Schneider
COMPUTER DIVISION

BASIC
Lehrbuch

Schneider
COMPUTER DIVISION

BASIC Lehrbuch

Teil 2 MEHR BASIC

von Jan Padwick und George Tappenden

Schneider Computer Division
Silvastraße 1
8939 Türkheim

Schneider BASIC

Lehrbuch und Cassetten
Teil 2: Mehr BASIC

SW 156

Erste Ausgabe 1985
Originalausgabe in Englisch

Deutsche Bearbeitung: ESCON GmbH, Freising

INHALT

VORWORT

Dies ist der zweite Teil eines Selbststudium-Kurses über die Programmierung in BASIC auf dem Schneider Personal Computer CPC464. Während Teil 1 für Anfänger vorgesehen war, ist dieser Teil für alle geeignet, die mit den Grundlagen der Programmierung vertraut sind und die meisten BASIC-Befehle bereits verstehen.

Die beiden Datencassetten, die diesen gedruckten Text begleiten, sind wichtiger Bestandteil dieses Kurses.

Die Datencassette A enthält:

- ☐ Programme zur Demonstration der beschriebenen Techniken
- ☐ Spiele zu Ihrem Vergnügen und zur Darstellung der Möglichkeiten des CPC464

Die Datencassette B enthält:

- ☐ Lernziel-Kontrolltests zur Sicherstellung, daß Sie das vorhergehende Kapitel verstanden haben.

Der dritte Teil dieser Serie behandelt die fortgeschrittensten Techniken des Schneider BASIC und ist für erfahrene Programmierer gedacht.

Kapitel 1
HIER SIND WIR WIEDER

Wenn Sie Teil 1 dieser Serie durchgearbeitet haben, dann haben Sie eine ganze Reihe von Schneider-BASIC Anweisungen kennengelernt und bestimmt schon Ahnung von den Möglichkeiten des Schneider Farbcomputers CPC464. Mit vielen Erwartungen haben Sie bestimmt versucht, mit Ihrem neuen Wissen einige Programme selbst zu erstellen und hatten die Genugtuung, daß Sie erfolgreich abliefen – zwar nicht auf das erste Mal – aber sie liefen. Dieser Teil des Kurses beabsichtigt, das im ersten Teil gelernte zu festigen und einen erweiterten BASIC-Befehlsvorrat zu vermitteln.

Sollten Sie Teil 1 übersprungen haben, so wollen wir hier noch einmal anmerken, daß das Schneider-BASIC nicht notwendigerweise so arbeitet, wie auf anderen Computern und umgekehrt. Dies kommt daher, weil es viele Anweisungen und Funktionen enthält, die auf anders konstruierten Anlagen nicht verfügbar sind.

Jedesmal, wenn ein neues Schlüsselwort oder eine Erweiterung einer vorher behandelten Anweisung eingeführt wird, wird dies in der äußeren Spalte vermerkt, damit Sie leicht zurück blättern können, um Ihr Gedächtnis aufzufrischen.

WIE WIRD DIESES BUCH BENUTZT?

Obwohl beim Schreiben dieses Buches Wert darauf gelegt wurde ein Minimum an Fachausdrücken zu verwenden, wäre es ziemlich ermüdend, wenn wir beispielsweise einen Spaten als ein „holzbestieltes, mit einem gehärteten Stahlblatt versehenes Gartenwerkzeug" bezeichnet hätten. Wie bei jedem Spezialgebiet ist es unmöglich, auf bestimmte Ausdrücke mit spezieller Bedeutung zu verzichten, und die Datenverarbeitung macht dabei keine Ausnahme. So wollen wir mit einer Übersicht der wichtigsten Anweisungen beginnen und die Befehle des Schneider-BASIC beschreiben.

Jedes Kapitel dieses Buches nimmt ein oder zwei Abende in Anspruch. In der Regel enthält es:

- ☐ Erklärungen in gedruckter Form
- ☐ praktische Arbeit am Computer
- ☐ Beispiele zur selbständigen Programmierung

Es gibt Übungen zur Wiederholung des Gelernten und es gibt bei jedem Kapitel Lernzielkontrollen.

Überspringen Sie nie ein Kapitel. Durch das ganze Buch zieht sich eine schrittweise Erweiterung des Wissens, wobei immer auf das vorhergehende Kapitel aufgebaut wird. Wenn Sie meinen, daß ein Kapitel oder ein Abschnitt schwierig erscheint, lesen Sie ihn gleich noch ein- oder zweimal durch und arbeiten Sie ihn dann langsam ab. Die Übungen und Programme, die Sie selbst eingeben sollten, sollen Ihnen helfen, die ausgeführten Themen und Techniken zu verstehen und Sie sollten dies nicht unterschlagen. Vergewissern Sie sich durch die Lernzielkontrollen, daß Sie alles verstanden haben.

Nachdem Sie sich diesen Teil des Kurses erarbeitet haben, sollten Sie in der Lage sein, Ihre eigenen Computerspiele, Hobbyverarbeitung oder Geschäfts-Software zu programmieren – nicht ganz so professionell, aber Übung macht den Meister. Der dritte Teil des Kurses wird Sie weiter in die tieferen Einzelheiten des Schneider-BASIC einführen.

BEFEHLS-BESCHREIBUNGEN

Im ersten Teil lernten wir, daß ein Befehl aus einem Schlüsselwort und (meistens) einem oder mehreren Argumenten, wie beispielsweise beim Befehl zum Zeichnen einer Linie, besteht:

```
DRAW x,y
```

wobei die Argumente ‚x' und ‚y' die Graphik-Koordinaten der Bildschirmposition bestimmen, zu der die Linie gezeichnet werden soll. Wie Sie bestimmt wissen, erfolgt bei Eingabe eines falschen Zeichens oder bei Weglassen eines Zwischenraums oder eines Zeichens die Meldung „SYNTAX ERROR"! Aber was heißt Syntax? Und warum muß mitten in einem Befehl ein Komma geschrieben werden? Dies wird vor dem Ende dieses Kapitels alles klar.

Syntax

Im Deutschen ist der Satz „Der Schwanz wurde gewackelt vom kleinen Hund" ungeschickt formuliert. Es gibt (hoffentlich) nicht viele unter uns, die die Sprachregeln derart mißachten. Die Syntax, das heißt die Regel, nach welcher die einzelnen Worte im Satz zu plazieren sind, wurde nicht beachtet.

Ebenso besteht eine Überlegenheit des menschlichen Verstandes über den Computer, so daß dieser die Aussage noch verstehen kann und Unebenheiten ausgleicht.

Schlimmer ist es, wenn die Verbindung der einzelnen Satzteile unklar ist:

Schwanz / wackelt / Hund

Wenn das Verbindungswort „durch" nicht an der richtigen Stelle angegeben ist, so ist nicht klar, wer was wackelte.

Im BASIC sind die Regeln bedeutend besser bestimmt. Bei IF-THEN-ELSE ist es keine Frage, wohin bei Übereinstimmung verzweigt werden soll. Die gleiche Aussage wäre in der Form IF-ELSE-THEN zu erreichen; diese zweite Form gibt es jedoch in BASIC nicht. In gleicher Weise macht es in einem deutschen Text kaum einen Unterschied, wenn zwischen zwei Worten der Zwischenraum vergessen wird; es ist genauso verständlich, lediglich etwas schwieriger zu lesen. Wenn Sie jedoch einen Zwischenraum an einem kritischen Punkt beim Schreiben eines BASIC-Programms vergessen, wird der CPC464 nicht in der Lage sein zu verstehen, was Sie ihm mitteilen wollten und wird oft den Ablauf des Programms zurückweisen, bis Sie den Fehler behoben haben.

Es kommt in diesem Buch eine Vielzahl neuer und oft komplizierter Befehle und Funktionen auf Sie zu, die Sie lernen müssen. Um Zeit und Platz zu sparen, wird eine Art Kurzschrift zur Beschreibung der Syntax jedes Befehles verwendet. Dies bringt einige ungewöhnliche Darstellungen und Punktsetzungen mit sich, die weiterer Erklärungen bedürfen. Studieren Sie die folgenden Erläuterungen sorgfältig. Sie können jederzeit darauf zurückgreifen, bis die Ausdrücke Ihnen geläufig sind.

Befehlserläuterungen

Klammern
Bestimmte Befehle und Funktionen erfordern, daß die Argumente in Klammern eingeschlossen sind, z.B. CHR$(97). Runde Klammern werden auch verwendet, um die Reihenfolge arithmetischer Operationen zu bestimmen.

In diesem Buch verwenden wir zur Beschreibung von Befehlen noch zwei weitere Arten von Klammern. Die erste ist die „spitze Klammer" „><", was bedeutet, daß der eingefügte Text anderswo erläutert ist. Die zweite Art, die „eckige Klammer" „[]", sagt Ihnen, was wahlweise noch angegeben werden kann.

Standard
Wenn der CPC464 angeschaltet oder zurückgesetzt wird, werden eine Reihe von Dingen, wie z.B. der Bildschirm, der Bildschirmrand und die Textfarben automatisch gesetzt. Zusätzlich gibt es Befehle, wie DRAW und PLOT, welche, wenn INK einmal angegeben wurde, in der weiteren Folge dieses Argument nicht mehr benötigen. Solche Werte sind als Standardwerte bekannt

und wir werden später noch sehen, daß es viele Befehle gibt, denen Standard- oder Ersatzwerte zugewiesen sind.

Ausdruck
siehe ganzzahliger Ausdruck, numerischer Ausdruck oder Zeichenketten-Ausdruck

Ganzzahliger Ausdruck
Dies ist ein numerischer Ausdruck, der ein reelles oder ganzzahliges Ergebnis darstellt, das auf die nächste ganze Zahl gerundet ist.

Ganze Zahl
Dies ist eine Zahl, die keinen Dezimalteil besitzt (integer).

Schlüsselwort
Ein für BASIC reserviertes Wort, das ein Befehl, wie z.B. RUN, MOVE, PRINT, oder BASIC-Funktion, wie RND und INT sein kann.

Numerischer Ausdruck
Dies kann eine von vier Bedeutungen haben:

- ☐ eine Zahl, wie 20473.611
- ☐ eine numerische Variable, wie „TOTAL"
- ☐ das Ergebnis einer numerischen Operation, wie TOTAL/252*PROZENT
- ☐ eine Funktion, wie „LOG(x)"

Reelle Zahl
Zahlen, die Dezimalstellen aufweisen (real).

Trennzeichen
Diese werden in BASIC benutzt, um Anfang und Ende von Schlüsselwörtern und ihren Argumenten zu bestimmen. Die wichtigsten sind:

Komma	(,)
Strichpunkt	(;)
Zwischenraum	( )

In unserer Kommandobeschreibung verwenden wir auch den Doppelpunkt (:), um anzuzeigen, daß ein Argument aus einer Liste von Elementen besteht.

Zeichenketten-Ausdruck
Eine alphanumerische Information, welche eine der folgenden Möglichkeiten darstellt:

- ☐ eine Zeichenkette, in Anführungszeichen eingeschlossen, z.B. „Name"
- ☐ eine Zeichenketten-Variable, wie art$
- ☐ das Ergebnis einer Zeichenkettenoperation, wie z.B. art$ + „Name"
- ☐ eine Zeichenketten-Funktion, z.B. LEFT$ (A$,3)

Beispiele eines typischen Befehls

Betrachten Sie sorgfältig folgende Anweisung:

```
DRAW <x Koordinate>,<y Koordinate>
  [,<ink>]
```

wobei

<x Koordinate> = ganzzahliger Ausdruck
<y Koordinate> = ganzzahliger Ausdruck
<ink> = ganzzahliger Ausdruck

bedeutet.
Dabei wurden beide, <x Koordinate> und <y Koordinate>, als Ausdrücke beschrieben, die wir an dieser Stelle entweder als numerische Variable oder als numerische Werte einsetzen können, oder als eine Kombination von Werten, Operatoren und Variablen, die einen numerischen Wert ergeben. Diese beiden ganzzahligen Ausdrücke, sowie das Trennzeichen (,) müssen angegeben sein, ansonsten bringt der CPC464 die Meldung „SYNTAX ERROR".

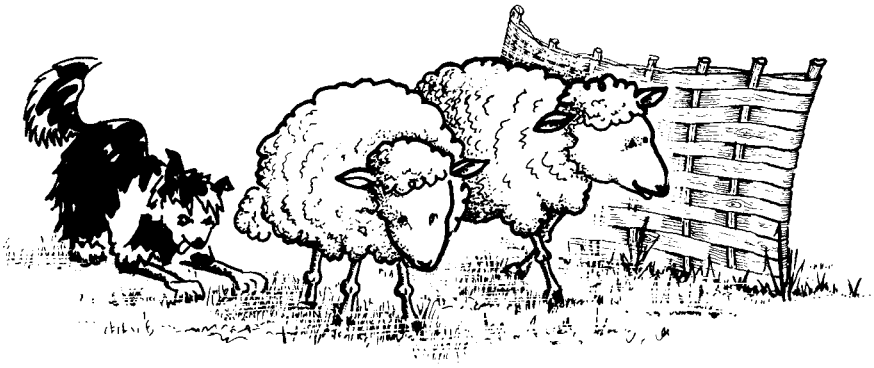
Im Gegensatz dazu ist ',<ink>' in eckigen Klammern angegeben. Wenn Sie diesen Teil des Ausdrucks weglassen, wird vom CPC464 angenommen, daß Sie mit der momentanen Farbe als Standardwert einverstanden sind und es erfolgt keine Fehlermeldung. Als allgemeine Regel gilt, wo eine wahlweise Angabe fehlt, wird dafür ein Standard- oder Ersatzwert zugeordnet.

So kann nun jeder Schneider-BASIC-Befehl in dieser Art beschrieben werden. Durch Anwendung der Klammern und Trennzeichen und jetzt durch die Kenntnis der unterschiedlichen Arten von Ausdrücken, müssen wir nicht allzuviel Zeit für die Einzelheiten eines Befehls verschwenden. Beachten Sie z.B., daß unser DRAW-Befehl ganzzahlige Ausdrücke für die Koordinaten benötigt. Dies bedeutet nicht, daß hier nur ganzzahlige Werte verwendet werden können, sondern der CPC464 rundet die Werte dieser Argumente auf den nächsten Bildschirmpunkt. So brauchen Sie selbst den Dezimalteil dieses Ausdrucks nicht zu beachten.

Wenn Sie in Ihrem Benutzerhandbuch für den CPC464 nachschauen, werden Sie feststellen, daß wir dort genau dieselben Konventionen anwenden. So können Sie schnell die Anwendung jedes Befehls verstehen, der nicht in den folgenden Kapiteln erläutert ist.

SPIELZEIT

Laden Sie das erste Programm„SCHAFE" von Ihrer Datencassette A. Sie können dann eine gute halbe Stunde opfern, als Bauer Ihren Schäferhund so zu dirigieren, daß dieser Ihre ziemlich dummen Schafe in ihr Gatter jagt. Viel Spaß dabei!



TEST

Für alle, die den Teil 1 nicht barbeitet haben, wäre es sinnvoll, die erste Lernzielkontrolle „SAT1" durchzuführen, um sicherzugehen, daß es keine offenen Punkte im Schneider-BASIC gibt, welche vor dem Kapitel 2 noch durchgearbeit werden sollten.

Auch wenn Sie Teil 1 studiert haben sollten, würde eine kleine Überprüfung nicht schaden!

Kapitel 2
EINZIEHEN

In diesem Kapitel werden wir viele alte Bekannte aus dem Teil 1 treffen: Zum Beginn das Haus. Unser ursprünglicher Entwurf war ziemlich langatmig mit all den MOVE's, so daß es jetzt an der Zeit ist, dies kürzer und ordentlich darzustellen. Dann werden wir uns die Variablen näher betrachten.

Bevor wir anfangen wollen wir jedoch zunächst einen Blick auf die Kommandos werfen, die uns zum Programmeinstieg das Leben erleichtern.

ÜBER UND UNTER DER LINIE

Wenn Sie zur Ausgabe langer Programme am Bildschirm LIST eingegeben haben, war es wahrscheinlich etwas langweilig auf das Stück zu warten, das Sie sich ansehen wollten. Das Kommando läßt aber ein wahlweises Argument zu, welches die Sache enorm beschleunigt. Das Format sieht so aus:

```
LIST [<Zeilenbereich>]
```

LIST

wobei <Zeilenbereich> eine von vier Möglichkeiten darstellen kann. Am besten verstehen Sie alles, wenn Sie es wie folgt ausprobieren. Laden Sie das nächste Programm CHATEAU, von der Datencassette A, aber starten Sie es noch nicht. Geben Sie ein

```
LIST 100
```

Wie Sie sehen können, wird genau die Zeile 100 am Bildschirm ausgegeben.

```
LIST -100
```

Diesesmal gibt Ihnen der CPC464 alle Zeilennummern bis einschließlich 100 aus. Jetzt müssen Sie nicht mehr schnell die ESC-Taste drücken, um die Auflistung an der gewünschten Stelle abzubrechen.

Für das dritte Beispiel geben Sie

```
LIST 100-200
```

ein, um alle Zeilen zwischen und inclusive 100 und 200 zu erhalten. Wenn Sie sich vorstellen, daß Sie einen unbedeutenden Fehler in diesem Teil des Porgramms haben, können Sie leicht feststellen, wie nützlich dies ist, um lediglich diesen Abschnitt auf dem Bildschirm zu haben, wenn Sie ihn verändern wollen.

Und zuletzt probieren Sie bitte noch

```
LIST 700-
```

wobei Ihnen alle Zeilen von 700 bis zum Ende des Programms ausgegeben werden. Dies ist besonders nützlich, wenn Sie sich an die gängige Praxis halten, Ihre Unterprogramme an das Programmende zu legen und bei einer bestimmten Zeilennummer zu beginnen.

DELETE

Das nächste Kommando ist DELETE. Die Syntax sieht wie folgt aus:

```
DELETE <Zeilenbereich>
```

Sie werden feststellen, daß das Argument hier nicht wahlfrei ist, ansonsten wird der Zeilenbereich genauso behandelt, wie beim LIST-Kommando. Wenn Sie beispielsweise

```
DELETE 100-200
```

eingeben, entfernen Sie diese Zeilen aus dem Speicher, so daß Sie beim nächsten LIST nicht mehr auf dem Bildschirm erscheinen.

Bei Anwendung dieses Kommandos ist Vorsicht geboten. Wenn Sie keine Kopie des Programms gesichert haben, können Sie unglücklicherweise ein hartes Stück Arbeit verlieren. Es ist jedoch von Nutzen, wenn Sie Teile eines bestehenden Programms verwenden wollen. Laden Sie dann das Programm und löschen Sie alles, was Sie nicht haben wollen.

RENUM

Das nächste nützliche Kommando dieser Gruppe ist RENUM. Sie werden jetzt sehen, daß wir eine Art Kurzschrift benötigen, um die Syntax zu erklären:

```
RENUM [<neue Zeilennummer>]
  [,[<alte Zeilennummer>][,<Schrittweite>]]
```

Diese wahlweisen Argumente ermöglichen es Ihnen, die Zeilen Ihres aktuellen Programms neu durchzunumerieren, beginnend bei der Zeile <alte Zeilennummer> und dem Anfagnswert <neue Zeilennummer> im Abstand von <Schrittweite>. Wenn Sie irgend ein Argument nicht angeben, werden die Standardwerte angenommen. Diese entsprechen der Angabe

```
RENUM 10,,10
```

Das ausgelassene Argument sagt lediglich aus, daß mit der ersten Zeile des Programms die Neunumerierung beginnen soll.

Wenn Sie jetzt, nachdem Sie mit diesen Kommandos gespielt haben, von CHATEAU noch etwas übrig behalten haben, können Sie die verschiedenen wahlfreien Argumente ausprobieren, bevor Sie lediglich RENUM für sich alleine eingeben, um die Standardwerte der Argumente zu erhalten. Es ist zu beachten, daß RENUM automatisch die Verweise in GOTO und GOSUB angleicht.

AUTO

Abschließend wird ein Kommando behandelt, welches Ihnen erspart, jedesmal die Zeilennummer mit einzugeben. Dies ist AUTO und wird folgendermaßen angewendet:

```
AUTO [<Zeilennummer>][,<Schrittweite>]
```

Das Argument <Zeilennummer> ist die Zeile, mit der begonnen werden soll und <Schrittweite> ist der Nummernsprung zwischen den Zeilen. Wenn Sie beispielsweise eingeben:

```
AUTO 100,5
```

wird der CPC464 mit 100 beginnen, einen Zwischenraum lassen und den Cursor so stellen, daß Sie den Befehl in diese Zeile eintragen können. Wenn Sie die ENTER-Taste drücken, geschieht wiederum dasselbe, außer daß die Zeilennummer jetzt 105 ist.

Die automatische Numerierung wird durch das Drücken der ESC-Taste unterbrochen.

Wenn sich im Speicher bereits ein Zeile mit derselben Nummer befindet, wie sie gerade erzeugt wurde, so wird ein Stern (*) als Warnung ausgegeben. Wenn vor dem Drücken der Eingabetaste nichts unternommen wird, dann ersetzt die neue Zeile die alte.

VARIABLE VARIABLEN

Die Art und Weise, in der die Variablen in Teil 1 beschrieben wurden – benannte Speicherplätze für Zahlen und Zeichenketten – war nur ein Teil der Geschichte. Tatsächlich gibt es zwei Arten von numerischen Variablen, reelle und ganzzahlige; sowohl diese, wie auch die Zeichenkettenvariablen können subskribiert werden.

Reelle numerische Variablen

Reelle Zahlen oder Ausdrücke können einen Dezimalteil enthalten oder kleiner als Null sein. Eine andere Bezeichnung dafür ist Gleitkomma-Variable, da der Dezimalpunkt in seiner Position nicht fest vergeben ist. Es folgen Beispiele reeller Zahlen:

1234826.01
0.000000154768119
2.3

Wenn Sie eine Variable angeben, entweder durch LET oder implizit in einem Ausdruck, wird standardmäßig „reell" angenommen. Um in Ihren Programmen Klarheit zu haben, können Sie dies durch Anhängen eines „!" sicherstellen, z.B.

```
LET Laenge!=Messung*Koeff*0.0001
```

Ganzzahlige numerische Variablen

Wenn Sie nur mit ganzen Zahlen arbeiten wollen, d.h. keine Dezimalstellen vorhanden sind, ist es notwendig, dies durch das Hinzufügen eines Prozentzeichens „%" an den Namen zu bestimmen, z.B.

```
LET Prozent%=100*Gewinn/Kosten
```

Die Variable „Prozent%" enthält immer nur einen ganzzahligen Wert, und alle eventuellen Dezimalstellen werden zum nächsten ganzzahligen Wert gerundet.

Subskribierte Variablen

Diese Form wird bisweilen auch als Liste bezeichnet, weil sie statt eines Elements eine Vielzahl gleichartiger Elemente enthalten kann. Diese können reelle, ganzzahlige oder Zeichenketten-Variablen sein und werden folgendermaßen beschrieben:

```
<Variablennme>(Liste von: <Subskript>)
```

wobei <Subskript> ganzzahlige Ausdrücke bedeuten. Eine ganze Liste von Variablen kann somit unter einem gemeinsamen Namen, lediglich mit unterschiedlichen Subskripten, gespeichert werden.

Als Beispiel nehmen wir die ersten sechs Eishockey-Teams der 2. Bundesliga Süd von Ende Januar 1985:

Mannschaft	*Punkte*
SV Bayreuth	50
Augsburger EV	42
EV Füssen	33
EC Bad Tölz	31
VERE Selb	30
EV Landsberg	29

Nun unterstellen wir, daß wir die Information so abspeichern wollen, daß sie wöchentlich auf den neuesten Stand gebracht werden kann und die Liste in absteigender Reihenfolge ausgegeben wird. Wenn wir die Tabellenplätze von 1 bis 6 durchnumerieren, kann die Punktzahl für die auf diesem Platz stehende Mannschaft in einer subskribierten Variablen, genannt „Punkte(n)", gespeichert werden, wobei „n" dem Ligaplatz entspricht. Dabei gehen wir folgendermaßen vor:

Variable	*Inhalt*
Punkte (1)	50
Punkte (2)	42
Punkte (3)	33
Punkte (4)	31
Punkte (5)	30
Punkte (6)	29

Eine solche Liste wird auch als Tabelle bezeichnet und dies ist auch der Ausdruck, den wir von nun an dafür verwenden (exakt ausgedrückt ist unsere Punktetabelle eine eindimensionale Tabelle – warum, werden wir später sehen).

CHATEAU

Ich liege wahrscheinlich richtig mit meiner Vermutung, daß viele von uns dies als die französische Bezeichnung für „Schloß" erkannten. Was Sie vielleicht nicht wissen ist, daß damit auch eine hübsche Villa in einer der reichsten Gegenden Frankreichs bezeichnet wird.

Wenn Sie das Programm im ersten Teil dieses Kapitels bereits zerstört haben, können Sie die Cassette zurückspulen und das Programm erneut laden. Wir haben nun die Absicht Ihnen zu zeigen, wie ein Programm schrittweise entwickelt werden kann, so daß unser ursprüngliches „HAUS"-Programm mehrere Umbauten auf sich nehmen muß, um es in diese anspruchsvolle Klasse zu bringen.

Nach dem Laden des Programms können wir es ablaufen lassen. Es sieht fast aus wie das „HAUS"-Programm, finden Sie nicht? Wenn Sie jedoch auf die folgende Auflistung des Programms schauen, werden Sie feststellen, daß es vollkommen neu überarbeitet wurde.

```
100 ' Landhaus
110 ' (verbesserte VILLA)
120 '
130 ' DA 17/9/84
140 '
150 MODE 0 : BORDER 12
160 INK 0,12 : INK 1,3 : INK 2,6 : INK 3,17
170 PAPER 0
180 '
190 ' Umrisse
200 '
210 READ x,y
220 IF y=-1 THEN 290
230 IF x=0 THEN colour=y : GOTO 210
```

```
240 IF x<0 THEN MOVE -x,-y ELSE DRAW x,y,colour
250 GOTO 210
260 '
270 ' Zaun
280 '
290 FOR F=0 TO 620 STEP 20
300 MOVE F,0:DRAW F,60
310 NEXT F
320 MOVE 0,45:DRAW 620,45
330 '
340 ' Fenster-Rahmen
350 '
360 groesse=18
370 FOR i=1 TO 16
380 READ x,y:MOVE x,y
390 DRAWR 0,groesse : DRAWR groesse,0
400 DRAWR 0,-groesse: DRAWR -groesse,0
410 NEXT
420 '
430 END
440 '
450 ' Data-Angaben fuer die Ecken
460 ' paarweise: negative = MOVE
470 '           : 0,x      = Farbwechsel
480 '           : 0,-1     = Ende
490 '
500 DATA    0,   1
510 DATA -100,-50,  100,250, 400,250, 400,50,  100,50
520 DATA -400,-250, 600,250, 600,50,  400,50,  400,250
530 DATA 500,350,   600,250, 400,250
540 DATA -100,-250, 200,350, 500,350
550 '
560 ' Tuere
570 '
580 DATA 0,2
590 DATA -225,-50, 225,140, 275,140, 275,50
600 '
610 ' Grosse Fenster
620 '
630 DATA 0,3
640 DATA -120,-70,  120,130, 180,130, 180,70,  120,70
650 DATA -120,-170, 120,230, 180,230, 180,170, 120,170
660 DATA -320,-170, 320,230, 380,230, 380,170, 320,170
670 DATA -320,-70,  320,130, 380,130, 380,70,  320,70
680 DATA 0,-1
690 '
700 ' Kleine Fenster
710 '
720 DATA 130,78,  156,78,  130,103, 156,103, 130,178
730 DATA 156,178, 130,203, 156,203, 330,78,  356,78
740 DATA 330,103, 356,103, 330,178, 356,178, 330,203, 356,203
```

Als erstes werden Sie bemerkt haben, daß ein Apostrophzeichen (') an Stelle 'REM' benutzt wurde. Es bedeutet dasselbe; d.h. alles, was danach kommt, wird ignoriert. Wie Sie selbst sehen können, kann das Listenbild damit übersichtlicher gestaltet werden.

Und nun kommen zwei neue, für Sie genauso wichtige Befehle: DATA und READ. In unserem HAUS-Programm in Teil 1 waren eine Menge MOVE-Befehle hintereinander, um den Graphik-Cursor an den richtigen Platz zu stellen, bevor der GOSUB aufgerufen wurde. Auf eine einfachere Art können wir dasselbe erreichen, wenn wir alle Positionierungskoordinaten an die gleiche Stelle verweisen und in den MOVE- und DRAW-Befehlen Variablen verwenden. Dies ist genau das, was READ und DATA Ihnen ermöglicht.

Die Syntax von DATA ist:

DATA

```
DATA <Liste von:<Konstante>>
```

Es kann eine Liste von numerischen oder Zeichenkettenkonstanten angegeben werden, die durch Kommata getrennt werden. Es wird in der Tat alles angenommen, sogar eine Zeile mit Unsinn darin wird akzeptiert, solange sie kein Komma enthält.

Die Syntax für READ ist:

READ

```
READ <Liste von:<Variable>>
```

Eine <Variable> kann eine numerische oder eine Zeichenkettenvariable sein, die voneinander jeweils durch ein Komma getrennt sind.

Die folgenden Zeilen zeigen, wie diese beiden Befehle zusammenwirken.

```
10 FOR x=1 TO 4
20 READ N$
30 PRINT N$
40 NEXT
50 DATA Marlene,Vera,Fred,Peter
```

Jedesmal, wenn das Programm einen READ-Befehl ausführt, wird auf die Zeichenkettenkonstante in der DATA-Anweisung geschaltet und diese in die Variable N$ übertragen. So wird bei x=1 ‚Marlene', bei x=2 ‚Vera', bei x=3 ‚Fred' und bei x=4 ‚Peter' ausgegeben.

Es wird Sie nun interessieren, ob es eine Möglichkeit gibt, die Wiederholung von DATA-Anweisungen zu vermeiden, wenn sie mit genau denselben Werten öfter benötigt werden. Das Schlüsselwort, das Ihnen dabei helfen kann ist RESTORE. Hier ist die Syntax:

```
RESTORE [<Zeilennummer>]
```

RESTORE

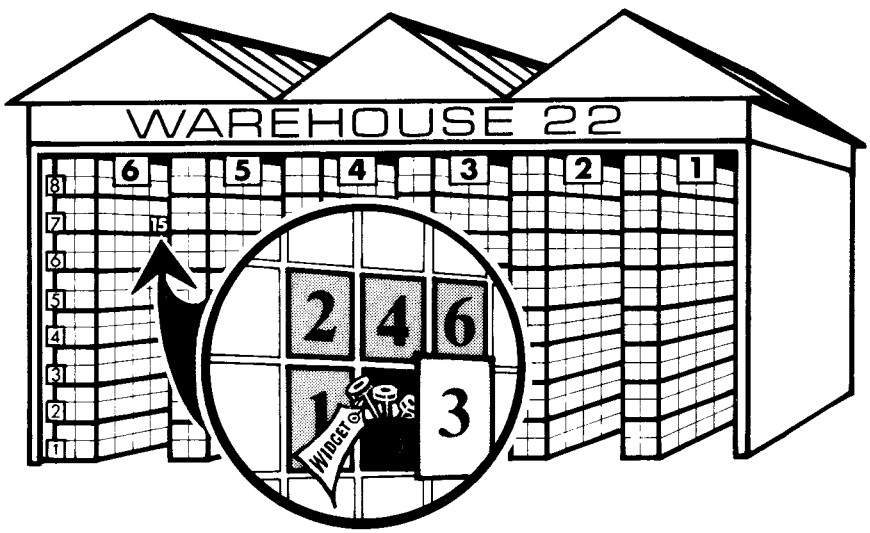
Um zu verstehen, was das Argument <Zeilennummer> bedeutet, müssen Sie sich daran erinnern, was im CPC464 jedesmal passiert, wenn er einen READ-Befehl durchführt. Ganz egal wieviel DATA-Anweisungen es gibt oder wo diese im Programm stehen, werden sie eine nach der anderen gelesen, in der Reihenfolge, in der sie angegeben wurden. Nach jedem READ-Befehl wird der Zeiger auf die nächste Konstante gesetzt, die beim nächsten READ-Befehl an der Reihe ist.

Das wahlfreie Argument in der RESTORE-Anweisung erlaubt es Ihnen, diesen Zeiger auf die erste Konstante einer DATA-Anweisung in einer bestimmten Zeilennummer zu setzen. Dies bedeutet, daß Sie während der Programmausführung Datenkonstanten wiederholen oder gezielt auswählen können. Wenn Sie das Argument nicht angeben, wird der Zeiger auf die erste DATA-Anweisung des Programms gesetzt.

TABELLEN

Wie versprochen, behandeln wir jetzt Tabellen. Früher in diesem Kapitel sahen wir, daß durch Verwendung subskribierter Variablen Tabellen aufgebaut werden können. Dadurch haben Sie den Effekt, eine gewisse Anzahl von Schubladen mit demselben Namen ansprechen zu können.

Sie haben z.B. eine eindimensionale Tabelle ‚NÄGEL' (a), wobei ‚a' die Nummer eines bestimmten Regals ist. Die Ausgabe von NÄGEL (3) ergibt die Anzahl von Nägeln im Regal 3. Wir können dies noch weiter unterteilen und kommen auf eine zweidimensionale Tabelle NÄGEL (a,b), wobei ‚a' die Nummer des Regals und ‚b' die Nummer des Kartons ist. Die Ausgabe von NÄGEL (15,3) zeigt Ihnen dann die Anzahl von Nägeln im Karton Nummer 3 des Regals mit der Nummer 15.



Und bevor Sie nun fragen, möchte ich Ihnen schon antworten, daß Sie auch dreidimensionale Tabellen behandeln können; aber darüber werden wir uns in diesem Buch nicht auslassen. In der Tat läßt der CPC464, wie Sie schon aus der Abbildung entnehmen können, soviele Dimensionen in einer Tabelle zu, wie Sie wollen (Zeilenlänge und Speicherplatz müssen beachtet werden).

Wie wenden Sie nun eine Tabelle an? Geben Sie Folgendes ein und lassen Sie es ablaufen:

```
10 Regal(5)=47
20 FOR n=0 TO 20
30 PRINT Regal(n);
40 NEXT
```

Beachten Sie, daß jede Tabellenposition, die als Subskript angegeben wird, als Element bekannt ist, wobei ab Element 0 gezählt wird. Die Zeile 10 weist dem Subskript 5, d.h. dem sechsten Regal den Wert 47 zu und die restlichen Zeilen dienen lediglich zum Ausdrucken der gesamten Tabelle. Was auf dem Bildschirm nun erscheint, sieht folgendermaßen aus:

```
0  0  0  0  0  47  0  0  0  0  0
Sbuscript out of range in 30
```

Die Fehlermeldung erscheint, weil ein neuer Befehl zu lernen ist; aber in einer Minute haben wir es hinter uns. Sie sehen, daß der CPC464 den Anfangswert der subskribierten Variablen auf Null gesetzt hat. Die ersten 5 Nullen stellen somit die Werte der Elemente 0, 1, 2, 3 und 4 dar, und die letzten fünf Nullen die Werte der Elemente 6 bis 20. Was passierte nun bei Element 11? Gut, wollen wir es so sagen: Der CPC464 hat angenommen, daß jede Tabelle, die wir verwenden, aus 10 oder weniger Elementen besteht, was auch die Fehlermeldung (Subskript außerhalb des gültigen Bereichs in Zeile 30) erklärt.

DIM

Für Tabellen mit mehr als 10 Elementen müssen wir die DIM-Anweisung anwenden, um die Maximalgröße der Tabelle zu definieren. Die Syntax sieht wie folgt aus:

```
DIM <Liste von:<subskribierte Variable>>
```

wobei <subskribierte Variable> auch für

```
<Variablenname>(Liste von:<ganzzahliger
  Ausdruck>
```

stehen kann. Jeder <ganzzahlige Ausdruck> gibt den höchsten erlaubten Wert für das entsprechende Subskript an.

Auf die Fehlermeldung können wir nun auf zwei verschiedene Arten reagieren. Die einfachste ist, die Schleife ganz einfach bei 10 anstatt bei 20 zu stoppen. Wenn Sie nun tatsächlich eine Tabelle mit 21 Elementen bearbeiten wollen, müssen Sie die folgende Zeile einfügen:

```
5 DIM Regal (20)
```

Versuchen Sie es!

SPIELZEIT

Das nächste Programm STADT läßt Sie nicht nur ein Haus, sondern eine Menge Häuser bauen.

TEST

Prüfen Sie ob Sie alles von dem neuen Lehrstoff verstanden haben, indem Sie SAT2 aufrufen, bevor Sie an das nächste Kapitel gehen.

Kapitel 3
MALEN MIT ZAHLEN

PEN

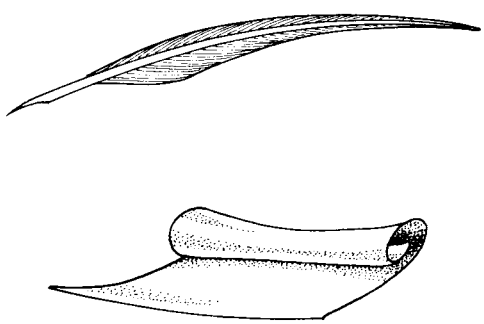
PAPER

INK

Wenn Sie bezüglich des Zusammenhangs zwischen PEN, PAPER und INK bis jetzt verwirrt waren, dann müssen Sie das Folgende lesen. Wenn nicht, dann lesen Sie es trotzdem, weil die Materie wahrscheinlich etwas komplexer ist als Sie annehmen!

Der CPC464 kann 27 verschiedene Farben ausgeben (siehe nachfolgende Tabelle), die in 16 unterschiedlichen Farbtöpfen bereitgestellt werden können. Die Anzahl der Farbtöpfe hängt jedoch vom gewählten Modus ab.

Im MODE 0 können wir 16 Farbtöpfe, im MODE 1 vier und im MODE 2 gerade noch zwei Farbtöpfe benennen. Sie müssen sich nun vorstellen, daß Sie bei jedem PRINT zur Ausgabe des Textes eine Feder und ein Blatt Papier verwenden, was beides zuvor in einen Farbtopf getaucht wurde.



Schalten Sie nun Ihr Gerät an oder machen Sie einen Warmstart (SHIFT/CTRL/ESC), dann kommen Sie in MODE 1 und Ihnen stehen somit 4 Farben gleichzeitig auf dem Bildschirm zur Verfügung. Der Bildschirm ist blau, da dies für die Farbe PAPER (Papier) gesetzt wurde, was als Standard für INK 0 (Farbe 1) gilt.

Der Text erscheint in gelb, da er durch den PEN (Stift) ausgegeben wurde, dem der Standardwert INK 1 (Farbe 24) zugewiesen ist.

Nummer	*Farbe*	*Nummer*	*Farbe*
0	schwarz	14	pastellblau
1	blau	15	orange
2	hellblau	16	rosa
3	rot	17	pastellmagenta
4	magenta	18	hellgrün
5	hellviolett	19	seegrün
6	hellrot	20	helles blaugrün
7	purpur	21	limonengrün
8	helles magenta	22	pastellgrün
9	grün	23	pastellblaugrün
10	blaugrün	24	hellgelb
11	himmelblau	25	pastellgelb
12	gelb	26	leuchtendweiß
13	weiß		

Tippen Sie PEN 2 ein. Jetzt erscheint der neue Text in hellem blaugrün (Farbe 20). Geben Sie jedoch PEN 3 ein, dann erscheint der Text in rot (Farbe 6). Tippen Sie PEN 4 ein, dann erscheint der Text blau – wie das Papier (PAPER). Dies ist dasselbe, als ob Sie PEN 0 eingetippt hätten, da bei diesem Modus, obwohl 16 Farben zur Verfügung stehen, nur 4 verwendet werden können. Tatsächlich wurde der Stift (PEN) auf Ink 0 gesetzt. In diesem Modus verwirrt der CPC464, wenn er auf einen Wert außerhalb des Bereiches 0 bis 3 gesetzt wird: er hat Ihre Anforderung für INK 4 (was außerhalb des gültigen Bereichs ist) in Ink 0 umgewandelt, was gültig ist. Ebenso würde er INK 5 nach INK 1, INK 6 nach INK 2, INK 7 nach INK 3 und INK 8 nach INK 0 usw. umwandeln.

Wenn Sie immer noch blaue Schrift auf blauem Hintergrund haben, so geben Sie ganz sorgfältig PAPER 1 ein, da Sie ja einen eventuellen Fehler nicht sehen können. Wenn Sie es geschafft haben, werden Sie nun das Wort ‚Ready' in blau auf gelbem Hintergrund sehen. Dies geschieht aus folgendem Grund: Der CPC464 gibt auf dem Bildschirm ein Zeichen aus, indem er ein Viereck

benutzt, das aus der Farbe des Hintergrunds besteht und groß genug für ein Zeichen ist. Dieses wird mit dem momentanen Stift (PEN) ausgegeben. Die gelbe Umrandung von ‚Ready' ist also das Ergebnis davon, daß die Zeichen auf einem neu definierten Hintergrund geschrieben wurden.

Bei Eingabe von CLS wird der gesamte Bildschirm, natürlich mit Ausnahme des Rahmens, auf die neue PAPER-Zuweisung gelb gesetzt.

Nun kommen wir zu den Tricks. Nehmen wir an, wir wollen nicht gelb, sondern weiß. Dies können wir durch die INK-Anweisung erreichen. Geben Sie ein:

```
INK 1,26
```

Alles was gelb war, ist jetzt weiß! Was ist geschehen? Um gelb in weiß zu verwandeln, hatten wir den Farbtopf 1, der mit gelb gefüllt war, zu leeren und mit weiß (Farbe 26) zu füllen. INK 1,26 bedeutet also: Ändere die Tinte (INK) des Topfes 1 auf die Farbe 26. Es bedeutet jedoch nicht, daß die Farbe 1 auf die Farbe 26 wechseln soll.

Sie erinnern sich, daß die Standardfarbe für INK 4 weiß war; obwohl sie im MODE 1 außerhalb des gültigen Bereichs liegt. Nun haben wir auch bei INK 1 weiß. Dies ist alles in Ordnung, denn wir können dieselbe Farbe in sovielen Töpfen haben, wie wir wollen.

Der CPC464 frischt natürlich fortwährend auf (50 mal je Sek.). Versuchen Sie die Farbe noch einmal zu wechseln, z.B. blau in orange. Blau befindet sich im Farbtopf (INK) 0 und orange hat die Farbe 15. Also geben wir ein:

```
INK 0,15
```

Nun verändern wir es wieder auf blau:

```
INK 0,1
```

FARBEN UND MODUS

In den Modi 1 und 2 können Sie jede Zahl im Bereich von 0 bis 15 für PEN und PAPER verwenden, der CPC464 wird diese Werte jedoch in die niedrigst möglichen Werte umformen, d.h. im MODE 2 ist für INK lediglich 0 oder 1 zulässig, egal wofür Sie INK angeben, PEN oder PAPER. Dies gilt jedoch nicht für die INK-Anweisung selbst. Es können alle Farben der 16 Tinten (INK) angegeben werden, obwohl Sie dies erst bei Wechsel des Modus sehen können.

Machen Sie nun einen Warmstart und geben Sie folgendes ein:

```
MODE 0:PEN 13
```

Beachten Sie, daß die Farbe des Textes pastellgrün ist.
Tippen Sie jetzt:

```
MODE 1
```

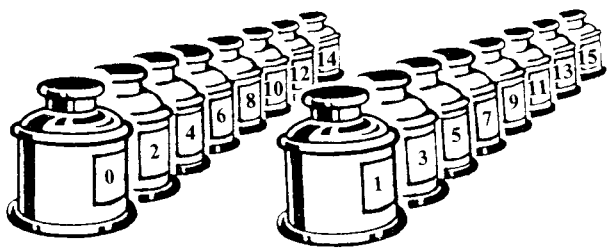
und dann

```
INK 13,16
```

Es verändert sich nichts, weil INK 13 bei Modus 1 außerhalb des gültigen Bereichs liegt. Nun geben Sie wieder ein:

```
MODE 0:PEN 13
```

Der Text erscheint nun in weiß, nicht in pastellgrün. Was wir damit bewiesen haben ist, daß Sie die Farbe in einem Topf wechseln können, auch wenn sie beim augenblicklichen Bildschirmmodus außerhalb des gültigen Bereichs liegt.



FARBIGE ZEICHNUNGEN

Hier sind die Syntaxbeschreibungen unserer alten Freunde, des PLOT- und des DRAW-Befehls:

PLOT

```
PLOT  <x-Koordinate>,<y-Koordinate>[,<ink>]
```

DRAW

```
DRAW  <x-Koordinate>,<y-Koordinate>[,<ink>]
```

INK – (Tinten-) Standardwerte

Tinte (INK)	*Farbe*
0	1
1	24
2	20
3	6
4	26
5	0
6	2
7	8
8	10
9	12
10	14
11	16
12	18
13	22
14	1, 24
15	16, 11

Die <x-Koordinate> und die <y-Koordinate> müssen numerische Ausdrücke sein. Wenn das INK-Argument nicht angegeben ist, so wird die Zeile oder Bildpunkt in der momentanen Farbe des Stifts (PEN) gezeichnet. Dieser hat den Standardwert 1. Jedesmal jedoch, wenn ein Befehl mit INK-Argument ausgeführt wird, wird die Zeile in der neuen INK-Farbe gezeichnet und dem Graphikstift zugewiesen.

Das Wechseln der PEN INK (Tintenfarbe) für den Textstift hat keinen Einfluß auf die INK, mit der Sie in einem PLOT- oder DRAW-Befehl arbeiten. Geben Sie ein:

```
DRAW 640,400
```

Jetzt sollten Sie eine Linie erhalten, die quer über den Bildschirm verläuft und die gleiche Farbe hat, wie der Text. Geben Sie jetzt ein:

```
MOVE 0,0:DRAW 640,400,3
```

Die Linie wird erneut gezeichnet, jedoch in INK 3.

Geben Sie nach einem Warmstart folgendes Programm ein:

```
 10 CLS
 20 r=40:y=200:c=1
 30 FOR x=40 TO 600 STEP 40
 40 FOR i=-r TO r STEP 2
 50 h=SQR(r*r-i*i)
 60 PLOT x-h,i+y,c
 70 DRAW x+h,i+y
 80 NEXT i
 90 c=c+1
100 NEXT x
```

Geben Sie MODE 0 ein und dann RUN. Sie sehen eine Reihe von Kreisen in unterschiedlichen Farben über den Bildschirm verteilt, die vom nächsten teilweise verdeckt werden. Lassen Sie sich nicht durch die Art und Weise verwirren, wie die Kreise gezeichnet wurden, denn es kommt uns auf die Farben an.

Zunächst erinnern wir uns, daß bei MODE 0 alle 16 Farbtöpfe zur Verfügung stehen und mit den Farben, die Sie sehen, gefüllt sind. Als zweites haben Sie die Farbe verändert, indem Sie nach den Koordinaten im PLOT-Befehl die INK-Nummer angegeben haben. Schauen Sie sich jetzt die Auflistung an. In Zeile 60 haben wir eine dritte Variable ‚c', die angibt, welche Tinte (INK), verwendet werden soll. In Zeile 90 sehen wir, daß ‚c' nach jedem Durchlauf der i-Schleife erhöht wird. Die Variable ‚c' hat einen Anfangswert von 1 (siehe Zeile 20). Lassen Sie das Programm in jedem Modus ablaufen, um den Lehrstoff dieses Kapitels zu festigen.

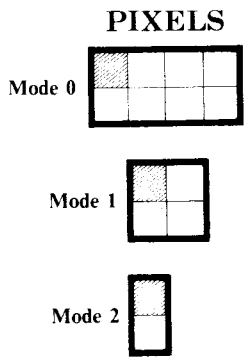
Abschließend lassen Sie das Programm im MODE 0 ablaufen und experimentieren ein wenig, indem Sie ein paar INK-Befehle eingeben, wie beispielsweise INK 6,12 oder INK 2,15.

FENSTER-GESTALTUNG

Wenn Sie die Handhabung von PEN, PAPER und INK einmal begriffen haben, werden Sie in der Lage sein, interessante Effekte zusammenzustellen. Noch effektvollere Bildschirmausgaben sind sogar möglich, wenn Sie einige tiefere Details über die Möglichkeiten des CPC464 erlernt haben und Text mit Graphik verbinden.

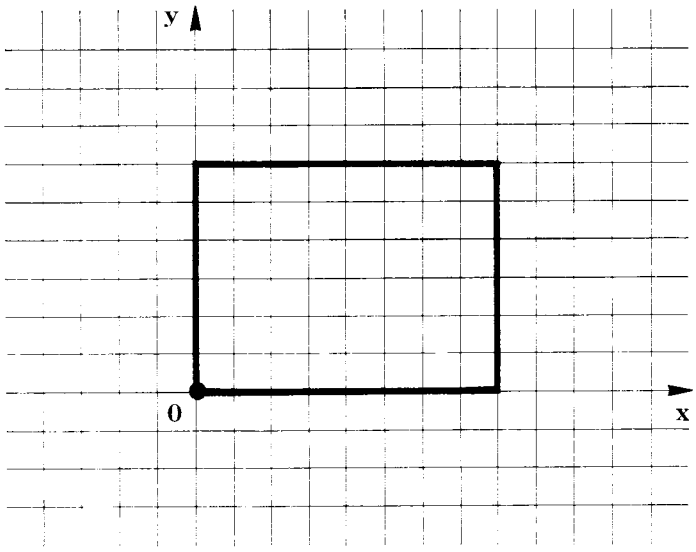
Zunächst wollen wir die ORIGIN-Anweisung behandeln. Diejenigen von uns, die sich in Mathematik gut auskennen, werden wissen, daß der Ausgangspunkt einer Zeichnung der Punkt ist, in dem sich die horizontale und die vertikale Achse schneiden, normalerweise bei x=0, y=0.

Der Bildschirm des CPC464 ist für die Ausgabe von 256000 eigenständigen Bildpunkten ausgelegt. Davon gibt es 640 in der Waagerechten (von 0 bis 639 von links nach rechts), und 400 in der Senkrechten (von 0 bis 399 von unten nach oben). Diese Punkte sind nicht gleichbedeutend mit Pixels, die abhängig vom Bildschirmmodus, aus einer unterschiedlichen Anzahl von Bildpunkten bestehen. Jeder Pixel kann durch die Koordinaten irgendeines Punktes, den er einschließt, bestimmt werden.



Auf jeden dieser Punkte kann individuell Bezug genommen werden, um Graphiken zu erstellen, usw. Die Ausgabe von 0,0 veranlaßt, daß der Punkt 0,0 (und alle anderen zum Pixel gehörenden Punkte) dem Graphikstift (PEN) zugewiesen werden.

Wenn Werte für x oder y ausgegeben werden sollen, die außerhalb des gültigen Bereichs liegen, wird keine Fehlermeldung angezeigt, sondern die PLOT- und DRAW-Befehle werden weiter ausgeführt. Dies wird ermöglicht, weil der CPC464 sich die Position außerhalb des Bildschirms merkt und die PLOT- oder DRAW-Befehle weiter ausführt, obwohl Sie auf dem Bildschirm nichts sehen.



ORIGIN Die ORIGIN-Anweisung erlaubt es Ihnen, die Koordinate 0,0 irgendwo auf dem Bildschirm oder außerhalb zu positionieren. Diese Möglichkeit ist bei der Graphikdarstellung sehr gebräuchlich. Machen Sie einen Warmstart und tippen Sie folgendes Programm ein:

```
 5 CLG
10 PLOT 0,0
20 DRAW 200,200
30 DRAW 400,200
40 DRAW 200,350
50 DRAW -100,350
60 DRAW -100,100
70 DRAW 0,100
80 DRAW 0,0
```

Lassen Sie jetzt mit RUN das Programm ablaufen und Sie werden feststellen, daß ein ungleichmäßiges Polygon ohne jede Bedeutung gezeichnet wurde. Beachten Sie, daß ein Teil davon nicht zu sehen ist, weil er außerhalb der linken Ecke des Bildschirms liegt. Jetzt geben wir ORIGIN 150,0 und RUN ein. Dies ist sehr schlau, stimmt's? Wir haben dem CPC464 damit mitgeteilt, daß der Punkt 150,0 relativ zur linken unteren Ecke mit der Koordinate 0,0 stehen soll und daß jeder PLOT- und DRAW-Befehl sich an diesem neuen Ausgangspunkt orientieren soll. Die linke untere Ecke des Bildschirms wird jetzt durch PLOT −150,0 angesprochen.

Lassen Sie sich nicht verwirren! Stellen Sie sich ein Stückchen Papier auf einem gewaltigen Zeichentisch vor: Sie können alles, was Sie wollen, auf dem gesamten Tisch darstellen, jedoch nur bei dem, was auf dem Papier ist, können Sie die Darstellung sehen. Der Rest sieht so aus, als wäre nichts passiert.

Versuchen Sie einige unterschiedliche Werte für den Ausgangspunkt einzugeben, wie z.B. −50,0 oder 200, −50. Wenn Sie dies gemacht haben, dann geben Sie noch einige zusätzliche Zeilen zum Programm ein:

```
 5 FOR o=0 TO 100 STEP 10
 6 ORIGIN 100+o,o
90 NEXT o
```

und lassen Sie es ablaufen.

Sie werden vorher auf den CLG-Befehl gestoßen sein, der einen ähnlichen Effekt wie CLS hat. Hier ist die Syntax:

CLG

```
CLG [<ink>]
```

CLG löscht das Graphikfenster und positioniert den Graphikcursor auf die Koordinate 0,0. Das wahlweise Argument <ink> füllt das Graphikfenster mit der angegebenen Farbe. Aber Moment mal, werden Sie sagen, was ist ein Graphikfenster? Dies ist zunächst der gesamte Bildschirm, bis Sie dem CPC464 mitteilen, daß Sie es anders wünschen. Sie erreichen dies durch Erweiterung der ORIGIN-Anweisung. Geben Sie ein:

```
ORIGIN 100,0,0,300,200,0
```

Dies veranlaßt den CPC464, den Ausgangspunkt auf die Koordinate 100,0 zu stellen und gleicheitig das Graphikfenster auf alle Pixels von 0-300 in der waagerechten, und von 0-200 in der senkrechten zu begrenzen. Dadurch wird außer dem linken unteren Viereck des Bildschirms alles abgeschnitten.

Es ist jetzt an der Zeit, die Syntax von ORIGIN zu betrachten:

```
ORIGIN <x>,<y>[,<links>,<rechts>,<oben>,
  <unten>]
```

Alle diese Argumente werden als ‚absolute Bildschirm-Koordinaten' bezeichnet und müssen numerische Ausdrücke sein. Eine absolute Bildschirm-Koordinate ist das gleiche, wie die Nummern, die Sie bei PLOT oder DRAW

angeben, außer daß Sie immer relativ zur aktuellen linken unteren Bildschirmecke steht. Es ist unerheblich, ob eine frühere ORIGIN-Anweisung die Koordinate 0,0 positioniert hat; es geschieht immer das gleiche. Mit anderen Worten, ein neues ORIGIN oder Fenster liegt nicht relativ zur zuletzt angegebenen Koordinate 0,0, sondern zur linken unteren Bildschirmecke.

Geben Sie CLS ein und starten das Programm mit RUN. Sie werden feststellen, daß die Graphikausgabe jetzt auf das linke untere Viereck des Bildschirms begrenzt ist. Lassen Sie sich mit LIST das Programm zwei- oder dreimal anzeigen und geben Sie dann CLG ein. Es wird dann lediglich das linke untere Viereck des Bildschirms gelöscht. Geben Sie folgendes ein:

```
CLS: PAPER 3: CLS
```

Sie haben jetzt einen roten Bildschirm; und jetzt geben Sie CLG ein. Unser Grafikfenster ist blau. Nun lassen Sie mit RUN das Programm ablaufen, um den vollständigen Effekt zu erhalten. Probieren Sie für das Fenster einige unterschiedliche Werte, wie z.B.

```
ORIGIN 100,0,301,600,400,201
```

Dadurch wird das Fenster zur rechten oberen Bildschirmecke bewegt. Um die Normalstellung wieder zu erreichen, geben Sie ein:

```
ORIGIN 0,0,0,639,399,0
```

WÖRTER UND BILDER

Es folgt eine Auflistung des nächsten Programms der Datencassette A, SINCOS, anhand dessen wir einige neue Schlüsselwörter behandeln und vertiefen wollen.

```
100 '
110 ' Sinus und Cosinus Graphen
120 '
130 ' von Ian Padwick
140 ' bearbeitet von DA 21/9/84
150 '
160 ' Aufbau des Bildschirms
170 '
180 MODE 1:BORDER 20
190 PRINT CHR$(23);CHR$(0); ' normale Graphik
200 INK 0,23:INK 1,20:INK 2,0:INK 3,26
210 ORIGIN 144,200,108,534,310,56 : CLG 2
220 DEG ' Winkel in sin/cos
230 PEN 2:LOCATE 4,4
240 PRINT"Graphen der Sinus und Cosinus Funktionen"
250 '
260 PLOT 0,-100,3 : DRAW 0,100
```

```
270 PLOT 360,-100 : DRAW 360,100
280 PLOT 0,0      : DRAW 360,0
290 PLOT 90,7     : DRAWR 0,-14
300 PLOT 180,7    : DRAWR 0,-14
310 PLOT 270,7    : DRAWR 0,-14
320 TAG
330 MOVE -20,104  : PRINT "1";
340 MOVE -20,4    : PRINT "0";
350 MOVE -36,-94  : PRINT "-1";
360 MOVE -4,-110  : PRINT "0";
370 MOVE 78,-110  : PRINT "90";
380 MOVE 158,-110 : PRINT "180";
390 MOVE 248,-110 : PRINT "270";
400 MOVE 338,-110 : PRINT "360";
410 MOVE 50,-128  : PRINT "Winkel in Grad";
420 '
430 ' Zeichnen der Sinus-Kurve
440 '
450 FOR x=0 TO 360
460 PLOT x,100*SIN(x),1
470 NEXT
480 MOVE 132,90
490 PRINT"SINUS";
500 '
510 ' Zeichnen der Cosinus-Kurve
520 '
530 FOR x=0 TO 360
540 PLOT x,100*COS(x),3
550 NEXT
560 MOVE 40,-80:PRINT"COSINUS";
570 '
580 ' Aufraeumen
590 '
600 TAGOFF : LOCATE 1,1
610 WHILE INKEY$="" : WEND
620 END
```

Nachdem das Programm gelaufen ist, sehen Sie sich das Listing sorgfältig an. Zunächst die Zeile 180. Der Bildschirmrand wird auf Farbe 20 gesetzt. In Zeile 210 finden wir eine ORIGIN-Anweisung, die das Graphikfenster in die Mitte des Bildschirms setzt. Der neue Ausgangspunkt ist 144,200 und das Fenster ist auf alle Punkte von 108 – 534 in der Waagrechten und von 56 – 310 in der Senkrechten begrenzt.

Der Befehl DEG in Zeile 220 sagt dem CPC464, daß er alle Berechnungen von Sinus und Cosinus in Winkelmaßgraden anstatt im Bogenmaß durchführen soll. Die Zeilen 260 – 310 zeichnen die Achsen für das Diagramm.

Das Schlüsselwort in Zeile 320, TAG, ermöglicht Ihnen einige clevere Dinge in Verbindung mit dem ausgegebenen Text. Wenn Sie ein gewöhnliches PRINT angeben, werden die Zeichen einzelnen ‚Zellen' zugewiesen, in die der Bildschirm aufgeteilt ist und auf die man mit dem LOCATE-Befehl zugreifen kann. Der Textcursor zeigt auf die Zelle, ab der das nächste Zeichen ausgegeben (PRINT) werden soll.

TAG

TAG ist die Abkürzung für ‚Text At Graphics'-Cursor (Texte in Zeichnungen) und bestimmt, daß, wenn er gesetzt ist, jede Textausgabe über den Graphikcursor anstatt über den Textcursor erfolgt. Da wir wissen, daß der Graphikcursor irgendwo auf dem Bildschirm oder außerhalb positioniert sein kann, sind wir bei der Anwendung des PRINT-Befehls äußerst flexibel. In dieser Übung verwenden wir TAG, um die Achsen und Kurven des Diagramms zu beschriften.

Der Grund, daß wir einen Spezialbefehl dafür haben, ist der, daß der CPC464 beim Zeichnen eines Zeichens auf dem Bildschirm viel langsamer arbeitet, wenn es sich über mehrere Zeichenstellen erstreckt.

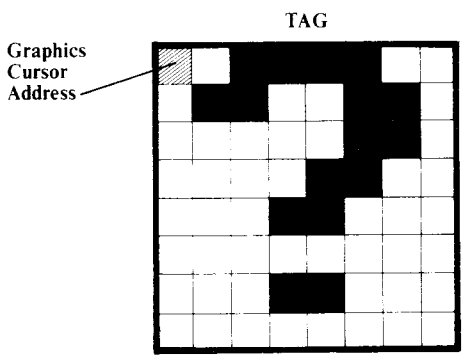
Der Graphikcursor wird durch die Befehle MOVE oder MOVER bewegt. Geben Sie LIST 320 – 410 ein; dann sehen Sie, wie die PRINT-Befehle arbeiten. Beachten Sie, daß der Strichpunkt hinter dem Anführungszeichen bewirkt, daß das Pfeil-Paar unterdrückt wird, welches ansonsten ausgegeben werden würde.

Geben Sie folgendes ein:

```
CLS:TAG:MOVE 50,50:PRINT"HALLO SIE DA"
```

und Sie werden die Pfeile sehen. Wenn wir TAG nicht anwenden würden, würden diese Zeichen den Cursor auf den Beginn der nächsten Zeile stellen.

Für jedes Zeichen nimmt der Graphikcursor die linke obere Ecke einer jeden 8 x 8-Zeichenzelle und wird für das nächste Zeichen um 8 Pixel nach rechts bewegt. Beachten Sie, daß, wenn Sie nun PRINT„LOLLO" eingeben, die Zeichen wieder beim Textcursor ausgegeben werden. TAG wird abgeschaltet, wenn ENTER eingegeben wurde und zwar weil die vorherige Zeile als eigene Zeile eingegeben wurde. Geben Sie jedoch TAG:PRINT„LOLLO" ein, dann werden die Zeichen wieder beim Graphicursor ausgegeben.



Beachten Sie auch, daß die Farbe für den TAG-Text die Graphikfarbe und nicht die Farbe für den Textstift (PEN) ist. Um die Farbe zu wechseln, müssen Sie das dritte Argument von PLOT oder DRAW angeben, da es keinen Befehl ähnlich dem PEN gibt, um den Graphikstift direkt zu wechseln. In gleicher Weise wird die Farbe für den Hintergrund der Zeichen aus dem Graphikfenster herangezogen und nicht von PAPER.

Die Zeilen 450 – 560 führen die Berechnungen und die Darstellung des Diagramms durch (mit denen Sie vielleicht vertraut sind oder auch nicht). Im Kapitel 6 werden Sie über die Anwendung von SIN und COS vollständig unterrichtet und es wird dort auf dieses Programm verwiesen. Zeile 600 beinhaltet das Schlüsselwort TAGOFF, was genau das bewirkt, was Sie sich jetzt darunter vorstellen – es stellt die Ausgabe des PRINT-Befehls wieder auf den Textcursor um. Beachten Sie, daß LOCATE den Textcursor auf dem Bildschirm auch dann bewegt, wenn TAG angeschaltet ist, obwohl nichts passiert, bis TAGOFF erkannt wird.

TAGOFF

KREISDIAGRAMM

Das nächste Programm auf der Datencassette A ist PIECHART. Es ist ein weiteres Programmbeispiel für die Benutzung von Text und Graphik auf dem gleichen Bildschrim.

Kreisdiagramme oder auch Tortengraphiken sind dem Verkaufsleiter, Marketing-Manager und Soziologen mindestens so bekannt wie Balken-Diagramme. Der Grund dafür liegt darin, daß Prozentanteile gezeigt werden können. Dieses Programm verlangt von Ihnen die Eingabe von Zahlen, die Sie dargestellt haben wollen – es führt für Sie sogar die Berechnungen der Prozentanteile durch. Probieren Sie es aus, bevor Sie die folgende Auflistung durcharbeiten.

```
100 ' Torten-Graphik
110 '
120 ' DA 30/9/84
130 '
140 ' Bildschirm
150 '
160 CLEAR
170 DIM name$(9),amount(9) ' veraenderbar
180 MODE 2   : BORDER 19
190 INK 0,23 : INK 1,0
200 PAPER 0  : PEN 1
210 PRINT CHR$(23);CHR$(0);
220 '
230 GOSUB 500 ' Vorspann
240 '
250 q$="J"
260 WHILE UPPER$(q$)="J"
270 '
280 GOSUB 650 ' Daten holen
290 CLS
```

```
300 GOSUB 930 ' Torte zeichnen
310 '
320 LOCATE 1,19
330 INPUT "Noch einmal? (J/N) ",q$
340 WEND
350 END
360 '
370 ' SUBROUTINE : Zeichnen eines Kreises
380 '      Mittelpunkt: x,y
390 '      Radius     : r
400 '
410 MOVE x,y+r
420 s=0.2 : 'Genauigkeit bei PLOT
430 FOR i=0 TO 2*PI+s STEP s
440 DRAW x+SIN(i)*r,y+COS(i)*r,1
450 NEXT
460 RETURN
470 '
480 ' SUBROUTINE : Ausgabe der Titelseite
490 '
500 CLS
510 LOCATE 31,2 : PRINT "Torten-Graphik"
520 MOVE 237,360: DRAW 357,360: DRAW 357,390: DRAW 237,390:
DRAW 237,360
530 WINDOW 5,80,5,25
540 PRINT "Dieses Programm zeigt eine einfache Unterroutine
zur Darstellung von Tor"
550 PRINT "tengraphiken aufgrund vorgegebener Informationen.
 Sie brauchen die Daten"
560 PRINT "nur in Form einer  numerischen Liste  einschliess
lich deren  Bezeichnung"
570 PRINT "einzugeben.  Das Programm ist so ausgelegt, dass
es bis zu zehn Eingaben"
580 PRINT "akzeptiert. Sie koennen das Programm aber so abae
ndern, dass es beliebig"
585 PRINT "viele Werte annimmt."
590 LOCATE 26,7:PRINT"Weiter mit SPACE"
600 WHILE INKEY$<>" " : WEND
610 RETURN
620 '
630 ' SUBROUTINE : Hole die 10 Eingaben
640 '
650 CLS
660 PRINT"Geben Sie bitte den Wert und die Bezeichnung jeder
 Eingabe an. Die Eingabe-"
670 PRINT"Routine wird beendet nach 10 Eingaben oder  wenn S
ie einen  negativen  Wert"
675 PRINT"fuer die Groesse eines Tortenstuecks eingeben"
680 PRINT
690 LOCATE 18,4 : PRINT " Wert" : LOCATE 35,4 : PRINT "Bezei
chnung"
700 entry=0:count=0
710 WHILE ( entry>=0 )  AND ( count<10 )
720 LOCATE   9,5+count : PRINT "Teil";count+1
```

```
730 LOCATE  19,5+count : INPUT "",entry : amount(count)=entr
y
740 LOCATE  35,5+count : INPUT "",name$(count)
750 count = count + 1
760 WEND
770 IF entry>=0 THEN count=count+1
780 count = count - 2 ' Anzahl der gueltigen Werte
790 sum=0
800 FOR i=0 TO count
810 sum=sum+amount(i)
820 NEXT i
830 PRINT : PRINT TAB(15);"Danke, druecken Sie jetzt SPACE,
um die Graphik zu sehen"
840 WHILE INKEY$ <> " " : WEND
850 RETURN
860 '
870 ' SUBROUTINE : Zeichne den Kuchen
880 ' Eingangs-Variable : x,y,r - Mittelpunkt und Radius des
 Keises
890 '                     : count - Anzahl Teilstuecke
900 '                     : amount(0) bis amount(count) - Teil
stuecke
910 '                     :
920 '
930 x=320 : y=180 : r=130
940 GOSUB 410 ' Zeichne den Kreis
950 '
960 p = 0 : oldp = 0
970 '
980 FOR i=0 TO count ' Hauptschleife
990 cum=0 ' Null-Zaehler
1000 '
1010 ' Zaehle bis zum aktuellen Graphen
1020 '
1030 FOR j=0 TO i : cum=cum+amount(j) : NEXT
1040 '
1050 oldp = p ' speichere den letzten Graphen
1060 p = ( cum/sum )*2*PI  ' Berechne den naechsten
1070 MOVE x,y ' Mittelpunkt des Kreises
1080 DRAWR SIN(p)*r,COS(p)*r,1 ' Zeichne zum Rand
1090 GOSUB 1160 ' Ausgabe der Bezeichnung
1100 NEXT i
1110 RETURN
1130 '
1140 ' SUBROUTINE : Drucke die Bezeichnung
1150 '
1160 middle=(p+oldp)/2  ' Mitte des Segmentes
1170 outer=r+25         ' ausserhalb des Kreises
1180 MOVE x+SIN(middle)*outer,y+COS(middle)*outer
1190 wid = LEN(name$(i))
1200 IF middle > PI THEN MOVER -8*wid,0 'bei links Bewegung
nach links
1210 TAG : PRINT name$(i); : TAGOFF
1220 RETURN
```

TEST

Wenn Sie SAT3 durcharbeiten, dann halten Sie dieses Buch bereit, damit Sie die Antworten heraussuchen können. Sie sitzen nicht in einer Prüfung – diese Lernzielkontrollen sind lediglich dafür vorgesehen, Ihnen einen Anhaltspunkt zu geben, ob Sie das Selbststudium verstanden haben, bevor Sie zum nächsten Kapitel weitergehen.

Kapitel 4
DU LIEBE ZEIT

SCHLEIFEN IN SCHLEIFEN

In Teil 1 dieses Kurses wurden die Schlüsselwörter FOR, NEXT und STEP eingeführt, um Ihnen zu demonstrieren, wie eine Reihe von gleichartigen Zaunpfählen auf den Bildschirm gezeichnet werden kann. Damit Sie nicht zurückblättern müssen, hier noch einmal diese Routine.

FOR-NEXT

```
650 FOR f=0 TO 620 STEP 20
660 MOVE f,0: DRAW f,60,15
670 NEXT f
```

Geben Sie die Routine ein und lassen Sie sie ablaufen, wenn Sie wollen. Es wurde bereits erklärt, daß der CPC464 verschiedene aufeinanderfolgende Werte für „f" verwendet, die jeweils um den Wert von STEP verändert werden. In unserem Fall beginnt die Schleife bei 0 und wird bei jedem Durchlauf bei NEXT jeweils erhöht, bis der Endwert von 620 erreicht ist. Der erste Wert von „f" ist also 0, der nächste 20, der nächste 40 und so weiter. Es sollte noch erwähnt werden, daß der Endwert der Schleife nicht unbedingt auch mit dem letzten uns zur Verfügung stehenden Wert identisch sein muß. So zum Beispiel, wenn Sie die Zeile 650 ändern in:

```
650 FOR f=0 TO 635 STEP 20
```

Der letzte Werte von „f" ist jetzt auch 620, da der nächste Wert 640 außerhalb des FOR-NEXT Bereiches liegt. Die Schrittgröße kann praktisch jeden Betrag darstellen einschließlich Brüche und negative Zahlen. Z.B.:

```
FOR f=0 TO 10 STEP 0.15
```

ergibt die Werte 0, 0.15, 0.3, 0.45, 0.6 usw. bis 9.9. Die 10 ist nicht der letzte Wert.

```
FOR f=10 TO -7 STEP -2
```

ergibt die Werte 10,8,6,4,2,0,−2,−4 und −6. Wieder ist −7 nicht der letzte Wert.

Im letzten Kapitel hatte das Programm, das eine Reihe von Scheiben auf den Bildschirm zeichnet, zwei FOR-NEXT Schleifen – eine innerhalb der anderen. Die innere Schleife (mit der Variablen i) zeichnete die Scheiben, während die äußere den Mittelpunkt jeder Scheibe positionierte.

FOR-NEXT Schleifen können auf drei verschiedene Arten angewandt werden:

- ☐ Wie in dem Beispiel von vorhin, indem wir bestimmte veränderte Werte für eine Variable haben wollen zur Ausführung von PRINT- oder PLOT-Befehlen bzw. zum Durchlauf bestimmter Speicher-Arrays.
- ☐ Wenn wir eine Reihe von Kommandos in einer bestimmten Anzahl wiederholen wollen.
- ☐ Wenn wir eine Verzögerung einbauen möchten, um die Geschwindigkeit, in der etwas abläuft, zu verringern.

Das folgende Programm verdeutlicht alle drei Prinzipien:

```
10 FOR l=1 TO 3

20 FOR n=10 TO 100 STEP 5

30 PLOT 0,n:DRAWR 100,100

40 FOR d=1 to 500:NEXT d

50 NEXT n

60 CLS

70 NEXT l
```

Die innerste Schleife in Zeile 40 ist die Warteschleife, in der der CPC464 immer von 1 bis 500 durchzählt, bevor er weitermacht. Versuchen Sie den Wert der Variablen von 500 auf z.B. 50 oder 5000 zu ändern, um eine Vorstellung dieses Prinzips zu bekommen – anschließend löschen Sie die betreffende Zeile ganz und sehen sich die Maximalgeschwindigkeit an.

Verschachtelung

Die nächste Schleife läuft von Zeile 20 bis 50 und verwendet die Variable „n", um den Graphik-Cursor zu positionieren. Die äußere Schleife von Zeile 10 bis 70 stellt sicher, daß der ganze Vorgang dreimal wiederholt wird. Die

Schleifen sind ineinander sozusagen „verschachtelt", also eine Schleife in die andere gesteckt. Im Prinzip kann jede beliebige Anzahl von Schleifen verschachtelt werden, solange sie sich nicht überlappen. Vertauschen Sie Zeile 50 und 70 und lassen Sie das Programm ablaufen.... Sie werden die Meldung erhalten: Unexpected NEXT in 70. Der CPC464 hat gemerkt, daß sich die Schleifen „l" und „n" überlappen und weigert sich, das Programm auszuführen. Wenn Sie die Variablenbezeichnung beim NEXT-Kommando weglassen, bezieht der CPC464 das NEXT auf das letzte FOR-Kommando. Wenn Ihr Programm aber falsch ist, heißt das nur, daß es vielleicht etwas länger läuft, bevor es zusammenbricht. Geben Sie also immer die Variablenbezeichnung an.

Beachten Sie bitte, daß nach Ausführung der Schleife die benutzte Variable nicht den letzten Wert des angegebenen Bereiches besitzt, sondern den ersten Wert außerhalb des Wertebereiches.

Lassen Sie das Programm nochmals laufen und geben Sie nach dessen Ausführung ein:

```
PRINT n;l
```

Sie werden die Zahlen 105 und 4 auf dem Bildschirm sehen: nicht 100 und 3, wie Sie vielleicht erwartet haben. Der CPC464 erlaubt es Ihnen auch eine Schleife zu verlassen, bevor deren Ende-Bedingung erreicht ist, und selbstverständlich können sämtliche Wertebereiche und Schrittangaben durch Variablen ersetzt werden.

```
10 LET a=7:LET b=73: LET s=1.2
20 FOR n=a TO b STEP s
30 IF n>50 THEN GOTO 60
40 PLOT 5,n
50 NEXT n
60 PRINT n
```

DANN UND WANN

Die FOR-NEXT Schleife ist nicht die einzige Schleife, die Sie beim CPC464 anwenden können. Es gibt auch eine WHILE-WEND Schleife mit folgender Syntax:

WHILE-WEND

```
WHILE <logischer Ausdruck>
```

Diese Schleifen können genauso wie FOR-NEXT Schleifen verschachtelt werden unter Beachtung derselben Regeln; zu jedem WHILE gehört auch ein WEND. Die Ausführung der WHILE-WEND Schleife hängt davon ab, ob eine bestimmte Bedingung wahr oder falsch ist. Der ganze Programmteil

zwischen dem WHILE und dem zugehörigen WEND wird solange ausgeführt, so lange die bei WHILE angegebene Bedingung wahr ist. Zum Beispiel:

```
10 WHILE s<1000
20 n=n+1
30 s=n*n*n
40 PRINT s;
50 WEND
60 PRINT
```

Lassen Sie dieses Programm laufen und Sie sehen die Kubikzahlen der Zahlen von 1 bis 10 auf dem Bildschirm. Das Programm wird durch die bei WHILE gegebene Bedingung nicht über diese Werte hinaus ausgeführt. Als jetzt der CPC464 wieder zu WEND gekommen war, war der Wert von „s" nicht mehr kleiner als 1000 und die Schleife wurde nach Zeile 60 hin verlassen.

Sie werden sich jetzt wohl denken, was denn der Vorteil einer WHILE-WEND Schleife gegenüber folgender Konstruktion ist:

```
10 IF s>=1000 GOTO 60
20 n=n+1
30 s=n*n*n
40 PRINT s;
50 GOTO 10
60 REM Ende der Schleife
```

Kurz gesagt, es ist das gleiche!

Aber der Vorteil der WHILE-WEND Schleife liegt darin, daß Sie nicht die Zeilennummer für Ihre GOTO-Anweisung heraussuchen müssen, und das kann viel Arbeit ersparen, besonders wenn Sie große Programmblöcke innerhalb der Schleife haben, oder viele Zeilen ändern bzw. hinzufügen und Sie vergessen haben, auf welche Zeile Sie zurückspringen müssen.

Sie können eine WHILE-WEND Schleife auch verlassen, bevor die Ende-Bedingung gesetzt wird; den CPC464 stört das nicht!

DIGITAL-UHR

Wenn Sie sich das Listing von *DIGITALC* ansehen, bemerken Sie, daß darin sieben FOR-NEXT Schleifen enthalten sind, sechs davon ineinander ver-

schachtelt. Es gibt auch eine sehr einfache WHILE-WEND Schleife in Zeile 170. Einiges in diesem Programm wird für Sie neu sein. Besonders interessant sind die Zeilen 1000 und 1180, in denen die Erfüllung einer Bedingung mehrere Auswirkungen nach sich zieht.

Programm *DIGITALC*

```
10 INK 1,15:PEN 1:MODE 1
20 FOR n=48 TO 57
30 n$(n-48)=CHR$(n)
40 NEXT n
50 PRINT TAB(14);"DIGITAL-UHR"
60 PRINT :PRINT
70 INPUT "Geben Sie die Start-Stunde ein (ENTER)";h
80 ht=INT(h/10)
90 x=h-ht*10
100 PRINT
110 INPUT "Geben Sie die Start-Minute ein (ENTER)";m
120 y=INT(m/10)
130 z=m-y*10
140 LOCATE 6,12:PRINT"Die Uhr beginnt bei";STR$(ht*10+x);":
";n$(y);n$(z);":00"
150 PEN 2:INK 2,1,24:SPEED INK 12,38
160 LOCATE 2,22:PRINT "  Starten der Uhr mit beliebiger Tas
te "
170 WHILE INKEY$="":WEND
190 MODE 0:PEN 2:INK 2,24
200 PRINT "    DIGITAL-UHR"
210 LOCATE 9,10:PRINT ":  :"
1000 IF ht=0 THEN v=9 ELSE v=2:LOCATE 7,10:PRINT n$(ht)
1010 FOR hu=x TO v
1020 LOCATE 8,10:PRINT n$(hu)
1030 FOR mt=y TO 5
1040 LOCATE 10,10:PRINT n$(mt)
1050 FOR mu=z TO 9
1060 LOCATE 11,10:PRINT n$(mu)
1070 FOR st =0 TO 5
1080 LOCATE 13,10:PRINT n$(st)
1090 FOR su=0 TO 9
1100 LOCATE 14,10:PRINT n$(su)
1110 FOR p=1 TO 929:NEXT p
1120 NEXT su
1130 NEXT st
1140 NEXT mu:z=0
1150 NEXT mt:y=0
1160 NEXT hu:x=0
1170 LOCATE 7,10
1180 IF ht=1 THEN ht=0:x=1:PRINT " ":ELSE ht=1:x=0:PRINT n$
(ht)
1190 GOTO 1000
```

In Kapitel 7 des 1. Teiles haben wir das IF-THEN-ELSE Kommando vorgestellt. Erinnern Sie sich?

IF-THEN-ELSE

```
IF geld>0 THEN PRINT "reich"
  ELSE PRINT "pleite"
```

Mit diesem Kommando wird das Wort „reich" ausgegeben, wenn die Variable größer ist als Null, und „pleite", wenn der Wert von „geld" kleiner oder gleich Null ist. In Kapitel 6 des ersten Teiles haben wir Ihnen bereits gesagt, wie man mehrere Kommandos in einer Zeile unterbringt. Die Kommandos müssen durch einen Doppelpunkt getrennt werden.

Bis jetzt haben Sie sich wahrscheinlich gedacht, daß es egal ist, ob Sie eine Reihe von Kommandos in eine Zeile setzen oder dafür mehrere Zeilen verwenden. Das ist auch richtig, solange keines der Kommandos eine IF-Anweisung ist. Ob in einer Zeile mit mehreren Anweisungen Kommandos ausgeführt werden, die hinter einem IF stehen, hängt davon ab, ob die Bedingung wahr oder falsch ist.

Betrachten Sie

```
IF geld>0 THEN PRINT "reich" ELSE PRINT
"pleite": PRINT "Leihen Sie mir bitte Geld"
```

und

```
IF geld>0 THEN PRINT "reich" ELSE PRINT
"pleite"
PRINT "Leihen Sie mir bitte Geld"
```

Sie denken sich vielleicht, es gibt keinen Unterschied zwischen diesen beiden Zusammenstellungen, aber da täuschen Sie sich. Im unteren Beispiel wird „Leihen Sie mir bitte Geld" gedruckt, egal welchen Wert „geld" hat. Im oberen Fall ist dies nicht so. „Leihen Sie mir bitte Geld" wird nur gedruckt, wenn „geld" *kleiner oder gleich Null* ist, genauso wie „pleite" nur in diesem Fall gedruckt wird. Dies ist natürlich von großer Bedeutung und kann folgendermaßen zusammengefaßt werden: Wenn die IF-Bedingung wahr ist, werden nur die Statements nach THEN bis zum ELSE ausgeführt; wenn die Bedingung nicht wahr ist, werden nur die Statements nach ELSE ausgeführt.

Auf die Zeile 1180 von DIGITALC bezogen sehen wir, daß die Variable „ht" bei einem Wert von 1 geändert wird auf 0, die Variable „x" auf 1 gesetzt wird, und ein Leerzeichen in Spalte 7 in Zeile 10 gedruckt wird. Wenn die Variable „ht" jedoch nicht den Wert 1 hat, wird sie auf 1 gesetzt, „x" wird auf 0 gesetzt und die subskribierte Variable n$(ht) wird in Spalte 7 in Zeile 10 gedruckt.

Wenn Sie richtig angewendet werden, sind Bedingungen mit mehreren Statements sehr nützlich und ersparen Ihnen eine Menge zusätzlicher Programmzeilen.

Ein weiterer Punkt bei DIGITALC ist die Verwendung subskribierter String-Variablen für den Ausdruck der Ziffern auf der Uhr. Warum verwenden wir nicht einfach die Variablen der FOR-Anweisung? Die Antwort darauf lautet, daß der CPC464 beim Ausdruck von Zahlen immer ein Leerzeichen vor und nach der Zahl setzt. Daher können Sie zwei Zahlen nicht direkt hintereinander plazieren, da das letzte Zeichen der ersten Zahl immer von dem führenden Leerzeichen der zweiten Zahl gelöscht wird. Beim Ausdruck einer String-Variablen gibt es jedoch keine begleitenden Leerzeichen. Es ist daher viel einfacher, wenn Sie beim Ausdruck der Zeichen den Umweg über String-Variablen nehmen, als wenn Sie die Zahlen selbst umarbeiten (dies ist auch möglich, siehe Kapitel 7).

Beachten Sie, daß das Programm DIGITALC keine Überprüfung der Eingaben vornimmt. Es akzeptiert also auch unsinnige Zeitangaben wie 10:76 oder 35:15. In einem der folgenden Kapitel werden wir noch Möglichkeiten aufzeigen, um solche Eingaben auf ihre logische Richtigkeit zu überprüfen. Außerdem werden Sie sich noch wundern, warum diese Uhr nicht von der internen Uhr des CPC464 Gebrauch macht, die durch das Schlüsselwort TIME angesprochen werden kann. Bei TIME werden aber einige knifflige Umwandlungen der auftretenden Strings benötigt, und das wird erst in Kapitel 7 erklärt. Dann wäre noch anzumerken, daß Sie bei der Eingabe von PRINT TIME eine Zahl erhalten, die die Zeit in 1/300 Sekunden angibt, welche vergangen ist, seit der Rechner eingeschaltet oder vollständig zurückgesetzt wurde (Zeiten für Cassettenoperationen werden nicht berücksichtigt).

PRINT TIME/300 ergibt dann natürlich die Zeitangabe in Sekunden und PRINT TIME/300/60 eine Zeit in Minuten. Die Verwendung von TIME macht die Uhr genauer, aber finden Sie jetzt zuerst einmal heraus, wie präzise die momentane Version ist.

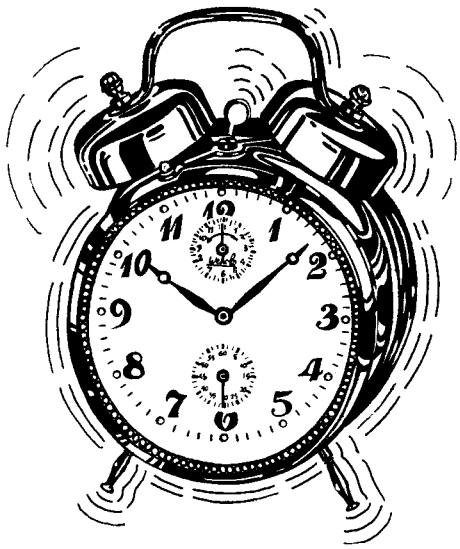
ALARM

Das nächste Programm auf der Datacassette A heißt ALARM. Dabei werden neben ein paar neuen Kommandos viele Kommandos verwandt, die in diesem Kapitel beschrieben wurden. Sie können das Programm tatsächlich dazu benutzen, um sich morgens aufwecken zu lassen oder um Ihrer kleinen Schwester die Zeit zu erklären.

Obwohl dieses Programm einige nicht ganz einfache Teile enthält, sollten Sie es sich dennoch auf dem Bildschirm ausgeben und sich ansehen, wie es zusammengesetzt ist.

TEST

Jetzt haben Sie wieder Gelegenheit, das in diesem Kurs Gelernte zu überprüfen. Grämen Sie sich nicht, wenn Sie einige Passagen des öfteren durchgehen müssen – das ist nun mal so beim Lernen. Also, laden Sie SAT4, und auf geht's.



Kapitel 5
NICHTS ALS ARBEIT

Früher wurden Computer fast ausschließlich von Geschäftsleuten gekauft und mehr oder weniger nur für Dinge wie Lagerverwaltung, Fakturierung oder Lohnabrechnung eingesetzt. Es gab sogar eine Firma, die ihr Personal mit folgender Bildschirmmeldung beim Einschalten des Gerätes daran erinnerte, welchem Zweck ein Computer dienen sollte:

Die Computer der Firma sind nur für Geschäftszwecke zu verwenden!

Absicht war die Leute davon abzuhalten, mit dem Computer während der Arbeitszeit zu spielen. Aber es funktionierte nicht!

Seit Computer auch für den Normalverdiener erschwinglich wurden, stellt sich dieses Problem kaum noch. Nur wenige kommerzielle Computer besitzen die Graphikfähigkeiten des CPC464, ganz abgesehen von den SOUND-Kommandos.

Die Ironie dabei ist, daß der CPC464, der eigentlich zum Spielen und Entwickeln von Spielprogrammen gekauft wurde, auch für die Verwendung kommerzieller Software geeignet ist. Er kann durchaus der optimale Computer für den Besitzer eines kleineren Geschäftes sein.

PROGRAMMIERUNG VON GESCHÄFTSPROGRAMMEN

Nur wenige Anwender kommerzieller Computer kennen sich mit der Programmierung aus (oder wollen sich auch gar nicht auskennen). Für die meisten ist es nur ein Teil Ihres Geschäftes und nicht das Hauptobjekt. Professionelle Maler brauchen auch nicht über die chemische Zusammensetzung der von ihnen verwendeten Farben Bescheid zu wissen, auch wenn sie bei der Erfüllung ihres Jobs schon einmal selbst die Farben mischen! Aus diesem Grund sind kommerzielle Programme sicherlich auch anders aufgebaut als solche, die ein Hobbyprogrammierer für sein eigenes Vergnügen erstellt.

Der erste Unterschied liegt in der Gestaltung der Bildschirm-Anweisungen für den Benutzer. Für den geschäftlichen Anwender müssen sie klar und eindeutig sein, und in jedem INPUT-Kommando sollte auch die Art und der Wertebereich der zu erwartenden Daten enthalten sein.

Der andere Unterschied ist die Notwendigkeit eines „Benutzerhandbuches" für das Programm. Wenn auch die Bildschirminformationen dem Benutzer bei der Eingabe von Daten gewisse Hinweise geben, wird es dieser doch ziemlich langweilig finden, bei jedem Programmlauf lange Erklärungen zu studieren, um herauszufinden, was überhaupt von ihm verlangt wird. Viele Leute ziehen es auch vor, einen kurzen Blick auf die Programmbeschreibung zu werfen, bevor sie sich an das Gerät setzen. Kurz gesagt, wir finden es alle hilfreicher, wenn wir bei unserer Arbeit etwas Anschauungsmaterial in unseren Händen haben.

Der größte Teil vom Rest dieses Kapitels dreht sich um das Programm KALKUL; es wird Ihnen Hinweise darauf geben, wie in etwa professionelle Software ausschaut. Auch wenn wir von Schneider uns darüber im klaren sind, daß der CPC464 nicht zur Steuerung eines Kernkraftwerkes geeignet ist, so kann er Ihnen doch zu Hause, in der Schule oder am Arbeitsplatz auf mannigfaltige Weise behilflich sein.

BILDSCHIRM-GESTALTUNG

Wie man was reinkriegt

INPUT

Hier haben wir gleich noch einen alten Freund aus Teil 1:

```
INPUT [<Ausgabetext>;]
  [<Liste von:<Variable>]
```

Wie Sie sehen, kann man mehr als eine numerische oder String-Variable in einem einzigen INPUT-Kommando verwenden. Die Variablen werden einfach durch Komma getrennt, wie Sie im nachfolgenden Beispiel sehen:

```
INPUT "Geben Sie vier Zahlen ein";a,b,c,d
```

In dieser Form des Kommandos wird ein Fragezeichen und ein Leerzeichen unmittelbar an das Ende des Ausgabestrings gesetzt. Wenn jetzt durch Komma getrennte Eingaben vom Keyboard gemacht werden, versucht der CPC464 diese Eingaben den entsprechenden Variablen in der Liste zuzuordnen. Wenn die Eingaben nicht von der gleichen Art sind, z.B. alphanumerisch statt numerisch, oder wenn unzureichende Eingaben gemacht werden, erhalten Sie die Meldung

```
Redo from start
```

und die Promptmeldung wird wiederholt.

Wenn Sie den Strichpunkt nach dem Prompt-String durch ein Komma ersetzen, wird das Fragezeichen und das nachfolgende Leerzeichen unterdrückt.

Wie man was rauskriegt

PRINT

Ein älterer Freund ist PRINT. Während INPUT der gebräuchlichste Weg für den Benutzer ist mit dem Programm zu sprechen, ist PRINT die einzige Methode für das Programm, Informationen an den Benutzer auszugeben. Abgesehen von den Cursor-Control-Befehlen LOCATE und TAG ist PRINT das einzige Kommando, das die Bildschirmdarstellung beeinflußen kann.

Die normale Syntax lautet:

```
PRINT <Ausgabe-Ausdruck>[<Trennzeichen>
  <Ausgabe-Ausdruck>...][Trennzeichen]
```

Wie Sie von Teil 1 wissen, kann das Trennzeichen entweder ein Komma sein, das den Cursor veranlaßt, auf den Anfang der nächsten Print-Zone zu springen, bevor der nächste <Textausdruck> behandelt wird (wir werden auf das ZONE-Kommando noch später zu sprechen kommen!), oder ein Strichpunkt, bei dem die <Textausdrücke> hintereinander ausgegeben werden. Ein Trennzeichen am Ende des Kommandos verhindert ebenso, daß das nächste PRINT-Kommando auf einer neuen Zeile ausgeführt wird.
Der <Ausgabe-Ausdruck> kann folgende drei Bedeutungen haben:

SPC

TAB

```
<Ausdruck>

SPC (<ganzzahliger Ausdruck>)

TAB (<ganzzahliger Ausdruck>)
```

SPC und TAB sehen vielleicht auf den ersten Blick so aus, als hätten sie die gleiche Funktion, da beide dazu benutzt werden, um zwischen <Ausgabe-Ausdrücken> Leerzeichen zu erzeugen. Der Unterschied liegt darin, daß SPC die Anzahl der durch den ganzzahligen Ausdruck bestimmten Leerzeichen ab der momentanen Position des Cursors einfügt, wohingegen bei TAB die Anzahl der Leerzeichen vom Anfang der Zeile an gezählt werden.

Wenn die Anzahl der bei SPC angegebenen Leerzeichen größer ist als die Zeilenlänge, subtrahiert der CPC464 automatisch die Anzahl der in einer Zeile vorkommenden Druckstellen solange, bis der Wert für die Anzahl der Leerstellen kleiner als die Zeilenlänge ist. SPC benötigt keinen nachfolgenden Strichpunkt, da ein Strichpunkt automatisch gesetzt wird, auch wenn SPC die letzte Angabe in der PRINT-Liste ist.

Wenn die durch TAB angesprochene Position größer als die momentane Cursor-Position ist, werden einfach solange Leerzeichen erzeugt, bis die erwünschte Position erreicht ist. (Wenn sich der Cursor bereits auf der richtigen Position befindet, passiert nichts). Ist die erwünschte Position bereits überschritten, wird die Position einfach auf einer neuen Zeile angesteuert. Ein Strichpunkt als Trennungszeichen wird in der gleichen Weise automatisch gesetzt wie bei SPC.

SPACE$

STRING$

Es gibt auch zwei String-Funktionen, die als <Ausgabe-Ausdruck> in einem PRINT-Kommando benutzt werden können: SPACE$ und STRING$.

SPACE$ ist fast, aber nicht ganz das gleiche, wie SPC. Wenn die damit angegebene Anzahl größer ist als die verbleibenden Zeichen in einer Zeile, wird eine neue Zeile angefangen und mit der Erzeugung von Leerzeichen fortgefahren. Die maximale Anzahl der Leerzeichen beträgt 255.

Strings mit Leerzeichen werden oft dazu benutzt, um Texte zu löschen ohne CLS verwenden zu müsssen. Es ist einfacher, wenn Sie eingeben

```
PRINT SPACE$(40)
```

anstatt

```
PRINT " "
```

Es gibt auch noch den Befehl STRING$, mit dem Sie einen String aus einem beliebigen Zeichen erzeugen können:

```
STRING$ (<ganzzahliger Ausdruck>,<TEXTZEI-
CHEN>)
```

Der <ganzzahlige Ausdruck> ist die Anzahl der Zeichen, aus denen Ihr String bestehen soll. Das <Textzeichen> ist entweder eine Ganzzahl, die den ASCII-Wert Ihres gewünschten Zeichens angibt, oder direkt ein String.

```
PRINT STRING$(10,90)          ergibt   zzzzzzzzzz
PRINT STRING$(16,"=")         ergibt   ================
PRINT STRING$(7,"CPC464")     ergibt   CCCCCCC
```

ZONE

Hier noch, wie vorhin versprochen, das ZONE-Kommando:

```
ZONE <ganzzahliger Ausdruck>
```

Wirklich nicht sehr kompliziert – dadurch wird die Länge der PRINT-Zonen auf dem Bildschirm neu gesetzt mit dem Wert von <ganzzahliger Ausdruck> (der Standardwert ist 13 – erinnern Sie sich?). Dies ist ein sehr nützliches Kommando, wenn Sie Tabellen übersichtlich gestalten wollen.

Die ordentliche Ausgabe

Die oben angeführten neuen Aspekte des PRINT-Kommandos haben alle mit der Positionierung von Wörtern und Zahlen auf dem Bildschirm zu tun. Trotzdem können sie nicht alle Möglichkeiten der Bildschirm-Gestaltung berücksichtigen, ohne zusätzlichen Programmieraufwand. Aus diesem Grund hat das Schneider BASIC eine sehr mächtige Funktion mit der Bezeichnung PRINT USING. Hier ist die Syntax:

```
PRINT USING <String-Ausdruck>;<Ausgabeliste>
```

Die <Ausgabeliste> setzt sich aus <Druck-Zeichen> zusammen mit Trennzeichen, genau so wie beim normalen PRINT-Kommando, aber der String-Ausdruck wird als „Schablonen Format" wie im folgenden Beispiel angewendet:

PRINT USING

```
PRINT USING "###.##";Endergebnis
```

Wenn „Endergebnis" den Wert 87.4473 hat, erhalten Sie auf dem Bildschirm

```
87.45
```

Solange der auszugebende Ausdruck von der richtigen Sorte ist, in diesem Fall also numerisch, druckt der CPC464 den Wert mit dem Komma an der richtigen Stelle aus und rundet dabei auf die vorgegebene Anzahl von Stellen.

Die Zeichen „." und „#" werden als „Felder zur Spezifizierung des Formats" bezeichnet. Es gibt elf davon, aber Sie werden jetzt sicher ungeduldig darauf warten, wieder etwas mit der Tastatur zu machen, so daß wir die Erklärung aller Zeichen lieber auf ein andermal verschieben (in Kapitel 8, Seite 55 des Benutzerhandbuches finden Sie eine Liste dieser Zeichen).

KALKULIEREN MIT KALKUL

KALKUL ist ein Beispiel für ein einfaches Kalkulationsprogramm, das von einem hauptberuflichen Maler oder Dekorateur angewendet werden könnte oder von einem „Self-Made-Man". Es ist ein bischen weiter entwickelt als die übliche Art, die in dieser Branche angewendet wird, bei der einfach ein festgelegter Preis pro Zimmer vergeben wird (natürlich auf Erfahrungswerten basierend) und dann darauf gehofft wird, daß die Kalkulation auch stimmt. Unser Programm setzt alle Materialien und Zeiten in Beziehung zur durchschnittlichen Größe der Zimmer eines Hauses, und zu der Anzahl von Zimmern – sowie zu den Arbeiten außerhalb des Hauses.

Das Programm ist, um im Computer-Jargon zu bleiben, „Menü-gesteuert"; d.h., daß der Anwender zwischen verschiedenen Prozeduren wählen kann, die ihm in einer Auswahl am Bildschirm (Menü) zur Verfügung stehen. Sie können sehen, daß die Menü-Routine in Zeile 640 anfängt, nach der Initialisierung, der Bildschirm- und der Anfangs-Routine. Dies ist die Hauptschleife des Programmes und stellt dem Benutzer fünf Optionen zur Verfügung. Abgesehen von „Programmende" sind diese Optionen entweder Eingabe-Prozeduren, bei denen der Benutzer eine Reihe von Werten eingeben muß, oder Ausgabe-Prozeduren, die Standard-Daten oder Ergebnisse auf dem Bildschirm ausgeben.

Beachten Sie, daß die Standardwerte für die Berechnung der Kalkulation in den Data-Anweisungen ab Zeile 3740 enthalten sind. Diese Werte sind

Vorschläge und keine unveränderlichen Werte. Bitte vergleichen Sie diese Werte mit den aktuellen Materialpreisen und Lohnkosten, bevor Sie sich verpflichten, einen 57 Zimmer Palast in Monte Carlo zu tapezieren!

Hier haben Sie das Programmlisting, das Sie möglichst aufmerksam lesen sollten.

```
2100 '
110 ' ESTIM (Kalkulation)
120 '
130 ' DA 29/9/84  DGC 28/11/84  ESCON 20/2/85
140 '
150 ' Initialisierung
160 '
170 CLEAR
180 tb%=45 ' Spalte fuer Eingaben der Antworten
190 true=-1:false=0
200 beep$=CHR$(7):bs$=CHR$(8):lf$=CHR$(10):cr$=CHR$(13)
210 x$=" "+bs$+CHR$(24)+" " : x1$=" "+CHR$(24)
220 s1$=CR$+lf$+SPACE$(5) : s2$=SPACE$(10)
230 cp.roll=12 'eine Rolle entspricht 12 qm
240 DEF FNcentre$(a$)=SPACE$(40-LEN(a$)/2)+a$
250 DEF FNeven(x,y)=ROUND( x/y ) * y  'Rundung von x zum na
echsten y
260 '
270 ' Initialisieren der Standardwerte
280 '
290 DIM m$(7),m(7),mu$(7),c$(4),c(4),a$(6),a(6),l$(1),l(1)
' optional
300 RESTORE 3790
310 FOR i=0 TO 7 : READ m$(i),m(i),mu$(i) : NEXT
320 FOR i=0 TO 4 : READ c$(i),c(i) : NEXT
330 FOR i=0 TO 6 : READ a$(i),a(i) : NEXT
340 FOR i=0 TO 1 : READ l$(i),l(i) : NEXT
350 READ rph$,rph
360 '
370 ' Bildschirm
380 '
390 MODE 2  : BORDER 0
400 INK 0,0 : INK 1,22
410 PAPER 0 : PEN 1
420 '
430 h$="Kalkulation fuer Renovierung"
440 GOSUB 1860 ' Vorspann
450 '
460 ' Menue
470 '
480 RESTORE 3760
490 FOR i=1 TO 5
500 READ t$
510 LOCATE 28,i*2+5
```

```
520 PRINT i;". ";t$
530 NEXT
540 LOCATE 32,20 : PRINT CHR$(18);
550 INPUT "Waehlen Sie (1-5) : ",i$ : i=VAL(i$)
560 IF i<1 OR i>5 THEN PRINT beep$; : GOTO 540
570 ON i GOSUB 670,2360,2660,1670,630
580 '
590 GOTO 430
600 '
610 ' Programm-Ende
620 '
630 LOCATE 1,22 : END
640 '
650 ' SUBROUTINE : Eingabe der Arbeits-Details
660 '
670 h$="Eingabe Details"
680 GOSUB 1860 ' Vorspann
690 LOCATE 1,5
700 '
710 ' Allgemeines
720 '
730 hd$="Name des Kunden":F%=79-tb%:GOSUB 2140:client$=v$
740 hd$="Adresse (Keine Kommas bitte)":F%=79-tb%:GOSUB 2140
:address$=v$
750 GOSUB 1810 : total.rooms = rooms
760 PRINT
770 '
780 ' Annahme Daten
790 '
800 hd$=x$+"INNEN-ARBEITEN"+x1$:GOSUB 2240:id.flag=v%
810 IF id.flag=false GOTO 1150 ' keine Innen-Dekoration
820 '
830 ' Decken
840 '
850 hd$=s1$+x$+"Decken"+x1$:GOSUB 2240:ce.flag=v%
860 IF ce.flag=false GOTO 930
870 hd$=s2$+"Decken tapeziert+gestrichen":GOSUB 2040:ce.pe=
v%
880 hd$=s2$+"Decken nur gestrichen":GOSUB 2040:ce.e=v%
890 ce.number=ce.pe+ce.e
900 ce.area.pap=ce.pe*a(0) : ce.area.emul=(ce.pe+ce.e)*a(0)

910 ce.cost=(ce.area.pap * ( m(1)/cp.roll ) ) + (ce.area.em
ul * (m(2)/c(2)) )
920 ce.time=(ce.area.pap / l(0)) + (ce.area.emul / l(1))
930 '
940 ' Waende
950 '
960 hd$=s1$+x$+"Waende"+x1$:GOSUB 2240:wa.flag=v%
970 IF wa.flag=false GOTO 1040
980 hd$=s2$+"Zimmer nur tapeziert":GOSUB 2040:wa.p=v%
990 hd$=s2$+"Zimmer nur gestrichen":GOSUB 2040:wa.e=v%
```

```
1000 wa.number=wa.p+wa.e
1010 wa.area.pap=wa.p*a(1) : wa.area.emul=wa.e*a(1)
1020 wa.cost=(wa.area.pap *  (m(0)/cp.roll ) ) + (wa.area.e
mul * (m(2)/c(2)) )
1030 wa.time=(wa.area.pap / l(0)) + (wa.area.emul / l(1))
1040 '
1050 ' Holzarbeiten
1060 '
1070 hd$=s1$+x$+"Holzarbeiten"+x1$:GOSUB 2240:ww.flag=v%
1080 IF ww.flag=false GOTO 1150
1090 hd$=s2$+"Zimmer mit Grund- und Hauptanstrich":GOSUB 20
40:ww.ut=v%
1100 hd$=s2$+"Zimmer mit einfachem Anstrich":GOSUB 2040:ww.
t=v%
1110 ww.number=ww.ut+ww.t
1120 ww.area.under=ww.ut*a(2) : ww.area.top=(ww.ut+ww.t)*a(
2)
1130 ww.cost=(ww.area.under * m(3)/c(0) ) + (ww.area.top *
m(4)/c(1) )
1140 ww.time=(ww.area.under+ww.area.top) / l(1)
1150 '
1160 ' Aussen-Arbeiten
1170 '
1180 PRINT : PRINT
1190 hd$=x$+"AUSSEN-ARBEITEN"+x1$:GOSUB 2240:od.flag=v%
1200 IF od.flag=false GOTO 1520 ' keine Aussen-Arb.
1210 '
1220 ' Tueren und Fenster
1230 '
1240 hd$=s1$+x$+"Tueren und Fenster"+x1$:GOSUB 2240:wd.flag
=v%
1250 IF wd.flag=false GOTO 1340
1260 wd.cost=0:wd.time=0
1270 doors.and.wind=total.rooms * a(3)
1280 hd$=s2$+"Alles nur gestrichen?":GOSUB 2240
1290 IF v% THEN wd.cost=doors.and.wind * m(6)/c(3) : wd.tim
e=doors.and.wind / l(1):GOTO 1340
1300 hd$=s2$+"Alles mit Grund- und Hauptanstrich?":GOSUB 22
40
1310 IF v% THEN wd.cost=doors.and.wind *( m(3)/c(0) +m(5)/c
(1) ):wd.time=doors.and.wind*2 / l(1):GOTO 1340
1320 hd$=s2$+"Alles nur mit Hauptanstrich?":GOSUB 2240
1330 IF v% THEN wd.cost=doors.and.wind * m(5)/c(1) :wd.time
=doors.and.wind / l(1)
1340 '
1350 ' Dachrinnen und Abflussrohre
1360 '
1370 hd$=s1$+x$+"Dachrinnen und Abflussrohre"+x1$:GOSUB 224
0:gd.flag=v%
1380 IF gd.flag=false GOTO 1450
1390 gd.cost=0:gd.time=0
1400 gd.length=total.rooms * a(4)
1410 hd$=s2$+"Alles mit Grund- und Hauptanstrich?" : GOSUB
```

```
2240
1420 IF v% THEN gd.cost=gd.length *( m(3)/c(0) + m(5)/c(1)
):gd.time=gd.length*2 / l(1) :GOTO 1450
1430 hd$=s2$+"Nur mit Hauptanstrich?" : GOSUB 2240
1440 IF v% THEN gd.cost= gd.length * m(5)/c(1) :gd.time=gd.
length / l(1)
1450 '
1460 ' Aussen-Waende
1470 '
1480 hd$=s1$+x$+"Aussen-Waende"+x1$:GOSUB 2240:ow.flag=v%
1490 IF ow.flag=false THEN  1520
1500 ow.cost = total.rooms * a(5) * ( m(7)/c(4) )
1510 ow.time = total.rooms * a(5) / l(1)
1520 '
1530 ' Rundung der Variablen
1540 '
1550 ce.cost=FNeven(ce.cost,0.01) : ce.time=FNeven(ce.time,
0.25)
1560 wa.cost=FNeven(wa.cost,0.01) : wa.time=FNeven(wa.time,
0.25)
1570 ww.cost=FNeven(ww.cost,0.01) : ww.time=FNeven(ww.time,
0.25)
1580 wd.cost=FNeven(wd.cost,0.01) : wd.time=FNeven(wd.time,
0.25)
1590 gd.cost=FNeven(gd.cost,0.01) : gd.time=FNeven(gd.time,
0.25)
1600 ow.cost=FNeven(ow.cost,0.01) : ow.time=FNeven(ow.time,
0.25)
1610 '
1620 PRINT : PRINT : GOSUB 1950 ' Leertaste
1630 RETURN
1640 '
1650 ' SUBROUTINE : Vorbereitung zur Ausgabe der Kalkulatio
n
1660 '
1670 h$="Ausgabe Kalkulation" : GOSUB 1860 ' Vorspann
1680 ' Ausgabe auf Bildschirm
1690 '
1700 str = 0 : GOSUB 3100
1710 '
1720 ' Ausgabe auf Drucker
1730 '
1740 REM:  str = 8 : GOSUB 3100  (ausgeschaltet)
1750 '
1760 RETURN
1770 '
1780 ' SUBROUTINE : Abfrage der Anzahl der Zimmer
1790 '               Beantwortung in 'rooms'
1800 '
1810 hd$="Gesamt-Zahl der Zimmer" : F% = 2
1820 GOSUB 2040 : rooms=v% : RETURN
1830 '
1840 ' SUBROUTINE : Ausgabe des Vorspanns
```

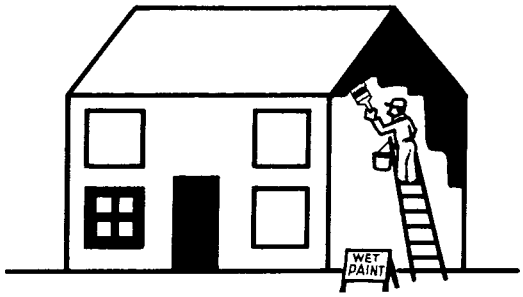
```
1850 '
1860 CLS
1870 temp1=LEN(h$):temp2=(80-temp1)/2
1880 PRINT TAB(temp2);CHR$(24);STRING$(temp1+2,131);CHR$(24
)
1890 PRINT TAB(temp2);CHR$(24);" ";h$;" ";CHR$(24)
1900 PRINT TAB(temp2);CHR$(24);STRING$(temp1+2,140);CHR$(24
)
1910 RETURN
1920 '
1930 ' SUBROUTINE : Warten auf die Leertaste
1940 '
1950 LOCATE 1,25
1960 PRINT FNcentre$("Druecken Sie "+CHR$(24)+" SPACE "+CHR
$(24)+" zum weitermachen");
1970 WHILE INKEY$ > "" : WEND
1980 WHILE INKEY$<>" " : WEND
1990 RETURN
2000 '
2010 ' SUBROUTINE : Eingabe-Routine (Nummer)
2020 '               ergibt die Zahl in V%
2030 '
2040 PRINT hd$;TAB(tb%);
2050 PRINT"[";
2060 PRINT STRING$(F%," ");"] Zahl  ";STRING$(F%+8,bs$);
2070 INPUT "",V%
2080 RETURN
2090 '
2100 ' SUBROUTINE : Eingabe-Routine (string)
2110 '               ergibt string in V$
2120 '               Feld-Laenge in F%
2130 '
2140 PRINT hd$;TAB(tb%);
2150 PRINT "[";
2160 LOCATE tb%+F%,VPOS(#0):PRINT"]";
2170 LOCATE tb%+1 ,VPOS(#0)
2180 INPUT "",V$
2190 RETURN
2200 '
2210 ' SUBROUTINE : Ja/Nein Routine
2220 '               wenn ja v%=wahr sonst v%=falsch
2230 '
2240 PRINT hd$;TAB(tb%);
2250 v%=1
2260 PRINT"[ ]  J - Ja  oder N - Nein"STRING$(35,8);
2270 g$=INKEY$:IF g$="" GOTO 2270
2280 IF UPPER$(g$)="J" THEN v%=true : PRINT"J";STRING$(1,bs
$);:GOTO 2320
2290 IF UPPER$(g$)="N" THEN v%=false: PRINT"N";STRING$(1,bs
$);:GOTO 2320
2300 IF v%<1 AND g$=cr$ THEN PRINT:RETURN
2310 PRINT beep$;
```

```
2320 GOTO 2270
2330 '
2340 ' SUBROUTINE : Ausgabe Standardwerte
2350 '
2360 GOSUB 2610
2370 PRINT : PRINT"MATERIAL";TAB(40);"KOSTEN";TAB(55);"EINH
EIT" : PRINT
2380 FOR i=0 TO 7
2390 PRINT " ";m$(i);
2400 PRINT TAB(40);USING "##.##";m(i);
2410 PRINT TAB(55);mu$(i)
2420 NEXT
2430 PRINT : PRINT"ANSTRICH-DICKE";TAB(40);"QUADRATMETER/LI
TER": PRINT
2440 FOR i=0 TO 4
2450 PRINT " ";c$(i);TAB(40);USING "###.#";c(i)
2460 NEXT
2470 GOSUB 1950 ' Naechster Bildschirm
2480 GOSUB 2610
2490 PRINT : PRINT"FLAECHE";TAB(40);"QUADRATMETER/ZIMMER" :
 PRINT
2500 FOR i=0 TO 5
2510 PRINT " ";a$(i);TAB(40);USING "###.#";a(i)
2520 NEXT
2530 PRINT : PRINT "MANN-STUNDEN";TAB(40);"QUADRATMETER/STU
NDE" : PRINT
2540 FOR i=0 TO 1
2550 PRINT " ";1$(i);TAB(40);USING "###.#";1(i)
2560 NEXT
2570 PRINT : PRINT "PREIS / STUNDE";TAB(40);"DM/STUNDE" : P
RINT
2580 PRINT " ";rph$;TAB(40);USING "##.##";rph
2590 GOSUB 1950 ' Leerzeichen
2600 RETURN
2610 h$="Kosten" : GOSUB 1860 ' Vorspann
2620 RETURN
2630 '
2640 ' SUBROUTINE : Berechnen der Material-Kosten
2650 '
2660 GOSUB 3000
2670 PRINT : PRINT"MATERIAL";TAB(40);"KOSTEN";TAB(50);"NEUE
 KOSTEN";TAB(65);"EINHEIT" : PRINT
2680 FOR i=0 TO 7
2690 PRINT " ";m$(i);TAB(40);USING "##.##";m(i);
2700 PRINT TAB(65);mu$(i)
2710 LOCATE 55,VPOS(#0)-1 : INPUT "",n
2720 IF n>0 THEN m(i)=n
2730 NEXT
2740 PRINT : PRINT"ANSTRICH-DICKE";TAB(40);"QUADRATMETER/LI
TER";TAB(60);"NEUER WERT": PRINT
2750 FOR i=0 TO 4
2760 PRINT " ";c$(i);TAB(40);USING "###.#";c(i);
2770 PRINT TAB(60);:INPUT "",n
```

```
2780 IF n>0 THEN c(i)=n
2790 NEXT
2800 GOSUB 1950 ' Naechster Bildschirm
2810 GOSUB 3000
2820 PRINT : PRINT"FLAECHE";TAB(40);"QM/ZIMMER";TAB(60);"NE
UER WERT" : PRINT
2830 FOR i=0 TO 5
2840 PRINT " ";a$(i);TAB(40);USING "###.#";a(i);
2850 PRINT TAB(60); : INPUT "",n
2860 IF n>0 THEN a(i)=n
2870 NEXT
2880 PRINT : PRINT "ARBEITS-LEISTUNG";TAB(40);"QM/STUNDE";T
AB(60);"NEUER WERT"
2890 PRINT
2900 FOR i=0 TO 1
2910 PRINT " ";l$(i);TAB(40);USING "###.#";l(i);
2920 PRINT TAB(60); : INPUT "",n
2925 IF n>0 THEN l(i)=n
2930 NEXT
2940 PRINT : PRINT "LOHNKOSTEN / STUNDE";TAB(40);"DM/STUNDE
";TAB(60);"NEUE LOHNKOSTEN"
2950 PRINT
2960 PRINT " ";rph$;TAB(40);USING "##.##";rph;
2970 PRINT TAB(60); : INPUT "",n
2975 IF n>0 THEN rph=n
2980 GOSUB 1950 ' Space
2990 RETURN
3000 h$="Berechnung der Kosten" : GOSUB 1860 ' Vorspann
3010 PRINT" Zum Ueberspringen ENTER druecken"
3020 RETURN
3030 '
3040 ' SUBROUTINE : Ausgabe der Kalkulation
3050 '             : str=Ausgabe-Kanal
3060 '             : ta - te = Tabulatoren
3070 '
3080 ' Haupt-Ueberschrift
3090 '
3100 ta=0 : tb=5 : tc=25 : td=45 : te=65 ' Position der Spa
lten
3110 IF (id.flag = false) AND (od.flag = false) THEN PRINT
beep$; : RETURN ' keine Zahlen
3120 PRINT #str,FNcentre$("Kalkulation der Zeit und des Mat
erials fuer die Arbeiten bei:")
3130 PRINT #str
3140 PRINT #str,FNcentre$(address$) : PRINT #str
3150 PRINT #str,FNcentre$("fuer "+client$) : PRINT #str
3160 '
3170 ' Innen-Arbeiten
3180 '
3190 id.cost=0 : id.time=0
3200 IF id.flag=false GOTO 3350
3210 PRINT #str,TAB(ta);"Innen-Arbeiten :" : PRINT #str
```

```
3220 PRINT #str,TAB(tb);"ARBEIT";TAB(tc);"ANZAHL DER ZIMMER
";TAB(td);"MATERIAL-KOSTEN";
3230 PRINT #str,TAB(te+1);"ZEIT"
3240 PRINT #str,TAB(tb);"------";TAB(tc);"----------------
";TAB(td);"---------------";
3250 PRINT #str,TAB(te+1);"----" : PRINT #str
3260 IF ce.flag THEN n=0 : n1=ce.number : c1=ce.cost : t1=c
e.time : GOSUB 3670 : id.cost=id.cost+ce.cost : id.time=id.
time+ce.time
3270 IF wa.flag THEN n=1 : n1=wa.number : c1=wa.cost : t1=w
a.time : GOSUB 3670 : id.cost=id.cost+wa.cost : id.time=id.
time+wa.time
3280 IF ww.flag THEN n=2 : n1=ww.number : c1=ww.cost : t1=w
w.time : GOSUB 3670 : id.cost=id.cost+ww.cost : id.time=id.
time+ww.time
3290 PRINT #str
3300 PRINT #str,TAB(td);"-------"; TAB(te-2);"--------"
3310 PRINT #str,TAB(tc);"Gesamt:";
3320 n=6 : n1=0 : c1=id.cost : t1=id.time : GOSUB 3670
3330 PRINT #str,TAB(td);"========"; TAB(te-2);"========"
3340 PRINT #str : PRINT #str
3350 '
3360 ' Aussen-Arbeiten
3370 '
3380 od.cost=0 : od.time=0
3390 IF od.flag=false GOTO 3520
3400 PRINT #str,TAB(ta);"Aussen-Arbeiten :" : PRINT #str
3410 PRINT #str,TAB(tb);"ARBEIT"; TAB(td);"MATERIAL-KOSTEN"
; TAB(te);"ZEIT"
3420 PRINT #str,TAB(tb);"------"; TAB(td);"---------------"
; TAB(te);"----"
3430 PRINT #str
3440 IF wd.flag THEN n=3 : n1=0 : c1=wd.cost : t1=wd.time :
 GOSUB 3670 : od.cost=od.cost+wd.cost : od.time=od.time+wd.
time
3450 IF gd.flag THEN n=4 : n1=0 : c1=gd.cost : t1=gd.time :
 GOSUB 3670 : od.cost=od.cost+gd.cost : od.time=od.time+gd.
time
3460 IF ow.flag THEN n=5 : n1=0 : c1=ow.cost : t1=ow.time :
 GOSUB 3670 : od.cost=od.cost+ow.cost : od.time=od.time+ow.
time
3470 PRINT #str,TAB(td);"-------"; TAB(te-2);"--------"
3480 PRINT #str,TAB(tc);"Gesamt:";
3490 n=6 : n1=0 : c1=od.cost : t1=od.time : GOSUB 3670
3500 PRINT #str,TAB(td);"======="; TAB(te-2);"========"

3510 PRINT #str : PRINT #str
3520 '
3530 t.cost=id.cost+od.cost : t.time=id.time+od.time
3540 PRINT#str,TAB(tb);"Gesamtkosten";
3550 n=6 : n1=0 : c1=t.cost : t1=t.time :GOSUB 3670
3560 PRINT #str,TAB(tb);"Arbeits-Kosten";TAB(td);USING "###
#.##";t1*rph;
```

```
3570 PRINT #str
3580 PRINT #STR,TAB(td-2);"==========";
3590 PRINT #str,TAB(tb);"Gesamt-Kalkulation";TAB(td);USING
"####.##";t.time*rph+t.cost
3600 PRINT #str : PRINT #str
3610 '
3620 GOSUB 1950 : RETURN
3630 '
3640 ' SUBROUTINE : Ausgabe-Zeile fuer die Verwendung von e
st. Der Drucker
3650 '         benoetigt n fuer a$,n1,c1,t1 : wenm n1=0 dann
 keine Druckausgabe
3660 '
3670 IF a$(n)>"" THEN PRINT #str,TAB(tb);a$(n);
3680 IF n1>0 THEN PRINT #str,TAB(tc);USING "##";n1;
3690 PRINT #str,TAB(td);USING "####.##";c1;
3700 PRINT #str,TAB(te);USING "###.##";t1;
3710 PRINT #str," std."
3720 RETURN
3730 '
3740 ' Data-Liste
3750 '
3760 DATA Eingabe der Arbeits-Details,Angabe der Kosten,Ber
ichtigung der Kosten
3770 DATA Ausgabe Kalkulation fuer den Kunden,Programmende
3780 '
3790 DATA Tapeten,9.00,Rolle (12x1m)
3800 DATA Deckentapete,9.00,Rolle (12x1m)
3810 DATA Emulsions-Farbe,8.72,Liter
3820 DATA Grundierung,8.81,Liter
3830 DATA Anstrichfarbe (innen),9.05,Liter
3840 DATA Anstrichfarbe (aussen),9.05,Liter
3850 DATA Lack,34.20,Liter
3860 DATA Wandfarbe,8.72,Liter
3870 '
3880 DATA Grundierung,46.5,Normal-Anstrich,48,Emulsions-Far
be,45
3890 DATA Lack,41,Wandfarbe,56
3900 '
3910 DATA Decken,9,Innenwaende,34,Innen-Holzarbeiten,3
3920 DATA Aussen-Fenster und -Tueren,3,Dachrinnen etc,0.5
3930 DATA Aussen-Waende,16,"",0
3940 '
3950 DATA tapezieren,6,anstreichen,1.5
3960 '
3970 DATA Lohnkosten pro Stunde,20.50
```



BENUTZERHANDBUCH FÜR KALKUL

Nun, wir haben doch am Anfang dieses Kapitels gesagt, daß das Benutzerhandbuch ein notwendiger Bestandteil eines Geschäfts-Programmes ist. Ohne dieses Thema zu sehr bearbeiten zu wollen, sollen die folgenden Absätze auch einem Computer-Laien helfen, KALKUL so anzuwenden, daß er dabei nicht verzweifelt.

Starten des Programmes

Wenn Sie KALKUL geladen haben und ablaufen lassen, sehen Sie zuerst eine Menüauswahl auf dem Bildschirm, die es Ihnen ermöglicht, die gewünschte Prozedur zu wählen. Nach dem Ende der Prozeduren 1-4 kehren Sie jedesmal in diese Bildschirmmaske zurück und können dann eine neue Funktion auswählen.

Vergessen Sie nicht, daß dieses Programm die Daten nur solange gespeichert hält, solange das System in Betrieb ist. Passen Sie auf, daß Sie nicht Funktion Nummer 5 (Programmende) wählen, solange Sie noch bestimmte Dinge sichern wollen. Nach einem erneuten Programmstart sind nämlich alle Daten gelöscht.

Eingabe der Einzelwerte

Zuerst geben Sie Name und Adresse des Kunden ein, so wie es die Bildschirm-Meldungen verlangen. Dann werden Sie nach der Anzahl der Zimmer in dem zu kalkulierenden Haus gefragt. Dazu gehören auch Küche, Bad, WC und große Gänge; Abstellraum und Speisekammer können Sie weglassen.

Nachdem Sie alle „Ja-Nein" Fragen mit Ja beantwortet haben, geben Sie die Anzahl der Räume ein, in denen eine bestimmte Art von Arbeit ausgeführt werden soll.Sie können entweder Null eingeben oder einfach ENTER drükken, wenn eine bestimmte Arbeit überhaupt nicht anfällt.

Kosten

Option 2 gibt die Preise für Löhne, Material usw. aus, auf denen die Kalkulation basiert.

Berichtigung der Kosten

Diese Option (Nr. 37) bringt der Reihe nach alle Kosten auf den Bildschirm, so daß Sie jeweils einen neuen Wert eingeben können. Wenn Sie einen oder mehrere Werte nicht ändern wollen, springen Sie einfach mit ENTER weiter. Diese Änderungen sind nur wirksam, solange das Programm läuft.

Ausdruck der Kalkulation

Sie können die Kalkulation auf dem Bildschirm ausgeben, wenn Sie Option Nr. 4 auswählen. Dies ist eine Zusammenstellung der kalkulierten Kosten, berechnet aus den auszuführenden Arbeiten und den jeweiligen Preisen, die Sie eingegeben haben. Beachten Sie, daß jeweils nur eine Kalkulation durchgeführt werden kann und daß bei der Eingabe eines neuen Jobs alle vorherigen Daten gelöscht werden.

Es ist jedoch möglich auf Option 3 zurückzugehen (ohne das Programm zu beenden), einige der Kostenfaktoren zu ändern – z.B. eine billigere Tapete – und eine neue Version des gleichen Kalkulationsprogrammes zu erstellen.

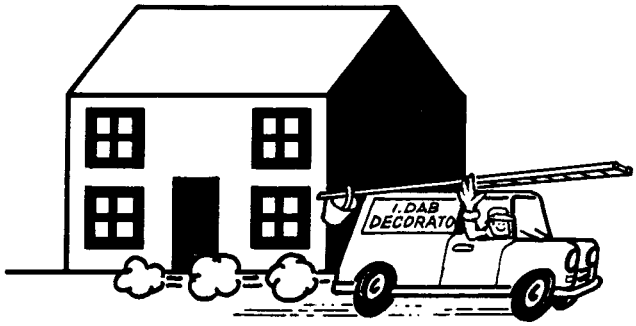
WEITERE VERBESSERUNGEN

Es gibt eigentlich kein Programm, das man nicht irgendwie verbessern könnte. KALKUL macht da keine Ausnahme. Auch professionelle Geschäftsprogramme werden laufend geändert und erweitert, damit Sie den veränderten Anforderungen der Benutzer genügen. Wir haben keine Zweifel, daß Sie dieses Programm auch Ihren Bedürfnissen anpassen wollen.

Hier sind ein paar Anregungen, wie Sie das Programm verbessern könnten:

- ☐ Berücksichtigen Sie verschiedene Haustypen (Bungalow, DHH etc.)
- ☐ Legen Sie die Farbe bzw. Tapetenart für jeden Raum fest
- ☐ Kalkulieren Sie alle Preise für Farben und Tapeten
- ☐ Fügen Sie Farben- und Tapetenmuster hinzu
- ☐ Berücksichtigen Sie die Miete für Leitern und andere Gegenstände

Sie werden sicher noch ein halbes Dutzend anderer Verbesserungen im Kopf haben, wobei wohl eine der wichtigsten eine Routine für die Ausgabe der Kalkulation auf Ihrem Schneider NLQ401 Matrix-Drucker wäre. Ein paar Hinweise dafür geben wir Ihnen in Kapitel 12.



Seien Sie nicht zu ehrgeizig. Computer haben sich schon oft einen schlechten Ruf zugezogen, nur weil ein Programmierer so begeistert war, daß er vor lauter programmieren sein System so kompliziert gemacht hat, daß zum Schluß keiner mehr durchblicken konnte.

TEST

Bis jetzt haben wir nur gearbeitet und noch nicht gespielt. Aus diesem Grunde sind wir so freundlich und quälen Sie nicht auch noch mit einem Test in diesem Kapitel.

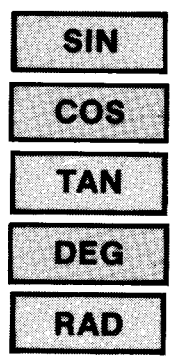
Kapitel 6
ZURÜCK ZUR SCHULE

Die Glücklichen unter Ihnen, die sich schon etwas näher mit Mathematik beschäftigt haben, haben sicher schon von Sinus, Cosinus, Tangens, Radian, Logarithmus und Exponenten gehört. Die anderen haben jetzt das Vergnügen.

Dieses Kapitel soll nicht den Mathematik-Unterricht ersetzen – dafür gibt es eine ganze Reihe sehr guter Mathematik-Bücher – aber wir brauchen ein paar Grundlagen, um die mathematischen Funktionen, die der CPC464 unterstützt, erklären zu können. Der CPC464 ist ein cleveres Kerlchen und er kann einige Rechenoperationen ziemlich rasch durchführen. Obwohl in den meisten Programmen höchstens die vier Grundrechenarten vorkommen, sollten Sie doch über Dinge wie SIN, LOG, oder EXP Kenntnisse besitzen, damit Sie wissen, wann Sie sie in Ihren Programmen verwenden (und wann nicht).

TRIGO

Machen wir also weiter .... Die Schlüsselwörter SIN, COS, TAN und ATN beziehen sich alle auf Winkel und sind trigonometrische Funktionen. In allen Rechenoperationen mit Winkeln nimmt der CPC464 an, daß die Winkel in Radian und nicht in Grad gemessen sind. Ein Radian ist ein spezieller Winkel und entspricht 57,295 Grad; ein Kreis besteht dann aus 2*PI (6,2838) Radianen. Obwohl die Arbeit mit Radianen ziemlich ungewohnt ist, wird sie gerade dadurch oft leichter. Wenn Sie jedoch in Grad rechnen wollen, um nicht alles auf Radian-Maß umrechnen zu müssen, gibt es auf dem CPC464 das Schlüsselwort DEG, das wir bereits in Kapitel 3 kennengelernt haben. Es sagt dem CPC464, daß er in Grad rechnen soll. Analog dazu gibt es das Schlüsselwort RAD, mit dem wieder auf das Bogen-Maß (Radian) umgestellt werden kann.



Geben Sie folgendes Programm ein:

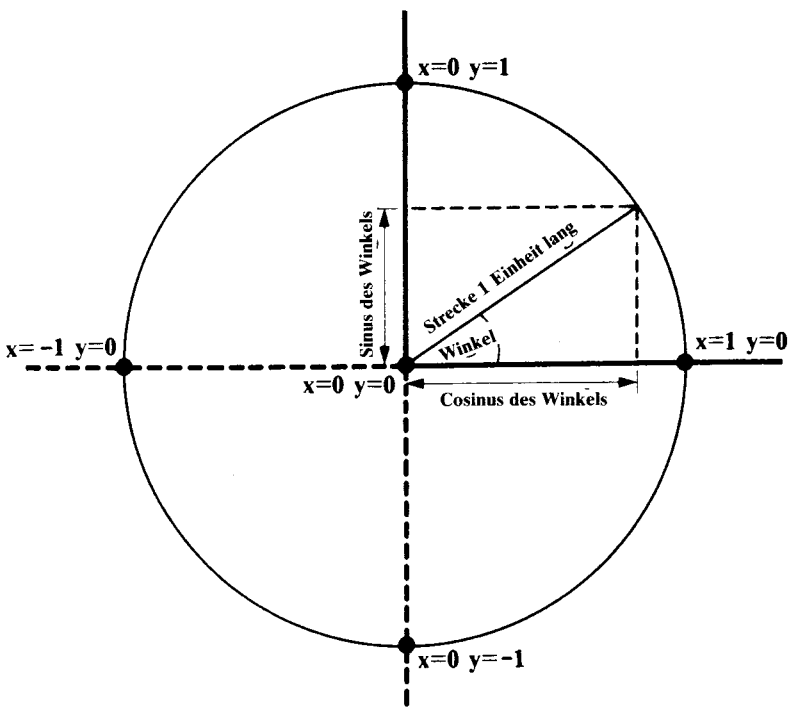
```
 5 DEG
10 ORIGIN 300,200
20 FOR n=1 TO 360
30 x=SIN (n)
40 y=COS (n)
50 PLOT 100*x,100*y
60 NEXT n
```

Benutzen wir jetzt das Programm, um einige Aspekte zu demonstrieren.

Das ORIGIN-Kommando setzt den Punkt 0,0 etwa in die Bildschirmmitte (wie es bereits in Kapitel 3 erklärt wurde). Die Variable „n" wird von 1 beginnend jeweils um 1 erhöht, bis sie den Wert 360 erreicht. Die Sinus- und Cosinus-Werte von „n" (erinnern Sie sich an SINCOS) werden mit 100 multipliziert und dann jeweils auf den Bildschirm gePLOTtet.

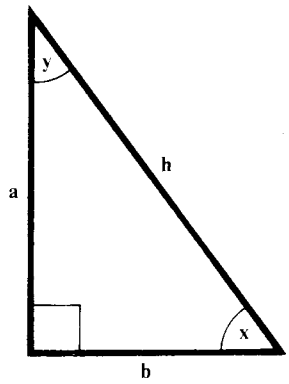
Lassen Sie jetzt das Programm ablaufen, und Sie sehen, wie der Kreis Grad für Grad gePLOTtet wird.

Für diejenigen, die sich immer noch wundern, wie unser Kreis-Programm funktioniert, haben wir das ganze in einem Diagramm dargestellt.



Dreiecke

SINus, COSinus und TANgens werden oft benutzt, um Dreiecke zu definieren. Wenn Sie einige Angaben über das Dreieck haben, können Sie mit Hilfe dieser Funktionen alle anderen Angaben herausfinden. Diese trigonometrischen Funktionen stellen die Beziehungen zwischen den Längen der verschiedenen Seiten dar. In einem rechtwinkligen Dreieck (ein Winkel = 90 Grad), kann der Sinus eines der beiden anderen Winkel herausgefunden werden, indem die Länge der dem Winkel gegenüberliegenden Seite durch die Länge der längsten Seite (Hypotenuse) geteilt wird.



Genauso wie in Mathematik-Büchern schreiben wir auch im Schneider-BASIC statt „der Sinus von Winkel x" einfach nur SIN(x), genauso wie wir statt „Quadrat-Wurzel von x" einfach SQR(x) schreiben.

SQR

Auf das Diagramm bezogen bedeutet das

sin(x)=a/h

und genauso
sin(y)=b/h

Der Cosinus des Winkels kann berechnet werden, indem die Länge der nächstliegenden Seite zum Winkel durch die Länge der Hypotenuse geteilt wird:

cos(x)=b/h und cos(y)=a/h

Beachten Sie, daß sin(x)=cos(y) ist, und umgekehrt. Dies ist eine feste Vorgabe bei rechtwinkligen Dreiecken und macht deren Berechnung relativ einfach.

Der Tangens wird bestimmt, indem die dem Winkel gegenüberliegende Seite durch die nächstliegende Seite zum Winkel geteilt wird:

tan(x)=a/b und tan(y)=b/a

Zur Lösung jedes beliebigen rechtwinkligen Dreiecks genügt es, wenn uns zwei Seiten oder eine Seite und ein Winkel bekannt sind. Wenn a=20 und h=50 können Sie sin(x) und cos(y) folgendermaßen herausfinden:

```
LET a=20: LET h=50: PRINT a/h
```

Sie bekommen als Ergebnis 0.4; 0.4 ist sowohl der Wert von sin(x) als auch von cos(y):

```
SIN(x)=COS(y)=0.4
```

Mit diesen Kenntnissen können wir jetzt relativ einfach berechnen, wie groß die Winkel „x" und „y" in Grad sind. Bei einem wissenschaftlichen Rechner würden Sie jetzt wohl zunächst die INV Taste und dann SIN oder COS drükken, um Ihr Ergebnis zu erhalten. Tatsächlich aber benutzen Sie die inversen Funktionen von SIN und COS. Diese nennt man Arcussinus und Arcuscosinus. Arcustangens gibt es auch.

ATN

Von diesen Arcus-Funktionen besitzt der CPC464 nur die Funktion Arcus-Tangens:

```
ATN (<numerischer Ausdruck>)
```

Die Bestimmung des Arcus-Sinus und des Arcus-Cosinus ist zwar ein wenig knifflig, aber nicht unmöglich, da die Funktionen alle untereinander in Beziehung stehen. Wir konvertieren den SINus- bzw. COSinus-Wert einfach in den entsprechenden TANgens und benutzen dann ATN um den Winkel zu bestimmen. Bevor Sie sich durch dicke Mathematik-Bücher durchschlagen oder stundenlang Ihren Kugelschreiber strapazieren, um eine Lösung zu finden, sagen wir Ihnen gleich, wie es gemacht wird:

QUADRATWURZELN

Der CPC464 hat eine Funktion für die Quadratwurzel. Sie lautet SQR und wird folgendermaßen angewandt: PRINT SQR(64) ergibt als Ergebnis 8, da 8 im Quadrat (in BASIC schreibt man dafür 8↑2) 64 ergibt. Geben Sie folgendes ein:

```
LET tangens=0.4/SQR(1-(0.4↑2))
PRINT ATN(tangens)
```

Sie sollten als Antwort 23.578 erhalten, also ungefähr 23,5 Grad. Machen Sie jetzt das gleiche für die COSinus-Funktion:

```
PRINT ATN(SQR(1-(0.4↑2))/0.4)
```

Sie sollten 66.421 erhalten, also kanpp 66,5 Grad. Wenn wir jetzt beide Ergebnisse zusammenzählen, sollten wir 90 erhalten, da die Winkelsumme eines Dreiecks immer 180 ergibt, und 90 Grad bereits der rechte Winkel beansprucht.

```
23.578+66.421+90=180
```
(gerundet)

Wir kennen jetzt also die Winkel; was wir noch wissen wollen, ist die Länge der Seite „b". Wir wissen z.B., daß sin(y)=b/h. Wir können das leicht auflösen zu b=h*sin(y). Wir können das gleiche auch mit allen anderen Funktionen machen, wie in der folgenden Übersicht gezeigt wird.

sin(y)=b/h wird zu b=h*sin(y)
cos(x)=b/h wird zu b=h*cos(x)
tan(x)=a/b wird zu b=a/tan(x)
tan(y)=b/a wird zu b=a*tan(y)

Jetzt haben wir also vier Möglichkeiten, um die Länge von „b" zu bestimmen und alle sollten das gleiche Ergebnis zeigen. Probieren Sie es selbst aus, indem Sie eingeben:

```
PRINT 50*SIN(66.421)
```
(ergibt 45.825...)

```
PRINT 50*COS(23.578)
```
(ergibt 45.825...)

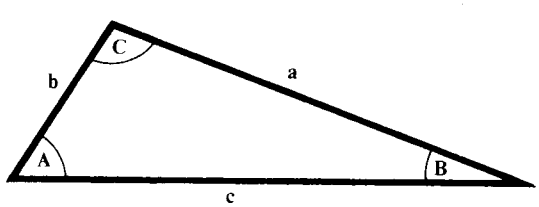
```
PRINT 20/TAN(23.578)
```
(ergibt 45.826...)

```
PRINT 20*TAN(66.421)
```
(ergibt 45.824...)

Wir bekommen tatsächlich immer das gleiche Ergebnis, und dabei auch das richtige! Das war ein Beispiel für die Lösung eines rechtwinkligen Dreiecks. Wie wir bereits sagten, genügen uns zwei Seiten oder ein Winkel (abgesehen vom rechten Winkel) und eine Seite, um den Rest der Figur zu bestimmen.

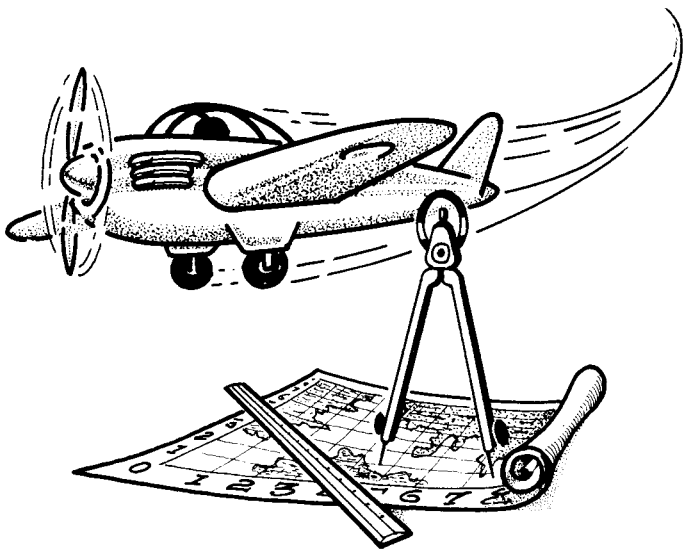
Wenn wir irgendein beliebiges Dreieck berechnen wollen, wird die Aufgabe etwas komplizierter. Aber mit den nötigen Informationen können wir alles herausfinden, was wir brauchen. Die Berechnungen sind natürlich etwas komplexer. Es gibt dabei zwei Grundregeln:

- *Die Sinus-Regel:* für jedes Dreieck
  $a/\sin(A) = b/\sin(B) = c/\sin(C)$
- *Die Cosinus-Regel:* für jedes Dreieck
  $a^2 = b^2 + c^2 - 2bc\cos(A)$



ABFLUG

An diesem Punkt werden Sie sich denken „Dies ist ja alles recht und schön, aber was kann ich jetzt wirklich damit anfangen?". ... Na gut. Stellen Sie sich vor, Sie sind der Pilot eines Flugzeuges und wollen genau Richtung Norden zu einem anderen Flughafen fliegen. An einem windstillen Tag würden Sie einfach Richtung Norden fliegen. Wenn wir aber Wind haben, wie es ja in der Regel der Fall ist, würden Sie jedoch den nächsten Flughafen verfehlen, weil Sie, abhängig von der Windrichtung, vom Kurs abgetrieben würden.



Berechnung Ihrer Richtung

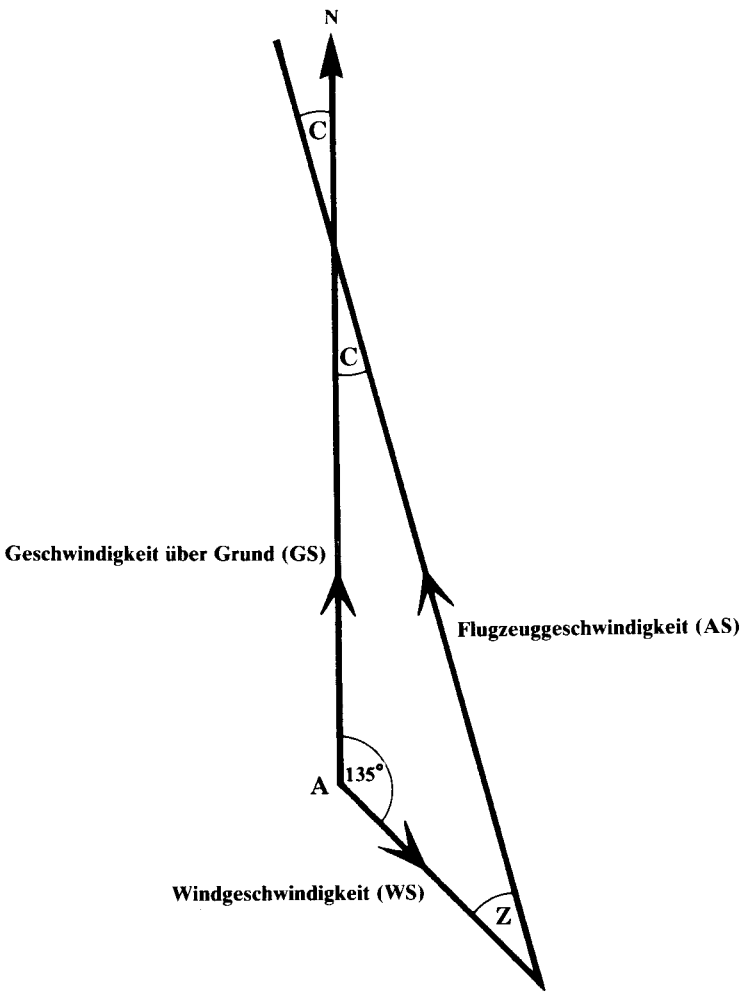
Sie müssen eben Ihren Kurs so berechnen, daß dadurch das Abtreiben durch den Wind ausgeglichen wird. Die Antwort darauf? Dreiecke!

Haben Sie schon mal von Vektoren gehört? Ein Vektor ist eine Linie mit einer Richtung und einer bestimmten Größe. Der Wind kann als Vektor angesehen werden; er hat eine Richtung, z.B. südlich, und eine Größe, z.B. Windstärke 6. Ein Flugzeugkurs ist auch ein Vektor: Richtung, z.B. Kurs 156 Grad und Größe, z.B. 315 Knoten.

Wir erstellen jetzt ein Dreieck mit Vektoren, deren Längen Geschwindigkeiten repräsentieren, nicht Entfernungen.

Beziehen wir uns auf das Diagramm... Der erste Vektor ist dann die Richtung, in die wir fliegen wollen, also genau Norden oder 0 Grad. Wir können jetzt eine gerade Linie von unserem Ursprungspunkt A ziehen, aber wir wissen nicht wie lange unsere Linie ist, da wir nicht wissen, welche Geschwindigkeit wir erreichen. Wenn wir z.B. mit 200 Knoten direkt gegen einen Wind

von 50 Knoten anfliegen, schaffen wir gegenüber dem Boden nur eine Geschwindigkeit von 150 Knoten. Diese Geschwindigkeit im Vergleich zum Boden wird logischerweise „Geschwindigkeit über Grund" genannt, um sie von der Flugzeuggeschwindigkeit, die nur von der Leistung unserer Triebwerke abhängt, zu unterscheiden.



Der zweite Vektor ist der Wind, gezeichnet von Punkt A aus. Wenn wir sagen, daß der Wind von Nord-Westen kommt, können wir genauso gut sagen, er bläst nach Süd-Osten. In unserem Fall interessiert uns mehr, wohin er bläst. Nehmen wir eine Windgeschwindigkeit von 20 Knoten an. Der dritte Vektor zur Vervollständigung unseres Dreiecks repräsentiert die Richtung unseres Flugzeuges; aber die wissen wir bis jetzt noch nicht genau. Alles, was wir davon wissen, ist seine Länge (bzw. Geschwindigkeit), sagen wir mal 200 Knoten.

Wir haben also ein Dreieck und wollen zwei Werte finden: Zuerst unseren Kurs, der durch den Winkel C vorgegeben ist, und die Länge des Vektors der Geschwindigkeit über Grund, so daß wir ausrechnen können, wie lange unsere Reise dauern wird. Wir können einige Werte einsetzen:

```
SIN(C)/20 = SIN(135)/200
```

daraus können wir ableiten:

```
SIN(C) = 20*SIN(135)/200
```

Jetzt geben Sie DEG ein und dann:

```
LET sinc = 20*SIN(135)/200: PRINT sinc
```

Sie sollten als Ergebnis –7.07107E-02 erahlten, was immer das auch bedeutet! Es ist wohl jetzt an der Zeit Ihnen zu zeigen, wie der CPC464 große und kleine Zahlen anzeigt.

Wenn Zahlen sehr groß oder sehr klein werden, ist es sehr umständlich, sie zu schreiben, so daß kurzerhand eine sog. wissenschaftliche Schreibweise benutzt wird. Damit werden Zahlen so angezeigt, als wären sie bestimmte Potenzen von 10. Eine Million (1.000.000) wird dann beispielsweise zu 10↑6 (10*10*10*10*10*10 = 1.000.000), 3000 wird zu 3*10↑3. Eine negative Potenz von 10 bedeutet, daß die Zahl so oft durch 10 geteilt wird, wie im Exponenten angegeben wird. So bedeutet z.B. 10↑-6 gleich 1/10/10/10/10/10/10 oder gleich 0,000001; und 0,000002 wird dann zu 2*10↑–6. Der CPC464 würde dies als 1E6, 3E3 oder 2E–06 anzeigen. Das E bedeutet „multipliziert mit der angegebenen Zehnerpotenz".

Unser Ergebnis von vorhin, 7.07E-02, bedeutet also 7.07*10↑–2, also 7.07*0.01 oder gleich 0.0707; dieser Wert steht bei uns immer noch für „sinc". Jetzt müssen wir also „sinc" in einen Winkel umwandeln. Erinnern Sie sich noch an die magische Formel von vorhin? Geben Sie ein:

```
LET tanc = sinc/SQR(1-sinc↑2):PRINT tanc
```

Der Wert von „tanc" ist 7.08881E↑–2 oder gleich 0.0708881. Wir können jetzt eingeben:

```
PRINT ATN(tanc)
```

und erhalten als Ergebnis 4.05480723. Winkel C ist also etwa 4.05 Grad, so daß unser Kurs 4.05 Grad westlich von Nord sein muß, oder 355.95 Grad in der üblichen Angabe (Kurse werden in Grad von Norden aus im Uhrzeigersinn gemessen).

Flugplan

Der Wind bläst gegen die Bewegungsrichtung des Flugzeuges in einer bestimmten Größe, so daß Sie vermutlich etwas abgetrieben werden. Wieviel?

Wir müssen herausfinden, wie groß unser Vektor für die Geschwindigkeit über Grund ist, also Seite GS unseres Dreiecks. Wir können wieder die Sinus-Regel anwenden:

gs/sin(Z) = 200/sin(135)

Wir wandeln wieder um in

gs = sin(Z)*200/sin(135)

Jetzt hängen wir aber, da wir den Winkel „Z" nicht kennen! Wir können Ihn leicht herausfinden, da uns die anderen beiden bekannt sind, und die Winkelsumme aller drei Winkel zusammen 180 Grad beträgt:

Z = 180–(135+4.05)

also

Z = 40.95

PRINT SIN(40.95)*200/SIN(135) ergibt 185.375162. Unsere Geschwindigkeit über Grund beträgt also etwas mehr als 185 Knoten. (Ein Knoten entspricht einer nautischen Meile pro Stunde oder 1,852 km/h). Wir könnten die Cosinus-Formel benutzen, um die Geschwindigkeit über Grund zu ermitteln:

$$gs = \sqrt{as^2 + ws^2 - 2as.ws.cos(Z)}$$

Oder in BASIC:

```
LET GS=SQR(200↑2+20↑2-(2*200*20*COS(40.95)))
```

ergibt 185.358424, also den gleichen Wert wie vorhin. Auf den ersten Blick erscheint die obige Zeile ziemlich komplex. Sie könnte auch in mehrere verschiedene kleinere Rechenoperationen aufgeteilt werden. Sie müssen sich auf alle Fälle an die Regel erinnern, daß Klammern genauso gesetzt werden wie FOR-NEXT Schleifen. Jede aufgemachte Klammer muß auch wieder geschlossen werden. Die innersten Klammern werden zuerst behandelt, dann der Reihenfolge nach von innen nach außen. Unser Winkel C könnte folgendermaßen berechnet werden:

$$c = \arctan \frac{\left(\frac{ws.\sin(Z)}{as}\right)}{\sqrt{\left[1 - \left(\frac{ws.\sin(Z)}{as}\right)^2\right]}}$$

Oder in BASIC:

```
LET C=ATN((20*SIN(135)/200)/SQR(1-
  (20*SIN(135)/200)↑2)
```

Diese Zeile beinhaltet alle Rechenoperationen, die wir bisher behandelt haben. Machen Sie einen Test, um ihre Richtigkeit zu überprüfen. Sie sollten wieder das gleiche Ergebnis erhalten, 4.05480723.

Wenn Sie jetzt das nächste Programm auf der Datacassette A mit dem Namen FLUGPLAN laden und ablaufen lassen, werden Sie sehen, daß darin alles enthalten ist, was wir bis jetzt behandelt haben. Das Listing des Programms ist nachstehend abgedruckt.

```
100 '
110 ' Flug-Plan
120 '
130 ' von Ian Padwick
140 ' DA 21/9/84  ESCON 20/2/85
150 '
160 ' Bildschirm
170 '
180 MODE 2:BORDER 20
190 INK 0,20:INK 1,0
200 LOCATE 30,3:PRINT CHR$(24) ; STRING$(15,32)     ; CHR$(24)
210 LOCATE 30,4:PRINT CHR$(24) ; "   Flug-Plan    " ; CHR$(24)
220 LOCATE 30,5:PRINT CHR$(24) ; STRING$(15,32)     ; CHR$(24)
230 WINDOW 5,79,8,24
240 '
250 DEG
260 CLS
270 '
280 ' Zufallswerte fuer Geschwindigkeit und Richtung
290 '
300 ws=ROUND(RND*50)
310 wd=ROUND(RND*359)
320 '
330 ' Geschwindigkeit holen
340 '
350 PRINT:PRINT"Wind-Geschwindigkeit:";ws:"Knoten"
360 PRINT:PRINT"Wind-Richtung:";wd;"Grad"
370 PRINT: INPUT"Geben Sie die Geschwindigkeit Ihres Flugzeugs ein:",as
380 IF as<12 THEN PRINT "Zu langsam!" : GOTO 370
390 '
400 ' Berechnungen
410 '
420 IF wd>=180 THEN wd=wd-180 ELSE wd=wd+180
430 sinc=ws*SIN(wd)/as
440 c=ATN(sinc/SQR(1-sinc^2))
450 IF c<0 THEN co=ABS(c) ELSE co=360-c
460 co=ROUND(co,2)
470 IF wd=180 OR wd=0 THEN gs=as+ws*COS(wd):GOTO 510
480 IF wd>180 THEN z=180-((360-wd)+co)ELSE z=180-(wd+c)
490 gs=ABS(SIN(z)*as/SIN(wd))
500 gs=ROUND (gs,1)
510 t=100/(gs/60)
520 t=ROUND (t,1)
```

```
530 '
540 ' Ausgabe des Ergebnisses
550 '
560 PRINT:PRINT"Der Flugkurs betraegt";co;"Grad"
570 PRINT:PRINT"Ihre Geschwindigkeit ueber Grund betraegt";gs;"Knoten"
580 PRINT:PRINT"Voraussichtliche Flugzeit:";t;"Minuten"
590 '
600 ' Wiederholung
610 '
620 PRINT:PRINT:PRINT "Druecken Sie irgendeine Taste....."
630 WHILE INKEY$="":WEND
640 GOTO 260
```

In diesem Listing finden wir wieder ein paar neue Schlüsselwörter. Das Programm generiert die Windgeschwindigkeit mit Zufallszahlen und fragt ab, welche Geschwindigkeit Ihr Flugzeug haben soll. Dann berechnet das Programm den zu fligenden Kurs und gibt Ihnen an, wie lange Sie für einen 100 *Meilen*-Trip benötigen. Warnung...! Geben Sie nicht eine Geschwindigkeit ein, die geringer als die Windgeschwindigkeit ist, da Sie sonst Ihr Ziel nie erreichen!

ABS

Ein neues Schlüsselwort lautet ABS. ABS gibt Ihnen den ABSoluten Wert einer Zahl zurück; das bedeutet, daß negative Zahlen in positive Zahlen umgewandelt werden, aber positive Zahlen nicht verändert werden.

Die einzigen Zeilen, die noch einer Erklärung bedürfen, sind die Zeilen 470 und 480. Zeile 470 behandelt zwei Spezialfälle; und zwar Windrichtung genau Süd oder genau Nord (0 oder 180 Grad). In diesem Fall können wir die Berechnung nicht fortführen, da wir dabei eine Division durch Null erhalten würden: sin(0)=sin(180)=0. Stattdessen addieren bzw. subtrahieren wir einfach die Windgeschwindigkeit sinngemäß. COS(wd) in Zeile 470 kann nur 1 oder –1 ergeben und stellt den elegantesten Weg zur Lösung dieses Problems dar.

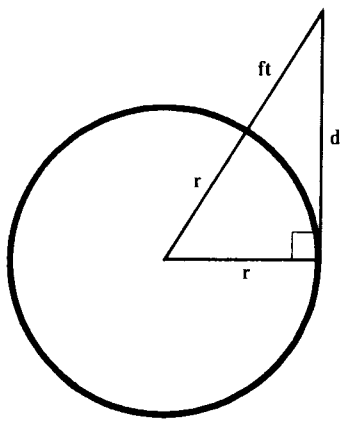
Zeile 480 berechnet den Winkel „Z". Wenn „wd" größer als 180, also ein reflexiver Winkel ist, dann müssen wir diesen Winkel von 360 abziehen, um den komplementären Winkel zu erhalten (Innenwinkel).

Soweit das Auge reicht

Kommen wir jetzt zu einem anderen Problem. Stellen Sie sich vor, daß wir nach dem Abheben wissen wollen, wie weit wir, also die Entfernung bis zum Horizont, sehen können. Wir wollen z.B. wissen, welche Höhe wir brauchen, um in eine Entfernung von 100 Kilometern sehen zu können. Geben Sie das

folgende Programm ein und starten Sie es. Sehen Sie sich das Listing in Verbindung mit dem Diagramm an und versuchen Sie herauszufinden, wie das Programm funktioniert.

```
10 r=6373   REM Durchmesser der Erde in
   Kilometern

20 INPUT "Geben Sie die Flughoehe in
   Metern ein:",met

30 km=met/1000

40 d=(km+r)↑2-r↑2

50 d=SQR(d)

60 d=ROUND (d,2)

70 PRINT:PRINT "Die Entfernung zum
   Horizont betraegt";d;"km"

80 PRINT:PRINT:PRINT

90 GOTO 20
```



Sinus und Cosinus tauchen in den mathematischen Berechnungen vieler Spiele auf. Beim Billard z.B. werden die neuen Richtungen zweier kollidierender Kugeln damit berechnet, oder wenn eine Kugel an die Bande trifft. Diese Berechnungen treten z.B. auch auf bei allen Gegenständen mit einer drehähnlichen Bewegung – z.B. bei der Berechnung der Kolbengeschwindigkeit eines Motors, bei einem schwingenden Pendel oder bei einem Gewicht, das sich an einem Gummiband befindet. Diese Bewegungen können alle durch Sinus- und Cosinusfunktionen beschrieben werden.

Wenn Sie auf Ihrem Computer die Realität so gut wie möglich simulieren wollen (und so wird es auch in den besseren Spielprogrammen gemacht), sind trigonometrische Funktionen eine gute Hilfe.

Natürlich gäbe es noch eine Menge anderer mathematischer Funktionen wie LOG, LOG10, EXP oder SGN, die wir Ihnen an dieser Stelle erklären könnten. Aber diese werden ohnehin im Handbuch ausreichend erklärt.

ZEIT FÜR PRIMZAHLEN

Zum Abschluß sehen Sie hier noch ein Programm, das im Prinzip überhaupt nichts vernünftiges macht. Setzen Sie den CPC464 zurück und geben Sie folgendes Programm ein:

```
10 MODE 0

20 DIM a(2000)

30 a(0)=2: x=0

40 FOR n=3 TO 10001 STEP 2

50 p=0

60 IF a(p)^2)n THEN x=x+1: a(x)=n: PEN 1: GOTO 90

70 IF n/a(p)=INT(n/a(p)) THEN PEN 3: GOTO 90

80 p=p+1: GOTO 60

90 PRINT TAB(8);n

100 NEXT n
```

Damit würde jetzt eine endlose Anzahl einzelner Zahlen ausgedruckt. Primzahlen werden gelb, alle anderen Zahlen rot ausgedruckt. Eine Primzahl ist eine Zahl, die nur durch sich selbst oder durch 1 teilbar ist. Es gibt eine ganze Menge davon, wie Sie sehen, aber es wird immer schwieriger welche zu finden, je größer die Zahlen werden.

Obwohl das Programm alle Werte von „n" ausgibt, werden nur die Primzahlen in den Speicherbereich „a" eingeladen. Das Programm benutzt diese Zahlen, und versucht damit weitere Werte von „n" zu teilen. Wenn eine Zahl durch keine der bisherigen Primzahlen geteilt werden kann, muß es selbst eine Primzahl sein.

Wenn das Programm abgelaufen ist, oder Sie es unterbrochen haben, können Sie die Werte der Tabelle ausgeben, indem Sie folgendes eingeben:

```
FOR n=0 TO 2000: PRINT a(n): NEXT n
```

Sie brauchen dazu nicht einmal eine Zeilennummer zu vergeben. Wieviele Primzahlen gibt es wohl zwischen 1 und 10.000? Bei welchem Wert von „n" ist der Speicherbereich voll? Wenn der Speicherbereich so groß wäre wie der Speicher des CPC464, wie groß wäre dann die größtmögliche Primzahl?

TEST

Das war jetzt ganz schön viel, nicht wahr? Gehen Sie jetzt nochmal die wichtigsten Punkte dieses Kapitels durch, bevor Sie die Lernkontrolle SAT6 starten.

Kapitel 7
MIT WÖRTERN SPIELEN

Der CPC464 hat keine direkt eingebaute Textverarbeitung, aber er kann Text verarbeiten und er hat zahlreiche Schlüsselwörter, mit denen raffinierte Sachen gemacht werden können. Sie sollten sich daran erinnern, daß eine Variable, deren Wert in Anführungszeichen (") definiert wurde, oder alles, was Sie unter PRINT in Anführungszeichen gesetzt haben, "String" genannt wird.

Ein String ist aber nicht unbedingt ein Wort. Es ist irgendeine Folge von Zeichen, die zwischen zwei Anführungszeichen steht.
Wenn eine Variable einen String beinhalten soll, müssen Sie an den Variablen-Namen ein $-Zeichen anhängen, um dem CPC464 mitzuteilen, daß es sich um einen String handelt. Hier sind einige Strings; geben Sie sie auf Ihrem CPC464 ein.

```
LET a$="Schneider"
LET c$="CPC464"
LET r$="45↑%) lko+-CF#!"
```

Innerhalb eines Strings können Sie jedes der 256 Zeichen des CPC464 setzen. Wir können keine mathematischen Operationen mit Strings durchführen außer sie aneinanderzufügen (verketten) oder sie zu vergleichen.

Versuchen Sie:

```
LET z$=a$+" "+c$: PRINT z$
```

Sie sollten jetzt am Bildschirm „Schneider CPC464" sehen. Die Inhalte von a$ und c$, die zu z$ zusammengefügt wurden, sind jetzt *Substrings* von z$, d.h. Strings innerhalb eines größeren Strings.

WIE LANG IST EIN STRING?

Was können wir also tun mit Strings? Nun, wenn Sie die Schlüsselwörter in diesem Kapitel benutzen, können Sie sie zerlegen in kleinere Strings, sie vertauschen, verdrehen, nach bestimmten Zeichen durchsuchen und vieles mehr.

LEN

Das erste, was wir mit einem String machen können, ist seine Länge herauszufinden, d.h. aus wievielen Zeichen er besteht. Wir benutzen dazu das Kommando LEN, das folgende Struktur hat:

```
LEN (<String Ausdruck>)
```

PRINT LEN (a$) wird z.B. 9 ergeben da „Schneider" aus 9 Buchstaben besteht. PRINT LEN (c$) ergibt 7. PRINT LEN z$ ergibt 17. Wir könnten die Länge eines Strings aus verschiedenen Gründen brauchen, z.B. um das Programm zu veranlassen einen String genau in die Mitte des Bildschirmes zu plazieren:

```
10 INPUT "Geben Sie Ihren Namen ein: ", name$
20 IF LEN(name$)> 40 THEN PRINT "Name zu lang":GOTO 10
30 l=LEN(name$)
40 PRINT TAB(20-l/2);name$: PRINT
50 GOTO 10
```

Sie werden sicher schon festgestellt haben, daß Zahlen mit unterschiedlicher Stellenzahl beim Ausdrucken nach links ausgerichtet werden. Wenn wir mehrere unterschiedlich lange Zahlen ausdrucken, wollen wir jedoch in der Regel, daß die Einer, Zehner, Hunderter usw. in einer Linie ausgerichtet sind. Sie bekommen z.B.

```
1
12
97
4032
255
```

wollen aber

```
   1
  12
  97
4032
 255
```

Eine geschickte Art dies zu erreichen wäre die Länge der Zahlen zu ermitteln und dann eine TAB-Position wie in dem Beispiel vorher zu berechnen.

Aber wenn Sie PRINT LEN(n) eingeben, wobei „n" unsere Zahl ist, erhalten Sie die Meldung

```
Type mismatch
```

Das liegt daran, daß Zahlen nicht in der selben Art gespeichert werden wie String-Ausdrücke. Um dies zu vermeiden, müssen wir unsere Zahl in einen String konvertieren. Wir können dies mit dem Schlüsselwort STR$ erreichen:

STR$

```
STR$ (<Numerischer Ausdruck>)
```

Wir können n$ = STR$(n) eingeben und dann PRINT LEN(n$) oder gleich in einem Zug PRINT LEN(STR$(n)). Vergessen Sie nicht, daß der CPC beim Ausdruck einer Zahl immer ein Leerzeichen vor und hinter die Zahl setzt. Das Leerzeichen vor der Zahl stellt immer das Vorzeichen dar und steht in diesem Fall für +. Bei negativen Zahlen steht natürlich ein Minuszeichen davor. Die Verwendung von STR$ unterdrückt automatisch das letzte Leerzeichen, behält aber das Vorzeichen bei. Wenn Sie die Länge einer Zahl ermitteln wollen, bekommen Sie also immer eine Stelle mehr als die Zahl Ziffern hat. Zum Beispiel:

```
PRINT LEN (STR$(464))
```

gibt als Ergebnis 4 statt 3

Das gibt keine Probleme, solange Sie dies mit einberechnen. (Später werden Sie noch lernen, wie man das führende Vorzeichen unterdrückt).

Jetzt können wir also ein kleines Programm schreiben, um unser Vorhaben auszuführen:

```
10 INPUT "Geben Sie eine Zahl ein: ",n
20 l=LEN(STR$(n))
30 PRINT TAB(20-1);n
40 GOTO 10
```

STRINGS ALS ZAHLEN

Wir können jetzt also Zahlen in Strings umwandeln. Aber können wir auch Strings in Zahlen umwandeln? Selbstverständlich!

Die Funktion VAL gibt uns die Möglichkeit den numerischen Wert eines Strings zu finden. VAL kommt vom englischen „value" = „Wert".

VAL

```
VAL (<String Ausdruck>)
```

Benutzen wir dazu unsere Variable a$ („Schneider"). Welchen Wert wird sie haben? Versuchen wir es, indem wir eingeben:

```
PRINT VAL(a$)
```

Sie hat keinen Wert! Das liegt daran, daß der String-Ausdruck ursprünglich aus numerischen Zeichen abgeleitet sein oder am Anfang aus numerischen Zeichen bestehen muß, wenn dem String ein Wert zugewiesen werden soll.

Geben Sie ein:

```
n$="65536"
```

dann

```
PRINT n$:PRINT VAL(n$)
```

Sie werden am Bildschirm 65536 und darunter 65536 nochmal mit einem Leerzeichen davor sehen. VAL(n$) wird zu einer Zahl, mit der mathematische Operationen durchgeführt werden können:

```
PRINT VAL(n$)/256*4
```

ergibt als Ergebnis 1024.

Probieren Sie VAL aus mit folgenden Strings: „CPC464", „464CPC", „–464Sch". Sie sollten als Ergebnis 0, 464 und –464 erhalten.

Wann können wir diese Funktion verwenden? Stellen wir uns vor, wir haben ein Programm, das die Eingabe einer Zahl am Bildschirm verlangt und jemand gibt aus Versehen (oder vielleicht absichtlich) einen Buchstaben ein. Die Meldung

```
?Redo from start
```

würde am Bildschirm erscheinen und die Bildschirmdarstellung beeinträchtigen:

```
10 INPUT "Geben Sie eine Zahl ein (1-9): ",n
20 IF n<1 OR n>9 GOTO 10
30 PRINT n
40 GOTO 10
```

Starten Sie das Programm mit RUN und geben Sie einen Buchstaben ein. Sehen Sie was passiert? Nun, wenn der INPUT-Befehl stattdessen einen String akzeptieren würde, könnte der Fehler mit VAL vermieden werden. Ändern Sie die Variable „n" in den Zeilen 10 und 30 in n$ und geben Sie dann ein:

```
20 IF VAL(n$)<1 OR VAL(n$)>9 GOTO 10
```

Wenn Sie jetzt einen Buchstaben eingeben, wird das Programm nicht in Unordnung geraten.

Stellen Sie sich vor, daß der Zahlenbereich nicht von 1–9 sondern von 0–9 reicht. Die Eingabe eines Buchstabens würde mit VAL(N$) den Wert 0 ergeben, der gültig wäre. Wie können wir feststellen, ob 0 oder ein Buchstabe

eingegeben wurde? Dafür gibt es noch ein Schlüsselwort: ASC. Das ist die Kurzbezeichnung für ASCII, das wiederum die Abkürzung für American Standard Code for Information Interchange ist. Dies ist eines der wenigen Dinge, die fast alle Mikrocomputer gemeinsam haben – einen ASCII-Zeichensatz. Jedes Zeichen des CPC kann durch einen ASCII-Wert erzeugt werden. Die Nummern 32 bis 126 sind genau die gleichen, wie sie jeder Computer mit ASCII-Zeichensatz verwendet.

ASCII UND ASCII

Mit ASC bekommt man den ASCII-Wert des ersten Zeichens einer Zeichenkette nach der Syntax:

```
ASC (<String Ausdruck>)
```

Geben Sie ein:

```
PRINT ASC("0")
```

und Sie erhalten 48, den ASCII-Wert für 0. Geben Sie ein

```
PRINT ASC("Schneider")
```

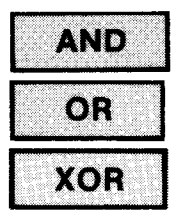
und Sie erhalten 83, den ASCII-Wert für S (großes „S"; das kleine „s" hat den Wert 115). So können Sie die 1 in Zeile 20 auf 0 ändern und hinzufügen:

```
25 IF VAL(n$)=0 AND ASC(n$)<>48 GOTO 10
```

Wenn Sie das Programm jetzt mit RUN starten werden Sie feststellen, daß nur die Zahlen 0–9 angenommen werden. Das ist aber nicht ganz richtig.... Wenn Sie „9Schneider" eingeben, wird dies angenommen! Wie können wir das vermeiden? Wenn wir wollen, daß nur ein einziges Zeichen eingegeben wird, können wir die Zeile 20 folgendermaßen ändern:

```
20 IF LEN(n$)>1 THEN GOTO 10
```

Oder wir könnten INKEY$ statt INPUT verwenden, um zu unserem Zeichen zu gelangen.
Probieren Sie es selbst aus, wie es am besten geht.



Zwei weitere Schlüsselwörter, die wir bereits verwendeten, sind AND und OR. Ihr Gebrauch erklärt sich eigentlich von selbst, aber zu OR gibt es noch eine Anmerkung. Normalerweise könnten wir sagen „wenn a=10 oder b=20 dann ist c=30" mit der Bedeutung, daß c=30 wenn a= 10 oder b=20 ist, aber nicht wenn a=10 und b=20 ist. Solch eine Bedingung wird als exclusive ODER (XOR) – Verknüpfung bezeichnet. XOR erlaubt uns zu sagen: „wenn das eine oder das andere aber nicht beide ....". Die Zeile:

```
IF n>50 XOR n<73 THEN PRINT n
```

gibt den Wert von„n" aus wenn eine und nur eine der Bedingungen wahr ist.

OR erlaubt uns zu sagen „wenn eine der beiden oder alle zwei Bedingungen dann....".
Die Zeile:

```
IF n>50 OR n<73 THEN PRINT n
```

gibt den Wert von „n" aus wenn eine oder beide Bedingungen wahr sind.

Stellen wir uns noch vor, daß wir nur Nummern von 1–5als Eingabe akzeptieren wollen (z.B. bei einer Menüauswahl). Die nächsten vier Zeilen führen diese Abfragen in verschiedenen Arten durch...

```
20 IF VAL(n$)<1 OR VAL(n$)>5 THEN....falsch
20 IF VAL(n$)>0 AND VAL(n$)<6 THEN....richtig
20 IF ASC(n$)<49 OR ASC(n$)>53 THEN....falsch
20 IF ASC(n$)>48 AND ASC(n$)<54 THEN....richtig
```

ZEICHEN AUS ZAHLEN

CHR$

Die entgegengesetzte Funktion von ASC ist CHR$, das Sie vorher schon kennengelernt haben. CHR$ bringt uns den zu einer Zahl zugehörigen ASCII-Wert:

```
CHR$(<Numerischer Ausdruck>)
```

Das nächste Programm stellt uns den vollständigen Zeichensatz dar:

```
10 For n=0 to 255
20 PRINT CHR$(1);CHR$(n):
30 next n
```

Der Ausdruck von CHR$(1) hat keine Wirkung. Dies ist ein Control-Zeichen und ermöglicht uns den Ausdruck der ASCII-Zeichen 0 bis 31. Sie sind auf andere Art nicht darstellbar. Eine Auflistung dieser Control-Zeichen können Sie auf Seite 2 im Kapitel 9 des *Schneider CPC464 Bedienungshandbuches* finden.

LINKS, RECHTS UND MITTE

In Kapitel 4 dieses Buches gibt es ein Programm mit dem Namen DIGITALC, das die Zehner- und Einer-Stellen der Stunden und Minuten separat ausdruckt, so daß es aussieht, als wären es zweistellige Zahlen. Erinnern Sie sich, daß dies normalerweise nicht möglich ist, da die Stelle für das Vorzeichen der zweiten Zahl eine Leerstelle zwischen den beiden Zahlen verursacht. Wir wissen, daß wir STR$ dazu benützen können, das nachfolgende Leerzeichen zu unterdrücken, aber wie unterdrücken wir die Vorzeichen-Stelle?

LEFT$
RIGHT$
MID$

Hier sind drei Kommandos des Schneider BASIC, um solche Dinge zu regeln: LEFT$, MID$ und RIGHT$. Alle drei werden gebraucht, um Teile eines Strings abzuschneiden. LEFT$ macht dies vom linken Ende, RIGHT$ vom rechten Ende und MID$ macht dies überall. Die Syntax für RIGHT$ ist die gleiche wie bei LEFT$:

```
LEFT$(<String Ausdruck>,<ganzzahliger
Ausdruck>)
```

Bei LEFT$ und RIHGT$ gibt der <ganzzahlige Ausdruck> an, wieviele Zeichen vom linken bzw. rechten Ende weg der String-Ausdruck beinhalten soll.

```
MID$(<String Ausdruck>,<ganzz. Ausdruck>
[,<ganzz. Ausdruck>])
```

Bei MID$ definiert der erste ganzzahlige Ausdruck das erste Zeichen des Substrings und der zweite (optionale) ganzzahlige Ausdruck definiert die Anzahl der Zeichen, die der neue Ausdruck ab dem ersten festgelegten Zeichen beinhalten soll. Hier sind einige Beispiele wie diese Kommandos verwendet werden (z$="Schneider CPC464"):

```
PRINT RIGHT$(z$,3)      ergibt 464
  "   RIGHT$(z$,6)      ergibt CPC464
  "   RIGHT$(z$,16)     ergibt Schneider CPC464
  "   LEFT$(z$,3)       ergibt Sch
  "   LEFT$(z$,11)      ergibt Schneider C
  "   LEFT$(z$,16)      ergibt Schneider CPC464
  "   MID$(z$,7)        ergibt der CPC464
  "   MID$(z$,7,3)      ergibt der
  "   LEFT$(z$,11,3)    ergibt CPC
```

Jeder Teil irgendeines Strings kann mit Hilfe dieser Funktionen dargestellt werden. Wozu brauchen wir das ? Das hängt davon ab, was wir tun wollen. Jetzt wollen wir einmal die Vorzeichen-Stelle unterdrücken. Wenn „n" eine einstellige Zahl ist, können Sie eine der beiden Möglichkeiten benutzen:

```
PRINT RIGHT$(STR$(n),1)
```

oder

```
PRINT MID$(STR$(n),2)
```

Wenn „n" eine mehrstellige Zahl ist, müssen Sie LEN wieder verwenden. Versuchen Sie dieses Programm:

```
10 CLS: PAPER 3
20 INPUT "Geben Sie eine Zahl ein: ",n
30 PRINT n
40 PRINT RIGHT$(STR$(n),LEN(STR$(n))-1)
50 GOTO 20
```

Zweck des Hintergrundfarbwechsels ist es, daß Sie die Leerstellen besser sehen können. In Zeile 40 ist STR$(n) der String-Ausdruck und LEN(STR$(n))–1 der gannzahlige Ausdruck. Wir subtrahieren 1, da LEN(STR$(n)) eine Stelle mehr als „n" hat – erinnern Sie sich? Die folgenden Programmzeilen können die gleichzahligen Programmzeilen in DIGITALC ersetzen. Also, laden Sie das Programm und geben Sie ein:

```
180 t=TIME
1070 LOCATE 13,10:Print "0"
1080 WHILE s<60
1090 s=INT((TIME-t)/300)
1100 s$=STR$(s)
1110 LOCATE 14+(s>9),10
1120 PRINT RIGHT$(s$,LEN(s$)-1)
1130 WEND: s=0: t=TIME
```

Diese Zeilen ersetzen die zweite Schleife und machen von der internen Uhr des CPC464 Gebrauch, die durch das Schlüsselwort TIME angesprochen wird.

Sie können sehen, daß Zeile 1120 das führende Leerzeichen unterdrückt und daß s$ vorher als STR$(s) in Zeile 1100 definiert wurde. Die Variable „s" beinhaltet die Differenz zwischen dem aktuellen Stand von TIME und dem Wert von „t", das den Wert von TIME darstellt, als die Uhr in Gang gesetzt wurde (Zeile 180). Die Differenz steigt um 300 Einheiten jede Sekunde, deshalb Zeile 1090.

TIME

Die WHILE-WEND Schleife stellt sicher, daß „t" alle 60 Sekunden auf den neuen Wert von TIME gesetz wird.

Schauen Sie auf Zeile 1110. Hier wird ein interessantes Konzept benutzt. Die Bedingung in der Klammer, s>9, wird entweder als wahr oder als falsch ausgewertet: entweder ist „s" größer als 9 oder nicht. Bei wahr wird der Wert der Bedingung –1, bei falsch wird der Wert 0. Der Wert wird dann dazu benutzt den Cursor auf Spalte 14 zu setzen, wenn „s" eine einstellige Zahl ist, oder auf Spalte 13 wenn es sich um eine zweistellige Zahl handelt. Alle Bedingungen laufen auf Werte von 0 oder –1 hinaus, unabhängig davon ob Sie falsch oder wahr sind.

Um das zu überprüfen geben Sie folgendes ein:

```
v=17
```

dann

```
PRINT v>4;v<20;v<2;v>50;v=17;v*2>50;
SQR(v)>4;v/2<v/3;v+3=20
```

und Sie sollten folgendes Ergebnis bekommen

```
-1 -1 0 0 -1 0 -1 0 -1
```

Diese logischen Werte können für Sie recht nützlich sein, wenn Sie einmal besser programmieren können; aber machen Sie sich jetzt nicht allzuviele Gedanken darüber.

SUCHE NACH WÖRTERN

Hier haben wir gleich eine neue Funktion für Sie: INSTR. Damit können Sie einen String nach Substrings untersuchen. Die ausgegebene Zahl ist die Position des ersten Zeichens des Substrings innerhalb des Hauptstrings.

INSTR

```
STRING INSTR([<ganzzahliger Ausdruck>,]
<String Ausdruck>,<Ausdruck>)
```

Der ganzahlige Ausdruck ist optional und gibt die Position innerhalb des Strings an, ab welcher gesucht werden soll. Wenn er weggelassen wird, wird von vorne begonnen. Der erste Textausdruck gibt an, in welchem String gesucht werden soll, während der zweite bestimmt nach welchem Ausdruck gesucht werden soll. Wenn z.B. z$=„Schneider CPC464" ist, dann ergibt die Anweisung PRINT INSTR(z$, „d") als Ergebnis 7, da „d" das siebente Zeichen des Strings ist.

```
PRINT INSTR(z$,"CPC")
```

ergibt als Ergebnis 11, da das erste Zeichen von CPC das elfte Zeichen des durchsuchten Strings ist.

```
PRINT INSTR(15,z$,"4")
```

Sie erhalten als Ergebnis 16, da die Suche erst nach dem fünfzehnten Zeichen von z$ begonnen wurde.

Dieses Kommando ist sehr nützlich in dem Programm HANGMAN, das Sie in Teil 1 dieses Selbstlern-BASICs bereits kennengelernt haben. Das folgende Programm zeigt Ihnen, wie der Befehl INSTR benützt wurde:

```
10 CLS
20 LOCATE 1,1: INPUT "Geben Sie ein Wort ein: ",w$
30 w$=UPPER$(w$)
40 LOCATE 1,3: "Raten Sie einen Buchstaben: "
50 l$=INKEY$
60 IF l$="" GOTO 50
70 l$=UPPER$(l$)
80 LOCATE 1,3: PRINT SPACE$(16)
90 p=1
```

```
100 p=INSTR(p,w$,l$)
110 IF p=0 GOTO 40
120 LOCATE 10+p,10: PRINT MID$(w$,p,1)
130 p=p+1: GOTO 100
```

Beachten Sie, daß das Programm nicht beendet wird, wenn Sie alle Buchstaben durch haben. Sie können ausarbeiten, wie es gemacht wird, daß das Programm zur richtigen Zeit stoppt. Außerdem haben wir bei diesem Programm noch ein anderes neues Schlüsselwort kennengelernt – UPPER$ –, das alle kleinen Buchstaben (a–z) innerhalb des zugehörigen Strings in die entsprechenden Großbuchstaben umwandelt.

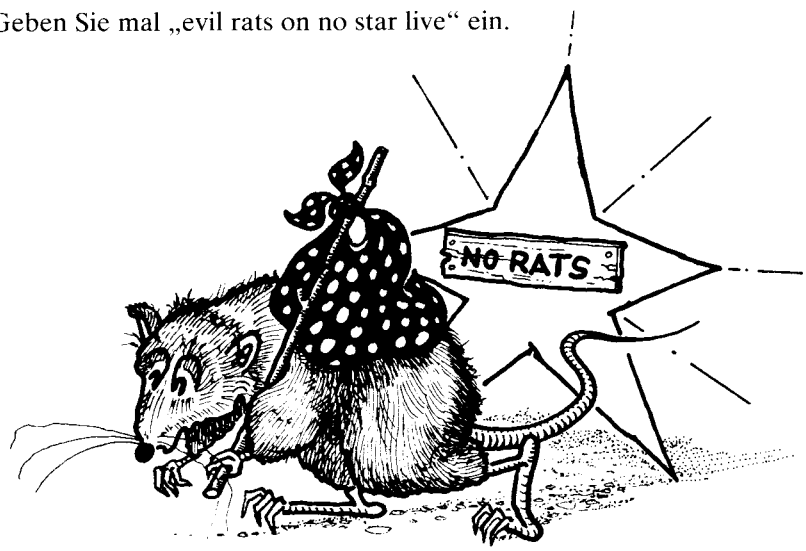
UPPER$

VON HINTEN NACH VORNE

Das abschließende Programm demonstriert eine weitere Art, wie Strings manipuliert werden können. Diesmal wird alles, was Sie eingeben, umgekehrt ausgegeben. Sie sollten herausfinden, wie dies gemacht wird.

```
10 PRINT "Geben Sie einen Text ein: "
20 INPUT w$
30 FOR n=LEN(w$) TO 1 STEP -1
40 b$=b$+MID$(w$,n,1)
50 NEXT
60 PRINT "  ";b$: PRINT: b$=""
70 GOTO 10
```

Geben Sie mal „evil rats on no star live“ ein.



SPIELZEIT

Unser Spiel ist diesmal sehr passend. Es ist eine Hilfe bei Wortspielen in manchen Zeitungen, bei denen Sie möglichst verschiedene Wörter aus den Buchstaben eines langen Wortes bilden müssen. Laden Sie also WORD-PUZL und vergnügen Sie sich ein wenig.

TEST

Starten Sie jetzt SAT7. Man weiß nie, aber vielleicht schreiben Sie selbst einmal solche Rätsel-Spiele für eine Zeitung.

Kapitel 8
BEWEGTE BILDER

Das Prinzip der „bewegten Bilder" ist eigentlich schon lange bekannt. Man plaziert ein Bild, ersetzt es dann durch ein anderes Bild mit kleinen Differenzen, ersetzt dieses dann wieder durch ein neues leicht verändertes Bild usw. Wenn man diesen Vorgang schnell genug und mit konstanten Zeitabständen ablaufen läßt, entsteht der Eindruck einer Bewegung.

BLINKENDE BILDSCHIRMBEREICHE

Eine der einfachsten Methoden, um auf eine Mitteilung am Bildschirm oder auf eine bestimmte Graphik aufmerksam zu machen, ist die Verwendung des INK-Kommandos, das zwei verschiedene Farben benutzt. Das Kommando hat folgende Syntax:

```
INK <Farbstift>,<Farbe>[,<Farbe>]
```

Wenn die zweite <Farbe> mitbenutzt wird, wechselt der CPC464 seine INK immer zwischen zwei verschiedenen Farben.

Geben Sie folgendes ein, nachdem Sie den Rechner zurückgesetzt haben:

```
INK 1,24,1
```

Sie sehen jetzt einen aufblinkenden Bildschirm. Dies liegt daran, daß die zweite angegebene Farbe die gleiche ist wie in der INK des Bildschirmhintergrundes (PAPER). Es ist nicht so, daß die Schrift die Hälfte der Zeit weggelassen wird, sondern vielmehr so, daß Sie die Hälfte der Zeit mit blauem Farbstift (PEN) auf blauem Hintergrund (PAPER) schreiben! Jede beliebige Farb-Kombination kann somit angegeben werden – probieren Sie es selbst aus mit Farben Ihrer Wahl und sehen Sie was passiert.

Die Geschwindigkeit, mit der die Farben wechseln, kann mit einem weiteren Schlüsselwort verändert werden: SPEED INK. Hier ist die Syntax des Kommandos:

SPEED INK

```
SPEED INK <ganzzahliger Ausdruck>,
  <ganzzahliger Ausdruck>
```

Der erste Ausdruck gibt an, wie lange die erste Farbe auf dem Bildschirm erscheint, der zweite Ausdruck die Dauer der zweiten Farbe. Die Zeiten zwischen den Farbwechseln werden in Einheiten von 0,02 Sekunden gemessen.

Hier ist ein kurzes Programm, um Ihnen den Gebrauch des INK-Kommandos mit blinkenden Farben zu verdeutlichen:

```
 10 REM Blinkende Nachricht Programm
 20 CLS
 30 MODE 0
 40 INK 1,24,1
 50 FOR a=50 TO 1 STEP -1
 60 LOCATE 3,12
 70 SPEED INK a,a
 80 PRINT "Blinkende Nachricht"
 90 FOR x=1 TO 200:NEXT
100 NEXT a
110 GOTO 10
```

Blinkende Texte oder Graphiken sind dann sehr sinnvoll, wenn Sie die Aufmerksamkeit auf einen bestimmten Teil des Bildschirms legen wollen.

Im Mode 0 blinken die INKs 14 und 15 von Hause aus; 14 blinkt mit den Farben blau und hellgelb, 15 wechselt zwischen himmelblau und rosa.

SCHNELLE BEWEGUNGEN

Wenn Sie Spiel-Programme schreiben, sind die Bewegungen darin eigentlich erst die wirklich interessanten Zutaten. Blinkende Graphiken und Texte können natürlich auch benützt werden, aber die Essenz der meisten Arcade-ähnlichen Programme ist die Auseinandersetzung mit graphischen Elementen, die der Spieler kontrollieren kann und solchen, auf die er keinen Einfluß hat. So müssen wir also wissen, wie wir unsere Graphik über den Bildschirm bewegen können.

Um Bewegung in ein Bild zu bringen, muß man nicht immer das komplette Bild durch ein anderes ersetzen. Nur die bewegten Bestandteile des Bildes müssen regelmäßig abgeändert werden. Computer-Spiele funktionieren auf diese Art. Nicht das komplette Bild wird ersetzt, sondern nur immer die Teile, die verändert werden. Bewegung wird erzeugt, indem der zu bewegende Teil des Bildes an seiner alten Position gelöscht und an einer neuen Stelle im Bild wieder aufgebaut wird.

Eine andere Methode, um den Eindruck einer Bewegung zu erreichen, ist den zu bewegenden Teil des Bildes in all seinen möglichen Positionen zu zeichnen, aber so, daß er nicht sichtbar ist. D.h. es wird in der Farbe des Hintergrundes vorgezeichnet. Wenn dann die Bewegung ablaufen soll, kann dann jede Position „eingeschaltet" und „abgeschaltet" werden, aber nicht, indem immer wieder neu gezeichnet wird, sondern indem die ursprüngliche Farbe geändert wird. Das kann demonstriert werden, wenn Sie das Programm auf Seite 32 in Kapitel 3 eingeben.

Stellen Sie sicher, daß das Programm in MODE 0 abläuft und fügen Sie dann folgende Extra-Zeilen hinzu (am leichtesten in MODE 1 oder 2):

```
110 FOR n=2 TO 15
120 INK n,1
130 NEXT n
140 FOR n=1 TO 15
150 INK n,24
160 INK n-1,1
170 FOR x=1 TO 80: NEXT x
180 NEXT n
190 INK 15,1
200 GOTO 140
```

Beim Ablaufen des Programmes in MODE 0 (mit RUN) sollten Sie die Scheiben zuerst wie gewöhnlich sehen, dann werden außer der ersten alle unsichtbar werden und die erste Scheibe wird sich schnell über den Bildschirm bewegen.

Wenn Sie genug davon haben und zweimal ESC drücken, werden Sie höchstwahrscheinlich blauen Text auf blauem Hintergrund haben. Geben Sie sorgfältig INK 1,24 ein und der Text sollte wieder erscheinen. Schauen Sie sich jetzt das Listing an. Sie sollten jetzt verstehen, wie das Programm abläuft – wenn nicht ändern Sie mal da oder dort einen Wert und beobachten, was beim Ablauf des Programms passiert! Mit dieser Methode sollte es keine Verständnisschwierigkeiten mehr geben.

Durch die Art und Weise, in der der CPC464 seine Farben organisiert, können Sie dieses Verfahren bei jedem Zeichen, das durch einen PRINT- oder PLOT-Befehl erzeugt wurde, anwenden.

Tanzende Figur

Das einzige Problem bei der Darstellung von Bewegungen mit der Farbänderungs-Methode ist die Limitierung durch die Anzahl der verfügbaren INKs. Auch müssen Sie aufpassen, daß Sie nicht ein unsichtbares Zeichen, das Sie später irgendwann einmal sichtbar machen wollen, überschreiben. Aus diesen Gründen ist die Methode Zeichen mit PRINT darzustellen und wieder zu löschen die feinere Art, um Dinge zum Laufen zu bringen.

Beginnen wir mit der einfachsten Art von Bewegung; einem kleinen Mann, der über den Bildschirm tanzt.

Der CPC464 ist mit vier Zeichen ausgestattet, die direkt für diesen Zweck geschaffen wurden. Sie haben die ASCII-Codes 248 bis einschließlich 251 und sind in Anhang III, Seite 12 und 13 des CPC464 Benutzerhandbuchs aufgeführt. Hier ist ein sehr leichtes Programm, das zeigt, wie diese Zeichen benutzt werden können:

```
 10 REM Taenzer Programm
 20 CLS:MODE 0
 30 DATA 248,250,249,251
 40 FOR a=1 TO 4: READ tanz(a):next
 50 b=1
 60 FOR c=1 TO 4
 70 LOCATE b,12
 80 PRINT CHR$(tanz(c));
 90 FOR x=1 TO 500:next
100 LOCATE b,12:PRINT " ";
110 b=b+1
120 NEXT c
130 IF b<20 THEN GOTO 60
140 GOTO 50
```

Das erste, was wir machen, ist ein Speicherfeld (array) zu laden mit den ASCII-Werten der Zeichen. Dies geschieht in der gleichen Reihenfolge, in der wir sie brauchen.

Dann haben wir innerhalb der Schleife „b", die vom 1 bis 20 geht, eine FOR NEXT Schleife eingerichtet, um jeweils die nächste Figur aus unserer Speichermatrix zu erhalten. Diese wird dann mit dem LOCATE-Befehl an der nächsten x-Koordinate, die durch den Wert von „b" errechnet wird, dargestellt.

Nachdem das jeweilige Zeichen aus unserem Zeichensatz auf den Bildschirm plaziert wurde, tritt die Warteschleife „x=1 to 500" in Kraft. Bevor dann das nächste Zeichen mit einer neuen Position auf dem Bildschirm dargestellt wird, wird die alte Figur gelöscht, indem sie einfach mit einem Leerzeichen überschrieben wird.

Wenn das Programm durchgelaufen ist, löschen Sie Zeile 100 und starten Sie es von neuem. Dadurch werden Sie sehen, wozu diese Programmzeile dient. Sie können weiter experimentieren und z.B. die Zeile 90 mit der Warteschleife löschen.

Wo kommen denn die alle her?

In Teil 1 dieses Lehrbuches haben Sie ein Spiel mit dem Namen BOMBER kennengelernt, das dazu dienen sollte, Ihnen die Grundlagen der Koordinaten-Geometrie verständlich zu machen, indem Sie die Position eines außerirdischen Angreifers schätzen mußten. Wie Sie sich erinnern, tauchte die kleine Kreatur an verschiedenen Bildschirmpunkten auf, die mit Hilfe des RND-Befehls berechnet wurden. Dieses Kommando ist grundlegend, um Buchstaben oder Graphiken mit unvorhersehbaren Bewegungen zu versehen.

RND

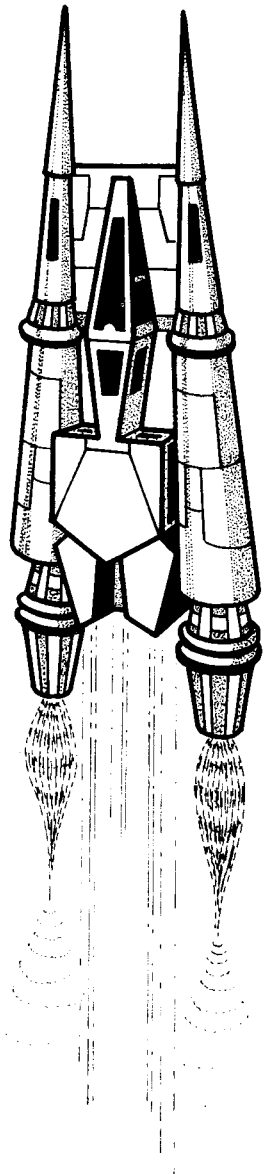
Das folgende Programm erstellt die größte Raumschifflotte, die Sie jemals am Bildschirm gesehen haben.

```
10 REM Invasion vom All
20 CLS:MODE 0
30 LOCATE 19*RND+1,24*RND+1
40 PRINT CHR$(239);
50 FOR t=1 TO 999*RND
60 NEXT
70 GOTO 30
```

Das Interessante an diesem Programm ist, daß die Raumschiffe dank des RND-Befehls in der Zeitschleife (Zeile 50) auch mit verschiedenen Zeitabständen erscheinen.

LOCATE

Vergessen Sie nicht, daß Sie „1" zu den beiden Parametern des LOCATE-Befehls hinzuaddieren müssen. Andernfalls könnte das Produkt 19*RND bzw. 24*RND kleiner als 1 werden und der CPC464 würde das Programm mit der Fehlermeldung „Improper Argument" abbrechen, da der LOCATE-Befehl nur Paramter akzeptiert, die größer oder gleich 1 sind.



Einer über den Durst

Stellen wir uns vor, unser kleiner Tänzer sei ein wenig zu wackelig gewesen, aber versucht immer noch, eine gerade Linie über den Bildschirm zu ziehen. Wir können dies erreichen, indem wir zu dem Programm hinzufügen:

```
65 d=INT(3*RND+5)
```

INT

Wie Sie sich erinnern werden, stellt der INT-Befehl sicher, daß aus einem numerischen Ausdruck ein ganzzahliger Ausdruck wird, indem die Dezimalstellen einfach abgeschnitten werden. Die Multiplikation mit 3 und das

Hinzuaddiren von 5 innerhalb des RND-Befehls dient nur dazu, das Ergebnis innerhalb eines bestimmten Bereiches zu halten.

Die neue Variable „d" kann nun als zweiter Parameter für den LOCATE-Befehl dem Programm hinzugefügt werden:

```
 10 REM Trinker Programm
 20 CLS:MODE 0
 30 DATA 248,250,249,251
 40 FOR a=1 TO 4:READ tanz(a):NEXT
 50 b=1
 60 FOR c=1 TO 4
 65 d=int(3*rnd+5)
 70 LOCATE b,d
 80 PRINT CHR$(tanz(c));
 90 FOR x=1 TO 500:NEXT
100 LOCATE b,d:PRINT " ";
110 b=b+1
120 NEXT c
130 IF b<20 THEN GOTO 60
140 GOTO 50
```

Wenn Sie dieses abgeänderte Programm laufen lassen, werden Sie sehen, daß unser Tänzer jetzt ein bischen unsicher auf den Füßen ist!

NOCH EIN ZIEGELSTEIN IN DER MAUER

Eines der ersten Computerspiele hatte den Namen „Brickout". Der Spieler mußte einen „Schläger" an der Unterseite des Bildschirms vor und zurück bewegen, um dann im richtigen Augenblick einen springenden Ball nach oben zu schlagen, um aus einer „Mauer" einen „Stein" herauszuschlagen. Wir wollen nun dieses Programm in einigen Teilen selbst entwickeln, um das Prinzip der Programmierung von Videospielen zu verstehen. Wie immer werden wir dabei wieder einige neue Schlüsselwörter kennenlernen.

Setzen Sie den Computer zurück und legen Sie sich eine leere Datencassete bereit, um das Programm abzuspeichern.

Zuerst legen wir die Grenzen unseres Bildschirmes fest. Wir könnten dazu den Rand des Bildschirmes nehmen (Border), aber aus Gründen, die Sie später kennenlernen werden, definieren wir uns unsere eigene Begrenzung. Die Zeilennummern sind wohlüberlegt gewählt – ändern Sie sie also noch nicht:

```
10 MODE 1:CLS:GOTO 200
200 LOCATE 1,1: PRINT CHR$(136);
210 FOR c=2 TO 39
220 LOCATE c,1: PRINT CHR$(140);
230 NEXT
240 LOCATE 40,1: PRINT CHR$(132);
250 FOR l=2 TO 24
260 LOCATE 1,l: PRINT CHR$(138);
270 LOCATE 40,l: PRINT CHR$(133);
280 NEXT
300 LOCATE 1,25: PRINT CHR$(130);
310 FOR c=2 TO 39
320 LOCATE c,25: PRINT CHR$(131);
330 NEXT
340 LOCATE 40,25: PRINT CHR$(129);
350 LOCATE 1,1
```

Zeile 350 wurde eingebaut als Hilfsmaßnahme, um das Scrollen des Bildschirms zu verhindern.

Sie sollten mit diesem Programm keine Probleme haben. Momentan erscheint noch die Ready-Meldung auf dem Bildschirm, die wir später auch noch unterdrücken. Als nächstes wollen wir noch einen springenden Ball....:

```
20 x=1: y=-1
30 c=INT(RND*37+2)
40 l=INT(RND*22+2)
50 IF c+x>39 OR c+x<2 THEN x=-x
60 IF l+y>24 OR l+y<2 THEN y=-y
```

```
 80 LOCATE c,l: PRINT " ";
 90 c=c+x:l=l+y
100 LOCATE c,l: PRINT CHR$(224);
110 GOTO 50
350 GOTO 20
```

Wenn Sie das Programm mit RUN starten, sollten Sie einen „Ball" innerhalb der Grenzen des Bildschirmbereiches umher springen sehen.

Aber er ist ein bischen schnell – oder? Wenn Sie seine Geschwindigkeit etwas herabsetzen wollen, können Sie eine Zeile mit einer Warteschleife einfügen:

```
105 FOR d=1 TO 50: NEXT
```

Die Zeilen 20 bis 40 legen die Anfangswerte der Variablen „x" und „y" (um den Ball zu bewegen) sowie „c" und „l" (Position des Balles) fest. In den Zeilen 50 und 60 wird geprüft, ob die nächste Position des Balles außerhalb der Grenzen liegt (wir benutzen die Spalten 2 bis 39 sowie die Zeilen 2 bis 24). Wenn dies der Fall ist, wird der Wert von „x" bzw „y" einfach invertiert.

Mit Zeile 80 wird der Ball an der alten Position gelöscht, indem einfach an dieser Stelle ein Leerzeichen plaziert wird. In Zeile 90 wird die neue Position berechnet und durch Zeile 100 der Ball auf die neue Position gebracht. Zeile 110 komplettiert die Schleife. Eigentlich ganz einfach!

Am besten wäre es, wenn Sie jetzt das Programm abspeichern (SAVE), da wir das Programm verändern wollen, aber nachher wieder den Originalzustand brauchen.

Gleichmäßige Bewegungen des Balles

Jede neue Position des Balles ist notwendigerweise ein volles Zeichen von der alten Position entfernt. Obwohl die Geschwindigkeit des Balles fast den Eindruck vermittelt, als würde er sich gleichmäßig bewegen, ist diese Art der Bewegung noch nicht zufriedenstellend. Um gleichmäßigere Bewegungen zu erreichen müssten wir den Ball jeweils ein halbes Zeichen oder noch weniger weit bewegen. Dies können wir mit dem Kommando TAG erreichen.

Erinnern Sie sich, daß Sie mit TAG anstelle des Text-Cursors den Graphik-Cursor für PRINT-Ausgaben verwenden können. Mit dem Graphik-Cursor stehen Ihnen 640 Bildpunkte in der horizontalen und 400 Bildpunkte in der vertikalen Richtung zur Verfügung (andere Bereiche könnten auch angenommen werden, liegen aber außerhalb des Bildschirm-Bereichs!). Vergessen Sie auch nicht, daß in MODE 1 jedes Pixel durch vier verschiedene Koordinaten-Punkte angesprochen werden kann (zwei vertikale und zwei

horizontale – siehe Kapitel 3). Das bedeutet, daß wir im Graphik-Modus ein Zeichen um 16 Positionen verschieben müssen, um das gleiche Resultat zu erzielen, als wenn wir es im Text-Modus um ein Zeichen verschieben.

Ändern wir das Programm jetzt. Das erste, was wir machen, ist, daß wir den Graphik-Modus einschalten, indem wir folgende Zeile einfügen:

```
45 TAG
```

Der Zahlenbereich, der mit PRINT angesprochen werden kann, ist jetzt viel größer, so daß wir die Zeilen 50 und 60 entsprechend ändern müssen.

```
50 IF c+x>609 OR c+x<16 THEN x=-x

60 IF l+y>383 OR l+y<30 THEN y=-y
```

Die Zeilen 30 und 40 müssen auch abgeändert werden, um dem Ball eine gültige Anfangsposition zu geben:

```
30 c=INT(RND*592+16)

40 l=INT(RND*337+42)
```

Wenn der TAG-Befehl in Aktion ist, hat LOCATE keine Wirkung und wir müssen LOCATE durch MOVE ersetzen:

```
 80 MOVE c,l:  PRINT " ";

100 MOVE c,l: PRINT CHR$(224);
```

Wenn Sie vorher Zeile 105 eingefügt haben, ist es Zeit, sie wieder zu löschen.

Das erste, was Ihnen nicht gefallen wird, ist die geringe Geschwindigkeit. Das liegt daran, daß das Programm jedes Pixel zweimal bestimmen muß. Die Geschwindigkeit kann ohne Einfluß auf die Gleichmäßigkeit verdoppelt werden, wenn Sie die Parameter „x" und „y" verdoppeln.

```
20 x=2: y=-2
```

Lassen Sie das Programm erneut laufen und sehen Sie sich den Erfolg an. Wenn Sie sich nicht sicher sind, was eine bestimmte Zeile oder Variable für eine Funktion hat, ändern Sie sie und versuchen Sie anhand des Ergebnisses beim erneuten Ablauf des Programmes ihre Wirkung zu verstehen.

„Schlagen" des Balles

Wenn Sie mit der Version zwei des Programmes genug experimentiert haben, laden Sie wieder das Originalprogramm, sofern Sie es vorher abgespeichert haben. Wenn Sie es nicht auf Cassette gespeichert haben, ändern Sie das Programm wieder in die ursprüngliche Form zurück. Jetzt ist es an der

Zeit, eine „Schlag-Routine" in unser Programm einzubauen. Der Schläger wird sich an der unteren Seite des Bildschirmbereiches auf und ab bewegen.

Als erstes löschen Sie dazu die Zeilen 300 – 340, die die untere Begrenzung festlegen. (Der Ball wird natürlich immer noch abprallen). Jetzt brauchen wir unseren Schläger. Für diesen Zweck benutzen wir drei CHR$(131)-Zeichen. Das PRINTen des Schlägers muß innerhalb der Hauptschleife des Ball-Programmes geschehen. Deshalb lassen wir diesen Vorgang ablaufen, nachdem der Ball in Zeile 100 gePRINTet wurde. Da nur die horizontale Position des Schlägers verändert werden soll, brauchen wir nur eine Variable.

Um drei CHR$(131)-Zeichen zu PRINTen könnten Sie folgendes eingeben:

```
PRINT CHR$(131);CHR$(131);CHR$(131)
```

Aber erinnern Sie sich an die STRING$-Funktion von Kapitel 7:

```
STRING$(3,131)
```

Fügen Sie jetzt folgende Zeilen ein:

```
4    sch$=" "+STRING$(3,131)+" "
109 LOCATE b,25: PRINT sch$;
```

Die Leerzeichen vor und nach dem Schläger dienen dazu, den „alten" Schläger zu löschen, wenn er sich bewegt. Wir müssen jetzt den Wert der Variablen „b" in einem Bereich zwischen 1 (links) und 36 (rechts) ändern, da der Schläger (einschließlich Leerzeichen) 5 Zeichen lang ist, und Spalte 36 die am weitesten rechts stehende Spalte ist, die noch adressiert werden kann, ohne daß die Begrenzung überschritten wird.

Mit dem Schlüsselwort INKEY$ können wir feststellen, ob eine Taste momentan gedrückt ist oder nicht. Wählen wir eine Taste für die Funktion „links" aus – sagen wir Q, und eine für „rechts" – in diesem Fall P. Sie müssen sich vergewissern, daß CAPS LOCK außer Funktion ist, um statt „q" und „p" nicht „Q" und „P" zu bekommen. Sie können folgendes eingeben:

```
101 ky$=INKEY$

102 IF ky$="q" THEN b=b-1

103 IF ky$="p" THEN b=b+1
```

Natürlich könnte „b" zu groß oder zu klein werden und das Programm dadurch abstürzen. So sollten wir also noch folgende Zeilen hinzufügen:

```
104 IF b>36 THEN b=36
105 IF b<1 THEN b=1
```

Dann geben wir „b" noch einen Anfangswert in Zeile 20:

```
20 x=1: y=-1: b=18
```

Wenn Sie jetzt das Programm ablaufen lassen, werden Sie feststellen, daß der Ball wie gewöhnlich springt und Sie den Schläger durch Drücken von „p" oder „q" (aber nicht beide gleichzeitig) bewegen können. Besonders schnell reagiert der Schläger nicht auf das INKEY$-Kommando – aber es gibt noch eine weitere Möglichkeit.

INKEY

Das Schlüsselwort INKEY ist ähnlich wie das INKEY$-Kommando, fragt aber die Tastatur anders ab und wartet nicht auf eine Antwort der Tastatur. Die Syntax lautet:

```
INKEY (<ganzzahliger Ausdruck>)
```

wobei der ganzzahlige Ausdruck eine der Tasten-Nummern darstellt, die in Anhang 3 auf Seite 16 des Benutzerhandbuches beschrieben sind. Sie werden feststellen, daß Q die Nummer 67 hat und P die Nummer 27.

Geben Sie die geänderten Zeilen ein:

```
101      (damit wird diese Zeile gelöscht)

102 IF INKEY(67)=0 THEN b=b-1

102 IF INKEY(27)=0 THEN b=b+1
```

Sie werden feststellen, daß der Schläger jetzt schneller reagiert als zuvor. Die folgende Zeile kann die Zeilen 102, 103, 104 und 105 ersetzen und ist ein viel eleganterer Weg, um unser Ziel zu erreichen:

```
102 b=b+(INKEY(67)=0
  AND b>1)-(INKEY(27)=0 AND b<36)
```

Bei dieser Art wird von dem logischen Wert der Bedingung innerhalb der Klammern Gebrauch gemacht. Der Wert ist 0, wenn die Bedingung falsch ist und −1, wenn sie wahr ist (siehe Kapitel 7).

Jetzt, da der Schläger schneller ist, können wir noch ein oder zwei Zeilen einfügen um herauszufinden, ob der Ball den Schläger berührt. Der Ball berührt den Schläger, wenn der nächste Zeilen-Wert des Balles 25 ist UND der nächste Spalten-Wert entweder b+1, b+2 oder b+3 ist (da der Schläger eine Länge von drei Zeichen hat).

Sie könnten folgende Zeile eingeben:

```
IF l+y=25 AND (c=b+1 OR c=b+2 OR
  c=b+3) THEN ..... ein Treffer
```

Geben Sie dafür lieber folgende Zeilen ein:

```
60 IF l+y<2 THEN y=-y
65 IF l+y>24 THEN y=-y: IF c>b AND c<b+4
  THEN s=s+1 ELSE s=s-5
```

Die Variable „s“ wird dazu benutzt, um den Spielstand anzuzeigen. Wenn Sie den Ball treffen, bekommen Sie einen Punkt hinzu, wenn Sie Ihn verfehlen, werden Ihnen 5 Punkte abgezogen!

Fügen Sie folgende Zeile ein, um den Spielstand anzuzeigen:

```
106 LOCATE 16,1: PRINT "SPIELSTAND =";s;
```

Das Programm wird nun immer komplexer und die Geschwindigkeit des Balles verlangsamt sich ein wenig. Somit ist die Verwendung des TAG-Modus also unpraktisch, wenn hohe Geschwindigkeiten erforderlich sind.

Kollisionen, Kollisionen

Soweit haben wir jetzt also unser Wissen ausgenutzt, um die Grenzen des Spielfeldes zu bestimmen und Kollisionen festzulegen. Die Begrenzungen müssten eigentlich gar nicht auf dem Bildschirm angezeigt sein, da nicht sie die Kollisionen bestimmen, sondern die Bedingungen im Programm. Was wäre, wenn wir irgendein Zeichen in den Spielbereich setzen? Wie könnten wir eine Kollision entdecken, ohne daß uns die vorherige Position des Zeichens bekannt ist?

Natürlich müssen wir wissen, was wir suchen, oder zumindest etwas davon wissen; z.B. die Gestalt oder die Farbe. Der CPC464 hat kein spezielles Schlüsselwort, um die Bedingungen innerhalb eines ganzen Zeichens abfragen zu können, wie z.B. Nummer des Zeichens oder die Farbe. Dafür gibt es ein Schlüsselwort, um die Bedingungen einer speziellen Koordinate abzufragen. Dieses Schlüsselwort ist TEST mit folgender Struktur:

TEST

```
TEST (<ganzzahliger Ausdruck>,
  <ganzzahliger Ausdruck>)
```

wobei der erste ganzzahlige Ausdruck die x-Koordinate und der zweite ganzzahlige Ausdruck die y-Koordinate angibt. PRINT TEST(100,100) ergibt als Ergebnis die Farbnummer (INK) an der Koordinaten-Position (100,100).

Wenn Sie mit PLOT 150,300,3 einen Punkt bestimmen und dann PRINT TEST (150,300) eingeben, erhalten Sie als Ergebnis 3. Wenn Sie also die Existenz eines Zeichens innerhalb des Zeichenbereiches feststellen wollen, müssen Sie die Spalten- und Zeilenwerte zuerst in „x“- und „y“-Koordinaten umrechnen, bevor sie TEST anwenden können. Die Umrechnung der Koordinaten erfordert genaue Überlegungen, vor allem in der vertikalen Richtung, da die Zeilen vom oberen zum unteren Ende numeriert sind, während die „y“-Koordinaten von unten nach oben ansteigen. Je mehr Sie mathematisch begabt sind, desto leichter werden Sie sich tun.

Eine Lösung dieses Dilemmas liegt in der Verwendung einer zweidimensionalen Speichermatrix, die so groß ist wie die Anzahl der Spalten und Zeilen. Geben Sie z.B. DIM a (40,25) ein. Jede subskribierte Variable wäre dann ein „Schattenbild“ des Bildschirms. Wenn Sie z.B. folgendes eingegeben haben:

```
LOCATE 15,7: PRINT "*": LOCATE 34,13:
  PRINT "O"
```

dann füllen Sie damit auch die Speicherbereiche folgendermaßen auf:

```
a(15,7)=1: a(34,13)=2
```

wobei die Nummern 1 und 2 ein einzelnes Zeichen kennzeichnen. Sie könnten auch den ASCII-Code abspeichern, wenn Sie wollen:

```
a(15,7)=ASC("+"):a(34,13)=ASC("O")
```

Um eine Kollision auf dem Bildschirm zu erkennen, müssen Sie regelmäßig die Speicherinhalte abtesten. Wir wollen jetzt noch ein paar Zeilen abändern bzw. einfügen:

```
5    DIM a(40,25)

40   l=INT(RND*10+13)

107  IF a(c,l)=1 THEN a(c,l)=0:s=s+1:x=-x:y=-y

300  PEN 3

310  FOR l=8 TO 12

320  FOR c=2 TO 39

330  LOCATE c,l: PRINT CHR$(143);

340  a(c,l)=1

350  NEXT c

360  NEXT l

370  PEN 1

380  GOTO 20
```

Die Zeilen 300 – 360 bauen die Mauer auf und die dazugehörige Speichermatrix, die in Zeile 5 dimensioniert wurde. Zeile 40 stellt sicher, daß der Ball immer unterhalb der Mauer startet. Bei einer Kollision wird die Richtung des Balles umgekehrt und Ihr Spielstand um 1 erhöht.

AUF GEHT'S

So wie das Spiel momentan ist, kann man es kaum als spannend bezeichnen. Aber es war ja auch nicht Absicht, ein spannendes Spiel zu programmieren, sondern vielmehr sollte Ihnen gezeigt werden, wie ein Spiel entwickelt und ausgebaut werden kann. Auch sollten Sie dabei die einzelnen Abläufe und Probleme verstehen, die sich bei der Entwicklung eines Programmes ergeben. Jetzt liegt es an Ihnen, wie Sie das Programm weiter gestalten. Hier sind einige Verbesserungsvorschläge:

- ☐ Lassen Sie den Ball verschwinden, wenn er den Schläger verfehlt hat und ersetzen Sie ihn durch einen neuen.
- ☐ Ändern Sie die Variablen, die die Richtung des Balles angeben, so daß er auch mit anderen Winkeln als mit 45 Grad abprallen kann.
- ☐ Lassen Sie den Ball von der Ecke des Schlägers abprallen, wie Sie es normalerweise erwarten.
- ☐ Lassen Sie den Ball im gleichen Winkel von der Mauer abprallen, in der er aufprallt.
- ☐ Fügen Sie Töne hinzu.
- ☐ Bauen Sie einige Hindernisse in das Spielfeld ein.

SPIELZEIT

Das nächste Programm auf der Datencassette A heißt BLITZ. Mit diesem Spiel können Sie all Ihre aufgestauten Agressionen abbauen. Ihr Flugzeug verliert ständig an Höhe über der Stadt und läuft Gefahr, in eines der Gebäude zu stürzen. Ihre einzige Chance ist, die ganze Ansiedlung zu bombadieren, bis Sie dem Erdboden gleich ist, um sicher zu landen!

Glückliche Landung.

TEST

Wenn Sie sich in einer kleinen Pause von der anstrengenden Bombardierung erholt haben, machen Sie mit SAT8 weiter.

Kapitel 9
SOUND FX

Als wir die SOUND-Kommandos in Teil 1 dieses Kurses kennenlernten, mussten wir uns dabei auf die einfachsten Dinge beschränken. Jetzt, da Sie schon mehr Programmierkenntnisse besitzen, können wir etwas mehr ins Detail gehen und Dinge behandeln, die wir vorher ausgelassen haben. Unter Berücksichtigung unserer erweiterten Kenntnisse können wir das SOUND-Kommando folgendermaßen beschreiben:

SOUND

```
SOUND <Kanal-Status>,<Ton-Periode>
  [,<Dauer>[,<Lautstaerke>[,<Lautstaerken-
  variation>[,<Tonvariation>[,<Geraeusch-
  folge>]]]]]
```

Alle Parameter sind ganzzahlige Ausdrücke. Lassen Sie sich durch die Anzahl und Stellung der Klammern nicht verwirren. Das bedeutet nur, daß Sie Argumente von rechts beginnend weglassen können, aber nicht irgendwo von der Mitte aus.

KANÄLE

Der <Kanal-Status> ist der komplizierteste Teil unseres SOUND-Kommandos. Im Benutzer-Handbuch können Sie nachlesen, daß dies eine Ganzzahl zwischen 1 und 255 ist! Keine Angst, im Moment wollen wir nur wissen, wie die drei Kanäle des CPC464 ausgewählt werden.

Diese Kanäle werden mit A, B und C bezeichnet und sind mit ihrer Tonperiode und ihrer Lautstärkenvariation völlig unabhängig voneinander. Es kann jedoch mehr als ein Kanal in einem SOUND-Kommando angesprochen werden, wie Sie in der folgenden Übersicht sehen:

Kanal Status	*Kanal*
1	nur A
2	nur B
3	A und B
4	nur C
5	A und C
6	B und C
7	A, B und C

Das mag zwar ein wenig unübersichtlich aussehen, aber es funktioniert!

Sie könen auch Ihre Schneider HiFi-Anlage an der Rückseite des CPC464 anschließen, wenn Sie das ganze in Stereo hören wollen. Kanal A ist links, Kanal B rechts und Kanal C ist eine Mischung der beiden.

VARIATIONEN

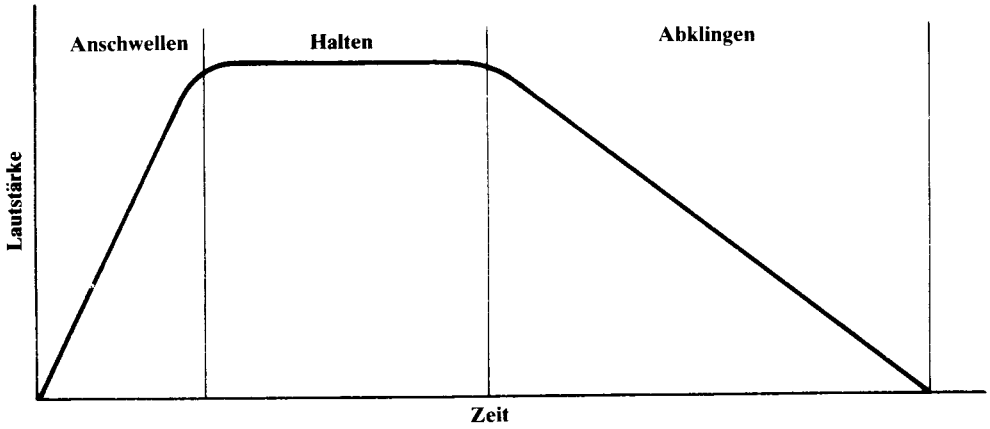
Mit der Lautstärken-Hüllkurve kann die <Lautstärke>, die beim SOUND-Kommando angegeben wird, variiert werden. Mit der Ton-Hüllkurve kann die Tonhöhe (Pitch) verändert werden, die in <Periode> angegeben ist. Diese Abänderungen können wirksam sein, solange der Ton abläuft (Tondauer). Sie wissen ja bereits, daß Sie bis zu 15 Lautstärken- und bis zu 15 Ton-Hüllkurven in einem Programm angeben können. Was Sie nicht wissen ist, daß jede Hüllkurve wieder bis zu fünf Abschnitte besitzen kann! Lesen Sie weiter!

Lautstärken-Hüllkurve

Eine Hüllkurve mit 5 Abschnitten ist auf den ersten Blick ganz schön entmutigend. Schauen wir uns einfach zuerst eine Hüllkurve mit drei Abschnitten an. Laden Sie von der Datencassette A das Programm **ZOUNDS** und starten Sie es. Wenn Sie die Tonhüllkurve zunächst ignorieren, stehen Ihnen drei Möglichkeiten zur Auswahl: „Anschwellen", „Halten" und „Abklingen". Das Diagramm zeigt Ihnen, was diese Angaben bedeuten.

Es ist ungefähr wie bei einer Geschichte mit Einleitung, Hauptteil und Ende. Anders als bei einer Geschichte ist jedoch oft die Einleitung und das Ende am interessantesten. Wenn Sie das Programm **ZOUNDS** ablaufen lassen, können Sie die verschiedenen Töne vergleichen, die durch unterschiedliche Möglichkeiten von „Anschwellen", „Halten" und „Abklingen" verursacht werden.

Typische Töne mit einem schnellen Anstieg findet man z.B. bei „Schlaginstrumenten" wie Gitarren. Andere Beispiele dafür sind Donner, Trommelschläge und Explosionen.



Töne mit einem langsamen Anstieg findet man beispielsweise bei langen Orgeltönen oder bei Blechinstrumenten, z.B. bei einer Tuba. Umgekehrt hört der Ton auch sofort auf, wenn Sie bei einer Tuba zu blasen aufhören (schnelles Abklingen), wogegen der Ton einer Gitarre lange nachklingt (langsames Abklingen).

Somit können Sie sehen, wie eine mehrteilige Lautstärken-Hüllkurve es uns ermöglicht, natürliche Töne so originalgetreu wie möglich nachzuvollziehen. Schauen wir uns jetzt das ENV-Kommando nochmal an:

ENV

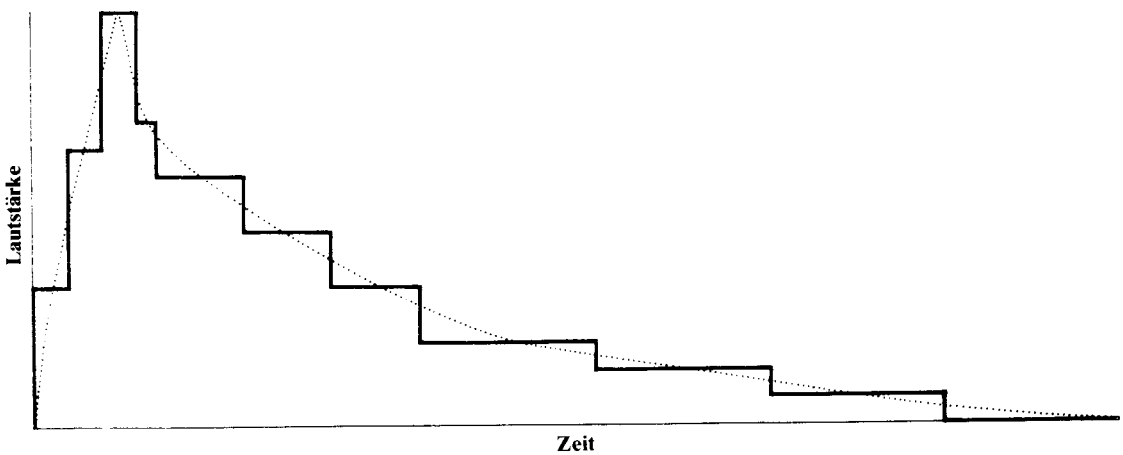
```
ENV <Huelkurvennummer>[,<Abschnitte>]
```

Wie Sie bereits wissen, beinhaltet jeder <Abschnitt>

```
<Schrittanzahl>, <Schrittweite>, <Pausezeit>
```

und wie Sie unlängst erst herausgefunden haben, können Sie dabei bis zu fünf Abschnitte angeben.

Hier ist ein Beispiel:



Die Codierung dafür lautet:

```
ENV 1, 3,5,2, 1,-4,1, 4,-2,5, 1,0,5, 3,-1,10
```

Wenn diese Hüllkurve mit einem SOUND-Kommando verbunden wird, dessen Anfangslautstärke Null ist, erreichen wir folgende Veränderung der Lautstärke:

Abschnitt	*Wirkung*
1	Mäßiges Anschwellen auf Maximal-Lautstärke
2	Schnelles Abklingen auf Pegel 11
3	Mittleres Abklingen auf Pegel 3
4	Kurzes Anhalten
5	Langsames Abklingen auf Pegel 0

Sie sehen, daß das Anhalten des Tones einfach durch die Bestimmung eines Tones mit der Veränderung Null erreicht wurde – die Länge der Pause wird durch die „Periode" bestimmt.

Jetzt ist es an der Zeit, daß Sie mit Ihren eigenen Lautstärke-Hüllkurven experimentieren. Wenn Ihrem SOUND-Kommando eine „Dauer" von Null zugewiesen wird, bedeutet dies, daß die Dauer vollständig vom ENV-Ausdruck kontrolliert wird. Sie können dies im folgenden Listing des Programmes **ZOUNDS** sehen. Versuchen Sie für ENV 13, ENV 14 und ENV 15 Ihre eigenen Hüllkurven zu erstellen.

```
100 ' Zounds : Huellkurven Demos
110 '
120 ' von George Tappenden
130 ' DA 25/9/84  ESCON 25/2/85
140 '
150 CLEAR
160 DEF FNt(x$)=INSTR("FSLMH",x$)-1
170 FOR i=0 TO 4:READ n$(i):NEXT
180 READ e$,a$,s$,d$,t$
190 '
200 ' Bildschirm
210 '
220 MODE 2   : BORDER 26
230 INK 0,26 : INK 1,0
240 PAPER 0  : PEN 1
250 '
260 ' Beschreibung
270 '
280 PRINT STRING$(10,"*");" LAUSTAERKE-HUELLKURVEN ";S
TRING$(12,"*");
290 PRINT SPACE$(6):STRING$(4,"*");" TON-HUELLKURVEN "
;STRING$(3,"*")
300 PRINT : PRINT
```

```
310 PRINT e$,a$,s$,d$,e$,t$
320 PRINT
330 FOR i=1 TO 12
340 READ a$,b$,c$
350 a=FNt(a$):b=FNt(b$):c=FNt(c$)
360 PRINT i,n$(a),n$(b),n$(c)
370 NEXT
380 FOR i=1 TO 4
390 READ tn$
400 LOCATE 53,4+i*2 : PRINT i
410 LOCATE 66,4+i*2 : PRINT tn$
420 NEXT
430 '
440 ' Lautstaerke-Huellkurven
450 '
460 ENV 1,1,15,1,1,0,40,1,-15,1
470 ENV 2,1,15,1,1,0,40,15,-1,4
480 ENV 3,1,15,1,1,0,5,15,-1,4
490 ENV 4,1,15,1,1,0,5,1,-15,1
500 ENV 5,5,3,1,1,0,40,1,-15,1
510 ENV 6,5,3,1,1,0,40,15,-1,4
520 ENV 7,5,3,1,1,0,5,1,-15,1
530 ENV 8,5,3,1,1,0,5,15,-1,4
540 ENV 9,15,1,2,1,0,40,1,-15,1
550 ENV 10,15,1,2,1,0,40,15,-1,4
560 ENV 11,15,1,2,1,0,5,1,-15,1
570 ENV 12,15,1,2,1,0,5,15,-1,4
580 '
590 ' Ton-Huellkurven
600 '
605 ENT -1,1,0,1
610 ENT -2,2,1,2,2,-1,2
620 ENT 3,100,1,2
630 ENT 4,100,-1,2
650 '
660 ' Benutzer-Auswahl
670 '
680 WHILE ( n>=0 ) AND ( t>=0 )
690 '
700 LOCATE 10,22 : PRINT "  "
710 LOCATE 10,23 : PRINT "  "
720 LOCATE  1,20 : PRINT "Geben Sie die Huellkurven-Nu
mmer ein"
730 PRINT
740 INPUT "fuer Lautstaerke ";n
750 INPUT "fuer Ton         ";t
760 '
770 ' Abspielen
780 '
790 FOR zeiten=1 TO 3
800 LOCATE 6,(n+5)   : PRINT "*****"
810 LOCATE 58,(2*t+4): PRINT "*****"
820 SOUND 7,200,0,0,n,t
```

```
830 FOR abklingen = 1 TO 1500 : NEXT
840 LOCATE 6,(n+5)    : PRINT SPACE$(5)
850 LOCATE 58,(2*t+4): PRINT SPACE$(5)
860 FOR abklingen = 1 TO 500 : NEXT
870 NEXT
880 '
890 WEND
900 END ' Programmende
910 '
920 ' Data-Anweisungen fuer die Beschreibung
930 '
940 DATA schnell,langsam,lang,mittel,kurz
950 DATA Huellkurve,Anstieg,Halten,Abklang,Typ
960 DATA F,L,F,F,L,S,F,H,S,F,H,F
970 DATA M,L,F,M,L,S,M,H,F,M,H,S
980 DATA S,L,F,S,L,S,S,H,F,S,H,S
990 DATA ruhig,vibrato,fallend,ansteigend
```

Ton-Hüllkurven

Vieles, was wir über die Struktur der Lautstärken-Hüllkurve gesagt haben, trifft auch auf die Ton-Hüllkurve zu. Diesmal ist es die Tonhöhe, die während des Ablaufs eines SOUND-Kommandos verändert werden kann. Hauptanwendungsgebiet bei der Darstellung von Klang-Effekten ist die Simulation von Geräuschen schneller Flugzeuge oder Fahrzeuge.

Wenn ein Düsenjäger über uns hinwegfliegt, wird das Geräusch, das er verursacht, immer lauter und lauter, bis er direkt über uns ist, und dann wird es immer weicher, je weiter er weg ist. Dabei ändert sich auch die Tonhöhe durch den sog. Doppler-Effekt – sie sinkt kontinuierlich von dem Zeitpunkt an, zu dem das Flugzeug direkt über uns ist, bis es außer Hörweite ist. Sie können eine beschleunigte Version im **ZOUNDS**-Programm hören, wenn Sie ENV 5 und ENT 3 benutzen. Dieser Effekt wird oft in Computer-Spielen angewendet, um Echtheit vorzutäuschen und Spannung zu erzeugen.

Der gegenteilige Effekt tritt verständlicherweise auf, wenn Sie ein startendes Flugzeug simulieren, bei dem der Pilot Vollgas geben muß, bis die Turbinen ihre Maximalleistung erreichen. Die Höhe des Tones steigt dann kontinuierlich an, bis die Turbinen ihre maximale Drehzahl erreichen. Sie können sich den Eindruck eines ansteigenden Tones verschaffen, wenn Sie ENT 4 mit irgendeiner der länger anhaltenden Lautstärken-Hüllkurven im Programm **ZOUNDS** verbinden. Beachten Sie, daß der ENT-Ausdruck keinen Einfluß auf die Länge des Tones hat (im Gegensatz zu ENV), sondern nur auf die Tonhöhe.

ENT

Die Struktur des Ausdrucks ist die gleiche wie bei der Lautstärken-Hüllkurve:

```
ENT <Huellkurvennummer>,<Abschnitte>
```

wobei bis zu 15 <Hüllkurvennummern> und bis zu fünf <Abschnitte> angegeben werden können. Jeder <Abschnitt> beinhaltet:

```
<Schrittanzahl>,<Schrittweite>,<Pausezeit>
```

Alles sind ganzzahlige Ausdrücke. Die <Schrittweite> gibt die gewünschte Änderung der Tonperiode an, und ist positiv, wenn Sie die Tonhöhe herabsetzen wollen, und negativ, wenn Sie die Tonhöhe heraufsetzen wollen. Wenn Sie darüber nachdenken werden Sie feststellen, daß dies so genau richtig ist!

Wenn Sie ein Minuszeichen vor die Hüllkurvennummer setzen, wird das Kommando so oft wiederholt, bis die im SOUND-Kommando angegebene Dauer des Tones erreicht ist. Dies ist z.B. nützlich bei der Darstellung eines Vibrato-Tones.

SPIELZEIT

Wenn Sie jetzt einiges davon hören möchten, was an Klang-Effekten in Ihrem CPC464 steckt, laden Sie `ZAPPOW2`, das nächste Programm auf der Datencassette A, und lassen Sie es ablaufen. Dieses Programm ist in diesem Manual nicht aufgelistet, aber Sie können es auf dem Bildschirm anschauen, wenn Sie wollen.

Unser Favorit ist der Zug!

TEST

Viele Aspekte der SOUND-Kommandos werden Ihnen bereits aus Teil 1 bekannt sein. Das wirklich Interessante dabei ist, daß Sie die verschiedenen Elemente zusammenstellen können, daß Sie den Klang den Sie hören wollen, auch bekommen. Lassen Sie SAT9 laufen, um zu testen, wie weit Sie das Prinzip der mehrteiligen Hüllkurven verstanden haben.

Kapitel 10
MUSIK

Bevor Sie mit desem Kapitel anfangen, sollten Sie sich noch einmal verdeutlichen, daß es vier Arten von Menschen auf dieser Welt gibt. Da gibt es die, die Computer programmieren und Noten lesen können; diejenigen, die nicht Computer programmieren, aber Noten lesen können; die, die Computer programmieren, aber keine Noten lesen können; und diejenigen, die weder Computer programmieren noch Noten lesen können.

Gerechterweise sagen wir Ihnen gleich, daß Sie sich mit dem, was folgt, etwas hart tun werden, wenn Sie wirklich keine Noten lesen können. Es wäre also durchaus verständlich, wenn Sie gleich zum nächsten Kapitel über springen.

BOGEY MANN

In Teil 1 zeigten wir Ihnen einige Programmzeilen, die Sie eingeben konnten, um eine „bekannte Weise" anzustimmen. Wie Sie wissen, hieß die Melodie *Colonel Bogey*. Seitdem haben Sie vielleicht selbst schon ein paar Lieder ausprobiert. Hier ist zur Erinnerung nochmal das Original:

```
10 REM bekannte Weise
20 SOUND 1,213
30 SOUND 1,253,60
40 SOUND 1,0,40
50 SOUND 1,253
60 SOUND 1,239
70 SOUND 1,213
80 SOUND 1,127,40
```

```
 90 SOUND 1,0,1
100 SOUND 1,127,40
110 SOUND 1,159,60
120 END
```

Aber dies ist eine sehr ungeschickte Art, um Musik zu programmieren – besonders wenn es sich um längere Stücke handelt. Sie werden jetzt lernen, wie man die Kommandos DATA und READ anwendet, um eleganter und mit weniger Aufwand Musik auf dem CPC464 zu spielen.

Statt einer Serie von SOUND-Kommandos ist es besser, wenn Sie als Argumente Variablen benutzen. Zum Beispiel:

```
SOUND kan,per,dau,lau,ev,et,ger
```

Sie können jetzt alle Werte in DATA-Anweisungen angeben und vor der Ausführung des SOUND-Kommandos mit READ die Variablen aktualisieren.

Kommen wir wieder zu *Colonel Bogey* zurück. Der Einfachheit halber lassen wir Lautstärke und Hüllkurve beiseite und benutzen nur Kanal 1. Unsere Hauptschleife sieht dann folgendermaßen aus:

```
10 FOR zaehler=1 TO 10
20 READ per,dau
30 SOUND 1,per,dau
40 NEXT zaehler
50 END
```

Wie Sie sehen, spielt diese Routine jede Melodie, die aus 10 Noten besteht, die in der DATA-Anweisung untergebracht sind. Um unser neues Programm zu vervollständigen müssen wir noch eingeben:

```
60 DATA 213,20,253,60,0,40,253,20,239,20,
213,20,127,40,0,1,127,40,159,60
```

Wenn Sie dieses Programm laufen lassen, wird es sich anhören wie die ursprüngliche Version von *Colonel Bogey* (vorausgesetzt Sie haben sich nicht vertippt). Aber Sie werden sich fragen, warum die ganzen Umstände? Nun, da gibt es viele Vorteile. Zunächst einmal ist es viel leichter, bestimmte Konstanten in DATA-Anweisungen zu ändern, als sich durch endlos viele SOUND-Kommandos durchzuarbeiten, die nur an einer Stelle geändert werden müssen. Ein anderer Vorteil ist, daß Sie leicht bestimmte Argumente

für einen Teil oder für die ganze Melodie ändern können. Versuchen Sie mal das SOUND-Kommando folgendermaßen zu ändern:

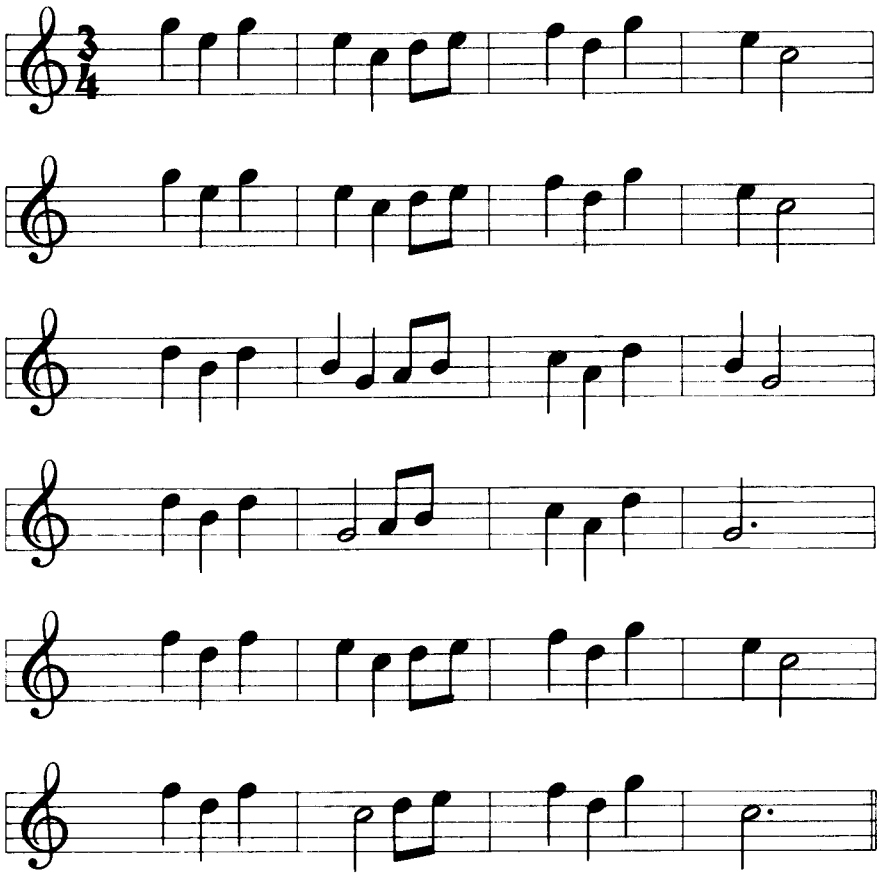
```
30 SOUND 1,per*2,dau
```

Es könnte keinen einfacheren Weg geben, um eine Melodie um eine Oktave zu verändern.

ORANGEN UND ZITRONEN

Gehen wir jetzt zurück zur Musik und schauen wir uns eine neue Melodie an – diesmal einen Walzer.

Orangen und Zitronen **Volksweise**



Wie viele andere Molodien besteht auch diese aus einer Reihe von Zeilen mit je vier Takten. Aber das Interessante daran ist, daß in allen sechs Zeilen nur zwei Grund-Muster an Noten vorkommen. Die ersten beiden Zeilen sind natürlich identisch. Aber auch die dritte Zeile ist gleich, nur drei Halbtöne tiefer. Die vierte Zeile verwendet das zweite Muster, obwohl es nur etwas anders als das erste ist, aber die fünfte Zeile hat wieder das erste Muster. Schließlich ist die sechste Zeile wieder genauso wie die vierte Zeile, aber um fünf Halbtöne höher.

Die Melodie in BASIC

```
100 REM Orangen und Zitronen
110 REM
120 FOR zeile=1 TO 6
130 REM
140 IF zeile=1 THEN RESTORE 360
150 IF zeile=2 THEN RESTORE 360
160 IF zeile=3 THEN RESTORE 360
170 IF zeile=4 THEN RESTORE 410
180 IF zeile=5 THEN RESTORE 360
190 IF zeile=6 THEN RESTORE 410
200 REM
210 FOR note=1 TO 12
220 READ tonhoehe,dauer
230 REM
240 IF zeile=1 THEN periode=tonhoehe
250 IF zeile=2 THEN periode=tonhoehe
260 IF zeile=3 THEN periode=tonhoehe*1.335
270 IF zeile=4 THEN periode=tonhoehe
280 IF zeile=5 THEN periode=tonhoehe
290 IF zeile=6 THEN periode=tonhoehe/1.335
300 REM
```

```
310 SOUND 7,periode,dauer
320 NEXT note
330 NEXT zeile
340 END
350 REM
360 REM erstes Muster
370 REM
380 DATA 159,40,190,40,159,40,190,40,239,40,213,20
390 DATA 190,20,179,40,213,40,159,40,190,40,239,80
400 REM
410 REM zweites Muster
420 REM
430 DATA 213,40,253,40,213,40,319,40,319,40,284,20
440 DATA 253,20,239,40,284,40,213,40,319,80,319,40
```

Wie Sie sehen beinhaltet das Programm eine Schleife, die die Noten einer Zeile abspielt. Diese ist wiederum innerhalb einer zweiten Schleife, in der die entsprechende Zeile ausgewählt wird. Bei der Zeilenauswahl wird das RESTORE-Kommando dazu verwendet, um den Pointer auf den entsprechenden Block mit Data-Anweisungen zu setzen, so daß mit dem READ-Kommando in Zeile 220 die entsprechenden Daten in die Variablen „tonhöhe" und „dauer" eingelesen werden können.

Die Wechselbeziehung zwischen Musik-Noten und den Angaben, die aus den Data-Bereichen in „tonhöhe" eingelesen werden, werden in Anhang VII des Benutzerhandbuches dargestellt.

Die Hilfs-Variable „tonhöhe" wird dann dazu benutzt, um mit Hilfe der zweiten Zeilen-Auswahl die Variable „periode" zu generieren. Eine annähernde Darstellung der Verschiebung um fünf Halbtöne wird erreicht, indem „tonhöhe" mit 1.335 multipliziert bzw. dividiert wird.

Wir kommen nun zu unserem einzigen SOUND-Kommando in Zeile 310. Sie werden bemerkt haben, daß die Kanalangabe den Wert 7 hat, sodaß alle drei Kanäle gleichzeitig aktiv sind.

Wenn Sie sich das Musikstück, das ein paar Seiten zurück liegt, ansehen, werden Sie bemerken, daß das zweite Muster keine 12, sondern nur 10 Noten hat. Warum haben wir dann 12 Konstanten in den DATA-Anweisungen? Wenn Sie genau hinsehen werden Sie feststellen, daß wir die zwei langen Gs geteilt haben.

Weiterentwicklungen

Unsere vorherige Melodie wurde absichtlich so einfach wie möglich gehalten, damit Sie zunächst einmal das allgemeine Prinzip der Musikprogrammierung verstehen. Die Lautstärke zwischen den einzelnen Sätzen wurde nicht verändert; Lautstärken- und Tonvariationen wurden völlig außer acht gelassen. Aber warum sollten wir diese Dinge nicht auch in das Programm einbauen? Sie könnten die Hauptschleife in eine weitere Schleife einbauen, in der die Melodie öfter wiederholt wird. Wie wäre es mit verschiedenen Lautstärkehüllkurven für jeden Durchlauf der Schleife?

Eine weitere Möglichkeit wäre, daß Sie Ihre HiFi-Anlage an den Stereo-Ausgang auf der Rückseite des CPC464 anschließen (an der kleinen Buchse mit der Bezeichnung „I/O"). Sie könnten dann jeweils den Kanal zwischen den einzelnen Zeilen wechseln, sodaß Sie einen „Ruf und Antwort"-Effekt zwischen den beiden Stereo-Kanälen erreichen.

Auf jeden Fall sollten Sie jetzt in der Lage sein, die meisten einfachen Melodien auf dem CPC464 zu programmieren und dabei die kreativen Effekte, die mit dem Sound-Kommando im Schneider-BASIC möglich sind, so gut es geht auszunutzen.

KLASSISCHE MUSIK

Das nächste Programm auf Datencassette A heißt MENUETT von J.S. Bach. Es gibt kaum einen anderen Komponisten, dessen Musikstücke für so viele Instrumente transponiert wurden. Selbst wenn er ursprünglich ein Stück für ein Cembalo geschrieben hatte, so ist es heute sicher auch für Klavier, Orgel, Gitarre, Orchester oder für Blaskapelle umgeschrieben. Schließen wir uns diesem Trend an – mit einer Version für den Schneider CPC464.

Abgesehen davon, daß dies eine besonders gute Aufführung des Stücks ist, gibt es einige interessante Aspekte, über die man reden sollte, bevor oder nachdem Sie das Stück gespielt haben.

Es ist unsere Absicht Ihnen zu zeigen, wie gut man ernste Musik auf dem CPC464 spielen kann. Dabei mußten wir einige Kommandos und Tricks anwenden, für die Sie in diesem Teil des Kurses keine Erklärung finden werden. Aber es hält Sie niemand auf, wenn Sie sich dieses Programm am Bildschirm anschauen wollen.

Als erstes werden Sie sehen, daß wir drei Stimmen für die Melodie, Begleitung und Baß-Zeilen verwenden. Dies setzt eine genaueste Beschreibung der komplizierten Operationen der drei Kanäle voraus. Damit müssen wir auf Teil drei dieses Kurses warten. Sie werden auch das eine oder andere unbekannte Kommando finden. Wenn Sie diese Kommandos verstehen wollen, schauen Sie ruhig im Benutzerhandbuch nach.

Nachdem das alles gesagt ist, geben Sie doch jetzt zuerst einmal das Programm auf dem Bildschirm aus und sehen Sie sich an, wie es zusammengesetzt ist.

TEST

Keine Angst – Ihr Musikwissen wird jetzt nicht abgefragt. Lassen Sie SAT10 ablaufen, um zu überprüfen, ob Sie die Schlüsselwörter und Techniken, die in diesem Kapitel verwendet wurden, verstanden haben.

Kapitel 11
ABENTEUER

Eine der populärsten Arten von Computer-Spielen ist das Abenteuer, bei dem Sie in eine Welt der Fantasie gelangen, die nur in der Vorstellung des Programmierers existiert. Durch die Eingabe von Kommandos und Informationen werden Sie in eine Geschichte integriert, die der Computer durch Mitteilungen (auf dem Bildschirm) wie Ihren Standort in Raum und Zeit, Ihre Umgebung und das letzte Ereignis der Geschichte, lebendig macht. Das faszinierende daran ist, daß Sie den Lauf der Ereignisse selbst beeinflußen können und das Spiel jedesmal anders verläuft.

Viele Abenteuer-Spiele basieren auf der Idee eines Irrgartens oder eines großen Gebäudes mit vielen Räumen, in dem Sie Ihren Weg finden müssen auf der Suche nach nützlichen oder wertvollen Gegenständen. Genauso gut kann es Fallen und Zufallsereignisse oder sogar (in der Geschichte) tödliche Gefahren geben. Sinn des Spieles ist letztendlich die Wiederbeschaffung eines bestimmten Gegenstandes oder die erfolgreiche Flucht.

Laden Sie als nächstes das Programm `ADVENTUR` von der Datencassette A. Dies ist das Beispiel, anhand dessen wir mit Ihnen in die Programmierung von Abenteuer-Spielen einsteigen wollen.

ROLAND IM HAUS

ADVENTUR ist typisch für diese Art von Programmen. Die Geschichte hat folgenden Inhalt: Sie wollen einen ruhigen Nachmittag mit der Programmierung Ihres CPC464 verbringen, aber irgendjemand hat alle Dinge, die Sie dazu brauchen, in verschiedenen Räumen Ihres Hauses versteckt. Sie müssen alles finden und Ihr System irgendwo lückenlos aufbauen.

Es gibt drei Arten von Kommandos:

- *Richtungs-Kommandos:*
  n = nord
  o = ost
  s = süd
  w = west
  h = hinauf
  a = abwärts

- *Kommandos mit einem Wort*
  help
  schau
  zeige
  beende

- *Kommandos mit zwei Wörtern*
  hole ....
  leg ....
  öffne ....
  schliesse ....
  ziehe ....
  stecke ....

Geben Sie jedesmal eines der obigen Komandos ein, wenn Sie das Prompt-Zeichen (>) auf dem Bildschirm sehen. Wenn Sie nicht mehr weiterwissen, können Sie jederzeit „help" eingeben.

SPIELZEIT

Schalten Sie mal ein wenig ab und genießen Sie einfach das Spiel. Gerade so ein einfaches Abenteuer-Spiel wie dieses kann einen leicht „süchtig" machen. Sie brauchen also nicht überrascht zu sein, wenn Sie sich eine Zeit lang nicht mehr um dieses Buch kümmern können.

NUR ZUR WIEDERHOLUNG

Wenn Sie das Abenteuer ein paarmal gespielt haben und wenn Sie herausgefunden haben, wo alles liegt und was Sie brauchen, um das Spiel zu beenden, möchten Sie vielleicht selbst ein paar Ihrer Ideen einbauen. Vielleicht möchten Sie noch ein paar Räume oder Gegenstände hinzufügen, oder vielleicht wollen Sie die Räume umbenennen und eine Höhle daraus machen. Was immer Sie auch tun wollen – Sie werden nichts ändern können, solange Sie nicht verstehen, wie das Programm aufgebaut ist.

Wenn Sie das Programm-Listing am Ende dieses Kapitels studieren, werden Sie ziemlich erstaunt sein über die Komplexität dieses Programmes. Machen Sie sich keine Sorgen; Sie werden schnell sehen, wie einfach es eigentlich ist, und Sie sollten erkennen, daß die verwendeten Kommandos alle irgendwann

einmal in den vorherigen Kapiteln erklärt wurden. Es gibt jedoch ein paar Punkte, die Sie wissen sollten, bevor Sie beginnen.

Einige Anmerkungen

Sie werden einige Variablen entdeckt haben, hinter denen ein Prozent-Zeichen steht (%). Das erste Mal tritt dies in Zeile 500 auf:

```
FOR i% = 0 TO 5
```

Dieses Prozent-Zeichen teilt dem CPC464 mit, daß er die Variable in einer bestimmten Art behandeln muß.

Beim Umgang mit Zahlen, z.B. beim Rechnen oder Abspeichern, benutzt der CPC464 die sog. „Gleitkomma"-Arithmetik. Dies ist auch wirklich sinnvoll, da Sie damit auch Zahlen mit Dezimalstellen hinter dem Komma berücksichtigen können, z.B. pi (3.1415926).

Solche Zahlen werden „reelle" Zahlen genannt.

In Teil 1 dieses Kurses wurde Ihnen bereits erklärt, daß es trotz der Cleverness des CPC464 möglich ist, daß Sie ein Ergebnis erhalten, das nicht ganz Ihren Vorstellungen entspricht. Sie haben z.B. eine Rechnung in Ihrem Programm, deren Ergebnis 5 sein sollte. In einer Programmzeile haben Sie stehen:

```
IF x=5 THEN GOTO 2000
```

Sie merken jedoch, daß sich der CPC464 weigert, auf Zeile 2000 zu gehen und lassen sich den Wert von „x" ausdrucken. Als Antwort erhalten Sie 4.9999999. Was passierte war ganz einfach, daß irgendwo im Speicher eine kleine Ungenauigkeit multipliziert wurde und dadurch groß genug wurde, um die Ursache für diesen Fehler zu sein. Sie glauben nicht, daß das passieren kann? Probieren Sie folgendes aus:

```
10 FOR n=1 TO 10 STEP .15

20 PRINT n

30 NEXT n
```

Wenn Sie sich über dieses Verhalten im Klaren sind, können Sie solche Probleme mit Schlüsselwörtern wie ROUND oder INT umgehen.

Vorbeugen ist besser als heilen. Und Sie können sich vor Dezimal-Fehlern schützen indem Sie dem CPC464 mitteilen, daß er die Zahl als Integer (Ganzzahl) behandeln soll. Dafür steht dann das Prozent-Zeichen. Sie merken schon, auch wenn nur 5+4 zusammengezählt werden, rechnet der CPC464 mit Gleitkomma-Zahlen wie folgt:

5.0000000 + 4.0000000 = 9.0000000

Sie und ich würden das nicht so machen. Wenn wir ganze Zahlen sehen, benutzen wir auch die Arithmetik für ganze Zahlen; bei Zahlen mit Dezimalkomma benutzen wir die Gleitkomma-Arithmetik. Dem CPC464 muß man dies aber genau mitteilen. Bei einem Abenteuer-Spiel brauchen wir normalerweise keine Zahlen mit Kommastellen, d.h. wir können mit Integer-Zahlen arbeiten um sicherzugehen, daß keine Fehler vorkommen. Manchmal wird das Programm dadurch auch etwas schneller.

Wahr oder falsch?

Als Sie das Listing angeschaut haben, mögen Sie sich vielleicht gedacht haben, daß einige der IF-Anweisungen nicht vollständig sind, z.B. in Zeile 1010:

```
IF INSTR(inv$,command$) THEN GOSUB 1730
```

Die INSTR-Funktion wird mit nichts verglichen, wie kann also eine Entscheidung getroffen werden? In Kapitel 7 wurde kurz auf den logischen Wert einer Bedingung hingewiesen. Sie hat den Wert −1 wenn die Bedingung wahr ist, und 0 wenn sie falsch ist. In der Zeile:

```
IF x=5 then...
```

ist die Bedingung x=5. Sie können herausfinden, ob die Bedingung wahr ist oder nicht wenn Sie eingeben:

```
PRINT x=5
```

Sie können das mit jeder Bedingung machen, da die Bedingung nur wahr oder falsch sein kann. Soweit ist es ja in Ordnung, aber was ist mit dieser Zeile:

```
IF x THEN...
```

Wenn „x" was ist? Der logische Wert von „x" hängt vom aktuellen Wert von „x" ab. Die Bedingung „x" ist wahr, wenn „x" verschieden von Null ist und falsch, wenn „x" Null ist. So einfach ist das. Das Schlüsselwort INSTR gibt eine Zahl zurück, die die Position des ersten Zeichens des gesuchten Strings im durchsuchten String angibt. Wenn der gesuchte String gefunden wurde, ist das Ergebnis Nicht-Null, ansonsten ist es Null.

Jetzt sollten Sie verstehen, wie der Befehl in Zeile 1010 (und alle ähnlichen) funktioniert. Diese Art zu programmieren kann für Sie oft von Vorteil sein.

DIE GESTALTUNG VON „ADVENTUR"

Stellen Sie sich vor, Sie müssten ein Abenteuer-Spiel von Anfang an gestalten (keine Sorge, das müssen Sie jetzt nicht machen). In Teil 1 dieses Selbstlern-BASIC (Kapitel 9) wurde Ihnen eine Programmiertechnik mit dem Namen „Programm-Entwicklungs-Sprache", kurz PDL (= Programming

Development Language) vorgestellt, damit Sie das logisch strukturierte Denken üben. Wir hatten damals das Beispiel eines Postboten, der die Post in seiner Straße auslieferte.

Im letzten Teil dieses Kapitels wird Ihnen jetzt gezeigt, wie

- ☐ PDL in ein ablauffähiges Programm umgesetzt wird
- ☐ das Programm modifiziert werden kann, um die Geschichte zu ändern.

Ein kleines PDL

Es gibt viele Wege, um ein PDL zu schreiben, und niemand sollte behaupten, daß seine Methode besser ist als die eines anderen. Abgesehen von der begrenzten Anzahl von Schlüsselwörtern liegt der Sinn darin, daß alle Routinen und Subroutinen in eine logische Programm-Struktur eingebaut werden – das ganze nennt man dann „strukturierte Programmierung".

Nachfolgend sehen Sie die PDL eines Abenteuer-Spiels. Wie Sie sehen, ist die PDL ziemlich kompakt, obwohl das Programm in BASIC rund 10 KB Speicherplatz beansprucht.

```
WENN ein Kommando eingegeben wurde
DANN pruefe ob das Kommando einen Sinn hat
   WENN JA:
     Kommando zur Bewegung-
       WENN die Richtung nicht erlaubt ist
       DANN gib eine Nachricht
       ANSONSTEN gehe auf die neue Position
     Kommando fuer eine Aktion
       WENN die Aktion nicht moeglich ist
       DANN gib eine Nachricht
       ANSONSTEN fuehre die Aktion aus
     Instruktionen (help, list, quit, look)
       fuehre die Instruktion aus
   WENN die Ende-Bedingung des Spiels erreicht
        wurde
   DANN begluekwuensche den Spieler
ANSONSTEN warte auf ein Kommando
```

Natürlich ist dies nur ein grober Überblick und gibt nicht die speziellen Subroutinen wieder, die notwendig sind, um alle detaillierten Aufgaben auszuführen. Aber Sie erhalten damit einen klaren Überblick über das Programm und können diesen Ablaufplan als „Aufhänger" für die anderen noch auszuführenden Subroutinen benutzen.

```
100 ' Arnold Abenteuer
110 '
120 ' DA 30/9/84  ESCON 27/2/85
130 '
140 ' Initialisierung
150 '
160 CLEAR
170 scrwidth=40 ' oder auf 80 aendern und MODE 2 ben
utzen
180 '
190 MODE scrwidth/40-0.25
200 BORDER 24
210 INK 0,3: INK 1,24
220 PAPER 0: PEN 1
230 LOCATE 12,5
240 PRINT"Arnold Abenteuer"
250 LOCATE 1,8
260 PRINT" In diesem Abenteuer versuchen Sie "
270 PRINT" einen ruhigen Nachmittag vor Ihrem "
280 PRINT" CPC464 zu verbringen. Aber jemand hat "
290 PRINT" all die Dinge, die Sie dazu brauchen, "
295 PRINT" im ganzen Haus verlegt."
300 PRINT
310 PRINT" Ihre Aufgabe ist es, alle Dinge zu "
320 PRINT" finden und das System ohne Stoerung "
330 PRINT" aufzubauen. Vorsicht vor der Katze!"
340 PRINT:PRINT
350 PRINT" Zum Starten irgendeine Taste druecken "
360 WHILE INKEY$="" : WEND
370 CLS
380 BORDER 23:INK 0,23
390 INK 1,1:INK 2,5:INK 3,8
400 '
410 ' Karte lesen
420 '
430 rooms=11 : items=11  'Anzahl der Zimmer und Gege
nstaende(-1)
440 fixed=7 ' die ersten (fixed-1) Gegenstaende sind
 beweglich, alle anderen nicht
450 held=0:heldmax=4  ' max. Anzahl der greifbaren O
bjekte
460 goes=0 ' Anzahl Schritte
470 '
480 '
490 DIM ex$(5) ' optional
500 FOR i%=0 TO 5
510 READ ex$(i%) : NEXT
520 '
```

```
530 ' Einlesen der Raeume
540 '
550 DIM loc$(rooms),dir$(rooms),dest$(rooms)
560 FOR i%=1 TO rooms
570 READ loc$(i%),dir$(i%),dest$(i%)
580 NEXT
590 '
600 ' Einlesen der Objekte
610 '
620 DIM object$(items),objloc(items)
630 FOR i%=0 TO items
640 READ object$(i%),objloc(i%)
650 NEXT
660 '
670 closed=-1:open=0:onn=-1:off=0
680 backdoor=closed:frontdoor=closed
690 switch=off:plugged=off
700 DIM sw$(1) : sw$(0)="AUS" : sw$(1)="EIN"
710 position=3 'Anfangs-Position
720 direction$="NSOWAH"
730 get$="NIMMHOLHEBFASS"
740 put$="LEGSTELLEHINTERLASSE"
750 pul$="SCHIEBEZIEHEDRUECKEDREHE"
760 inv$="ZEIGELISTE"
770 clo$="SCHLIESSEVERSPERRE"
780 ope$="OEFFNE"
790 loo$="SCHAUKONTROLLIEREPRUEFE"
800 qui$="BEENDEHALTEUNTERBRECHESTOP"
810 plu$="VERBINDESTECKE"
820 hel$="HELP"
830 nul$=CHR$(0) ' fuer Leerzeichen
840 '
850 GOSUB 2140 : REM umschauen (zum Start)
860 '
870 ' Haupt-Schleife
880 '
890 PRINT
900 INPUT ">",command$
910 IF command$="" THEN 890
920 IF LEFT$(command$,1)=" " THEN command$=MID$(comm
and$,2) : GOTO 910
930 command$=UPPER$(command$)
940 goes=goes+1
950 '
960 ' Ueberpruefen des Strings
970 '
980 ' ein Wort
990 '
1000 IF LEN(command$)=1 THEN GOSUB 1270 : GOTO 1190
:REM  bewegen
1010 IF INSTR(inv$,command$) THEN GOSUB 1730 : GOTO
1190 : REM zeigen
1020 IF INSTR(loo$,command$) THEN GOSUB 2120 : GOTO
1190 : REM schauen
```

```
1030 IF INSTR(qui$,command$) THEN GOSUB 2070 : GOTO
1190 : REM aufgeben
1040 IF INSTR(hel$,command$) THEN GOSUB 2380 : GOTO
1190 : REM helfen
1050 '
1060 ' zwei Woerter
1070 '
1080 d%=INSTR(command$," ")
1090 IF d%=0 THEN PRINT" Das verstehe ich nicht":GOT
O 1210
1100 noun$=MID$(command$,d%+1) : command$=LEFT$(comm
and$,d%-1)
1110 IF INSTR(noun$," ") THEN P$="Bitte immer nur zw
ei Woerter zugleich":GOSUB 2680: GOTO 890
1120 IF INSTR(get$,command$) THEN GOSUB 1380 : GOTO
1190 : REM nehmen
1130 IF INSTR(put$,command$) THEN GOSUB 1520 : GOTO
1190 : REM hinlegen
1140 IF INSTR(pul$,command$) THEN GOSUB 1650 : GOTO
1190 : REM schieben
1150 IF INSTR(clo$,command$) THEN GOSUB 1840 : GOTO
1190 : REM schliessen
1160 IF INSTR(ope$,command$) THEN GOSUB 1950 : GOTO
1190 : REM oeffnen
1170 IF INSTR(plu$,command$) THEN GOSUB 2270 : GOTO
1190 : REM versperren
1180 PRINT "Ich weiss nicht wie ich das machen soll"
:
1190 '
1200 GOSUB 2470 ' ist das Abenteuer fertig
1210 '
1220 GOTO 890
1230 '
1240 '
1250 ' bewegen
1260 '
1270 d%=INSTR(direction$,command$)
1280 IF d%=0 THEN P$="Ich weiss nicht was Sie meinen
":GOSUB 2680: GOTO 1340
1290 d%=INSTR(dir$(position),command$)
1300 IF d%=0 THEN P$="Diese Richtung koennen Sie nic
ht gehen":GOSUB 2680: GOTO 1340
1310 IF (position=4 OR (position=3 AND d%=1)) AND ba
ckdoor=closed THEN P$="Die hintere Tuer ist geschlos
sen":GOSUB 2680: GOTO 1340
1320 IF (position=1 OR (position=2 AND d%=2)) AND fr
ontdoor=closed THEN P$="Die vordere Tuer ist geschlo
ssen":GOSUB 2680: GOTO 1340
1330 position=ASC( MID$( dest$(position),d%,1 ) )-64
 : GOSUB 2140
1340 command$=nul$ : RETURN
1350 '
1360 ' nehmen
```

```
1370 '
1380 o%=0
1390 FOR i%=0 TO items
1400 IF INSTR(UPPER$(object$(i%)),noun$) THEN o%=o%+
1 : GOSUB 1440 : GOTO 1430
1410 NEXT
1420 IF o%=0 THEN P$="Ich weiss nicht was das ist":G
OSUB 2680
1430 command$=nul$ : RETURN
1440 IF i%>=fixed THEN P$=object$(i%)+" kann nicht m
itgenommen werden" :GOSUB 2680: GOTO 1480
1450 IF held=heldmax THEN p$="Ihre Haende sind voll"
:GOSUB 2680:GOTO 1480
1460 IF objloc(i%)=position THEN P$= "Hab ich aufgeh
oben":GOSUB 2680:held=held+1:objloc(i%)=0 : GOTO 148
0
1470 IF objloc(i%)=0 THEN P$="Sie haben diesen Gegen
stand bereits":GOSUB 2680 ELSE P$="Dieser Gegenstand
 ist nicht hier":GOSUB 2680
1480 RETURN
1490 '
1500 ' hinstellen
1510 '
1520 o%=0
1530 FOR i%=0 TO items
1540 ff%=0
1550 IF INSTR(UPPER$(object$(i%)),noun$) THEN o%=o%+
1 : GOSUB 1600
1560 IF o%>0 THEN i%=items
1570 NEXT
1580 IF o%=0 THEN P$="Ich weiss nicht was das ist":G
OSUB 2680
1590 command$=nul$ : RETURN
1600 IF objloc(i%)=0 THEN P$=+object$(i%)+" haben Si
e hingelegt":GOSUB 2680:held=held-1:objloc(i%)=posit
ion ELSE P$=object$(i%)+" haben Sie nicht dabei":GOS
UB 2680
1610 RETURN
1620 '
1630 ' drehen
1640 '
1650 d%=INSTR("SCHALTERTUER",noun$)
1660 IF d%=0 THEN P$="Kein Punkt":GOSUB 2680: GOTO 1
690
1670 IF d%=7 THEN ' Weiter zur Routine Tuer oeffnen
1680 IF position=2 THEN switch=-(1+switch): P$="Der
Griff rastet ein in die "+sw$(ABS(switch))+" Positio
n":GOSUB 2680 ELSE P$="Ich sehe keinen Griff":GOSUB
2680
1690 command$=nul$ : RETURN
1700 '
1710 ' aufzeigen
```

```
1720 '
1730 temp=0
1740 P$="Sie tragen gerade ":GOSUB 2680
1750 o%=0
1760 FOR i%=0 TO items
1770 IF objloc(i%)=temp THEN P$=" "+object$(i%)+" ":
GOSUB 2680:o%=o%+1
1780 NEXT
1790 IF o%=0 THEN P$="nichts ":GOSUB 2680
1800 command$=nul$ : RETURN
1810 '
1820 ' schliessen
1830 '
1840 d%=INSTR("TUER",noun$)
1850 IF d%<>1 THEN P$="Kann ich nicht schliessen":GO
SUB 2680: GOTO 1910
1860 IF position>4 THEN P$="Welche Tuer?":GOSUB 2680
: GOTO 1910
1870 IF (position=1 OR position=2) AND frontdoor=clo
sed THEN P$="Die vordere Tuer ist bereits geschlosse
n":GOSUB 2680: GOTO 1910
1880 IF (position=3 OR position=4) AND backdoor=clos
ed THEN P$="Die hintere Tuer ist bereits geschlossen
":GOSUB 2680: GOTO 1910
1890 IF (position=1 OR position=2) THEN P$="Sie habe
n die Vorder-Tuer geschlossen":GOSUB 2680: frontdoor
=closed
1900 IF (position=3 OR position=4) THEN P$="Sie habe
n die Rueck-Tuer geschlossen":GOSUB 2680: backdoor=c
losed
1910 command$=nul$ : RETURN
1920 '
1930 ' oeffnen
1940 '
1950 d%=INSTR("TUER",noun$)
1960 IF d%<>1 THEN P$="Kann ich nicht oeffnen":GOSUB
 2680: GOTO 2030
1970 IF position>4 THEN P$="Welche Tuer?":GOSUB 2680
: GOTO 2030
1980 IF (position=1 OR position=2) AND frontdoor=ope
n THEN P$="Die Vorder-Tuer ist bereits offen":GOSUB
2680: GOTO 2030
1990 IF (position=3 OR position=4) AND backdoor=open
 THEN P$="Die Rueck-Tuer ist bereits geschlossen":GO
SUB 2680: GOTO 2030
2000 IF (position=1 AND frontdoor=closed) OR (positi
on=4 AND backdoor=closed) THEN P$="Pech gehabt. Sie
haben sich selbst ausgeschlossen":GOSUB 2680:GOTO 25
80
2010 IF position=2 THEN P$="Sie haben die Vorder-Tue
r geoeffnet":GOSUB 2680: frontdoor=open
2020 IF position=3 THEN P$="Sie haben die Rueck-Tuer
 geoeffnet" :GOSUB 2680: backdoor=open
```

```
2030 command$=nul$:RETURN
2040 '
2050 ' aufgeben
2060 '
2070 P$="Wollen Sie wirklich aufhoeren (J/N) ":GOSUB
 2680
2080 INPUT "", command$
2090 IF UPPER$(command$)="J" THEN END
2100 command$=nul$ : RETURN
2110 '
2120 ' schauen
2130 '
2140 P$="Derzeitiger Aufenthaltsort: "+loc$(position
):GOSUB 2680
2150 P$=". Ausgaenge fuehren nach ":GOSUB 2680
2160 FOR i%=1 TO LEN(dir$(position))
2170 temp$=MID$(dir$(position),i%,1)
2180 P$=ex$(INSTR(direction$,temp$)-1)+", ":GOSUB 26
80
2190 NEXT
2200 P$="und Sie sehen: ":GOSUB 2680
2210 temp=position
2220 GOSUB 1750
2230 command$=nul$ : RETURN
2240 '
2250 ' verschliessen
2260 '
2270 IF position=5 THEN P$="Ihre Familie schaut Fern
sehen und haelt Sie auf.":GOSUB 2680:GOTO 2340
2280 IF objloc(1)<>10 THEN p$="Der Monitor ist nicht
 hier":GOSUB 2680:GOTO 2340
2290 IF objloc(3)<>10 THEN p$="Der Computer ist nich
t hier":GOSUB 2680:GOTO 2340
2300 IF position<>10 THEN P$="In diesem Raum befinde
t sich keine Steckdose.":GOSUB 2680: GOTO 2340
2310 IF INSTR("CTM640MONITORSCHNEIDERCPC464COMPUTER"
,noun$)=0 THEN P$="Ich kann das nicht anstecken.":GO
SUB 2680: GOTO 2340
2320 plugged=onn
2330 IF switch=off THEN P$="Nichts passiert.":GOSUB
2680 ELSE P$="Ihr Computer ist betriebsbereit.":GOSU
B 2680
2340 command$=nul$ : RETURN
2350 '
2360 ' Hilfe
2370 '
2380 RESTORE 3190
2390 p$="Die Kommandos heissen :" : GOSUB 2680
2400 FOR h1=0 TO 25
2410 READ p$:p$=" "+p$:GOSUB 2680
2420 NEXT
2430 command$=nul$ : RETURN
2440 '
2450 ' "Ende des Spiels?"
```

```
2460 '
2470 fr=10 'letztes Zimmer
2480 IF position<>fr THEN RETURN
2490 ready=(objloc(1)=fr) AND (objloc(3)=fr) AND (ob
jloc(4)=fr) AND (objloc(5)=fr) AND (objloc(6)=fr)
2500 ready=ready AND switch AND plugged
2510 catout=((objloc(0)=4) OR (objloc(0)=1)) AND bac
kdoor=closed AND frontdoor=closed
2520 IF NOT catout AND objloc(3)=fr AND objloc(0)<>0
 THEN P$=" Die Katze springt auf das Keyboard":GOSUB
 2680: objloc(0)=fr
2530 ready=ready AND catout
2540 IF NOT ready THEN RETURN
2550 PRINT
2560 p$="Gut gemacht. Sie haben das Abenteuer beende
t in":GOSUB 2680
2570 p$=STR$(goes)+" Schritten. Jetzt koennen Sie ei
nen ruhigen Nachmittag  mit Ihrem Computer geniessen
":GOSUB 2680
2580 PRINT:PRINT
2590 P$="Wollen Sie nochmal spielen? (J/N) ":GOSUB 2
680
2600 INPUT "", command$
2610 IF UPPER$(command$)="J" THEN RUN
2620 END
2630 '
2640 ' Ausgabe-Routine
2650 ' P$=ist der zu druckende Text
2660 ' Zuruecksetzen des Cursors in die folgende Pos
ition
2670 ' dafuer wird vorher CRLF benoetigt >
2680 p$=p$+" "
2690 ln=LEN(p$):xp=POS(#0)
2700 IF INSTR(p$," ")=0 AND (ln+xp)>(scrwidth-2) THE
N PRINT:xp=0
2710 FOR pr%=1 TO ln-1
2720 sp=INSTR(pr%,p$," ")
2730 IF sp+xp >(scrwidth-2) THEN PRINT :xp=xp-scrwid
th
2740 PRINT MID$(p$,pr%,1);
2750 NEXT
2760 RETURN
2770 '
2780 '
2790 ' Richtungen
2800 '
2810 DATA Norden,Sueden,Osten,Westen,Ab,Hinauf
2820 '
2830 ' Raeume Nr:(rooms)
2840 ' Name,Richtungen zum Verlassen des Raumes,Spez
ifizierung der angrenzenden Raeume (A=1 B=2...)
2850 '
2860 DATA Vorgarten,N,B
2870 DATA Gang,NSOH,CAEH
2880 DATA Kueche,NS,DB
```

```
2890 DATA Hinter-Garten,S,C
2900 DATA Wohnzimmer,W,B
2910 DATA Badezimmer,O,H
2920 DATA Vorderes Schlafzimmer,N,H
2930 DATA Oberer Gang,NSOWHA,IGJFKB
2940 DATA Hinteres Schlafzimmer,S,H
2950 DATA Abstellkammer,W,H
2960 DATA Dachraum,A,H
2970 '
2980 ' Objekte  Nr:(items+1)
2990 ' Name,Ausgangs-Pos ( <=rooms)
3000 '
3010 DATA schwarze Katze:,2
3020 DATA CTM640 Monitor:,3
3030 DATA Fernseher:,5
3040 DATA Schneider CPC464 Computer:,7
3050 DATA Arbeitstisch:,11
3060 DATA Kuechen-Stuhl:,3
3070 DATA Computer Handbuch:,9
3080 '
3090 ' unbewegliche Objekte
3100 '
3110 DATA roter Schalter:,2
3120 DATA Steckdose:,10
3130 DATA Rosen-Beet:,1
3140 DATA rostiger Muelleimer:,4
3150 DATA unbezahlbares altes Gemaelde:,11

3160 '
3170 ' Hilfs-Kommandos
3180 '
3190 DATA BEENDE,DREHE,DRUECKE,FASS,HALTE,HEB,HELP
3200 DATA HINTERLASSE,HOL,KONTROLLIERE,LEG,LISTE,NIM
M,OEFFNE
3210 DATA PRUEFE,SCHAU,SCHIEBE,SCHLIESSE,STECKE,STOP
3220 DATA UNTERBRECHE,VERBINDE,VERSPERRE,ZEIGE,ZIEHE
,HINTERLASSE
```

Umsetzung in BASIC

Wenn Sie das Listing dieses Programmes durchgesehen und verstanden haben, können Sie diesen Teil überspringen, ansonsten lesen Sie weiter. Das Programm startet mit dem Ausdruck der Titelseite und wartet dann bei Zeile 360 auf Ihre Anweisung. Durch Drücken irgendeiner Taste beginnt das eigentliche Programm.

Die Zeilen 430 – 460 legen einige Variablen fest, deren Zweck später noch klar werden wird. Die Zeilen 490 – 510 lesen die ersten sechs Ausdrücke aus der Data-Liste (Zeile 2810) in den Speicherbereich „ex$"ein. Die Zeilen 550 – 580 lesen die nächsten 33 Data-Statements (3*11, Zeilen 2860 –

2960) in die Speicherbereiche „loc$", „dir $" und „dest$". In den Zeilen 620–650 werden die nächsten 24 Data-Statements in die Speicherbereiche „object$" und „objloc" eingelesen. Ab Zeile 670 werden noch einige Variablen auf ihren Anfangswert gesetzt, einschließlich 11 merkwürdiger String-Variablen. Jede von diesen ist aus einer Anzahl von Wörtern geformt, die alle den gleichen Sinn haben und in der Hauptschleife des Prorgramms dem CPC464 helfen herauszufinden, was Ihr Kommando bedeutet.

Machen wir an dieser Stelle einen kurzen Rückblick auf die „Speicherbereiche", die wir bereits in Kapitel 2 behandelt haben. Speicherbereiche ermöglichen Ihnen einen relativ leichten Zugriff auf bestimmte „Listen von Dingen". Um diese darzustellen lassen Sie das Programm mit RUN laufen und drücken Sie auf ESC, wenn das Prompt-Kommando erscheint.

Geben Sie dann die folgenden Zeilen ein:

```
5000 FOR n=0 TO 11

5010 PRINT n;" ";loc$(n)

5020 NEXT n
```

Geben Sie jetzt nicht RUN ein, da sonst alle Variablen verloren sind. Geben Sie stattdessen ein:

```
CLS:GOTO 5000
```

Sie werden dann eine Liste aller Data-Ausdrücke (Elemente) von Speicherbereich „loc$" sehen. Versuchen Sie das gleiche noch mit anderen Speicherbereichen. Alle sind eindimensionale Speicherbereiche und bilden die Basis des vorliegenden Programmes. Mit Hilfe dieser Tabellen ist der CPC464 immer auf dem laufenden, er weiß, was sich wo befindet und in welchem Zustand es ist.

Schauen wir uns wieder das Listing an. Meldungen auf dem Bildschirm werden durch die Subroutine ab Zeile 2680 ausgegeben. Was ausgegeben wird, wird durch die Variable „p$" bestimmt. In Zeile 2140 finden wir, daß „p$" aus zwei zusammengehängten Strings gebildet wird. Der erste String ist ein Standard-String mit dem Inhalt „derzeitiger Standort", der zweite hängt von Ihrem jeweiligen Aufenthaltsort ab. Ihre ursprüngliche Position ist in der Küche, da die variable Position in Zeile 710 auf drei gesetzt wurde und Element drei des Speicherbereiches „loc$" gleich „Küche" ist.

Die Subroutinen bei 2150 und 2200 fassen „p$" in der gleichen Art und Weise zusammen, um die Ausgänge beziehungsweise den Standort auszudrücken. Sobald dieser Vorgang erledigt ist, liegt es an Ihnen, Ihr Kommando einzugeben.

Wenn Sie dies tun, wird Ihrer Eingabe jeweils die Variable „kommando$" zugewiesen. Diese wird dann geprüft und analysiert bevor irgendeine In-

struktion ausgeführt werden kann. Diese „Sinngebung" eines Strings nennt man „Analyse"; daher die Anmerkung des Programmierers in Zeile 960. Anschließend wird dann die Bedeutung der INSTR-Funktion klar. Nach Zeile 1000 wird Ihre Eingabe auf einen erkennbaren String hin untersucht. Wenn ein String erkannt wird (d.h. wenn er Bestandteil einer dieser kuriosen Variablen aus den Zeilen 720 – 820 ist), wird die Anweisung ausgeführt; anderenfalls bekommen Sie eine Fehlermeldung. Für jede Aktion gibt es eine eigene Subroutine, die im Programm-Listing deutlich abgesetzt ist.

Erst wenn Sie alle betreffenden Objekte in den entsprechenden Räumen gefunden und noch einige andere Dinge erfüllt haben, endet das Spiel. Wenn Sie die Routine ab Zeile 2470 anschauen, können Sie herausfinden, was Sie noch alles tun müssen, um das Programm zu beenden. (Wenn Sie die Bedingungen lieber durch Spielen herausfinden wollen, überspringen Sie die nächsten drei Absätze!)

Beginnen Sie bei Zeile 2470. Das letzte Zimmer ist Zimmer 10, d.h. es ist das zehnte Element des Speicherbereichs „loc$". Dies ist die Abstellkammer. Bei Zeile 2480 wird abgefragt, ob Ihre momentane Position Zimmer 10 ist, d.h. ob die variable Position = 10. Wenn dies nicht der Fall ist, geht das Spiel weiter.

Wenn es das richtige Zimmer ist, gehen wir weiter auf Zeile 2490, die auf den ersten Blick recht kompliziert aussieht, aber in Wirklichkeit ganz einfach ist. Der Wert der Variablen „fertig" hängt von den Werten der fünf Bedingungen in dieser Zeile ab. Wenn „fertig" wahr sein soll, müssen alle fünf Bedingungen wahr sein. Diese fünf Bedingungen sind einfach Elemente des Speicherbereiches „objloc", die Sie im entsprechenden Zimmer aufsammeln und zusammenstellen müssen. Wenn Sie auf die Liste mit den Data-Angaben (Zeilen 3010 – 3070) sehen, werden Sie merken, daß die Elemente 1,3,4,5 und 6 den CTM640 Monitor, den Schneider CPC464 Computer, den kleinen Arbeitstisch, den Küchenstuhl und das Computer-Buch darstellen. Die Nummer nach jedem Ausdruck in dieser Aufstellung ist die Nummer des Zimmers, in dem weitergemacht werden soll. Diese Nummer muß zum Schluß gleich 10 sein.

Die Variable „fertig" hängt nicht nur von den richtigen Gegenständen ab. Sie können in Zeile 2500 sehen, daß „fertig" auch davon abhängt, ob die Variablen „schalter" und „sperre" wahr sind. Diese werden in Zeile 1650 bzw. 2270 gesetzt. Und schließlich sehen Sie in Zeile 2530, daß „fertig" auch von „catout" abhängt, das in Zeile 2510 gesetzt wird. Wenn „catout" wahr sein soll, muß das Element 0 des Speicherbereiches „loc$" (die Katze) entweder in Stellung 1 oder 4 sein (vorderer oder hinterer Garten) und beide Türen müssen geschlossen sein. Wenn dies nicht der Fall ist, und Sie die Katze nicht halten (Zeile 2520), dann springt die Katze auf das Keyboard.

Um es nochmal zusammenzufassen: Wenn Sie den Monitor, den CPC464, das Buch, den Tisch und den Stuhl im Abstellraum haben, den großen roten Schalter umgedreht, den Rechner angesteckt, die Katze in den Garten gesetzt sowie Vorder- und Hintertüre geschlossen haben, dann ist das Spiel aus.

Das ist der Inhalt dieses Abenteuer-Spieles. Wenn Sie Ihren Ort verändern, Dinge mitnehmen oder den Status bestimmter Dinge verändern (z.B.Schließen einer Tür), dann werden die Speicherbereiche und anderen Variablen jeweils geändert bzw. auf den neuesten Stand gesetzt. Wenn Sie ein paar Seiten zurückblättern, werden Sie sehen, daß unser PDL genau nach diesem System strukturiert war.

IHR HAUS KÖNNTE EINE BURG SEIN

Um Ihnen dieses Abenteuer-Programm noch verständlicher zu machen, bauen wir jetzt einige Modifikationen in dieses Programm ein. Ein Raum wird noch hinzugefügt, ein paar Gegenstände mehr und ein weiterer Gegenstand, den Sie in den Abstellraum bringen müssen. Sind Sie daran interessiert? Dann lesen Sie weiter . . . .

Wie wäre es mit einer Garage im Osten des Vorgartens? Das erste, was wir tun müssen, ist die Variable „zimmer" in Zeile 430 von 11 auf 12 erhöhen. Damit müssen wir später auch die Dimensionierung der Speicherbereiche ändern. Als nächstes fügen Sie ein zwölftes DATA-Statement zu der Auflistung ab Zeile 2860 hinzu. Jede Zeile besteht aus drei Daten. Zuerst der Name des Raumes – Garage. Der zweite Teil ist die Richtung, in der man den Raum verlassen kann. In diesem Fall nach Westen in den Vorgarten – W. Zum Schluß noch einen Code-Buchstaben für den bzw. die angrenzenden Orte. Die Orte sind von A – K für die Elemente 1 bis 11 des Speicherbereiches „loc$" codiert. Der Vorgarten ist Element 1, also Buchstabe A. Geben Sie folgende Zeile ein:

```
2970 DATA Garage,W,A
```

Jetzt müssen wir noch die Daten für den Vorgarten ändern. Legen Sie die Zeile 2860 folgendermaßen fest:

```
2860 DATA Vorgarten,NO,BL
```

Das O wurde hinzgefügt, da Sie jetzt auch den Garten nach Osten verlassen können, das L wurde angehängt da die Garage als zwölftes Element des Speicherbereiches den Code-Buchstaben L darstellt (zwölfter Buchstabe des Alphabets).

Stellen wir jetzt noch etwas in die Garage. Ein Auto wäre eine gute Idee! Zuerst müssen Sie die Variable „items" in Zeile 430 von 11 auf 12 erhöhen, dann fügen Sie eine zwölfte DATA-Zeile in die Objekt-Liste ab Zeile 3010 hinzu. Beachten Sie, daß einige Gegenstände beweglich sind, andere wiederum nicht. Die Anzahl der beweglichen Gegenstände ist durch die Variable „fixed" in Zeile 440 festgelegt. Momentan hat sie den Wert 7, was bedeutet, daß nur die ersten 7 Objekte in der Data-Anweisung bewegt werden können. Da wir das Auto nicht bewegen wollen, brauchen wir „fixed" nicht ändern. Außerdem müssen wir sichergehen, daß unsere neue DATA-Zeile nicht zu den ersten sieben Anweisungen gehört. Geben Sie folgende Zeile ein.

```
3160 DATA Gebrauchtwagen,12
```

Die 12 gibt die Nummer des Ortes an (Garage) – wenn Sie das Programm ablaufen lassen (RUN), steht ein Auto in der Garage!

Stellen wir jetzt noch etwas in die Garage, das Sie mitnehmen können – ein Fahrrad. Ändern Sie nochmals die Variable „items" (Zeile 430), diesmal auf 13. Dann ändern Sie Variable „fixed" von 7 auf 8.

Unsere DATA-Zeile muß nun eine der ersten acht sein, so geben wir also ein:

```
3080 DATA neues Fahrrad,12
```

Lassen Sie das Programm jetzt ablaufen. Sie sollten jetzt das Fahrrad bewegen können, aber nicht das Auto.

Zum Schluß können wir noch die Ende-Bedingung so ändern, daß Sie noch eine Datacassette benötigen, bevor das Spiel aus ist.

Erhöhen Sie zuerst die Variablen „items" und „fixed" wieder jeweils um eins und fügen Sie dann jeweils noch folgende Zeile hinzu:

```
3075 Datacassette,9
```

Die 9 ist die Nummer des hinteren Schlafzimmers, das neunte Element des Speichers „loc$". Um die Datacassette zu einem der Gegenstände zu machen, die Sie aufsammeln sollen, müssen Sie es zu der Zeile hinzufügen, die prüft, ob die Variable „fertig" wahr ist. Fügen Sie also am Ende der Zeile 2490 hinzu:

```
AND (objloc(7)=fr)
```

Die Datacassette ist das siebte Element im Speicherbereich und muß jetzt im letzten Raum sein. Lassen Sie das Programm laufen, um sicherzugehen, daß dies tatsächlich der Fall ist!

Jetzt sollten Sie wirklich wissen, wie dieses Programm abläuft. Warum sollten wir dann nicht noch mehr Räume, einen Keller oder einen Schuppen hinzufügen oder warum nicht gleich unser Haus in ein Schloß mit Bankettsällen, Kerkern, Türmen, Prinzessinnen, die gerettet werden müssen und Drachen, gegen die es zu kämpfen gilt, umwandeln? Das ist einfacher zu machen, als Sie denken. Nehmen Sie sich Zeit, denken Sie logisch, Schritt für Schritt. Gerade wenn Sie sich nicht sicher sind, probieren Sie es erst recht. Das ist der beste Weg zum Lernerfolg. Lernen ist auch eine Art Abenteuer!

TEST

Wenn Sie mit dem ABENTEUER-Spiel Ihre Sinne getestet haben, ist es Zeit, daß Sie auch Ihr Gedächtnis mit SAT11 überprüfen.

Kapitel 12
WAS NUN?

DER TOLLE SCHNEIDER

Vor langer, langer Zeit (in Teil 1 dieses Kurses) gaben wir Ihnen einen Hinweis, daß das numerische Tastenfeld auf der rechten Seite Ihres CPC464 auch noch für andere Dinge benutzt werden kann, als nur zur bequemen Eingabe von Ziffern. Sicher benutzen Sie auch schon die CTRL-ENTER Funktion als Kurzform zum Laden und Starten von Programmen; was Sie jetzt lernen, ist die Definition Ihrer eigenen zeitsparenden Funktionstasten, mit denen Sie die Eingabe oder den Ablauf von Programmen erheblich beschleunigen können.

Eines der ersten Dinge, die Sie normalerweise gleich nach dem Einschalten des Rechners festlegen, ist die Definition der Zeichen, die beim Drücken der separaten numerischen Tastenfelder erscheinen sollen. Sie können diese jederzeit ändern. Drücken Sie zuerst die Null-Taste auf dem separaten Tastenfeld. Sie sehen eine Null auf dem Bildschirm. Geben Sie jetzt mal folgendes ein:

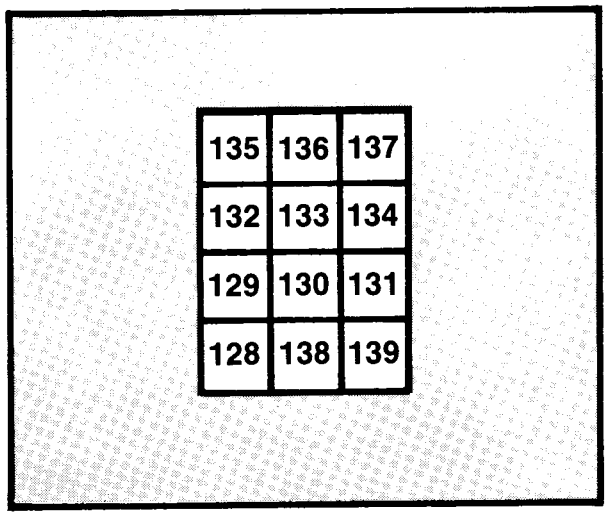
```
KEY 128,"Hallo"
```

Welch eine Überraschung, wenn Sie jetzt die Null-Taste wieder drücken. Denn Sie sehen jetzt statt einer Null „Hallo" auf dem Bildschirm. Unser neues Schlüsselwort KEY ermöglicht es uns, irgendeinen beliebigen String auf eine der zwölf Tasten des separaten Tastenfeldes zu legen. Die Struktur des Kommandos lautet:

KEY

```
KEY <Tastaturnummer>,<Textausdruck>
```

Der Ausdruck <Tastaturnummer> ist ein ganzzahliger Ausdruck zwischen 128 und 140, der die aktuelle Tastaturnummer laut nachfolgendem Diagramm angibt.



Sie merken sicher, daß die Numerierung nur bis 139 und nicht bis 140 reicht. Die dreizehnte Taste ist unser alter Freund ENTER, der in Verbindung mit CTRL bekanntermaßen folgendes ergibt:

```
run"
```

Der Textausdruck kann dann z.B. ein direkt auszuführendes Kommando während der Programmeingabe sein. Geben Sie folgendes ein:

```
KEY 129,"mode 2"
```

Wenn Sie jetzt die „1" drücken, sehen Sie:

```
mode 2
```

auf dem Monitor. Wenn Sie darauf die ENTER-Taste drücken, wird das Kommando genauso ausgeführt, als wenn Sie es über die normale Tastatur eingetippt hätten.

Das ist aber noch nicht alles! Sie können auch eingeben:

```
KEY 129,"mode 2"+Chr$(13)
```

Erinnern Sie sich an diese Nummer? Es ist der ASCII-Code für CR, oder in gutem Englisch, Carriage Return, Wagenrücklauf (Carriage bedeutet Wagen und bezieht sich auf den Wagen einer Schreibmaschine, der mit Hilfe dieses Steuerzeichens bewegt wird). Wie Sie vielleicht mitbekommen haben, ist das exakt das gleiche, als wenn Sie die ENTER-Taste drücken. Wenn Sie also die „1" auf Ihrem separaten Tastenfeld drücken, wird das Kommando sofort ausgeführt und MODE 2 eingestellt. Versuchen Sie es selbst.

Hier ist ein Beispiel für ein kurzes Programm, das Sie möglicherweise auf eine spezielle „Programmentwicklungscassette" aufnehmen könnten. Sie

können auch Ihre Lieblingseinstellung für die Farbe von BORDER, PEN, PAPER und INK darin aufnehmen:

```
10 key 128,"mode 0"+chr$(13)

20 key 129,"mode 1"+chr$(13)

30 key 130,"mode 2"+chr$(13)

40 key 131,"list"

50 key 132,"list"+chr$(13)

60 key 133,"save""

70 key 134,"renum"

80 key 135,"auto"
```

Wenn Sie ein langes Programm schreiben bzw. entwickeln und diese Funktionen auf einfachen Tasten haben, ersparen Sie sich viel Langeweile. Sie werden herausfinden, daß verschiedene Programmtypen auch verschiedene Tastenbelegungen brauchen. Wenn in einem Programm z.B. viele Bildschirmmeldungen vorkommen, ist es am besten, Sie legen sich „PRINT" auf eine der Funktionstasten, um dieses Wort nicht andauernd eintippen zu müssen.

SCHNELLES LADEN

Eine weitere Stärke des CPC464 ist die Geschwindigkeit, mit der Daten auf die Datacassette gespeichert werden. Um maximale Zuverlässigkeit zu gewährleisten sind beide Datacassetten für diesen Kurs mit der geringeren Geschwindigkeit aufgezeichnet (1000 Baud für die technisch Interessierten). Es ist aber möglich, die Aufzeichnungsgeschwindigkeit zu verdoppeln, wenn man folgendes eingibt:

```
SPEED WRITE 1
```

Jedes folgende SAVE-Kommando wird dann mit dieser neuen Geschwindigkeit ausgeführt (2000 Baud).

Wenn Programme, die mit einer Geschwindigkeit von 2000 Baud aufgezeichnet wurden, eingelesen werden, wird automatisch die richtige Lesegeschwindigkeit gewählt. Natürlich können Sie damit Programme mit verschiedenen Aufzeichnungsgeschwindigkeiten auf ein und derselben Cassette abspeichern. Wenn Sie hochwertige Cassetten nehmen und den Lesekopf stets sauber halten, werden Sie ziemlich sicher keine Probleme beim Lesen schnell aufgezeichneter Daten haben.

Um die ursprüngliche Geschwindigkeit wieder einzustellen, geben Sie ein:

SPEED WRITE

```
SPEED WRITE 0
```

HARD COPY

„Hard Copy" bedeutet in der Computer-Umgangssprache einen Ausdruck von Daten oder des Programm-Listings auf Papier. Dieser Abschnitt ist vor allem für die Besitzer des Schneider Matrixdruckers NLQ401 gedacht.

Wenn Sie Ihren Drucker an der dafür vorgesehenen Stelle an der Rückseite des CPC464 angeschlossen haben, schalten Sie Ihren Rechner ein und geben den Befehl PRINT oder LIST ein – nichts passiert. Das liegt daran, daß alle Eingabe- und Ausgabedaten in „Kanälen" behandelt werden, und der Druckerkanal nicht bestimmt wurde. Es gibt 10 solcher Kanäle mit den Nummern 0 – 9, die wir uns aber erst in Teil 3 dieses Lehrbuches genauer anschauen wollen. Momentan wollen wir uns auf die Kanäle 0 und 8 beschränken.

Der Kanal ist ein optionaler Parameter im PRINT- und LIST-Kommando, dessen Standardwert 0 ist. Zum Beispiel:

```
LIST 2000-
```

gibt auf dem Bildschirm einen Ausdruck aller Programmzeilen ab Nummer 2000 aus, da 0 der Bildschirmkanal ist. Wenn Sie jedoch eingeben:

LIST#8

```
LIST 2000-,#8
```

dann werden alle Zeilen ab 2000 auf dem angeschlossenen Drucker ausgegeben.

Genauso können Sie eingeben:

PRINT #8

```
PRINT #8,"Dies ist ein Beispiel fuer Druck
  ausgabe"
```

Wenn Sie jetzt zu dem Programm-Listing von KALKUL in Kapitel 5 zurückblättern, scheint es, als hätte die Subroutine zum Ausdruck der Kalkulation die unbenutzte Variable „#str". Wenn Sie auf die Print-Anweisungen in dieser Subroutine schauen, werden Sie sehen, daß sie aber tatsächlich die Kanal-Nummer für das PRINT-Kommando darstellt. Da der Kanal aber bisher nicht bestimmt wurde, ist es Kanal Null – also der Bildschirm. Setzen Sie also „str" auf 8 und Sie bekommen Ihre Kalkulation ausgedruckt.

DATEIVERWALTUNG

Unser letztes Programm auf Datencassette A ist möglicherweise das beste aus unserer Programm-Reihe.

Wenn Sie Ihre Schallplattensammlung, Ihre Lieblingskochrezepte oder die Namen und Adressen Ihrer Freunde archivieren wollen, ist das genau das richtige Programm für Sie. Es ist wie ein elektronisch verwaltetes Regal, für das Sie eine Menge verschiedener Aufkleber zur gleichen Zeit benutzen können. Wenn Sie Ihre „Datenbank" erst einmal aufgebaut haben, können Sie unter verschiedenen Gesichtspunkten zugreifen. Auf Suppen z.B., oder Reggae oder was auch immer.

Nach dem Programm DATABASE finden Sie ein Beispiel für eine entsprechende Datei. Wenn im Hauptmenü gefragt wird, welche Option Sie wollen, geben Sie „R" ein. Dann wird nach einem Dateinamen gefragt. Geben Sei ein

```
Laender
```

und lernen Sie ein bischen Geographie.

TEST

Dieser abschließende Test, SAT12, behandelt noch einmal alle Themen dieses Kurses. Wie wir bereits sagten, ist dies kein Examen, sondern nur eine Hilfe für Sie, damit Sie bestimmte Dinge, die Sie nicht verstanden haben, wiederholen können, bevor Sie die nächsthöhere Stufe der Programmierung in Angriff nehmen.

Die nächsthöhere Stufe ist Teil drei dieses Kurses. Von jetzt ab werden Sie sich in so fremdartige Dinge wie Binär-Arithmetik, Hexadezimal-Zahlen und Unterbrechungen einarbeiten müssen. Viel Glück!

VERZEICHNIS DER PROGRAMME

Datacassette A enthält die folgenden Programme in der gleichen Reihenfolge, wie sie auch in diesem Buch angesprochen werden. Datacassette B enthält die Selbsttests (SAT), die der Leser nach jedem Kapitel (außer nach Kapitel 5) durcharbeiten sollte.

Kapitel 1

SCHAF

Kapitel 2

CHATEAU
STADT

Kapitel 3

SINCOS
PIECHART

Kapitel 4

DIGITALC
ALARM

Kapitel 5

KALKUL

Kapitel 6

FLUGPLAN

Kapitel 7

WORTPUZL

Kapitel 8

BLITZ

Kapitel 9

ZOUNDS
ZAPPOW2

Kapitel 10

MENUETT

Kapitel 11

ADVENTUR

Kapitel 12

DATABASE
LÄNDER

VERZEICHNIS DER SCHLÜSSEL-WÖRTER

Im folgenden finden Sie – Kapitel für Kapitel – eine Liste aller Schneider BASIC-Schlüsselwörter, die in diesem Manual behandelt wurden. Es wurden nicht alle Variationen und Möglichkeiten behandelt, obwohl dies eigentlich ein Buch für Fortgeschrittene ist. Kenntnisse über die internen Abläufe des CPC464 sind notwendig zum richtigen Verständnis.

Die mit einem (*) versehen Schlüsselwörter wurden bereits in Teil 1 dieses Lehrbuches eingeführt, sind aber in diesem Teil nochmals ausführlicher erklärt.

Kapitel 2

DIM
READ
DATA
AUTO
RENUM
RESTORE
DELETE
LIST*

Kapitel 3

PEN*
PAPER*
INK*
PLOT*
DRAW*
ORIGIN*
CLG
TAG
TAGOFF

Kapitel 4

FOR-NEXT*
IF-THEN-ELSE*
WHILE-WEND

Kapitel 5

ZONE
PRINT*
PRINT USING
SPACE$
STRING$
SPC
TAB
INPUT*

Kapitel 6

SIN
COS
TAN
DEG
PI
RAD
ATN
SQR
ABS

Kapitel 7

INSTR
ASC
VAL
LEN
LEFT$
RIGHT$
MID$
STR$
TIME*
CHR$
AND
OR
XOR
UPPER$

Kapitel 8

SPEED INK
RND*
INT
LOCATE*
TEST
INKEY

Kapitel 9

SOUND*
ENV*
ENT*

Kapitel 12

KEY
PRINT &8
LIST &8
SPEED WRITE

INDEX