET A\$ : REM TASTE HOLEN

250 IF A\$="" THEN HIRES 6.7 : REM BEI "" LOESCHEN

260 CALL PLOT

## 16.2 PADDLE UND JOYSTICK

Wir kommen nun zu einer Art von Steuergeräten, die gerne Einsatz bei Spielen finden, aber auch für viele andere Dinge verwendet werden. Zunächst zu den Paddles:

Ihr CBM 64 besitzt insgesamt 4 sogenannte A/D-Wandler (Analog/Digital - Wandler). Dies sind elektronische Schaltungen, die es ermöglichen, einen analogen Wert, d.i. ein Wert, der keine Abstufungen besitzt (etwa die Temperatur) in einen digitalen, also mit Zahlen angebbaren Wert umzusetzen. Dieser digitale Wert ist Teil einer Skala von endlich vielen Werten (hier 256), mit denen man rechnen kann. Üblicherweise werden an diese A/D-Wandler, die ebenfalls über die zwei Controlports erreichbar sind, sogenannte Paddles angeschlossen. Dies sind Drehknöpfe (Drehpotentiometer), mit denen man Werte von 0-255 durch Drehung nach links oder rechts einstellen kann. Diese Werte lassen sich dann in einem geeigneten Programm verwerten. Sie können z.B. zur Steuerung verschiedener Parameter der Soundbefehle dienen und so schöne Effekte erzeugt werden.

Die Joysticks sind wohl die (für den CBM 64) am meisten verbreiteten Anschlußgeräte, da sie sich für besonders für Spiele ausgezeichnet eignen. Joysticks sind richtige Steuerknüppel, die man in alle Richtungen ziehen und drücken kann. In aller Regel haben Sie noch einen sogenannten Feuerknopf, der ebenfalls betätigt werden kann. Sie können nun abfragen, in welche Richtung der Joystick zeigt und ob der Feuerknopf gedrückt ist oder nicht. Entsprechend können Sie z.B. Objekte über den Bildschirm bewegen oder Schüsse abfeuern. So gut wie alle Spiele für den 64er werden mit diesen Knüppeln betrieben, die sich natürlich gleichfalls hervorragend zur Cursorsteuerung oder ähnlichen Dingen eignen.

## 16.2.1 POT

```
FORMAT      :POT (P)
PARAMETER   : - P: LEGT DIE PADDLE-
              NUMMER FEST
BEISPIEL    :X = POT (1)
FUNKTION    :ABLESEN DER PADDLESTELLUNG
```

### ERLÄUTERUNGEN

POT ist nun der entsprechende Befehl zur Feststellung des aktuellen Paddlewertes, also der Stellung des mit p angegebenen Paddles. POT wird ebenfalls wie eine normale Funktion in Rechnungen eingebaut. Die Handhabung brauchen wir nicht zu beschreiben, da wir darauf schon bei den PEN-Befehlen eingegangen sind.

### BEISPIEL:

```
100 REM #####
110 REM ##                ##
120 REM ## POT-BEISPIEL-1 ##
130 REM ##                ##
140 REM #####
150 REM
160 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
170 P = POT(0) : REM PADDLEWERT AUSLESEN
180 PRINT P
190 GOTO 170
```

-----

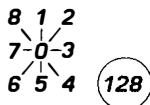
```
100 REM #####
110 REM ##                ##
120 REM ## POT-BEISPIEL-2 ##
130 REM ##                ##
140 REM #####
150 REM
160 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
170 P0 = POT(0) / 6.4 : REM PADDLEWERT 0 AUSLESEN
180 P1 = POT(1) / 10.3 : REM PADDLEWERT 1 AUSLESEN
190 PRINT AT(P0,P1) "*" : REM AN ANGESPROCHENER STELLE
```

16.2.2 JOY

FORMAT :JOY  
PARAMETER : ----  
BEISPIEL :J = JOY  
FUNKTION :ABLESEN DER JOYSTICKSTELLUNG

ERLÄUTERUNGEN:

Auf eben die gleiche Weise, wie schon bei den vorherigen Befehlen, läßt sich mittels JOY herausfinden, in welche Richtung der Joystick gedrückt wurde. Leider läßt sich hiermit lediglich der Joystick in Port 2 bestimmen. Port 1 wurde wahrscheinlich aus dem Grunde nicht übernommen, da gewisse Stellungen des Joysticks hier Buchstaben und andere Zeichen an den Rechner schicken. Der Befehl nimmt für jede Richtung einen bestimmten Wert an, der im Folgenden beschrieben ist:



Beispiel: steht der Joystick in Hoch-Stellung, so erhält die Funktion den Wert 1. Ist er schräg links nach oben gedrückt, so ist 8 das Ergebnis. Wenn Sie gleichzeitig den Feuerknopf betätigen, so wird zu diesen Werten die Zahl 128 hinzuaddiert. Zur Funktion des Joysticks sehen Sie sich bitte die Einleitung zu diesem Abschnitt an.

BEISPIEL:

```
100 REM #####  
110 REM ##           ##  
120 REM ## JOY-BEISPIEL ##  
130 REM ##           ##  
140 REM #####
```

```
150 REM
160 IF JOY=1 OR JOY=129 THEN PRINT "HOCH"
170 IF JOY=3 OR JOY=131 THEN PRINT "RECHTS"
180 IF JOY=5 OR JOY=133 THEN PRINT "UNTEN"
190 IF JOY=7 OR JOY=135 THEN PRINT "LINKS"
200 IF JOY>=128 THEN PRINT "FEUERKNOPF"
210 GOTO 160
```



## 17. KAPITEL

### ANHANG

#### 17.1 LÖSUNGEN - TESTS

Im folgenden seien hier die Lösungen zu den Testaufgaben in den einzelnen Kapiteln angeführt.

#### ABSCHNITT LÖSUNGEN

---

2.1.3	1) A	2) C	3) C	4) C	
2.2.4	1) A	2) A	3) B,C	4) B,C	
2.3.5	1) B	2) A	3) C	4) B	
	5) B	6) B			
3.1.6	1) C	2) D	3) A,B	4) A	5) B
3.2.5	1) A	2) D	3) C	4) B	5) C
5.1.5	1) B	2) B	3) B	4) A	5) D
5.2.6	1) A,B	2) A	3) B	4) B	5) B
6.3	1) B	2) B	3) A,B,C		
7.1.5	1) B	2) A	3) B	4) C	5) B
7.2.3	1) C	2) A,C	3) A	4) B,D	
8.5	1) C	2) A,C	3) D	4) C	5) C
	6) C	7) B	8) A	9) C	
9.1.6	1) B	2) B,D,F	3) C,D		
9.2.5	1) C	2) C			
9.3.4	1) B,D	2) C			
9.4.6	1) C	2) B,C	3) C		
9.5.5	1) D	2) A,D			
10.3	1) D	2) C	3) B	4) B	
11.2.3	1) A,B	2) C	3) B		
12.2.7	1) B	2) C	3) B,C		
12.3.10	1) C,E,F	2) B	3) A,B,C		
12.4.3	1) A,C	2) C	3) A,D		
12.5.3	1) B,C	2) B			
13.5	1) B,C,D	2) B			
14.1.5	1) B,C	2) D	3) B,D		
14.2.6	1) B,D	2) A			
15.7	1) A	2) B	3) B,D,E		

## 17.2 FARBTABELLE

Bei den verschiedensten Graphikbefehlen müssen Sie die Farbe, in der Sie zeichnen wollen, in Form eines Farbcodes angeben. Jeder der 16 wählbaren Farben ist ein solcher Code zugeordnet. Sie sind im Folgenden aufgelistet. Gleichzeitig werden die Tastendruckkombinationen genannt, die Sie von Hand aus eingeben können, um die Farbe des Textes während eines PRINT - Statements zu verändern. Dabei steht k für die Kontroll- und c für die Commodoretaste, die jeweils gleichzeitig mit der dahinter angegebenen Taste gedrückt werden müssen:

<i>CODE..FARBE.....TASTEN</i>	<i>----</i>	<i>CODE..FARBE.....TASTENO</i>
<i>0...SCHWARZ.....K 1</i>		<i>8...ORANGE.....C 1</i>
<i>1...WEIß.....K 2</i>		<i>9...BRAUN.....C 2</i>
<i>2...ROT.....K 3</i>		<i>10...HELLROT.....C 3</i>
<i>3...TÜRKIS.....K 4</i>		<i>11...GRAU.1.....C 4</i>
<i>4...VIOLETT.....K 5</i>		<i>12...GRAU.2.....C 5</i>
<i>5...GRÜN.....K 6</i>		<i>13...HELLGRÜN.....C 6</i>
<i>6...BLAU.....K 7</i>		<i>14...HELLBLAU.....C 7</i>
<i>7...GELB.....K 8</i>		<i>15...GRAU.3.....C 8</i>

## **17.3 FEHLERMELDUNGEN**

Nachfolgend sind sämtliche Fehlermeldungen des Simon's Basic aufgeführt, die noch zu den alten, Ihnen bekannten Ausgaben des Commodore - Betriebssystems hinzukommen. Sie fehlen leider in dem zum Simon's Basic mitgelieferten Handbuch völlig.

### **PROC NOT FOUND**

wird gegeben, wenn eine symbolische Sprungadresse angesteuert wurde, die Sie nicht in Ihrem Programm als eine solche definiert haben. Diese Fehlermeldung entspricht etwa der Fehlermeldung UNDEF'D STATEMENT ERROR, die bekanntlich beim Ansprung nicht vorhandener Zeilennummern ausgegeben wird.

### **INSERT TOO LARGE**

entsteht, wenn Sie bei den String-Befehlen INST und INSERT (Kapitel 8.1/8.2) entweder eine Zeichenposition angeben, die nicht existiert oder der einzusetzende String größer ist als der Hauptstring.

### **STRING TOO LARGE**

ist diejenige Fehlermeldung, die ausgegeben wird, wenn der resultierende String bei den Befehlen INST und INSERT die Länge 255 überschreitet und hat damit die gleiche Funktion wie Meldung STRING TOO LONG ERROR des normalen Basic.

### **NOT BINARY CHAR**

signalisiert Ihnen, daß Sie bei dem Befehl % zur Umwandlung einer dezimalen in eine duale Zahl irgendwo statt 1 oder 0 ein anderes Zeichen eingegeben haben, oder die Binärzahl zu lang ist. Diese Meldung taucht ebenfalls bei dem Befehl WAVE auf, wenn Sie z.B. das Zeichen % vor die anzugebende Binärzahl stellen, daß hier entbehrt werden kann (und muß).

### **NOT HEX CHAR**

Diese Fehlermeldung entspricht der soeben besprochenen und wird gesendet, wenn Sie nach dem Befehl \$ keine Hexadezimalziffer eingegeben haben.

### **END PROC WITHOUT EXEC**

entspricht voll und ganz der Fehlermeldung RETURN WITHOUT GOSUB ERROR und resultiert aus einem END PROC, das nicht vorher durch ein EXEC angesteuert wurde. Es wird also versucht eine nicht existierende Unterprogrammebene zu schließen

### **END LOOP WITHOUT LOOP**

wird analog der Meldung NEXT WITHOUT FOR ERROR ausgegeben, also wenn im Laufe des Programms eine END LOOP gesendet wird, das nicht vorher durch LOOP eröffnet wurde.

### **UNTIL WITHOUT REPEAT**

Diese Meldung erscheint ebenfalls analog zu NEXT WITHOUT FOR ERROR, gilt jedoch für REPEAT-Schleifen

### **STACK TOO LARGE**

zeigt eine zu große Verschachtelung in Ihrem Programm an (ähnlich OUT OF MEMORY ERROR). Sie sollten ein paar Unterprogramm- oder Schleifenebenen entfernen.

### **BAD CHAR FOR A MOB**

signalisiert ein falsches, zuviele oder zu wenig Zeichen innerhalb einer Sprite- oder Zeichendefinition. Sie taucht gleichfalls auf, wenn Sie zu wenig Zeilen für die Definition verwendet haben oder zwischen dem DESIGN-Befehl und der Definition eine Basiczeile steht.

### **BAD MODE**

ist eine Fehlermeldung, die ebenso universal verwendet wird wie ILLEGAL QUANTITY ERROR und ebenfalls eine Bereichsüberschreitung bei Simon's Basic - Befehlen anzeigt.

## 17.4 ALPHABETISCHE BEFEHLSÜBERSICHT

\$	93	EXOR	89	OPTION	35
%	92	FCHR	152	OUT	62
ANGL	256	FCOL	156	PAGE	35
ARC	250	FETCH	180	PAINT	261
AT	120	FILL	159	PENX	366
AUTO	22	FIND	37	PENY	367
BFLASH	131	FLASH	127	PLACE	99
BFLASH 0	133	FRAC	88	PLAY	360
BLOCK	242	GLOBAL	83	PLOT	226
CALL	77	HI COL	219	POT	370
CENTRE	116	HIRES	204	PROC	75
CGOTO	79	HRDCOPY	193	RCOMP	68
CHAR	279	IF/THEN/ELSE	67	REC	237
CHECK	335	INSERT	95	REPEAT/UNTIL	70
CIRCLE	245	INST	97	RENUMBER	24
CMOB	319	INV	163	RESET	182
COLD	49	JOY	371	RETRACE	45
COPY	193	KEY	16	RIGHT	171
CSET 0/1	302	LEFT	168	RLOCMOB	329
CSET 2	213	LIN	122	ROT	273
DELAY	36	LINE	233	SCRLD	191
DESIGN 0/1	310	LOCAL	82	SCRSV	189
DESIGN 2	296	LOOP/EXIT IF/ END LOOP	72	SECURE 0	65
DETECT	333	LOW COL	215	TEST	228
DIR	187	MEM	292	TEXT	283
DISAPA	64	MERGE	28	TRACE	43
DISK	185	MMOB	321	UP	174
DISPLAY	19	MOB OFF	325	USE	117
DIV	88	MOB SET	315	VOL	345
DOWN	176	MOD	85	WAVE	348
DRAW	267	MOVE	162		
DUMP	47	MULTI	207		
DUP	109	MUSIC	356		
END PROC	75	NO ERROR	55		
ENVELOPE	353	OFF	129		
ERRLN	52	OLD	50		
ERRN	52	ON ERROR	52		
EXEC	78				

## 17.5 SIMON'S BASIC DATEN

- UMFASST 16 KBYTE
- BELEGT 8 KBYTE BASIC-SPEICHER (\$8000 BIS \$9FFF)
- BELEGT TEILWEISE SPEICHERBEREICH UNTER DEM BASIC-ROM (\$A000-\$BFFF)
- BELEGT GROBE TEILE ZWISCHEN \$C000 BIS \$CFFF (PAINT, KEY)
- SCHALTET TEILWEISE AUF BASIC-ROM UND BASIC-RAM
- KEIN RUN/STOP SCHUTZ MIT POKE 788.52 UND KEIN RUN/STOP-RESTORE SCHUTZ MIT POKE 792.193 MÖGLICH, DA INTERRUPT-VEKTOR "VERBOGEN"
- LÄUFT MIT DISKOMAT (SUPERTWIN UND DISKMONITOR), WENN BEI SUPERTWIN DER MODUS "SUPERTWIN MIT IEC-BUS AUSGEWÄHLT WIRD
- LÄUFT NICHT MIT DER 80-ZEICHEN KARTE MAXI 64
- ERMÖGLICHT KEINE HARDCOPY MIT COMMODORE VC-1526 UND MPS-802

# DATA BECKER'S NEUE BÜCHER UND PROGRAMME FÜR COMMODORE

## Spickzettel ade.

Ein neues DATA BECKER BUCH, das den Einsatz des COMMODORE 64 in der Schule entscheidend mitprägen dürfte, wurde von Professor VOß geschrieben. Besonders für Schüler der Mittel- und Oberstufe geschrieben, enthält das Buch viele interessante Problemlösungs- und Lernprogramme, die besonders ausführlich und leicht verständlich beschrieben sind. Sie ermöglichen ein intensives und anregendes Lernen, unter anderem mit folgenden Themen: Satz des Pythagoras, quadratische Gleichungen, geometrische Reihen, Pendelbewegungen, mechanische Hebel, Molekülbildung, exponentielles Wachstum, Vokabeln lernen, unregelmäßige Verben, Zinseszinsrechnung. Ein kurzer Überblick über die Grundlagen der EDV, eine knappe Wiederholung der wichtigsten BASIC-Elemente und eine Einführung in die Grundzüge der Problemanalyse vervollständigen das Ganze. Mit diesem Buch machen die Hausaufgaben wieder Spaß!

DAS SCHULBUCH ZUM COMMODORE 64, 1984, über 300 Seiten, DM 49,-

## Tempo!

MASCHINENSPRACHE FÜR FORTGESCHRITTENE ist bereits das zweite Buch von Lothar Engelsch zum Thema Maschinenprogrammierung mit dem COMMODORE 64. Hier wird von der Problemanalyse bis zum Maschinen-sprachealgorithmus in die Grundlagen der professionellen Maschinen-sprache-programmierung eingeführt. In diesem Buch finden Sie unter anderem folgende Themen behandelt: Problemlösungen in Maschinensprache, Programmierung von Interruptroutinen, Interruptquellen beim COMMODORE 64, Interrupts durch CIA's und Videocontroller, Programmierung der Ein-Ausgabe-Bausteine, die CIA's des COMMODORE 64, Timer, Echtzeituhr, parallele und serielle Ein-/Ausgabe, BASIC-Erweiterungen, Programmierung eigener BASIC-Befehle und -Funktionen, Möglichkeiten zur Einbindung ins Betriebssystem sowie viele weitere Tips & Tricks zur Maschinenprogrammierung. Dieses Buch sollte jeder haben, der wirklich intensiv mit der Maschinensprache des COMMODORE 64 arbeiten will.

MASCHINENSPRACHE FÜR FORTGESCHRITTENE, 1984, ca. 200 Seiten, DM 39,-



## Macht Druck.

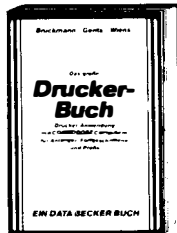
DAS GROSSE DRUCKERBUCH für Drucker-Anwender mit COMMODORE-Computern ist endlich da! Es enthält eine riesige Sammlung von Tips & Tricks, Programmlistings und Hardwareinformationen. Rolf Brückmann und Klaus Gerits beschäftigen sich mit Sekundäradressen, Anschluß einer Schreibmaschine am Userport, Druckerschnittstellen (Centronics, V24, IEC-Bus), hochauflösender Grafik, Text- und Grafikhardcopy, Grafik mit Standardzeichensatz, formatierter Datenausgabe, Plakatschrift, Textverarbeitung und vieles mehr. Zusätzlich wird das Betriebssystem des MPS8012erlegt, mit Prozessorbeschreibung (8035), Blockschaltbild und einem ausführlich kommentierten ROM-Listing. Thomas Wiens schrieb den Teil über die Programmierung des Plotters VC-1520: Handhabung des Plotters, Programmierung von Sonderzeichen, Funktionendarstellung, Kuchen und Säulendiagramme, Entwurf dreidimensionaler Gegenstände. Natürlich wieder viele interessante Listings. Unentbehrlich für jeden, der einen COMMODORE 64 oder VC-20 und einen Drucker besitzt.

DAS GROSSE DRUCKERBUCH, 1984, über 300 Seiten, DM 49,-

## Tausend- sassa.

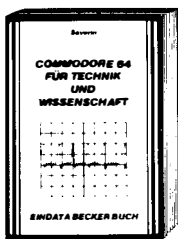
Fast alles, was man mit dem COMMODORE 64 machen kann, ist in diesem Buch ausführlich beschrieben. Es ist nicht nur spannend zu lesen wie ein Roman, sondern enthält neben nützlichen Programmlistings vor allem viele, viele Anwendungsmöglichkeiten des C64. Dabei wurde besonderer Wert darauf gelegt, daß das Buch auch für Laien leicht verständlich ist. Eine Auswahl aus der Themenvielfalt: Gedichte vom Computer, Einladung zur Party, Diplomarbeit - professionell gestaltet, individuelle Werbeflyer, Autokosten im Griff, Baukostenberechnung, Taschenrechner, Rezeptkartei, Lagerliste, persönliches Gesundheitsarchiv, Diätplan elektronisch, intelligentes Wörterbuch, kleine Notenschule, CAD für Handarbeit, Routenoptimierung, Schaufensterwerbung, Strategiespiele. Teilweise sind Programmlistings fertig zum Eintippen enthalten, soweit sich die „Rezepte“ auf 1-2 Seiten realisieren ließen. Wenn Sie bisher nicht immer wußten, was Sie mit Ihrem 64er alles anfangen sollten, nach dem Lesen des IDEENBUCHES wissen Sie's bestimmt!

DAS IDEENBUCH ZUM COMMODORE 64, 1984, über 200 Seiten, DM 29,-



## Prof. 64.

Ein faszinierendes Buch, um in die Welt der Wissenschaft einzusteigen, hat Rainer Severin geschrieben. Zunächst werden Variablentypen, Rechengenauigkeit und nützliche POKE-Adressen des COMMODORE 64 bezüglich den Anforderungen wissenschaftlicher Probleme analysiert. Verschiedene Sortieralgorithmen wie Bubble, Quick und Shell-Sort werden miteinander verglichen. Die Programmbeispiele aus der Mathematik nehmen dabei eine zentrale Stelle im Buch ein: Nullstellen nach Newton, numerische Ableitung mit dem Differenzenquotienten, lineare und nichtlineare Regression, Chi-Quadrat-Verteilung und Anpassungstest, Fourieranalyse und -synthese, Skalar-, Vektor- und Spatprodukt, ein Programmpaket zur Matrizenrechnung für Inversion, Eigenwerte und vieles weitere mehr. Programme aus der Chemie (Periodensystem), Physik, Biologie (Schadstoffe in Gewässern – Erfassung der Meßwerte), Astronomie (Planetenpositionen) und Technik (Berechnung komplexer Netzwerke, Platinenlayout am Bildschirm) und viele weitere Softwarelistings zeigen die riesigen Möglichkeiten auf, die der Computer in Wissenschaft und Technik hat.



COMMODORE 64 FÜR TECHNIK UND WISSENSCHAFT, 1984, über 200 Seiten, DM 49,-

## Grundkurs.

Das neue BASIC-Trainingsbuch zum C-64 ist eine ausführliche, didaktisch gut geschriebene Einführung in das CBM BASIC V2. Alle Befehle werden ausführlich erläutert. Dieses Buch geht aber über eine reine Befehlsbeschreibung hinaus, es wird eine fundierte Einführung in die Programmierung gegeben. Von der Problemanalyse bis zum fertigen Algorithmus lernt man das Entwerfen eines Programmes und den Entwurf von Datenflußplänen. ASCII-Code und verschiedene Zahlensysteme wie hexadezimal, binär und dezimal sind nach der Lektüre des Buches ohne Fremdworte mehr. Die Programmierung von Schleifen, Sprüngen, bedingten Sprüngen lernt man leicht durch *learning by doing*. So enthält das Trainingsbuch viele Aufgaben, Übungen und unzählige Beispiele. Den Schluß des Buches bildet eine Einführung ins professionelle Programmieren, in der es um mehrdimensionale Felder, Menuesteuerung und Unterprogrammtechnik geht. Endlich ein Buch, das Ihnen wirklich hilft, solide und sicher BASIC zu lernen.



BASIC TRAININGSBUCH ZUM COMMODORE 64, 1984, ca. 250 Seiten, DM 39,-

## Sang und Klang!

Der COMMODORE 64 ist ein Musikgenie. DAS MUSIKBUCH hilft Ihnen, die riesigen Klangmöglichkeiten des C64 zu nutzen. Die Themenbreite reicht von einer Einführung in die Computermusik über die Erklärung der Hardwaregrundlagen des COMMODORE 64 und die Programmierung in BASIC bis hin zur fortgeschrittenen Musikprogrammierung in Maschinensprache. Einiges aus dem Inhalt: Soundregister des COMMODORE 64, Gate-Signal, Programmierung der "ADSR"-Werte, Synchronisation und Ring-Modulation, Counterprinzip, lineare und nichtlineare Musikprogrammierung, Frequenzmodulation, Interrupts in der Musikprogrammierung und vieles mehr. Zahlreiche Beispielprogramme, komplette Songs und nützliche Routinen ergänzen den Text. Geschrieben wurde das Buch von Thomas Dachsel, dem Autor der weltbekannten Musikprogramme Synthimath und Synthesound. Erschließen Sie sich die Welt des Sounds und der Computermusik mit dem Musikbuch zum C-64!



DAS MUSIKBUCH ZUM COMMODORE 64, über 200 Seiten, DM 39,-

## Nützlich.

Das Trainingsbuch zu MULTIPLAN bietet eine gute Einführung in die Grundlagen der Tabellenkalkulation. Dabei wird großer Wert auf ein möglichst schnelles Einarbeiten in die wichtigsten Befehle gelegt, so daß man bald sicher mit MULTIPLAN arbeiten kann, ob nun auf dem COMMODORE 64 oder einem anderen Rechner. Am Ende wird man in der Lage sein, den umfangreichen Befehlssatz von MULTIPLAN auch kommerziell zu nutzen. Übungen am Ende jedes Kapitels sorgen dafür, daß man das Gelernte lange behält. Grundlage des Buches sind viele Seminare, die der Autor zu MULTIPLAN konzipiert und erfolgreich durchgeführt hat.



DAS TRAININGSBUCH ZU MULTIPLAN, 1984, ca. 250 Seiten, DM 49,-

## Für Tüftler.

Ein hochinteressantes Buch für Hobbyelektroniker hat Rolf Brückmann vorgelegt. Er ist ein engagierter Techniker, für den der Computer Hobby und Beruf zur gleichen Zeit ist. Vor allem aber kennt er den C-64 in- und auswendig. So werden einführend die Schnittstellen des COMMODORE 64 detailliert beschrieben und kurz die Funktionsweise der CIAs 6526 erläutert. Hauptteil des Buches sind die Beschreibungen der vielfältigen Einsatzmöglichkeiten des COMMODORE 64. Die vielen Schaltungen, von Rolf Brückmann alle selbst





entwickelt, sind jeweils umfangreich dokumentiert und leichtverständlich erklärt. Die Reihe der hier ausführlich behandelten Anwendungen mit dem COMMODORE 64 ist äußerst umfangreich: Motorsteuerung, Stoppuhr mit Lichtschranke, Lichtorgel, A/D-Wandler, Spannungsmessung, Temperaturmessung und vieles mehr. Dazu kommen noch eine Reihe kompletter Schaltungen zum Selberbauen, wie ein EPROM Programmiergerät für den C-64, eine EPROM-Karte, ein Frequenzzähler und Sprachein-/ausgabe (!). Zusätzlich sind jeweils Schaltplan, Softwareerstellung und zu einigen Schaltungen sogar zusätzlich Platinenlayouts vorhanden.

DER COMMODORE 64 UND DER REST DER WELT, 1984, ca. 220 Seiten, DM 49,-

## Computerkünstler.

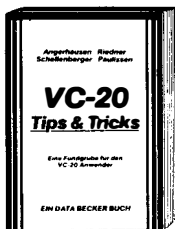
Das Grafikbuch zum COMMODORE 64 Buch aus der Bestseller-Serie von DATA BECKER stammt aus der Feder von Axel Plenge. Es geht weit über die reine Hardware-Beschreibung der Grafikeigenschaften des C-64 hinaus. Der Inhalt reicht von den Grundlagen der Grafikprogrammierung bis zum Computer Aided Design. Es ist ein Buch für alle, die mit ihrem C-64 kreativ tätig sein wollen. Themen sind z. B.: Zeichensatzprogrammierung, bewegte Sprites, High-Resolution, Multicolor-Grafik, Lightpenanwendungen, Betriebsarten des VIC, Verschieben der Bildschirmspeicher, IRQ-Handhabung, 3-Dimensionale Grafik, Projektionen, Kurven, Balken- und Kuchendiagramme, Laufschriften, Animation, bewegte Bilder. Viele Programmlistings und Beispiele sind selbstverständlich. Das COMMODORE-BASIC V2 unterstützt die herausragenden Grafikeigenschaften des C-64 bekanntlich kaum. Hier helfen die vielen Beispielprogramme in diesem Buch weiter, die die faszinierende Welt der Computergrafik jedermann zugänglich machen. Kompetent ist der Autor dazu wie kaum ein anderer, schließlich hat er das äußerst leistungsfähige Programm SUPERGRAFIK geschrieben.

DAS GRAFIKBUCH ZUM COMMODORE 64, 1984, 295 Seiten, DM 39,-



## Vielfalt.

Auf dem neuesten Stand ist VC-20 TIPS & TRICKS von Dirk Paulssen gebracht worden, der über hundert Seiten hinzufügte. Bisher schon enthalten waren Informationen über Speicheraufbau des VC-20 und die Erweiterungsmöglichkeiten, ein Graffkapitel über programmierbare Zeichen, Laufschrift und die Supererweiterung. Stark erweitert wurde der Abschnitt über POKES und andere nützliche Routinen. Obesum die Programmierung der Funktionstasten, Programme die sich selber starten, „Maus“-Simulation mit dem Joystick oder die Änderung von Speicherbereichen geht, man ist immer wieder über die Fülle der Möglichkeiten erstaunt. Der Clou dieses



Buches sind aber die vielen Programmlistings. Die BASIC-Erweiterungen allein stellen schon ein erstklassiges Toolkit dar: APPEND (Anhängen von Programmen, AUTO (automatische Zellenummerierung), BASIC-Befehle auf Tastendruck, PRINT POSITION, UNNEW, Stringsgrößer als 88 Zeichen einlesen und vieles mehr. Die Bandbreite reicht von Spielen wie Goldgräber oder Starshooter bis zu nützlichen Programmen wie Cassetteninhaltsverzeichnis und -katalog mit automatischem Suchen nach Dateien und einem Terminkalender. Für den VC-20 Anwender ist dieser 324 Seiten-Wälzer eine wahre Fundgrube, in der es immer etwas neues zu entdecken gibt.

VC-20TIPS & TRICKS, 3. erweiterte und überarbeitete Auflage, 1984, 324 Seiten, DM 49,-

## Interessant.

Einen guten Einstieg in PASCAL bietet dieses Trainingsbuch. Es gibt eine leichtverständliche Einführung, sowohl in UCSD-PASCAL wie auch in PASCAL64, wobei allerdings EDV- und BASIC-Grundkenntnisse vorausgesetzt werden. Der Autor, Ottmar Korbmacher, ist Student der Mathematik. Ihm gelingt es, in einem sprachlich aufgelockerten Stil mit vielen interessanten Beispielprogrammen, dem Leser Programmstrukturen, Ein-/Ausgabe, Arithmetik und Funktionen, Prozeduren und Rekursionen, Sets, Files und Records näherzubringen. Die Übungsaufgaben am Ende jeden Kapitels helfen dabei, das Gelernte zu vertiefen. Ein Anhang mit allen PASCAL-Schlüsselwörtern, der ansich schon ein umfangreiches Lexikon darstellt, macht das Buch für jeden PASCAL-Anwender interessant.

DAS TRAININGSBUCH ZU PASCAL, 1984, ca. 250 Seiten, DM 39,-



## Bewährt.

Die bereits dritte Auflage von VC-20 INTERN ist wieder erheblich erweitert worden. Das Buch beschäftigt sich ausführlich mit der Technik und dem Betriebssystem des VC-20. Dazu gehört natürlich zuerst einmal ein ausführlich dokumentiertes ROM-Listing. Dazu gehört auch die Belegung der Zeropage, dem wichtigsten Speicherbereich für den 6502-Prozessor, eine übersichtliche Auflistung der Adressen aller Betriebssystemroutinen, ihrer Bedeutung und ihrer Übergabeparameter. Dies ermöglicht dem Programmierer endlich, den VC-20 von Maschinensprache aus sinnvoll einzusetzen. Denn warum Routinen, die bereits vorhanden sind, noch einmal schreiben? Weiterer Inhalt: Einführung in die Maschinensprache – Maschinensprachemonitor, Assembler, Disassembler – Verbindung von Maschinensprache- und BASIC-Programmen – Beschreibung der wichtigen IC's des VC-20 – Blockschaltbild – drei Original COMMODORE-Schaltpläne. Das Buch braucht jeder der sich intensiv mit der Maschinenspracheprogrammierung des VC-20 auseinandersetzen möchte.

VC-20 INTERN, 3. Auflage, 1984, ca. 230 Seiten, DM 49,-



## Starthilfe!

Das sollte Ihr erstes Buch zum COMMODORE 64 sein: 64 FÜR EINSTEIGER ist eine sehr leicht verständliche Einführung in Handhabung, Einsatz, Ausbaumöglichkeiten und Programmierung des COMMODORE 64, die keinerlei Vorkenntnisse voraussetzt. Sie reicht vom Anschluß des Geräts über die Erklärung der einzelnen Tasten und Funktionen sowie die Peripheriegeräte und ihre Bedienung bis zum ersten Befehl. Schritt für Schritt führt das Buch Sie in die Programmiersprache BASIC ein, wobei Sie nach und nach eine komplette Adressverwaltung erstellen, die Sie anschließend nutzen können. Zahlreiche Abbildungen und Bildschirmfotos ergänzen den Text. Viele Anwendungsbeispiele geben nützliche Anregungen zum sinnvollen Einsatz des COMMODORE 64. Das Buch ist sowohl als Einführung als auch als Orientierung vor dem 64er Kauf gut geeignet.



64 FÜR EINSTEIGER, 1984, ca. 200 Seiten, DM 29,-

## Von A bis Z.

So etwas haben Sie gesucht: Umfassendes Nachschlagewerk zum COMMODORE 64 und seiner Programmierung. Allgemeines Computerlexikon mit Fachwissen von A-Z und Fachwörterbuch mit Übersetzungen wichtiger englischer Fachbegriffe - das DATA BECKER LEXIKON ZUM COMMODORE 64 stellt praktisch drei Bücher in einem dar. Es enthält eine unglaubliche Vielfalt an Informationen und dient so zugleich als kompetentes Nachschlagewerk und als unentbehrliches Arbeitsmittel. Viele Abbildungen und Beispiele ergänzen den Text. Ein Muß für jeden COMMODORE 64 Anwender!



DAS DATA BECKER LEXIKON ZUM COMMODORE 64, 1984, 354 Seiten, DM 49,-

## Fundgrube.

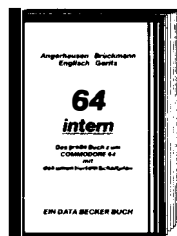
64 TIPS & TRICKS ist eine hochinteressante Sammlung von Anregungen zur fortgeschrittenen Programmierung des COMMODORE 64, POKE's und andere nützliche Routinen, interessanten Programmen sowie interessanten Programmertips & tricks. Aus dem Inhalt: 3D-Graphik in BASIC - Farbige Balkengraphik - Definition eines eigenen Zeichensatzes - Tastaturbelegung und ihre Änderung - Dateneingabe mit Komfort - Simulation der Maus mit einem Joystick - BASIC für Fortgeschrittene - C-64 spricht deutsch - CP/M auf dem COMMODORE 64 - Druckeranschluß über den USER-Port - Datenübertragung von und zu anderen Rechnern - Expansion-Port - Synthesizer in Stereo - Retten einer nicht ordnungsgemäß geschlossenen Datei - Erzeugen einer BASIC-Zeile in BASIC - Kassettenpuffer als Datenspeicher - Sortieren von Stringfelder - Multitasking auf dem COMMODORE 64 - POKE's und die Zeropage - GOTO, GOSUB und RESTORE mit berechneten Zeilennummern, INSTR und STRING-Funktion - Repeat-Funktion für alle



Tasten - und vieles andere mehr. Alle Maschinenprogramme mit BASIC-Ladeprogrammen. 64 Tips & Tricks ist eine echte Fundgrube für jeden COMMODORE 64 Anwender. Schon über 65000mal verkauft! 64 TIPS & TRICKS, 1984, über 300 Seiten, DM 49,-

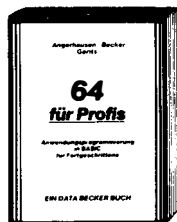
## Know-how!

350 Seiten dick ist die 4. erweiterte und überarbeitete Auflage von 64 INTERN geworden. Das bereits über 65000mal verkaufte Standardwerk bietet jetzt noch mehr Informationen. Hinzugekommen ist ein Kapitel über den IEC-Bus und viele, viele Ergänzungen, die sich im Laufe der Zeit angesammelt haben. Ebenfalls überarbeitet und noch ausführlicher ist jetzt die Dokumentation des ROM-Listings. Weitere Themen: genaue Beschreibung des Sound- und Video-Controllers mit vielen Hinweisen zur Programmierung von Sound und Grafik, der Ein-/Ausgabesteuerung (CIAS), BASIC-Erweiterungen (RENEW, HARDCOPY, PRINTUSING), Hinweise zur Maschinenprogrammierung wie Nutzung der E/A-Routinen des Betriebssystems, Programmierung der Schnittstelle RS 232, ein Vergleich VC20 - C-64 - CBM zur Umsetzung von Programmen. Dies und viele weitere Informationen machen das umfangreiche Werk zu einem unentbehrlichen Arbeitsmittel für jeden, der sich ernsthaft mit Betriebssystem und Technik des C-64 auseinandersetzen will. Zum professionellen Gehalt des Buches tragen auch zwei Original-COMMODORE-Schaltpläne zum Ausklappen und zahlreiche ausführlich beschriebene und dokumentierte Fotos, Schaltbilder und Blockdiagramme bei. 64 INTERN 4. überarbeitete und erweiterte Auflage, 1984, ca. 350 Seiten, DM 69,-



## Erfolgreich.

64 für Profis zeigt, wie man erfolgreich Anwendungsprobleme in BASIC löst und verrät die Erfolgsgeheimnisse der Programmierprofis. Vom Programmwurf über Menüsteuerung, Maskenaufbau, Parametrisierung, Datenzugriff und Druckausgabe bis hin zur guten Dokumentation wird anschaulich mit vielen Beispielen dargestellt wie Profi-Programmierung vor sich geht. Besonders stolz sind wir auf die völlig neuartige Datenzugriffsmethode QUISAM, die in diesem Buch zum ersten Mal vorgestellt wird. QUISAM erlaubt eine beliebige Datenanzahl, die dynamisch mit der Eingabe der Daten wächst. Eine lauffertige Literaturstellenverwaltung veranschaulicht die Arbeitsweise von QUISAM. Neben diesem Programm finden Sie noch weitere Programme zur Lager- und Adressverwaltung, Textverarbeitung und einen Reportgenerator. Alle diese Programme sind mit Variablenliste versehen und ausführlich beschrieben. Damit sind diese für Ihre Erweiterungen offen und können von Ihnen an Ihre persönlichen Bedürfnisse angepaßt werden. Steigen Sie in die Welt der Programmierprofis ein.



64 FÜR PROFIS, 2. Auflage, 1984, ca. 300 Seiten, DM 49,-

## Rundum gut!

Endlich ein Buch, das Ihnen ausführlich und verständlich die Arbeit mit der Floppy VC-1541 erklärt. Das große Floppybuch ist für Anfänger, Fortgeschrittene und Profis gleichermaßen interessant. Sein Inhalt reicht von der Programmspeicherung bis zum DOS-Zugriff, von der sequentiellen Datenspeicherung bis zum Direktzugriff, von der technischen Beschreibung bis zum ausführlich dokumentierten DOS-Listing, von den Systembefehlen bis zur detaillierten Beschreibung der Programme auf der Test-Demo-Diskette. Exakt beschriebene Beispiel- und Hilfsprogramme ergänzen dieses neue Superbuch. Aus dem Inhalt: Spielern von Programmen – Floppy-Systembefehle – Sequentielle Datenspeicherung – relative Datenspeicherung – Fehlermeldungen und ihre Ursachen – Direktzugriff – DOS-Listing der VC-1541 – BASIC-Erweiterungen und Programme – Overlay-technik – Diskmonitor – IEC-Bus und serieller Bus – Vergleich mit den großen CBM-Floppies. Ein Muß für jeden Floppy-Anwender! Bereits über 45.000mal verkauft.

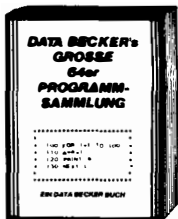
DAS GROSSE FLOPPY-BUCH, 2. überarbeitete Auflage, 1984, ca. 320 Seiten, DM 49,-



## Füttern erwünscht!

Diese beliebte umfangreiche Programmsammlung hat es in sich. Über 50 Spitzenprogramme für den COMMODORE 64 aus den unterschiedlichsten Bereichen, von attraktiven Superspielen (Senso, Pengo, Master Mind, Seeschlacht, Poisson Square, Memory) über Grafik- und Soundprogramme (Fourier 64, Akustograph, Funktionsplotter) und mathematische Programme (Kurvendiskussion, Dreieck) sowie Utilities (SORT, RENUMBER, DISK INIT, MENUE) bis hin zu kompletten Anwendungsprogrammen wie „Videothek“, „File Manager“ und einer komfortablen Haushaltsbuchführung, in der fast professionell gebucht wird. Der Hit zu jedem Programm sind aktuelle Programmtips und Tricks der einzelnen Autoren zum Selbermachen. Also nicht nur abtippen, sondern auch dabei lernen und wichtige Anregungen für die eigene Programmierung sammeln.

DATA BECKER'S GROSSE 64er PROGRAMMSAMMLUNG, 1984, 250 Seiten, DM 49,-

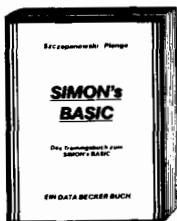


# Bestseller aus bester Hand

## BASIC-PLUS.

SIMON'S BASIC ist ein Hit – wenn man es richtig nutzen kann. Auf über 300 Seiten erklärt Ihnen das DATA BECKER Trainingsbuch detailliert den Umgang mit den über 100 Befehlen des SIMON'S BASIC. Alle Befehle werden ausführlich dargestellt, auch die, die nicht im Handbuch stehen! Natürlich zeigen wir auch die Macken des SIMON'S BASIC und geben wichtige Hinweise wie man diese umgeht. Natürlich enthält das Buch viele Beispielprogramme und viele interessante Programmiertricks. Weiterer Inhalt: Einführung in das CBM-BASIC 2.0 – Programmierhilfen – Fehlerbehandlung – Programmschutz – Programmstruktur – Variablen – Zahlenbehandlung – Eingabekontrolle – Ein-/Ausgabe Peripheriebefehle – Graphik – Zeichensatzstellung – Sprites – Musik – SIMON'S BASIC und die Verträglichkeit mit anderen Erweiterungen und Programmen. Dazu ein umfangreicher Anhang. Nach jedem Kapitel finden Sie Testaufgaben zum optimalen Selbststudium und zur Lernerfolgskontrolle.

DAS TRAININGSBUCH ZUM SIMON'S BASIC, 2. überarbeitete Auflage, 1984, ca. 380 Seiten, DM 49,-

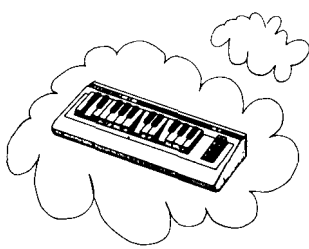


## Schrittmacher.

Eine leicht verständliche Einführung in die Maschinenspracheprogrammierung für alle, denen das C-64 BASIC nicht mehr ausreicht. Sie lernen Aufbau und Arbeitsweise des 6510-Mikroprozessors kennen und anwenden. Dabei werden die Analogien zu BASIC Ihnen beim Verständnis helfen. Ein weiteres Kapitel beschäftigt sich mit der Eingabe von Maschinenprogrammen. Dort erfahren Sie auch alles über Monitor-Programme sowie über Assembler. Zum einfachen und komfortablen Erstellen Ihrer eigenen Maschinensprache enthält das Buch einen kompletten ASSEMBLER, damit Sie gleich von Anfang an komfortabel und effektiv programmieren können. Weiterhin finden Sie dort einen DISASSEMBLER, mit dem Sie sich Ihre Maschinenprogramme oder die Routinen des BASIC-Interpreters und des BASIC-Betriebssystems ansehen können. Ein besonderer Clou ist ein in BASIC geschriebener Einzelschrittsimulator, mit dem Sie Ihre Programme schrittweise ausführen können. Dabei werden Sie nach jedem Schritt über Registerinhalte und Flags informiert und können den logischen Ablauf Ihres Programmes verfolgen. Eine unschätzbare Hilfe, besonders für den Anfänger. Als Beispielprogramm finden Sie ausführlich beschriebene Routinen zur Grafikprogrammierung und für BASIC-Erweiterungen. Natürlich sind alle Beispiele und Programme auf den C-64 zugeschnitten.

DAS MASCHINENSPRACHEBUCH ZUM COMMODORE 64, ca. 200 Seiten, DM 39,-





## SYNTHIMAT

SYNTHIMAT verwandelt Ihren COMMODORE 64 in einen professionellen, polyphonen, dreistimmigen Synthesizer, der in seinen unglaublich vielen Möglichkeiten großen Systemen kaum nachsteht.

### SYNTHIMAT In Stichworten:

drei Oszillatoren (VCOs) mit 7 Fußlagen und 8 Wellenformen – drei Hüllkurvengeneratoren (ADSRs) – ein Filter (VCF) mit 8 Betriebsarten und Resonanzregulierung – VCF mit Eingang für externe Signalquelle – ein Verstärker (VCA) – Ringmodulation mit allen drei VCOs – 8 softwaremäßig realisierte Oszillatoren (LFOs) – kräftiger Klang durch polyphones Spielen – zwei Manuale (Solo und Begleitung) – speichern von bis zu 256 Klangregistern – schneller Registerwechsel – speichern von 9 Registerdateien auf Diskette – „Bandaufnahme“ auf Diskette durch direktes Spielen – keine lästige Noteingabe – speichern von bis zu 9 „Bandaufnahmen“ je Diskette – integrierte 24 Stunden-Echtzeituhr – einstellbares PITCH-BENDING – farblich gekennzeichnete, übersichtlich angeordnete Module – umfangreiches Handbuch – läuft mit einem Diskettenlaufwerk – Diskettenprogramm.

DM 99,-

# NEU Superbase 64

Für viele ein Traum, für die meisten bisher zu teuer: die Rede ist von einer echten Datenbank für den 64er. SUPERBASE 64 füllt eine Lücke. Nicht allein die Kapazität, die verwaltet werden kann, bewegt sich in professionellen Regionen, die ausgeprägten Fähigkeiten des SUPERBASE 64 im Rechnen und Kalkulieren lassen dieses Paket beinahe als Rund-Um-Software erscheinen.

### SUPERBASE 64 In Stichworten:

maximale Datensatzlänge 1108 Zeichen, verteilt auf bis zu 4 Bildschirmseiten – bis zu 127 Felder pro Datensatz, wobei Textfelder bis zu 255 Zeichen lang sein können – insgesamt 15 Einzeldateien können zu einer SUPERBASE-Datenbank verknüpft werden – Speicherkapazität nur durch Diskette begrenzt – umfangreiche Auswertungsmöglichkeiten und komfortabler Report-Generator – Kalkulationsmöglichkeiten und Rechnen – Import: (Einlesen von externen Daten) und Export: (Ausgabe von SUPERBASE Dateien als sequentielle Datei) Funktionen ermöglichen Datenaustausch mit anderen Programmen – durch leistungsfähige, eigene Datenbanksprache auch als kompletter Anwendungsgenerator verwendbar.

DM 398,-



## STRUKTO 64

STRUKTO 64 ist eine fantastische neue Programmiersprache für strukturiertes Programmieren mit dem C-64 und für alle Programmierer geeignet, die den C-64 als Allround-Computer einsetzen und auf einfache Weise anspruchsvolle Programme erstellen wollen.

### STRUKTO 64 In Stichworten:

Interpretersprache, die die Vorzüge von BASIC und PASCAL vereint – strukturiertes Programmieren – übersichtliche Programme – leichte Erlernbarkeit – einfache Bedienung – eingebauter Toolkit erleichtert das Eingeben und Verbessern von Programmen – leichteres Arbeiten mit der Floppy – Sprite-Editor ermöglicht das Einlesen der Sprite-Formen direkt vom Bildschirm – Graphikbedienung wird mit gut durchdachten Befehlen unterstützt – Abspielen von Musik ist unabhängig vom Programmablauf möglich – ca. 80 neue Befehle – lieferbar als Diskettenprogramm – ausführliches deutsches Handbuch.

DM 99,-



## MASTER 64

MASTER 64 ist ein professionelles Programmentwicklungssystem für den C-64, das es Ihnen ermöglicht, die Programmentwicklungszeit auf einen Bruchteil der sonst üblichen Zeit zu reduzieren. MASTER 64 bietet einen Programmkomfort, den Sie nutzen sollten.

### MASTER 64 In Stichworten:

70 zusätzliche Befehle – Bildschirmmaskengenerator – definieren von Bildschirmzonen – Eingabe aus Zonen – formatierte Ausgabe – Abspelcherung von Bildschirmhalten – Arbeiten mit mehreren Bildschirmmasken – ISAM Dateiverwaltung, in der Datensätze über einen Zugriffsschlüssel angesprochen werden können – Datensätze bis zu 254 Zeichen – Schlüsselgröße bis zu 30 Zeichen – Dateigröße nur von Diskettenkapazität abhängig – Zugriff über Schlüssel und Auswahlmasken – Bildschirm- und Druckmaskengenerator – Erstellung beliebiger Formulare und Ausgabemasken – BASIC-Erweiterungen – Toolkitfunktionen – Mehrfachgenaue Arithmetik (Rechnen mit 22 Stellen Genauigkeit).

DM 198,-

## TEXTOMAT

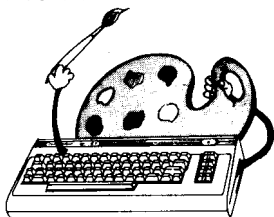
Das Bearbeiten von Texten gehört zum wichtigsten Betätigungsfeld von Homecomputer-Anwendern. So ist es auch nicht verwunderlich, daß eine Unzahl verschiedener Textprogramme für den 64er angeboten wird. TEXTOMAT zeichnet sich dadurch aus, daß er auch vom Einsteiger sofort benutzt werden kann. Über eine Menuezeile können alle Funktionen angewählt werden. Selbstverständlich beherrscht TEXTOMAT deutsche Umlaute und Sonderzeichen.



### TEXTOMAT in Stichworten:

Diskettenprogramm – durchgehend menuegesteuert – deutscher Zeichensatz auch auf COMMODORE-Druckern Rechenfunktionen für alle Grundrechenarten – 24.000 Zeichen pro Text im Speicher – beliebig lange Texte durch Verknüpfung – horizontales Scrolling für 80 Zeichen pro Zeile – läuft mit 1 oder 2 Floppies – frei programmierbare Steuerzeichen – Formularsteuerung für Randeinstellung u.s.w. – komplette Bausteinverarbeitung – Blockoperationen, Suchen und Ersetzen – Serienbriefschreibung mit DATAMAT – formatierte Ausgabe auf Bildschirm – an fast jeden Drucker anpaßbar – ausführliches deutsches Handbuch mit Übungslektionen.

DM 99,-



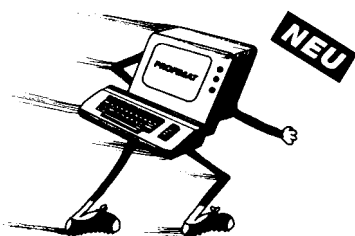
## PAINT PIC

Malen (!) mit dem Computer, Welch eine faszinierende Idee. Mit dem Malprogramm PAINT PIC für den COMMODORE 64 wird diese Idee Realität. Mit PAINT PIC ist es auch für den Einsteiger leicht, fantastische Computerbilder zu erstellen. Man kann die Bilder auf Diskette abspeichern und wieder laden und selbstverständlich steht auch weiterhin der COMMODORE-Zeichensatz zur Verfügung. Wichtig: PAINT PIC benötigt keine zusätzliche Hardware.

### PAINT PIC in Stichworten:

Programmsteuerung: Tastatur – Steuerung des Stifts: Cursorstasten und eckige Klammer (diag.) (Joystick kann benutzt werden) – Routinen: Linien, Rechtecke, Dreiecke, Parallelogramme, Kreise, Kreisbögen, Ellipsen, Bestimmung von Mittelpunkt, und perspektivischer Linie, Kopieren und Drehen von Teilbildern, Verdoppeln, halbieren und spiegeln von Teilbildern – Modi: Malstiftmodus (schmale Linie) Pinselmodus (8 verschiedene Breiten) (Art der Linie selbst definierbar) – Textmodus (kompl. Zeichensatz COMMODORE) (Hoch-Tiefschrift) – Speichern: Teilbilder (Blöcke) oder ganze Bilder – Menue: 1 Hauptmenue mit 8 Untermenues – mit ausführlichem deutschem Handbuch – Diskettenprogramm – Bilder kann man auf Diskette abspeichern.

DM 99,-



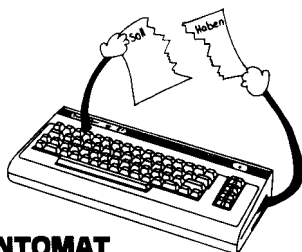
## PROFIMAT

Wer sich tiefer in die Inneren des Computers begeben will, kommt ohne besonderes Werkzeug nicht aus. Einerseits muß der volle Einblick in alle Speicherbereiche möglich sein, andererseits soll der Umgang mit Maschinenprogrammen so komfortabel wie möglich gestaltet sein. PROFIMAT hat Lösungen für beide Probleme: Der Maschinensprache-Monitor PROFI-MON bietet alle Hilfsmittel zum Umgang mit Maschinenprogrammen; PROFI-ASS ist ein Macro-Assembler, der das Schreiben von Maschinenprogrammen fast so einfach macht wie das Programmieren in BASIC.

### PROFIMAT in Stichworten:

Registerinhalte und Flags anzeigen – Speicherinhalte anzeigen – Maschinenprogramme laden, ausführen und speichern – Speicherbereiche durchsuchen, vergleichen, füllen und verschieben – echter Einzelschrittmodus – Setzen von Unterrechnungspunkten – schneller Trace-Modus – Rückkehr zu BASIC – formatfreie Eingabe – Verketzung beliebig vieler Quellprogramme – erzeugter Objektcode kann in Speicher oder auf Diskette gehen – formatiertes Assemblerlisting – ladbare Symboltabellen – redefinierbare Symbole – Operatoren – Unterstützung der Fließkommaarithmetik – bedingte Assemblerlisting – Assemblerschleifen – MACROS mit beliebigen Parametern.

DM 99,-



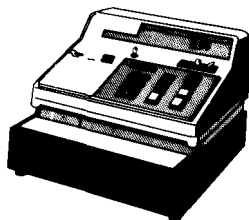
## KONTOMAT

KONTOMAT ist ein menuegesteuertes Einnahme-Uberschubprogramm nach § 4(3) EStG mit Kassenbuch, Bankkontenüberwachung, automatischer Steuerbuchung, AFA Tabellenerstellung, Kontenblättern, Ermittlung der USt-Voranmeldungswerte und Monats- und Jahresabrechnung. Der neue KONTOMAT ist voll parametrisiert und läßt sich damit an Ihre Bedürfnisse anpassen. Für alle Gewerbetreibenden, die nicht laut HGB zur Buchführung verpflichtet sind. KONTOMAT ist für den gewerblichen Einsatz, aber auch als Lernprogramm oder zur Haushaltsbuchführung geeignet.

## KONTOMAT In Stichworten:

Diskettenprogramm – maximal 120 Konten – Beträge mit bis zu 6Vor- und 2Nachkommastellen – 4 Mehrwert- und Vorsteuersätze – intervallmäßige Belegeingabe – 4 Buchungssarten (SOLL, HABEN, SOLL/HABEN und HABEN/SOLL) – Anzeige der Soll- und Habensumme bei mehrfachen Buchungssätzen – komfortable Belegeingabe mit Datum, Buchungstext, Steuerkennzeichen und Betrag – Druck des Journals während der Belegeingabe – Druck von umfangreichen Kontenblättern – Druck einer Summen- und Saldenliste mit Monats- und Jahresumsatzsummen – betriebswirtschaftliche Auswertung mit Druckausgabe – Ermittlung und Druckausgabe der Umsatzsteuerzahllast – Speicherung der Anlagegüter und automatische Abschreibung am Jahresende – übersichtliche Afa-Liste – arbeitet mit 1 oder 2 Laufwerken – umfangreiches deutsches Handbuch.

DM 148,-



## FAKTUMAT

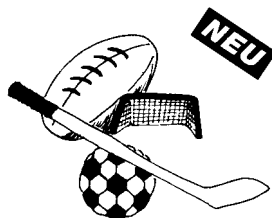
Mit FAKTUMAT ist das Schreiben von Rechnungen kein Alptraum mehr. Eine Sofortfakturierung mit integrierter Lagerbuchführung. Individuelle Anpassung von Steuersätzen, Maßeinheiten und Firmendaten. Kunden- und Artikelstamm voll pflegbar. Schneller Zugriff auf Kunden- und Artikeldaten, über freidefinierbaren, 6-stelligen Schlüssel. Automatische Fortschreibung von Artikel- und Kundendaten, individuell nutzbar. Alles in allem die Arbeits- und Zeiterparnis, die Sie sich schon immer gewünscht haben.

## FAKTUMAT In Stichworten:

voll menügesteuert – läuft mit einer oder zwei Floppies – Diskettenwechsel (eine Floppy) nur beim Wechsel vom Hauptmenue ins Unterprogramm und umgekehrt – mit Ausnahme des Ausschaltens der Floppy während der Verarbeitung werden alle Fehler abgefangen (z. B. Drucker nicht eingeschaltet – arbeitet mit 1525, 1526 (?), MPS 801, EPSON Drucker und DATA BECKER Interface – voll parametrisiert: Firmenkopf, MWSt. und Rabattsätze, Größe der Dateien beliebig wählbar – 5 Zeilen für Firmenkopf je 30 Zeichen (erste Zeile erscheint auf der Rechnung in Breitschrift – 4 Mehrwertsteuer-Sätze; während der Rechnungsschreibung können also Artikel mit unterschiedlichem Mehrwertsteuer-Satz verrechnet werden – 10 Rabattsätze (Rabattsatz 1 vorbelegt mit 0%), bei der Rechnungsschreibung kann jedem Artikel ein Rabattsatz zugewiesen werden – maximal 1900 Artikel bei 50 Kunden oder 950 Kunden bei 100 Artikel (max. Artikel =  $1000 \cdot \text{Kunden} + 2$ ); max. Kunden =  $\lfloor 2000 \cdot \text{Artikel} \rfloor / 2$ ) – manuelle Eingabe von Artikeln und/oder Kunde während der Rechnungsschreibung – d.h. es können mehr Artikel verrechnet werden als überhaupt in die Datei passen (bei Verzicht auf Lagerbuchführung) bzw. es können Rechnungen an Kunden geschrieben werden, die nicht erfaßt wurden –

integrierte Lagerbuchführung mit Ausgabe einer Inventurliste – Rechnungsbeträge und Datum werden in der Kundendatei festgehalten – Druck von: Rechnung (mit Abbuchen aus Lager), Rechnung (ohne Abbuchen aus Lager), Lieferscheine – deutsches detailliertes Handbuch mit Übungs- und Anwendungstext – deutsche Bedienung innerhalb des Programms (z. B. „Artikel nicht vorhanden“ anstelle „RECORD NOT PRESENT“).

DM 148,-



## UNI-TAB

Heute schon die Bundesliga-Tabelle von morgen kennen, das geht mit UNI-TAB. Alle Rechnereien, die man ohne dieses Programm nie machen würde, lassen sich in Sekundenschnelle durchführen. Wer will, kann mit simulierten Spielergebnissen den Weltmeister '86 vorausberechnen. Aber nicht nur Fußball-Ligen können tabellarisch erfaßt werden, fast alle Sportarten sind UNI-TAB-fähig. Gam am Rande: für viele Sportarten stehen die bekannten Piktogramme zur Verfügung.

## UNI-TAB In Stichworten:

Menüsteuerung über die Funktionstasten mit leicht verständlichen Auswahlmöglichkeiten – Bedienerfreundlich (Mannschaften werden über Kennzahlen gesteuert) – Ligen mit 4 bis 20 Mannschaften können verwaltet werden (6 bis 38 Spieltage möglich) – unsinnige Ligen (z. B. 13 Mannschaften sollen 5 Spieltage absolvieren) sind ausgeschlossen – favorisierte Mannschaft kann während des Programmablaufs durch reverse Darstellung gekennzeichnet werden – Tabelle kann geändert werden (wichtig bei Spielanullierungen) – drei verschiedene Tabellenarten können abgespeichert und später eingelesen werden (die aktuelle Tabelle tunabhängig von der Vollständigkeit eines Spieltages), der komplette Spieltag (Vollständigkeit und Nummer des Spieltages werden automatisch errechnet), die simulierte Tabelle (der Anwender kann so selbst Schicksal spielen und seinen Tip später mit dem tatsächlichen Geschehen vergleichen) – zwei verschiedene Arten der Saisonübersicht (die statistische Übersicht zeigt an, welchen Tabellenplatz das jeweilige Team bei welchem Punkte- und Torverhältnis an den einzelnen Spieltagen einnimmt; die graphische Übersicht zeigt die Leistungskurve jeder Mannschaft) – alle Tabellen und Graphiken sind als Hardcopy auf einem Drucker darstellbar – bei Fehlbildung (z. B. gewünschte Druckausgabe bei nicht eingeschaltetem Drucker) erscheinen leicht verständliche deutsche Fehlermeldungen.

DM 69,-

### ***DAS STEHT DRIN:***

Das TRAININGSBUCH ZUM SIMON'S BASIC erklärt detailliert den Umgang mit den über 100 Befehlen des SIMON'S BASIC. Alle Befehle werden ausführlich dargestellt, auch die, die nicht im Handbuch stehen (!). Natürlich zeigen die Autoren auch die „Macken“ des SIMON'S BASIC und geben wichtige Hinweise, wie man diese umgeht. Selbstverständlich enthält das Buch auch zahlreiche Programmbeispiele und Programmiertricks. Nach jedem Kapitel folgen Testaufgaben zum optimalen Selbststudium und zur Lernkontrolle.

Aus dem Inhalt:

- Programmierhilfen
- Fehlerbehandlung
- Programmschutz
- Programmstruktur
- Stringoperationen
- Ein-/Ausgabe Peripheriebefehle
- Graphik
- Zeichensatzerstellung
- Sprites
- Musik
- Steuernde Peripherie

### ***UND GESCHRIEBEN HABEN DIESES BUCH:***

Axel Plenge, Student, ist vielen bereits als Autor der SUPERGRAPHIK, des MATHEMAT und des GRAFIK-BUCHES bekannt.

Norbert Szczepanowski, EDV-Kaufmann und Bestsellerautor schreibt die erfolgreichen Einsteigerbücher bei DATA BECKER.

***ISBN 3-89011-009-6***