Am Beispiel des Sinclair-Logo wird in die Programmierung nach dem Top-Down-Prinzip eingeführt.

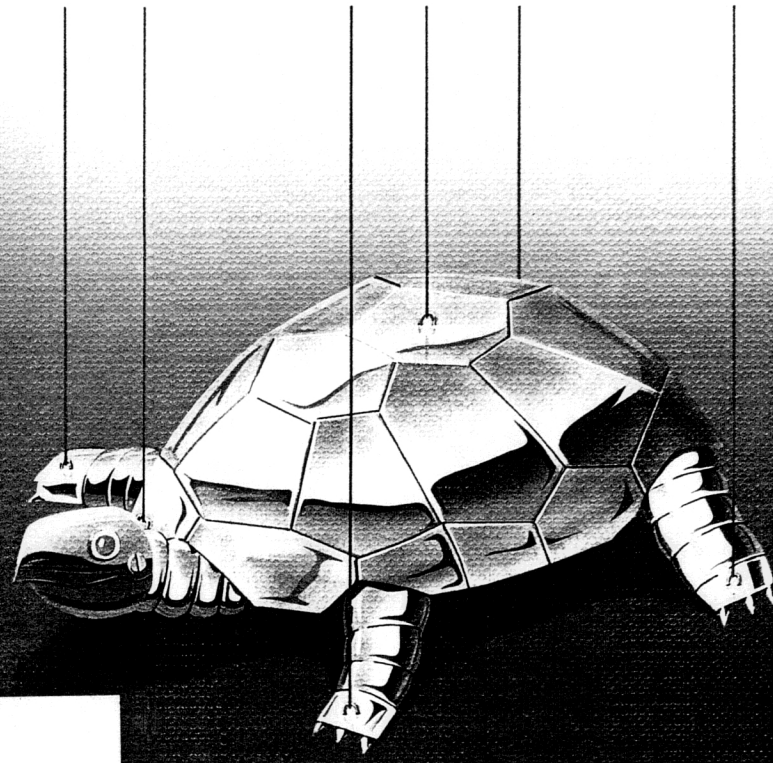
Abgesehen von den gängen Kenntnissen, die das Logo-Handbuch vermittelt, werden keine Grundlagen vorausgesetzt.

Logo ist ein sehr leistungsfähiges Programmierwerkzeug. Die vielfältigen Möglichkeiten dieser Sprache werden mit einfachen und dennoch interessanten Beispielen erläutert.

Obwohl diese Sprache wegen ihrer Anschaulichkeit sehr für Kinder geeignet ist, findet auch der erfahrene Programmierer noch Neues. Das Sinclair-Logo unterscheidet sich nur unwesentlich von der ursprünglichen Logo-Idee.

Erika Hölscher

Logo auf dem Spectrum



lüthig

Erika Hölscher · Logo auf dem Spectrum

Erika Hölscher

Logo auf dem Spectrum

Dr. Alfred Hüthig Verlag Heidelberg

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Hölscher, Erika:
Logo auf dem Spectrum / Erika Hölscher. –
Heidelberg : Hüthig, 1986.
ISBN 3-7785-1121-1

Printed in Germany
Satz: Druckerei Bitsch GmbH · Birkenau

Vorwort

Logo wurde vor etwa 10 Jahren schon für den Bildungsbereich entwickelt. Man wollte einerseits eine Sprache, die leicht verstanden und benutzt werden kann, andererseits aber auch ein leistungsfähiges Werkzeug, das das funktionelle Denken beim Programmieren unterstützt.

Hinter Logo steckt nicht einfach eine Programmiersprache, sondern eher eine Denkweise. Stellen wir uns vor, ein Kind kenne die Wörter „gehen", „nehmen", „wiederkommen" und „ball". Nun definieren wir das neue Wort „holen" mit „gehen nehmen ball wiederkommen". So funktioniert Logo.

Diese Struktur von Logo ist der Grund dafür, daß es einfach zu verstehen und zu erlernen ist und gleichzeitig sehr leistungsfähig sein kann. Was Logo letzten Endes kann, hängt vom Benutzer ab.

Sinn dieses Buches soll es sein, die Idee von Logo aufzuzeigen. Das hier Erarbeitete kann dann die Grundlage für eigene Ideen des Lesers sein.

Inhaltsverzeichnis

Arbeiten mit diesem Buch

Voraussetzungen

Sie sollten über einen Computer und ein Logo-Programm verfügen. Es wird nicht vorausgesetzt, daß Sie schon damit gearbeitet haben.
Als Grundlage für dieses Buch diente der ZX-Spectrum mit dem Sinclair-Logo. Die Benutzer anderer Systeme werden unter Umständen bemerken, daß bei ihnen ein Befehl anders lautet.

Vorgehensweise

Im ersten Kapitel werden wir uns die Grundbegriffe erarbeiten. Dort finden Sie die Erklärung für die wichtigen Begriffe „Prozeduren", „Variable" usw.

Ein Problem stellt sich all denen, die kein Englisch können. Im Gegensatz zu anderen Programmiersprachen kommt Logo diesem Aspekt aber entgegen, weil hier Umgangsenglisch ausreicht und notfalls ein Wörterbuch helfen kann.
Zu den Befehlen werde ich allerdings die Übersetzung anfügen, ebenso für die wichtigsten Meldungen des Systems. Danach dürften kaum noch Probleme auftreten, wenn etwas aus dem Handbuch entnommen werden muß.

In den weiteren Kapiteln werden wir kleine Programme entwickeln. Dabei wird Grafik im Zusammenhang mit Mathematik im Vordergrund stehen. Es ist nicht beabsichtigt (und auch im Rahmen dieses Buches nicht möglich), umfassende Lösungen anzubieten. Die Programme aus diesem Buch sollen vielmehr Anregungen und Lösungshinweise für eigene Problemstellungen bieten.

Kapitel 1: Grundlagen

1.1 Primitives

Jede Programmiersprache stellt dem Programmierer einen Grundwortschatz zur Verfügung. Mit diesen eingebauten Befehlen können wir den Computer veranlassen, etwas bestimmtes zu tun. In Logo nennt man diese bereits vorhandenen Befehle Primitives.

Diese Primitives werden in zwei Gruppen eingeteilt:

Befehle

Ein Befehl ist eine Anweisung an den Computer, die dieser sofort ausführen kann, z.B. „Drucke auf den Bildschirm". Der Befehl alleine reicht allerdings oft nicht; hier fehlt z.B. noch die Angabe, was denn gedruckt werden soll. Diese Angaben nennt man Parameter.

Funktionen

Eine Funktion liefert immer ein Ergebnis, z.B. 3+4. Mit diesem Ergebnis kann der Computer ohne zusätzlichen Befehl nichts tun; LOGO antwortet in dem Fall unseres obigen Beispiels etwa „You don't say what to do with 7" – Du sagst nicht, was ich mit 7 tun soll. Eine Funktion verlangt immer, daß wir Werte (im weitesten Sinne) eingeben, die verarbeitet werden sollen. Diese Werte nennt man Operatoren oder Argumente.

1.2 Prozeduren

Logo unterscheidet sich von dem sehr verbreiteten BASIC dadurch, daß die Anweisungen nicht der Reihe nach numeriert werden, sondern zu sogenannten Prozeduren zusammengefaßt werden. Diese Prozeduren werden behandelt wie die Primitves; d.h. es können sowohl Befehle als auch Funktionen von uns definiert werden.

Unterscheiden können wir sie nur über die folgenden zwei Befehle: POTS (Print OuT procedureS) druckt alle definierten Prozeduren aus.
.PRIMITIVES dagegen ergibt eine Liste aller in Logo vorhandenen Grundwörter.

1.3 Variable

Eine Variable ist eine Speicherstelle für veränderliche Werte. Sie kann durch das Programm oder auch durch Eingaben während eines Programms verändert werden. Dabei ist nicht die Speicherstelle gemeint, wo der Wert tatsächlich steht. Das braucht uns nicht zu interessieren. Über den Namen können wir den Wert erhalten. Die Speicherorganisation ist Sache des Computers.

1.3.1 Lokale Variable

In Logo sind grundsätzlich alle Variablen örtlich gültig, d.h. nach der Ausführung der Prozedur sind sie nicht mehr definiert. Wollen wir sie danach drucken lassen, so antwortet Logo „... has no value" (... hat keinen Wert). Die Punkte stehen hier stellvertretend für den Variablennamen.

1.3.2 Globale Variable

Unter globalen Variablen verstehen wir solche, die auch nach der Ausführung einer Prozedur noch verfügbar sind. Diese Variablen haben allerdings einige Nachteile:
Durch versehentliche Ansprache einer solchen Variablen kann das Programm nicht ordnungsgemäß abgearbeitet werden, weil es möglich ist, daß ein nicht zulässiger Wert erreicht wird.
Jede Variable belegt Platz im Speicher. Damit sollten wir aber vorsichtig umgehen. Deshalb ist es günstiger, so wenig globale Variablen zu benutzen wie möglich.
Mit **MAKE "name wert** ist es allerdings möglich, eine Variable global zu definieren; beispielsweise wenn sie von einer anderen Prozedur auch benötigt wird.

1.3.3 Namen

Variablennamen beginnen in Logo mit einem Doppelpunkt oder einem Anführungszeichen. Der Doppelpunkt wird beim Aufruf gesetzt, z.B. **PRINT :zahl**. Das Anführungszeichen ist bei der Definition eines Platzhalters als global erforderlich, z.B. **MAKE "zahl 10**.
Danach folgen die Buchstaben und ggf. Ziffern. Das Sinclair-Logo läßt beliebig lange Namen zu, das erste Zeichen des Namens muß jedoch ein Buchstabe sein. Es wird bei Logo aber zwischen Klein- und Großbuchstaben unterschieden. In der Regel werden wir die Namen so wählen, daß wir aus dem Namen erkennen können, wofür sie gebraucht werden. Dabei werden wir die normale Rechtschreibung benutzen, weil das leichter lesbar ist.

1.4 Grafische Darstellungen

In den späteren Kapiteln werden unsere Programme etwas länger werden. Daher ist es sinnvoll, sich den Ablauf des Programms mithilfe von Grafiken deutlich zu machen bzw. vorher festzulegen. Wir werden dazu **Struktogramme** verwenden, die der Struktur von Logo am ehesten entsprechen und sehr leicht zu verstehen sind. Grundsätzlich gilt, daß in solchen Diagrammen keine Befehle aus einer Sprache verwandt werden sollten. Dadurch ist es für jeden lesbar und auch in andere Sprachen übertragbar.

1.5 Der Editor

Unter einem Editor können wir uns ein Textverarbeitungsprogramm vorstellen. Damit schreiben und verbessern wir die Prozeduren. Die Bedienung werden wir im Laufe der Programmentwicklung lernen. In den Logo-Handbüchern sollte sich ein Abschnitt finden, in dem die Möglichkeiten dargestellt sind.

1.5.1 Zeileneditor

Zur Neudefinition können wir einen Zeileneditor aufrufen. D.h., unser **Cursor** (Schreibstellenanzeiger) darf sich nur innerhalb eines bestimmten Bereiches bewegen. Nach dem Verlassen einer Programmzeile ist es nicht möglich, dort wieder hinzufahren und zu ändern.

Wenn wir mit diesem Befehl versuchen, eine bereits definierte Prozedur zu definieren, wird Logo melden „... is already defined" (... ist schon definiert).
Diese Art Editor wird mit **TO name** aufgerufen und durch die Eingabe **END** alleine in einer Zeile wieder verlassen.

1.5.2 Bildschirmeditor

In Logo steht uns aber auch ein Bildschirmeditor zur Verfügung. D.h. wir können mit dem Cursor (Pfeiltasten) an jede beliebige Stelle auf dem Bildschirm gehen und dort weiterschreiben.
Mit diesem Befehl können sowohl neue Prozeduren definiert als auch bereits definierte geändert werden. Durch diesen Aufruf schaltet Logo automatisch auf den Textbildschirm um (s.u.).
Diesen Editor können wir mit **EDIT "name** aufrufen. Er wird mit Extended Mode C (s.u.) wieder verlassen.

1.5.3 Abkürzungen

Wir werden oft bestimmte Wörter benötigen, die ich der Einfachheit halber abkürzen werden:

CAPS Shift	=	CS
SYMBOL Shift	=	SS
EXTENDED MODE	=	EX

1.5.4 Cursorfunktionen

Cursor rechts	CS + 8
Cursor Zeilenende	EX + 8
Cursor links	CS + 5
Cursor Zeilenanfang	EX + 5
Cursor hoch	CS + 7
Cursor Listing Anfang	EX + 7
Cursor runter	CS + 6
Cursor Listing Ende	EX + 6
Zeichen löschen	CS + 0

Dies sind nur die zunächst wichtigsten Funktionen.

1.6 Text- und Grafik

Bei Sinclair ist es nicht üblich, zwischen Text und Grafik zu unterscheiden, was ohne Probleme eine Mischung von beidem auf dem Bildschirm ermöglicht. Dennoch wird zunächst zwischen Grafik- und Textbildschirm unterschieden.
Mit jedem Grafikbefehl wird der Grafikbildschirm automatisch aufgerufen. War er schon aufgerufen, wird er nicht gelöscht. Für den Text können dann die beiden unteren Bildschirmzeilen benutzt werden; es ist auch möglich, innerhalb des Grafikbereichs zu schreiben.
Der Textbildschirm ist nur ohne Grafik zu benutzen. Er ist sinnvoll für längere Texte wie z.B. Erklärungen. Er muß extra aufgerufen werden; eine vorhandene Grafik wird dadurch gelöscht.

Kapitel 2: Einfache Prozeduren

2.1 Der Spectrum lernt Manieren

Wenn wir morgens an unseren Arbeitsplatz kommen, sagen wir „Guten Tag". Warum soll ein Computer nicht auch höflich sein können? Wir müssen es ihm nur beibringen!

2.1.1 Befehle

PRINT heißt drucke. Der Druckbefehl in Logo kann sehr unterschiedlich sein. In unserem Fall folgen jeweils Texte aus mehreren Wörtern; diese nennt man eine **Liste**. Sie wird in eckige Klammern eingeschlossen. Wenn nur ein Wort folgt, könnten wir diesem auch ein Anführungszeichen voranstellen, statt es in eckige Klammern zu setzen. Im Gegensatz zu BASIC kann PRINT in Logo nicht ohne Parameter aufgerufen werden. Wenn nur eine leere Zeile gedruckt werden soll, so müssen wir eine leere Liste drucken lassen.

TEXTSCREEN ruft den Textbildschirm auf bzw. löscht ihn. In unserer ersten Prozedur wollen wir noch auf Grafik verzichten.

Beide Befehle dürfen in Logo abgekürzt werden: PRINT kann auch mit PR befohlen werden, TEXTSCREEN mit TS.

2.1.2 Eingeben der Prozedur

Wie schon eingangs erwähnt, gibt es zwei Möglichkeiten, den Editor aufzurufen. Die speziell für Neudefinitionen gedachte Art **TO name** stellt uns nur einen Zeileneditor zur Verfügung. Wir werden daher grundsätzlich **EDIT "name** benutzen, dadurch haben wir einen Bildschirmeditor.

Geben wir also EDIT''TAG (gefolgt von ENTER) ein. Logo antwortet jetzt mit dem Textscreen (Bildschirm); in der ersten Zeile steht **TO TAG**, der Cursor befindet sich auf dem ersten Zeichen. Nun gehen wir eine Zeile tiefer (Cursor abwärts: CS + 6) und tippen die Prozedur ein:

```
TO TAG
TEXTSCREEN
PRINT [Guten Tag]
PRINT []
PRINT [Jetzt kannst Du mit LOGO]
PRINT [lernen.]
PRINT []
END
```

Es ist nicht erforderlich, nach jedem Befehl eine neue Zeile zu beginnen, aber übersichtlicher und bei späteren Änderungen eines Programms auch einfacher zu handhaben. Eine neue (Bildschirm-) Zeile erreichen wir mit ENTER.

Das Logo-Wort END braucht immer eine eigene Zeile.
Probieren wir ein wenig die Cursortasten aus: Cursor hoch (CS + 7), Cursor rechts (CS + 8) und Cursor links (CS + 5) usw.

Der Computer soll Höflichkeit lernen. So ganz in Ordnung ist diese Prozedur aber nicht: Im ersten Druckbefehl fehlt der Punkt am Ende des Satzes. Wir gehen auf die eckige Klammer am Ende dieser Zeile und tippen den Punkt ein. Er wird automatisch eingefügt. Logo befindet sich (auf dem Spectrum) immer im **Einfügemodus (INSERT)**, wobei die Zeichen links vom Cursor erscheinen. Sicher wird es immer wieder vorkommen, daß wir uns vertippen. Natürlich können wir diese falschen Zeichen löschen: **DELETE** (CS + 0) löscht das Zeichen links vom Cursor.

Die Logo-Wörter habe ich hier groß geschrieben und werde das auch weiterhin tun. Es ist aber nicht unbedingt erforderlich, sie auch in Großbuchstaben einzugeben. Logo ist da sehr tolerant und versteht auch „print" oder „Print". Um die Eingabe in Großbuchstaben zu erleichtern, können wir auf CAPS LOCK (CS + 2) umschalten. Durch ein kleines „l" oder ein großes „C" unten rechts wird uns der eingestellte Modus angezeigt. Auch der Tastaturton verändert sich.

Unsere Prozedur ist nun soweit fertig. Darum können wir jetzt den Editor verlassen: **EX + C** bringen uns wieder in den Direktmodus. Den EX erreichen wir – wie gewohnt – durch CS + SS. Unten rechts steht jezt ein „E".

Logo meldet jetzt: „TAG defined" (die Prozedur TAG ist definiert). Wir können sie jetzt aufrufen, indem wir im Direktmodus eingeben: „TAG" (gefolgt von ENTER). Normalerweise sollten jetzt die beiden in eckige Klammern gesetzten Texte auf dem Bildschirm erscheinen, getrennt durch eine Leerzeile.

Wenn das nicht der Fall ist, haben wir uns irgendwo vertan und Logo wird uns melden „I don't know how to ... in TAG" (Ich kenne das Wort ... in der Prozedur TAG nicht). Vielleicht haben wir PRONT statt PRINT geschrieben. Dann werden wir den Editor einfach noch einmal aufrufen und das falsche Wort verbessern.

```
TO TAG
TEXTSCREEN
PRINT [Guten Tag.]
PRINT []
PRINT [Jetzt kannst Du mit LOGO]
PRINT [lernen.]
PRINT []
END
```

2.2 Ein wenig Grafik

Logo ist berühmt und bekannt wegen seiner leistungsfähigen Grafik-Befehle. Davon wollen wir natürlich auch etwas haben. Eine Sache, die in BASIC immer schwierig zu bewältigen ist, sind Spiralen. Wir werden jetzt Logo beibringen, eine Spirale zu malen.

2.2.1 Befehle

SHOWTURTLE (zeige die Schildkröte, manchmal auch Igel genannt) schaltet den Grafikschirm ein. Nach diesem Aufruf steht die Schildkröte immer da, wo sie zuletzt stand, zu Beginn in der Mitte des Schirms mit dem Kopf nach oben. Beim Spectrum wird sie durch ein kleines Dreieck dargestellt. Die Abkürzung heißt ST. **HIDETURTLE** läßt sie verschwinden. Sie kann aber weiter malen. Dieser Befehl hat die Kurzform HT.

FORWARD n befiehlt der Schildkröte, um n Pixel vorwärts zu gehen. Der Gegenbefehl dazu heißt **BACK n**. Die Kürzel heißen FD bzw. BK.

RIGHT n veranlaßt sie, sich um den Winkel n (in Grad) nach rechts zu drehen. **LEFT n** bewirkt eine Drehung nach links. Hier sind die Kurzformen RT bzw. LT erlaubt.

2.2.2 Rekursive Prozeduren

Diese Prozedur wird sich selbst aufrufen. Deshalb wird sie rekursiv genannt. Damit das nicht endlos geschieht, brechen wir sie mit einer Bedingung ab:

IF bedingung [Befehlsliste]

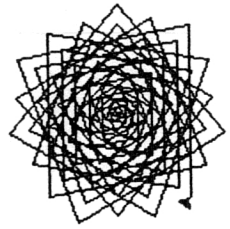
STOP müßte in den eckigen Klammern stehen, wenn die Prozedur aufhören soll.
Eine **Bedingung** vergleicht beispielsweise den Wert einer Variablen mit einem vorgegebenen Endwert. Dafür können die **Vergleichsoperatoren** > (größer), < (kleiner) und = (gleich) benutzt werden.

2.2.3 Variable – Parameter

Diese Prozedur wird Variable benutzen, die wir gleich hinter den Namen schreiben werden. Dadurch wird erreicht, daß wir beim Aufruf der Prozedur die Art der Spirale bestimmen können. Wir benutzen drei Variable, d.h. diese Prozedur hat drei Parameter.

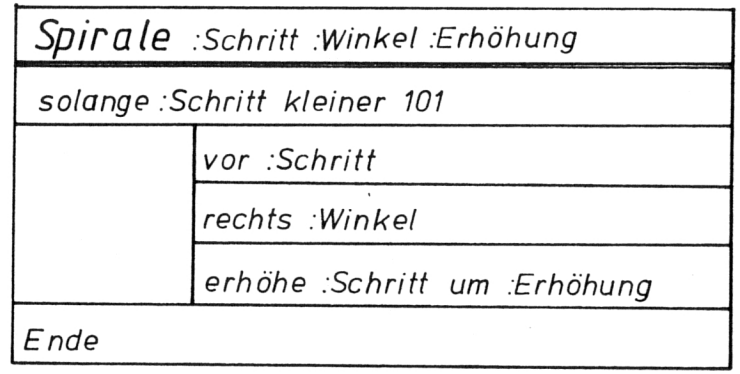
```
TO SPIRALE :Schritt :Winkel :Erh
oehung
IF :Schritt > 100 [STOP]
FORWARD :Schritt
RIGHT :Winkel
SPIRALE :Schritt + :Erhoehung :W
inkel :Erhoehung
END
```

Die schönsten Werte erreicht man meiner Ansicht nach mit kleinen Werten für :Laenge und :Schritt und Angaben zwischen 91 und 179 für :winkel. Der folgende Ausdruck erfolgte mit SPIRALE 1 100 1.



Spirale 1 100 1

Den Ablauf dieser Routine kann man sehr schön in dem Struktogramm erkennen:



2.3 Verknüpfen der beiden Prozeduren

Um einem anderen vielleicht die Fähigkeiten unseres Computers zu zeigen, könnten wir diese beiden Prozeduren verknüpfen. Dafür gibt es zwei Möglichkeiten:

TAG ruft SPIRALE auf: Dazu müßten wir die Prozedur TAG verändern. Bei nur zwei definierten Prozeduren ist es kein Problem, den Überblick zu behalten, aber bei vielen kann es schon schwieriger werden. Vielleicht übersehen wir eine andere Querverbindung, bei der diese Änderung stört.

Zweckmäßiger ist es, eine neue Prozedur zu schreiben, die ihrerseits die beiden anderen aufruft. Sollten diese beiden Routinen auch von anderen benötigt werden, so stört das nicht weiter. Die neue Prozedur könnten wir ja DEMO nennen; vielleicht könnte sie auch zwei Spiralen malen. Dazu brauchten wir noch zwei Befehle: **WAIT n** setzt eine Pause von n/50 Sekunden. **CLEARSCREEN** löscht den Grafikschirm und setzt die Schildkröte an die Startposition. Dieser Befehl darf abgekürzt werden: CS.

```
TO DEMO
TAG
WAIT 200
SPIRALE 1 100 1
HIDETURTLE
WAIT 200
CLEARSCREEN
SPIRALE 1 170 1
WAIT 200
CLEARSCREEN
END
```

Kapitel 3:
Ein Mathematik-Programm

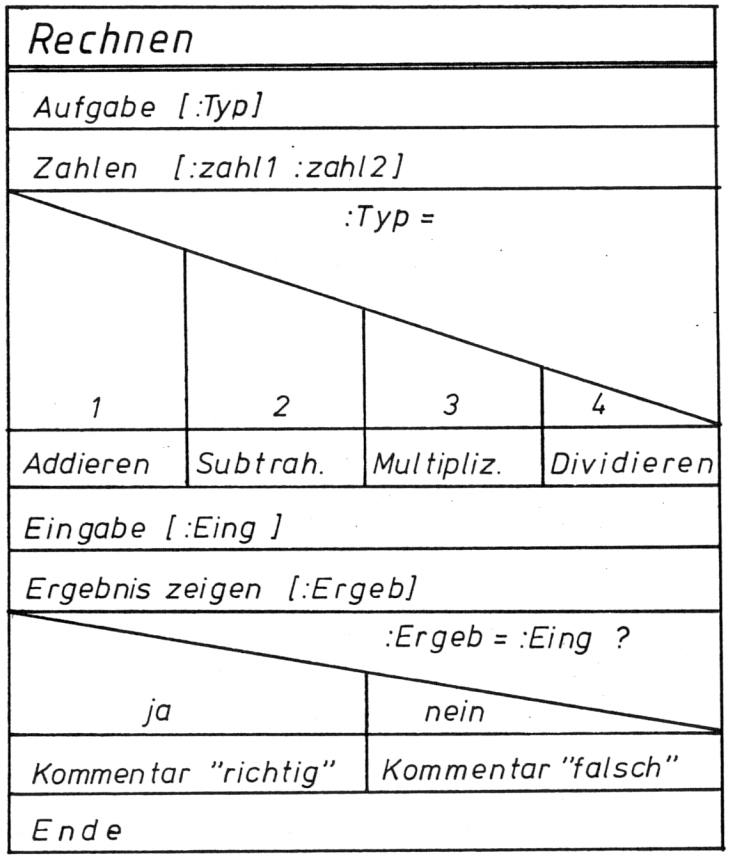
3.1 Pflichtenheft

Wenn wir ein größeres Programm schreiben wollen, ist es günstig, sich erst einmal klar zu machen, was das Programm können soll. Diese Aufgabenliste nennt man das Pflichtenheft.

Wir wollen ein Programm schreiben, das mathematische Aufgaben stellt und löst, kombiniert mit etwas Grafik. Was muß es können?

- Aufgabentyp auswählen (Zufall)
- Aufgabe erstellen und zeigen
- Aufgabe ausrechnen
- Grafik zeigen
- Ergebnis erfragen
- Ergebnis anzeigen
- Kommentar, je nach Richtig oder Falsch

Ein Struktogramm zeigt noch etwas genauer, wie das Programm aussehen wird. Ein Struktogramm zeigt – wie schon der Name sagt – die Struktur des Programms.



Im Struktogramm können schon die wichtigsten globalen Variablen eingetragen sein. Hier sind sie zur Kennzeichnung in eckige Klammern gesetzt.

3.1.1 Die Auswahlroutine

Zuerst brauchen wir lt. Struktogramm die Routine, die den Aufgabentyp zufällig aus vier verschiedenen Möglichkeiten auswählt. Dazu fehlen uns ganz offensichtlich noch zwei Befehle:
Die **Zufallsanweisung** brauchen wir, um zwischen einer gegebenen Anzahl Werte auszuwählen. **RANDOM x** ist eine Funktion, die einen scheinbar zufälligen Wert berechnet. Dabei ist 0 der kleinste mögliche Wert und das **Argument** der Funktion (x) die Anzahl der möglichen Werte. Damit wir Werte zwischen 1 und 4 erhalten, müssen wir 1 zu dem Ergebnis der Funktion addieren: **1 + RANDOM 4** ergibt nun als kleinsten Wert immer 1 und als größten 4.

Ermittelt die Funktion als Ergebnis beispielsweise eine 2, so wird noch die 1 addiert, das Endergebnis ist also 3:

RANDOM	4 =	0	1	2	3
+		1	1	1	1
=		1	2	3	4

Dieses Ergebnis müssen wir an die nächste Prozedur weitergeben. Eine Möglichkeit dafür ist die Definition der Variablen als Typ global. Um eine Variable global zu definieren, wird der Befehl **MAKE ”** verwendet. Dem Anführungszeichen folgt der Variablenname:
MAKE ”Typ 1 + RANDOM 4
Das war's schon! Die Routine ist nur sehr kurz. Hier das Listing:

```
TO AUSWAHL
MAKE "Typ 1 + RANDOM 4
END
```

Wenn wir prüfen möchten, ob das auch funktioniert, können wir einen sehr praktischen Befehl benutzen: **REPEAT** heißt „wiederhole“. Hinter den Befehl müssen wir zunächst die Anzahl der Wiederholungen eintragen und dann in eckigen Klammern die Anweisungsfolge, die wiederholt werden soll:

REPEAT 20 [AUSWAHL PRINT :Typ] und ENTER.
Die Prozedur Aufgabe wird jetzt 20 mal aufgerufen. Da die Variable :Typ global definiert ist, können wir sie anschließend ausdrucken lassen.

3.1.2 Die Zahlen bestimmen

Der nächste Schritt zu unserem Gesamtprogramm ist nun ganz einfach: Wir brauchen noch zwei zufällige Zahlen, die wir später für die Aufgaben benutzen. Die Anweisungen sind dieselben wie in der Prozedur Aufgabe, nur daß wir hier natürlich größere Zahlen zulassen sollten:

```
TO ZAHLEN
MAKE "Eins 50 + RANDOM 50
MAKE "Zwei 1 + RANDOM 50
END
```

Damit es bei der Subtraktion keine Probleme gibt, sind die Zufallsanweisungen so ausgelegt, daß die erste Zahl über 50 liegt und die zweite unter 50. So kommt bei der Subtraktion immer ein positives Ergebnis heraus.

Die Entscheidung, welche Rechenart ausgeführt werden soll, überlassen wir dem Hauptprogramm. Wir kümmern uns zuerst um die einzelen Routinen, die wir noch brauchen.

Für alle folgenden Routinen brauchen wir eine neue Variable, die global definiert werden muß: Ergeb soll das Ergebnis der Rechnung aufnehmen.

3.1.3 Addieren

Die Addition als solche ist einfach:
MAKE "Ergeb :Eins + :Zwei
Diesen Befehl kannten wir ja schon. Nun soll aber auch die Aufgabe gezeigt werden, d.h. ausgedruckt. Auch dieser Befehl ist nicht neu:
TYPE :Eins TYPE "+ TYPE :Zwei TYPE "=
Im Gegensatz zu PRINT beinhaltet TYPE keinen Zeilenvorschub. Die vier Druckbefehle werden so ausgeführt, daß alle Zeichen bzw. Variablenwerte in einer Zeile ausgedruckt werden.

Nun wird es komplizierter. Das Programm soll die Aufgabe ja auch grafisch darstellen. Die einfachste Möglichkeit, eine Addition zeichnerisch zu verdeutlichen, sind Linien.
Dazu setzen wir die Schildkröte erst einmal an einen bestimmten Startpunkt:
SETPOS [-80 20]
Mit SETPOS (Setze Position) kann die Schildkröte an jede beliebige Stelle gesetzt werden. Da die Position 0,0 in der Mitte des Bildschirms ist, sind Werte für links bzw. unterhalb der Mitte als negative Zahlen einzugeben. Probieren wir den Befehl aus.
Wie dumm: Die Schildkröte macht einen Strich vom Nullpunkt zur neuen Position. Das wollen wir aber nicht. Mit „Stift hoch" können wir erreichen, daß die Schildkröte bei den folgenden Bewegungen nicht zeichnet: **PENUP** (kurz PU).

Die Linie soll nach rechts gezeichnet werden. Also müssen wir die Schildkröte mit dem Kopf nach rechts stellen. Dafür gibt es zwei Möglichkeiten: **RIGHT 90** kennen wir schon. Dieser Befehl ist aber davon abhängig, wie die augenblickliche Position des Kopfes ist. Das kann ärgerlich sein. Universeller ist **SETH 90**. Wie schon bei RIGHT oder LEFT ist der Parameter als

Winkel einzugeben, wobei 0 und 360 den Kopf der Kröte nach oben stellen. −90 und 270 haben denselben Effekt: Der Kopf der Schildkröte steht nach links.

Ab jetzt soll die Kröte wieder zeichnen. Dazu muß der Stift wieder aufgesetzt werden. **PENDOWN** heißt diese Anweisung in Logo (kurz PD). Nun muß nur noch die Linie für die Zahl Eins gezeichnet werden. Das ist kein Problem: **FORWARD :Eins** ist uns schon bekannt. Die weiteren Zeichnungen für die zweite Zahl und das Ergebnis funktionieren im Prinzip genauso.

Hinweisen möchte ich noch auf die Positionsänderung zwischen der Linie für die Zahl Eins und der für die Zahl Zwei: Die Kröte bleibt auf derselben horizontalen Position und geht nur 20 Punkte tiefer. Dadurch werden die beiden Linien so gesetzt, daß das Ergebnis ablesbar ist.

```
TO ADDIEREN
MAKE "Ergeb :Eins + :Zwei
TYPE :Eins TYPE "+ TYPE :Zwei TY
PE "=
PENUP
SETPOS [-80 20] SETH 90
PENDOWN
FORWARD :Eins
PENUP RIGHT 90 FORWARD 20 LEFT 9
0 PENDOWN
FORWARD :Zwei
END
```

Das nötige Löschen des Bildschirms werden wir unabhängig von der folgenden Aufgabe benötigen. Daher fehlt es in dieser Routine und wird später im Hauptprogramm ausgeführt.

3.1.4 Subtrahieren

Diese Prozedur ist nun wirklich nicht mehr schwierig. Jeder sollte versuchen, sie alleine zu verwirklichen. Dabei soll mithilfe der gezeichneten Linien wieder das Ergebnis ablesbar sein.

```
TO SUBTRAHIEREN
MAKE "Ergeb :Eins - :Zwei
TYPE :Eins TYPE "- TYPE :Zwei TY
PE "=
PENUP
SETPOS [-80 20] SETH 90
PENDOWN
FORWARD :Eins
PENUP RIGHT 90 FORWARD 20 RIGHT
90 PENDOWN
FORWARD :Zwei
END
```

3.1.5 Multiplizieren

Für die Multiplikation müssen wir uns etwas anderes einfallen lassen; sie ist nicht ganz so einfach abzubilden.
Eine Multiplikation kann als Fläche dargestellt werden. Wir lassen die Schildkröte ein Rechteck malen mit den Seiten Eins und Zwei. Das Ergebnis ist dann die Fläche des Rechteckes.
Außer bei der Zeichenroutine unterscheidet sich auch die Multiplikation nicht von den beiden vorhergehenden Prozeduren. Neue Befehle brauchen wir nicht dafür.

```
TO MULTIPLIZIEREN
MAKE "Ergeb :Eins * :Zwei
TYPE :Eins TYPE "* TYPE :Zwei TY
PE "=
PENUP
SETPOS [-80 -20] SETH 0
PENDOWN
REPEAT 2 [FORWARD :Eins RIGHT 90
 FORWARD :Zwei RIGHT 90]
END
```

Wie wir sehen, ist diese Prozedur durch den Repeat-Befehl sogar sehr viel kürzer. Es kann nämlich da weitergezeichnet werden, wo sich die Kröte befindet.

Zu beachten ist noch, daß hier erstmals ein Rechenzeichen auftaucht, das anders geschrieben wird als normalerweise: Das Multiplikationszeichen ist der kleine Stern „*".

3.1.6 Dividieren

Hier stellt sich noch ein Problem: Das Ergebnis soll ganzzahlig sein. Es wird aber meist so sein, daß die erste zufällig gewählte Zahl nicht ganzzahlig durch die zweite teilbar ist. Welchen Ausweg gibt es? Nun, wir könnten natürlich solange neue Zahlen bestimmen, bis diese Bedingung erfüllt ist. Das kann unter Umständen etwas länger dauern, abhängig davon, welche Zahlen gerade ermittelt werden.
Geschickter ist es, zunächst das Produkt der beiden Zahlen zu ermitteln und es dann mit der Zahl Eins zu tauschen:
MAKE "hilf :Eins * :Zwei MAKE "Ergeb :Eins MAKE "Eins :hilf
Damit haben wir die Garantie, daß die gestellte Divisionsaufgabe immer aufgeht, wie man so schön sagt.

Um die Divisionsaufgabe in der Computersprache zu schreiben, muß für die Division der Schrägstrich (/) eingesetzt werden. Wir könnten natürlich auch einen Doppelpunkt schreiben, wie wir es oft auf dem Papier tun. Dies geht aber nur im Druckbefehl; wenn der Computer dividieren soll, muß der Schrägstrich eingesetzt werden.

Die grafische Darstellung erfolgt wieder als Fläche. Es ist natürlich nicht mehr so eindeutig wie bei den Strichrechenarten, was die Aufgabe ist und was das Ergebnis. Dennoch können wir es erst einmal so lassen.

```
TO DIVIDIEREN
MAKE "hilf :Eins * :Zwei
MAKE "Ergeb :Eins
MAKE "Eins :hilf
TYPE :Eins TYPE "/ TYPE :Zwei TY
PE "=
PENUP
SETPOS [-80 -20] SETH 0
PENDOWN
REPEAT 2 [FORWARD :Eins RIGHT 90
 FORWARD :Zwei RIGHT 90]
END
```

Diese Darstellung hat einen Nachteil: Die eine Seite des Rechtecks wird so lang, daß sie nicht mehr auf den Bildschirm paßt. Also müssen wir dafür sorgen, daß wieder die größte der drei Zahlen (Eins) als Fläche dargestellt wird. Das ist ja jetzt ganz einfach. Wir ändern also die Prozedur Dividieren.

```
TO DIVIDIEREN
MAKE "hilf :Eins * :Zwei
MAKE "Ergeb :Eins
MAKE "Eins :hilf
TYPE :Eins TYPE "/ TYPE :Zwei TY
PE "=
PENUP
SETPOS [-80 -20] SETH 0
PENDOWN
REPEAT 2 [FORWARD :Ergeb RIGHT 9
0 FORWARD :Zwei RIGHT 90]
END
```

Damit die Zeichnung nicht über den Rand geht, wenn die eine Seite länger als 80 Punkte ist, setzen wir den Startpunkt einfach etwas nach links, wie schon bei der Addition. Die zweite Seite ist nie länger als 50 Punkte, deshalb brauchen wir nicht weiter oben mit dem Zeichnen zu beginnen.

3.1.7 Fenster

Bei diesen beiden letzten Routinen ist es vielleicht – je nach Zufall – einigen bereits aufgefallen, was passiert, wenn die Schildkröte über den Rand malen soll. Geben wir einmal direkt ein: FORWARD 300; das führt auf jeden Fall über den Bildschirm hinaus. Beim Spectrum ist voreingestellt, daß die Schildkröte auf der gegenüberliegenden Seite wieder hereinkommt, wenn sie den Bildschirm verläßt. Diese Einstellung wird **WRAP** genannt.

Mit **WINDOW** kann erreicht werden, daß die Schildkröte außerhalb des Schirms weitermacht, ohne auf der anderen Seite wieder hereinzukommen: CLEARSCREEN WINDOW FORWARD 300 RIGHT 90 FORWARD 20 RIGHT 90 FORWARD 300 (und ENTER) zeigt, wie das funktioniert. In diesem Modus kann die Schildkröte maximal 32768 Punkte vom Nullpunkt entfernt werden.

Der dritte Modus heißt **FENCE**. Wenn wir diesen Modus einstellen und versuchen, über den Rand zu zeichnen, meldet Logo „Turtle out of bounds", die Schildkröte würde den Bildschirm verlassen. Der entsprechende Befehl wird überhaupt nicht ausgeführt.

3.2 Eingabe des Ergebnisses

Damit wir auch wirklich Rechnen üben, soll das Programm jetzt nach dem Ergebnis fragen bzw. zulassen, daß es eingegeben wird. Dazu brauchen wir den Eingabebefehl **READLIST**. In der globalen Variablen Eing wird der Wert gespeichert.

```
TO EINGABE
MAKE "Eing READLIST
END
```

Nun haben wir alle erforderlichen Einzelroutinen geschrieben. Mit **POTS** können wir alle bisher definierten Prozeduren anzeigen lassen, um zu prüfen, ob das auch stimmt.

```
TO EINGABE
TO DIVIDIEREN
TO MULTIPLIZIEREN
TO SUBTRAHIEREN
TO ADDIEREN
TO ZAHLEN
TO AUSWAHL

?
```


3.3 Das Hauptprogramm

Diese einzelnen Prozeduren werden zweckmäßigerweise durch eine weitere Prozedur in der erforderlichen Reihenfolge aufgerufen.
Die nötige Vergleichsfrage für die Verzweigung kennen wir schon.
Naheliegend wäre also folgende Prozedur Rechnen:

```
TO RECHNEN
SHOWTURTLE CLEARSCREEN
CLEARTEXT
AUSWAHL
ZAHLEN
IF :Typ = 1 [ADDIEREN]
IF :Typ = 2 [SUBTRAHIEREN]
IF :Typ = 3 [MULTIPLIZIEREN]
IF :Typ = 4 [DIVIDIEREN]
EINGABE
TYPE [Das Ergebnis ist] PR :Erge
b
IF :Ergeb = :Eing [PRINT "RICHTI
G.]
IF NOT ( :Ergeb = :Eing ) [PRINT
 "FALSCH.]
WAIT 200
END
```

Wir erinnern uns: wenn bei PRINT nur ein einzelnes Wort gedruckt werden soll, können diesem Wort auch die Anführungsstriche vorangestellt werden. Es könnte genausogut in eckigen Klammern stehen; dann hätten wir zwei geschachtelte Klammern.

Nur: wenn wir dies jetzt kaufen lassen (RECHNEN), druckt die Prozedur immer ''FALSCH.'' Warum?
Mit **PONS** erreichen wir, daß alle globalen Variablen ausgedruckt werden:

```
MAKE "Eing [1425]
MAKE "hilf "2898
MAKE "Ergeb "1425
MAKE "Zwei "15
MAKE "Eins "95
MAKE "Typ "3
```

Und da sehen wir es: In der Variablen Eing ist eine Liste mit einem Element gespeichert, in der Variablen Ergeb ein Wort. Listen werden durch eckige Klammern begrenzt; sie können mehrere Wörter enthalten, müssen es aber nicht. Ein Wort im Sinne von Logo wird immer nur durch ein vorangestelltes Anführungszeichen gekennzeichnet, das Ende des Wortes erkennt Logo an dem Leerzeichen. Daraus folgt auch, daß ein Logo-Wort kein Leerzeichen enthalten darf.

Für den Vergleich der Variableninhalte kommt es nicht nur darauf an, ob das Wort dasselbe ist, sondern auch der Typ der Variablen – Wort oder Liste – ist mit entscheidend.

Wir müssen den Vergleich also anders gestalten. Ein Wort kann Bestandteil einer Liste sein, daher können wir fragen, ob das Wort der Variablen Ergeb in der Liste Eing enthalten ist. Die Funktion hierfür heißt **MEMBERP**. Die Frage IF MEMBERP :Ergeb :Eing ... stellt dann fest, ob das in der ersten Variablen gespeicherte Wort in der zweiten enthalten ist.

3.3.1 Pausen

Die Pause verwenden wir hier wieder, um dem Benutzer des Programms die Möglichkeit zu geben, in Ruhe den Text zu lesen.
Dann sieht das Hauptprogramm so aus:

```
TO RECHNEN
SHOWTURTLE CLEARSCREEN
CLEARTEXT
AUSWAHL
ZAHLEN
IF :Typ = 1 [ADDIEREN]
IF :Typ = 2 [SUBTRAHIEREN]
IF :Typ = 3 [MULTIPLIZIEREN]
IF :Typ = 4 [DIVIDIEREN]
EINGABE
TYPE [Das Ergebnis ist] PR :Erge
b
WAIT 200
IF MEMBERP :Ergeb :Eing [PRINT "
RICHTIG.]
IF NOT MEMBERP :Ergeb :Eing [PRI
NT "FALSCH.]
WAIT 200
END
```


Kapitel 4: Verbesserungen

4.1 Kürzen des Programms

Der Arbeitsspeicher eines Computers ist begrenzt. Daher könnte es bei größeren Programmen dazu kommen, daß der Platz nicht ausreicht. Welche Möglichkeiten gibt es nun, ein Programm zu kürzen?

4.1.1 Vermeiden globaler Variablen

Die Variable Typ benutzen wir, um aus vier Möglichkeiten eine Auswahl zu treffen. Dazu lassen wir in der Prozedur Aufgabe einen zufälligen Wert zwischen 1 und 4 ermitteln. Im Hauptprogramm treffen wir dann die Entscheidung.
Das geht auch anders. Wir könnten ja eine Prozedur Auswahl definieren, die einen Parameter hat. Dieser Parameter wird dann durch den Zufallsbefehl aus Aufgabe gegeben, wenn wir die Prozedur vom Hauptprogramm Rechnen aufrufen lassen:

```
TO AUSWAHL :Typ
IF :Typ = 1 [ADDIEREN]
IF :Typ = 2 [SUBTRAHIEREN]
IF :Typ = 3 [MULTIPLIZIEREN]
IF :Typ = 4 [DIVIDIEREN]
END
```

Jetzt müssen wir noch das Hauptprogramm ändern. Dies tun wir am besten sofort, damit wir es nicht vergessen:
EDIT ''RECHNEN (und ENTER).
Es gibt in Logo einen hilfreichen Befehl, mit dem ganze Zeilen auf einmal gelöscht werden können. Dazu müssen wir aber sicherstellen, daß der Befehl Auswahl wirklich alleine in einer Zeile steht. Sollte das nicht der Fall sein, so gehen wir auf das erste Zeichen des nächsten Befehls und drücken dann

ENTER. Dadurch wird ein Wagenrücklauf in das Programm eingefügt. Nun gehen wir mit dem Cursor auf das A von Auswahl und drücken EX + Y. Die ganze Zeile verschwindet.
Genauso verfahren wir mit den vier IF-Zeilen im Programm. Diese sind ja jetzt überflüssig. Wenn sie gelöscht sind, fügen wir nur die neue Zeile **AUSWAHL 1 + RANDOM 4** ein.
Das Programm Rechnen müßte jetzt so aussehen:

```
TO RECHNEN
SHOWTURTLE CLEARSCREEN
CLEARTEXT
ZAHLEN
AUSWAHL 1 + RANDOM 4
EINGABE
TYPE [Das Ergebnis ist] PR :Erge
b
WAIT 200
IF MEMBERP :Ergeb :Eing [PRINT "
RICHTIG.]
IF NOT MEMBERP :Ergeb :Eing [PRI
NT "FALSCH.]
WAIT 200
END
```

Wir haben hier gesehen, daß es oft mehrere Wege gibt, ein Problem zu lösen. Der Weg, der die wenigsten globalen Variablen benötigt, ist am günstigsten, wenn wir Platz sparen wollen.

Die Prozedur Aufgabe ist jetzt überflüssig. Wir können einzelne Prozeduren löschen mit **ERASE "...**, wobei wir statt der Punkte den Namen der Prozedur einsetzen.

4.2 Etwas Farbe

Bis jetzt ist unser Programm nur auf die Funktion ausgelegt. Wir könnten aber etwas Farbe einbringen, um das Bild hübscher zu machen. Es ist allerdings auch so, daß richtig und sparsam eingesetzte Farben eine Information beinhalten kann, so daß für den Benutzer die Aufnahme der Bildschirminformation vereinfacht wird.

Es gibt eine Reihe von Farbbefehlen in Logo, die es uns erlauben, die Farben gezielt zu setzen.

Für den Grafikbildschirm sind dies drei Befehle:

SETBG n setzt die Hintergrundfarbe (BG für BackGround = Hintergrund), n darf Werte zwischen 0 und 7 annehmen.

SETPC n setzt die Vordergrundfarbe (PC für PenCil = Stift), wobei n ebenfalls Werte zwischen 0 und 7 annehmen kann.

SETBR n schließlich setzt die Randfarbe (BR für BordeR = Rand), wieder mit Werten zwischen 0 und 7.

Für den Textbildschirm werden Vorder- und Hintergrundfarbe über einen Befehl gesteuert **SETTC [n m]**, wobei n für die Hintergrundfarbe und m für die Vordergrundfarbe steht. Auch hier sind wieder die Werte 0 bis 7 erlaubt.

Interessant ist hier noch die Möglichkeit, eine Figur oder einen Text zu erstellen, während Vorder- und Hintergrundfarbe gleich gesetzt sind. Wenn die Grafik fertig ist, wird die Hintergrundfarbe geändert und die Figur erscheint schlagartig. Dies geht nicht bei allen Computern; viele Modelle akzeptieren nicht gleiche Farben für Papier und Stift.

Die Routine Hintergrund werden wir mit drei Parametern definieren, damit sie universell nutzbar ist.

```
?

TO HINTERGRUND :paper :ink :bord
er
SETBR :border
SETPC :ink
SETBG :paper
SETTC SE :paper :ink
END
```

Wir dürfen natürlich nicht vergessen, diese Routine in Rechnen einzubauen. Vorher setzen wir ein CLEARSCREEN, damit eine alte Aufgabe vom Schirm gelöscht wird.
(zu SE siehe nächste Seite)

4.3 Etwas Ton

Es wäre auch ganz nett, wenn das Programm uns darauf aufmerksam machte, daß jetzt eine Eingabe verlangt wird. Dazu kann man sehr schön Töne einsetzen. Hinter den Ausdruck der Aufgabe (bzw. vor die Routine Eingabe) setzen wir also einen Tonbefehl:

SOUND [laenge hoehe].

Je nachdem, ob das Ergebnis richtig oder falsch ist, werden unterschiedliche Melodien gespielt. Da jetzt mehrere Befehle auszuführen sind, ist es zweck-

mäßiger, zwei neue Prozeduren Wahr und Falsch zu definieren. Man könnte natürlich diese Liste von Befehlen auch direkt in Rechnen einbauen. Dadurch wird diese Prozedur aber länger und unübersichtlicher.

```
TO RECHNEN
HINTERGRUND 6 Ø 6
SHOWTURTLE CLEARSCREEN
CLEARTEXT
ZAHLEN
AUSWAHL 1 + RANDOM 4
SOUND [Ø.5 12] EINGABE
TYPE [Das Ergebnis ist] PR :Erge
b
WAIT 2ØØ
IF MEMBERP :Ergeb :Eing [WAHR]
IF NOT MEMBERP :Ergeb :Eing [FAL
SCH]
WAIT 2ØØ
END
```

Die Prozedur Wahr übernimmt jetzt die Aufgabe, einen Kommentar zu drucken und eine aufsteigende Tonfolge zu spielen:

```
TO WAHR
PRINT [Das war richtig.]
SOUND [Ø.5 Ø]
SOUND [Ø.5 4]
SOUND [Ø.5 7]
SOUND [Ø.5 12]
END
```

Die Prozedur Falsch soll eine absteigende Tonfolge spielen und außer dem Kommentar auch noch das richtige Ergebnis drucken.
Wenn wir bisher in einer Zeile sowohl eine Liste (Inhalt der eckigen Klammer) als auch einzelne Wörter (z.B. Variableninhalte) ausdrucken lassen wollten, haben wir mehrere TYPE-Befehle hintereinander gesetzt.

Logo hat dafür aber eine Funktion: **SENTENCE** macht aus der folgenden Aufreihung von Listen und/oder Wörtern eine einzige Liste. Dieses Wort kann mit SE abgekürzt werden.

```
TO FALSCH
PRINT [Das war falsch.]
SOUND [Ø.5 Ø]
SOUND [Ø.5 -5]
SOUND [Ø.5 -8]
SOUND [Ø.5 -12]
END
```


4.4 Fehlerzähler

Bis jetzt haben wir jedesmal die Prozedur Rechnen direkt aufgerufen. Es ist schade, daß nirgendwo gezählt wird, wie oft sie aufgerufen wird und wieviele Fehler wir gemacht haben. Das kann leicht geändert werden. Wir schreiben eine Prozedur, die Rechnen beispielsweise 10 mal aufruft und zählt, wie oft die Prozedur Falsch aufgerufen wurde. Am Schluß wird das Ergebnis dieser Zählung dann ausgegeben.

Neue Befehle brauchen wir dafür nicht. Vom vorhandenen Programm müssen wir nur die Routine Falsch noch ändern; wir brauchen ja eine neue Variable.

```
TO TEST :Anzahl
MAKE "Fehler Ø
REPEAT :Anzahl [RECHNEN]
TEXTSCREEN CLEARTEXT
PRINT [Der Rechen - Test ist zu
Ende.]
PRINT []
TYPE SE [Du hast von  ] :Anzahl
PRINT [ Aufgaben]
PRINT SE :Fehler [ falsch geloes
t.]
END


TO FALSCH
PRINT [Das war falsch.]
SOUND [Ø.5 Ø]
SOUND [Ø.5 -5]
SOUND [Ø.5 -8]
SOUND [Ø.5 -12]
MAKE "Fehler :Fehler + 1
END
```

Kapitel 5: Statistik

Aufgrund seiner hervorragenden Grafikbefehle eignet sich Logo sehr gut für die grafische Darstellung von Daten, zum Beispiel Statistiken.

5.1 Kreis- oder Kuchendiagramme

Gerade Kreisdiagramme sind in BASIC recht schwierig zu programmieren, weil ein Algorithmus (Lösungsweg) gefunden werden muß, der es ermöglicht, Prozentwerte in Winkel umzusetzen. Der Winkel kann in den gängigen BASIC-Dialekten nicht direkt in Grafik umgesetzt werden. Die Umrechnung in die normalen Grafik-Befehle ist dann oft auch etwas ungenau.

5.1.1 Kreise zeichnen

Logo hat keinen direkten Befehl, um Kreise zu zeichnen. Eine solche Prozedur ist aber recht einfach zu erstellen, wenn wir erst einmal die richtige Grundidee haben.

Der Gesamtinnenwinkel eines Kreises ist 360 Grad. Wenn wir die Schildkröte also 360 Mal um einen Schritt vorwärts schicken und dann um einen Winkel von 1 Grad drehen lassen, müßten wir am Ausgangspunkt ankommen:

```
TO KREIS
REPEAT 360 [FORWARD 1 RIGHT 1]
END
```

Die Prozedur arbeitet einwandfrei, aber leider braucht sie etwas lange. Wenn wir nun jeweils zwei Schritte vorwärts gehen und einen Winkel von 2 Grad nehmen, müssen wir nur 180 Wiederholungen einsetzen:

```
TO KREIS
REPEAT 180 [FORWARD 2 RIGHT 2]
END
```

Auch das gibt einen sauberen Kreis. Dasselbe mit den Werten 90 für die Wiederholung, 4 für vorwärts und Winkel arbeitet auch noch mit einem guten Ergebnis:

```
TO KREIS
REPEAT 90 [FORWARD 4 RIGHT 4]
END
```

Aber die Sache hat einen Haken: der Durchmesser ist festgelegt (115 Pixels = Bildpunkte). Was machen wir, wenn wir variable Kreise zeichnen möchten? Wir brauchen einen Eingabewert für die Prozedur, der uns entweder den Umfang oder aber (noch besser) den Radius vorgibt. Die Vorgabe des Radius' ist deshalb besser, weil es dann einfacher zu bestimmen ist, wohin wir die Schildkröte vor dem Kreiszeichnen setzen müssen, damit der Mittelpunkt des Kreises die Home-Position (der Koordinatenursprung) ist. Um den Kreis aufzuteilen, brauchen wir ja später den Mittelpunkt, denn die „Tortenstücke" werden durch Linien vom Kreismittelpunkt zum Umfang (Radien) abgeteilt.

```
TO KREIS :Radius
REPEAT 36 [FORWARD 2 * :Radius /
 36 * 3.14 RIGHT 10]
END
```

Ganz einfach, nicht? Wenn man erst einmal darauf gekommen ist!

5.1.2 Schrift im Grafikbildschirm

Solche Statistiken wollen beschriftet sein, sonst sind sie wertlos. Sie müssen zum einen das Datum, zum anderen die Art der dargestellten Daten enthalten, wobei eine Aufschlüsselung erfolgen muß, was die Grafik darstellen soll und in welchem Maßstab.

Beim Sinclair-BASIC wird zwischen Schrift und Grafik im Grunde nicht unterschieden. Bei Logo können wir – wegen der gelungenen Anpassung an den Spectrum – auch einfach in den Grafikbildschirm hineinschreiben. Wir müssen nur vorher den Cursor an die entsprechende Stelle bringen:

SETCUR [spalte zeile]

BASIC-Kennern wird auffallen, daß die Angabe andersherum ist als gewohnt. Dies ist aber die bei allen Rechnern übliche Angabereihenfolge: erst

den Wert für die waagerechte Position, dann den für die senkrechte. Oben links befindet sich die Position 0 0, die Zeilen werden also von oben nach unten gezählt.

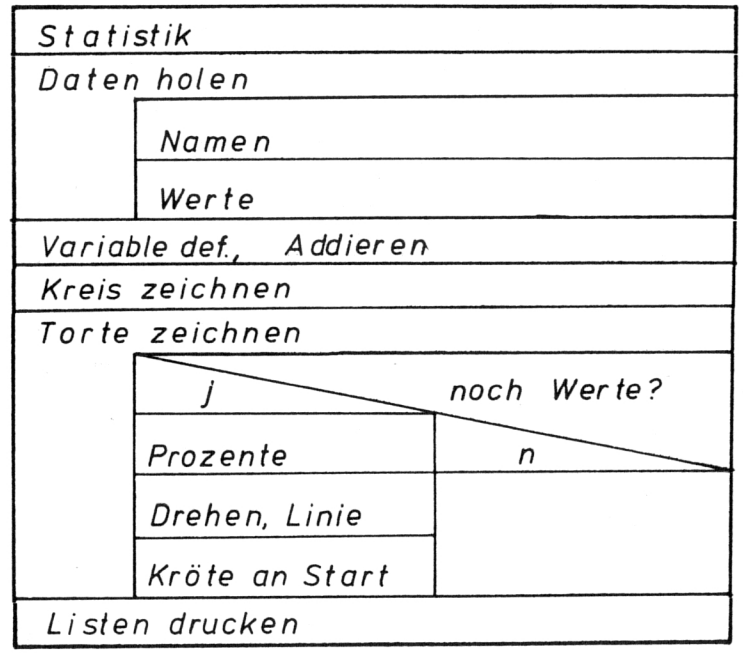
Probieren wir es einfach mal aus:

KREIS 50 SETCUR [2 3] PRINT [15.04.85]

Oha, er druckt nur bis zum zweiten Punkt. Die Datumsangabe in der Liste wird von Logo als Zahl interpretiert; wir sollten daran denken, daß Logo einerseits keinen Unterschied macht zwischen Zahl und Wort, und andererseits im Englischen das Dezimalkomma ein Dezimalpunkt ist. Ein Ausweg aus dieser Misere ist TYPE ''15.04 TYPE''. TYPE ''85.

5.2 Ein Statistik-Programm

Wir wollen unsere neuen Kenntnisse in ein kleines Statistik-Programm umsetzen. Am besten ist es, wenn wir uns den Ablauf dieses Programms kurz skizzieren:



Nun können wir uns der Reihe nach um die einzelnen Programmteile kümmern.

5.2.1 Dateneingabe

Wir könnten natürlich für jede Kuchengrafik die Daten selbst umrechenen und dann die Grafikbefehle direkt geben. Aber – warum sollen wir rechnen, wenn der Computer doch soviel schneller rechnet?
Mit anderen Worten: wir brauchen eine Prozedur, in der wir die Daten eingeben können, so daß das Programm sie später benutzen kann.

Den Eingabebefehl READLIST kennen wir schon. Er liefert eine Liste ab. Diese Liste hätten wir gerne als Wort (im Sinne von Logo). Dafür gibt es eine Funktion, **WORD**, die allerdings zwei Eingaben verlangt. Diese Klippe zu umschiffen, ist nicht weiter schwierig: als zweites Argument geben wir ein leeres Wort ein.

Einmal werden wir Eingabe brauchen, um den Namen der Daten zu speichern, und beim zweiten Mal werden wir die Anzahl (absolut) speichern.

Beide Datengruppen werden wir in je einer Liste ablegen; zusammengehörende Daten haben dann die gleiche Position in den beiden Listen. Damit die Listen nicht auch Listen enthalten, machen wir aus der Eingabe gleich ein Wort.
Es ist sinnvoll, bei der Eingabe von Namen und Werten einen erklärenden Text auszugeben. Dies übernehmen die beiden folgenden Routinen, die auch die Eingaberoutine jeweils aufrufen.

Die Eingabe-Werte wollen wir als Liste zusammenfassen. Die Funktion SE kennen wir ja schon.

Diese – und die notwendige Initialisierung der Variablen – fassen wir in einer weiteren Prozedur Daten zusammen. Wir wollen uns hier auf 5 Daten beschränken, theoretisch ist die Anzahl frei wählbar.

Damit wäre die Eingabe der Daten fertig. Nun müssen die Daten noch ausgewertet werden.


```
TO EINGABE
MAKE "Wort FIRST READLIST
END


TO DATEN
MAKE "Namen []
MAKE "Werte []
REPEAT 5 [NAMEN WERTE]
END


TO NAMEN
TYPE [Bitte den naechsten Namen
ein - geben:]
EINGABE
MAKE "Namen SENTENCE :Namen :Wor
t
END


TO WERTE
TYPE [Bitte den naechsten Wert e
in - geben:]
EINGABE
MAKE "Werte SENTENCE :Werte :Wor
t
END
```

5.2.2 Verarbeitung der Daten

Zunächst einmal müssen die Zahlenwerte addiert werden, um die Prozentwerte berechnen zu können. Dafür benutzen wir eine weitere Funktion, die uns Logo stellt: **FIRST** (erstes). Damit können wir nacheinander die einzelnen Werte aus der Liste holen und addieren.
Anmerkung: Es wäre ein sinnvolle Ergänzung, bei der Eingabe der Zahlenwerte sicherzustellen, daß es auch Zahlen sind. Die Funktion dafür heißt **NUMBERP** und hat als Argument ein Wort.
Die Funktion **BUTFIRST** (but = außer) verkürzt die Eingabeliste um das erste Element. Mit **EMPTYP** (aus empty = leer und Typ) fragen wir ab, ob in der Liste noch ein Wort steht.

```
TO ADDIEREN :Werte
IF EMPTYP :Werte [STOP]
MAKE "Summe :Summe + FIRST :Wert
e
ADDIEREN BUTFIRST :Werte
END
```

5.2.3 Die Grafik

Nun fehlt uns noch die Grafik zu diesem Programm. Zunächst brauchen wir einen Kreis und dann eine Zeichenroutine, die die entsprechenden Radien im Winkel der Prozentwerte zeichnet. Den Kreis haben wir schon.

Unsere Radien-Zeichen-Prozedur muß zunächst feststellen, wieviel Prozent der erste Wert unserer Werteliste ist. Danach muß dieser Prozentwert als Anteil von 360 ermittelt werden. Für uns ist das nichts anderes als der Dreisatz (auf den sich im kaufmännischen Bereich alles zurückführen läßt).

```
TO PROZENT :Wert
MAKE "Prozent 100 / :Summe * :We
rt
END

TO TORTE :Werte
IF EMPTYP :Werte [STOP]
PENUP SETPOS [0 0] PENDOWN
PROZENT FIRST :Werte
RIGHT :Prozent * 3.6
FORWARD :Radius
TORTE BUTFIRST :Werte
END
```

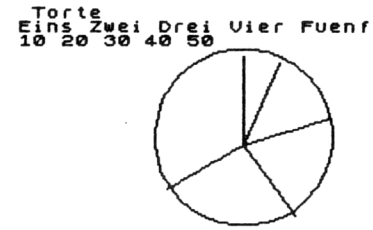
5.2.4 Ausgeben der Texte

In dem Programm Statistik fassen wir alle Prozeduren zusammen und setzen auch die notwendigen Variablen Radius und Summe. Summe kann nicht von Addieren auf 0 gesetzt werden, weil Addieren sich selbst aufruft und somit Summe nur den letzten Wert erhielte.

```
TO STATISTIK
TEXTSCREEN
DATEN
MAKE "Summe 0
ADDIEREN :Werte
MAKE "Radius 50
PENUP SETX - :Radius PENDOWN
KREIS :Radius
PENUP HOME PENDOWN
TORTE :Werte HIDETURTLE
SETCUR [1 1]
PRINT [Torte]
PRINT :Namen
PRINT :Werte
END
```

Wenn wir nun Statistik aufrufen, wird der Bildschirm gelöscht und sinnvollerweise auf Text umgeschaltet. Dadurch ist Platz genug für alle fünf Eingaben, so daß wir bei der Dateneingabe eine Kontrolle haben.

Leider müssen wir bei der Tortengrafik feststellen, daß unser Kreis doch nicht so genau ist, wie wir ihn brauchten, damit die Radien immer auf dem Umfang auskommen. Dies hat dann aber zur Folge, daß die Kreis-Routine länger braucht. Der Durchmesser des Kreises aus der ersten Routine Kreis beträgt 115 Bildpunkte. Wir könnten natürlich diesen hier einsetzen.



5.2.5 Variablennamen und Prozedurnamen

Gerade in diesem Programm können wir gut sehen, daß Logo zwischen lokalen und globalen Variablen und Prozeduren auch dann unterscheidet, wenn sie dieselben Namen haben. In Daten lassen wir die globalen Variablen Werte und Namen definieren. Die Prozedur Addieren benutzt als Eingabe ebenfalls die Variable Werte, die allerdings lokal ist. Die globale Variable wird durch Addieren nicht verändert.
Dies können wir leicht nachprüfen, indem wir uns mit **PONS** die Namen aller globalen Variablen und ihres derzeitigen Inhalts ausgeben lassen.
Wir sehen aber auch am Ende des Statistikprogramms am Ausdruck der Variablenwerte, daß noch alle Daten so enthalten sind, wie sie eingegeben wurden.

5.2.6 Der Text im Bild

Wir haben einfach die beiden Listen in das Bild gedruckt. Da nun die Angaben nicht alle gleichlang sind, steht nicht jeder Wert unter seinem zugehörigen Namen. Eine Möglichkeit der Ausrichtung von Texten werden wir jetzt ausprobieren. Wir brauchen dazu die Funktion **CHAR**, die eine Zahl zwischen 0 und 255 als Argument hat. Als Ergebnis liefert sie das Zeichen, das diesem ASCII-Code zugeordnet ist.

ASCII steht für American Standard Code for Information Interchange, Amerikanischer Standard-Zeichencode für den Datenaustausch. Der ASCII-Code hat nur 128 Zeichen (von 0 bis 127), weitere Zeichen sind rechnerspezifische Zeichen. Im Spectrum-Handbuch ist die vollständige Codetabelle enthalten. Die Codes 0 - 31 sind nichtdruckbare Steuerzeichen, die eigentlichen Schriftzeichen beginnen mit 32 (Leerzeichen).

Es gibt auch die Möglichkeit, eckige Klammern und Anführungszeichen in solchen Texten auszudrucken. Diese Zeichen werden von Logo ja normalerweise als Begrenzer für Wörter oder Listen benutzt und nicht mit ausgedruckt. Dafür können wir aber die Funktion CHAR n benutzen, wobei n in der Regel einen Wert zwischen 32 und 127 haben wird.

Wir brauchen außerdem eine Funktion, die uns die Länge der einzelnen Wörter ausgibt: **COUNT** hat als Argument ein Wort oder eine Liste und gibt die Anzahl der Elemente an. Bei Wörtern als Eingabe zu COUNT erhalten wir also die Anzahl der Buchstaben, bei Listen die Anzahl der Wörter/ Listen, die Elemente dieser Liste sind.

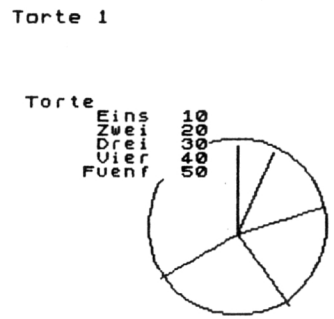
Mithilfe dieser beiden Funktionen können wir uns eine Funktion Fueller definieren, die die einzelnen Wörter rechtsbündig ausgibt. Als Eingabe für diesen Befehl brauchen wir die beiden Listen Namen und Werte.

```
TO FUELLER :Anzahl
REPEAT :Anzahl [TYPE CHAR 32]
END


TO AUSDRUCK :Namen :Werte
IF EMPTYP :Namen [STOP]
FUELLER 10 - COUNT FIRST :Namen
TYPE FIRST :Namen
TYPE CHAR 32
FUELLER 3 - COUNT FIRST :Werte
PRINT FIRST :Werte
AUSDRUCK BUTFIRST :Namen BUTFIRS
T :Werte
END


TO STATISTIK
TEXTSCREEN
DATEN
MAKE "Summe 0
ADDIEREN :Werte
MAKE "Radius 50
PENUP SETX - :Radius PENDOWN
KREIS :Radius
PENUP HOME PENDOWN
TORTE :Werte HIDETURTLE
SETCUR [0 0]
AUSDRUCK :Namen :Werte
END
```


Mit dem geänderten Programm erhalten wir dieses Bild:



Der Text erscheint zu weit rechts, er zerstört teilweise die Grafik. Das können wir in Ausdruck beim ersten Aufruf von Fueller ändern, indem wir dort statt 10 einen kleineren Wert nehmen, von dem dann die Anzahl der Buchstaben abgezogen wird.

Kapitel 6: Der Speicher

6.1 Der Arbeitsspeicher

6.1.1 NODES

Der Arbeitsspeicher ist aufgrund des dort befindlichen Logo-Interpreters nicht mehr voll für uns verfügbar. Wenn nach dem Laden der Befehl **PRINT NODES * 5** gegeben wird, erhalten wir den ungefähren Wert an freien Bytes für Programme.

6.1.2 ERASE

Wenn wir nun – wie im vorigen Kapitel – an einigen Prozeduren etwas herumprobiert haben, ist der Arbeitsspeicher mit überflüssigen Prozeduren und vielleicht auch Variablen gefüllt. Die können wir wieder löschen. Dafür gibt es den Befehl **ERASE**, gefolgt von dem/den Namen der zu löschenden Prozeduren. Wenn es mehrere Namen sind, erfolgt die Eingabe in Form einer Liste, ansonsten als Wort.

Probieren wir das doch einfach mal aus:
TEXTSCREEN POTS (ENTER) druckt alle definierten Prozeduren. Auf Text schalten wir, damit alles auf einmal auf den Schirm geschrieben werden kann.
ERASE [AUSDRUCK FUELLER] POTS
ergibt alle Prozeduren mit Ausname der eben mit ERASE gelöschten.

Geben wir nun ein PRINT NODES * 5, so erhalten wir einen höheren Wert als vorher. Trotzdem ist noch Platz verschwendet.

6.1.3 RECYCLE

Mit dem Befehl **RECYCLE** können wir Logo veranlassen, den Müll aus dem Speicher zu entfernen, also den Arbeitsspeicher neu zu organisieren. In Englisch nennt man das Garbage Collection. Der Arbeitsspeicher enthält

nämlich außer den Prozeduren und den Variablen noch viel mehr: (fast) alles, was wir getippt haben. Dies können wir mit PRINT **.CONTENTS** ja mal feststellen.

Dies ist aber nicht (mehr) nötig. Mit RECYCLE erreichen wir, daß zumindest das gelöscht wird, was mit nicht mehr vorhandenen Prozeduren zusammenhängt, der Speicher also so organisiert wird, daß möglichst viel Platz frei ist.

6.2 Speichern der Prozeduren auf Massenspeichern

Damit wir mit **ERALL** (lösche alles) den Arbeitsspeicher wirklich aufräumen können, sollten wir unsere Programme vorher auf Cassette oder Cartridge ablegen.

6.2.1 Wahl des Massenspeichers

Logo erlaubt auch die Verwendung der Microdrives. Ob wir auf Cassette oder Microdrive speichern, wählen wir mit **SETDRIVE**, gefolgt von einer Zahl zwischen 0 und 8. Mit 0 wird der Cassettenrecorder angesprochen, mit 1 - 8 die Microdrives 1 bis 8.

6.2.2 Speichern der Daten

Das Sichern der Prozeduren und Variablen erfolgt mit dem Befehl **SAVE**, der zwei Angaben benötigt: Die erste Angabe ist der Filename, unter dem es abgelegt werden soll (Programmname). Diese Eingabe ist ein Wort. Die zweite Eingabe ist die Liste der Prozeduren, die unter diesem Namen abgelegt werden soll.
SAVE ''stat [STATISTIK DATEN NAMEN WERTE KREIS ...]
Damit wir keine wichtige Prozedur vergessen, ist es günstig, vorher mit TEXTSCREEN POTS alle Namen ausgeben zu lassen.
Es geht auch, mit **SAVEALL ''name** alle Prozeduren abzuspeichern. Dann sollten wir vorher die überflüssigen Programm(teile) mit ERASE ''name gelöscht und mit RECYCLE den Speicher neu organisiert haben.

Bildschirminhalte können mit **SAVESCR ''name** auf Cassette oder Cartridge gesichert werden.

Bei der Verwendung von Microdrives ist zu beachten, daß nicht zwei Files denselben Namen haben dürfen. Soll also ein geändertes Programm neu gespeichert werden, so muß vorher das alte File auf der Cartridge gelöscht werden. Der Befehl hierzu heißt **ERASEFILE**, gefolgt von dem Namen in Form eines Wortes.

Im allgemeinen darf beim Spectrum der Filename 10 Zeichen lang sein. Logo fügt an die von uns vergebenen Filenamen noch LOG (für Prozeduren) oder SCR (für Bildschirminhalte) an. D.h. wir haben nur 7 Buchstaben für den Namen zur Verfügung.

6.2.3 Sehr lange Programme

Wenn wir mal ein Programm schreiben, das beim besten Willen nicht mehr ganz in den Arbeitsspeicher paßt (bei der Ausführung eines Programmes braucht der Logo-Interpreter noch Platz für eigene Systemwerte), bleibt uns nichts anderes übrig, als ein Programm so aufzubauen, daß es Teile nachlädt, die gebraucht werden und Teile löscht, die es nicht mehr braucht. Es kann ja jede Prozedur als eigenes File auf dem Datenträger gespeichert werden. Hierbei empfiehlt sich natürlich die Verwendung der Microdrives, weil sie – im Gegensatz zum Cassettenrecorder – vom Computer selbst gesteuert werden können. Außerdem haben sie – im Verhältnis zum Preis – wirklich kurze Ladezeiten, was die Verwendung des Recorders in solchen Fällen unzweckmäßig scheinen läßt.

Kapitel 7: Der Drucker

Es ist natürlich schön, wenn man seine Programme auch über den Drucker listen lassen kann.

Der Druck wird eingeschaltet mit **PRINTON**, und mit **PRINTOFF** können wir ihn wieder ausschalten. Jeder Druckbefehl geht nach PRINTON zusätzlich an den Drucker, auch Systemmeldungen, nicht aber Befehlseingaben.

7.1 ZX-Printer, Alphacom 32, Seikosha GP 50 S

Diese Drucker haben das nötige Interface (Schnittstelle) für den Spectrum bereits eingebaut. Sie werden einfach an den Busstecker angesteckt.
Diese Drucker können den Befehl **COPYSCREEN** verarbeiten, weil die nötige Software zur Steuerung bereits eingebaut ist.
Dieser Befehl kopiert den Bildschirminhalt auf den Drucker. Andere Drucker als die obengenannten brauchen dafür eine zusätzliche Routine.

7.2 Drucker am Interface 1

Über den RS232-Anschluß des Interface 1 kann jeder Drucker mit serieller Schnittstelle angesprochen werden. Es muß nur vom BASIC aus der Textkanal des Interface 1 auf Strom 3 (der immer für LPRINT – Ausgabe auf Drucker benutzt wird) gelegt werden. Dies kann auch nach dem Laden von Logo noch erfolgen: BYE verläßt den Logo-Interpreter, OPEN #3,''T'' öffnet den Druckerkanal und RUN startet den Logo-Interpreter wieder.
Wenn der Drucker häufig gebraucht wird, ist dem Logo-Programm vielleicht am besten eine kleine Laderoutine voranzustellen, die diese Arbeit erledigt und anschließend Logo lädt.

Da Logo sehr häufig die eckigen Klammern benötigt, die im deutschen Zeichensatz auf den Buchstaben Ä und Ü liegen, sollten – falls möglich – auch die Steuerzeichen für das Umschalten auf den englischen Zeichensatz von der Laderoutine aus gesendet werden. Hinweise dazu finden sich im Druckerhandbuch.

7.3 Drucker über Centronics-Schnittstelle

Die häufigste Art, den Drucker anzusprechen, ist eine parallele Schnittstelle. Leider gibt es so viele verschiedene Centronics-Schnittstellen für den Spectrum, daß nicht sichergestellt werden kann, ob diese mit Logo zusammen arbeiten können. Einige davon haben die nötigen Steuerprogramme im Interface eingebaut und müssen nur mit einem Befehl initialisiert werden, andere benötigen zusätzlich Steuersoftware im RAM. Logo erlaubt das Laden und starten von Maschinencode (**.BSAVE**, **.BLOAD** UND **.CALL**). Schlimmstenfalls liegt dieser Code in einem Bereich, den Logo braucht. Ist das der Fall, so kann dieser Drucker nicht mit Logo zusammen eingesetzt werden, es sei denn, das Drucker-Steuerprogramm wird so umgeschrieben, daß es an einer anderen Speicherstelle stehen kann.

7.4 Der COPY-Befehl

Wichtig: den Befehl **COPYSCREEN** können nur die dem Sinclair-Drucker ähnlichen Drucker ausführen. Die anderen Drucker benötigen dafür immer eine spezielle Maschinencode-Routine, die sich von der im Spectrum-ROM unterscheidet. Wenn sich diese mit Logo „verträgt", d.h. nicht denselben Speicherbereich belegt, kann die Hardcopy über einen Maschinencodeaufruf von Logo aus durchgeführt werden.

Sollte dies nicht gehen, bleibt nur eins: Bildschirm abspeichern (**SAVESCR** ...) und später von einem gesonderten Programm auf den Drucker bringen lassen.

Kapitel 8: Grafik und Mathematik

Logo bietet – wie wir im Anhang sehen können – eine Fülle von mathematischen Funktionen. Außerdem haben wir sehr viele Grafikmöglichkeiten. Dadurch bietet es sich geradezu an, mathematische Probleme in Grafik umzusetzen.

8.1 Koordinatenkreuz

Wenn wir mit CLEARSCREEN den Grafikbildschirm ansprechen, befindet sich die Schildkröte immer in der Mitte, mit dem Kopf nach oben. Wenn wir noch berücksichtigen, daß der Spectrum horizontal 256 Punkte und vertikal 176 Punkte für Grafik bietet, ist es ganz einfach, ein Kreuz zu zeichnen:

```
TO KREUZ
CLEARSCREEN
PENUP HOME PENDOWN
SHOWTURTLE
RIGHT 90 FORWARD 256
LEFT 90 FORWARD 176
HIDETURTLE PENUP
END
```

Das **PENDOWN** am Anfang und **PENUP** am Ende sollten wir uns angewöhnen, damit wir immer von denselben Voraussetzungen ausgehen können, wenn wir die Schildkröte brauchen. **HOME** stellt sicher, daß die Schildkröte sich immer im Ursprung mit dem Kopf nach oben befindet.

8.2 Maßstab

Zu einer „ordentlichen" Zeichnung gehört ein Maßstab im Koordinatenkreuz. Diesen zu zeichnen ist etwas langwieriger, aber es treten keine neuen Probleme auf.

Mit SETPOS können wir die Schildkröte überall positionieren, so daß der geeignetste Punkt für den Start direkt angesprungen werden kann.

```
TO MASSSTAB
SHOWTURTLE
PENUP SETPOS [-2 -80] SETH 90
REPEAT 17 [PENDOWN FORWARD 4 PEN
UP BACK 4 LEFT 90 FORWARD 10 RIG
HT 90]
PENUP SETPOS [-120 2] SETH 180
REPEAT 25 [PENDOWN FORWARD 4 PEN
UP BACK 4 LEFT 90 FORWARD 10 RIG
HT 90]
HIDETURTLE PENUP HOME
END
```

8.3 Rand

Bei einer Grafik dieser Art sieht – vor allem auf dem Drucker – ein Rand hübsch aus. Der Rand ist ein Rechteck. Wir müssen nur an der richtigen Stelle anfangen:

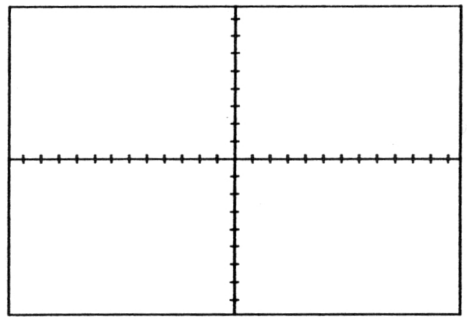
```
TO RAND
PENUP SETPOS [-128 -88]
PENDOWN
FORWARD 175 RIGHT 90
FORWARD 255 RIGHT 90
FORWARD 175 RIGHT 90
FORWARD 255
PENUP HOME
END
```

8.4 Papier

Da wir diese drei Prozeduren sicher öfter brauchen werden, ist es günstig, sie in einer zusammenzufassen. In der neuen Prozedur Papier werden dann auch gleich die Farben gesetzt: **SETPC** setzt die Stiftfarbe, **SETBG** setzt die Hintergrundfarbe und **SETBR** setzt die Randfarbe:

```
TO PAPIER
SETBR 6 SETBG 7 SETPC 0
KREUZ MASSSTAB RAND
END
```

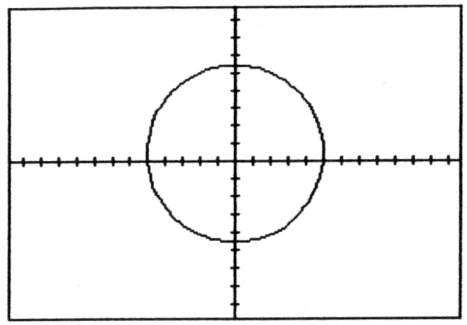

Das Ergebnis nach dem Aufruf Papier sieht dann so aus.



Papier

8.5 Kreise

Wir hatten uns schon mit Kreisen beschäftigt. Dabei war uns aufgefallen, daß unser Kreis offensichtlich nicht so ganz genau war; denn die Radien waren oben zu kurz und unten zu lang. Nun können wir mithilfe unseres Koordinatenkreuzes und der gekürzten Prozedur Kreis feststellen, wie groß der Fehler beim Zeichnen des Kreises ist:



Kreis 50 auf Papier

Nun sehen wir ganz klar: Der Kreis an sich ist richtig, nur ist er oben etwas zu lang und um die gleiche Anzahl Punkte unten zu kurz. Wenn wir Kreis mit verschiedenen Werten aufrufen, werden wir sehen, daß der „Fehler" vom Radius abhängig ist. Wir müßten die Schildkröte vor dem Zeichnen des Kreises also nicht nur nach links (**SETX**) verschieben, sondern um einen entsprechenden Wert auch nach unten (**SETY**).

```
TO KREIS :Radius
PENUP SETX - :Radius
SETY - :Radius / 10
SETH 0 PENDOWN
REPEAT 36 [FORWARD 2 * :Radius /
 36 * 3.14 RIGHT 10]
END
```

Wenn wir jetzt mehrere Kreise ausprobieren, stellen wir fest, daß zwar immer noch nicht alle Kreise ganz genau liegen; aber der Fehler ist sehr viel kleiner geworden und längst nicht mehr so störend.

8.6 Geometrie

Kreise mit dem Mittelpunkt im Koordinatenursprung sind sicher nicht das, was wir am häufigsten brauchen. Damit wir unser Papier mit anderen geometrischen Figuren und Kreisen mit anderen Mittelpunkten als dem Nullpunkt füllen können, brauchen wir neue Routinen.

8.6.1 Der Kreis

Zunächst ändern wir die Definition für die Prozedur Kreis so ab, daß wir nun auch den Mittelpunkt vorgeben können. Dazu brauchen wir zunächst zwei weitere Parameter für den Befehl, nämlich X und Y für die Postion. Dann müssen wir die beiden Befehle SETX und SETY anpassen, so daß sie jetzt in entsprechender Entfernung zu X und Y gezeichnet werden. Die (lokale) Varibale X ist dabei von der Positionsvariablen x zu unterscheiden. Logo tut das auch. Die waagerechte Position der Schildkröte können wir mit **PRINT XCOR** erfahren. Dasselbe gilt (natürlich) für Y.

In der Kreis-Prozdur setzen wir den Mittelpunkt auf den Bildschirm. Dazu verwenden wir den Befehl **DOT**, der die Koordinaten des Punktes als Liste verlangt (s.u.). DOT verändert die Position der Schildkröte nicht.


```
TO KREIS :Radius :X :Y
PENUP DOT SE :X :Y
SETX :X - :Radius
SETY :Y - :Radius / 10
SETH 0 PENDOWN
REPEAT 36 [FORWARD 2 * :Radius /
 36 * 3.14 RIGHT 10]
END
```

8.6.2 Unser Papier

Wir sollten an unserem Papier zum Zeichnen noch eine kleine Ergänzung anbringen: Am Ende der Prozedur fügen wir den Befehl WINDOW ein, WRAP würde uns sonst Ärger machen, wenn mal etwas über den Bildschirmrand hinausgeht. Ähnliches gilt für FENCE: in diesem Modus würde die Zeichenroutine abgebrochen mit der Meldung Turtle out of bounds. WINDOW entspricht am ehesten dem, was wir auch auf dem Zeichenpapier tun: Wir denken uns, das Papier wäre größer und kommen dann automatisch an der richtigen Stelle wieder auf das Blatt. Genau das passiert bei WINDOW auch. Die Gefahr, daß wir dabei an die äußersten Grenzen (32768 nach oben und unten sowie rechts und links) kommen, ist recht gering!

```
TO PAPIER
WRAP
SETBR 6 SETBG 7 SETPC 0
KREUZ MASSSTAB RAND
WINDOW
END
```

8.6.3 Gerade zeichnen

Gerade werden durch zwei Punkte bestimmt. Diese müssen wir eingeben können. Die Schildkröte soll diese Gerade dann zeichnen. Es ist in der Geometrie üblich, die genannten Punkte durch kleine Querstriche zu kennzeichnen.

Nun stehen wir vor dem ersten Problem: wir kennen zwar die Positionen der Punkte, aber für eine Gerade brauchen wir mehr. Wir müßten – damit die Schildkröte beim Zeichnen schnell ist – mindestens einen Punkt an der Außenkante des Bildschirms kennen, den diese Gerade treffen würde. Die Richtung ließe sich dann mit der Funktion **TOWARDS** ermitteln, die eine Liste als Eingabe fordert. Damit kann der Kopf der Schildkröte so gedreht werden, daß er auf den in der Liste genannten Punkt zeigt. Um dies mit RT oder LT zu erreichen, müßten wir zum einen die augenblickliche Richtung der Schildkröte und zum anderen den Winkel zum gesuchten Punkt kennen. Diesen

Punkt kann man natürlich berechnen: aus zwei bekannten Punkten kann die Funktionsgleichung abgeleitet werden und dann der entsprechende Wert herausgesucht werden. Aber das ist aufwendig. Beim Schreiben des Programms können wir nicht voraussehen, an welcher Seite die Schildkröte auf die Außenkante stoßen muß.
Eine andere Möglichkeit wäre es, Schritt für Schritt an die Kante heranzugehen und mit vier IF-Abfragen festzustellen, ob die Schildkröte außerhalb unseres Fensters ist. Das würde zwar funktionieren, wäre aber sehr langsam.

Ein Mittelweg ist die hier vorgeschlagenen Lösung des Problems: Die Schildkröte wird auf den einen Punkt gesetzt und die Richtung mithilfe von TOWARDS festgelegt. Mit FORWARD 300 erreicht sie – egal wie sie steht – immer einen Punkt außerhalb des Fensters. Danach setzen wir sie nochmal auf den Punkt und drehen sie um 180 Grad. Dann lassen wir sie einfach noch einmal 300 Pixel (Bildpunkte) vorwärts gehen.

Die Koordinaten der Punkte geben wir über die Prozedur Koordinaten ein, die außerdem noch einen Text druckt, damit wir auch wirklich Zahlen eingeben. Denn selbst bei Programmen, die man gut kennt, kommt es vor, daß man nicht mehr weiß, was man an dieser oder jener Stelle eingeben muß.

Einen der beiden Punkte müssen wir in einer neuen Variablen Start speichern, weil die Variable Position ja durch den zweiten Aufruf von Koordinaten verändert wird. Wir brauchen aber wegen der Querstriche beide Punkte hinterher noch.

```
TO KOORDINATEN
TYPE [Bitte x und y eingeben:]
MAKE "Position READLIST
END


TO GERADE
SHOWTURTLE
KOORDINATEN
MAKE "Start :Position
KOORDINATEN
SETPOS :Position
SETH TOWARDS :Start
PD RT 90 FD 2 BK 4 FD 2 LT 90
SETPOS :Start
SETH TOWARDS :Position
RT 90 FD 2 BK 4 FD 2 LT 90
FD 300
PU SETPOS :Position RT 180
PD FD 300
RAND PU HOME HT
END
```


8.6.4 Strecken

Strecken unterscheiden sich von Geraden dadurch, daß die genannten Punkte die Endpunkte sind, während Geraden unendlich lang sind (theoretisch). Das heißt, wir müssen jetzt die Länge berechnen. Der – logisch gesehen – einfachste Weg führt über den Satz des Pythagoras: Quadrat der einen Kathete + Quadrat der zweiten Kathete = Quadrat der Hypothenuse.

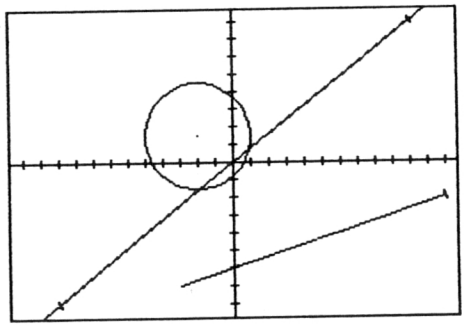
Wir sehen also unsere zu zeichnende Strecke als Hypothenuse eines rechtwinkligen Dreiecks an. Die eine Kathete errechnet sich dann aus der Differenz zwischen der X-Koordinate des zweiten Punktes und der X-Koordinate des ersten Punktes. Analog ermitteln wir die zweite Kathete aus den beiden Y-Werten. Beide werden quadriert (der genaueste Weg beim Computer führt immer noch über die Multiplikation). Dann wird mit der Funktion **SQRT** (SQuare RooT heißt Quadratwurzel) aus der Summe der beiden Kathetenquadrate die Länge der Hypothenuse berechnet. Dies übernimmt die Prozedur Laenge.

Nun setzen wir die Schildkröte auf den ersten Punkt, stellen die Richtung fest und sagen ihr, sie soll um soviele Punkte vorwärts gehen, wie die Prozedur Länge ergeben hat. An den jeweiligen Endpunkten lassen wir sie noch die kurzen Querstriche senkrecht zur Strecke ausführen, genau wie bei der Geraden auch.

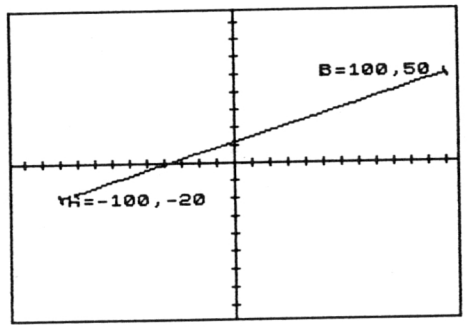
```
TO STRECKE
MAKE "Strecke []
PU ST
REPEAT 2 [KOORDINATEN DOT :Posis
tion MAKE "Strecke SE :Strecke :
Postion]
SETPOS SE FIRST :Strecke FIRST B
UTFIRST :Strecke
PD
SETH TOWARDS SE FIRST BF BF :Str
ecke FIRST BF BF BF :Strecke
RT 90 FD 2 BK 4 FD 2 LT 90
LAENGE
FD :Laenge
RT 90 FD 2 BK 4 PU HOME HT
END


TO LAENGE
MAKE "Kat1 FIRST BF BF BF :Strec
ke - FIRST BF :Strecke
MAKE "Kat2 FIRST BF BF :Strecke
- FIRST :Strecke
MAKE "Laenge :Kat 1 * :Kat1 + :K
at2 * :Kat2
MAKE "Laenge SQRT :Laenge
END
```

Alle drei Grafik-Möglichkeiten sehen auf unserem Papier vielleicht so aus:



Geometrie mit LOGO



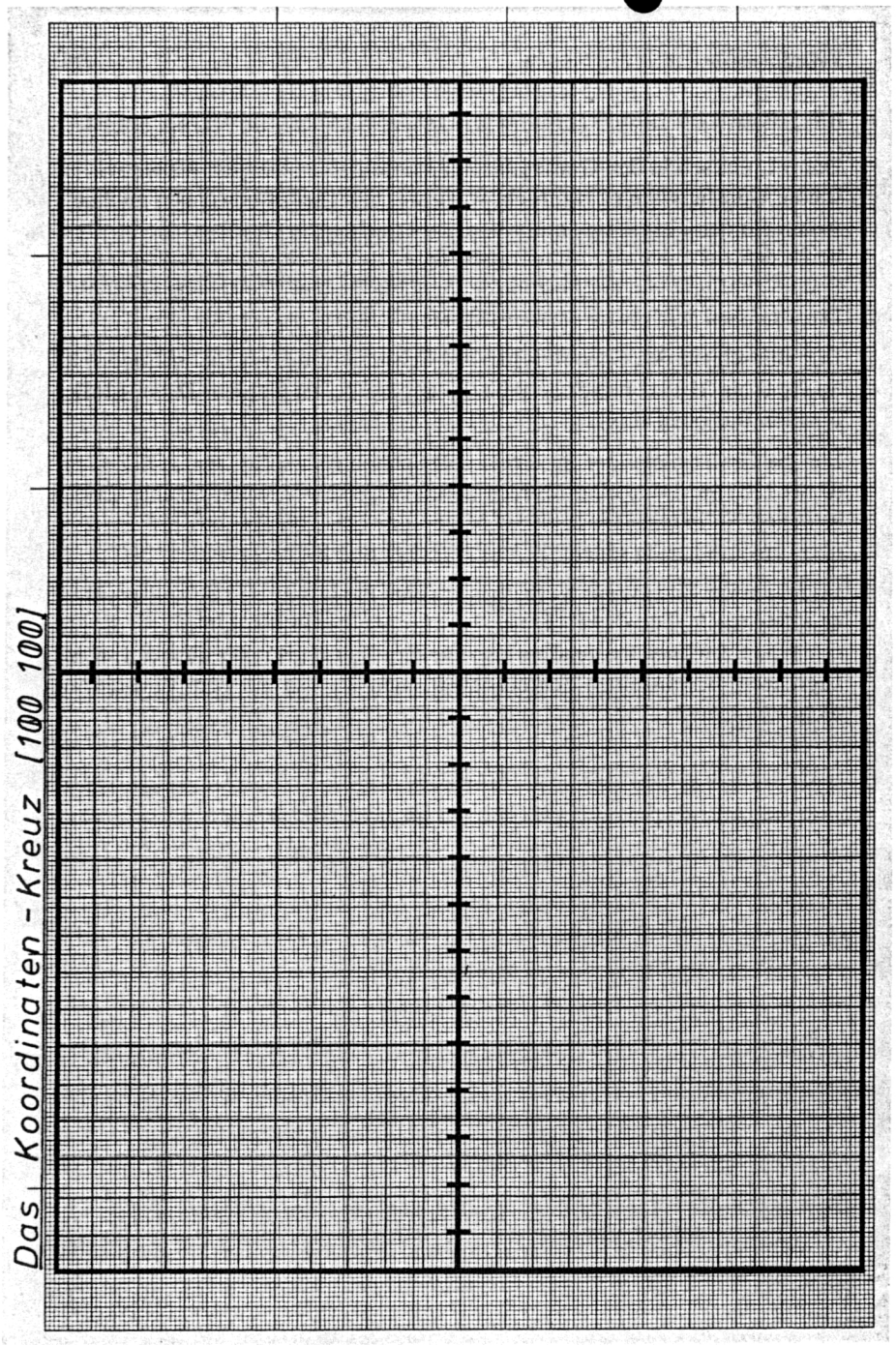
8.6.5 Texte im Bild

Daß wir einfach in den Grafikschirm hineinschreiben können, wissen wir ja schon. Günstig wäre es natürlich, wenn wir nicht selbst berechnen müßten, wohin wir den Text setzen müssen, damit er neben der Grafik erscheint. Das heißt, daß wir einen Algorithmus benötigen, der die Schreibstelle (Zeile und Spalte) abhängig von der Position der Schildkröte bzw. Zeichnung ermittelt. Dazu müssen wir etwas weiter ausholen.

Der Bildschirm des Spectrum hat 704 Schreibpositionen, die in Logo mit Spalte 0 bis 31 und mit Zeile 0 bis 21 angesprochen werden. Dabei ist die Position 0,0 oben links.

Für die Grafik bietet der Spectrum 256 Spalten (0 bis 255) und 176 Zeilen (0 bis 175) an. Die Zeilen werden normalerweise von unten gezählt. Logo setzt den Nullpunkt jedoch in die Mitte und zählt von dort aus 128 Punkte nach links, 88 nach unten, 127 nach rechts und 87 nach oben.

Im folgenden Bild ist das Verhältnis unseres Papiers zu den Schreibpositionen dargestellt. Die dicken Linien entsprechen genau den Linien, die Logo mit Papier zeichnet; sie gehören also mit zum Bildschirm. Die dünnen Linien begrenzen die einzelnen Schreibstellen, sind also so auf dem Bildschirm nicht zu erzeugen.



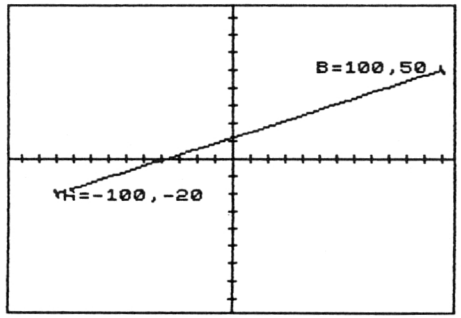
Eine Schreibstelle auf dem Bildschirm entspricht immer 8 * 8 Punkten. Um von der Schildkrötenposition auf die Schreibstelle umzurechnen, müssen wir für die Zeile den Positionswert abziehen. Außerdem müssen wir die senkrechte Verschiebung zur BASIC-Pixel-Position in Rechnung stellen, indem wir 88 abziehen. Diesen Wert müssen wir noch durch 8 dividieren. Dabei benutzen wir die Funktion **INT**, die uns den ganzzahligen Teil des Ergebnisses liefert, weil SETCUR nur ganze Zahlen verarbeiten kann. Hier tauchen zum ersten Mal runde Klammern auf: zum Rechnen werden sie auch in Logo gebraucht.
Die Berechnung der Spalte ist ähnlich, nur werden in beiden Fällen – sowohl bei der Grafik als auch beim Schreiben – die Spalten von links nach rechts gezählt. Deshalb müssen wir hier die waagerechte Verschiebung von 128 Punkten addieren, bevor wir durch 8 teilen.

Der Befehl **OVER 1** stellt sicher, daß von dem TYPE-Befehl nur die tatsächlich berührten Punkte gelöscht werden und nicht der ganze Bereich dieser Schreibposition. Dies sollte aber trotzdem bei der Eingabe der Position berücksichtigt werden. Auf dem Papier würden wir ja auch in das Kästchen daneben schreiben.
TYPE statt PRINT hat hier den Vorteil, daß – wegen des nicht ausgeführten Wagenrücklaufes – auch in der untersten Zeile des Bildschirms geschrieben werden kann, ohne daß der Bildschirm um eine Zeile nach oben geschoben wird. Der einzige kritische Punkt ist die Schreibposition unten rechts. Der Cursor rückt um eine Stelle weiter – und das ist die nächste Zeile.

Schreiben rechnet also die Position in die entsprechende Schreibstelle um. Wenn dabei der Text so lang ist, daß er über den rechten Bildschirmrand hinausgeht, wird in der nächsten Zeile weitergeschrieben. Dies sollte bei der Text- und Positionseingabe berücksichtigt werden. WINDOW gilt nur für die Schildkröte, nicht für das Drucken von Texten.

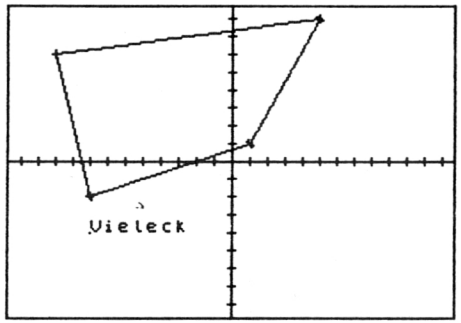
```
?
TO SCHREIBEN
OVER 1
TYPE [Text: ]
MAKE "Text READLIST
TYPE [Position: ]
MAKE "Cur READLIST
MAKE "Spalte FIRST :Cur
MAKE "Spalte :Spalte + 128
MAKE "Spalte INT ( :Spalte / 8 )
MAKE "Zeile LAST :Cur
MAKE "Zeile 175 - :Zeile - 88
MAKE "Zeile INT ( :Zeile / 8 )
DOT :Cur
SETCUR SE :Spalte :Zeile
TYPE :Text
END

?
```



8.6.6 Vielecke

Vielecke zu konstruieren, ist mit unserer Prozedur Strecke nicht mehr schwer. Wir müssen nur die entsprechende Anzahl von Strecken zeichnen lassen. Vielleicht wäre es hier hübscher, das Kennzeichnen der Eckpunkte wegzulassen. In der Schule ist dies jedoch unüblich.




```
Vieleck
```


Kapitel 9:

Selbstdefinierte Grafikzeichen

9.1 Bits und Bytes

Damit wir die Möglichkeit nutzen können, Zeichen selbst zu definieren, müssen wir uns zunächst ansehen, wie diese Zeichen vom Spectrum gespeichert werden.

Die Schriftzeichen sind als Bitmuster im Speicher abgelegt. Wir hatten bei der Grafik- und Schriftverteilung im letzten Kapitel schon bemerkt, daß jedes Zeichen beim Spectrum 8 * 8 Punkte belegt.

Ein **Bit** ist genaugenommen nichts anderes als ein kleiner Schalter, der an oder aus sein kann. Jede Zeile eines Buchstabens umfaßt **8 Bit = 1 Byte**. Die Bits werden von rechts nach links gezählt, beginnend mit Null. Jedes Zeichen braucht demnach 8 Bytes für seine vollständige Darstellung. Diese Art der Speicherung ist der Grund dafür, daß der Spectrum nicht zwischen der Grafik und der Schrift unterscheidet. Mit anderen Worten: deshalb ist es ohne Tricks möglich, Text und Grafik zu mischen.

9.2 Binärzahlen

Der Computer kann nur mit Hilfe von solchen kleinen Schaltern gesteuert werden. Die Zustände eines Stromschalters können mathematisch 0 für aus und 1 für ein genannt werden. Daraus folgt, daß wir vor der Änderung einer Speicherstelle die neue, dorthinzuschreibende Binärzahl ermitteln bzw. umgekehrt die Binärzahl in unser „gewöhnliches“ Dezimalsystem umrechnen.

9.3 Umwandlungen

9.3.1 Binär nach Dezimal

128	64	32	16	8	4	2	1
0	1	0	1	0	1	0	1

Das Binär- oder Dualsystem ist – genau wie das Dezimalsystem – ein Stellenwertsystem. Das bedeutet, daß eine Ziffer ihren Wert in der Zahl erst durch ihre Position erhält. Im Dezimalsystem erkennen wir beispielsweise die Ziffernfolge 2 3 3 als 233 = 2 * 100 (10 ^ 2) + 3 * 10 (10 ^ 1) + 3 * 1 (10 ^ 0). ^ ist das Zeichen für „hoch". Genauso können wir jetzt den dezimalen Wert der oben dargestellten Dualzahl ermitteln: 0*128 (2^7) + 1*64 (2^6) + 0*32 (2^5) + 1*16 (2^4) + 0*8 (2^3) + 1*4 (2^2) + 0*2 (2^1) + 1*1 (2^0) = 85. Nebenbei: es ist mathematisch definiert, daß eine mit 0 potenzierte Zahl immer 1 ergibt.

9.3.2 Dezimal nach Binär

Das Verfahren, eine gegebene Dezimalzahl in eine Binärzahl umzurechenen, ist für uns zwar nicht so wichtig, aber es sei der Vollständigkeit halber gezeigt. Man nennt es das Divisionsrestverfahren. Die Zahl wird so oft durch 2 dividiert, bis das Ergebnis 0 ist. Der sich dabei ergebende Rest ist jeweils eine Ziffer der Dualzahl:

```
85 : 2 = 42 Rest 1
42 : 2 = 21 Rest 0
21 : 2 = 10 Rest 1
10 : 2 =  5 Rest 0
 5 : 2 =  2 Rest 1
 2 : 2 =  1 Rest 0
 1 : 2 =  0 Rest 1
```

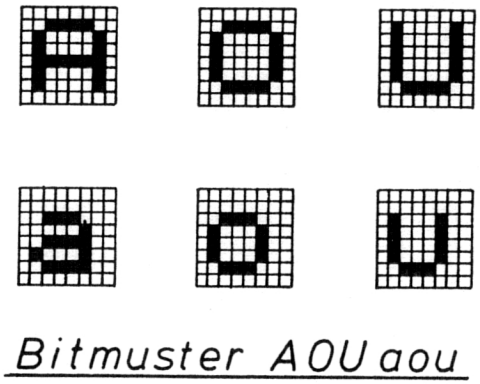
Dabei ist die erste ermittelte Ziffer die letzte der Dualzahl. Um in diesem Fall auf 8 Bit = 1 Byte zu kommen, müssen wir unsere Zahl um die führende 0 für das 7. Bit ergänzen: 01010101. (Wir erinnern uns: die Zählung beginnt mit 0.)

9.4 Selbstdefinierbare Zeichen

Diese Rechnerei ist notwendig, um die 21 vom Spectrum bereitgestellten frei definierbaren Zeichen gestalten zu können. Dafür ist im Speicher ein bestimmter Bereich freigehalten, der nach dem Einschalten des Computers die Großbuchstaben A bis U enthält.

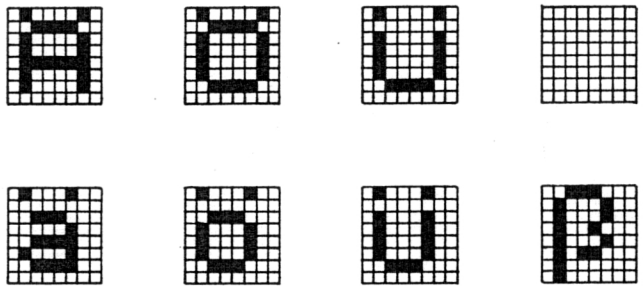
9.4.1 Umlaute definieren

Wenn wir die deutschen Sonderzeichen definieren wollen, sollten wir uns erst einmal ansehen, wie im Spectrum die entsprechenden Buchstaben ohne die Pünktchen aussehen. Sonst passen unsere Zeichen vielleicht nicht zum normalen Zeichensatz:



Bitmuster AOU aou

Wie wir sehen, sind bei (fast) allen Zeichen des Spectrum die äußeren Bits nicht gesetzt. Das heißt, daß wir einfach nur noch je zwei Punkte auf die Buchstaben setzen müssen, damit wir die deutschen Umlaute bekommen. Nur beim großen U sollten wir zwei Punkte des ursprünglichen Buchstaben weglassen, damit ein Punkt zwischen unseren Pünktchen und dem Buchstaben frei ist. Es ist troztdem noch vom kleinen ü zu unterscheiden, weil es breiter ist. Für das ß müssen wir uns ein neues Zeichen ausdenken, das hierzu paßt. Wie das aussehen könnte, zeigt das folgende Bild.



Raster für die UDG's

UDG's steht für englisch User Defined Graphics, benutzerdefinierte Grafikzeichen.
Ob wir das ß so oder vielleicht anders definieren, ist Geschmackssache. Hier wurde ein Raster von 8 * 8 Kästchen mit je 2 mm Kantenlänge benutzt. Aus einiger Entfernung läßt sich bei dieser Größe die endgültige Wirkung ganz gut beurteilen. Andererseits lassen sich Kästchen dieser Größe noch recht gut zeichnen und ausmalen.

9.4.2 Die dezimalen Werte

Nun geht's ans Rechnen: wir brauchen die dezimalen Werte jedes Bytes (jeder Punktzeile), um diese in den Speicher schreiben zu können:

Ä : 64+2=66, 32+16+8+4=60, 66, 66,
64+32+16+8+4+2=126, 66, 66, 0
Ö : 66, 60, 66, 66, 66, 66, 60, 0
Ü : 66, 0, 66, 66, 66, 66, 60, 0
ä : 64+4=68, 0, 32+16+8=56, 4, 60, 68, 60, 0
ö : 68, 0, 56, 68, 68, 68, 56, 0
ü : 68, 0, 68, 68, 68, 68, 56, 0
ß : 56, 68, 68, 64+8=72, 68, 64+16+8=88, 64, 64

Für jeden ausgemalten Punkt haben wir die Binärziffer 1 angenommen, für jeden leeren die Binärziffer 0. Aus der Summe der einzelnen Stellenwerte ergibt sich dann die Dezimalzahl. Es zeigt sich, daß wir gar nicht so viel rechnen mußten; es sind wenige Bytezeilen, die sich immer wiederholen.

9.4.3 Definieren der Zeichen

Nun brauchen wir einige neue Befehle: Wir müssen bestimmen, welches der 21 Zeichen neu definiert werden soll. Dann müssen wir die für dieses Zeichen reservierten 8 Byte mit den eben ermittelten Werten beschreiben.

Der Befehl zum Ändern einer Speicherstelle heißt **.DEPOSIT adresse wert**. Die frei definierbaren Grafikzeichen beginnen im Spectrum bei der Adresse 65368. Damit wir diese Adresse jeweils vom Computer ermitteln lassen können, müssen wir sie irgendwie in einen Zusammenhang mit dem ASCII-Code des entsprechenden Buchstabens in Verbindung bringen. Der ASCII-Code für das erste infrage kommende Zeichen ist 65 (A). Jedes Zeichen benötigt 8 Byte. Die Funktion **ASCII "buchstabe** ergibt als Ergebnis den Code des genannten Zeichens.
Also ermitteln wir das erste Byte für jeden der 21 möglichen Buchstaben durch (ASCII "buchstabe − 65) * 8 und addieren dazu die Adresse des allerersten Bytes dieses Bereichs. Für A ergibt diese Formel den Wert 65368, was völlig richtig ist, für B 65376, also 8 Bytes höher. Auch das ist richtig.

Als zweiten Parameter unseres Definitionsbefehls brauchen wir noch die Liste mit den Werten, die immer acht Elemente enthalten sollte. Nun werden diese 8 Werte der Reihe nach in die 8 Speicherstellen geschrieben. Die Variable für die Speicherstelle muß jedesmal um 1 erhöht werden, die Liste der Werte um das erste Element verkürzt.

```
?
TO GRAFIK :B :W
MAKE "Byte ( ASCII :B - 65 ) * 8
 + 65368
REPEAT 8 [.DEPOSIT :Byte FIRST :
W MAKE "W BF :W MAKE "Byte :Byte
 + 1]
END

?
```

Nun können wir für unsere deutschen Sonderzeichen eine Prozedur definieren, die den Befehl Grafik mit den entsprechenden Buchstaben und Werten jeweils aufruft. Dabei müssen die zu definierenden Zeichen als Großbuchstaben eingegeben werden.

```
?
TO UMLAUTE
GRAFIK "A [66 60 66 66 126 66 66
 0]
GRAFIK "B [66 60 66 66 66 66 60
0]
GRAFIK "C [66 0 66 66 66 66 60 0
]
GRAFIK "D [68 0 56 4 60 68 60 0]
GRAFIK "E [68 0 56 68 68 68 56 0
]
GRAFIK "F [68 0 68 68 68 68 56 0
]
GRAFIK "G [56 68 68 72 68 88 64
64]
END

?
```

Wenn wir diese Prozedur jetzt aufrufen, hat es den Anschein, als ob überhaupt nichts passiert. Trotzdem dauert es einen kleinen Moment, bis sich der Logo-Interpreter wieder meldet.

9.4.4 Benutzen dieser Zeichen

Diese Sonderzeichen wollen wir natürlich jetzt auch in Texten benutzen. Sie haben die Codes 144 – 164. Mit CHAR zahl können wir ein Zeichen ausdrucken lassen. Mit PRINT CHAR 144 beispielsweise erhalten wir unser großes Ä, das Ö hat den Code 145, Ü ist über 146 zu erhalten usw. Wenn wir die deutschen Zeichen öfter benötigen, ist es sinnvoll, immer dasselbe Programm dafür zu verwenden. Dadurch können wir die Codes für die am häufigsten gebrauchten Zeichen schnell auswendig.

Bei der Ansprache der Drucker können diese Grafikzeichen normalerweise nicht ausgegeben werden. Die Ausnahme sind hier natürlich der ZX-Printer und die (fast) genauso arbeitenden Drucker Alphacom 32 und Seikosha GP 50 S. Die über das Interface 1 angesteuerten Drucker können diese Zeichen nicht ausgeben. Sie werden immer ein Fragezeichen drucken, wenn ein Zeichen mit einem Code über 127 verlangt wird. Bei den über ein Centronics-Interface angesteuerten Druckern hängt es vom Interface ab, ob der Drucker den Spectrum-Schriftsatz benutzt oder nicht.

9.4.5 Ein Beispiel für Text mit Sonderzeichen

Mit der folgenden kleinen Prozedur wurde der darunter abgebildete Bildschirminhalt erzeugt:

```
?
TO DEMO
RAND OVER 1
PR [Dies ist eine Demonstration
des]
PR [neuen Zeichensatzes mit den]
PR [deutschen Sonderzeichen:]
PR []
PR SE CHAR 144 [hat den Code 144
]
PR SE CHAR 145 [hat den Code 145
]
PR SE CHAR 146 [hat den Code 146
]
PR SE CHAR 147 [hat den Code 147
]
PR SE CHAR 148 [hat den Code 148
]
PR SE CHAR 149 [hat den Code 149
]
PR SE CHAR 150 [hat den Code 150
]
PR []
TYPE [Diese Zeichen k]
TYPE CHAR 148 PRINT [nnen mit]
PR [.BSAVE "name [adresse laenge
]]
PR [gesichert werden.]
END

?
```

```
Dies ist eine Demonstration des
neuen Zeichensatzes mit den
deutschen Sonderzeichen:

Ä hat den Code 144
Ö hat den Code 145
Ü hat den Code 146
ä hat den Code 147
ö hat den Code 148
ü hat den Code 149
ß hat den Code 150

Diese Zeichen können mit
.BSAVE "name [adresse laenge]
gesichert werden.
```

9.5 Andere Grafikzeichen

Mit der eben definierten Prozedur Grafik lassen sich alle 21 UDG's des Spectrum ansprechen. Hier kann man natürlich auch andere Zeichen definieren. So wären vielleicht die kleinen Buchstaben des griechischen Alphabetes

interessant, um gezeichnete Winkel benennen zu können, wie es von der Schule verlangt wird. Dazu muß nur das Zeichen in ein möglichst ähnliches Bitmuster übertragen werden. Mit der Prozedur Schreiben aus dem letzten Kapitel kann dann ein Winkel bezeichnet werden.

9.6 Abspeichern der Zeichen bzw. von MCode

Es ist natürlich möglich, diese Prozedur mit SAVE ''UMLAUTE auf einem Datenträger abzulegen. Wir können aber auch den Inhalt des Speicherbereiches direkt ablegen und die fertigen Werte später laden. Das bringt einen Zeitgewinn, weil nach dem Laden der Prozedur diese ja auch noch ausgeführt werden müßte. Nach dem Laden des Speicherbereiches steht die Information schon dort und kann direkt benutzt werden. Der Befehl für das Sichern des Speicherbereiches heißt **.BSAVE ''name [start laenge]**
Der für die Sonderzeichen benötigte Bereich beginnt bei 65368. Die Länge ergibt sich aus der mit 8 multiplizierten Zahl der definierten Zeichen, hier also 56 Bytes. **Mit .BLOAD ''name adresse** kann dieses File in den Speicher geladen werden.

Diese beiden Befehle gelten allgemein für jede Art von Maschinencode, beispielsweise für die den Drucker nötige Steuersoftware. Es besteht kein grundsätzlicher Unterschied zwischen beiden beim Speichern oder Laden. Maschinencode wird aufgerufen durch .CALL adresse, wobei adresse das 1. Byte des Codes ist. Die Grafikzeichen werden zwar genauso abgespeichert wie Maschinencode, können aber nur über Druckbefehle aufgerufen werden.

Kapitel 10:
Mathematische Funktionen

Im Zusammenhang mit Grafik ist auch die Darstellung von mathematischen Funktionen interessant. Hier soll nur an einem kurzen Beispiel gezeigt werden, wie so eine Prozedur aussehen könnte.

Manchmal könnte man z.B. auch Kreisbögen benötigen und nicht nur Kreise. Da die mathematischen Hintergründe hier etwas zu weit führen würden, sind nur die Listings abgedruckt. Damit können Kreise und auch Kreisbögen, aber natürlich auch zusammengesetzte Figuren gezeichnet werden; die Schildkröte bleibt ja dort stehen, wo sie aufhört zu zeichnen.

KREISR und KREISL zeichnen einen Kreis, R steht dabei für rechtsherum und L für linksherum. BOGENR und BOGENL arbeiten analog. Sie verlangen aber außer der Eingabe des Radius' noch die Winkelgröße. Die vier anderen Prozeduren rechnen die Winkelangaben in Bogenmaß um; das Spectrum-Logo verfügt nicht über eine solche Funktion.

```
?
TO KREISR :radius
BOGENR :radius 360
END


?
TO BOGENR :radius :winkel
RT 2.5
BOGENR1 :radius * .017453 :winke
l
LT 2.5
END


?
TO BOGENR1 :radius :winkel
REPEAT :winkel / 5 [FD :radius *
 5 RT 5]
RBOGENR :radius ( REMAINDER :win
kel 10 )
END
```

```
?
TO RBOGENR :groesse :summe
FD :groesse * :summe
RT :summe
END


?
TO KREISL :radius
BOGENL :radius 360
END


?
TO BOGENL :radius :winkel
LT 2.5
BOGENL1 :radius * .017453 :winke
l
RT 2.5
END


?
TO BOGENL1 :radius :winkel
REPEAT :winkel / 5 [FD :radius *
 5 LT 5]
RBOGENL :radius ( REMAINDER :win
kel 10 )
END


?
TO RBOGENL :groesse :summe
FD :groesse * :summe
LT :summe
END


?
```

Als kleines Beispiel für mathematische Funktionen sollen hier **SQRT** für Quadratwurzel (engl. Square root) und **PRODUCT** dienen. Mit der Prozedur ABSTAND, die als Eingabe zwei Listen erfordert, kann der Abstand zweier Punkte berechnet werden. Die Prozedur QUADRAT berechnet das Quadrat einer Zahl.

Der Befehl **OUTPUT** (kurz OP) bewirkt, daß die beiden definierten Prozeduren wie LOGO-Funktionen arbeiten. D.h. sie können nur mit einem Befehl zusammen benutzt werden, beispielsweise mit PRINT oder FORWARD.


```
?
TO QUADRAT :X
OUTPUT PRODUCT :X :X
END

TO ABSTAND :P1 :P2
OP SQRT SUM QUADRAT ( FIRST :P1
) - ( FIRST :P2 ) QUADRAT ( LAST
 :P1 ) - ( LAST :P2 )
END

?
```

Anhang 1:

Alle Wörter des Sinclair-Logo (Primitives)

Alle Primitives von Logo sind hier mit einer Erklärung genannt. Die Angaben in runden Klammern verweisen auf das Kapitel, in denen das Primitive zuerst vorkommt.

AND	logische Verknüpfung zweier Bedingungen, die dann beide wahr sein müssen, damit die Gesamtbedingung wahr ist
ARCCOS winkel	Winkelfunktion, ergibt den Arcus-Cosinus des gegebenen Winkels
ARCCOT winkel	Winkelfunktion, ergibt den Arcus-Cotangens des gegebenen Winkels
ARCSIN winkel	Winkelfunktion, ergibt den Arcus-Sinus des gegebenen Winkels
ARCTAN winkel	Winkelfunktion, ergibt den Arcus-Tangens des gegebenen Winkels
ASCII ''buchstabe	Wortfunktion, ergibt den Standard-Code des gegebenen Zeichens (9.4.3)
BACK (BK) n	(2.2.1) Zurück, Grafikbefehl
BACKGROUND (BG)	Pseudovariable, enthält Code der Hintergrundfarbe
BRIGHT n	verändert die Helligkeit bei PRINT und TYPE (NORMAL)
BUTFIRST (BF) Liste/Wort	Wortfunktion, ergibt alle Elemente des Argumentes außer dem ersten (5.2.2)
BUTLAST (BL) Liste/Wort	Wortfunktion, ergibt alle Elemente des Argumentes außer dem letzten

BYE — verläßt Logo (ins BASIC). Logo kann mit RUN neu gestartet werden (7.2)

CATALOG — ergibt den Inhalt der Cartridge in dem gewählten Microdrive (SETDRIVE)

CHAR n — Gegenfunktion zu ASCII, ergibt das Zeichen mit dem gegebenen Code (5.2.6)

CLEAN — löscht den Grafikschirm, ohne die Position der Kröte zu beeinflussen

CLEARSCREEN (CS) — löscht den Grafikschirm und setzt die Kröte auf den Nullpunkt (2.3)

CLEARTEXT (CT) — löscht den gedruckten Text, ohne den Bildschirm zu verändern

COPYDEF ''neuername ''name — kopiert die Prozedur name in die Prozedur neuername, die alte Prozedur wird nicht gelöscht

COPYSCREEN — kopiert den Bildschirm-Inhalt auf den ZX-Drucker oder vergleichbare (7.1)

COSINE (COS) Winkel — Winkelfunktion, ergibt den Cosinus

COTANGENT (COT) Winkel — Winkelfunktion, ergibt den Cotangens

COUNT Liste/Wort — Wortfunktion, ergibt die Anzahl der Elemente des Argumentes (5.2.6)

CURSOR — Pseudovariable, ergibt die Position des Cursors (Schreibstellenanzeiger)

DEFINE ''prozedurname |parameter- und befehlslisten — definiert die Prozedur prozedurname mit den Parametern der Liste und den Befehlen der Liste

DEFINEDP ''name — Funktion, ergibt wahr, wenn name der Name einer Prozedur ist

DIV zahl1 zahl 2 — ergibt den Quotient der beiden Zahlen (/)

DOT [a b] — setzt einen Punkt auf die angegebene Position (6.8.1)

EDIT (ED) ''name — ruft den Bildschirmeditor auf und druckt – falls vorhanden – die Prozedur aus (1.5.2)

EMPTYP — Funktion, ergibt wahr (true), wenn das Argument eine leere Liste bzw. ein leeres Wort ist (5.2.2)

END — beendet jede Prozedur (2.1.2)

EQUALP — Funktion, ergibt wahr (true), wenn die beiden Argumente gleich sind

ERALL — löscht alle Prozeduren und Variablen (6.2)

ERASE (ER) ''name — löscht die Prozedur mit dem gegebenen Namen (4.1.1/6.1.2)

ERASEFILE ''name — löscht ein File auf Cartridge in dem angewählten Drive (SETDRIVE). Der Typ des Files muß mit angegeben werden, weil er von Logo bei SAVE zum Teil des Namens wird (6.2.2)

ERN ''name — löscht die Variable mit dem angegebenen Namen

ERNS — löscht alle Variablen

ERPS — löscht alle Prozeduren

FENCE — begrenzt den Arbeitsbereich der Schildkröte auf den Bildschirm (3.1.7)

FIRST Liste/Wort — Funktion, ergibt das erste Element des Argumentes (5.2.2)

FLASH — der nachfolgende Text wird blinken (NORMAL)

FORWARD (FD) n — Vorwärts, Grafikbefehl (2.2.1)

FPUT Wort Liste — Funktion, ergibt eine neue Liste mit dem ersten und zweiten Argument als Elemente

HEADING — Pseudovariable, enthält die Kopfposition der Schildkröte (Winkel)

HIDETURTLE (HT) — Verstecke Schildkröte (2.2.1)

HOME — bringt die Kröte in die Ursprungslage (8.1)

IF bedingung Liste1 <LISTE2> (ist optional) — Wenn die Bedingung erfüllt ist, wird die Befehlsliste1 ausgeführt, andernfalls Liste2 (2.2.2)

INT zahl	Zahlenfunktion, ergibt den ganzzahligen Teil der Zahl (Nachkommastellen werden abgeschnitten) (8.6.5)
INVERSE	gibt den nachfolgenden Text mit vertauschten Vorder- und Hintergrundfarben
ITEM n Wort/Liste	Funktion, ergibt das spezifische Element des folgenden Argumentes
KEYP	Funktion, prüft die Tastatur auf Drücken einer gültigen Taste(nkombination)
LAST Liste/Wort	Funktion, ergibt das spezifizierte Element des Argumentes
LEFT (LT) n	Links, dreht die Kröte um den angegebenen Winkel gegen den Uhrzeigersinn (2.2.1)
LIST Wort/Liste Wort/Liste...	Funktion, ergibt eine Liste, die alle gegebenen Argumente in unveränderter Form enthält
LISTP :name	Funktion, ergibt wahr (true), wenn das Argument eine Liste ist
LOAD ''name	lädt ein File mit dem gegebenen Namen von Cassette oder Microdrive (SETDRIVE)
LOADD ''name	lädt alles, was mit SAVED gesichert wurde, wieder in den Editor
LOADSCR ''name	lädt wie LOAD, jedoch einen Bildschirm
LPUT Wort/Liste	Funktion, fügt das erste Argument als letztes Element an das zweite an
MAKE ''name wert	Befehl, eine Variable global (immer gültig) zuzuweisen
MEMBERP Wort/Liste	Funktion, ergibt wahr (true), wenn das erste Argument Element des zweiten ist (3.3)
NAMEP ''name	Funktion, ergibt wahr (true), wenn das Argument einen Wert hat (eine Variable ist)
NODES	Funktion, liefert als Ergebnis die Zahl der freien Speicherplätze dividiert durch fünf (6.1.1)
NORMAL	schaltet die BRIGHT und/oder FLASH aus

NOT bedingung	logische Verknüpfung mit einer Bedingung, kehrt das Ergebnis der Bedingung um, ergibt also falsch, wenn die Bedingung wahr ist und umgekehrt
NUMBERP ''name	Funktion, ergibt wahr (true), wenn das Argument eine Zahl ist (5.2.2)
OR	logische Verknüpfung zweier Bedingungen, ergibt wahr, wenn mindestens eine der beiden Bedingungen wahr ist.
OUTPUT (OP) ergebnis	Befehl, ein Ergebnis der Prozedur an die aufrufende Prozedur weiterzugeben; diese Prozedur ist eine Funktion (10)
OVER n	nachfolgender Text wird bei OVER 1 gedruckt, ohne den vorher geschriebenen Text zu löschen, OVER 0 setzt normal (8.6.5)
PENCOLOUR (PC)	Pseudovariable, enthält den Code der Schreibfarbe
PENDOWN (PD)	Stift runter, bei der nächsten Bewegung zeichnet die Kröte (3.1.3)
PENERASE (PE)	Stift löschen, jeder gezeichnete und berührte Punkt wird gelöscht, es wird kein neuer gezeichnet
PENREVERSE (PX)	Stift umgekehrt, wird ein bereits gezeichneter Punkt berührt, wird dieser gelöscht, ansonsten gezeichnet
PENUP (PU)	Stift hoch, die Kröte zeichnet nicht mehr (3.1.3)
PO ''name	Print Out, druckt die genannte Prozedur aus
POALL	druckt Namen und Definitionen aller Prozeduren aus
PONS	druckt alle Variablen mit Inhalt aus (5.2.5)
POPS	druckt die Definitionen aller Prozeduren aus
POSITION (POS)	Pseudovariable, enthält die Koordinaten der Krötenposition
POTS	druckt die Namen (und Parameter) aller Prozeduren aus (3.2)

PRIMITIVEP ''name	ergibt wahr, wenn das Wort ein Primitive ist
PRINT (PR)	Druckbefehl, der mindestens einen Parameter benötigt. Führt nach dem Ausdrucken des (der) Parameter(s) einen Wagenrücklauf aus (2.1.1)
PRINTOFF	schaltet Druckerausgabe aus (7)
PRINTON	schaltet Druckerausgabe ein, der Bildschirm wird dabei nicht abgeschaltet, das heißt, es wird sowohl auf den Drucker als auch auf den Bildschirm ausgegeben. (7) Andere als der ZX-Printer müssen evtl. vorher initialisiert werden (BYE)
PRODUCT zahl1 zahl2	mathematische Funktion, ergibt das Produkt der Zahlen (*), es können auch mehr als zwei sein (10)
RANDOM zahl	Zahlenfunktion, liefert eine ganze zufällige Zahl zwischen 0 und zahl-1 (3.1.1)
READCHAR (RC)	Funktion, liest die Tastatur und ergibt das Zeichen, dem die gedrückte Taste entspricht
READLIST (RL)	Funktion, erwartet eine Eingabe von der Tastatur, die mit ENTER abgeschlossen werden muß. Liefert eine Liste ab. (3.2)
RECYCLE	ordnet den Arbeitsspeicher neu, um Platz zu schaffen, engl. garbage collection, Müllsammlung (6.1.3)
REMAINDER zahl1 zahl2	mathematische Funktion, ergibt den Rest, der bei der Division von zahl1 durch zahl2 bleiben würde (Modulo)
REPEAT zahl Befehlsliste	Wiederholt die Befehlsfolge in der Liste sooft, wie die Zahl angibt (3.1.1)
RIGHT (RT) n	Rechts, dreht die Kröte um den angegebenen Winkel im Uhrzeigersinn (2.2.1)
ROUND zahl	mathematische Funktion, ergibt den (nach 5/4) gerundeten ganzzahligen Wert der Zahl

RUN befehlsliste	führt die Befehlsfolge in der Liste aus
SAVE ''name liste	sichert die in der Liste genannten Prozeduren unter dem Namen auf Band oder Cartridge (SETDRIVE)
SAVEALL ''name	sichert alle Prozeduren auf Band oder Cartridge (6.2.2)
SAVED ''name	sichert alles derzeit im Editor befindliche unter dem gegebenen Namen auf Band oder Cartridge
SAVESCR ''name	sichert den Bildschirminhalt unter dem gegebenen Namen auf Band oder Cartridge (6.2.2)
SCRUNCH	Pseudovariable, enthält den derzeitigen Maßstab für das Koordinatenkreuz, in dem die Kröte zeichnet (normal: 100 100)
SENTENCE (SE) Wort/Liste Wort/Liste	Funktion, ergibt eine neue Liste, die die Elemente aller Argumente enthält (4.3)
SETBG n	Setze Hintergrundfarbe (4.2)
SETBORDER (SETBR) n	Setze Randfarbe (4.2)
SETCURSOR (SETCUR) [x y]	benötigt zwei Parameter (in einer Liste), setzt den Cursor auf die damit angegebene Spalte und Zeile (0-31, 0-21) (5.102)
SETDRIVE n	wählte den Massenspeicher (Cassette oder Microdrive) aus, 0 ist Cassette, 1 - 8 ist Drive 1 bis 8 (6.2.1)
SETHEADING (SETH) n	Setze Richtung, stellt die Schildkröte in Richtung des angegebenen Winkels (3.1.3)
SETPC n	Setze Stiftfarbe (4.2)
SETPOS [x y]	Setzt die Schildkröte auf die angegebene Position, ohne Richtungsänderung (3.1.3)
SETSCRUNCH (SETSCR) [x y]	Verändert den Maßstab für das Verhältnis zwischen waagerechten und senkrechten Linien
SETTC [Hg Vg]	Farbbefehl, benötigt zwei Parameter in einer Liste, Hintergrundfarbe, Vordergrundfarbe (4.2)

SETX n	setzt die Kröte auf die angegebene X-Koordinate, ohne den Y-Wert zu verändern (8.5)
SETY n	setzt die Kröte auf die angegebene Y-Koordinate, ohne den X-Wert zu verändern (8.5)
SHOW name	Befehl mit einem Parameter, zeigt eine Liste mit Begrenzern (eckige Klammern), ein Wort jedoch ohne Anführungszeichen
SHOWNP	Funktion, ergibt wahr (true), wenn die Schildkröte sichtbar ist
SHOWTURTLE (ST)	zeige Schildkröte (2.2.1) SINE (SIN) n Winkelfunktion, ergibt den Sinus des Winkels
SOUND laenge hoehe	ergibt einen Ton von der Länge laenge (in Sekunden) und der Höhe hoehe (4.3) 0 = C, Veränderung um 1 = 1 Halbtonschr.
SQRT zahl	mathematische Funktion, ergibt die Quadratwurzel der Zahl (8.6.4)
STARTROBOT	startet den evtl. angeschlossenen Roboter. Die zur Steuerung nötigen Daten müssen vorher bereitgestellt werden.
STOP	Befehl, die laufende Prozedur abzubrechen und die aufrufende Prozedur weiter zu bearbeiten (2.2.2)
STOPROBOT	Stoppt den Roboter
SUM zahl1 zahl2	mathematische Funktion, ergibt die Summe der Zahlen, es können auch mehr sein (+)
TANGENT (TAN) winkel	Winkelfunktion, ergibt den Tangens
TEXT ''name	Funktion, die die Befehlszeilen der Prozedur name als Listen liefert
TEXTCOLOUR (TC)	Pseudovariable, enthält Textfarben
TEXTSCREEN (TS)	macht den ganzen Bildschirm für Text verfügbar, löscht auch Texte (2.1.1)
THING ''name	Funktion, ergibt den Inhalt des Arguments

TO name	Aufruf des Zeileneditors zur Neudefinition einer Prozedur, evtl. noch mit Variablennamen zur Parameterübergabe
TOPLEVEL	ähnlich STOP, bricht jedoch das gesamte Programm ab, während STOP an die aufrufende Prozedur übergibt
TOWARDS [x y]	Funktion, ergibt die Richtung, in der die Schildkröte stehen müßte, um diesen Punkt zu erreichen (mit FORWARD) (8.6.3)
TRUE	wahr kein Befehl!
TYPE	Druckbefehl mit mindestens einem Parameter (s. PRINT), der nach dem Druck keinen Wagenrücklauf ausführt (der Cursor bleibt hinter dem letzten Zeichen) (3.1.3)
WAIT pause	ergibt eine Pause von pause/50 Sekunden (2.3)
WINDOW	setzt den Arbeitsbereich der Schildkröte so, daß der Bildschirm nur ein Fenster zeigt (3.1.7)
WORD wort1 wort2	Funktion, fügt die gegebenen Argumente zu einem Wort zusammen (5.2.1)
WORDP ''name	Funktion, ergibt wahr (true), wenn das Argument ein Wort ist
WRAP	setzt den Arbeitsbereich der Schildkröte so, daß sie auf der gegenüberliegenden Seite wieder hereinkommt, wenn sie den Bildschirm verläßt (3.1.7)
XCOR	Pseudovariable, enthält waagerechte Schildkrötenposition (8.6.1)
YCOR	Pseudovariable, enthält senkrechte Schildkrötenposition (8.6.1)
.BLOAD ''name adresse	lädt das File name an die angegebene Adresse (Maschinencode) (7.3/9.6)
.BSAVE ''name start laenge	sichert den angegebenen Speicherbereich als Code unter dem Namen name auf Band oder Cartridge (SETDRIVE) (7.3/9.6)

.CALL adresse	ruft ein Maschinencode-Programm auf, das bei adresse beginnt (7.3/9.6)
.CONTENTS	gibt alles aus, was Logo kennt, auch Prozeduren, Variablen und was sonst eingegeben wurde. Braucht viel Platz (6.1.3)
.DEPOSIT adresse n	schreibt den Wert n in die Speicherstelle adresse (9.4.3)
.EXAMINE adresse	Funktion, liest den Wert der Speicherstelle adresse
.PRIMITIVES	zeigt alle Logo-Grundwörter
.RESERVE n	Reserviert n Bytes für Maschinencode
.RESERVED	Funktion, liefert Start und Ende des reservierten Bereichs.
.SERIALIN	liest die serielle Schnittstelle des Interface 1 und ergibt jeweils ein Byte zwischen 0 und 255 (Zeichen)
.SERIALOUT	sendet ein Byte mit der gewählten Baudrate über die serielle Schnittstelle des Interface 1
.SETSERIAL n	setzt die Baudrate (Übertragungsrate Bit pro Sekunde) für die serielle Schnittstelle des Interface 1. Voreingestellt ist der Wert 9600. Erlaubte Werte können dem Interface 1 Handbuch entnommen werden

Anhang 2:

Meldungen des Logo-Interpreters

Bad file name	Unzulässiger Filename Es wurde versucht, einen Namen mit mehr als 10 Buchstaben zum Sichern auf Band zu benutzen.
I don't know how to ...	Ich kenne ... nicht (2.1.2) Es wurde eine nicht definierte Prozedur aufgerufen.
Not enough inputs to ...	Nicht genug Parameter für ... Ein Befehl verlangt mehr Angaben als gegeben wurden, z.B. PRINT ohne weitere Angaben.
Not enough items in ...	Der Parameter (Wort oder Liste) hat zuwenig Elemente
Not enough space to proceed	Nicht mehr genug Speicherplatz, um weiterzumachen Der Arbeitsspeicher ist voll. Abhilfe: RECYCLE.
Overflow	(mathematischer) Überlauf zu große Zahl als Ergebnis
STOPPED!	Prozedur unterbrochen Jede Anweisung (auch im Direktmodus) kann mit BREAK (CS + Space) abgebrochen werden.
Too many inside parantheses	Zuviele innere Klammern
Turtle out of bounds	Schildkröte verläßt den zulässigen Bereich
You're at toplevel	Du bist im Kommandomodus

You don't say what to do with ...	Du sagst nicht, was ich mit ... tun soll Eine Funktion wurde ohne Verarbeitungsbefehl aufgerufen.
... does not output to ...	Das Ergebnis einer Funktion kann nicht durch den angegebenen Befehl ausgegeben werden
... doesn't like ... as input	Der Befehl kann den Parameter nicht verarbeiten
... is already defined	... ist schon definiert
... is not true or false	... ist weder wahr noch falsch
... is not a word	... ist kein Wort
... defined	... ist definiert,
... has no value	... hat keinen Wert
... is a primitive	... ist ein Logo-Grundwort
... can't divide by zero	... kann nicht durch 0 teilen (math. nicht definiert)
... number too big	Zahl zu groß

Wenn diese Fehler während einer Programmausführung auftreten, werden sie mit dem Zusatz „in ..." gemeldet. Die Punkte stehen ersatzweise für die Prozedur, in der dieser Fehler auftrat.

Hinweis: Die Stelle im Programmablauf, an der der Fehler gemeldet wird, ist nicht notwendigerweise die Stelle, an der er entsteht. Es ist vielmehr der Zeitpunkt, zu dem der Interpreter bemerkt, daß ein Fehler aufgetreten ist. Beispielsweise könnte es sein, daß vergessen wurde, einer bestimmten Variablen einen Wert zuzuweisen. Diese Zuweisung muß natürlich vor dem Aufruf der Variablen erfolgen, d.h. ... has no value sagt uns, daß wir an geeigneter Stelle vor diesem Befehl/dieser Prozedur noch eine Variablenzuweisung einbauen müssen.

Sachwortregister

Die in Klammern hinter dem Wort genannten Zahlen sind die [Seitenzahlen, auf denen der Begriff erwähnt wurde].
LOGO-Wörter sind hier nicht verzeichnet, wohl aber deutsche [Suchwörter, die ein Primitive repräsentieren].

Hans-Peter Wagner

Btx auf Ihrem PC

1986, ca. 180 S., kart.,
ca. DM 40,—
ISBN 3-7785-1245-5

Die neuen Informations- und Kommunikationstechnologien haben einen tiefen Einfluß auf unser Arbeitsleben und sogar auf den Privatbereich. Diese Technologien stehen jedoch erst am Anfang.

Zwei Meilensteine auf diesem Weg stellen der Mikrocomputer für die Datenverarbeitung und Bildschirmtext für die Kommunikation dar. Beide bringen die neuen Technologien einer breiten Masse von Anwendern nahe. Das zeigen nicht zuletzt sinkende Preise und wachsende Benutzerfreundlichkeit. Die Kombination von PC-Technologie und BTX-Technologie wird aus zwei Richtungen forciert. Einmal soll der PC in Zukunft zum multifunktionalen Terminal werden, der als „Schaufenster" zur elektronischen Datenverarbeitung und Kommunikation dient. Schließlich hat man auch erkannt, daß die Fähigkeiten von BTX mit der normalen Fernbedienung kaum zur Geltung kommen.

Was liegt also näher, beide Bereiche miteinander zu verbinden. Tips für das derzeit Machbare und Perspektiven für die künftige Entwicklung will dieses Buch in einer einfachen, verständlichen Sprache aufzeigen.

Hüthig

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Hüthig

Ludwig Balla

dBASE III für Einsteiger und Fortgeschrittene

Nutzung und Programmierung des Datenbanksystems leicht gemacht

1986, ca. 200 S., kart.,
ca. DM 40,—
ISBN 3-7785-1253-6

dBASE III ist eine Weiterentwicklung des erfolgreichen Datenbank-Programmes dBASE II. Es zeichnet sich durch hohe Leistungsfähigkeit und große Flexibilität aus.

„dBASE III - Ein praktischer Ratgeber" beschreibt die Konzipierung von eigenen Datenbanken und die Umsetzung in ablauffähige Programme. Die dBASE-Befehle werden in einem konkreten Anwendungsfall erklärt, zusammen mit den Programmierungsgrundregeln wie Schleifen und Abfragen. So entstehen Programmbausteine, die der Anwender in eigene Projekte einbinden kann.

Die Unterschiede zwischen dBASE II und dBASE III werden erklärt, wobei besonders auf die Probleme eingegangen wird, die bei der Konvertierung alter dBASE II-Programme nach dBASE III entstehen.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Hüthig

Berthold Jurow

So schützt man Programme

Programmschutz für den Commodore 64

1986, ca. 200 S., kart.,
ca. DM 35,—
ISBN 3-7785-1274-9

Der Programm- und Kopierschutz ist nicht nur für kommerzielle Software-Firmen von immer größerer Bedeutung, sondern auch für jeden engagierten Programmierer ein faszinierendes Thema. Das Buch beschreibt alle relevanten Techniken vom einfachen LIST-Schutz bis hin zum professionellen Disketten-Kopierschutz am Beispiel des COMMODORE 64.

Mit Hilfe der beschriebenen Schutz-Verfahren findet der Leser auch eine umfassende Einführung in die interne Struktur des COMMODORE 64, in den Aufbau und Arbeitsweise des Betriebssystems sowie in das DOS der VC-1541. Dabei wird gezeigt, wie das Betriebssystem und das DOS nach persönlichen Wünschen manipuliert werden können.

Anhand von vielen lauffähigen Programmen wie LIST-Schutz, AutoStart, Kodierverfahren, bis hin zum kompletten Disketten-Kopierschutzgenerator zeigt der Autor, wie Programme gegen den unbefugten Zugriff geschützt werden können.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1