DuMont's Handbuch zum ›Schneider CPC‹

Randvoll mit Informationen, Programmen, Grafik und Musik Tips und Tricks

Die beste Gelegenheit, sich mit Ihrem »Schneider« anzufreunden!

Andreas Werminghoff

DuMont's Handbuch zum Schneider CPC 464

Andreas Werminghoff

dumont creativ COMPUTER


Hallo,Fans
Hallo,Fans

Andreas Werminghoff

DuMont's Handbuch zum ›Schneider CPC‹

Die beste Gelegenheit, sich mit Ihrem »Schneider« anzufreunden!

DuMont Buchverlag Köln

Meinen Eltern

Alle technischen Angaben und Programme dieses Buches wurden vom Autor mit größter Sorgfalt erarbeitet und vor Reproduktion überprüft. Trotzdem sind Fehler nicht ganz auszuschließen. Daher möchten Autor und Verlag darauf hinweisen, das keinerlei Garantie, juristische Verantwortung oder Haftung irgendeiner anderen Art für Folgen übernommen werden kann, die auf fehlerhafte Angaben in diesem Buch zurückgehen. Für die Aufklärung eventueller Fehler sind Autor und Verlag jederzeit dankbar.

Satz und Druck: Rasch, Bramsche
Buchbinderische Verarbeitung: Bramscher Buchbinder Betriebe

Printed in Germany ISBN 3-7701-1724-7

Inhaltsverzeichnis

Einleitung

Ich bin verliebt. Mir war schon lange klar, daß meine Vorliebe für Computer irgendwann einmal krankhafte Züge annehmen würde; seitdem ich nun den Schneider CPC 464 kenne, ist es soweit. Als ich in der Presse die ersten überschwenglichen Ankündigungen las, habe ich an eine Zeitungsente geglaubt. Dann, als das Gerät auf meinem Schreibtisch stand und ich ausprobieren konnte, zu was es fähig war, mußte ich zugeben, daß der Firma ein Treffer gelungen war. Je näher ich diesen Computer kennenlernte, desto besser gefiel er mir. (Allerdings stellte ich im Laufe der Zeit auch fest, daß mein geliebter Computer einige »schiefe Zähne« hat – doch davon später.) Zunächst möchte ich über die positiven Eigenschaften dieses Computers sprechen.

Das Revolutionäre am Schneider CPC ist schon bei der ersten Bekanntschaft zu sehen. Er ist der erste Computer in dieser Preisklasse, der von Beginn an als komplettes System ausgeliefert wird – also inclusive Monitor und Speichermöglichkeit. Dazu kommt noch ein Super-BASIC, das einige Optionen hat, die bis dahin nur in Maschinensprache oder mit einem gehörigen Geldbetrag zu verwirklichen waren. Das BASIC ist so durchdacht, daß man weitgehend auf lästige PEEKs und POKEs verzichten kann. Den Benutzer läßt es hoffen, daß seine Programme auch auf späteren Schneider-Computern laufen werden.

Einige Worte noch zur Geschichte des Schneiders. Wie eigentlich kommt ein Unternehmen der Unterhaltungselektronik dazu, sich auf dem heiß umkämpften Computermarkt zu tummeln? Das ist schnell gesagt: Firma Schneider unterhält schon seit Jahren Geschäftsverbindungen zu der Firma Amstrad in England. Diese Firma stellt nämlich integrierte Schaltkreise her, die auch in Produkten von Schneider Verwendung finden. Als Amstrad zusammen mit Locomotive Software einen Computer entwickelte, bot sich die bestehende Geschäftsverbindung an, um einen Vertrieb in deutschsprachigen Ländern zu organisieren. Daß es der Firma Schneider gelang, den Computer unter ihrem bekannten Markennamen zu verbreiten, tat ein übriges, und der Erfolg gab schon jetzt der Marktpolitik recht: In dem Vierteljahr nach Erscheinen des Computers wurden alleine im deutschsprachigen Raum 40000 Exemplare verkauft – für einen Marktneuling ein absoluter Rekord. In meinen Telefonaten mit Schneider stellte sich heraus, daß niemand darüber erstaunter war als Schneider selbst; die Mitarbeiter von Schneider kamen mir manchmal vor wie die kritische Mutter eines Wunderkindes, die nicht so recht an das Talent ihres Sprößlings glauben will.

Mir geht es da vollkommen anders: Ich bin davon überzeugt, daß der Schneider seinen Weg machen wird. Es kann nur eine Frage der Zeit sein, bis für diesen Computer ähnlich viel Software auf dem Markt sein wird wie für Geräte, die sich schon seit Jahren der Beliebtheit der Hacker erfreuen. Dieses Buch soll dabei allen, die sich der Möglichkeiten des Schneiders nicht voll bewußt sind, eine Stütze sein.

Bei der Erstellung des Buches halfen mir mit Rat: Bernd Dinow und das Softwarehaus ESCON und mit Tat: Walter Müller, Percy Dibold und die netten Leute vom SATURN-Computershop.

All diesen »Guten Geistern« meinen tiefen Dank und den Wunsch, daß sie soviel Freude am Schneider CPC 464 haben mögen, wie ich beim Schreiben dieses Buches empfand.

März 1985 *Andreas Werminghoff*

Der erste Besuch des Schneiders

Der erste Kontakt mit einem neuen Computer, vor allen Dingen dann, wenn er im Wohnzimmer steht, geht immer über BASIC. Im vorliegenden Fall finden Sie nicht nur einen neuen Computer, sondern auch ein neues BASIC. »Oh je«, werden die Fachleute stöhnen »schon wieder ein neuer BASIC-Dialekt!«. Es gibt zu Seufzern aber absolut keinen Anlaß: Das BASIC von Locomotive-Software ist, betrachtet man die Syntax der einzelnen Befehle, weitgehend kompatibel zum Microsoft-BASIC. (Das kann zum Beispiel heißen, daß man Listings, die für den IBM-PC geschrieben wurden, sehr schnell auf das Schneider-BASIC umschreiben kann.) Zusätzlich dazu bietet der Schneider noch einige Optionen, die das Programmieren zum Vergnügen machen. Wir werden uns im Laufe der Zeit mit diesen Eigenheiten vertraut machen. Für diejenigen unter meinen Lesern, die mit der Programmiersprache BASIC noch nicht so vertraut sind und denen die Übungsbeispiele aus dem Handbuch nicht alles sagen, folgt erst eine kleine Einführung in BASIC; Fachleute sollten auf Seite 16 weiterlesen.

BASIC ist die Programmiersprache, die aufgrund ihrer leichten Erlernbarkeit innerhalb von zwanzig Jahren zur meistgebrauchten im Mikrocomputerbereich wurde. Sie wurde 1964 von den Professoren John Kemeny und Tom Kurtz mit Hilfe ihrer Studenten am Dartmouth College entwickelt. Ziel der Entwicklung war eine Sprache, die sowohl Texte als auch Zahlen bequem verarbeiten können sollte. Die Wörter dieser Sprache sind Befehle. Allerdings sind es keine Befehle, die direkt an das Gehirn des Computers – den Mikroprozessor – gerichtet sind, sondern sie gelten einem Programm. Dieses Programm läuft in dem Augenblick los, in dem Sie den Computer anschalten: Es ist der BASIC-Interpreter. Da unser Computer leider kein Englisch kann, brauchen wir einen Übersetzer, der unser Kauderwelsch in klare, deutliche Maschinensprache übersetzt. Es gibt verschiedene Übersetzer, allerdings nur einen, der fest eingebaut ist. Dieser eine macht sich jedesmal, wenn er mit einer Arbeit fertig ist, bemerkbar. Er meldet sich mit:

```
Ready
```

Wenn Sie Ihren Computer einschalten (Achtung! Richtige Reihenfolge! Erst Monitor – dann Tastatur!), sehen Sie das READY (Fachleute nennen es »Prompt«) zum erstenmal. In der kurzen Zeit, die zwischen Einschalten und Erscheinen des Schriftzuges vergeht, hat

der Computer schon Schwerarbeit geleistet. Er hat sein gesamtes Gedächtnis (den Speicher) in Ordnung gebracht (das heißt, er hat den für uns normal Sterbliche erreichbaren Teil überall auf Null gestellt), er hat dafür gesorgt, daß die Tastatur benutzbar ist und daß der Bildschirm beim Einschalten immer in der gleichen Farbe erscheint.

Es gibt zwei Möglichkeiten, den Computer in BASIC zu benützen:

1. Den Direktmodus

Im Direktmodus wird jeder Befehl, den Sie dem Computer geben, sofort ausgeführt und sofort anschließend wieder vergessen. Als Beispiel tippen Sie bitte in Ihren Schneider CLS ein und drücken danach die ENTER-Taste. Die Auswirkungen dieses Befehls sehen Sie sofort: Der Bildschirm wird gelöscht und der Computer meldet sich mit Ready.

2. Den Programmodus

Im Programmodus werden Befehle erst dann ausgeführt, wenn Sie dem Computer sagen, daß er sie befolgen soll. Der Befehl, mit dem Sie dem Computer mitteilen, daß er mit der Arbeit an einem Programm beginnen soll, heißt RUN. Wie jede BASIC-Zeile, wird auch dieser Befehl durch die ENTER-Taste abgeschlossen. (Von nun an werde ich ⟨CR⟩ schreiben, wenn Sie die ENTER-Taste drücken sollen: Also tippen Sie RUN⟨CR⟩.) Wenn Sie Ihr Gerät eben erst eingeschaltet haben, meldet sich Ihr Computer sofort mit Ready. Es gab in seinem Speicher nämlich kein Programm, das er hätte ausführen können. Also müssen wir wissen, wie eine Programmzeile in BASIC aussieht; der Aufbau ist immer gleich: Sie beginnt mit einer Zeilennummer und danach kommt ein Befehl. Da Sie schon den Befehl kennen, der den Bildschirm löscht, heißt jetzt unsere erste Programmzeile:

```
10 CLS<CR>
```

Es geschieht nichts. Erst müssen Sie dem Computer sagen, daß er das Programm bearbeiten soll. Den Befehl dazu kennen Sie schon: RUN⟨CR⟩. Wenn Sie diesen Befehl geben, wird der Bildschirm gelöscht. Ihr Programmtext ist verschwunden, aber nicht vergessen. Den Beweis liefert Ihnen der Befehl

```
LIST<CR>
```

Ihr Bildschirm sieht jetzt folgendermaßen aus:

```
Ready

LIST

10 CLS

Ready
```

Jedesmal, wenn Sie jetzt RUN⟨CR⟩ eingeben, wird Ihr Bildschirm gelöscht.

Zu den Befehlen RUN und LIST gehört noch ein dritter – NEW. Diesen Befehl benützen Sie bitte mit äußerster Vorsicht. Er löscht nämlich das BASIC-Programm im Speicher. Da unser Programm so kurz ist, daß der Verlust nicht schmerzt, probieren Sie ihn ruhig aus. Tippen Sie NEW⟨CR⟩ und danach LIST⟨CR⟩ – Ihr Programm ist nun vom Computer vergessen.

Die Befehle RUN, LIST und NEW gehören zur Gruppe der Systembefehle. Das sind Befehle, die nur in wenigen Fällen im Programm verwendet werden sollten, da sie gravierende Änderungen im Speicher auslösen.

Der nächste Befehl, mit dem wir uns vertraut machen wollen, ist der Befehl:

PRINT

Dieser Befehl bringt das auf den Bildschirm, was nach dem PRINT-Befehl steht, und sorgt außerdem dafür, daß der Cursor (das helle Kästchen, das Sie unter dem READY sehen) an den Beginn der nächsten Zeile springt. Sie können den PRINT-Befehl immer durch ein Fragezeichen abkürzen, also ? statt PRINT. Versuchen Sie nun mal folgendes:

PRINT hallo<CR>

Statt des zu erwartenden »hallo« erscheint auf Ihrem Bildschirm eine Null (durch den Schrägstrich gut von dem Großbuchstaben O zu unterscheiden). Woran liegt's?

Computer heißt übersetzt Rechner, und das muß man wörtlich nehmen. Wenn ein Computer überhaupt etwas gerne tut, dann rechnen. Oder anders ausgedrückt, müssen Sie es dem Computer immer ausdrücklich sagen, wenn er Texte verarbeiten soll, weil er sonst davon ausgeht, daß es sich um Zahlen oder ähnliches handelt. Texte (das richtige Wort ist »Zeichenkette« oder »String«) werden in BASIC durch Anführungszeichen ('') begrenzt. Ein String muß zu Beginn und am Ende durch Gänsefüßchen eingeschlossen sein. Um auf unserem Bildschirm ein freundliches »hallo« zu erhalten, müssen wir also schreiben:

PRINT "hallo"<CR>

Jetzt ist das Ergebnis wie erwartet; auf dem Bildschirm sehen Sie unter Ihrem Befehl:

hallo

Ready

Bevor wir uns mit weiteren Befehlen befassen, noch einige Worte zu den Zeilennummern: Sie dürfen beim Schneider im Bereich von 1 bis 65535 liegen. Es dürfen nur ganze Zahlen sein, und jede Zeilennummer darf nur einmal vorkommen. Bitte schreiben Sie auf Ihrem Schneider folgende Zeilen:

```
20 PRINT "computer"<CR>

10 PRINT "schneider"<CR>
```

Tippen Sie dann LIST⟨CR⟩ und schauen Sie Ihr Programm an. Ihr Computer war so freundlich, die Zeilen in der richtigen Reihenfolge zu ordnen:

```
10 PRINT "schneider"

20 PRINT "computer"
```

Alles, was mit einer Zeilennummer beginnt, wird vom Computer automatisch als Programmzeile interpretiert. Die Zahl sagt dem Computer, daß er die Programmzeilen einordnen bzw. abarbeiten soll. Geben Sie nun eine neue Zeile 10 ein:

```
10 PRINT "basic"<CR>
```

und dann listen Sie das Programm. Sie erhalten:

```
10 PRINT "basic"

20 PRINT "computer"
```

Also gilt immer die letzte Version einer Programmzeile. Die vorherige Version wird überschrieben. Schreiben Sie nun eine Programmzeile ohne Inhalt:

```
10<CR>
```

Wenn Sie nun LIST⟨CR⟩ eingeben, erhalten Sie:

```
20 PRINT "computer"
```

Das Eingeben einer Zeilennummer ohne Inhalt löscht also die entsprechende Programmzeile. Wenn Sie versuchen, eine Programmzeile zu löschen, die nicht existiert, so wirft Ihnen Ihr Schneider eine Meldung vor die Füße:

```
Line does not exist
```

Die Programmzeile existiert eben auch nicht. Unser Computer kann natürlich rechnen. Der Weg ist ganz einfach: Zuerst kommt der Befehl, daß das Folgende ausgegeben werden soll, dann das, was er ausgeben soll, nämlich das Ergebnis der Rechnung, die nach PRINT steht. Vorsicht: Der Computer rechnet genau das, was Sie ihm sagen; wenn Sie also die Aufgabe

```
9+3
---
6+6
```

rechnen wollen, können Sie es auf keinen Fall in dieser Form eingeben, da Sie ja alles in einer Zeile unterbringen müssen. Sie brauchen auch noch die Information, wie das Zeichen für die Division in BASIC aussieht: Es ist der Schrägstrich von unten links nach oben rechts. Sollten Sie nun versuchen, die obige Aufgabe so zu lösen:

```
PRINT 9+3/6+6<CR>
```

so werden Sie als Ergebnis 15.5 erhalten – zweifellos ein falsches Ergebnis zu der Ihnen gestellten Aufgabe. Der Computer hat, da auch er die Rechenregeln kennt, Punktrechnung vor Strichrechnung erledigt und seine Arbeit so geleistet:

```
3/6=0.5
0.5+9+6=15.5
```

Bis jetzt haben wir noch nichts gelernt, was den Unterschied zwischen Programm- und Direktmodus sinnvoll macht. Egal, ob wir schreiben

```
PRINT"Computer"
```

oder

```
10 PRINT"Computer"
```

der Schneider wird immer einmal »Computer« schreiben, und sich dann mit READY melden. Ein Unterschied zwischen beiden liegt in der Wiederholbarkeit der zweiten Version, der zweite in der Tatsache, daß wir die zweite Version auf Kassette aufnehmen können. Tippen Sie im Direktmodus:

```
SAVE"PROG1"<CR>
```

und Ihr Computer meldet sich mit der Bitte:

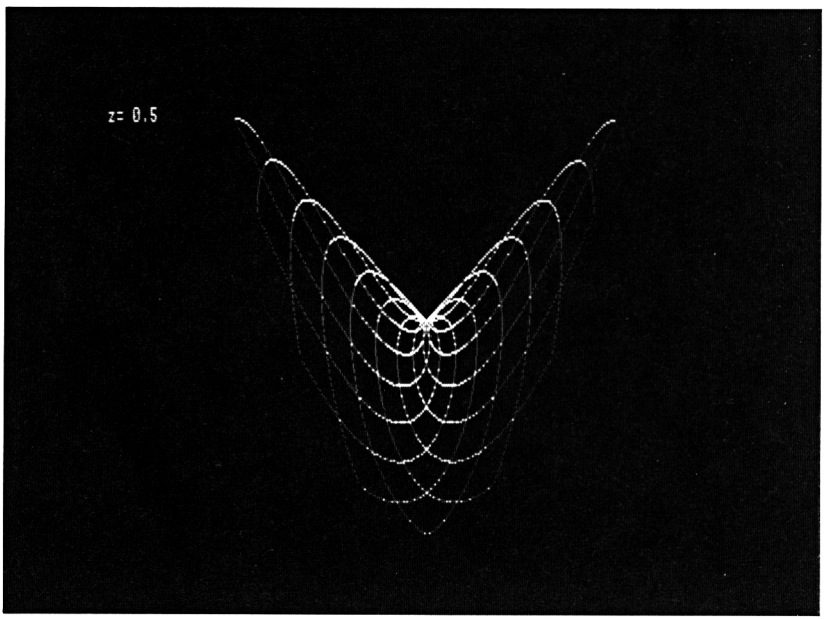
```
Press PLAY and REC then any key
```

Folgen Sie dieser Bitte nicht, sondern drehen Sie den Lautstärkeregler rechts auf laut – also nach oben (Vorsicht! Mir ist es beim erstenmal gelungen, statt dessen meinen Computer auszuschalten). Danach drücken Sie auf eine Taste; da der Schneider nicht überprüfen kann, ob Sie den Recorder tatsächlich auf Aufnahme gestellt haben, wird er

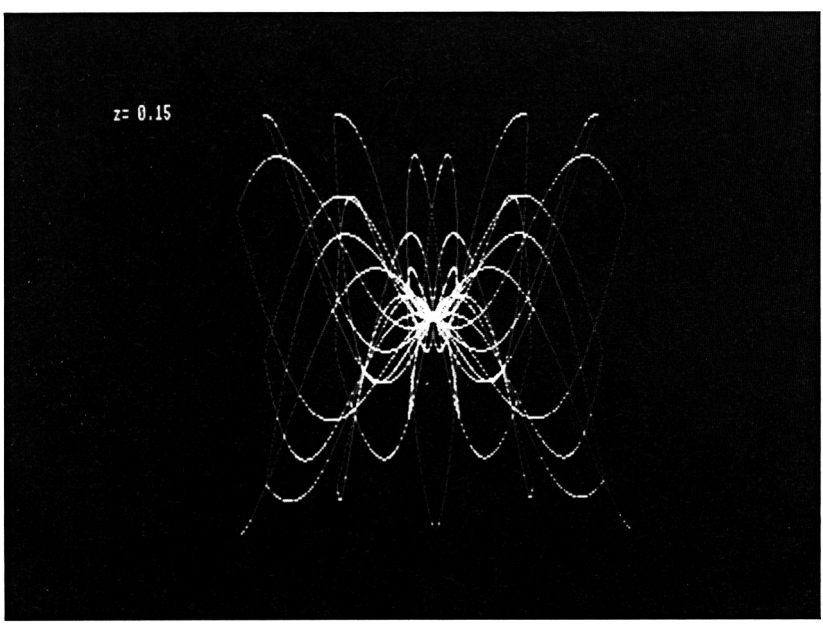
jetzt versuchen, das Programm im Speicher auf Kassette zu kopieren. Sie merken es an dem piepsenden Geräusch, das aus dem Lautsprecher kommt, und an dem Schriftzug:

```
Saving"PROG1"block 1
```

Zum Speichern eines Programmes werden wir noch einmal kommen, wenn unsere Programme etwas länger werden. Löschen Sie nun Ihren Speicher mit NEW⟨CR⟩.



```
10 INPUT"Faktor";z
20 MODE 2
30 ORIGIN 320,200
40 DEG
50 PRINT"z=";z
60 DEFINT i,x,y
70 FOR i=-32000 TO 32000
80 IF INKEY$<>"" THEN RUN
90 x=(i MOD 200)*SIN(i*z)
100 y=(i MOD 200)*COS(i)
110 PLOT x,y
120 NEXT
Ready
```



Man nehme...

Wenn Sie Ihr eigenes Süppchen in BASIC kochen wollen, müssen Sie die Zutaten, die Sie zur Verfügung haben, gut kennen. Die einzelnen Elemente der Sprache sind:

Befehle
Funktionen
Konstanten
Variablen
Operatoren

Befehle

Jeder Befehl hat seine genau vorgeschriebene Grammatik – die Syntax. Befehle sagen dem Computer, was er machen soll. CLS zum Beispiel ist ein Befehl an den Computer, den Bildschirm zu löschen. Die Befehle lassen sich in vier große Gruppen einteilen:

1. Systembefehle
Diese Befehle haben Einfluß auf das Programm und den Inhalt des Programmspeichers. Der Befehl LIST zum Beispiel bringt den Programmtext auf den Bildschirm und unterbricht das Programm, wenn er innerhalb des Programmtextes eingesetzt ist; der Befehl NEW löscht das Programm usw. Systembefehle sollten nur in Ausnahmefällen in ein Programm selbst eingesetzt werden.

2. Eingabebefehle
Sie sorgen dafür, daß Informationen während eines laufenden Programms in das Programm gelangen können. Der meistgebrauchte Eingabebefehl ist INPUT.

3. Verarbeitungsbefehle
Diese Befehle steuern den Programmablauf (IF,GOTO) und ändern den Inhalt von Variablen (Zuweisung).

4. Ausgabebefehle
Alles, was an Informationen aus einem laufenden Programm zu der Peripherie (Monitor, Recorder, Diskettenlaufwerk, Lautsprecher usw.) gelangt, wird durch einen Ausgabebefehl dorthin geschickt (z. B. PRINT).

Funktionen

Auch Funktionen sind Arbeitsanweisungen für den Computer. Mit Funktionen wird aus einem bestimmten Wert, dem Argument, ein anderer ermittelt. Da eine Funktion aber nichts an einem Wert ändern kann, muß man das Ergebnis dieser Funktion mit einem Befehl weiterverarbeiten (PRINT SIN(3) oder a$=LEFT$(b$,5)). Funktionen lassen sich in drei Gruppen aufteilen:

1. Gleitkommafunktionen
Diese ermitteln aus Ihrem Argument einen numerischen Wert, der entweder Nachkommastellen haben kann oder größer als +32767 bzw. kleiner als −32768 werden kann (SIN,TAN usw.).

2. Integer- oder Ganzzahlfunktionen
Solche Funktionen erzeugen eine Zahl ohne Nachkommastellen im Bereich von −32768 bis +32767 (VPOS,ASC).

3. String- oder alphanumerische Funktionen
Hier wird aus dem Argument eine Zeichenkette oder ein einzelnes Zeichen erzeugt (CHR$,MID$).

Konstanten

Konstanten sind Informationen, die im Programmtext stehen und während eines laufenden Programmes nicht geändert werden können. Es gibt numerische Konstanten (234,0.12) und alphanumerische Konstanten ("SCHNEIDER").

Variablen

Will man Informationen verändern, braucht man »Plätze« innerhalb des Programms, in denen der Computer die Informationen aufbewahren kann, die sich während des Programmablaufs ändern. Damit man diese Informationen weiterverarbeiten kann, gibt man den »Plätzen« Namen, anhand derer man dann die Informationen, die sich an diesen Plätzen befinden, ansprechen kann. Die »Plätze« heißen Variablen. Es gibt beim Schneider drei verschiedene Typen von Variablen:

Gleitkommavariablen
Integer- oder Ganzzahlvariablen
String- oder alphanumerische Variablen

Hier eine Aufstellung über die Unterschiede:

Variablentyp	*Gleitkommavariablen*	*Integervariablen*	*Zeichenketten-Stringvariablen*
Name	Für alle Typen gleich: Keine BASIC-Wörter, maximal 40 Zeichen; das erste Zeichen muß ein Buchstabe sein.		
Kennzeichnung	Normalerweise nicht nötig, bei Verwendung von DEFINT oder DEFSTR: !	%	$
Möglicher Inhalt	Ziffern 0 bis 9, Vorzeichen + und – Dezimalpunkt (!) . E für × 10 hoch	Ziffern 0 – 9 Vorzeichen + und – E für × 10 hoch	Alle druckbaren Zeichen (es gibt 256 verschiedene)
Maximaler Inhalt	-1.70141×10^{38} bis $+1.70141 \times 10^{38}$	– 32768 bis + 32767	Max. 255 Zeichen

Achtung! Der Variablenname DEC$ führt zu einem Syntax error! Einige Variablen benützt der Schneider selbst. Sie können den Inhalt abfragen, ihn aber nicht durch eine Zuweisung ändern; dazu zählen: SQ(×), TIME, PI, EOF, ERR, ERL und HIMEM.

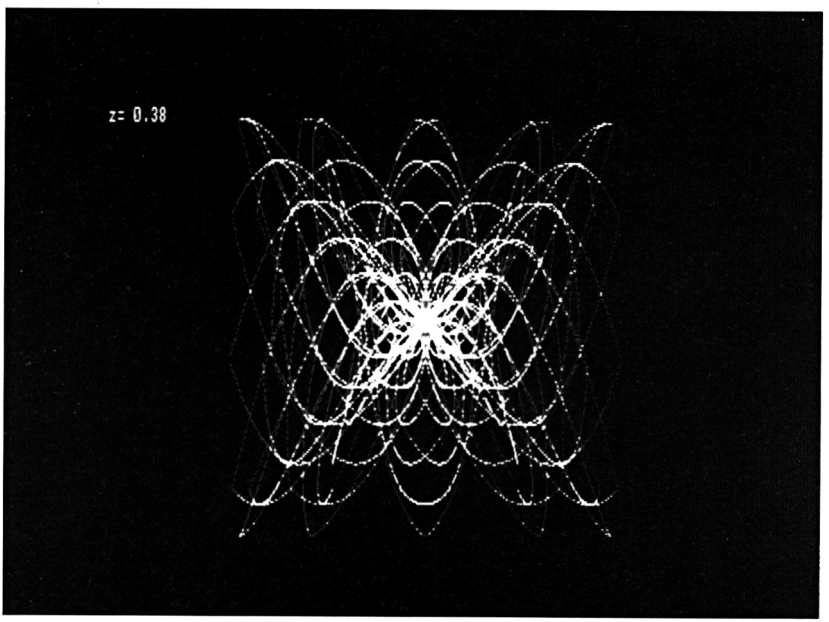
Operatoren

Operatoren verknüpfen zwei oder mehrere Ausdrücke miteinander und erzeugen aus diesen Ausdrücken einen neuen Wert; dieser Wert kann dann weiterverarbeitet werden. Man unterscheidet folgende Arten:

Arithmetische Operatoren: + – * / ^ \ MOD
Logische Operatoren: < > <> <= >= =
Stringoperatoren: +
Operatoren der Aussagenlogik: NOT AND OR XOR

Ausdrücke

Ausdrücke können aus Kombinationen von Konstanten, Variablen und Funktionen gebildet werden. Fast immer kann eine Konstante oder Variable in einem Befehl durch einen Ausdruck ersetzt werden. Ausnahmen bilden die Zeilennummern: Sie müssen immer als konstante Werte angegeben werden.



Das Malerprogramm

(Bitte achten Sie beim Eintippen der folgenden Programmzeilen darauf, die gleichen Zeilennummern wie ich zu benutzen.)

Ich habe einen Freund, der Anstreicher ist. Er ist ein wirklich guter Anstreicher, hat aber für die Rechnerei, die oft erledigt werden muß, überhaupt kein Verständnis. Da er von den Vorteilen der Datenverarbeitung gehört hatte, kam er zu mir und bat mich, ihm ein Programm zu schreiben, das in der Lage sein sollte, Kostenvoranschläge zu liefern. Mit seiner Hilfe habe ich erst ein Programm geschrieben, das eine Fläche ausrechnen konnte. Eine Fläche berechnet sich nach der Formel: Fläche = Höhe × Breite. Eine Zeile 40

```
40 PRINT hoehe × breite
```

brachte uns nicht weiter, weil noch keine Werte für Höhe und Breite vorlagen. Also haben wir in den Zeilen 10 und 30 Höhe und Breite einem bestimmten Wert zugeordnet.

```
10 hoehe = 3
```

```
30 breite = 6
```

Mit diesem Ergebnis ist dann mein Freund beruhigt nach Hause gegangen. Bis zu seinem nächsten Besuch wollen wir uns überlegen, was die Zeilen 10 und 30 bedeuten: Mit dieser Art von Befehlen weisen wir der Variablen, die vor dem Gleichheitszeichen steht, den Inhalt zu, der nach dem Gleichheitszeichen kommt. Man kann die Zeile 40 ersetzen durch

```
40 flaeche = hoehe × breite
```

und den PRINT-Befehl später setzen:

```
60 PRINT''Einzelflaeche: ''flaeche
```

Zwei Tage später stand mein Freund wieder vor der Tür und machte mir bittere Vorwürfe: Es könne ja nicht der Sinn eines Programmes sein, daß man jedesmal, wenn man eine Fläche ausrechnen wolle, das Programm ändern müsse. Er suchte nach einem Befehl, mit dem man eine Variable füllen kann, wenn das Programm schon läuft. Der Befehl

```
INPUT
```

hilft uns da weiter. Ändern Sie die Zeilen 10 und 30 in:

```
10 INPUT hoehe
```

```
30 INPUT breite
```

und lassen Sie das Programm dann laufen. Bei der Verarbeitung von INPUT macht der Computer der Reihe nach folgendes:

1. Ein Fragezeichen und eine Leerstelle werden ausgedruckt;
2. die Tastatur wird freigegeben (der Cursor erscheint);
3. der Computer wartet auf das Drücken der ENTER-Taste;
4. der Computer versucht, das, was zwischen dem Fragezeichen und dem Drücken der ENTER-Taste eingegeben wurde, in die Variable zu füllen, die nach INPUT steht;
5. falls das nicht möglich ist, erscheint eine Fehlermeldung auf dem Bildschirm und der Computer macht weiter bei 1;
6. der Programmablauf wird fortgesetzt.

Die Fehlermeldung, die bei Falscheingaben erzeugt wird, heißt

```
?Redo from Start
```

Sie erscheint dann, wenn Ihre Eingabe ein Komma enthält oder – was oft vorkommt – Sie statt 1000 looo schreiben (kleines L und dreimal der Buchstabe o).

Das gleiche Problem hatte auch mein Freund. Als er eine Woche später wieder zu mir kam, erzählte er mir seine Sorgen. Er hatte mit dem Programm gearbeitet, als das Telefon klingelte. Sein Computer steht, damit ihn jeder gebührend bewundern kann, im Wohnzimmer – das Telefon aber im Büro. In der Zeit, in der er telefonierte, kommt sein Sohn, ein vierjähriger Steppke, ins Wohnzimmer, sieht, daß Papa endlich mal den Computer allein gelassen hat, und spielt solange herum, bis der Bildschirm voller Fehlermeldungen ist: Der Vater war nicht sehr begeistert. Er wollte von mir eine Programmerweiterung, die ihm eine Meldung auf dem Bildschirm ausgibt, was er eingeben sollte. Da man Programmzeilen einfügen kann, habe ich zwei Zeilen 5 und 15 hinzugefügt:

```
5 PRINT"Hoehe"
```

```
15 PRINT"Breite"
```

und ihn mit dieser Version nach Hause geschickt. Es dauerte nicht einmal drei Stunden, dann stand er wieder vor meiner Tür, diesmal wutschnaubend. Er hatte die neue Version gerade gestartet, als das Telefon klingelte. Die Meldung Hoehe war, als er vom Telefonieren zurückkam, schon längst vom Bildschirm verschwunden: Sein Sohn hatte mal wieder »programmiert«. Sein Vater hätte aber die Meldung nach einer Falscheingabe gerne wieder auf dem Bildschirm gehabt. Nach längerem Suchen in Fachbüchern und Manuals fand ich die Lösung, die die Zeilen 5 und 15 überflüssig machte:

```
5<CR>

15<CR>

10 INPUT"Hoehe";hoehe

30 INPUT"Breite";breite
```

Die Bedeutung des Semikolons können Sie auf Seite 158 finden. Jetzt konnte mein Freund beruhigt telefonieren und ich mich wieder meinem Schneider widmen (das dachte ich jedenfalls). Eine Woche danach stand der Maler wieder vor meiner Tür, und ich mußte unwillkürlich an das Märchen vom Fischer und seiner Frau denken. Nachdem er Platz genommen hatte, rechnete er mir vor: Sein nächster Auftrag sei, einen Häuserblock innen zu streichen, der insgesamt 100 Zimmer habe. Nicht nur, daß er pro Zimmer fünfmal die Breite und fünfmal die Höhe einzugeben hätte, zusätzlich müßte er auch fünfmal RUN⟨CR⟩ tippen. Er wollte einen Befehl, der den Computer an den Beginn des Programms zurückschickt, wenn dieser mit der Bearbeitung der Zeile 50 fertig ist, also so etwas wie:

```
80 GEHNACH 10
```

Da der Computer aber leider kein Deutsch versteht, müssen wir es ihm in Englisch sagen:

```
80 GOTO 10
```

Mein Freund war begeistert. Daß mitten in der Nacht mein Telefon klingelte, hatte ich schon erwartet: Mir war es gelungen, eine sogenannte Endlosschleife zu programmieren; es gibt keine Möglichkeit, aus diesem Programm herauszukommen – es sei denn, man kennt den Segen der ESC(ape) Taste. Drückt man auf die ESC-Taste, wenn eine Eingabe gefordert wird, so erscheint neben dem Fragezeichen:

```
*Break*
```

und darunter:
Break in . . .

mit Angabe der Zeile, in der unterbrochen wurde. Damit ist dieses Problem vorerst gelöst. Es ist zwar nicht die feine englische Art, ein Programm so zu beenden, aber es funktioniert, solange man diese Taste nicht außer Kraft setzt.

Bei mir klingelt's. Raten Sie mal, wer das sein kann. Richtig! Mein Freund, der Anstreicher. Er ist schon wieder schlecht gelaunt. Er wirft mir vor, ich hätte die Fähigkeiten des Computers nur zu einem Bruchteil ausgenützt; schließlich heiße Computer doch »Rechner«, und er säße Abend für Abend an seinem Schreibtisch und addiere lange Zahlenkolonnen, um die Gesamtfläche zu errechnen. Da ich so etwas kommen gesehen hatte, habe ich die Zeilennumerierung so gewählt, daß ich die Errechnung der Gesamtfläche noch hineinpacken kann:

```
50 gesflaeche = gesflaeche + flaeche
```

```
70 PRINT"Gesamtflaeche:" gesflaeche
```

Erinnern Sie sich daran, wie der Befehl in Zeile 40 (die Zuweisung) verarbeitet wird? Erst wird der Wert des Ausdrucks rechts vom Gleichheitszeichen ermittelt, und dann wird das Ergebnis in die Variable links vom Gleichheitszeichen gefüllt. Diese Reihenfolge macht es möglich, daß wir auf den Inhalt einer Variablen etwas addieren können, ohne jedesmal eine neue Variable benutzen zu müssen. Bei dieser Gelegenheit habe ich auf Wunsch in das Programm noch ein richtiges Ende eingebaut. Allerdings hilft eine Zeile

```
90 END
```

nicht weiter, da der Computer immer noch in der Zeile 80 nach 10 geschickt wird. Wir müssen es also schaffen, die Zeile 80 dann zu überspringen, wenn der Benutzer des Programmes es will. Man kann so etwas bewerkstelligen, indem man dem Benutzer sagt, daß bei einer bestimmten Eingabe das Programm zu Ende ist, und diese Eingabe abfragt. Zuerst ändern wir die Eingabezeile:

```
10 INPUT"Hoehe (99=Ende)";hoehe
```

Dann müssen wir dem Computer folgende Logik in BASIC klarmachen: Wenn der Inhalt der Variablen »hoehe« gleich der Zahl 99 ist, gehnach 90. Das Ganze in BASIC lautet:

```
20 IF hoehe = 99 THEN GOTO 90
```

Jetzt endet unser Malerprogramm, wenn auf die Frage »Hoehe (99=Ende)?« die Variable hoehe mit 99 gefüllt wird.

Der IF-Befehl ist die Möglichkeit, den Computer eine Entscheidung treffen zu lassen. Nach dem IF steht eine Aussage; der Computer überprüft, ob diese Aussage wahr ist, und wenn das der Fall ist, führt er den Befehl bzw. die Befehle aus, die nach dem THEN stehen. Wenn die Aussage nicht wahr ist, führt er die Anweisung aus, die nach dem IF-Befehl kommt – es sei denn, Sie hätten eine ELSE – Option eingesetzt. Diese sorgt dafür,

daß das, was nach dem ELSE steht, dann ausgeführt wird, wenn die Aussage hinter IF nicht wahr ist. Die Aussage nach IF können Sie auf verschiedene Art formulieren:

IF a=b THEN... ist dann wahr, wenn a gleich b ist.
IF a>b THEN... ist dann wahr, wenn a größer als b ist.
IF a<b THEN... ist dann wahr, wenn a kleiner als b ist.
IF a<>b THEN... ist dann wahr, wenn a und b verschieden sind.
IF a<=b THEN.. ist dann wahr, wenn a entweder kleiner oder gleich b ist.
IF a>=b THEN.. ist dann wahr, wenn a entweder größer oder gleich b ist.

Jetzt haben wir zwar ein Programm, das eine Fläche und eine Gesamtfläche errechnet, aber noch kein Programm, das meinem Freund hilft, einen Kostenvoranschlag zu erstellen. Vorher noch das gesamte Programm ohne die Zeile 90:

```
10 INPUT"Hoehe (99=Ende)";hoehe

20 IF hoehe = 99 THEN GOTO 90

30 INPUT"Breite";breite

40 flaeche = hoehe * breite

50 gesflaeche  = gesflaeche + flaeche

60 PRINT"Einzelflaeche "flaeche

70 PRINT"Gesamtflaeche "gesflaeche

80 GOTO 10
```

Und nun der Rest:

```
90 INPUT"Farbbedarf pro qm";farbbedarf

100 INPUT"Farbpreis pro kg";farbpreis

110 INPUT"Lohn pro qm";lohn

120 lohnkosten = gesflaeche * lohn

130 farbkosten = farbbedarf*farbpreis*gesflaeche
```

```
140 netto       = lohnkosten + farbkosten
150 mwst        = 0.14 * netto
160 brutto      = netto + mwst
170 PRINT"Materialkosten     "farbkosten
180 PRINT"Lohnkosten         "lohnkosten
190 PRINT"Nettopreis         "netto
200 PRINT"MWSt   14%         "mwst
210 PRINT"-------------------------------"
220 PRINT"Gesamtpreis        "brutto
230 PRINT"==============================="
240 END
```

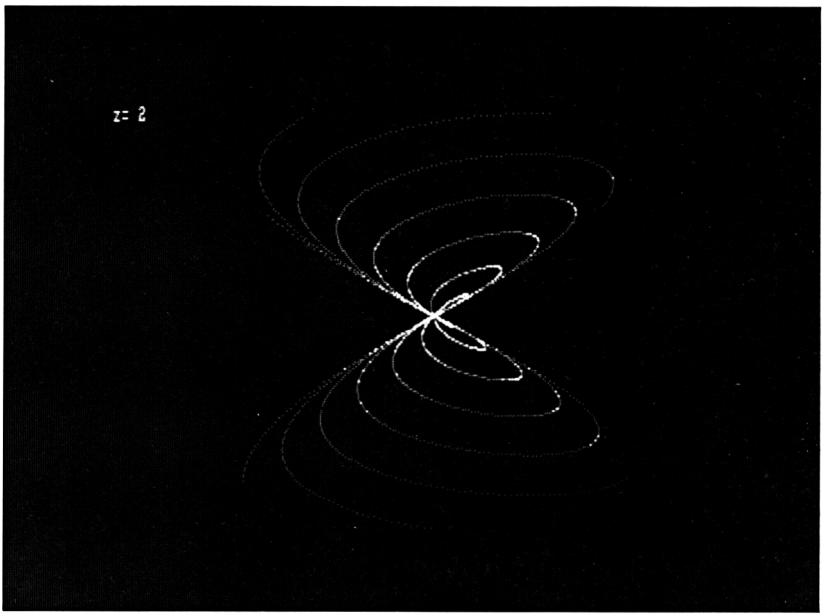
Diese Programm können wir jetzt speichern. Wenn sich der Computer mit Ready gemeldet hat, tippen Sie:

SAVE »Maler« <CR>

Legen Sie dann eine leere Kassette in das Kassettenlaufwerk und drücken Sie auf PLAY und REC. Der Schneider sagt Ihnen, was er jetzt macht. Er speichert den 1. Block des Programms MALER (Saving MALER block 1). Ein Block besteht aus dem Header (so etwas wie ein Titel für den Computer) und 2048 Einzelinformationen. Der Header enthält den Programmtitel, die Länge des Programms und den Platz im Speicher, wo das Programm abgelegt werden soll. Zusätzlich ist auch noch gespeichert, um welche Art von Programm es sich handelt, ob es beim Speichern geschützt wurde und in welcher Geschwindigkeit es gespeichert wurde. Die einzelnen Möglichkeiten von SAVE finden Sie auf Seite 205.

Einige Worte noch zu den Variablen. Es gibt natürlich auch Variablen, die Texte aufnehmen können; diese müssen dann aber extra gekennzeichnet werden und zwar durch ein angehängtes Dollarzeichen ($). Außerdem gibt es noch die sogenannten Integervariablen. Sie werden gekennzeichnet durch ein angehängtes Prozentzeichen (%). Aus

Gründen der Vollständigkeit kann man auch die ›normalen‹ numerischen Variablen, die sogenannten Gleitkommavariablen, extra kennzeichnen und zwar durch ein Ausrufezeichen (!).



Das Sammlerprogramm

(Bitte achten Sie auch hier auf die Zeilennummern.)

Der Mensch ist von Natur aus Sammler. Jeder fängt klein an, mit Steinen, Blättern oder Bindfäden, arbeitet sich dann langsam hoch zu Bierdeckeln, Kronenkorken oder Streichholzschachteln und erreicht dann die Vollendung mit Briefmarken, Kunst, Schallplatten oder Videokassetten (in der letzten Zeit ist noch die Gattung der Programmsammler hinzugekommen). Alle Sammler bewegt ein Gedanke: Wie halte ich Ordnung in meiner Sammlung, und woher weiß ich, ob ich einen bestimmten Gegenstand schon habe? Ich stelle Ihnen hier ein Programm vor, das eine Schallplattensammlung von einigen hundert Titeln verwalten kann und in dieser Sammlung jeden Titel von einem bestimmten Interpreten oder den Interpreten zu einem bestimmten Titel finden kann. Außerdem soll es sämtliche Platten auflisten. Falls Sie einen Drucker besitzen, können Sie das Programm leicht so umändern, daß es eine Liste auf den Drucker schreibt.

Was ist dabei zu beachten? Das Programm soll so aufgebaut sein, daß man auch in zwei Jahren noch weiß, was man da programmiert hat, also wird es aufgeteilt in kleine Blöcke. Es ist eine Riesenmenge von Daten zu verwalten. Jede einzelne Information einer Variablen zuzuordnen, wäre nicht von Vorteil, weil der Computer immer nur eine Information verarbeiten kann. Also brauchen wir einen Befehl, der in der Lage ist, Daten beliebiger Art in einem Programm aufzuheben und diese Daten erst auf Anforderung in eine beliebige Variable zu füllen. Der Befehl, den wir dazu brauchen, heißt

DATA

Er kann ausschließlich in einem Programm verwendet werden, weil der Computer die Informationen nach DATA im Direktmodus nicht wiederfinden würde. Tippen Sie eine Zeile:

```
10 DATA Elvis, Hounddog
```

und starten Sie diese Zeile mit RUN. Das Ergebnis ist mager. Außer Ready oder Syntax error in 10, wenn Sie sich verschrieben haben, erscheint nichts auf dem Bildschirm. Der DATA-Befehl hebt nur Informationen für eine spätere Leseanweisung auf. Also geben Sie dem Computer einen Befehl:

```
20 READ a
```

Wenn Sie dieses Programm laufen lassen, erhalten Sie plötzlich eine Fehlermeldung in 10. Da die Zeile 10 eben reibungslos gelaufen ist, kann der Fehler trotz der Meldung nur in Zeile 20 liegen. Sie haben versucht, die numerische Variable a mit einer Zeichenkette zu füllen. So etwas führt normalerweise bei einer Zuweisung zu einem »Type mismatch error«. Sie müssen selbst darauf achten, daß die Information, die Sie aus einer DATA-Zeile in eine Variable nach READ lesen, in diese Variable hineinpaßt. Verbessern Sie Ihre Zeile 20 in:

```
20 READ a$
```

und, damit Sie auch sehen können, was geschieht, fügen Sie eine Zeile 30 hinzu:

```
30 PRINT a$
```

Jedesmal, wenn Sie jetzt RUN eingeben, erscheint »Elvis« auf dem Bildschirm. Um an die zweite Information heranzukommen, können Sie zwei Wege gehen. Hier der erste:

```
40 READ b$
```

```
50 PRINT b$
```

Oder, wenn Sie das Ganze lieber kürzer hätten:

```
20 READ a$,b$
```

```
30 PRINT a$,b$
```

In beiden Fällen geschieht das gleiche: Der Computer liest die erste Information in die erste Variable (hier a$) und die zweite Information in die zweite Variable (b$). Wichtig ist nicht der Name der Variablen, sondern die Reihenfolge, in der eingelesen wird. Fügen Sie eine Zeile 11 hinzu:

```
11 DATA Bassey,Goldfinger
```

und eine Zeile

```
40 GOTO 20
```

Ihr Programm wird zwar von beiden Platten Interpret und Titel auf den Bildschirm bringen, sich danach aber mit einer Fehlermeldung

```
DATA exhausted in 20
```

verabschieden. Die Arbeit, die der Computer bei einer READ-Anweisung erledigen muß, setzt sich aus einigen Aufgaben zusammen:

Es wird im gesamten Programm nach einer DATA-Zeile gesucht. Die erste Information in dieser Zeile wird in die erste Variable nach READ gefüllt. Der Computer setzt intern einen Zeiger auf den Beginn der nächsten Information; dieser Beginn wird entweder durch ein Komma oder durch das Wort DATA gekennzeichnet. Wenn die letzte Information gelesen worden ist, deutet der Zeiger auf einen BASIC-Befehl. Es ist nicht möglich, diesen in eine Variable zu füllen – und deshalb gibt es die Fehlermeldung.

Wenn Sie die Informationen in einer DATA-Zeile mehrmals lesen wollen, brauchen Sie einen Befehl, der den Lesezeiger auf die erste DATA-Zeile zurücksetzt. Dieser Befehl heißt:

```
RESTORE
```

Nach RESTORE kann eine Zeilennummer stehen, die dafür sorgt, daß die nächste Information aus der entsprechenden Zeile gelesen wird. Setzen Sie jetzt eine Zeile ein:

```
35 RESTORE
```

Wenn Sie sich am Elvis sattgesehen haben, versuchen Sie es mit:

```
35 RESTORE 20
```

Das hilft anscheinend auch nicht weiter, denn jetzt haben Sie ununterbrochen Bassey und Goldfinger auf dem Schirm.

Wir müssen also dafür sorgen, daß der Zeiger erst dann zurückgesetzt wird, wenn alle Informationen aus den DATA-Zeilen gelesen sind. Erinnern Sie sich an das Malerprogramm: Wenn da eine Arbeit nicht mehr durchgeführt werden sollte, haben wir die Variable »hoehe« mit 99 gefüllt; eine ähnliche Konstruktion können wir auch hier benutzen: Wir füllen die Variablen a$ und b$ beim letzten Lesen mit dem Buchstaben x und überprüfen dann den Inhalt nach dem Lesen. Damit die beiden x nicht auf dem Bildschirm erscheinen, überspringen wir auch die Zeile mit dem PRINT-Befehl. Und hier die Änderungen:

```
19 DATA x,x
```

```
35 <CR>
```

```
25 IF a$="x" then RESTORE:GOTO 40
```

Jetzt können Sie sich an beiden Namen gleichzeitig sattsehen. Wenn Ihnen der Aufbau dieses Programms klar ist, können wir uns mit dem Programm beschäftigen, für das Sie diese Befehle brauchen.

```
NEW<CR>
```

In den Zeilen 10 bis 999 stehen unsere Schallplatten mit Interpret und Titel:

```
10 DATA Elvis,Hounddog

20 DATA Nena,99 Luftballons

30 DATA Jackson,Thriller

 .

 .

 .

999 DATA x,x
```

Danach folgt das Hauptmenue, also der Teil, in dem der Benutzer des Programms zwischen den Programmpunkten wählen kann, die das Menue anbietet. Wir beginnen mit einem REM, dem einzigen Befehl, der buchstäblich nichts macht. Alles, was nach einem REM steht, wird vom Computer nicht verarbeitet. REM kann durch ein Apostroph abgekürzt werden ('). Es wird benützt, damit der Programmierer sich hinterher noch in seinem Programm zurechtfindet.

```
1000 'Hauptmenue

1010 CLS

1020 PRINT:PRINT

1030 PRINT"      Hauptmenue"

1040 PRINT

1050 PRINT"  <1> Suchen nach Interpret"

1060 PRINT"  <2> Suchen nach Titel"

1070 PRINT"  <3> Liste drucken"

1080 PRINT"  <4> Programmende"
```

```
1090 PRINT:PRINT

1100 INPUT"      Ihre Wahl (1-4)";wahl

1110 IF wahl=1 THEN GOTO 2000

1120 IF wahl=2 THEN GOTO 3000

1130 IF wahl=3 THEN GOTO 4000

1140 IF wahl=4 THEN GOTO 5000

1150 GOTO 1000
```

Wenn Sie dieses Programm starten und bei der Frage »Ihre Wahl (1–4)?« eine Zahl eingeben, gibt es zwei Möglichkeiten: bei einer Zahl zwischen 1 und 4 erhalten Sie eine Fehlermeldung »Line does not exist in ...«. Bei jeder anderen geschieht nichts, außer daß der Bildschirm neu aufgebaut wird. Um diese Fehlermeldungen abzuschaffen, müssen wir einen »Rahmen« für unsere einzelnen Programmteile schaffen:

```
2000 'Suchen nach Interpret

2999 GOTO 1000

3000 'Suchen nach Titel

3999 GOTO 1000

4000 'Liste drucken

4999 GOTO 1000

5000 'Programmende

5999 END
```

Mit diesen Zeilen ist das Skelett unseres Programms fertig; als nächstes soll etwas Fleisch an die Rippen. Machen wir uns klar, was der erste Programmteil tun soll. Der Benutzer soll einen Interpreten eingeben, und der Computer soll in den DATA-Zeilen nach genau diesem Interpreten suchen. Wenn er ihn in einer Zeile gefunden hat, soll er den Interpreten und den Titel ausdrucken.

```
2020 CLS

2030 PRINT:PRINT

2040 INPUT"Nach welchem Interpreten soll
          gesucht werden";such$

2050 PRINT

2060 READ a$,b$

2120 IF a$<>such$ THEN GOTO 2060

2130 PRINT"Interpret: "a$

2140 PRINT"Titel:     "b$

2150 PRINT

2160 GOTO 2060
```

Jetzt findet das Programm aus Ihren DATA-Zeilen den Interpreten und den Titel, wenn er vorhanden ist, beendet seine Sucherei aber immer mit:

```
DATA exhausted in 2060
```

Da wir in der Zeile 999 schon unser Dateiende mit x,x markiert haben, können wir unser Dateiende abfragen und danach ins Hauptmenue zurückspringen:

```
2070 IF a$<>"x" THEN GOTO 2120

2080 PRINT"Dateiende erreicht!"

2090 PRINT

2100 INPUT"Menue=ENTER";x$

2110 GOTO 2999
```

Der INPUT in 2100 ist nur dazu da, um die Ausgabe solange auf dem Bildschirm festzuhalten, bis der Benutzer sie gelesen hat. Sie werden eine unangenehme Überraschung

erleben, wenn Sie das Programm laufen lassen, denn es wird beim ersten Suchen reibungslos laufen, beim zweiten mit einer Fehlermeldung abbrechen. Da wir beim ersten Suchen bis zum Dateiende lesen, müssen wir erst unseren Lesezeiger wieder auf den Beginn der DATA-Zeilen setzen, bevor wir nach einer weiteren Information suchen.

```
2010 RESTORE
```

Der zweite Teil des Programms ist mit dem ersten identisch – bis auf zwei Änderungen:

```
3010 RESTORE

3020 CLS

3030 PRINT:PRINT

3040 INPUT"Nach welchem Titel soll
          gesucht werden";such$

3050 PRINT

3060 READ a$,b$

3070 IF a$<>"x" THEN GOTO 3120

3080 PRINT"Dateiende erreicht!"

3090 PRINT

3100 INPUT"Menue=ENTER";x$

3110 GOTO 3999

3120 IF b$<>such$ THEN GOTO 3060

3130 PRINT"Interpret: "a$

3140 PRINT"Titel:     "b$

3150 PRINT

3160 GOTO 3060
```

In der Zeile 3040 füllen wir den Suchstring mit dem Titel, nach dessen Interpreten wir suchen, und müssen deshalb nicht die erste Information, die wir einlesen (a$) mit unserem such$ vergleichen, sondern die zweite, in der der Titel steht.

Der nächste Programmpunkt, den wir uns vornehmen, ist »Liste drucken«. Da ich davon ausgehe, daß die meisten noch keinen Drucker besitzen, gebe ich die Liste auf den Bildschirm aus. Grundsätzlich sind die Probleme bei Drucker und Bildschirm die gleichen: Wir müssen dem Benutzer die Möglichkeit geben, jederzeit ins Hauptmenue zurückzukehren, und den Bildschirm löschen, wenn er vollgeschrieben ist. Dafür können wir uns die Abfrage auf einen bestimmten Inhalt (außer Dateiende) sparen:

```
4010 RESTORE

4020 zaehler=0

4030 CLS

4040 PRINT

4050 READ a$,b$

4060 IF a$<>"x" THEN GOTO 4110

4070 PRINT"Dateiende erreicht!"

4080 PRINT

4090 INPUT"Menue=ENTER";x$

4100 GOTO 4999

4110 PRINT"Interpret: "a$

4120 PRINT"Titel:      "b$

4130 PRINT

4140 zaehler=zaehler + 1

4150 IF zaehler < 6 THEN GOTO 4050
```

```
4160 zaehler=0

4170 PRINT

4180 INPUT"<m>=Menue <ENTER>=weiter";x$

4190 IF x$="m" THEN GOTO 4999

4200 CLS

4210 GOTO 4050
```

Die Variable »zaehler« wird nach jeder Ausgabe um eins hochgesetzt. Wenn 6 Schallplatten auf dem Bildschirm stehen, wird sie wieder auf Null zurückgesetzt und der Bildschirm für die nächste Ausgabe vorbereitet.

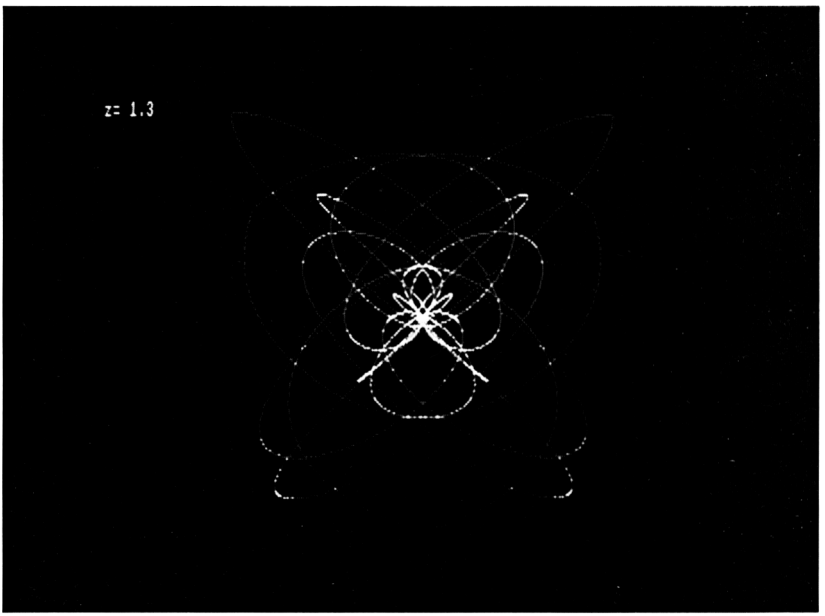
Das einzige, was uns in diesem Programm noch fehlt, ist das Programmende. Wir haben zwar eine Zeile 5999 END; so etwas kann aber bei einer Dateiverwaltung unangenehm werden, weil wir uns nicht vor Falscheingaben schützen können. Also gehört in jedes anständige Programm vor das Ende eine Abfrage, die den Benutzer vor unbeabsichtigtem Programmende schützt.

```
5010 CLS

5020 PRINT:PRINT

5030 INPUT"Sind Sie sicher j/n";x$

5040 IF x$<>"j" THEN GOTO 1000

5050 CLS

5060 PRINT"Tschuess..."
```

Für den Anfang ist das Programm, das wir gerade geschrieben haben, sehr schön. Mich stören noch folgende Punkte:

1. Der Inhalt des Suchstrings muß absolut identisch sein mit dem Inhalt der DATA-Zeile;
2. die Eingabe im Hauptmenue kann eine Redo-Meldung erzeugen;
3. die Ausgabe auf dem Bildschirm könnte wesentlich schöner sein.

Zu 1. Beschäftigen Sie sich mit den Beispielprogrammen zu UPPER$ und MID$
Zu 2. Die Funktionen INKEY$ und VAL helfen weiter.
Zu 3. Mit den Befehlen LOCATE und WINDOW können Sie den Bildschirm verwalten.



Mach's noch einmal, Sam!

Die Datenverarbeitung wird hauptsächlich in zwei Bereichen eingesetzt: Wenn eine Aufgabe zu viel Rechenarbeit erfordert, um sie in einem vertretbaren Zeitraum mit der Hand zu lösen, und wenn eine Aufgabe so gestaltet ist, daß sie mit immer anderen Daten den selben Verarbeitungsweg durchlaufen muß. Ein einfaches Beispiel dafür ist das 1×1. Um das Einmaleins von sieben auf dem Bildschirm erscheinen zu lassen, gibt es mehrere Wege:

```
10 PRINT "1 * 7 ="1*7

20 PRINT "2 * 7 ="2*7

30 PRINT "3 * 7 ="3*7

40 PRINT "4 * 7 ="4*7

50 PRINT "5 * 7 ="5*7

60 PRINT "6 * 7 ="6*7

70 PRINT "7 * 7 ="7*7

80 PRINT "8 * 7 ="8*7

90 PRINT "9 * 7 ="9*7

100 PRINT "10 * 7 ="10*7
```

Sieht nicht sehr effizient aus, nicht wahr? Der zweite Weg ist schon besser:

```
10 a = a + 1

20 IF a>10 THEN END

30 PRINT a "*" 7 "=" a*7

40 GOTO 10
```

Noch bequemer geht es mit einem neuen Befehl:

```
10 FOR a=1 TO 10

20 PRINT a "*" 7 "=" a*7

30 NEXT a

40 END
```

Wenn der Computer auf solch eine Konstruktion mit FOR stößt, sucht er zuallererst nach einem NEXT. Hinter NEXT kann, muß aber in den meisten Fällen nicht, der Name der gleichen Variablen stehen, die hinter FOR steht. Benutzen Sie einen anderen Namen, kommt es zu einer Fehlermeldung:

```
Unexpected NEXT
```

Lassen Sie das NEXT vollkommen weg, erhalten Sie eine andere Meldung:

```
NEXT missing
```

Will man dem Computer auf deutsch die Befehle geben, die in Zeile 10 stehen, müßte man folgendermaßen vorgehen:

1. Weise der Variablen a den Wert 1 zu.
2. Überprüfe, ob der Inhalt von a größer ist als 10.
3. Wenn das der Fall ist, gehe zur Zeile 40.
4. Führe alle Befehle zwischen dem FOR und dem NEXT aus.
5. Addiere 1 zu dem Inhalt von a.
6. Mache weiter bei 2.

Sie können mit FOR..NEXT alle Vorgänge steuern, die in einer vorhersehbaren Anzahl ablaufen. Zusätzlich haben Sie noch eine Erweiterung des Befehls, nämlich STEP, die

dem Computer sagt, was er bei Punkt 5 zu a addieren soll. Die Arbeitsweise des FOR-..NEXT können Sie sich mit einem kleinen Porgramm verdeutlichen:

```
10 INPUT"Anfangswert ";anfang

20 INPUT"Endwert     ";ende

30 INPUT"Schrittweite";schritt

40 a=0

50 FOR i=anfang TO ende STEP schritt

60 a=a+1

70 PRINT "Beim" a ". Durchlauf hat i den Wert" i

80 NEXT i

90 IF a=0 THEN PRINT "Die Schleife wurde nicht
durchlaufen"

100 PRINT "Nach dem Ende der Schleife hat i den
Wert" i

110 GOTO 10
```

Nach einigen Versuchen werden Sie feststellen, daß der Wert von i nach dem Verlassen der Schleife vom Endwert verschieden ist. Das liegt daran, daß bei Punkt 2 nicht auf Gleichheit, sondern auf größer untersucht wird. Nur wenn eine Schleife nicht durchlaufen wird, ändert sich der Wert von i nicht. Sie können statt i eine beliebige numerische Variable nehmen. Wenn Sie eine Schrittweite von 0 benutzen, programmieren Sie damit eine Endlosschleife. Häufig benützt man die Variable von FOR..NEXT innerhalb der Schleife, wie in unserem 1×1, um damit zu rechnen. Es kann allerdings zu seltsamen Ergebnissen führen, wenn man den Inhalt der Variablen zwischen FOR und NEXT ändert. Fügen Sie in das Programm noch eine Zeile 65 ein:

```
65 INPUT"Neuer Wert von i";i
```

Selbst der beste Programmierer weiß nicht, wieviel Schleifendurchläufe der Computer verarbeiten muß. Ein häufiger Einsatz von FOR..NEXT ergibt sich bei der Erstellung von Graphiken. Um einen Baum aus Sternchen zu zeichnen, könnte man so vorgehen:

```
10 CLS

20 FOR i=0 TO PI STEP 0.2

30 ende = 20 - 20 * SIN(i)

40 FOR j=1 TO ende

50 PRINT" ";

60 NEXT j

70 ende = 40 * SIN(i)

80 FOR j=1 to ende

90 PRINT"*";

100 NEXT j

110 PRINT

120 NEXT i

130 FOR i=1 TO 7

140 FOR j=1 TO 17

150 PRINT" ";

160 NEXT j

170 FOR j=1 TO ende

180 PRINT"*";

190 NEXT j
```

```
200 PRINT

210 NEXT i
```

In diesem kleinen Programm sind einige Sachen neu: Sie arbeiten mit zwei Schleifen, und Sie benutzen eine Funktion (SIN(i)). Ferner finden Sie in diesem Programm zwei Hauptschleifen: Die erste läuft von Zeile 20 bis 120 (sie ist für die Krone verantwortlich) und die zweite von Zeile 130 bis 210 (in dieser wird der Stamm geschaffen). In jeder dieser Hauptschleifen sind wieder zwei kleinere eingebettet; die jeweils erste davon sorgt dafür, daß das erste Sternchen in einer Zeile gebührend vom Rand entfernt ist, während die zweite die richtige Anzahl Sternchen ausdruckt. Die PRINT-Befehle in den Zeilen 110 und 200 schließlich setzen den Cursor auf den Beginn der nächsten Zeile. Die jeweils erste Schleife innerhalb der Hauptschleife kann man durch einen einzigen Druckbefehl ersetzen:

```
50<CR>

60<CR>

40 PRINT TAB(ende);

150<CR>

160<CR>

140 PRINT TAB(ende);
```

Auch die anderen kleinen Schleifen kann man einfacher programmieren, indem man eine Funktion benutzt, die aus einem Zeichen (unserem '*') eine ganze Kette von Sternchen machen kann:

```
STRING$(zahl,Zeichen)
```

Mit diesen beiden Angaben liefert diese Funktion eine Zeichenkette, die genau »zahl« Zeichen lang ist und aus lauter »Zeichen« besteht.

```
80 PRINT STRING$(ende,"*")

90<CR>

100<CR>
```

```
110<CR>

170 PRINT STRING$(ende,"*")

180<CR>

190<CR>

200<CR>
```

Wenn Sie glauben, daß damit die Kürzungsmöglichkeiten erschöpft sind, täuschen Sie sich:

```
30 PRINT TAB(20-20*SIN(i));STRING$(40*sin(i),"*")

40<CR>

70<CR>

80<CR>

140 PRINT TAB(17);STRING$(6,"*")

170<CR>
```

Benützen Sie nun RENUM, um die Zeilennumerierung in Ordnung zu bringen:

```
RENUM<CR>
```

und Sie stellen fest, daß aus einem Programm von 21 Zeilen eins von 7 Zeilen geworden ist. Da kein einziges IF in diesem Programm ist, könnten Sie das ganze Programm bequem in einer Zeile unterbringen, weil beim Schneider eine Programmzeile bis zu 255 Zeichen lang sein darf. Können Sie beim Lesen folgender Zeile herausfinden, was nach RUN geschieht?

```
10 CLS:FOR i=1 TO 20:PRINT TAB(21-i);

STRING$(2*i,"*"):NEXT i :FOR i=1 TO 3:

PRINT TAB(18);STRING$(6,"*"):NEXT i
```

Wo wir schon einmal beim Programmkürzen sind – haben Sie noch das Sammlerprogramm von Seite 30? In dieses Programm setzen wir eine Zeile ein:

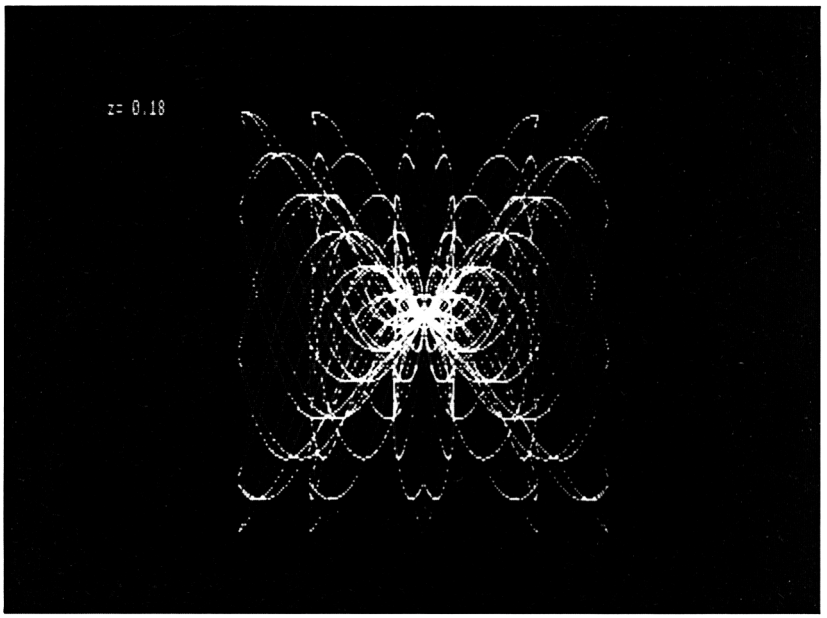
```
991 READ a$,b$:if a$<>"x" THEN zahl=zahl+1:GOTO 991
```

Wenn der Computer diese Zeile verarbeitet hat, steht in der Variablen »zahl« genau die Zahl unserer Schallplatten. Statt immer auf Dateiende zu untersuchen, können wir nun diese Zahl in einer FOR..NEXT-Schleife einsetzen. Als Beispiel die entsprechenden Änderungen im zweiten Teil:

```
3000 'Suchen nach Titel

3010 RESTORE

3020 CLS

3030 PRINT:PRINT

3040 INPUT"Nach welchem Titel soll

          gesucht werden";such$

3050 PRINT

3055 FOR i=1 TO zahl

3060 READ a$,b$

3120 IF b$<>such$ THEN GOTO 3160

3130 PRINT"Interpret: "a$

3140 PRINT"Titel:     "b$

3150 PRINT

3160 NEXT i

3170 PRINT"Dateiende erreicht!"

3180 PRINT
```

```
3190 INPUT"Menue=ENTER";x$

3999 GOTO 1000
```

Nicht nur, daß das Programm kürzer geworden ist, es ist auch übersichtlicher. Führen Sie in den anderen Programmteilen die entsprechenden Änderungen selbst durch.



Immer diese Abhängigkeiten

Schauen Sie sich in dem Sammlerprogramm die Zeilen 1110 bis 1140 an:

```
1110 IF wahl = 1 THEN GOTO 2000

1120 IF wahl = 2 THEN GOTO 3000

1130 IF wahl = 3 THEN GOTO 4000

1140 IF wahl = 4 THEN GOTO 5000
```

Es gibt einen Befehl, der diese 4 Zeilen in eine zusammenfassen kann:

```
1110<CR>

1120 ON wahl GOTO 2000,3000,4000,5000

1130<CR>

1140<CR>
```

Abhängig von »wahl« verzweigt das Programm in die Zeile 2000, 3000, 4000 und 5000. Wenn der Benutzer eine Zahl eingibt, die größer ist als 4, dann arbeitet der Computer in der Zeile 1150 weiter. Das gleiche geschieht, wenn eine Null eingegeben bzw. nur die ENTER-Taste gedrückt wird. Bei einer gebrochenen Zahl wird erst gerundet und dann abhängig von dem Rundungsergebnis in eine Programmzeile verzweigt. Eine Eingabe 2.7 verzweigt nach 4000, während 2.3 nach 3000 verzweigt (aber das ist ja im Moment nicht tragisch). Unangenehm wird die Sache bei der Eingabe negativer Zahlen; dann bricht das Programm mit einer Fehlermeldung ab:

```
Improper argument in 1120
```

Damit das nicht passiert, müssen wir negative Zahlen vorher abweisen:

```
1110 IF wahl<0 THEN GOTO 1000
```

Diesen ON-Befehl gibt es auch noch in einem anderen Zusammenhang. Doch das ist eine andere, spätere Geschichte ...

Geteiltes Leid...

Bei dem Schreiben des Sammlerprogramms haben Sie sich bestimmt darüber geärgert, daß Sie einige Programmzeilen zweimal, manche sogar dreimal schreiben mußten. Die Zeilen zum Ausdrucken einer Schallplatte kamen in jedem Programmteil außer dem Schluß einmal vor. Wenn man einen eigenen Programmteil schriebe:

```
6000 PRINT"Interpret: "a$

6010 PRINT"Titel:     "b$

6020 ON wahl GOTO 2140,3140,4120
```

könnte man diesen Teil aus Zeile 2130, 3130, 4120 mit GOTO 6000 aufrufen. Die Sache hat einen Haken: Stellen Sie sich ein ähnliches Programm vor, in dem Sie keine Variable »wahl« zur Verfügung haben; dann könnte es zu Schwierigkeiten führen, dem Computer klarzumachen, in welche Zeile er zurückspringen soll. Wir haben in unserem Befehlsvorrat einen Befehl, der den Programmfluß in eine bestimmte Zeile lenkt und einen anderen, der dafür sorgt, das genau wieder an dieser Stelle zurückgesprungen wird. Der erste heißt:

```
GOSUB
```

Danach muß eine existierende Zeilennummer stehen. Also z. B.

```
GOSUB 1000
```

Zu einem GOSUB gehört immer ein:

```
RETURN
```

Das ist der Befehl, der das Programm genau wieder an die Stelle zurückschickt, die es durch das GOSUB verlassen hatte. Der GOSUB-Befehl hat zwei große Vorteile: Daß er Schreibarbeit spart, habe ich schon angedeutet; der zweite Vorteil liegt in der Tatsache begründet, daß selbst ein guter Programmierer Schwierigkeiten hat, sich in einem Programm zurechtzufinden, das aus einem Block besteht. Es ist wesentlich vernünftiger, erst eine Steuerung für das Programm zu schreiben und dann die einzelnen Teile fertigzustellen. Als Beispiel noch einmal unser Schallplattenprogramm:

```
10 GOSUB 100:'Vorbereitung

20 GOSUB 1000:'Hauptmenue

30 ON wahl GOSUB 2000,3000,4000,5000

40 GOTO 20

1000 'Hauptmenue

1999 RETURN

2000 'Suchen nach Interpret

2999 RETURN

3000 'Suchen nach Titel

3999 RETURN

4000 'Ende

4999 RETURN

5000 'Ausgabe Schallplatte

5999 RETURN
```

Jetzt können Sie sich daran machen, die einzelnen Programmteile fertigzustellen. Höchstwahrscheinlich wird sich ihr Computer mit einer Fehlermeldung verabschieden, weil Sie alle Befehle GOTO 1000 durch RETURN ersetzt haben. Machen wir uns an einem Beispiel klar, was alles geschehen kann:

```
10 GOSUB 10
```

Das ist der kürzeste Weg, um Ihren Computer lahm zu legen. Wenn der Computer in diese Zeile läuft, wird er zu der Zeile springen, die hinter dem GOSUB steht. Davor wird er sich aber die Stelle merken, von der er kommt (man spricht in solchen Fällen von der »Adresse«). Er wird in Zeile 10 weiterarbeiten, sich dort als erstes die Adresse merken, in Zeile 10 weiterarbeiten, und so weiter – bis er seinen ganzen für solche Zwecke vorgesehenen Speicherplatz verbraucht hat. Danach:

```
Memory full in 10
```

Einen anderen Weg, Ihr Programm unsanft zu beenden, bietet folgendes Programm:

```
10 GOSUB 20

20 PRINT"Hallo, ich bin die Zeile 20!"

30 RETURN
```

Da der Computer bei der Rückkehr aus einem Unterprogramm in seinem Gedächtnis nachsehen muß, wohin er zurückkehren soll, muß er jede Adresse, die schon erledigt ist, vernichten (löschen). Wenn er auf ein RETURN stößt und es keine Adresse mehr gibt, zu der er zurückkehren kann – Fehlermeldung und Ende:

```
Unexpected RETURN in 30
```

Vor allen Dingen dann, wenn Sie aus einem Unterprogramm ein weiteres Unterprogramm aufrufen, womöglich von diesem aus noch ein drittes, kann es zu Fehlern kommen. Der Anfänger versucht häufig, aus einem Unterprogramm mit einem GOTO herauszukommen, und ärgert sich später grün, wenn das Programm nicht läuft.

Der einzige Weg, so etwas zu vermeiden, führt über eine vernüftige Dokumentation. Sparen Sie deshalb bei der Erstellung eines Programmes nicht mit REM-Zeilen.

Die Schalter, die die Welt bedeuten

Was denkt der Schneider, wenn er denkt? Das Ganze ist eine Angelegenheit von acht Schaltern. Diese allerdings in 65536facher Ausfertigung. Wir haben also 65536 Gruppen von jeweils 8 Schaltern. Eine solche Gruppe heißt Byte, und jeder Schalter heißt Bit. Wenn Sie im Gedächtnis des Schneiders etwas ändern wollen, können Sie das nur, indem Sie eine Gruppe ansprechen, also ein Byte. Der Befehl dazu lautet:

POKE

Hinter dem POKE steht die Nummer der Gruppe, die Sie ansprechen wollen, die sogenannte Adresse. Alle Bytes sind durchnumeriert mit den Zahlen von 0 bis 65535. Außerdem steht nach der Adresse, von dieser durch ein Komma getrennt, die Stellung der einzelnen Schalter. Diese Stellung kann man durch eine Zahl von 0 bis 255 ausdrücken. Und jetzt kommt die Hauptfrage:

Wie kann man mit 8 Schaltern die Zahlen von 0 bis 255 ausdrücken? Wir fangen ganz klein an und betrachten erst einen Schalter. Nehmen wir an, daß ein Schalter, der auf AN steht, die Zahl 1 repräsentiert. Der gleiche Schalter, auf AUS gestellt, steht für 0. Also kann man mit einem Bit die Zahlen 0 und 1 ausdrücken. Nimmt man nur ein weiteres Bit dazu, verdoppelt sich die Zahl der Möglichkeiten:

0
1
10
11

Das entspräche unseren Zahlen 0 bis 3. Ein weiteres Bit und wir erhalten die Zahlen 0 bis 7:

100
101
110
111

Bei jedem weiteren Bit verdoppeln sich wieder die bisherigen Möglichkeiten. Es gibt also:

1 Bit	2 Möglichkeiten	0–1
2 Bit	4 Möglichkeiten	0–3
3 Bit	8 Möglichkeiten	0–7
4 Bit	16 Möglichkeiten	0–15
5 Bit	32 Möglichkeiten	0–31
6 Bit	64 Möglichkeiten	0–63
7 Bit	128 Möglichkeiten	0–127
8 Bit	256 Möglichkeiten	0–255

Damit ist die Zahl der möglichen Schalterstellungen festgehalten. Wie aber können wir aus einer Schalterstellung (betrachten wir von jetzt an immer die gesamte Gruppe) die entsprechende Zahl herausfinden? Da kommen wir nicht ganz ohne Mathematik aus – keine Bange, es bleibt einfach. Zur Erinnerung:

$1 = 10^0 = 1$
$10 = 10^1 = 10$
$10 * 10 = 10^2 = 100$
$10 * 10 * 10 = 10^3 = 1000$

Die Zahl 234 ließe sich also auch folgendermaßen ausdrücken:
$4*1+3*10+2*100+0*1000+5*10000$; oder:

$$\begin{aligned} &4*10^0 \\ +&3*10^1 \\ +&2*10^2 \\ +&0*10^3 \\ +&5*10^4 \end{aligned}$$

Sie merken, welch wichtige Rolle die 10 in dieser Rechnung spielt: Sie ist die »Basis« unseres Zahlensystems. Die Basis des Zahlensystems, mit dem unser Schneider rechnet, ist die 2. Vergessen Sie nicht, daß wir in unserem Zahlensystem keine Möglichkeit haben, die Zahl 10 mit *einer* Ziffer auszudrücken. Genausowenig kann man in einem Zahlensystem, das auf der Basis 2 beruht, die Zahl 2 mit *einer* Ziffer ausdrücken. Statt dessen brauchen wir wie im Zehnersystem immer dann, wenn wir eine Zahl beim Zählen erreicht haben, die sich genau durch 2 hoch irgendwas ausdrücken ließe, ein weiteres Bit.

Jetzt kommen wir der Sache schon etwas näher: Das Bit, das ganz rechts steht, steht für 2 hoch 0 und nennt sich deshalb Bit Nr. 0, das nächste weiter links steht für 2 hoch 1 und heißt Bit 1; die Bits sind also von rechts nach links durchnumeriert mit den Zahlen von 0 bis 7. Mit dieser Zahl ist also die Zahl gemeint, mit der man 2 potenzieren muß, um den Wert dieses Bits zu erhalten. Der Rest ist ganz einfach; schauen wir uns ein Byte an, dessen Inhalt folgendermaßen aussieht:

1 0 1 1 1 0 0 1

Diese Zahl bedeutet von rechts nach links:
1 mal 2 hoch 0 = 1
0 mal 2 hoch 1 = 0
0 mal 2 hoch 2 = 0
1 mal 2 hoch 3 = 8
1 mal 2 hoch 4 = 16
1 mal 2 hoch 5 = 32
0 mal 2 hoch 6 = 0
1 mal 2 hoch 7 = 128

Addiert man die Zahlen auf der rechten Seite der Gleichheitszeichen zueinander, so erhält man als Entsprechung unserer Schalterstellung eine Zahl, die auch Otto Normalverbraucher verständlich ist, nämlich 185.

Wollen wir mit unserem POKE-Befehl das Byte mit der Nummer 1 auf den Wert 185 bringen, so müssen wir schreiben:

```
POKE 1,185
```

Es ist bei unserem Schneider möglich, die Schalterstellung der einzelnen Bits direkt einzugeben. Dazu muß man vor der Schalterstellung die Zeichen &x setzen. (Im normalen Sprachgebrauch redet man natürlich nicht von Schalterstellung, sondern von Dual- oder Binärzahlen.) Also könnten wir unseren Befehl auch so formulieren:

```
POKE 1,&x10111001
```

Wir wissen bis jetzt, daß in ein Byte die Zahlen von 0 bis 255 hineinpassen. Wie aber werden Zahlen ausgedrückt, die größer sind als 255? Ganz einfach – man nimmt noch ein Byte dazu. Damit hat man 16 Bits (Schalter) zur Verfügung, und mit diesen 16 Bits kann man Zahlen von 0 bis 65535 ausdrücken. Einige Neugierige unter Ihnen werden fragen, wie es denn mit negativen Zahlen aussieht? Leider nicht mehr ganz so einfach.

Um die zu einer Dezimalzahl gehörende Dualzahl auf dem Bildschirm sichtbar zu machen, haben wir auf dem Schneider eine eigene Funktion:

```
PRINT BIN$ (185)
```

Sie werden voller Wiedersehensfreude unsere Schalterstellung erkennen: 10111001. Versuchen Sie diese Funktion mal mit einer negativen Zahl z. B. –2. Das Bild auf Ihrem Monitor sieht folgendermaßen aus: 1111111111111110. Nach alldem, was Sie bisher über Dualzahlen gelernt haben, muß das eine ziemlich große Zahl sein (für diejenigen, die es genau wissen wollen: 65534). Das ist auch so. Prüfen Sie es nach:

```
PRINT BIN$ (65534)
```

Auf Ihrem Bildschirm erscheint die gleiche Zahlenkette wie bei –2. Die Erklärung liegt in der Art begründet, wie negative Zahlen in der Dualarithmetik ausgedrückt werden. Negative Zahlen werden durch das »Zweierkomplement« ihres positiven Partners gebildet. Um dahin zu gelangen, müssen wir uns erstmal das Einerkomplement einer Zahl anschauen.

Das Einerkomplement ist schnell gebildet: Jede Eins innerhalb einer Dualzahl wird durch eine Null ersetzt; jede Null durch eine Eins. Da der Computer zur Darstellung einer negativen Zahl immer zwei Byte braucht, betrachten wir das Einerkomplement der Zahl 127, davor die Zahl dargestellt in zwei Byte: 0000000001111111. Das Einerkomplement dazu lautet also: 1111111110000000. Wie kommt man vom Einerkomplement zum Zweierkomplement? Ganz einfach, indem man auf das Einerkomplement 1 aufaddiert. Unser Zweierkomplement zu 127 heißt also: 1111111100000001. Sie können es auch gerne nachprüfen:

PRINT BIN$ (−127)

Das Ergebnis wird Ihnen bekannt vorkommen. Jede negative Dualzahl hat also für unseren Computer zwei Bedeutungen: Erstens bedeutet es eine negative Zahl zwischen −1 und −32768 und zweitens eine positive Zahl zwischen 32768 und 65535.

Das Rechnen mit dualen Zahlen ist einfach: Wir brauchen uns nur die Addition zu betrachten; alles weitere nimmt uns in BASIC der Computer ab.

Fangen wir wieder mit uns bekannten Zahlen an:

$$\begin{array}{r} 99 \\ +\ 1 \\ \hline 100 \end{array}$$

In dem Augenblick, in dem unsere zwei Stellen nicht mehr ausreichen, nehmen wir automatisch eine dritte zu Hilfe. Diese dritte Stelle sowie jede weitere, die auf diese Art gebraucht wird, nennen wir Übertrag. Ein Übertrag wird jedesmal dann gebildet, wenn bei der Addition das Ergebnis mit der Stellenzahl nicht ausgedrückt werden kann, die bis jetzt benutzt wurde. Erinnern wir uns: Mit einer Stelle können wir bei Dualzahlen nur die Zahlen 0 und 1 ausdrücken. Das heißt, daß jede Addition von einer 1 auf eine 1 einen Übertrag auslöst. Insgesamt lassen sich bei der Addition von 2 einstelligen Dualzahlen folgende Ergebnisse vorstellen:

$$\begin{array}{cccc} 1 & 1 & 0 & 0 \\ \underline{1} & \underline{0} & \underline{1} & \underline{0} \\ 10 & 01 & 01 & 00 \end{array}$$

Bei zweistelligen (das gleiche gilt auch bei mehrstelligen) Zahlen muß man unter Umständen den Übertrag im Kopf behalten. Sie kennen das; früher haben Sie die Aufgabe 78 + 45 = so gerechnet:

8+5=13, davon haben Sie die 3 aufs Papier geschrieben und die 1 als Übertrag im Kopf behalten. Als nächstes haben Sie 7+4 addiert und zu dem Ergebnis das hinzugezählt, was Sie noch im Kopf hatten. Auf dem Papier stand als Ergebnis 123. Das ganze der Reihe nach nochmal als Zahlenbeispiel:

$$\begin{array}{ccc} & 1 & \\ 78 & 78 & 78 \\ \underline{45} & \underline{45} & \underline{45} \\ & 3 & 123 \end{array}$$

Genauso rechnet man mit Dualzahlen:

```
                1       1       1
 101          101     101     101       101
 111          111     111     111       111
 ---          ---     ---     ---       ----
   .            0      00     100      1100
```

Zur Probe: 101(Dezimalzahl 5)+111(Dezimalzahl 7)=1100(Dezimalzahl 12)

Ein wichtiges Kapitel im Umgang mit Dezimalzahlen haben wir bis jetzt noch gar nicht erwähnt – nämlich die Operationen NOT, AND, OR und XOR. Diese Operationen helfen uns, einzelne Bits eines Bytes auf einen bestimmten Wert zu bringen. Beginnen wir mit der einfachsten, nämlich dem NOT. Wenden wir diese Operation auf eine Binärzahl an, so sorgt sie dafür, daß alle Stellen dieser Zahl in das Gegenteil verkehrt werden. Aus der Zahl 1100 wird dann also 0011. Man kann verallgemeinert sagen: wenn für eine Stelle einer Zahl gilt, daß sie 1 ist, so wird sie 0, ist sie 0, so wird sie 1.

Die eben erwähnten Operatoren haben noch eine zweite Wirkung: Man kann damit Aussagen bearbeiten. Eine Aussage ist auch für einen Computer eine Information mit einem Wahrheitsgehalt – sie kann entweder wahr oder falsch sein. Angenommen, wir füllen die Variable A mit 3 (A=3), dann führt die Programmzeile

```
IF A=3 THEN PRINT"Schneider" ELSE PRINT"Computer"
```

zum Ausdruck des Wortes »Schneider«. Negieren wir die Aussage, schreiben wir also

```
IF NOT(A=3) THEN PRINT"Schneider"
           ELSE PRINT"Computer"
```

so wird das Wort »Computer« ausgedruckt, weil die Aussage NOT(A=3) falsch ist.

Bezeichnet man eine wahre Aussage mit dem Wert 1, eine falsche mit dem Wert 0 und die Aussage mit dem Namen X, so ergibt sich als Ergebnis einer NOT-Operation folgendes Bild:

X	Z = NOT X
0	1
1	0

Wesentlich häufiger als der Operator NOT wird der Operator AND eingesetzt. Dieser Operator verknüpft zwei Dualzahlen so miteinander, daß im Ergebnis eine Stelle nur dann auf 1 steht, wenn bei beiden Ausgangszahlen genau diese Stelle auf 1 stand. Die Verknüpfung von 1110 und 0011 durch ein AND ergibt also 0010. AND, auf zwei Aussagen mit den Namen X und Y angewandt, führt zum folgenden Ergebnis:

X	Y	Z = X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

Zwei durch AND verknüpfte Aussagen sind also nur dann wahr, wenn beide wahr sind. Das Gleiche in BASIC:

```
10 INPUT"1. Zahl";a

20 INPUT"2. Zahl";b

30 IF a=1 AND b=2 THEN PRINT"Das war brav!"

40 GOTO 10
```

Ein weiterer, sehr häufig benutzter Operator ist das OR. Es verknüpft zwei Dualzahlen so miteinander, daß im Ergebnis eine Stelle immer dann auf 1 steht, wenn in einer oder beiden Ausgangszahlen genau diese Stelle auf 1 stand. Die Verknüpfung von 1001 und 0101 durch OR führt zu 1101. OR auf zwei Aussagen bezogen, gibt:

X	Y	Z = X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

Zwei durch OR verknüpfte Aussagen sind immer dann wahr, wenn mindestens eine der beiden Aussagen wahr ist. Damit wir BASIC nicht ganz vergessen:

```
10 INPUT"1. Zahl";a

20 INPUT"2. Zahl";b

30 IF a=1 OR b=2 THEN PRINT"Mindestens eine war

richtig!"

40 GOTO 10
```

Der letzte Operator dieser Art, den Sie beim Schneider benützen können, ist das XOR – die Abkürzung für »Exclusiv Oder«. Auf Dualzahlen bezogen, führt er bei der Verknüpfung von zwei Zahlen dazu, daß im Ergebnis eine Stelle nur dann auf 1 steht, wenn ausschließlich in einer der beiden Ausgangszahlen diese Stelle auf 1 stand. Also ergibt 1101 XOR 1011 als Ergebnis 0110. Bezogen auf zwei Aussagen X und Y, erhalten wir die Tabelle:

X	Y	Z = X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

Zwei durch XOR verknüpfte Aussagen sind nur dann wahr, wenn nur eine Aussage wahr ist. Und nochmal BASIC:

```
10 INPUT"1. Zahl";a

20 INPUT"2. Zahl";b

30 IF a=1 XOR b=2 THEN PRINT"Schoen, wenn einer

reicht!"

40 GOTO 10
```

Schauen wir uns an, was wir mit diesen Operatoren mit dem Inhalt eines Bytes anstellen können. Tippen Sie bitte dieses kleine Programm ab (die fehlende Zeile 60 werden wir von Fall zu Fall einsetzen):

```
10 MODE 2

20 INPUT"Neuer Inhalt";a

30 IF a<0 OR a>255 THEN 10

40 PRINT BIN$(a,8),a

50 POKE 53000,a

60 diese Zeile wird ersetzt
```

```
70 PRINT BIN$(b,8),c

80 POKE 53000,b

90 IF INKEY$="" THEN 90

100 GOTO 10
```

Versuchen wir es mit NOT:

```
60 b=NOT a
```

Egal, welche Zahl wir auf die Frage »Neuer Inhalt?« eingeben, das Programm bricht mit der Fehlermeldung »Improper Argument« ab, weil ein Byte zu klein ist, um eine negative Zahl aufzunehmen. Als nächstes das AND:

```
60 b=a AND &x11110000
```

Das entspricht der Dezimalzahl 240. Egal, welche Zahl Sie jetzt eingeben, die letzten 4 Stellen werden bei der zweiten Zahl auf dem Bildschirm 0 sein: So können Sie eine Stelle eines Bytes auf 0 setzen, ohne die anderen Stellen zu verändern. ANDen sie das entsprechende Byte mit 255 minus dem Wert der Stelle, die Sie auf 0 setzen wollen. Nun das OR:

```
60 b=a OR &x00001111
```

(Dezimalzahl 15). Nun wird Ihre zweite Zahl am Schluß immer vier mal die 1 haben: Mit OR können Sie also gezielt eine Stelle eines Bytes auf 1 setzen, ohne den Rest zu verändern. Verknüpfen Sie den Inhalt eines Bytes durch OR mit dem Wert der Stelle, die auf 1 gesetzt werden soll und füllen Sie das Ergebnis wieder in dasselbe Byte. Zum Schluß das XOR:

```
60 b=a XOR &x11000011
```

(Dezimalzahl 195). Vielleicht dauert es eine Zeit, bis Sie merken, daß jetzt die ersten beiden und die letzten beiden Stellen der zweiten Zahl immer das Gegenteil von der Originalzahl zeigen. Mit XOR können Sie also ein Bit umschalten – egal, was vorher darin stand.

Mit Geduld und Spucke werden Sie sich im Innenleben Ihres Schneiders so auskennen wie in Ihrem Schreibtisch.

Wen der Himmel strafen will, ...

(*Bitte in* MODE 1 *eingeben*)
Sie erinnern sich doch noch an meinen Freund, den Maler? Er hat mich wieder besucht. Eigentlich wollte ich ihm gar nicht erst öffnen, aber dann ... Na ja, Sie wissen ja, was man aus Freundschaft alles tut. Bei einem Bier erzählte er mir von seinen Sorgen.

Nach seinen Erfahrungen mit der Datenverarbeitung hatte er einen Verein gegründet, den VDCGA e. V. (Verein der Computergeschädigten Anstreicher). Mir schwante sofort Fürchterliches, und in meinem Kopf regte sich der Gedanke, ihm mit einem anderen Verein Konkurrenz zu machen (VDAGP = Verein der Anstreichergeschädigten Programmierer). Ich nahm aber davon Abstand, als ich von ihm hörte, welche Arbeit solch ein Verein macht. Da mein Freund einen Schneider zur Verfügung hat, hatte man ihm die gesamte Verwaltungsarbeit übertragen. Weit davon entfernt, die Arbeit des Programmierens selbst übernehmen zu wollen, hatte er sich an mich erinnert, und trug mir nun seine Wünsche vor.

Er wollte ein Programm haben, das sein Adressenmaterial verwalten konnte. Ich machte ihm den Vorschlag, es doch mit einem Programm wie dem Sammlerprogramm zu versuchen, doch wollte er nichts davon wissen, da er keine Lust hatte, bei jeder Adressänderung das Programm ändern zu müssen. Im Klartext: er wollte von mir eine Adressverwaltung, bei der die Daten und das Programm voneinander getrennt sein sollten. Da er damit rechnete, daß sein Verein sehr schnell wachsen würde, sollte ich auch nicht allzu großzügig mit dem für das Programm benötigten Speicherplatz umgehen. Das Programm sollte folgende Aufgaben erledigen können:

1. Die Daten sollten von einer Kassette geladen werden können.
2. Man sollte Adressen einfügen können.
3. Es sollte nach einer bestimmten Adresse gesucht werden können.
4. Eine Adresse mußte geändert werden können.
5. Es sollten Adressen aus der Datei gelöscht werden.
6. Alle Adressen sollten der Reihe nach auf dem Monitor erscheinen.
7. Die Adressen sollten gespeichert werden.
8. Ein Programmende mit Warnungen vor möglichen Fehlern.

Na dann, beginnen wir mit dem Menue: Damit wir uns nicht immer mit der ENTER-Taste aufhalten müssen, wenn wir nur ein einziges Zeichen eingeben wollen, nehmen wir dafür eine neue Funktion: INKEY$. Diese Funktion fragt die Tastatur ab und liefert, wenn

eine Taste gedrückt wurde, das entsprechende Zeichen. Allerdings wartet INKEY$ nicht auf das Drücken einer Taste. Deshalb müssen wir selbst dafür sorgen, daß das Programm solange in einer Zeile bleibt, bis eine gültige Taste gedrückt wird. Da INKEY$ einen String liefert, müssen wir, wenn endlich eine Zahlentaste gedrückt wurde, das entsprechende Zeichen in eine Zahl umwandeln. Das geschieht mit der Funktion VAL. Damit haben wir alles zusammen, was wir für unser Hauptmenue brauchen.

```
200 'Hauptmenue
210 MODE 2
220 PRINT"Hauptmenue":PRINT
230 PRINT"<1> Laden"
240 PRINT"<2> Einfuegen"
250 PRINT"<3> Suchen"
260 PRINT"<4> Aendern"
270 PRINT"<5> Loeschen"
280 PRINT"<6> Listen"
290 PRINT"<7> Speichern"
300 PRINT"<8> Programmende"
310 PRINT:PRINT"Bitte eine Zahl von 1 bi
s 8 ";
320 wahl$=INKEY$:IF wahl$="" THEN GOTO 3
20
330 IF wahl$<"1" OR wahl$>"8" THEN GOTO
320
340 wahl=VAL(wahl$)
350 RETURN
```

In der Zeile 350 steht deshalb RETURN, weil wir das gesamte Programm in einzelne Unterprogramme aufteilen wollen, die wir aus einer gemeinsamen Steuerung aufrufen können. Das ist der nächste Programmteil:

```
10 GOSUB 100:'vorbereitung
20 GOSUB 200:'Hauptmenue
30 ON wahl GOSUB 1000,2000,3000,4000,500
0,6000,7000,8000
40 GOTO 20
```

Sie sehen, daß vor dem Hauptmenue noch ein Programmteil steht, in der eine Vorbereitung laufen soll. Die meisten größeren Programme haben solch eine Vorroutine, weil es immer einige Dinge zu erledigen gibt, bevor wir mit der richtigen Arbeit anfangen können. Der entsprechende Programmteil beginnt ab Zeile 100:

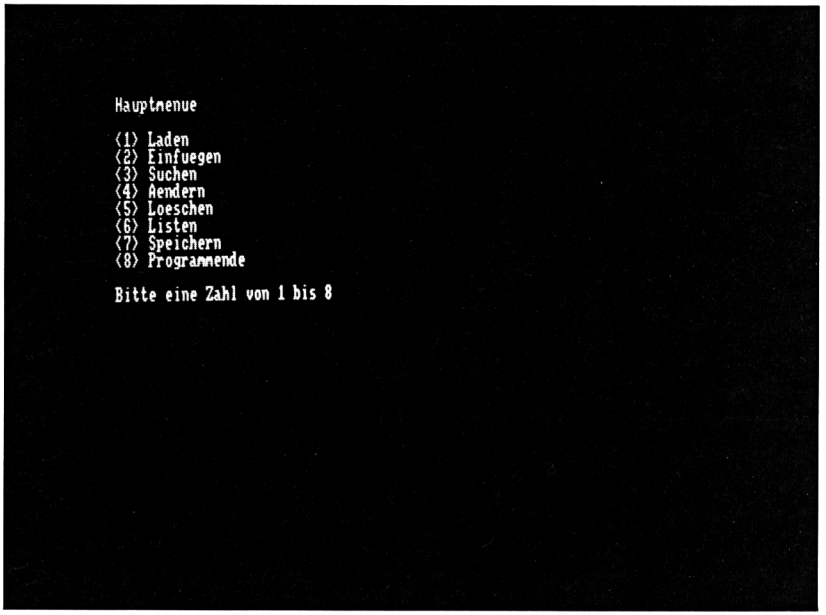
```
100 'Vorbereitung
110 zahl=300
120 DIM adress$(zahl+1),feld$(7),laenge(
7),eingabe$(7)
130 FOR i=1 TO 7
140 READ feld$(i),laenge(i)
150 NEXT i
160 DATA Anrede,8,Vorname,10,Nachname,20
,Strasse,30,PLZ,7,Ort,20,Telefon,15
170 ersatz$=CHR$(143)
180 RETURN
```

Die einzige neue Anweisung steht in der Zeile 120: DIM. Diese Anweisung reserviert im Computer Platz für Variablen. Für einen Anfänger in der Datenverarbeitung ist dieser Befehl etwas schwer zu verstehen.

Bis jetzt brauchten wir für jede Information, die wir in einer Variablen aufheben wollten, eine neue Variable. Alle diese Variablen mußten unterschiedliche Namen haben. Jetzt haben wir die Möglichkeit, diese Informationen im Grunde genommen in Variablen unterzubringen, die sich durch eine zusätzliche Nummer unterscheiden. Wir haben in unserem Fall gleichartige Informationen, nämlich Adressen, die wir in einer Tabelle speichern wollen. Jede einzelne Adresse können wir jetzt ansprechen, indem wir den Namen der Tabelle und die Nummer des entsprechenden Tabellenplatzes nennen. Vorher müssen wir aber dem Schneider sagen, wie groß die Tabelle sein und wieviel Plätze sie haben soll: Genau das sagen wir dem Computer mit dem DIM-Befehl. Dahinter muß der Name der Tabelle stehen und die Anzahl der Tabellenplätze. Die Anzahl der Tabellenplätze habe ich in der Variablen »zahl« gespeichert; das gibt die Möglichkeit, die Zahl der Tabellenplätze nachträglich zu ändern. Unsere Adressen sind in der Tabelle »adress$« gespeichert.

Man kann mit einer DIM-Anweisung mehrere Tabellen dimensionieren. Die Tabelle »feld$« ist zur Aufnahme der sogenannten Feldnamen gedacht; jede Adresse besteht ja aus mehreren Informationen. Um die Namen für diese Informationen nicht jedesmal neu tippen zu müssen, hebe ich sie in der Tabelle »feld$« auf. Jetzt brauche ich nur noch die einzelnen Tabellenplätze auszudrucken und habe genau diese Namen auf dem Bildschirm. In der Zeile 130 und 150 lese ich aus den DATA-Zeilen die Namen in die Tabelle ein. Zusätzlich zu den Feldnamen habe ich noch eine kleine Tabelle, in der aufgehoben wird, wie lang die einzelne Information sein darf; diese Beschränkung ist notwendig, damit alle Adressen gleich lang sind. Die Länge der Einzelinformationen ist in der Tabelle »laenge« gespeichert. Die letzte Tabelle, nämlich »eingabe$«, ist zur zeitweiligen Aufnahme der Teilinformationen zu einer Adresse bei der Eingabe neuer Daten gedacht.

Neu in diesem Programmteil ist auch die Funktion CHR$; diese Funktion erzeugt genau wie INKEY$ ein einziges Zeichen, das durch die Zahl in der Klammer bestimmt



wird. Da alle Zeichen, die dem Schneider zur Verfügung stehen, von 0 bis 255 durchnumeriert sind, kann man jedes einzelne Zeichen über diese Zahl ansprechen. Die Numerierung ist im Bereich von 0 bis 127 durch den sogenannten ASCII-Code genormt (American Standard Code for Information Interchange). Das Zeichen mit der Nummer 143 kann man nicht über die Tastatur erzeugen; es sieht aus wie eine inverse Leerstelle. Dieses Zeichen benutze ich später, um nicht benötigte Stellen in der Adresse zu füllen. Mit den Zeilen von 100 bis 180 haben wir alle nötigen Vorbereitungen getroffen, um uns unserer Adressverwaltung zu widmen.

Der erste Teil, den wir uns vornehmen, ist die Eingabe neuer Adressen. Dabei wird Ihnen auch verständlich, wozu die einzelnen Tabellen benötigt werden. Zuerst der komplette Programmteil.

```
2000 'Einfuegen
2010 CLS
2020 belegt=belegt+1:IF belegt<=zahl THE
N GOTO 2040
2030 PRINT"Datei ist voll!":FOR i=1 TO 2
000:NEXT i:GOTO 2240
2040 PRINT USING"Dateiname: \
   \  Satz Nr. ###";datei$,belegt
```

```
2050 PRINT:PRINT
2060 FOR i=1 TO 7
2070 PRINT USING"\                    \: ";fel
d$(i);
2080 INPUT eingabe$(i)
2090 IF LEN(eingabe$(i)) > laenge(i) THE
N PRINT"Wort zu lang!";CHR$(13);CHR$(11)
;:GOTO 2070
2100 NEXT
2110 PRINT:PRINT"Alles ok? j/n"
2120 i$=UPPER$(INKEY$):IF i$<>"J" AND i$
<>"N" THEN 2120
2130 IF i$="N" THEN belegt=belegt-1:GOTO
 2010
2140 FOR i=1 TO 7
2150 IF LEN(eingabe$(i)) < laenge(i) THE
N eingabe$(i)=eingabe$(i)+ersatz$:GOTO 2
150
2160 NEXT
2170 FOR i=1 TO 7
2180 adress$(belegt)=adress$(belegt)+ein
gabe$(i)
2190 NEXT
2200 PRINT"M=Menue    W=Weiter"
2210 i$=UPPER$(INKEY$):IF i$<>"M" AND i$
<>"W" THEN 2210
2220 IF i$="W" THEN GOTO 2010
2230 aendern=1
2240 RETURN
```

Schauen wir uns die Zeilen der Reihe nach an:

Zeile 2000 dient zur späteren Identifizierung des Programmteiles.

2010 Der Bildschirm wird gelöscht.

2020 Die Anzahl der belegten Plätze in unserer Tabelle wird um 1 erhöht, und es wird überprüft, ob die höchstmögliche Anzahl überschritten ist. Wenn das nicht der Fall ist, wird die nächste Zeile übersprungen.

2030 Information für den Benutzer, daß die Datei voll ist, und Sprung auf das Ende des Unterprogrammes.

2040 Informationen zu dem PRINT USING-Befehl entnehmen Sie bitte dem Wörterbuch »BASIC-Deutsch« Seite 193.

2050 Zwei Leerzeilen werden ausgedruckt.

2060 Die folgende Schleife mit der Laufvariablen i wird siebenmal durchlaufen, da zu jeder Adresse sieben Einzelinformationen gehören (siehe Zeile 160).

2070 Der Name der Einzelinformation wird ausgedruckt.

2080 Der durch den derzeitigen Wert der Laufvariablen bestimmte Tabellenplatz der Tabelle »eingabe$« wird durch INPUT gefüllt.

2090 In dieser Zeile sehen Sie wieder eine neue Funktion: LEN. Diese Funktion ermittelt die Länge eines Strings; der Name des Strings muß in Klammern nach LEN folgen. Hier wird diese Funktion benutzt, um zu prüfen, ob die eingegebene Information die maximale Länge für die Einzelinformation überschreitet. Ist das der Fall, so wird eine Nachricht an den Benutzer ausgegeben und der Cursor durch das Zeichen CHR$(13) an den Beginn der Zeile gesetzt und durch CHR$(11) eine Zeile höher gesetzt. Damit sind wir wieder am Beginn der Zeile, in der der Feldname ausgedruckt wurde. Jetzt springen wir in die entsprechende Zeile. Diese Schleife wird so lange durchlaufen , bis sich der Benutzer bequemt, die Anzahl der zulässigen Zeichen nicht zu überschreiten.

2100 Ende der FOR... NEXT Schleife. Wenn die Schleife verlassen wurde, sind alle Plätze der Tabelle »eingabe$« mit irgendetwas gefüllt – unter Umständen auch mit einem Nullstring, also einem String der Länge 0. Als nächstes müssen wir alle Plätze der Tabelle »eingabe$« auf die maximale Länge bringen, die uns für die Einzelinformation zur Verfügung steht. Das geschieht in der nächsten FOR... NEXT Schleife. Vorher müssen wir allerdings den Benutzer noch fragen, ob er bei der Eingabe der Daten Fehler gemacht hat.

2110 Die Frage »Alles ok? j/n« wird ausgedruckt.

2120 Die Funktion INKEY$ wird benutzt, um die Tastatur abzufragen. Durch die Funktion UPPER$ wird die gedrückte Taste, wenn möglich, in einen Großbuchstaben umgewandelt. Danach wird die Variable »i$« mit dem Ergebnis gefüllt und es überprüft, ob in »i$« entweder N oder J vorliegt. Wenn nicht, weiter bei Zeile 2120.

2130 Hat der Benutzer des Programmes die Variable »i$« mit N gefüllt, so wird die Anzahl der belegten Plätze um 1 erniedrigt (so wird vermieden, daß ein Tabellenplatz mit einer Falschinformation gefüllt wird), und es wird zum Beginn des Programmteils verzweigt.

2140 Hier beginnt die Schleife, in der die einzelnen Tabellenplätze von »eingabe$« auf die richtige Länge gebracht werden.

2150 Solange die Länge des gerade behandelten »eingabe$« – abhängig von der Laufvariablen »i« – kürzer ist als die maximale Länge, die für die Einzelinformation erlaubt ist, so lange wird an »eingabe$« ein einzelnes Zeichen angehängt (siehe Zeile 170).

2160 Ende der Schleife.

2170 Die nächste Schleife sorgt dafür, daß der Tabellenplatz »adress$«, der bestimmt ist durch den Inhalt der Variablen »belegt«, gefüllt wird.

2180 Danach hängen wir an den Inhalt des Tabellenplatzes die einzelnen Informationen aus der Tabelle »eingabe$« an.

2190 Ende der Schleife.

2200 Der Benutzer wird gefragt, ob er mit dem Buchstaben M ins Hauptmenue will oder mit dem Buchstaben W weitere Eingaben machen will.

2210 Warten auf Tastendruck M oder W.

2220 Wenn der Benutzer die Variable »i$« mit W gefüllt hat, Sprung zum Beginn des Unterprogrammes.

2230 Die Variable »aendern« wird auf den Wert 1 gesetzt. Sie dient im Programmteil PROGRAMMENDE zur Ausgabe einer Warnung an den Benutzer, wenn die letzte Änderung noch nicht gespeichert wurde.

2240 Ende des Unterprogrammes.

Der nächste Programmteil, den wir uns vornehmen, ist der Teil, in dem wir die Adressen wieder auseinandernehmen. Da jede komplette Adresse in einem langen String gespeichert ist und dieser String neben den tatsächlichen Informationen noch die Füllzeichen (»ersatz$«) enthält, benutzen wir die Tabelle »laenge«, um ihn wieder auseinanderzunehmen. Hier wieder der komplette Programmteil:

```
9000 'Adresse ausgeben
9010 CLS
9020 PRINT USING"Dateiname: \
   \  Satz Nr. ###";datei$,nummer
9030 PRINT
9040 z3=1
9050 FOR z1=1 TO 7
9060 PRINT USING"\              \";feld$(z1
);
9070 FOR z2=z3 TO z3+laenge(z1)-1
9080 IF MID$(adress$(nummer),z2,1)<>ersa
tz$ THEN PRINT MID$(adress$(nummer),z2,1
);
9090 NEXT z2
9100 z3=z2
9110 PRINT
9120 NEXT z1
9130 PRINT
9140 RETURN
```

Und jetzt die Erläuterungen:

9000 Information für den Programmierer.

9010 Bildschirm löschen.

9020 Ausgabe des Dateinamens und der Nummer der Adresse, die ausgegeben werden soll.

9030 Leerzeile ausgeben.

9040 Die Variable »z3« wird benutzt, um aus dem »adress$« jeweils ein Zeichen herauszupicken. Sie muß mit dem ersten Zeichen anfangen und wird deshalb auf 1 gesetzt.

9050 In der Laufvariablen »z1« wird von 1 bis 7 gezählt; damit werden die Einzelinformationen aus unserem »adress$« angesprochen. Wir haben insgesamt 7 verschiedene.

9060 Der Name der Einzelinformation wird ausgedruckt.

9070 Beginnend bei dem derzeitigen Inhalt von »z3« wird in der Laufvariablen »z2« gezählt, bis ein Wert erreicht ist, der sich aus dem jetzigen Inhalt von »z3« ergibt, verringert um 1 plus der maximalen Länge der Einzelinformation. Das heißt: Beim ersten Schleifendurchlauf wird von 1 bis 8 gezählt, beim zweiten Durchlauf von 9 bis 18, beim dritten von 19 bis 38, beim vierten von 39 bis 68, beim fünften von 69 bis 75, beim sechsten von 76 bis 95 und beim siebten und letzten von 96 bis 110. Damit wissen Sie auch endlich, wie lang jede einzelne Adresse ist: genau 110 Zeichen.

9080 Und jetzt wird's kompliziert: Um diese Zeile verstehen zu können, müssen Sie die Bedeutung der Funktion MID$ kennen. Diese Funktion ist in der Lage, aus einem String ein Stück herauszupflücken. Sie braucht dazu die Informationen, welcher String gemeint ist, an welcher Stelle begonnen werden soll und wie lang das Stück ist, das herausgenommen werden soll. Hier ist der String »adress$«; der Beginn wird bestimmt durch den Wert der Laufvariablen »z2«, und die Länge ist immer gleich, nämlich 1. Das Zeichen, das auf diese Weise ermittelt wurde, wird mit »ersatz$« verglichen. Wenn die beiden Zeichen unterschiedlich sind, wird der ermittelte Buchstabe ausgedruckt.

9090 Ende der Schleife und damit Beginn beim nächsten Zeichen von »adress$«.

9100 Nach Verlassen der Schleife zeigt »z2« genau auf den Punkt innerhalb von »adress$«, an dem die neue Einzelinformation beginnt. Dieser Wert wird für den nächsten Schleifendurchlauf von »z1« in der Variablen »z3« gespeichert.

9110 Durch den PRINT-Befehl wird der Cursor in die nächste Zeile gesetzt.

9120 Ende der Schleife mit der Laufvariablen »z1« und damit Beginn bei der nächsten Einzelinformation innerhalb von »adress$«.

9130 Leerzeile

9140 Ende des Unterprogrammes.

Dieses Unterprogramm wird jedesmal dann aufgerufen, wenn wir eine Adresse ausdrucken wollen. Das kann in den Programmpunkten SUCHEN, AENDERN, LOESCHEN und LISTEN geschehen. Wir haben uns damit die Arbeit gespart, diesen Programmteil jedesmal neu tippen zu müssen.

Es gibt noch ein Unterprogramm, das aus verschiedenen Menuepunkten angesprungen werden kann: Es ist die Meldung, daß keine Datei im Speicher ist, wenn die Variable »belegt« noch auf 0 steht – was immer dann geschieht, wenn keine Adresse im Speicher ist. Hier das Programm:

```
10000 'Keine Datei im Speicher
10010 CLS
10020 PRINT "Keine Datei im Speicher!"
10030 FOR i =1 TO 2000
10040 NEXT i
10050 RETURN
```

Eigentlich ist ein Kommentar unnötig. Dieser Programmteil beschränkt sich auf das Löschen des Bildschirms, eine Meldung an den Benutzer, einer Warteschleife und dem Ende des Unterprogrammes.

Um uns das dauernde Eintippen von neuen Adressen zu ersparen, nehmen wir uns als nächstes die Programmteile SPEICHERN und LADEN vor. Beginnen wir mit dem Speichern.

Bevor Sie das entsprechende Programm zu Gesicht bekommen, einige Informationen darüber, wie sich der Computer zu dem Kassetten-Recorder verhält. Obwohl beide in einem Gehäuse sitzen, hat der Computer keine Möglichkeit festzustellen, in welchem Status sich der Recorder befindet, das heißt, ob die PLAY- oder REC-Taste gedrückt wird; er muß sich in dieser Hinsicht auf den Benutzer verlassen. Um den Benutzer zu den notwendigen Aktivitäten zu veranlassen, gibt der Computer Meldungen auf dem Bildschirm aus, die dem Benutzer sagen, was er zu einem bestimmten Zeitpunkt tun soll. Diese Meldungen kann man unterdrücken, was wir allerdings nicht tun sollten.

In unserer Tabelle ›adress$‹ sind die Informationen enthalten, die wir auf Kassette speichern wollen. Den Befehl, den wir dazu brauchen, ist ein fast normaler PRINT-Befehl; irgendwie müssen wir allerdings dem Computer sagen, daß die Ausgabe nicht auf den Bildschirm, sondern auf den Kassetten-Rekorder soll: Dazu fügen wir an das PRINT ein #9 an. Alles, was wir so drucken, geht auf die Kassette. Vorher müssen wir aber den Kassetten-Rekorder für die Aufnahme von Daten vorbereiten; das bewirkt der Befehl OPENOUT. Danach muß der Name stehen, unter dem die Informationen gespeichert werden sollen. Wenn wir mit der Speicherung fertig sind, werden wir als ordentliche Menschen das, was wir mit OPEN geöffnet haben, mit CLOSEOUT wieder schließen. Und nun das Programm:

```
7000 'Speichern
7010 IF belegt >0 THEN GOTO 7040
7020 GOSUB 10000
7030 GOTO 7130
7040 CLS
7050 INPUT"Bitte geben Sie den Dateiname
n ein";datei$
7060 IF LEN(datei$)>16 THEN PRINT"Name z
u lang!":GOTO 7050
```

```
7070 OPENOUT datei$
7080 FOR i=1 TO belegt
7090 PRINT#9,adress$(i)
7100 NEXT i
7110 CLOSEOUT
7120 aendern=0
7130 RETURN
```

Hier wieder einige Erläuterungen:

7000 Nachricht für den Programmierer.

7010 Falls eine Adresse im Speicher ist, werden die nächsten beiden Zeilen übersprungen.

7020 Sprung in das Unterprogramm, das dem Benutzer sagt, daß keine Datei im Speicher ist.

7030 Sprung an das Ende des Unterprogrammes SPEICHERN.

7040 Bildschirm löschen.

7050 Eingabe des Dateinamens, unter dem die Adressen gespeichert werden sollen.

7060 Prüfen, ob der eingegebene Dateiname die erlaubte Höchstlänge von 16 Zeichen überschreitet. Wenn ja, weiter bei 7050.

7070 Vorbereiten des Kassetten-Rekorders zur Aufnahme der Adressen unter dem Namen, der gerade eingegeben wurde.

7080 Beginn einer Schleife, die in der Laufvariablen »i« von 1 bis zur Anzahl der belegten Tabellenplätze zählt.

7090 Ausgabe eines Tabellenplatzes auf die Kassette.

7100 Ende der Schleife.

7110 Soviel zum Thema »ordentliche Menschen«.

7120 Die Variable »aendern« wird auf 0 gesetzt, damit bei Programmende keine Warnung erfolgt.

7130 Ende des Unterprogrammes.

Damit wir unsere schönen gespeicherten Daten auch wieder in den Computerspeicher bekommen, befassen wir uns als nächstes mit dem Programmpunkt LADEN. Der Vorgang ist fast der gleiche wie beim Speichern; allerdings müssen hierbei die Adressen gezählt werden, die von der Kassette kommen. Wieder muß der Kassetten-Rekorder für die Arbeit mit den Daten vorbereitet werden. Der Befehl dazu heißt OPENIN plus Dateiname. Um Daten vom Rekorder in den Computer zu bringen, benutzen wir den uns bekannten INPUT-Befehl mit dem Zusatz #. Damit der Computer nicht versucht, Informationen zu lesen, die hinter der letzten Adresse liegen, wird beim Speichern der Adressen in dem Augenblick, in dem Sie den Befehl CLOSEOUT geben, auf das Band eine Ende-Markierung geschrieben: diese Markierung erkennt der Computer beim Lesen wieder. Ist das Dateiende erreicht, nimmt die Variable EOF, die der Computer für solche

Fälle bereithält, einen Wert an, der von 0 verschieden ist; diesen Wert können wir überprüfen. Ist das Ende der Datei erreicht, schließen wir die Datei und kehren zurück ins Hauptmenue. Und nun der entsprechende Programmteil:

```
1000 'Laden
1010 CLS
1020 INPUT"Bitte Dateiname eingeben";dat
ei$
1030 belegt=1
1040 OPENIN datei$
1050 INPUT#9,adress$(belegt)
1060 IF EOF THEN GOTO 1090
1070 belegt=belegt+1
1080 GOTO 1050
1090 CLOSEIN
1100 RETURN
```

Da beim normalen Arbeiten mit einem Dateiverwaltungsprogramm mit dem Laden der Daten begonnen wird, beginnt dieses Programm bei Zeile 1000.

1000 Nachrichten an den Programmierer.

1010 Bildschirm löschen.

1020 Eingabe des Dateinamens. Sie können genau wie im Programmpunkt SPEICHERN eine Prüfroutine auf die zulässige Namenslänge einbauen.

1030 Die Anzahl der belegten Plätze (»belegt«) wird auf den Anfangswert 1 gesetzt.

1040 Die Datei wird zum Lesen geöffnet.

1050 Ein Tabellenplatz wird von der Kassette gefüllt.

1060 Es wird geprüft, ob das Dateiende erreicht ist; wenn ja, werden die nächsten beiden Zeilen übersprungen.

1070 Die Anzahl der belegten Tabellenplätze wird um 1 erhöht.

1080 Weiter beim Lesen.

1090 Schließen der Datei.

1100 Ende des Unterprogrammes.

Damit wir auch wissen, ob das Laden und Speichern unserer Daten richtig funktioniert, brauchen wir als nächstes den Programmteil LISTEN.

```
6000 'listen
6010 IF belegt >0 THEN GOTO 6040
6020 GOSUB 10000
6030 GOTO 6100
6040 FOR nummer=1 TO belegt
```

```
6050 GOSUB 9010
6060 PRINT "M=Menue ENTER=Weiter"
6070 i$=UPPER$(INKEY$):IF i$ <>CHR$(13)
AND i$<>"M" THEN GOTO 6070
6080 IF i$ ="M" THEN GOTO 6100
6090 NEXT nummer
6100 RETURN
```

Die Erklärung:

6000 bis 6030 siehe 7000 bis 7030.

6040 Dieser Programmteil wird (beginnend bei 1) durchlaufen, bis alle belegten Tabellenplätze erfaßt sind. Gezählt wird in den Variablen »nummer«, weil wir diesen Variablennamen in dem Unterprogramm verwenden, das eine Adresse auf den Bildschirm bringt (Zeile 9000ff).

6050 Sprung in das Unterprogramm zum Ausdrucken der Adresse.

6060 Nachricht an den Benutzer, welche Möglichkeiten er jetzt zur Verfügung hat.

6070 Eingabe eines Zeichens mit INKEY$ und Überprüfung auf Zulässigkeit.

6080 Hat der Benutzer M eingegeben, Sprung ans Ende des Unterprogrammes.

6090 Ende der Schleife.

6100 Ende des Unterprogrammes.

Jetzt kommen der Reihe nach drei Programmteile, deren Logik etwas schwerer zu verstehen ist. Wir beginnen mit Menuepunkt LOESCHEN.

```
5000 'Loeschen
5010 IF belegt>0 THEN GOTO 5040
5020 GOSUB 10000
5030 GOTO 5160
5040 CLS
5050 PRINT "Welche Adresse soll geloesch
t werden 1 -";belegt;:INPUT nummer
5060 IF nummer <1 OR nummer> belegt THEN
 GOTO 5040
5070 GOSUB 9010
5080 PRINT "Soll Adresse wirklich geloes
cht werden J/N"
5090 i$=UPPER$(INKEY$): IF i$ <> "N" AND
 i$ <>"J" THEN GOTO 5090
5100 IF i$ ="N" THEN GOTO 5160
5110 FOR i = nummer TO belegt
5120 adress$(i)=adress$(i+1)
```

```
5130 NEXT i
5140 aendern=1
5150 belegt=belegt-1
5160 RETURN
```

Die Erklärung:

5000 bis 5030 siehe 7000 bis 7030.

5040 Bildschirm löschen.

5050 Da jede Adresse eine Nummer hat, nämlich die Nummer ihres Tabellenplatzes, fragen wir den Benutzer nach der Nummer der Adresse, die gelöscht werden soll. Gleichzeitig informieren wir ihn über den Bereich der Adressen, die er löschen kann. Dieser Bereich beginnt bei 1 und endet bei dem Inhalt der Variablen »belegt«.

5060 Überprüfung auf Zulässigkeit der Eingabe.

5070 Ausgeben der Adresse mit der eingegebenen Nummer im Unterprogramm 9000ff.

5080 bis 5090 Sicherheitsabfrage.

5100 Bei Eingabe von N für »nein« Sprung ans Unterprogrammende.

5110 bis 5130 Beginnend bei der eingegebenen Nummer, werden alle Tabellenplätze um einen Platz nach unten verschoben. Der oberste Tabellenplatz wird dadurch mit einem Nullstring gefüllt.

5140 Die Variable »aendern« wird für das Programmende auf 1 gesetzt.

5150 Die Zahl der belegten Plätze wird um 1 erniedrigt.

5160 Ende des Unterprogrammes.

Damit wir die Möglichkeit haben, in unserer Adressammlung Adressen mit einer gemeinsamen Eigenschaft zu finden, brauchen wir einen Programmteil SUCHEN.

```
3000 'Suchen
3010 IF belegt>0 THEN GOTO 3040
3020 GOSUB 10000
3030 GOTO 3230
3040 CLS
3050 gefunden=0
3060 INPUT"Nach welchem Begriff suchen S
ie";such$
3070 FOR i=1 TO belegt
3080 FOR j=1 TO 110
3090 vergleich$=MID$(adress$(i),j,LEN(su
ch$))
3100 IF vergleich$<>such$ THEN GOTO 3160
3110 gefunden=1:nummer=i:GOSUB 9010
3120 PRINT:PRINT"<W>=Weiter    <M>=Menue"
```

```
3130 i$=UPPER$(INKEY$):IF i$<>"M" AND i$
<>"W" THEN 3130
3140 IF i$="M" THEN GOTO 3230
3150 GOTO 3170
3160 NEXT j
3170 NEXT i
3180 IF gefunden=0 THEN PRINT such$;" wu
rde nicht gefunden!"
3190 IF gefunden=1 THEN PRINT"Weitere Ad
ressen mit "such$" wurden nicht gefunden
!"
3200 PRINT:PRINT"W=Weitersuchen M=Menue"
3210 i$=UPPER$(INKEY$):IF i$<>"M" AND i$
<>"W" THEN 3210
3220 IF i$="W" THEN GOTO 3040
3230 RETURN
```

Die Erklärungen:

3000 bis 3030 siehe 7000 bis 7030.

3040 Bildschirm löschen.

3050 Die Variable »gefunden« wird benötigt, um den Text am Ende der Datei den jeweiligen Gegebenheiten anzupassen.

3060 Eingabe des gesuchten Begriffes in die Variable »such$«.

3070 Wir haben es hier mit zwei geschachtelten FOR...NEXT Schleifen zu tun. Die erste wählt, beginnend beim ersten Tabellenplatz bis zum letzten, die jeweilige Adresse aus.

3080 Die zweite sucht von Beginn des Tabellenplatzes, der gerade in Arbeit ist, nach dem Vorkommen von »such$«.

3090 Die beiden Strings, die verglichen werden sollen, sind »such$« und ein Stück von »adress$(i)«, das bei dem jeweiligen Wert von j beginnt und genauso lange ist, wie »such$«. Dieses entsprechende Stück wird in der Variablen »vergleich$« gespeichert.

3100 Wenn die beiden Strings nicht gleich sind, werden die nächsten 4 Zeilen übersprungen.

3110 Diese Zeile und die folgenden drei werden nur erreicht, wenn in der Adresse ein Stück gefunden wurde, das mit dem eingegebenen »such$« identisch ist. Die Variable »gefunden« wird auf 1 gesetzt. Der Variablen »nummer« wird der Inhalt von i zugewiesen und das Unterprogramm angesprungen, das die Adresse ausdruckt, die auf Tabellenplatz »nummer« steht.

3120 bis 3130 Hier wird gefragt, ob der Benutzer weitersuchen oder ins Hauptmenue zurück will.

3140 Beantwortet der Benutzer die Frage mit M, Sprung ans Ende des Unterprogrammes.

3150 Falls in einer Adresse der »such$« zweimal vorkommt, würde es normalerweise dazu führen, daß die gleiche Adresse zweimal ausgedruckt wird. Um das zu vermeiden, wird das Ende der inneren FOR. . .NEXT-Schleife übersprungen und damit dafür gesorgt, daß die Sucherei in der nächsten Adresse fortgesetzt wird.

3160 Ende der inneren Schleife; das heißt, daß der nächste »vergleich$« ein Zeichen später beginnt.

3170 Ende der äußeren Schleife. Alle Adressen sind durchsucht.

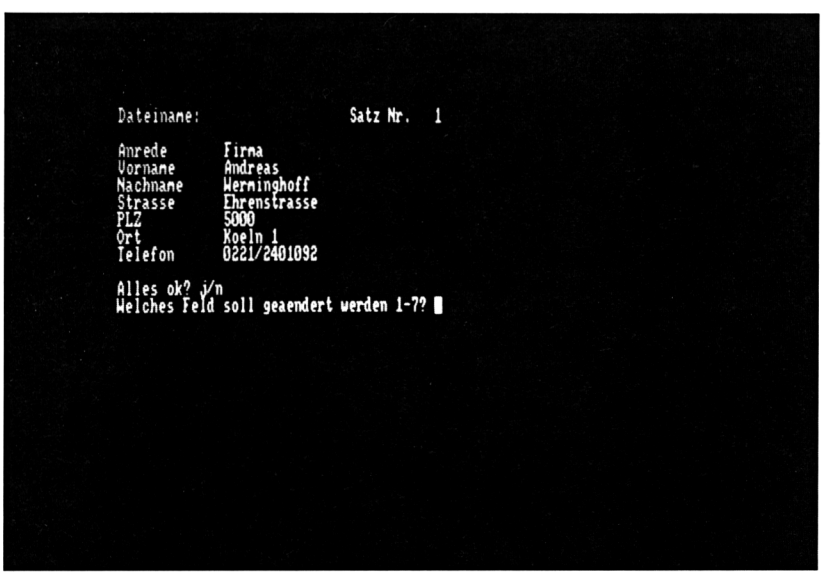
3180 und 3190 Abhängig von dem Inhalt der Variablen »gefunden« wird ein unterschiedlicher Text am Dateiende ausgedruckt.

3200 Leerzeile und erneute Wahlmöglichkeit für den Benutzer.

3220 Will der Benutzer weitersuchen, hat er die Variable i$ mit W gefüllt und das Unterprogramm beginnt von vorne.

3230 Ende des Unterprogrammes.

Da es selbst in den besten Familien vorkommt, daß man umzieht, muß jedes vernünftige Adressprogramm eine Möglichkeit bieten, Adressen zu AENDERN. Dieses Programm setzt sich aus verschiedenen Teilen zusammen: Ausgabe der zu ändernden Adresse, Ändern einer einzelnen Information, Einfügen der geänderten Information in die Originaladresse anstelle der alten Information, Frage nach weiteren Änderungswünschen und Rücksprung ins Hauptmenue.



```
4000 'Aendern
4010 IF belegt>0 THEN GOTO 4040
4020 GOSUB 10000
4030 GOTO 4270
4040 CLS
4050 PRINT"Welche Adresse soll geaendert
 werden 1 -";belegt;:INPUT nummer
4060 IF nummer<0 OR nummer>belegt THEN G
OTO 4040
4070 IF nummer=0 THEN GOTO 4270
4080 GOSUB 9010
4090 INPUT"Welches Feld soll geaendert w
erden 1-7";i
4100 IF i < 1 OR i > 7 THEN GOTO 4270
4110 INPUT"Neuer Inhalt";inhalt$
4120 IF LEN(inhalt$)>laenge(i) THEN PRIN
T"Wort zu lang!":GOTO 4110
4130 FOR j=LEN(inhalt$) TO laenge(i)
4140 inhalt$=inhalt$+ersatz$
4150 NEXT j
4160 FOR j=1 TO i-1
4170 beginn=beginn+laenge(j)
4180 NEXT j
4190 beginn=beginn+1
4200 MID$(adress$(nummer),beginn,laenge(
i))=inhalt$
4210 beginn=0
4220 GOSUB 9010
4230 PRINT"Alles ok? j/n"
4240 i$=UPPER$(INKEY$):IF i$<>"N" AND i$
<>"J" THEN GOTO 4240
4250 IF i$="N" THEN GOTO 4090
4260 aendern=1
4270 RETURN
```

Die Erklärungen:

4000 bis 4040 siehe 7000 bis 7040.

4050 Eingabe der Zahl, unter deren Nummer die zu ändernde Adresse in der Tabelle gespeichert ist, in die Variable »nummer«.

4060 Überprüfung der Eingabe auf Zulässigkeit.

4070 Falls der Benutzer aus Versehen den falschen Programmteil gewählt hat und auf die Frage nach der Nummer mit dem Drücken der ENTER-Taste antwortet, steht in der Variablen »nummer« die Zahl 0. Dieser Inhalt wird abgefragt und, wenn er 0 ist, wird an das Ende des Unterprogrammes AENDERN gesprungen.

4080 Sprung in das Unterprogramm, das die Adresse ausdruckt, die in der Tabelle unter »nummer« gespeichert ist.

4090 Da wir sieben Einzelinformationen haben, wird der Benutzer gefragt, welche von diesen sieben er ändern will.

4100 Überprüfen der Eingabe.

4110 Eingabe der neuen Einzelinformation in die Variable »inhalt$«.

4120 Überprüfen der Länge mit der Funktion LEN und dem Inhalt von »laenge« im Tabellenplatz »i«.

4130 bis 4150 Ergänzen des neuen Inhaltes mit Füllzeichen (»ersatz$« bis zur Maximallänge.

4160 bsi 4180 Um den Teil zu ermitteln, der geändert werden soll, werden die Längen der Einzelinformationen, die in der Tabelle »laenge« gespeichert sind, bis zu der Information addiert, die vor der zu ändernden steht. Wenn zum Beispiel die Einzelinformation »Nachname« geändert werden soll, wird die Länge von Anrede und Vorname addiert.

4190 Addiert man den Inhalt von »beginn« jetzt noch 1, erhält man genau den Anfangspunkt der zu ändernden Einzelinformation.

4200 Die Funktion MID$ kann man auch als Befehl einsetzen, indem man sie auf die linke Seite des Gleichheitszeichens bei einer Zuweisung setzt. Jetzt beschreibt sie den Teil eines Strings, der ersetzt werden soll. In unserem Fall soll aus dem entsprechenden Tabellenplatz (»adress$(nummer)«) ein Stück ersetzt werden, das bei »beginn« anfängt und so lang ist, wie in der Tabelle »laenge« für die zu ändernde Einzelinformation gegeben ist. Dieser Teil wird durch »inhalt$« ersetzt.

4210 Für weitere Änderungen wird »beginn« wieder auf 0 gesetzt.

4220 Die geänderte Adresse wird ausgedruckt (Unterprogramm 9000ff).

4230 und 4240 Frage nach weiteren Änderungswünschen und Eingabe der Antwort.

4250 Falls der Benutzer weitere Änderungen vornehmen will und dies durch die Eingabe von N kundtut, Sprung auf Eingabe der zu ändernden Einzelformation.

4260 Für das Programmende wird die Variable »aendern« auf 1 gesetzt.

4270 Ende des Unterprogrammes.

Sie werden sich fragen, warum ich das Programmende in ein Unterprogramm gesetzt habe. Dafür gibt es zwei Gründe: Erstens kann es passieren, daß der Benutzer diesen Programmteil aus Versehen anwählt, und zweitens muß ich ihm die Chance geben, Fehler wieder gut zu machen, die er begangen hat.

```
8000 'Ende
8010 CLS
```

```
8020 IF aendern=1 THEN PRINT"Achtung! Di
e letzte Aenderung wurde nicht gespeiche
rt!"
8030 PRINT:PRINT"Wollen Sie wirklich auf
hoeren <J>/<N>?"
8040 wa$=UPPER$(INKEY$):IF wa$<>"N" AND
wa$<>"J" THEN GOTO 8040
8050 IF wa$="J" THEN CLS:END
8060 RETURN
```

Die Erklärungen:

8000 bis 8010 Na, Sie wissen schon.
8020 Jetzt wissen Sie endlich, wozu die Variable »aendern« da war.
8030 Sicherheitsabfrage.
8040 Eingabe der Antwort.
8050 Bei Eingabe von J: PROGRAMMENDE
8060 Ende des Unterprogrammes.

Nachdem ich mit der Arbeit fertig war und das Ergebnis meinem Freund zeigte, hat er mich aus Dankbarkeit auf zehn Jahre zum Ehrenmitglied seines Vereins Computergeschädigter Anstreicher ernannt. Bei der ersten Mitgliederversammlung goß mir irgendeiner meiner Vereinsbrüder vor Liebe einen Eimer Farbe über den Kopf, und ein zweiter verteilte diese Farbe genüßlich mit einer Tapezierolle. Ich kündigte die Freundschaft, zog in eine andere Stadt und schreibe ab sofort nur noch Programme für Süßwarenhersteller.

Wenn Sie das Adressverwaltungsprogramm komplett nachvollziehen können und in der Lage sind, eine ähnliche Aufgabe mit dieser Vorlage selbst zu lösen, hat Ihnen dieser kleine Kursus in BASIC das gebracht, was er sollte. Falls es Sie interessiert, welche Möglichkeiten Ihr Schneider noch hat, beschäftigen Sie sich bitte mit den Beispielprogrammen zu den einzelnen Befehlen.

Wenn die Musik kommt...

Alle reden von den phantastischen Möglichkeiten, die der Schneider zur Klangerzeugung in BASIC hat. Die hat er zweifellos, nur muß man fast Tontechniker sein, um zu verstehen, welche Befehle man wie einsetzen muß, um das gewünschte Ergebnis zu erreichen. Ich werde hier nicht noch einmal die sehr technischen Erklärungen aus dem Handbuch wiederholen, sondern statt dessen – wie auch in den anderen Kapiteln dieses Buches – Programme vorstellen, die Ihnen die Auswirkungen bestimmter Befehle oder Parameter deutlich machen. Beginnen wir mit der einfachsten Form des Soundbefehles:

Die Schwingungsdauer kann jeden Wert im Bereich von 0 bis 4095 annehmen. Sie steuert die Höhe eines Tons, indem sie dem Computer angibt, innerhalb welcher Zeit eine Schwingung des Tons vollendet ist. Ein einfaches Beispielprogramm liefert alle zur Verfügung stehenden Töne:

```
10 FOR i=0 to 4095

20 SOUND 1,i

30 PRINT i

40 NEXT i
```

Sie werden feststellen, daß das Programm mit sehr hohen Tönen beginnt, die gut zu unterscheiden sind. Je größer i wird, desto tiefer wird der Ton und desto geringer wird der Unterschied zwischen zwei Tönen. Um alle verfügbaren Töne der Tonleiter zu spielen, muß man also einen anderen Weg gehen. Man sucht sich aus dem Handbuch die tiefsten Töne heraus, füllt sie in eine Tabelle, und errechnet von diesen Grundtönen die Töne in der jeweiligen Oktave, indem Sie sie durch eine Potenz von 2 teilen.

```
10 DATA 3822,3608,3405,3214
20 DATA 3034,2863,2703,2551
30 DATA 2408,2273,2145,2025
```

```
40 DIM ton(12)
50 FOR i=1 TO 12
60 READ ton(i)
70 NEXT i
80 FOR i=0 TO 7
90 FOR j=1 TO 12
100 t=ROUND(ton(j)/2^i)
110 PRINT t;
120 SOUND 1,t
130 NEXT j
140 NEXT i
```

Das, was aus dem Lautsprecher herauskommt, hört sich noch nicht nach Tonleiter an, obwohl die einzelnen Töne schon gut zu unterscheiden sind. Innerhalb einer Oktave gibt es für unsere Ohren 12 Töne, während zu einer Tonleiter nur 8 gehören. Wir müssen also die Töne in unseren DATA Zeilen etwas auswählen. Nehmen wir die C-Dur Tonleiter: Alle Töne, die nicht in die C-Dur Tonleiter gehören, werden entfernt. Damit man die Werte, die zu den einzelnen Tönen gehören, besser unterscheiden kann, habe ich vor jedem Wert den Notennamen ausgedruckt (in der deutschen Schreibweise, also B statt H).

```
10 DATA 3822,C,3405,D
20 DATA 3034,E,2863,F,2551,G
30 DATA 2273,A,2025,H
40 DIM ton(7),name(7)
50 FOR i=1 TO 7
60 READ ton(i),name$(i)
70 NEXT i
80 FOR i=0 TO 8
90 FOR j=1 TO 7
100 t=ROUND(ton(j)/2^i)
110 PRINT t;name$(j);
120 SOUND 1,t
130 NEXT j
140 PRINT
150 NEXT i
```

Hört sich schon besser an. Wenn wir noch die Dauer dieser Töne verlängern können, sind wir schon ein schönes Stück weiter. Dazu benötigen wir in unserem SOUND-Befehl einen neuen Parameter (wie Sie gehört haben, geht es auch ohne ihn).

SOUND 1, Schwingungsdauer, Länge

Erst die zulässigen Werte: Sie liegen zwischen −32768 und +32767. Die Länge eines Tons richtet sich bei normalen Musikinstrumenten nach der Zeit, in der der Ton verklungen ist. Wenn Sie eine Gitarrensaite stark zupfen, dauert der Ton, der erzeugt wird, wesentlich länger, als wenn Sie die Saite nur kurz berühren. Bei unserem Soundbaustein läuft die Sache etwas anders. Sie können einen Ton leise 5 Minuten oder laut eine 100stel Sekunde spielen – je nachdem, ob Sie für Länge einen Wert von 10000 oder 1 setzen. Wenn Sie die Länge auf 0 setzen, wird nicht, wie Sie vielleicht erwarten, kein Ton gespielt, sondern die Länge wird durch die Lautstärkenhüllkurve bestimmt. Doch davon später. Ändern Sie die Zeile 120 des letzten Programms in:

```
120 SOUND 1,t,j^2
```

Dann merken Sie deutlich die unterschiedlichen Tondauern. Ein weiterer Zusatz zum SOUND-Befehl gibt uns die Möglichkeit, die Lautstärke eines Tons zu verändern.

SOUND 1, Schwingungsdauer, Länge, Lautstärke

Der Wert für die Lautstärke darf Werte zwischen 0 und 15 annehmen. Wenn Sie diesen Parameter nicht benutzen, arbeiten Sie normalerweise mit dem Wert 4. Wenn Sie keine Lautstärkenhüllkurve (schon wieder dieses Ding) angeben, werden die Werte, die größer sind als 7, um 8 verringert. Die beiden SOUND-Befehle SOUND 1,239,50,4 und SOUND 1,239,50,12 bewirken also dasselbe.

Und jetzt kommt die Sache mit der Lautstärkenhüllkurve. Sie haben wahrscheinlich schon mit dem Befehl ENV gespielt und sich gewundert, daß das, was Sie danach hörten, von dem, was Sie dachten, zu programmieren, vollkommen verschieden war. Das wichtigste, was Sie wissen müssen, ist die Tatsache, daß der Wert, den die Lautstärke tatsächlich hat, die Zahl 15 nicht überschreiten kann! Schauen wir uns den Befehl an:

ENV Nr,Sz1,Sg1,Pl1⟨,Sz2,Sg2,Pl2,Sz3,Sg3,Pl3,Sz4,Sg4,Pl4,Sz5,Sg5,Pl5⟩

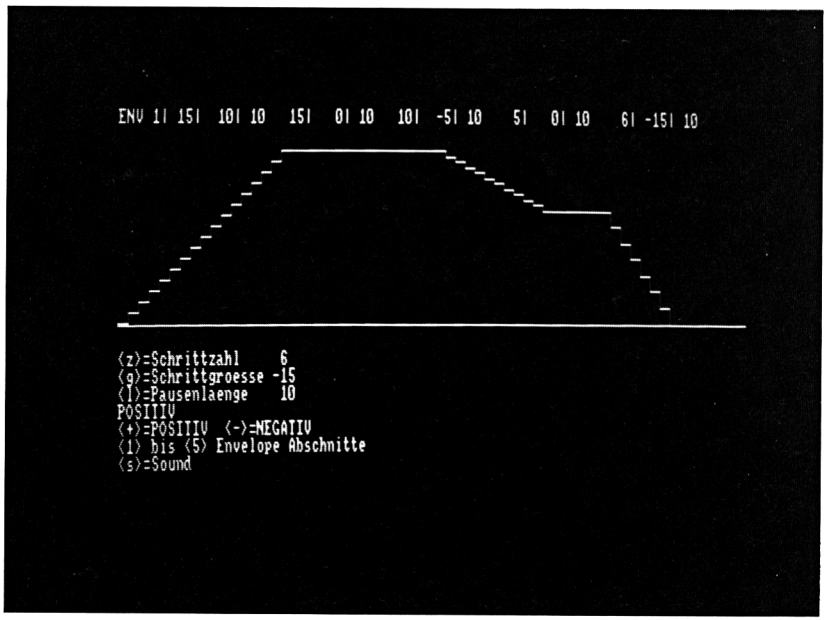
Dabei bedeuten:

Nr. Die Nummer der Hüllkurve	(1 bis 15)
Sz1 bis Sz5: Die Schrittzahl pro Abschnitt	(0 bis 127)
Sg1 bis Sg5: Die Schrittgröße pro Schritt	(−128 bis +128)
Pl1 bis Pl5: Die Pausenlänge nach jedem Schritt	(0 bis 255)

Wenn Sie den ENV-Befehl benützen, müssen Sie nur die ersten vier Werte einsetzen. Der Effekt, den Sie damit erreichen, ist allerdings in den meisten Fällen nicht ein Ohren-

schmaus, da Sie einen Ton anschwellen lassen und der Computer ihn am Ende ruckartig abbricht. Um zu verstehen, was die Auswirkungen dieses Befehles sind, folgt hier ein Demogramm, das Ihnen die Form einer Lautstärkenhüllkurve zeigt und die dazugehörigen Werte in ENV-Befehl:

```
10 MODE 2
20 PLOT 0,198
30 DRAW 640,198
40 LOCATE 1,15:PRINT"<z>=Schrittzahl
"
50 PRINT"<g>=Schrittgroesse "
60 PRINT"<l>=Pausenlaenge   "
70 PRINT
80 PRINT"<+>=POSITIV  <->=NEGATIV"
90 PRINT"<1> bis <5> Envelope Abschnitte
"
100 PRINT"<s>=Sound"
110 LOCATE 1,1
```



```
120 PRINT "ENV 1;  0;    0;  0     0;    0;
  0     0;    0;  0     0;    0;  0     0;
0;  0"
130 ORIGIN 0,200,0,640,360,200
140 s=1
150 a$=INKEY$:IF a$="" THEN 150
160 z=INSTR("12345szgl+-",a$)
170 ON z GOSUB 190,190,190,190,190,280,3
00,340,380,420,450
180 GOTO 150
190 p(z)=p
200 q(z)=q
210 r(z)=r
220 ENV 1,p(1),q(1),r(1),p(2),q(2),r(2),
p(3),q(3),r(3),p(4),q(4),r(4),p(5),q(5),
r(5)
230 LOCATE 1,1
240 PRINT USING"ENV 1;###;####;###  ###;
####;###  ###;####;###  ###;####;###  ##
#;####;###";p(1),q(1),r(1),p(2),q(2),r(2
),p(3),q(3),r(3),p(4),q(4),r(4),p(5),q(5
),r(5)
250 CLG:MOVE 0,0:FOR i=1 TO 5:FOR j=0 TO
 p(i):DRAWR r(i),0:DRAWR 0,q(i):NEXT:NEX
T
260 d=p(1)*r(1)+p(2)*r(2)+p(3)*r(3)+p(4)
*r(4)+p(5)*r(5)
270 RETURN
280 SOUND 1,239,d,0,1
290 RETURN
300 IF p=0 AND s=-1 OR p=127 AND s=1 THE
N RETURN
310 p=p+s
320 LOCATE 1,15:PRINT"<z>=Schrittzahl
 ";p
330 RETURN
340 IF q=-128 AND s=-1 OR q=127 AND s=1
THEN RETURN
350 q=q+s
```

```
360 LOCATE 1,16:PRINT"<g>=Schrittgroesse
 ";q
370 RETURN
380 IF r=0 AND s=-1 OR r=255 AND s=1 THE
N RETURN
390 r=r+s
400 LOCATE 1,17:PRINT"<l>=Pausenlaenge
 ";r
410 RETURN
420 s=+1
430 LOCATE 1,18:PRINT"POSITIV
      "
440 RETURN
450 s=-1
460 LOCATE 1,18:PRINT"NEGATIV
      "
470 RETURN
```

Mit diesem Programm können Sie jeden Abschnitt ihrer Lautstärkenhüllkurve auf einen der erlaubten Werte setzen und sich mit der Taste S den Ton anhören, den Sie erzeugt haben. Ein paar Erklärungen zu dem Programm:

10–140 enthalten Vorbereitungen und den Bildschirm beim Start.

150–180 Steuerung durch Tastendruck.

190–270 Durch das Drücken einer Taste von 1 bis 5 werden die Werte, die zu diesem Zeitpunkt für Schrittzahl, Schrittgröße und Pausenlänge auf dem Bildschirm stehen, dem jeweiligen Abschnitt zugeordnet, die neue Hüllkurve wird gezeichnet und die Werte werden in die Kopfzeile übernommen. Außerdem wird für den Ton eine neue Dauer berechnet, der sich aus der Summe der Länge der einzelnen Abschnitte ergibt. Die Länge jedes Abschnittes wiederum kann man errechnen, indem man die Schrittzahl mit der Pausenlänge multipliziert.

280–290 Der erzeugte Ton wird gespielt. Wichtig in diesem Zusammenhang ist, daß die Lautstärke des Tons auf 0 steht. Diese Lautstärke gibt an, mit welcher Lautstärke der Computer die Hüllkurve beginnt. Unter der Voraussetzung, daß Sie eine Hüllkurve für die Lautstärke angeben, kann der Anfangswert zwischen 0 und 15 liegen.

300–330 Die Schrittzahl wird um S erhöht, wenn S positiv ist, oder erniedrigt, wenn S negativ ist. Vorher wird die Gültigkeit des Wertes überprüft.

340–370 Das gleiche für die Schrittgröße und

380–410 für die Pausenlänge. Die Pausenlänge gibt an, wie lange ein Ton nach der Erhöhung der Lautstärke gespielt werden soll, bevor er noch lauter oder leiser wird.

420–470 Das Vorzeichen für die Veränderung der einzelnen Werte wird durch die Vorzeichentasten bestimmt; der Benutzer erhält die Information auf den Monitor.

Es ist ein Riesenunterschied, ob Sie in einer Laustärkenhüllkurve einen Parameter mit 0 füllen, oder ob Sie den entsprechenden Block ganz weglassen. Lassen Sie ihn ganz weg, also zum Beispiel ENV 1,7,2,10, so ist der erzeugte Ton 7 × 10 Hundertstel Sekunden lang; geben Sie aber z. B. nur den ersten und dritten an, so entsteht zwischen den beiden Teilen eine Pause, deren Länge immer fast zwei Sekunden beträgt, egal, wie lang der vorhergehende oder folgende Abschnitt ist oder wie hoch die Ausgangslautstärke liegt.

Wenn Sie Ihre Parameter für ENV so wählen, daß die Lautsärke größer als 15 würde, teilt der Computer den Wert für die Lautstärke durch 15 und arbeitet nur mit dem Rest weiter. In den meisten Fällen führt das zu unerwünschten Schwankungen in der Lautstärke. Eine Pausenlänge 0 hat die gleichen Auswirkungen wie eine Pausenlänge von 255. Das erklärt auch die Pausen durch nicht belegte Abschnitte.

Alle diese Angaben können Sie für 15 verschiedene Hüllkurven machen und bei der Komposition in den einzelnen SOUND-Befehlen einsetzen:

SOUND 1, Schwingungsdauer, Länge, (Ausgangs-)Lautstärke, Lautstärkenhüllkurvennummer

Benutzen Sie die computereigene Hüllkurve 0, so wird die Dauer eines Tons ausschließlich durch die Angabe im SOUND-Befehl gesteuert. Benutzen Sie eine Längenangabe zusammen mit dem ENV-Befehl, so ist die Längenangabe im SOUND-Befehl vorrangig. Nicht erledigte Teile aus der Kurve werden ignoriert.

Wie Sie wissen, wird der Klang eines Tons nicht nur durch die Tonhöhe und die Lautstärkenschwankungen erzeugt, sondern auch durch geringe Änderungen in der Tonhöhe. Sie haben im SOUND-Befehl einen weiteren Zusatz, der aus 15 Klanghüllkurven eine bestimmte auswählen kann. Der Befehl, mit dem Sie eine Klanghüllkurve definieren können, heißt:

ENT Nr,Sz1,Sg1,Pl1⟨,Sz2,Sg2,Pl2,Sz3,Sg3,Pl3,Sz4,Sg4,Pl4,Sz5,Sg5,Pl5⟩

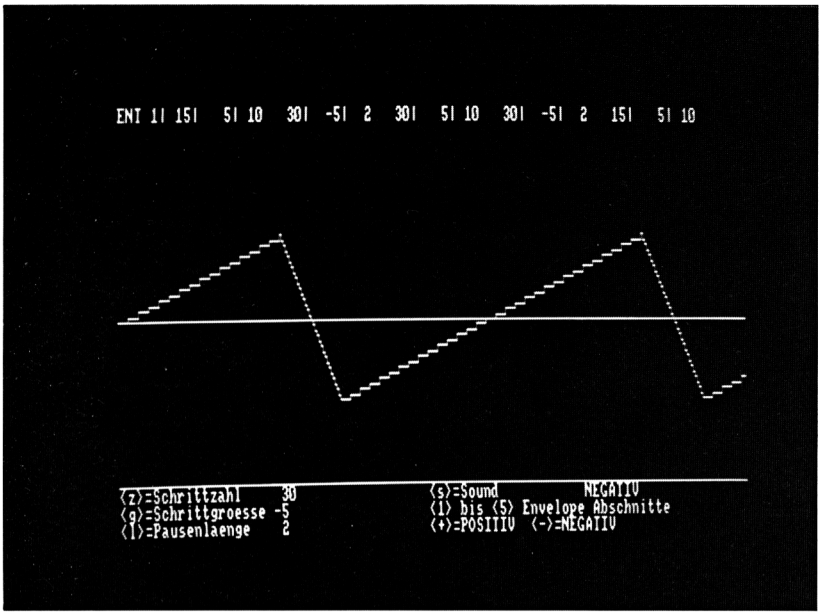
Auch hier bedeuten:

Nr: Die Nummer der Hüllkurve	(1 bis 15)
Sz1 bis Sz5: Die Schrittzahl pro Abschnitt	(0 bis 239)
Sg1 bis Sg5: Die Schrittgröße pro Schritt	(−128 bis +128)
Pl1 bis Pl5: Die Pausenlänge nach jedem Schritt	(0 bis 255)

Zur Demonstration wieder ein Programm, das Ihnen eine Kurve zeigt und mit dem Sie die Parameter für eine Klanghüllkurve ändern können:

```
10 MODE 2
20 PLOT 0,48
30 DRAW 640,48
```

```
40 LOCATE 1,23:PRINT"<z>=Schrittzahl
"
50 LOCATE 1,24:PRINT"<g>=Schrittgroesse
"
60 LOCATE 1,25:PRINT"<l>=Pausenlaenge
"
70 LOCATE 40,23:PRINT"<s>=Sound"
80 LOCATE 40,24:PRINT"<1> bis <5> Envelo
pe Abschnitte";
90 LOCATE 40,25:PRINT"<+>=POSITIV  <->=N
EGATIV";
100 LOCATE 1,1
110 PRINT "ENT 1:  0:    0:  0     0:    0:
  0     0:    0:  0     0:    0:  0     0:
0:  0";
120 ORIGIN 0,200,0,640,360,50
130 s=1
140 a$=INKEY$:IF a$="" THEN 140
150 z=INSTR("12345szgl+-",a$)
160 ON z GOSUB 180,180,180,180,180,270,2
90,330,370,410,440
170 GOTO 140
180 t(z)=t
190 v(z)=v
200 w(z)=w
210 ENT 1,t(1),v(1),w(1),t(2),v(2),w(2),
t(3),v(3),w(3),t(4),v(4),w(4),t(5),v(5),
w(5)
220 LOCATE 1,1
230 PRINT USING"ENT 1:###:####:###  ###:
####:###  ###:####:###  ###:####:###  ##
#:####:###";t(1),v(1),w(1),t(2),v(2),w(2
),t(3),v(3),w(3),t(4),v(4),w(4),t(5),v(5
),w(5)
240 d=t(1)*w(1)+t(2)*w(2)+t(3)*w(3)+t(4)
*w(4)+t(5)*w(5)
250 CLG:MOVE 0,0:DRAW 640,0:MOVE 0,0:FOR
 i=1 TO 5:FOR j=0 TO t(i):DRAWR w(i),0:D
RAWR 0,v(i):NEXT:NEXT
260 RETURN
```




```
270 SOUND 1,239,d,7,0,1
280 RETURN
290 IF t=0 AND s=-1 OR t=239 AND s=1 THE
N RETURN
300 t=t+s
310 LOCATE 1,23:PRINT"<z>=Schrittzahl
 ";t
320 RETURN
330 IF v=-128 AND s=-1 OR v=127 AND s=1
THEN RETURN
340 v=v+s
350 LOCATE 1,24:PRINT"<g>=Schrittgroesse
 ";v
360 RETURN
370 IF w=0 AND s=-1 OR w=255 AND s=1 THE
N RETURN
380 w=w+s
390 LOCATE 1,25:PRINT"<l>=Pausenlaenge
 ";w
400 RETURN
```

```
410 s=+1
420 LOCATE 60,23:PRINT"POSITIV"
430 RETURN
440 s=-1
450 LOCATE 60,23:PRINT"NEGATIV"
460 RETURN
```

Sie sehen schon am Programm, daß die beiden Befehle ENT und ENV sich im Aufbau nur wenig unterscheiden. Ein wesentlicher Unterschied ist, daß ENT im Gegensatz zu ENV keine Auswirkungen auf die Länge eines Tons hat. Damit Sie die Chance haben, Ihren erzeugten Ton in voller Länge zu hören, habe ich auch hier wieder die notwendige Dauer in Zeile 240 ausgerechnet und ihn in den SOUND-Befehl eingesetzt. Ein weiterer Unterschied ergibt sich aus der Tatsache, daß die Schwankungen in der Tonhöhe sowohl im positiven als auch im negativen Bereich stattfinden können. Deshalb empfahl es sich, die Nullinie der Kurve in die Mitte des Bildschirmfensters zu legen.

Sie werden sich vielleicht darüber wundern, daß der erzeugte Ton tiefer wird, wenn die Kurve nach oben geht. Da der ENV-Befehl sich genau wie der SOUND-Befehl auf die Schwingungsdauer eines Tons bezieht, bedeutet ein höherer Wert einen tieferen Ton. Auch bei ENV können durch die unbedachte Anwendung der Parameter für Schrittzahl und Schrittgröße Nebenwirkungen entstehen, die nicht im Sinne des Programmierers liegen. Sie wissen aus der Betriebsanleitung, daß der höchste Wert für die Schwingungsdauer 4095 betragen darf. Und jetzt rechnen Sie bitte mit: Ausgangswert in dem vorliegenden Programm ist bei dem SOUND-Befehl 239. Angenommen, wir arbeiten mit einer Schrittweite von 127 und einer Schrittzahl von 200. Weiter angenommen, daß der ENV-Befehl bei jedem Schritt die Schrittgröße auf den bisherigen Wert addiert. Sie werden auf einen Endwert für die Schwingungsdauer von 25639 kommen, ein Wert, der weit jenseits des Erlaubten liegt. Der Computer weiß sich auch in diesem Fall zu helfen: Er verarbeitet genau wie beim ENV-Befehl nur den Rest von der Division durch 4096. Sie erhalten so einen sehr stufigen Schnitt für Ihren Klang.

Ein Wort noch zu der Nummer der Klanghüllkurve. Setzen Sie vor der Nummer ein Minuszeichen, so sorgen Sie dafür, daß während eines Klanges, der länger ist als die von Ihnen definierte Hüllkurve, diese Hüllkurve solange wiederholt wird, bis der Klang erloschen ist. ein Beispiel:

```
10 ENV 1,15,1,10,10,0,10,15,-1,10

20 ENT 1,5,1,1,5,-1,1

30 SOUND 1,239,0,0,1,1
```

Hört sich ziemlich langweilig an. Setzen Sie nun vor die erste 1 in Zeile 20 ein Minuszeichen, und Sie haben das schönste Tremolo. Sie können diese Minuszeichen nicht in den

SOUND-Befehl schreiben (Improper argument). Damit haben wir unseren SOUND-Befehl schon wieder ein Stück länger:

SOUND 1, Schwingungsdauer, Länge, (Ausgangs-)Lautstärke, Lautstärkenhüllkurvennummer, Klanghüllkurvennummer

Jetzt noch etwas einfaches. Stellen Sie sich vor, Sie haben ein schönes Spiel mit Raumschiffen, Aliens und all diesem Zeug geschrieben. Das einzige, was noch fehlt, ist eine richtige Explosion. Dazu können Sie diese klangvolle Geschichte nicht gebrauchen. Es muß etwas Härteres her. Dabei hilft Ihnen der letzte Parameter im SOUND-Befehl.

Dieser Parameter legt fest, welches Geräusch dem Ton beigemischt werden soll. Er kann Werte zwischen 0 und 15 annehmen. Hören wir uns die einzelnen Geräusche an, ohne beigemischten Klang:

```
10 FOR I=1 TO 16

20 SOUND 1,0,200,7,0,0,I

30 PRINT I

40 FOR J=1 TO 2000:NEXT J

50 NEXT I
```

Sie können den Unterschied zwischen den einzelnen Klängen gut hören. Nun aber zu unserer Explosion. Wir müssen zu dem Geräusch noch eine Lautstärkenhüllkurve basteln. Vielleicht so:

```
10 ENV 1,15,1,10

20 ENT 1,200,10,10,1

30 SOUND 1,0,0,0,1,1,1

40 GOTO 30
```

Schon mal gehört, nicht wahr? Unser SOUND-Befehl sieht also jetzt so aus:

SOUND 1, Schwingungsdauer, Länge, (Ausgangs-)Lautstärke, Lautstärkenhüllkurvennummer, Klanghüllkurvennummer, Geräuschnummer

Haben Sie nicht auch das Gefühl, daß ich mich vor der Erklärung des allersten Arguments, das Sie im SOUND-Befehl benützen können, etwas drücke? Das hat einen

bestimmten Grund: Die Erklärung könnte so umfangreich werden, daß sie den Rahmen dieses Buches sprengen würde. Der erste Wert im SOUND-Befehl gibt den Kanalstatus an, und damit haben wir unseren SOUND-Befehl komplett:

SOUND Kanalstatus, Schwingungsdauer, Tonlänge, (Ausgang-)Lautstärke, Lautstärkenhüllkurve, Klanghüllkurve, Geräuschnummer

Sie haben gelesen, daß Sie mit dem Schneider drei Kanäle ansprechen können. Das stimmt – wenn Sie Ihren Schneider an die häusliche Stereoanlage anschließen. Sonst kommen alle drei Klänge aus dem Piepser unter der Tastatur. Bei jedem erzeugten Klang müssen Sie den Klang auf einen dieser drei Kanäle schicken. Bis jetzt ist alles auf dem linken Kanal gelandet. Den rechten steuern Sie an, wenn Sie sofort nach dem Sound eine 4 schreiben. Den Mittleren, indem Sie SOUND 2 benützen. Wenn Sie den Lautsprecher des Schneiders benützen, kommen alle daraus. Da unser Computer auf allen Kanälen gleichzeitig spielen kann, können Sie so einen Dreiklang erzeugen:

```
10 SOUND 1,239,200,7

20 SOUND 2,190,200,7

30 SOUND 4,159,200,7
```

Das geht solange gut, wie Ihre Töne die gleiche Länge haben; bei unterschiedlicher Länge müssen Sie dafür sorgen, daß ein Ton zusammen mit einem anderen gespielt werden kann. Dabei helfen Ihnen die Zweierpotenzen von 8 bis 32. Wenn Sie zur Nummer eines Kanals addiert werden, so sorgen sie dafür, daß der entsprechende Ton erst dann gespielt wird, wenn auf dem anderen Kanal die dazugehörige Gegenmarkierung erscheint. Sie können die Informationen über einen bestimmten Kanal mit der Variablen SQ abfragen; Ihr Computer hält für jeden Kanal eine Variable SQ bereit. Hinter SQ muß die Zahl stehen, die für den entsprechenden Kanal steht. Also

SQ(1) für Kanal A (links)
SQ(2) für Kanal B (rechts)
SQ(4) für Kanal C (mitte)

Zur Demonstration werden wir in regelmäßigen Abständen diese drei Variablen abfragen, sie in Dualzahlen umwandeln und ausdrucken.

```
10 MODE 2

20 EVERY 10 GOSUB 1000
```

```
999 WHILE INKEY$="":WEND:END

1000 PRINT BIN$(SQ(1),8);" ";BIN$(SQ(2),8);" ";
     BIN$(SQ(4),8)

1010 RETURN
```

Als nächstes setzen wir einige Töne für den ersten Kanal (A) ein:

```
30 SOUND 1,239,50

40 SOUND 1,579,50

50 SOUND 1,119,50

60 SOUND 1,1160,50
```

Drücken Sie nach dem letzten Ton auf irgendeine Taste und schauen Sie sich die erste Spalte an. Die letzten drei Stellen der Dualzahl geben die Anzahl der freien Plätze in der Tonwarteschlange an. Es können Dualzahlen von 0 bis 4 sein. Die erste Stelle einer jeden Spalte gibt an, ob der Kanal gerade arbeitet – ob also gerade ein Ton erzeugt wird. Der Schneider ist in der Lage, pro Kanal vier Töne zu speichern, ohne sich auf die Klangerzeugung konzentrieren zu müssen. Vielleicht erinnern Sie sich noch an das Programm von Seite 75? Da kam die Ready-Meldung vor Ende des letzten Tons.) Fügen Sie nun noch die beiden Töne:

```
70 SOUND 1+16,190,50

80 SOUND 1,159,50
```

an. Die Klangfolge ändert sich nicht; nur das Bild, das am Ende erscheint, ist anders. Bit 5 (Dezimalwert 16) ist an und die letzten drei Stellen zeigen an, daß nur noch zwei Plätze frei sind. Der Kanal wartet nun auf ein Rendezvous mit Kanal B. Tun wir ihm den Gefallen.

```
90 SOUND 2,358,50

100 SOUND 2,284,50
```

Abgesehen von einem etwas schrägen Klang am Anfang hat sich in Spalte 1 nichts geändert. Wir haben zwar Klänge auf Kanal B geschickt, aber es waren nicht die richtigen. Um Kanal A wieder in Aktion zu bringen, muß ein Ton von Kanal B mit einer

Meldung versehen sein, daß er auf ein Rendezvous mit Kanal A aus ist: Dazu addieren wir zu der Kanalnummer die Zahl 8.

```
110 SOUND 2+8,239,100

120 SOUND 2+32,190,50
```

Die letzte Zeile wartet auf ein Rendezvous mit Kanal C. Nur ein Ton von Kanal C, bei dem zu der Kanalnummer 16 addiert wird, kann dafür sorgen, daß der Ton von Zeile 120 erklingt, selbst, wenn dieser Ton keinen eigenen Klang erzeugt.

```
130 SOUND 4+16,0,1
```

In Bit 3, 4 und 5 von SQ (es wird von rechts gezählt und bei 0 begonnen) stehen die Informationen, auf welches Rendezvous der Kanal gerade wartet. Jetzt fehlen noch die Werte 64 und 128. Der Wert 64, auf die Kanalnummer addiert, blockiert die Ausgabe des Tons.

```
140 SOUND 1+64,100,50
```

Um das festzuhalten, wird Bit 6 in der Variablen SQ (1) auf 1 gesetzt. Der schöne Ton bleibt solange ungespielt, bis wir mit einem neuen Befehl den Wartezustand beenden:

RELEASE Kanalnummer

Die Kanalnummer kann einen Wert von 1 bis 7 annehmen. Damit können Sie einen, zwei oder alle drei Kanäle von der Warterei befreien. Damit Kanal A auch richtig wartet, fügen wir eine Zeitschleife ein und erst danach den neuen Befehl:

```
150 FOR i=1 TO 3000:NEXT

160 RELEASE 1
```

Es gibt eine weitere Möglichkeit, das Warten auf einen RELEASE-Befehl zu beenden: Angenommen, der Ton mit dem Hold-Kennzeichen (so nennt sich Bit 6 im Kanalstatus), ist der erste in der Folge der Töne, die noch zu spielen sind, so kann man ihn durch einen anderen ersetzen, indem man diesem ein eingeschaltetes Bit 7 mitgibt:

```
170 SOUND 1+64,200,50

200 FOR I=1 TO 3000:NEXT

210 SOUND 1+128,2000,50
```

Bit 7 (oder der Wert 128 auf die Kanalnummer addiert) löscht die Warteschlange und sorgt dafür, daß der Ton, der dieses Bit am Hals hat, sofort gespielt wird. Schauen Sie sich die Werte an:

```
180 SOUND 1,300,50

190 SOUND 1,400,50
```

Diese beiden Töne werden Sie nie zu Gehör bekommen, da Sie erst durch das Bit 6 aus Zeile 170 gebremst und dann durch das Bit 7 aus 210 vollkommen gelöscht werden – furchtbares Schicksal.

Damit haben wir fast alle Möglichkeiten der Klangerzeugung auf dem Schneider besprochen. Es sollte keine Kompositionslehre werden, sondern Ihnen wirklich komfortable Befehle verdeutlichen. Die Musik müssen Sie schon selbst machen. Nur eine Kleinigkeit fehlt uns noch: Wie kann ich dafür sorgen, daß mir eine Melodie nicht alle Zeit stiehlt, weil ich darauf warten muß, daß ein Platz in der Schlange eines Kanals frei wird? Auch daran ist gedacht. Es gibt den Befehl

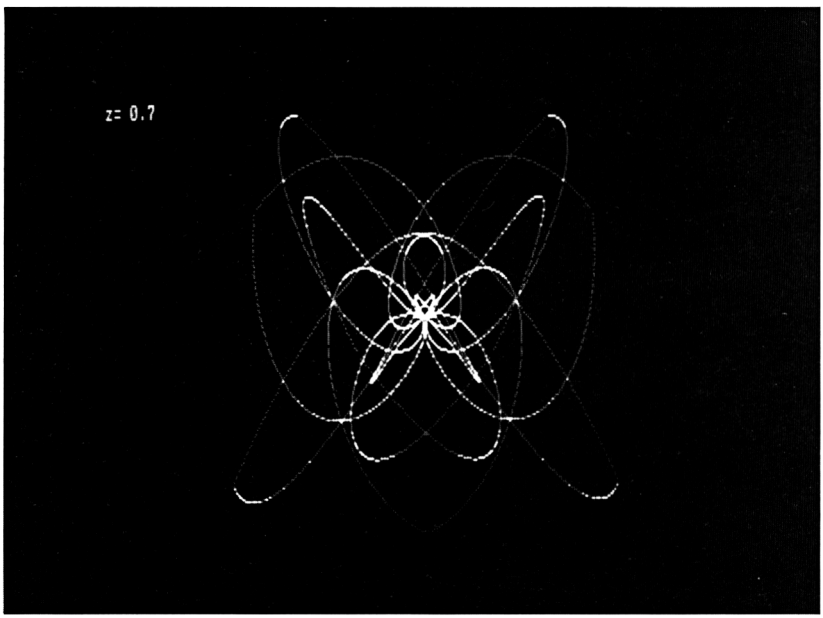
ON SQ (Kanalnummer) GOSUB Zeilennummer

Er überprüft den Zustand der Warteschlangen und verzweigt in die angegebene Zeilennummer, wenn ein Platz frei ist. Ein kleines Programm zum krönenden Abschluß:

```
10 ON SQ(1) GOSUB 1000
20 PRINT"Die Musik ist da....."
30 GOTO 10
1000 READ a,l
1010 IF l=0 THEN RESTORE
1020 SOUND 1,a,l
1030 RETURN
1035 DATA 478,40,0,1,426,40,0,1,379,40,0
,1
1040 DATA 358,40,0,1,319,80,0,1,319,80,0
,1
1050 DATA 284,40,0,1,284,40,0,1,284,40,0
,1
1060 DATA 284,40,0,1,319,80,0,1,284,40,0
,1
1070 DATA 284,40,0,1,284,40,0,1,284,40,0
,1
1080 DATA 319,80,0,1,358,40,0,1,358,40,0
,1
```

```
1090 DATA 358,40,0,1,358,40,0,1,379,40,0
,1
1100 DATA 379,80,0,1,426,40,0,1,426,40,0
,1
1110 DATA 426,40,0,1,426,40,0,1,478,160
1900 DATA 0,0
```

Ich hab' Ihnen ja gesagt, daß Sie die Musik selbst machen müssen.



Befehle in einem Zeichen

Die Steuerzeichen des Schneider, also die Zeichen mit den ASCII-Codes von 0 bis 31, bieten dem Programmierer Hilfsmittel, die in ihrer Effizienz kaum zu übertreffen sind. Innerhalb eines Programmes kann man sie auf zwei Arten einsetzen: Einmal kann man das Zeichen durch die Stringfunktion CHR$ erzeugen, und zum zweiten kann man das Zeichen innerhalb einer Stringkonstanten einsetzen, indem man die Taste CTRL zusammen mit einer anderen Taste drückt. Im folgenden Abschnitt werden alle Zeichen bis CHR$ (31) vorgestellt, ihr Erscheinungsbild auf dem Bildschirm, und – nach der Erläuterung – bei den meisten ein Beispielprogramm, das dieses Zeichen benützt. Da diese Zeichen, setzt man sie in eine Konstante ein, vom Drucker als Steuerkommandos mißverstanden werden, folgt hier erst eine Routine, durch die ein Listing erzeugt wird, in dem die Steuerzeichen durch zwei Zeichen ersetzt werden: das ^ und ein darauf folgendes anderes Zeichen. Da diese Schreibweise für Controlcodes allgemein üblich ist, ist sie auch Leuten verständlich, die weder den Schneider noch dieses Buch kennen.

```
10 OPENIN"

20 INPUT#9,a$

30 FOR I=1 TO LEN(a$)

40 b$=MID$(a$,i,1)

50 a=ASC(b$)

60 IF a<32 THEN b$="^"+CHR$(a+64)

70 c$=c$+b$

80 NEXT

90 PRINT#8,c$
```

```
100 c$=""

110 IF NOT EOF GOTO 20

120 CLOSEIN

130 END
```

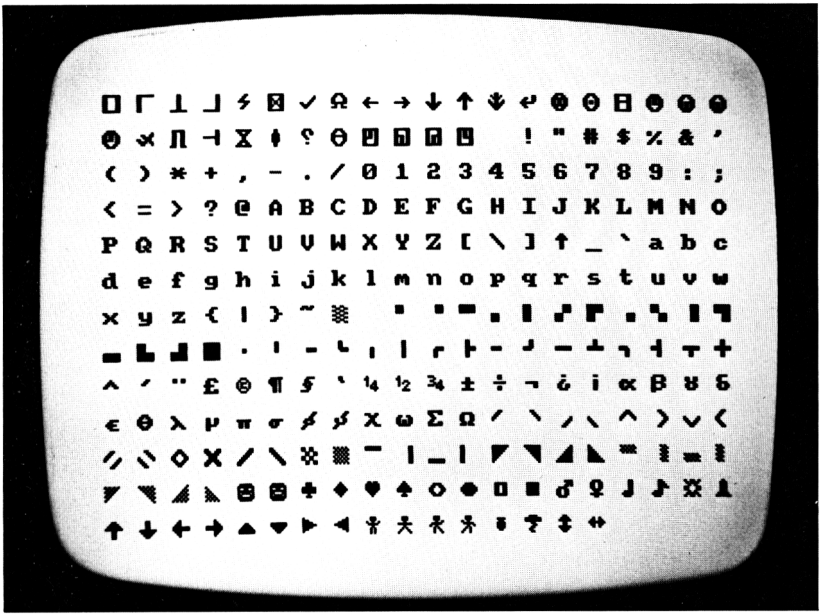
Bevor Sie dieses Programm an einem Ihrer Programme ausprobieren, sollten Sie das Programm, das Sie listen wollen, mit der Option SAVE„name",A abspeichern. Sonst würden Ihre BASIC-Wörter als 1 Byte lange Zeichen gespeichert.

Nun aber zu der Liste der Zeichen. Die Taste nach CTRL muß immer ungeshiftet bleiben – egal, ob die Taste zur Zeit Klein- oder Großbuchstaben erzeugt!

CHR$(0)
Name: Nul für Null
Erscheinungsbild:
Tasten: CTRL + @ erzeugt kein Bild auf dem Monitor!
Funktion: Keine

CHR$(1)
Name: SOH für Start of Header (Start Informationsvorspann)
Erscheinungsbild:
Tasten: CTRL + A
Funktion: CHR$(1) sorgt dafür, daß das nächste Zeichen als Zeichen interpretiert wird und nicht als Steuercode. So können alle Zeichen von 0 bis 31 ausgegeben werden, ohne daß sie ausgeführt werden.

```
10 MODE 1
20 INK 1,0
30 INK 0,26
40 BORDER 26
50 FOR i=0 TO 255
60 PRINT CHR$(1);CHR$(i);" ";
70 a=a+1:IF a=20 THEN PRINT:a=0
80 NEXT
90 IF INKEY$="" THEN 90
```



CHR$(2)
Name: STX für Start of Text
Erscheinungsbild:
Tasten: CTRL + B
Funktion: Der Cursor wird ausgeschaltet. Dieses Zeichen kann vor einem INPUT ausgegeben werden, wenn der Cursor stört. Zum Vergleich zwei INPUTs – der erste mit und der zweite ohne Cursor.

```
10 PRINT"Wie heissen Sie";
20 INPUT a$
30 PRINT"Wo wohnen Sie";CHR$(2);
40 INPUT b$
```

CHR$(3)
Name: ETX für End of Text
Erscheinungsbild:
Tasten: CTRL + C
Funktion: Der Cursor wird eingeschaltet. Gilt nur für INPUT. Eine PRINT-Ausgabe mit laufendem Cursor ist nur simuliert möglich.

```
10 PRINT"Wie heissen Sie";CHR$(2);
20 INPUT a$
30 PRINT"Wo wohnen Sie";CHR$(3);
40 INPUT b$
50 c$=a$+" "+b$
60 FOR i=1 TO LEN(c$)
70 PRINT MID$(c$,i,1)+CHR$(143)+CHR$(8);
80 FOR j=1 TO 100:NEXT
90 NEXT
100 PRINT" "
```

CHR$(4)
Name: EOT für End of Transmission (Ende der Übertragung)
Erscheinungsbild:

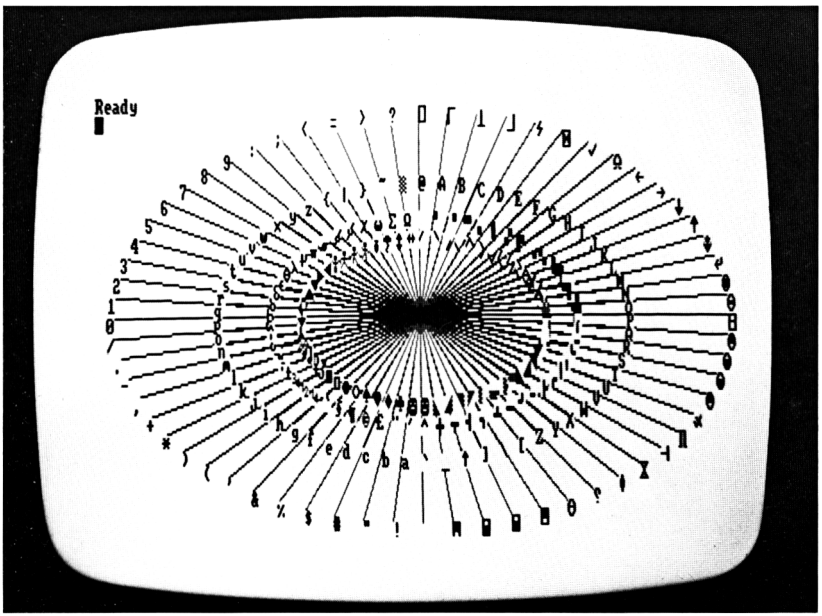
Tasten: CTRL + D
Funktion: Dieses Zeichen ist gleichbedeutend mit dem MODE – Befehl. Das Zeichen nach CHR$(4) gibt an, in welchen MODE geschaltet werden soll. Dabei kann jeder Code zwischen 0 und 255 verwendet werden. Der Code wird durch 4 geteilt, und der ganzzahlige Rest ergibt den Mode. Achtung: Sowohl ein Rest von 2 als auch ein Rest von 3 ergeben MODE 2!

```
10 FOR i=0 TO 255
20 PRINT CHR$(4);CHR$(i)
30 LOCATE 1,12
40 PRINT"Wir haben jetzt"
50 PRINT"Zeichen Nr.";i
60 PRINT
70 PRINT i"MOD 4 ="i MOD 4
80 FOR j=1 TO 800:NEXT
90 NEXT
```

CHR$(5)
Name: ENQ für Enquiry (Anfrage)
Erscheinungsbild:

Tasten: CTRL + E
Funktion: Ähnlich wie beim TAG-Befehl sorgt dieses Zeichen dafür, daß ein Zeichen auf der Graphikposition ausgegeben wird. Mit dem Transparentmodus – siehe CHR$(22) – und der Wahl des richtigen Zeichens können so bequem graphische Effekte programmiert werden.

```
10 MODE 2
20 DEG
30 ORIGIN 320,200
40 b=1
50 FOR i=0 TO 355 STEP 360/64
60 MOVE 0,0
70 DRAW 310/b*SIN(i),190/b*COS(i)
80 PRINT CHR$(5);CHR$(a);
90 IF a<255 THEN a=a+1
100 NEXT
110 IF b<2.5 THEN b=b+0.5:GOTO 50
120 END
```



CHR$(6)

Name: ACK für Acknowlege (bestätigen)
Erscheinungsbild:
Tasten: CTRL + F
Funktion: Der Textbildschirm wird wieder aktiviert, wenn er vorher mit CHR$(21) ausgeschaltet worden war. Da CHR$(6) ohne CHR$(21) keine Funktion hat, finden Sie ein Beispielprogramm dort.

CHR$(7)
Name: BEL für Bell (Glocke)
Erscheinungsbild:

Tasten: CTRL + G
Funktion: Ein Piepston ertönt, und die Tonwarteschlangen werden geleert. Das heißt, daß noch nicht gespielte Töne vergessen werden. Wie Sie wahrscheinlich schon gemerkt haben, wird CHR$(7) selbst vom Schneider dazu eingesetzt, um den Benutzer auf irgendetwas aufmerksam zu machen – meistens auf Falscheingaben. Diese Aufgabe wird beinahe immer von CHR$(7) wahrgenommen.

```
10 PRINT "Bitte druecken Sie auf die A-T
aste zusammen mit SHIFT und CTRL"
20 IF INKEY$<>"" THEN 20
30 a=INKEY(69)
40 IF a=-1 THEN 20
50 IF a=160 THEN 100
60 PRINT CHR$(7)
70 PRINT"Koennen Sie nicht Lesen?"
80 PRINT"MIT SHIFT UND CTRL!!!"
90 GOTO 20
100 IF INKEY$<>"" THEN 100
110 PRINT"Na endlich!"
120 END
```

CHR$(8)
Name: BS für Backspace (Schritt rückwärts)
Erscheinungsbild:

Tasten: CTRL + H
Funktion: Setzt den Cursor einen Schritt in der Zeile zurück (ohne das Zeichen zu löschen, das überschrieben wird!). Wird damit der Zeilenanfang überschritten, so setzt er auf das letzte Zeichen der Zeile darüber. Wäre damit die obere Bildschirm- oder Windowgrenze überschritten, wird der Bildschirm um eine Zeile nach unten gerollt. Zum Löschen eines Zeichens nimmt man die Zeichenfolge: CHR$(8) + CHR$(32) + CHR$(8).

```
10 DEF FN a$(x)=STRING$(x,8)+STRING$(x,3
2)+CHR$(13)
20 INPUT;"Ihr Name (max. 8 Zeichen)";na$
30 na=LEN(na$)
40 IF na<9 THEN 60
50 PRINT FN a$(na);:GOTO 20
60 PRINT:PRINT"Danke!"
```

CHR$(9)

Name: TAB (oft auch HT) für (horizontal) Tabulator
Erscheinungsbild:
Tasten: CTRL + I
Funktion: Setzt den Cursor und damit die nächste Druckposition um eine Stelle in der Zeile weiter. Wird damit das Zeilenende überschritten, so setzt er auf das erste Zeichen der Zeile darunter. Wäre damit die untere Bildschirm- oder Windowgrenze überschritten, wird der Bildschirm um eine Zeile nach oben gerollt. CHR$(9) bietet die Möglichkeit, relativ zu der aktuellen Schreibposition einen Punkt auf dem Bildschirm anzusteuern, ohne das LOCATE-Kommando zu benutzen.

```
10 MODE 2
20 FOR i=1 TO 400
30 PRINT".";
40 NEXT
50 LOCATE 1,1
60 INPUT;"Bitte Ihren Vornamen ",na$
70 PRINT STRING$(40-POS(#0),9);
80 INPUT;"Und Ihren Familiennamen ",nb$
```

CHR$(10)

Name: LF für Linefeed (Zeilenvorschub)
Erscheinungsbild:
Tasten: CTRL + J
Funktion: Setzt den Cursor um eine Zeile nach unten, ohne auf den Zeilenbeginn zu springen. Wird damit der untere Bildschirmrand überschritten, so wird der Bildschirm um eine Zeile nach oben gescrollt (gerollt). Wollen Sie, daß die Buchstaben genau übereinander stehen, müssen Sie vor jedem Drucken ein CHR$(8) einfügen.

```
10 MODE 2
20 FOR i=1 TO 2000
30 PRINT".";
40 NEXT
50 LOCATE 1,1
60 a$="Schneider Computer"
70 PRINT a$
80 FOR i=1 TO LEN(a$)
90 PRINT CHR$(10);
100 PRINT MID$(a$,i,1);
110 NEXT
```

CHR$(11)
Name: VT für Vertical Tabulator
Erscheinungsbild:

Tasten: CTRL + K
Funktion: Setzt den Cursor um eine Zeile nach oben, ohne auf den Zeilenbeginn zu springen. Wird damit der obere Bildschirmrand überschritten, so wird der Bildschirm um eine Zeile nach unten gescrollt.

```
10 MODE 2
20 FOR i=1 TO 2000
30 PRINT".";
40 NEXT
50 LOCATE 31,3
60 a$="Schneider Computer"
70 a=LEN(a$)
80 PRINT a$;
90 FOR i=1 TO a
100 PRINT CHR$(10);CHR$(8);
110 PRINT MID$(a$,i,1);
120 NEXT
130 PRINT CHR$(10);
140 PRINT STRING$(a,8);
150 PRINT a$;
160 PRINT STRING$(a,8);CHR$(11);
170 FOR i=1 TO a
180 PRINT MID$(a$,i,1);
190 PRINT CHR$(11);CHR$(8);
200 NEXT
```

CHR$(12)
Name: FF für Formfeed (Blattvorschub)
Erscheinungsbild:

Tasten: CTRL + L
Funktion: Löscht den aktuellen Ausgabebereich und setzt den Cursor in die linke obere Ecke. CLS ist gleichbedeutend mit PRINT CHR$(12);

CHR$(13)
Name: CR für Carriage Return (Wagenrücklauf)
Erscheinungsbild:

Tasten: ENTER oder CTRL + M

Funktion: Setzt bei der Druckausgabe den Cursor auf den Beginn der aktuellen Zeile (Achtung! CR löst keinen Zeilenvorschub aus!).

```
10 LINE INPUT"Ihre Nachricht? ";a$
20 PRINT CHR$(12);
30 a$=a$+CHR$(32)
40 FOR i=1 TO LEN(a$)
50 PRINT MID$(a$,i,20)+CHR$(13);
60 NEXT
```

CHR$(14)

Name: SO für Shift Out (Großschreibung aus)
Erscheinungsbild:
Tasten: CTRL + N
Funktion: Setzt die Farbstiftnummer für den aktuellen Hintergrund. Das Zeichen nach SO gibt die INK an, die verwendet werden soll. Dabei wird der ASCII-Code des Zeichens durch 16 geteilt, der ganzzahlige Rest ist das verwendete Argument.

```
10 MODE 0
20 a$=INKEY$:IF a$="" THEN 20
30 PRINT CHR$(14);a$;" ";
40 GOTO 20
```

CHR$(15)

Name: SI für Shift In (Großschreibung ein)
Erscheinungsbild:
Tasten: CTRL + O
Funktion: Setzt die Farbstiftnummer für die aktuelle Schreibfarbe. Das Zeichen nach SI gibt die INK an, die verwendet werden soll. Dabei wird der ASCII-Code des Zeichens durch 16 geteilt, der ganzzahlige Rest ist das verwendete Argument.

```
10 MODE 0
20 a$=INKEY$:IF a$="" THEN 20
30 a=ASC(a$)
40 PRINT CHR$(14);a$;
50 PRINT CHR$(15);CHR$(255-a);a$;
60 GOTO 20
```

CHR$(16)
Name: DLE für Data Link Escape (Datenverbund Ende)
Erscheinungsbild:
Tasten: CTRL + P
Funktion: Löscht das Zeichen unter dem Cursor und füllt die leere Stelle mit der aktuellen Hintergrundfarbe.

```
10 MODE 0
20 a$="Schneider Computer"
30 a=LEN(a$)
40 FOR i=1 TO a
50 PRINT a$;
60 FOR j=1 TO i
70 PRINT CHR$(8);
80 NEXT
90 PRINT CHR$(16)
100 NEXT
```

CHR$(17)
Name: DC1 für Device Control 1 (Gerätekontrolle 1)
Erscheinungsbild:
Tasten: CTRL + Q
Funktion: Löscht die aktuelle Zeile vom Beginn bis zur Cursorposition. Die leeren Stellen werden mit der aktuellen Hintergrundfarbe gefüllt.

```
10 MODE 0
20 a$="Schneider Computer"
30 a=LEN(a$)
40 FOR i=1 TO a
50 PRINT a$;
60 FOR j=1 TO i
70 PRINT CHR$(8);
80 NEXT
90 PRINT CHR$(17)
100 NEXT
110 PRINT a$
```

CHR$(18)
Name: DC2 für Device Control 2
Erscheinungsbild:
Tasten: CTRL + R

Funktion: Löscht die aktuelle Zeile ab Cursorposition. Die leeren Stellen werden mit der Hintergrundfarbe gefüllt.

```
10 PRINT"Ihren Namen bitte"+CHR$(18);
20 INPUT;"";a$
30 IF LEN(a$)>8 THEN PRINT CHR$(13);:GOT
O 10
```

CHR$(19)
Name: DC3 für Device Control 3
Erscheinungsbild:

Tasten: CTRL + S
Funktion: Löscht den gesamten Bildschirm bis zur Cursorposition in der aktuellen Hintergrundfarbe.

CHR$(20)
Name: DC4 für Device Control 4
Erscheinungsbild:

Tasten: CTRL + T
Funktion: Löscht den gesamten Bildschirm ab Cursorposition in der aktuellen Hintergrundfarbe.

```
10 MODE 2
20 a$="Schneider"
30 b$="Computer"
40 FOR i=1 TO 72
50 PRINT a$,
60 NEXT
70 FOR i=1 TO 72
80 PRINT b$,
90 NEXT
100 LOCATE 1,12
110 PRINT"<O>bere oder <U>ntere Haelfte
loeschen"+CHR$(18);
120 c$=LOWER$(INKEY$)
130 IF c$="o" THEN PRINT CHR$(19):GOTO 1
60
140 IF c$="u" THEN PRINT CHR$(20):GOTO 1
60
150 GOTO 120
160 FOR i=1 TO 2000:NEXT
```

```
170 LOCATE 1,1
180 GOTO 20
```

CHR$(21)

Name: NAK für Negative Acknowledge (Negative Bestätigung)
Erscheinungsbild:
Tasten: CTRL + U
Funktion: Schaltet alle PRINT-Vorgänge auf den Bildschirm ab, bis CHR$(6) gedruckt wird. Ermöglicht eine verdeckte Eingabe mit INPUT.

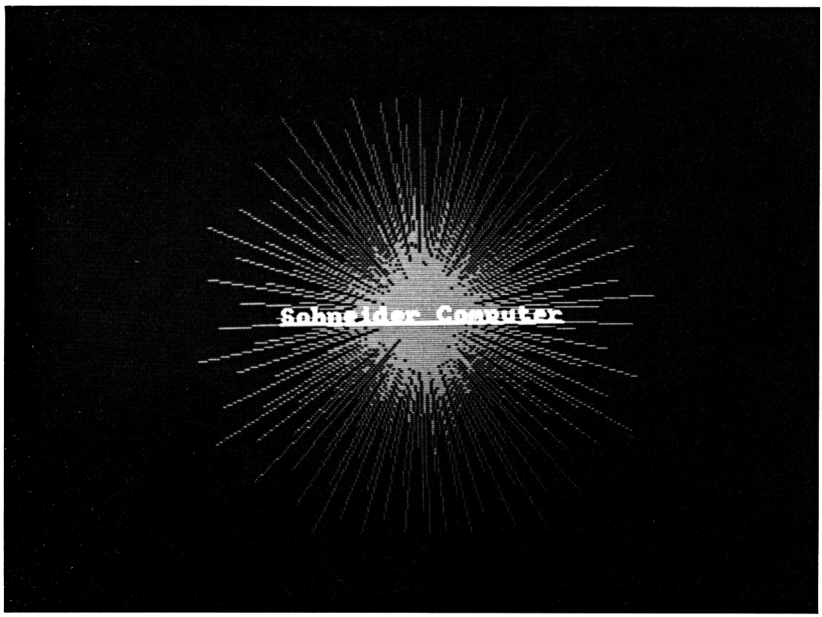
```
10 PRINT"Bitte geben Sie Ihr Passwort ei
n"+CHR$(21)
20 INPUT pa$
30 PRINT CHR$(6)
40 IF pa$<>"Pass" THEN PRINT CHR$(7);:GO
TO 10
50 PRINT "Vielen Dank, Sie haben das Pro
grammende erreicht!"
```

CHR$(22)

Name: SYN für Synchronization (Synchronisation)
Erscheinungsbild:
Tasten: CTRL + V
Funktion: Setzt zusammen mit dem nächsten Zeichen den Transparentmodus. Wenn das nächste Zeichen einen ungeraden ASCII-Code hat, wird der Transparentmodus (für den ganzen Bildschirm) eingeschaltet. Ein gerader ASCII-Code stellt den Normalzustand wieder her – dann kann in einer Druckposition nur ein Buchstabe stehen. Buchstaben, die sich überlagern, können nicht mit dem COPY-Cursor übernommen werden. Der Normalzustand wird auch durch eine MODE-Anweisung wieder hergestellt. Die häufigste Anwendung wird SYN wohl bei der Mischung von Graphik und Text finden.

```
10 MODE 1
20 a$="Schneider Computer"
30 b$="__________________"
40 LOCATE 12,13
50 PRINT a$+CHR$(22)+CHR$(1)
60 LOCATE 12,13
70 PRINT b$
80 DEG
90 ORIGIN 320,200
100 FOR i=0 TO 7200 STEP 30
```

```
110 j=j+1
120 PLOT 0,0
130 DRAW j*SIN(i+j),j*COS(i+j),3
140 NEXT
150 PEN 2
160 LOCATE 12,13
170 PRINT a$
180 LOCATE 12,13
190 PRINT b$
200 IF INKEY$="" THEN 200
```



CHR$(23)

Name: ETB für End of Text Block (Ende eines Text-Blocks)

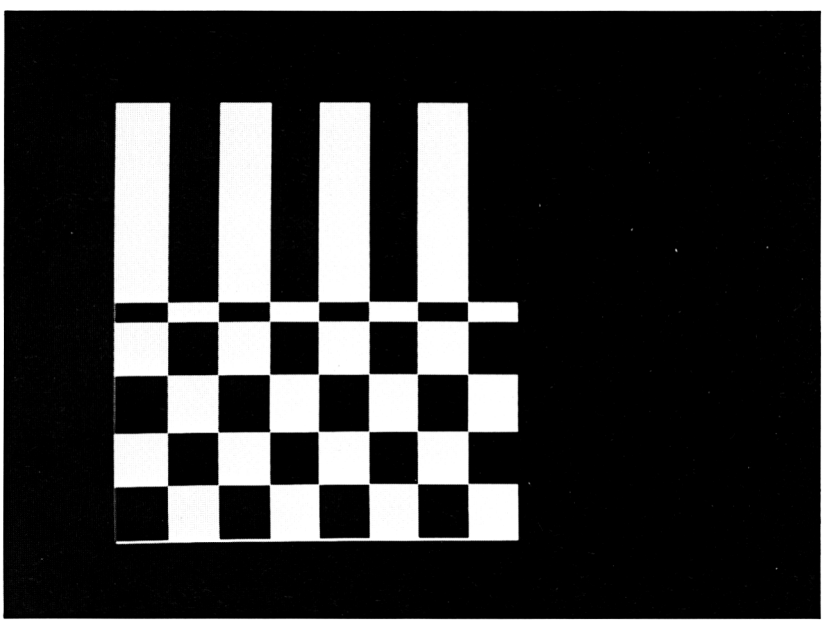
Erscheinungsbild:

Tasten: CTRL + W

Funktion: Setzt mit dem folgenden Zeichen den Modus für Graphikausgaben. Ist das folgende Zeichen ohne Rest durch 4 teilbar, so wird der Normalmodus gewählt. Ergibt sich ein Rest von 1, so wird ein Punkt, der in der gleichen Farbe schon existiert, ausgeschaltet, und ein Punkt, der nicht existiert, angeschaltet. In MODE 2 kann man so ein

negatives Bild erzeugen. Achten Sie darauf, daß beim Zeichnen in vertikaler Richtung immer eine Schrittweite 2 genommen wird, da Sie sonst einen Punkt direkt nacheinander ein- und wieder ausschalten.

```
10 MODE 2
20 FOR i=0 TO 6 STEP 2
30 FOR j=0 TO 50
40 PLOT i*50+j,0
50 DRAW i*50+j,400
60 NEXT j
70 NEXT i
80 PRINT CHR$(23)+CHR$(1)
90 FOR i=0 TO 6 STEP 2
100 FOR j=0 TO 50 STEP 2
110 PLOT 0,i*50+j
120 DRAW 400,i*50+j
130 NEXT j
140 NEXT i
```



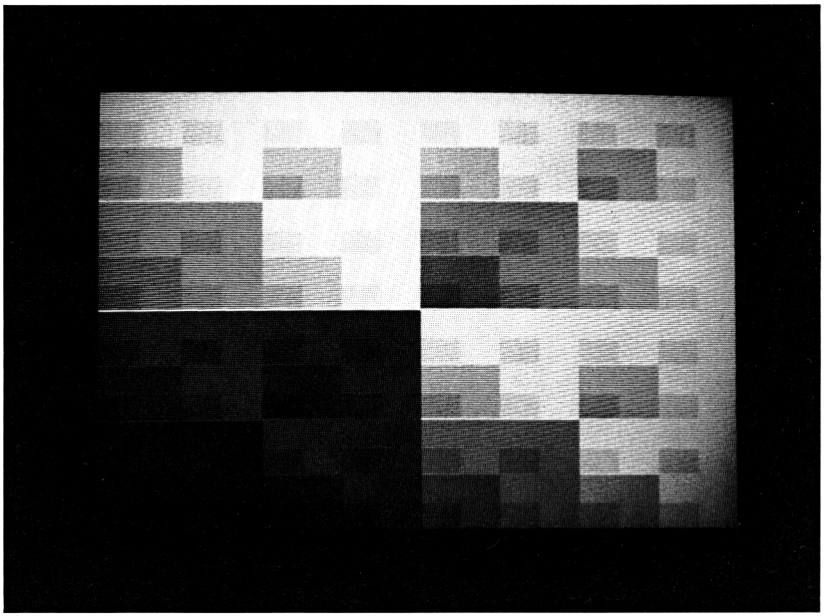
Nehmen Sie in Zeile 100 mal die Schrittweite 1 und schauen Sie sich dann Ihr Schachbrett an.

Ergibt sich bei der Teilung ein Rest von 2, so wird nur dann gezeichnet, wenn ein Punkt schon gezeichnet ist. Sie können so eine fertige Zeichnung anders einfärben, ohne die gesamte Zeichnung noch einmal durchführen zu müssen. Es wird dann in der Farbe gezeichnet, die sich aus der AND-Verknüpfung des Originalpunktes und des neu zu zeichnenden Punktes ergibt. Also gilt: Wenn Sie die Zeichnung in Farbe 8 durchgeführt haben und dann mit Farbe 7 darüberzeichnen, wird die Originalzeichnung gelöscht, weil 7 AND 8 gleich 0 ist.

```
10 MODE 0
20 PRINT CHR$(23)+CHR$(0)
30 g=200
40 f=15
50 ORIGIN 320,200
60 DEG
70 FOR i=0 TO 360
80 PLOT 0,0,15
90 DRAW g*SIN(i),g*COS(i)
100 NEXT
110 PRINT CHR$(23)+CHR$(2)
120 f=f-1
125 LOCATE 1,1
126 PRINT f
130 FOR i=-g TO +g STEP 2
140 PLOT -g,i,f
150 DRAW g,i
160 NEXT
170 IF f>0 GOTO 120
```

Etwas seltsam sind die Auswirkungen von ETB, wenn bei der Teilung ein Rest von 3 bleibt. Im Gegensatz zu der Information im Handbuch werden dann die Farben nicht durch ein XOR, sondern durch ein OR verknüpft. Ein Punkt der Farbe 0 wird also in der angegebenen Farbe gezeichnet, ein Punkt, der schon vorher die Farbe 8 hatte, und mit der Farbe 7 übermalt wird, erhält die Farbe 15.

```
10 MODE 0
20 FOR i=0 TO 15
30 INK i,i+10
40 NEXT
50 PRINT CHR$(23)+CHR$(0)
```



```
60 FOR i=0 TO 15
70 FOR j=0 TO 40
80 MOVE i*40+j,0
90 DRAW i*40+j,400,i
100 NEXT
110 NEXT
120 ORIGIN 0,0
130 PRINT CHR$(23)+CHR$(3)
140 FOR i=0 TO 15
150 FOR j=0 TO 25
160 MOVE 0,i*25+j
170 DRAW 640,i*25+j,i
180 NEXT
190 NEXT
200 IF INKEY$="" THEN 200
210 INK 1,26:INK 0,0
```

CHR$(24)
Name: CAN für Cancel (stornieren)
Erscheinungsbild:

Tasten: CTRL + X
Funktion: CAN ist der Reverseschalter, der Ihnen beim Schneider zur Verfügung steht. Vor allen Dingen in der kommerziellen Datenverarbeitung wird das reverse Ausdrucken von Texten zur Markierung von Informationen genützt. Bei jedem Gebrauch von CAN werden Schreib- und Hintergrundfarbe vertauscht.

```
10 MODE 1
20 a$="Schneider Computer"
30 FOR i=0 TO 15
40 PEN i
50 PAPER 15-i
60 PRINT a$;
70 PRINT CHR$(24);
80 PRINT a$;
90 PRINT CHR$(24);
100 NEXT
```

CHR$(25)
Name: EM für End of Medium
Erscheinungsbild:

Tasten: CTRL + Y
Funktion: Mit diesem Zeichen können Sie genau wie mit dem SYMBOL-Befehl benutzereigene Zeichen bilden. Voraussetzung ist aber auch hierbei, daß Sie mit dem SYMBOL AFTER-Kommando Speicherplatz dafür reservieren. Statt Zahlen (wie bei SYMBOL) müssen auf EM neun Zeichen folgen. Das erste Zeichen ist das, welches geändert werden soll, die weiteren geben die Bitmaske für die oberste bis unterste Reihe an. Die Programmierung ist nicht besonders platzsparend, da nur die Zeichen von 0 bis 127 über die Tastatur erreichbar sind, man viele Zeichen also mit CHR$(x) erzeugen muß.

CHR$(26)
Name: SUB für Substitute (Ersatz)
Erscheinungsbild:

Tasten: CTRL + Z
Funktion: Dieses Kommando funktioniert ähnlich wie das WINDOW-Kommando, allerdings mit einigen Unterschieden: Im Gegensatz zu WINDOW liegt die oberste linke Ecke nicht bei 1,1, sondern bei 0,0. Wenn Sie ein bestimmtes WINDOW schaffen wollen, müssen Sie es mit

```
PRINT#2,CHR$(26)+CHR$(0)+CHR$(10)+CHR$(0)+CHR$(20)
```

erzeugen. Dieses Fenster geht von den Positionen 1,1 bis zur Bildschirmposition 11,21.

CHR$(27)
Name: ESC für Escape (Abbruch)
Erscheinungsbild:

Tasten: CTRL + (eckige Klammer auf)
Funktion: Keine. Da ESC bei den meisten Druckern Steuerzeichen einleitet, ist es sehr sinnvoll, dieses Zeichen nicht für den Bildschirm zu nützen. So kann man eine Druckausgabe erst auf den Bildschirm schicken und erkennt dann die Steuerzeichen als Buchstaben.

CHR$(28)
Name: FS für File Separator (Datei-Trenner)
Erscheinungsbild:

Tasten: CTRL + \
Funktion: Wie INK-Kommando, nur daß hier Parameter von 0 bis 255 erlaubt sind. Der Computer sorgt durch eine Devision mit der erlaubten Anzahl dafür, daß der ganzzahlige Rest als Parameter gewertet wird. Hier müssen immer beide Farben angegeben werden, da sonst das nächste Zeichen als Parameter benutzt wird. Will man kein Blinken erzeugen, muß man zweimal das gleiche Zeichen als Parameter einsetzen. Es ist nicht möglich, mit diesem Zeichen nur die INK-Zuweisung für ein WINDOW zu ändern.

CHR$(29)
Name: GS für Group Separator (Gruppen-Trenner)
Erscheinungsbild:

Tasten: CTRL +]
Funktion: Setzt das Farbpaar für den Bildschirmrand. Auch hier müssen beide Farben angegeben werden.

CHR$(30)
Name: RS für Record Separator (Satz-Trenner)
Erscheinungsbild:

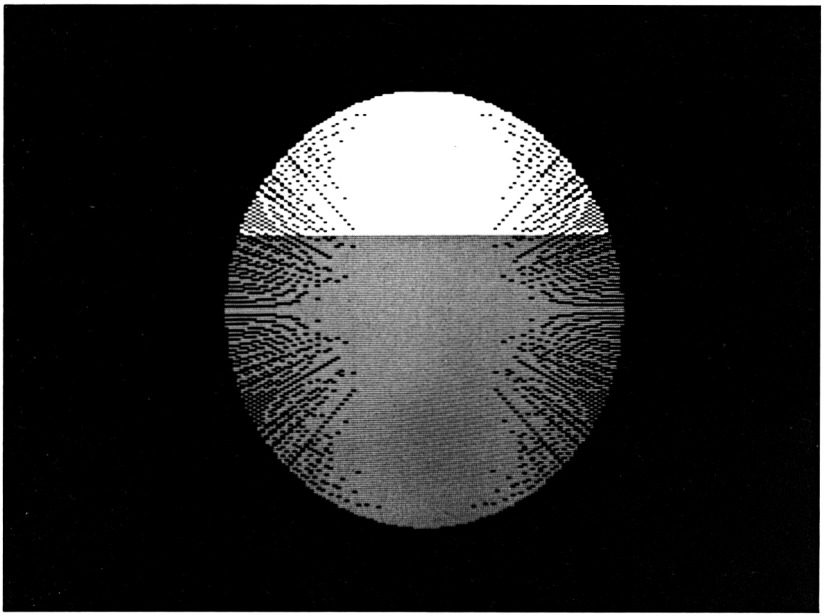
Tasten: CTRL +^
Funktion: Setzt den Cursor in die linke obere Ecke des Bildschirms bzw. des angesprochenen WINDOWS, ohne etwas zu löschen. Meiner Ansicht nach eine so praktische Geschichte, daß ich eine Funktionstaste damit belegt habe. Bewegt man den Cursor mit den Pfeiltasten nach oben, so schießt man oft über das Ziel hinaus. So geht es besser.

CHR$(31)
Name: US für Unit Separator
Erscheinungsbild:

Tasten: CTRL + _
Funktion: Wie LOCATE. Die beiden nächsten Zeichen bestimmen die Stelle, an der der

nächste Druckvorgang stattfindet. Das erste Zeichen die X-Position, das zweite die Y-Position. X-Werte von 80 bis 208 sorgen dafür, daß vor dem nächsten Drucken ein CHR$(13) ausgeführt wird. Ab 209 wird vor dem Drucken der Bildschirm nach unten gescrollt. Das gleiche geschieht bei X=0. Ein Y-Wert 0 sorgt ebenfalls dafür – und die Y-Werte ab 129 aufwärts. Y-Werte von 25 bis 128 lassen den Bildschirm vor dem Drucken nach oben scrollen.

Ein Wort noch zu den Namen der Steuerzeichen. ASCII-Code heißt »American Standard Code for Information Interchange«, also ein Code zum Informationsaustausch zwischen Computern. Damit die Daten auch genauso empfangen werden, wie man sie sendet, werden zur Steuerung des Informationsflusses die Zeichen von 0 bis 31 verwendet. Welche Aufgabe die Zeichen im einzelnen haben, deutet ihr Name an.



Und dann gab's da noch die Sonderzeichen ....

In diesem Kapitel stelle ich Ihnen (hoffentlich) alle die Zeichen vor, die sonst noch in einem Programm enthalten sein können, wenn man von den Buchstaben und Zahlen mal absieht, deren Bedeutung Ihnen ja klar ist (weil Sie sonst immer noch nur vor dem Fernseher säßen und sich nicht der Strapaze des Programmieren aussetzen wollten). Sie sind nach Ihrem ASCII-Code geordnet. Neben die Zeichen, die in einem deutschen Listing anders aussehen können, habe ich noch die Entsprechung aus dem deutschen Zeichensatz geschrieben. Zeichen, die nicht in dieser Liste erwähnt sind, dürfen ausschließlich in Strings verwendet werden.

!
Dieses Zeichen dient zur Kennzeichnung von Gleitkommavariablen, wenn durch DEFINT oder DEFSTR bestimmte Variablennamen als String- oder Integervariablen definiert wurde.

"
Bezeichnet in BASIC den Anfang oder das Ende von Stringkonstanten.

#
(No.sign) Wird im Zusammenhang mit dem Ansprechen bestimmter Ein- oder Ausgabeeinheiten genutzt.

$
(Dollarzeichen) Dient zur Kennzeichnung von Stringvariablen oder Stringfunktionen.

%
(Prozentzeichen) Dient zur Kennzeichnung von Integervariablen (Ganzzahlvariablen).

&
(Et-Zeichen) definiert eine Zahl als zum Hexadezimalsystem gehörig.

&x
Definiert eine Zahl als zum Dualsystem gehörig.

'

(Apostroph) entspricht dem BASIC-Befehl REM.

( )

Fehlende Klammerzeichen werden durch BASIC als Syntax Error abgewiesen. Achtung! Andere Klammern sind in BASIC innerhalb von Ausdrücken nicht erlaubt. Nochmal Achtung! Mehr als 52 nicht aufgelöste Klammern lassen Ihren Rechner abstürzen!

*

(Malzeichen) Arithmetischer Operator. Die Werte rechts und links von * werden durch eine Multiplikation verknüpft.

+

(Pluszeichen) Arithmetischer Operator. Die Werte rechts und links von + werden durch eine Addition verknüpft – es sei denn, eine höhere Rechenart hätte Vorrang.

,

(Kommazeichen) Trennungszeichen. Dient bei vielen Befehlen zur Trennung der Parameter voneinander.

–

(Minuszeichen) Arithmetischer Operator. Der Wert rechts vom Minuszeichen wird vom Wert links des Minuszeichens subtrahiert. Dient bei einigen Systembefehlen zur Beschreibung eines Zeilennummernbereiches. Achtung! Vorsicht beim Benutzen des Minuszeichens zum Unterstreichen: Mehr als 85 zu verarbeitende Minuszeichen führen nicht zu einer Fehlermeldung, sondern zum Absturz.

.

(Punkt) Ausschließliche Bedeutung: Dezimalpunkt.

/

(Divisionszeichen) Arithmetischer Operator. Der Wert links vom Divisionszeichen wird durch den Wert rechts des Divisionszeichens geteilt.

:

(Doppelpunkt) trennt in BASIC zwei Befehle innerhalb einer Programmzeile voneinander. (Kein Divisionszeichen!)

;

(Semikolon) Trennungszeichen. In einer Druckliste werden die einzelnen Teile durch Semikolon getrennt, wenn sich sonst Mißverständnisse bei der Interpretation der Liste

ergeben könnten, und die Teile direkt nacheinander ausgedruckt werden sollen. Paßt ein Teil der Druckliste nicht mehr komplett in die aktuelle Zeile, so werden Carriage Return und Line Feed ausgelöst.

<
(Kleinerzeichen) Logischer Operator. Es wird bei Einsatz dieses Zeichens überprüft, ob der Wert des Ausdrucks links vom < kleiner ist als der Wert des Ausdrucks rechts davon.

=
(Gleichheitszeichen) Hat in BASIC zwei Bedeutungen:
1. Logischer Operator; es wird bei Einsatz dieses Zeichens überprüft, ob die Werte der Ausdrücke rechts und links davon identisch sind.
2. Zuweisungszeichen; der Wert des Ausdrucks rechts vom Gleichheitszeichen wird in die Variable links vom Gleichheitszeichen gefüllt.

>
(Größerzeichen) Logischer Operator. Es wird bei Einsatz dieses Zeichens überprüft, ob der Wert des Ausdrucks links vom > größer ist als der Wert des Ausdrucks rechs davon.

?
(Fragezeichen) Kann in BASIC zur Abkürzung des PRINT-Befehls verwendet werden. Wird beim Listen in das Wort PRINT verwandelt.

@) oder §
(Klammeraffe) übergibt bei Diskettenbefehlen den Inhalt von Variablen an das Betriebssystem.

\ oder Ö
Dieses Zeichen liefert bei einer Division den ganzzahligen Anteil des Ergebnisses. Die Operanden dürfen keine Werte größer als 65535 annehmen; ein Overflow wäre die Folge.

^
(Hochzeichen) Arithmetischer Operator. Der Wert des Ausdrucks links vom^ wird mit dem Wert des Ausdrucks rechts davon potenziert. Erscheint auf dem Bildschirm als Pfeil nach oben.

| oder ö
(senkrechter Strich) leitet benutzereigene Befehle oder Diskettenbefehle ein.

Wörterbuch BASIC-Deutsch

Es folgt nun eine alphabetische Liste der BASIC-Wörter mit Anwendungsbeispielen, die den tatsächlichen Nutzen klarmachen und als Programmtips und Hilfsroutinen nützlich sein sollen. Einige BASIC-Wörter nennt man auch Funktionen; Funktionen ermitteln einen Wert – bzw. einen String – durch einen mathematischen Rechenvorgang aus einem anderen. Die Funktion y=sin(x) ermittelt aus dem Wert x (dem Argument) durch eine komplizierte Formel den Wert y. x und y haben also verschiedene Werte. Das muß nicht sein: Wenden Sie diese Formel auf den Wert 0 an, so erhalten Sie für y auch den Wert 0. y ist also von x abhängig. Funktionen sind keine Befehle! Sie dürfen nicht am Anfang einer BASIC-Zeile stehen.

Vorher sollten Sie sich noch mit den Begriffen Ganzzahl- oder Integerfunktion und Fließ- oder Gleitkommafunktion vertraut machten. Wichtig zur Beurteilung, ob es sich bei einer Funktion um eine Integer- oder eine Fließkommafunktion handelt, ist nicht das Argument, sondern das Ergebnis. Wenn das Ergebnis einer Funktion immer unverändert in eine Integervariable paßt, handelt es sich um eine Integerfunktion, sonst um eine Gleitkommafunktion. Ein gutes Beispiel dafür ist die Funktion PEEK(x), die als Ergebnis den Inhalt der Speicherstelle x liefert. x kann Werte annehmen zwischen 32768 und 65535, muß also kein Integerwert sein. Ergebnis von PEEK(x) wird aber immer eine Zahl zwischen 0 und 255 sein. Dieser Unterschied ist aus zwei Gründen wichtig. Integervariablen nehmen wesentlich weniger Platz im Speicher ein als Fließkommavariablen, und sie können auch schneller bearbeitet werden. Wenn Sie mehr über den Unterschied wissen wollen, schauen Sie auf S. 17 nach.

Erklärung der Abkürzungen:

gA ganzzahliger Ausdruck
gkA Gleitkommaausdruck
nA numerischer Ausdruck
sA Stringausdruck
znr Zeilennummer
〈 〉 Teile von Befehlen oder Funktionen, die bei der Erklärung in 〈 〉 stehen, können weggelassen werden, wenn sie nicht benutzt werden.

ABS (nA)

Gleitkommafunktion: Liefert den absoluten Wert des numerischen Ausdrucks. Die mathematische Schreibweise Y=|X| wird umgesetzt in Y=ABS(X).

Verwendungsmöglichkeiten: Um zu vermeiden, daß beim Wurzelziehen aus einer negativen Zahl die Fehlermeldung »Improper argument« erscheint, zieht man Wurzeln aus dem absoluten Wert, um den reellen Anteil des Ergebnisses zu erhalten:

```
10 INPUT"Irgendeine Zahl";z
20 x=ABS(z)
30 PRINT"Die Wurzel aus";z;" ist";x^(1/2)
```

Durch Überprüfung des Vorzeichens kann man nun auch noch, wenn nötig, andeuten, daß die Zahl einen imaginären Anteil hat:

```
10 a=-100
15 PRINT a
20 b=ABS(a)
25 PRINT b
30 c=b^(1/2)
35 PRINT c;
40 IF a<0 THEN PRINT"i" ELSE PRINT
```

Weitere Verwendung ergibt sich im kaufmännischen Bereich, wenn Soll oder Guthaben als positive Zahlen erscheinen sollen:

```
10 INPUT"Soll";s
20 INPUT"Haben";h
30 saldo=h-s
40 PRINT TAB(10);"Guthaben";TAB(20);"Schulden"
50 IF saldo<0 THEN PRINT TAB(20); ELSE PRINT TAB(1
60 PRINT ABS(saldo)
```

AFTER gA1, gA2 **GOSUB** znr

Befehl: Ruft nach Ablauf der in gA1 festgelegten Zeit (gemessen wird im Zeitgeber gA2 zu Einheiten von 0.02 sec.) das Unterprogramm auf, das bei znr beginnt.

gA1 darf Werte zwischen 0 und 32767, gA2 Werte von 0 bis 3 annehmen; wird er weggelassen, so wird Zeitgeber 0 benutzt.
Die Zeilennummer wird sofort geprüft, nicht erst nach Ablauf der Zeit.

Verwendungsmöglichkeit: Nach Ablauf einer Wartezeit können dem neuen Benutzer eines Programmes Hilfen zur Verfügung gestellt werden, die den erfahrenden Benutzer nur stören würden.

```
10 CLS
20 PRINT"bitte druecken Sie eine Taste"
30 AFTER 100,1 GOSUB 1000
40 a$=INKEY$
50 IF a$="" THEN 40
60 LOCATE 11,12:PRINT "     Danke!    ";
70 END
1000 LOCATE 11,12:PRINT"Schlafen Sie?";
1010 RETURN
```

gA AND gA

Operator: Verknüpft die binäre Entsprechung der ganzzahligen Ausdrücke durch ein logisches AND und liefert das Ergebnis.

Sollten sich bei der Auswertung des gA Nachkommastellen ergeben, so werden sie ignoriert.

Verwendungsmöglichkeit: Grundsätzlich gibt es zwei Hauptverwendungen für AND. Zum einen wird es benutzt, um bei Byteoperationen einzelne Bits zu löschen und die anderen unverändert zu lassen; im folgenden Beispiel sollen die ersten vier Bit der Speicherstelle 160 gelöscht werden, während die letzten vier erhalten bleiben sollen.

```
10 POKE 160,135
20 PRINT BIN$(PEEK(160),8)
30 PRINT BIN$(15,8)
40 POKE 160,PEEK (160) AND 15
50 PRINT BIN$(PEEK(160),8)
```

Die zweite Möglichkeit ergibt sich aus der Tatsache, daß der Computer aus einer wahren Bedingung den Wert −1 macht.

```
PRINT 100=100
-1
Ready
PRINT 100=50
0
Ready
PRINT -1 AND 0
```

```
0
Ready
PRINT 100=100 AND 100=50
0
Ready
```

Der letzte Wert wird in einer IF-Abfrage geprüft, in der mehrere Bedingungen mit AND verknüpft sind. Nur wenn alle Bedingungen wahr sind, also Ihre Verknüpfung den Wert –1 ergibt, wird der THEN-Zweig ausgeführt.

ASC(sA)

Ganzzahlfunktion: Liefert den ASCII-Wert des ersten Zeichens des Stringausdrucks.

Der Stringausdruck muß mindestens ein Zeichen enthalten, um Verwechslungen mit CHR$(0) zu verhindern. Ein Leerstring, d. h. ein String ohne Inhalt, erzeugt eine Fehlermeldung »Improper argument«.

Verwendungsmöglichkeit: Oft hat man Menues mit mehr als 9 Punkten. Da die ASCII-Codes der Buchstaben aufsteigend numeriert sind, kann man sie benutzen, um einzelne Programmteile anzuspringen.

```
10 CLS
20 PRINT"<A> Eingabe"
30 PRINT"<B> Loeschen"
40 PRINT"<C> Speichern"
50 PRINT"<D> Ende"
60 a$=INKEY$:IF a$="" THEN 60
70 ON ASC(UPPER$(a$))-64 GOSUB 1000,2000,3000,4000
80 GOTO 10
1000 PRINT"Programmpunkt <A>"
1010 IF INKEY$="" THEN 1010
1020 RETURN
2000 PRINT"Programmpunkt <B>"
2010 IF INKEY$="" THEN 2010
2020 RETURN
3000 PRINT"Programmpunkt <C>"
3010 IF INKEY$="" THEN 3010
3020 RETURN
4000 PRINT"Tschuess.."
4010 END
```

Wenn man mal wieder vergessen hat, welche Taste welchen ASCII-Code hat, hilft folgendes Prográmmchen:

```
10 a$=INKEY$:IF a$="" THEN 10
20 PRINT a$;ASC(a$): GOTO 10
```

(Versuchen Sie mal die Tastenkombinationen von CTRL plus den Cursortasten.)

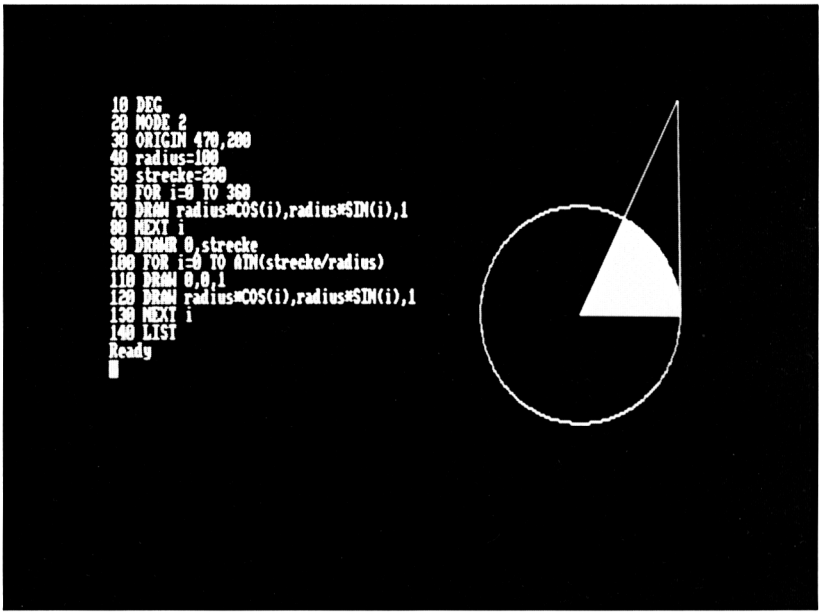
ATN(nA)

Gleitkommafunktion: Liefert den Arcustangens des numerischen Ausdrucks. Das Ergebnis liegt immer zwischen −pi/2 und +pi/2, wenn Sie mit Bogenmaß rechnen, bzw. zwischen −90 und +90, wenn mit Grad gerechnet wird.

Noch mal zur Erinnerung: Der Arcustangens einer Strecke ist der Winkel, der sich ergibt, wenn man die Strecke als Senkrechte auf den Schnittpunkt Radius/Kreis setzt und den anderen Endpunkt der Strecke mit dem Mittelpunkt des Kreises verbindet.

```
10 DEG
20 MODE 2
30 ORIGIN 470,200
40 radius=100
50 strecke=200
60 FOR i=0 TO 360
70 DRAW radius*COS(i),radius*SIN(i),1
80 NEXT i
90 DRAWR 0,strecke
100 FOR i=0 TO ATN(strecke/radius)
110 DRAW 0,0,1
120 DRAW radius*COS(i),radius*SIN(i),1
130 NEXT i
```

Zur Erläuterung:
10–50 Initialisieren der Variablen.
60–80 Zeichnen von Radius und Kreis.
90 Zeichnen der Senkrechten.
100–130 Ausfüllen des Kreisausschnittes, der durch den Arcustangens von Strecke bestimmt wird.



AUTO ⟨Znr⟩, ⟨Schrittweite⟩

Systemkommando: Gibt im Programmodus automatisch Zeilennummern vor, beginnend bei Znr. Die jeweils nächste Zeilennummer wird bestimmt durch die Schrittweite. Standard- oder Defaultwert für Znr und Schrittweite ist jeweils 10. Wenn nach der Zeilennummer ein Stern (*) erscheint, ist die Zeilennummer schon belegt.

Es ist nicht möglich, beim Gebrauch von AUTO die vorgegebene Zeilennummer zu ändern. Jedes eingegebene Zeichen wird als Programmtext interpretiert.

AUTO 100,50 erzeugt nacheinander die Zeilennummern 100,150,200, usw. Wollen Sie eine Zeilennummer nicht benutzen, so drücken Sie einfach ENTER. Das Verlassen von AUTO ist nur durch einmaliges Drücken von ESC möglich.

BIN$ ⟨gA1, gA2⟩

Stringfunktion: Liefert als Textausdruck die Binärzahl, die sich aus dem Argument gA1 ergibt.

Im Gegensatz zu der im Handbuch angegebenen Beschränkung »vorzeichenlos« darf gA1 im Bereich von −32768 bis 653535 liegen. Allerdings liefert der Bereich von

–32768 bis –1 die gleichen Zahlen wie der Bereich von +32768 bis 65535 (genauere Informationen finden Sie ab S. 49).

gA2 darf nur im Bereich von 1 bis 16 liegen, wenn Sie ihn benutzen. Er füllt den Textausdruck, wenn dieser kürzer ist, als gA2 angibt, mit führenden Nullen auf. Die Mißachtung der Grenzen für gA1 führt zu »Overflow«, der für gA2 zu »Improper argument«.

Verwendungsmöglichkeit: Neben der Möglichkeit, mit Hilfe von BIN$ Vorgänge im Computer zu verdeutlichen (siehe AND), ist der BIN$ bei der Erstellung von benutzereigenen Zeichen von unschätzbarem Wert, da man ihm den bisherigen Inhalt eines Zeichens sichtbar machen kann.

```
10 SYMBOL AFTER 0
20 INPUT"Welches Zeichen soll geaendert
werden";a
30 MODE 2
40 PRINT CHR$(1);CHR$(a)
50 FOR i=0 TO 7
60 a(i)=PEEK(&C000+i*&800)
70 PRINT "  "BIN$(a(i),8)
80 NEXT
90 FOR i=0 TO 7
100 LOCATE 1,i+2:INPUT a$
110 IF a$="" THEN 130
120 a(i)=VAL("&x"+a$)
130 GOSUB 1000
140 NEXT i
150 END
1000 SYMBOL a,a(0),a(1),a(2),a(3),a(4),a
(5),a(6),a(7)
1010 PRINT CHR$(30);CHR$(1);CHR$(a)
1020 FOR j=0 TO 7
1030 PRINT "  "BIN$(a(j),8);USING"  ###"
;a(j)
1040 NEXT:RETURN
```

Erläuterung: Der Beginn des Bildschirmspeichers liegt nach einem MODE-Befehl bei &c000 (dez. 49152). Die nächste Bildschirmzeile beginnt bei &c000+&800(2048). In MODE 2 entspricht jeder Punkt, der dem Farbwert von INK 1 hat, einem 1-Bit und jeder Punkt mit dem Farbwert von INK 0 einem 0-Bit. Das mache ich mir zunutze, indem ich nach jedem INPUT die jeweiligen BIN$ und für eine spätere Programmzeile die darauf folgenden Dezimalwerte ausdrucken lasse. Bei Drücken der ENTER-Taste wird der Inhalt übernommen.

BORDER gA1 ⟨, gA2⟩

Befehl: Färbt den nicht beschreibbaren Rand mit einem Farbwert zwischen 0 und 31. Dieser Farbwert ist unabhängig von dem jeweiligen Modus; es sind also auch in MODE 2 alle Farben zu verwenden.

Farbzahlen größer als 31 erzeugen ein »Improper argument«. Bei Verwendung von gA2 wechselt die Farbe zwischen gA1 und gA 2 in einer Schnelligkeit, die vom SPEED INK-Befehl bestimmt wird.

Verwendungsmöglichkeiten: Neben dem allgemeinen Bildschirmaufbau läßt sich BORDER wunderbar dazu verwenden, um den Benutzer zu warnen, falls er seinen Lautsprecher abgeschaltet hat.

```
5 PRINT"Bitte geben Sie eine Zahl zwisch
en 1 und 4 ein"
10 a$=INKEY$
20 IF a$="" THEN 10
30 IF a$<"1" OR a$>"4" THEN BORDER 26:FO
R zeit=1 TO 10:NEXT:BORDER 0:SOUND 1,200
,10
40 ON VAL(a$) GOSUB 1000,2000,3000,4000
50 GOTO 10
```

CALL gA1, (Liste von Parametern)

Befehl: Ruft ein Maschinenprogramm auf, das bei der Adresse gA1 beginnt. Die Liste von Parametern kann, wenn man die Adresse weiß, dem Maschinenprogramm übergeben werden. Meistens dient CALL dem Aufruf von Routinen im BASIC-Interpreter.

CALL 0 zum Beispiel erzeugt das gleiche Zurücksetzen des Rechners wie das gleichzeitige Drücken von SHIFT+CTRL+ESC.

CAT

Systembefehl: Eine ausführliche Beschreibung findet sich im Handbuch des Computers.

Weitere Verwendung: CAT eignet sich hervorragend dazu, den Motor des Kassettenrecorders einzuschalten, wenn man ein Stück Band löschen will, ohne etwas aufzunehmen.

CHAIN " ⟨Programmname⟩ " ⟨, Znr ⟩

Systembefehl: Lädt das Programm »Programmname« von der Kassette und startet es. Wenn Znr angegeben wurde, startet das Programm bei der Zeilennummer. Dieser Befehl wirkt wie die Kombination von LOAD »Programmname« und RUN Zeilennummer.

Wenn die Zeilennummer im CHAIN-Befehl größer ist als die größte Zeilennummer im nachgeladenen Programm, erhalten Sie die Fehlermeldung »Line does not exist«. Wenn Sie als Zeilennummer eine nicht vorhandene Zeile angeben, auf die noch eine Zeile folgt, erweitert der Interpreter die Fehlermeldung um »in xx«, wobei xx genau den Wert der folgenden Zeilennummer hat.

CHAIN MERGE " ⟨Programmname⟩ " ⟨, Znr ⟩ ⟨,DELETE Zeilenbereich⟩

Systembefehl: Mischt das angegebene Programm von der Kassette zu dem im Speicher. Bei doppelt vorhandenen Zeilen gilt die Zeile von der Kassette. Bevor das Programm bei Znr gestartet wird (im Sinne eines CONT-Befehles), können Sie aus Ihrem Ursprungsprogramm Zeilen löschen. Die Bedingungen finden Sie unter DELETE.

Wenn Sie den Befehlsteil DELETE... benützen, müssen Sie die Startzeilennummer angeben. Das Ersetzen durch ein Komma führt zu »Improper argument«.

CHR$ (gA)

Stringfunktion: Liefert das Zeichen, dessen ASCII-Code dem Wert des ganzzahligen Ausdrucks entspricht. Dieser darf nur Werte zwischen 0 und 255 annehmen.

Verwendungsmöglichkeiten: Diese Funktion wird benützt, um Zeichen zu erzeugen, die nicht über die Tastatur zu erreichen sind oder nicht gedruckt werden sollen.

```
1 ' CHR$(x)
10 MODE 0
20 DIM x(4)
30 FOR i=1 TO 4
40 READ x
50 a$(i)=CHR$(x)
60 NEXT i
70 DATA 248,250,249,251
80 FOR i=1 TO 4
90 LOCATE 10,12
100 PRINT a$(i)
110 FOR j=1 TO 150:NEXT j
120 NEXT i
130 GOTO 80
```

Wichtig bei der Benutzung des Druckers ist die Tatsache, daß Zeichen, die über die Tastenkombinationen CTRL+Buchstabentaste erzeugt werden, auch bei LIST#8 als Steuerzeichen interpretiert werden (siehe auch S. 91).

CINT(nA)

Ganzzahlfunktion: Ergibt unter der Voraussetzung, daß nA zwischen −32768 und +32767 liegt, den mit der ersten Nachkommastelle gerundeten Wert von nA.

Verwendungsmöglichkeiten: Eine gute Verwendung bietet CINT im Zusammenhang mit Zufallszahlen. Benutzt man zum Beispiel die RND-Funktion, um zwischen zwei Werten zu wählen, muß man meistens folgendermaßen programmieren:

```
10 FOR i=1 TO 1000
20 a=RND
30 IF a>0.5 THEN b=b+1
40 NEXT
50 PRINT b
```

Kürzer geht es so:

```
10 'CINT(x)
20 RANDOMIZE TIME
30 FOR i=1 TO 1000
40 a=RND
50 b=b+CINT(a)
60 NEXT
70 PRINT b
```

CLEAR

BEFEHL: Setzt alle Pointer zurück, löscht alle Variablen und zerstört alle Rücksprungadressen. Ein CONT nach einem CLEAR ist nicht möglich!

Verwendungsmöglichkeiten: CLEAR wird oft im Zusammenhang mit einem erneuten Programmstart benutzt, wenn man weder die Graphikzeichen noch den Titel erneut durchlaufen möchte, aber alle Variablen auf den Anfangswert setzen will.

```
10 MODE 0
20 a$="WUERFELSPIEL"
30 PLOT 0,0,1
```

```
40 FOR i=1 TO 200 STEP 4
50 MOVE 100,i:TAG:PRINT a$;
60 NEXT
70 TAGOFF
80 MODE 1
90 INPUT"Zahl der Spieler 1-6";a
100 IF a<1 OR a>6 THEN 80
110 DIM na$(a)
120 FOR i=1 TO a
130 PRINT"Spieler Nr."i
140 INPUT"wie heisst Du";na$(i)
150 NEXT i
160 CLS
170 PRINT"Spielregeln:"
180 PRINT"In jeder Runde darf jeder Spie
ler"
190 PRINT"sooft wuerfeln, wie er moechte
."
200 PRINT"Das Spiel geht an den naechste
n,"
210 PRINT"wenn der Spieler weitergibt od
er"
220 PRINT"eine >1< wuerfelt. Bei freiwil
liger"
230 PRINT"Spielabgabe werden die in der
Runde"
240 PRINT"Erreichten Punkte zum Spielerk
onto"
250 PRINT"addiert. Bei >1< werden alle W
uerfe"
260 PRINT"aus dieser Runde gestrichen."
270 PRINT"Sieger des Spieles ist derjeni
ge,"
280 PRINT"der zuerst 100 Punkte erreicht
 hat."
290 PRINT"ENTER=Spielbeginn"
300 WHILE INKEY$<>CHR$(13):WEND
310 DIM w(a)
320 FOR i=1 TO a
330 z=0:sp=0
```

```
340 CLS
350 PRINT na$(i)" ist dran":PRINT"Punkte
stand: "w(i)
360 zu=INT(RND*6+1)
370 z=z+1
380 PRINT"Wurf";z;": eine"zu
390 IF zu=1 THEN PRINT"Der Naechste bitt
e!":WHILE INKEY$="":WEND:GOTO 460
400 sp=sp+zu:in$=""
410 PRINT"<W>=weiter <A>=aufschreiben"
420 WHILE in$<>"A" AND in$<>"W":in$=UPPE
R$(INKEY$):WEND
430 IF in$="A" THEN w(i)=w(i)+sp
440 IF in$="W" THEN GOTO 360
450 IF w(i)>99 THEN PRINT"BRAVO! "na$(i)
" hat 100 Punkte erreicht!":GOTO 480
460 NEXT i
470 GOTO 320
480 PRINT"Noch ein Spiel? <J>/<N>"
490 WHILE in$<>"N" AND in$<>"J":in$=UPPE
R$(INKEY$):WEND
500 IF in$="N" THEN CLS:END
510 in$=""
520 PRINT"Die selben Spieler? <J>/<N>"
530 WHILE in$<>"N" AND in$<>"J":in$=UPPE
R$(INKEY$):WEND
540 IF in$="N" THEN CLEAR:GOTO 90
550 ERASE w:GOTO 310
```

Außerdem benutzt man CLEAR in einem linearen Programmablauf, um Variablen und Tabellen neu definieren zu können und den Speicherplatz für die alten Variablen wieder zur Verfügung zu haben (siehe auch ERASE).

CLG (gA)

Graphikbefehl: Löscht den für Graphik vorgesehenen Bildschirmbereich (siehe ORIGIN) in der durch gA festgelegten Farbe. Auch hier ist die Anzahl der zur Verfügung stehenden Farben durch den Modus festgelegt.

Die Auswirkungen sehen Sie am deutlichsten durch Vergleich der folgenden Programme.

```
10 FOR i=0 TO 2
15 MODE i
20 FOR j=0 TO 15
30 CLG j
40 FOR k=1 TO 300:NEXT
50 NEXT
60 NEXT
```

```
10 MODE 0
20 FOR i=1 TO 320
25 FOR j=1 TO 20:NEXT
30 ORIGIN 320,200,i,640-i,400-CINT(i*200
/320),CINT(i*200/320)
40 CLG (i MOD 15)
50 NEXT
```

Außerdem ist CLG die schnellste Möglichkeit, ein Rechteck zu zeichnen.

CLOSEIN
Systembefehl: Siehe OPENIN.

CLOSEOUT
Systembefehl: Siehe OPENOUT.

CLS (#gA)

Befehl: Löscht das durch gA angegebene Window mit der durch ein vorheriges PAPER-Kommando angegebenen Farbe. gA darf Werte zwischen 0 und 7 annehmen.

```
10 MODE 0
20 FOR i=1 TO 480:PRINT"*";:NEXT
30 FOR i=0 TO 7
40 links=(i MOD 4)*5+1
50 rechts=(i MOD 4)*5+5
60 oben=(i\4)*12+1
70 unten=(i\4)*12+12
80 WINDOW#i,links,rechts,oben,unten
```

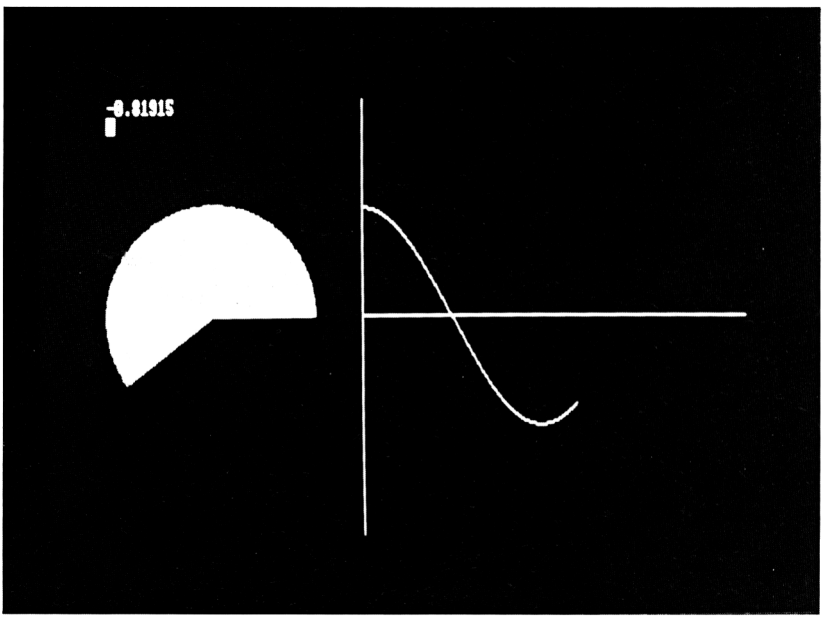
```
90 PAPER#i,i+2
100 CLS #i
110 FOR j=1 TO 400:NEXT
120 NEXT
```

CONT

Systembefehl: Setzt ein durch ⟨ESC⟩⟨ESC⟩, END oder STOP unterbrochenes Programm fort – es sei denn, das Programm wurde geändert oder CLEAR im Direktmodus eingegeben.

COS (GkA)

Gleitkommafunktion: Liefert den Wert des COS zu einem im Bogenmaß (oder – wenn nach DEG – in Grad) angegebenen Winkel. Dieser Wert bewegt sich immer zwischen −1 und +1. Der Cosinus eines Winkels ergibt sich aus dem Verhältnis Ankathete zu Hypothenuse.



```
10 MODE 2
20 ORIGIN 102,200
30 PLOT 150,-200,1
40 DRAWR 0,400,1
50 PLOT 150,0
60 DRAWR 400,0
70 DEG
80 FOR i=0 TO 360
90 PLOT 0,0,1
100 DRAW 100*COS(i),100*SIN(i),1
110 DRAW 100*COS(i),0,1
120 DRAW 0,0,1
130 PLOT 102*COS(i),102*SIN(i),1
140 PLOT i+150,100*COS(i),1
150 FOR j=1 TO 10
160 NEXT j
170 PLOT 0,0,0
180 DRAW 100*COS(i),100*SIN(i),0
190 DRAW 100*COS(i),0,0
200 DRAW 0,0,0
210 PRINT CHR$(30);USING"+#.#####";COS(i)
220 NEXT
```

CREAL (nA)

Gleitkommafunktion: Liefert den Wert von nA.

Verwendungsmöglichkeit: Keine, da eine Integervariable von Natur keine Nachkommastellen besitzt und eine Gleitkommavariable von sich aus in der Lage ist, Nachkommastellen aufzuheben.

DATA Konstante ⟨,Konstante,..⟩

Befehl: Speichert die nachfolgenden Konstanten für eine spätere READ-Anweisung.

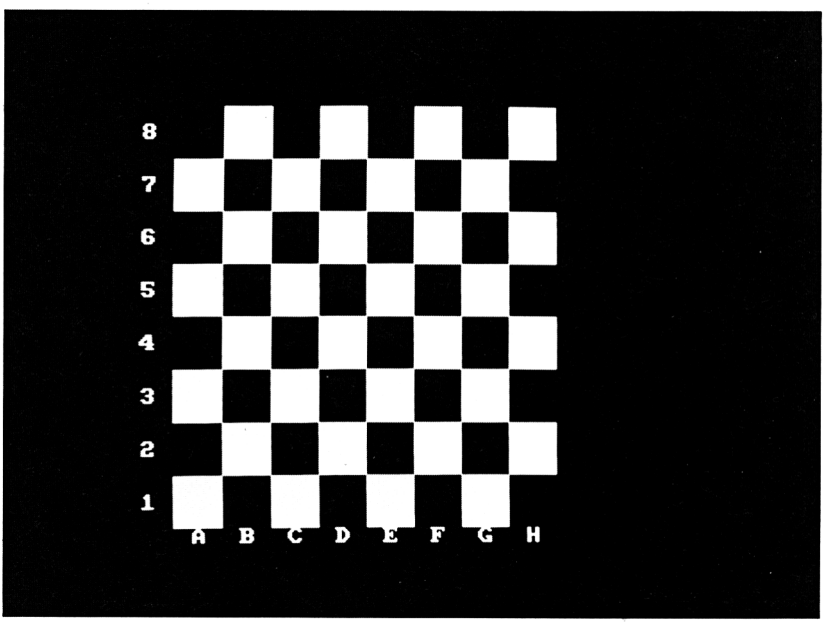
Die einzelnen Konstanten werden durch Kommata getrennt. Stringkonstanten müssen nicht in Anführungszeichen eingeschlossen werden, es sei denn, sie enthielten Kommata oder Doppelpunkte. Eine Konstante darf in einer DATA-Zeile nie Anführungszeichen enthalten, da sie immer als Anfang oder Ende eines Strings interpretiert werden würden.

Wenn in einer DATA-Zeile andere Befehle stehen, werden sie beim letzten READ als CHR$ interpretiert.

```
10 DATA Ernst, Dieter, "Ergebnis: 0", 12
20 DATA "Meier, Mueller",Koeln:PRINT
30 READ a$,b$,d$,e,f$,g$
40 PRINT a$,b$,d$,e,f$,g$
```

Verwendungsmöglichkeiten: DATA ist eine sehr platzsparende Möglichkeit, um Informationen im Programm selbst aufzuheben. Will man zum Beispiel eine lange Zeichenkette aus Graphikzeichen aufbauen, erspart man sich viel Tipparbeit, wenn man die dazugehörigen ASCII-Codes in einer DATA-Zeile aufhebt und den String in einer Schleife zusammenbaut; weitere Beispiele bei READ und RESTORE.

```
10 DATA 32,32,32,143,143,143,32,32,32,1
43,143,143,32,32,32,143,143,143,32,32,32
,143,143,143
20 DATA 143,143,143,32,32,32,143,143,143
,32,32,32,143,143,143,32,32,32,143,143,1
43,32,32,32
```



```
30 FOR i=1 TO 24
40 READ a:a$=a$+CHR$(a)
50 NEXT i
60 FOR i=1 TO 24
70 READ a:b$=b$+CHR$(a)
80 NEXT i
85 MODE 1
90 FOR i=8 TO 2 STEP -2
95 PRINT"    ";a$
100 PRINT i;a$
105 PRINT"    ";a$
106 PRINT"    ";b$
110 PRINT i-1;b$
115 PRINT"    ";b$
120 NEXT
130 PRINT"     A  B  C  D  E  F  G  H";
140 GOTO 140
```

DEF FN Funktionsname ⟨ (Parameterliste) ⟩=Ausdruck

Befehl: Definiert eine benutzereigene Funktion: Funktionsname kann jeder gültige Variablenname sein. In der Parameterliste können als sogenannte Dummyvariablen String- und numerische Variablen gemischt sein. »Ausdruck« muß syntaktisch richtig formuliert werden. Überprüft wird die Syntax des DEF-Befehls beim ersten Aufruf. In der Programmzeile erscheint auch die Fehlermeldung. Die Funktion wird aufgerufen mit FN.

Verwendungsmöglichkeiten: Eine der meist benützten Funktionen dient der Erzeugung von Zufallszahlen innerhalb eines bestimmten Bereiches:

```
10 DEF FNa(x,y)=INT(RND*(1+ABS(x-y)))+MI
N(x,y)
20 INPUT"Untere Grenze";u
30 INPUT"Obere Grenze";o
40 FOR i=1 TO 100
50 PRINT FNa(u,o);
60 NEXT i
70 PRINT
80 GOTO 20
```

Im Gegensatz zu vielen anderen Homecomputern ist der Schneider auch in der Lage, Stringfunktionen zu definieren. Als Beispiel eine Funktion, die eine Laufschrift von bis zu 252 Zeichen an beliebiger Stelle auf dem Monitor darstellt.

```
10 DEF FNa$(x$,x,y,z,l)=CHR$(31)+CHR$(x)
+CHR$(y)+MID$(x$+" ",z,l)
20 b$="Der Schneider CPC 464 bietet mehr
 Moeglichkeiten, als sich der Beginner t
raeumen laesst"
30 horiz=1
40 vert=12
50 lang=30
60 FOR beginn=1 TO LEN(b$)+1
70 PRINT FN a$(b$,horiz,vert,beginn,lang
)
80 PRINT FN a$(b$,10,14,beginn,20)
90 PRINT FN a$(b$,30,16,beginn,30)
100 NEXT
```

Oder eine Funktion, die einen Text links von der ersten Leerstelle ermittelt.

```
10 DEF FNa$(x$)=LEFT$(x$,INSTR(x$+" ",
" "))
20 a$="Computerschule VRIL"
30 PRINT FNa$(a$)
```

DEFINT buchstabe(nreihe)
DEFSTR buchstabe(nreihe)
DEFREAL buchstabe(nreihe)

Befehl: Diese Befehle sind alle gleich aufgebaut. Sie definieren alle Variablen, die mit einem der Buchstaben, die hinter DEF... stehen, beginnen, als diesem Typ zugehörig. Der Einsatz dieser Befehle verlangt vom Programmierer einen genauen Plan, wie die Variablennamen zu verteilen sind. Besonders DEFINT wird häufig benutzt, um Laufvariablen und Indices als Integervariablen zu definieren. Der Vorteil liegt im geringeren Platzbedarf im Programm und in der schnelleren Verarbeitung bei Programmablauf.

Die Gefahr bei Integerlaufvariablen liegt in der Möglichkeit von Endlosschleifen. Versuchen Sie folgendes Programm:

```
5 DEFINT i
10 s=2*PI/360
20 FOR i=0 TO 2*PI STEP s
```

```
30 si=SIN(i)
40 NEXT
50 PRINT si
```

Wenn Sie die Geduld verloren haben, schauen Sie sich mal den letzten Wert von i an. Durch die Verwendung einer Integervariablen i und einer Schrittweite kleiner als 0.5 wird i nie verändert.

DEG

Befehl: Schaltet von der standardmäßigen Winkelangabe in Bogenmaß (360 Grad = 2×pi) auf Winkelangabe in Grad um. Diese Schreibweise ist vor allen Dingen in der Graphik angenehm, weil man so mit Integervariablen für die Winkel arbeiten kann. Davon abgesehen ist vielen Leuten – mich eingeschlossen – die Gradeinteilung geläufiger.

Zur Demonstration ein kleines Programm, das einen rotierenden Sekundenzeiger zeigt. Gestellt wird die Uhr mit der Variablen z. Auf meinem Schneider lief sie mit z = 15 genau.

```
10 CLS
20 ORIGIN 320,200
30 r%=200
40 INPUT z
50 FOR i=0 TO 2*PI STEP PI/6
60 PLOT 0,0,1:DRAW r%*SIN(i),r%*COS(i)
70 FOR j=1 TO z:NEXT
80 NEXT i
90 DEG
100 FOR i=0 TO 360 STEP 30
110 PLOT 0,0,0:DRAW r%*SIN(i),r%*COS(i
)
120 FOR j=1 TO z:NEXT
130 NEXT i
140 RAD:GOTO 50
```

DELETE Zeilenbereich

Systembefehl: Löscht Programmzeilen innerhalb des Zeilenbereiches.

Bei der Benutzung dieses Befehls ist Vorsicht geboten. DELETE mit einem folgenden Minuszeichen (–) löscht das gesamte Programm. Die Fehlermeldung »Improper argu-

ment« erhalten Sie, wenn im angegebenen Zeilenbereich keine Programmzeilen zu finden sind. Der Einsatz von DELETE im Programm führt zum Programmabbruch.

DI

Befehl: Setzt die Priorität eines Programmteiles in Zusammenhang mit den Befehlen EVERY und AFTER durch. Während ein Programmteil mit DI vor Unterbrechungen geschützt ist, zählen die Zeitgeber weiter. Sobald mit EI der Interrupt wieder freigegeben wird, werden soviel »liegengebliebene« Unterprogramme zu EVERY-Befehlen wie möglich ausgeführt. Als kleines Beispiel eine abschaltbare Uhr. INPUT und Kassettenbefehle wirken wie DI.

```
1 CLS
5 WINDOW#1,1,40,2,25
10 EVERY 50 GOSUB 1000
30 IF INKEY$<>"" THEN schalter=NOT schalter
40 PRINT#1,"Schneider ";
50 IF schalter THEN DI:PRINT CHR$(30);"                "
55 IF NOT schalter THEN EI
60 GOTO 30
1000 x=POS(#0):y=VPOS(#0)
1010 a=a+1:IF a=60 THEN b=b+1:a=0
1020 IF b=60 THEN c=c+1:b=0
1030 PRINT CHR$(30);USING CHR$(24)+"
h###m##s##"+CHR$(24);c,b,a
1040 LOCATE x,y
1050 RETURN
```

DIM variable (gA1(, gA2,...))

Befehl: Legt im Speicher eine Tabelle für indizierte Variablen an. Der Speicherbedarf richtet sich nach der Anzahl der Elemente, dem Variablentyp und – bei Strings – nach dem Inhalt der Tabellenplätze. Wie groß die maximale Zahl der Tabellenplätze in einem Programm wirklich sein kann, läßt sich nur durch Versuche ermitteln, wenn das restliche Programm komplett fertig ist.

```
10 DIM a(b)
15 PRINT b;
```

```
20 ERASE a
30 b=b+10
40 GOTO 10
```

Glücklicherweise bietet das Schneider-Basic dazu den komfortablen Befehl ERASE. Grundsätzlich kann man davon ausgehen, daß eine eindimensionale Tabelle, also eine Tabelle mit nur einem Index, weniger Platz wegnimmt, als eine mehrdimensionale mit der gleichen Anzahl.

```
10 PRINT FRE(0)
20 DIM a(1000)
30 PRINT FRE(0)
40 CLEAR
50 DIM a(100,10)
60 PRINT FRE(0)
70 CLEAR
80 DIM a(10,10,10)
90 PRINT FRE(0)
```

Problematisch wird die Sache mit dem Speicherplatz in dem Augenblick, wenn Sie Tabellen für Strings definieren. Die DIM-Anweisung reserviert nämlich nur Platz für eine entsprechende Anzahl von Leerstrings. Mancher Programmierer hat schon eine böse Überraschung erlebt, als er nach DIM den freien Speicherplatz abfragte und in der Annahme, daß jetzt alles geregelt sei, begann, seine Tabelle zu füllen. Das führte zum ersten »Memory full«-Fehler. Also hat er sich eine Schleife aufgebaut, die folgendermaßen aussah:

```
10 platz=200

20 DIM a$(platz)

30 FOR i=1 TO platz

40 a$(i)="Irgendein String in der Länge des

   maximal auftretenden Tabelleninhalts."

50 NEXT i

60 PRINT FRE(0)
```

Leider kann auch in diesem Fall eine »Memory full«-Meldung auftreten. Versuchen Sie das Programm und fügen Sie eine Zeile 45 ein:

```
45 ? FRE(0)
```

Sie werden voller Erstaunen feststellen, daß sich der freie Speicherplatz nicht ändert. Falls Sie ein Programm in dieser Form testen wollen, müssen Sie die Zeile 40 ändern in:

```
40 a$(i)="Irgendein String in der Länge des
   maximal auftretenden Tabelleninhalts."+""
```

Nur unter diesen Voraussetzungen gibt Ihnen der Schneider die tatsächlich freien Plätze aus.

Verwendungsmöglichkeiten: Meistens wird die Tabellenverarbeitung für gleichartige Informationen benutzt, die zählbar sind und nach irgendwelchen Kriterien geordnet werden können.

```
10 DIM na$(100,6)
20 DIM fe$(6)
30 DATA Vorname,Nachname,Strasse,PLZ,Ort
,Telefon
40 FOR i=1 TO 6
50 READ fe$(i)
60 NEXT i
70 MODE 1
80 FOR i=1 TO 100
90 PRINT:PRINT"Adresse Nr."i".(ENTER=End
e)":PRINT
100 FOR j=1 TO 6
110 PRINT USING"\       \:";fe$(j);
120 INPUT na$(i,j)
130 IF na$(i,1)="" THEN GOTO 160
140 NEXT j
150 NEXT i
160 CLS
170 FOR k=1 TO i-1
180 PRINT"Adresse Nr."k
190 FOR j=1 TO 6
200 PRINT USING"\       \:";fe$(j);
210 PRINT na$(k,j)
220 NEXT j
```

```
230 PRINT:PRINT"Enter=Weiter":PRINT
240 IF INKEY$<> CHR$(13) THEN GOTO 240
250 NEXT k
```

DRAW gA1, gA2 ⟨, gA3⟩

Graphikbefehl: Zeichnet eine Linie von der aktuellen Graphik-Cursorposition zu den durch gA1 und gA2 definierten Koordinaten in der durch gA3 angegebenen Farbe. Wenn der letzte Parameter nicht benützt wird, wird in der Farbe gezeichnet, die zuletzt benutzt wurde.

gA1 und gA2 dürfen Werte von −32768 bis +32768 annehmen. Größere Werte werden durch »Overflow« abgewiesen. gA3 darf maximal 15 groß sein; ab 16 erfolgt ein »Improper argument«.

Benutzt man Koordinaten, die außerhalb des Bildschirmbereiches liegen, den man für Graphik definiert hat, so wird bis zur Bildschirmbereichsgrenze sichtbar gezeichnet, der Graphik-Cursor bewegt sich allerdings auf eine Position außerhalb des Bildschirms. Die Verarbeitungsgeschwindigkeit wird dadurch deutlich verringert.

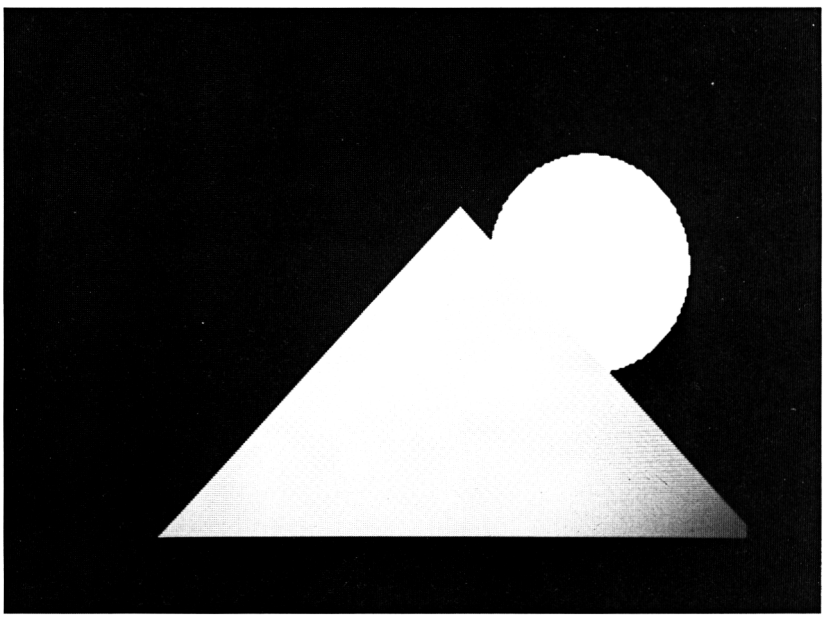
```
10 MODE 2
20 FOR i=0 TO 10000 STEP 50
30 LOCATE 1,1
40 PRINT i
50 MOVE 0,0
60 DRAW i,i,1
70 NEXT i
```

Verwendungsmöglichkeiten: In Zusammenhang mit MOVE wird DRAW häufig dazu benutzt, regelmäßige Flächen wie Kreise und Dreiecke auszumalen. Durch Mischung verschiedener Farben lassen sich dabei neue Farben malen.

```
10 MODE 1
20 PRINT CHR$(23);CHR$(0)
30 ORIGIN 480,250
40 DEG
50 FOR i=0 TO 180
60 MOVE 100*SIN(360-i),100*COS(360-i)
70 DRAW 100*SIN(i),100*COS(i),1
80 NEXT
90 ORIGIN 150,100
100 PRINT CHR$(23);CHR$(1);
110 FOR i=-100 TO 200 STEP 2
```

```
120 MOVE 400-i,i
130 DRAW i,i,3
140 NEXT
```

Informationen zu den in Zeile 20 und 100 verwendeten Steuerzeichen finden Sie ab Seite 91



DRAWR gA1, gA2⟨, gA3⟩

Graphikbefehl: Zeichnet von der aktuellen Graphik-Cursorposition zu einer Position, deren Koordinaten sich errechnen aus der aktuellen x-Koordinate + gA1 und der aktuellen y-Koordinate +gA2. Für die drei gA gelten die gleichen Beschränkungen wie bei DRAW.

Verwendungsmöglichkeiten: Will man eine Figur auf dem Bildschirm zeichnen, deren Position sich ständig ändert, so kann man mit DRAWR diese Figur so zeichnen, als ob der jeweilige Endpunkt einer Linie die Koordinaten 0,0 für die nächste Linie hätte.

Als Beispiel ein kleines Schiff. Beginnen wir rechts unten an den Koordinaten 0,0. Eine schräge Linie nach oben rechts zu 12,12. Diese Koordinaten nennen wir jetzt 0,0. Ein Strich nach links zu −20,0; einer nach oben usw. usw. . . .

```
10 MODE 1
20 FOR j=0 TO 400 STEP 30
30 FOR i=640 TO 48 STEP-4
40 MOVE i,j
50 GOSUB 120
60 GOSUB 120
70 NEXT i
80 PRINT CHR$(23);CHR$(0)
90 GOSUB 130
100 NEXT j
110 END
120 PRINT CHR$(30);CHR$(23);CHR$(1);
130 DRAWR 12,12,1
140 DRAWR -20,0
150 DRAWR 0,8
160 DRAWR -8,0
170 DRAWR 0,-8
180 DRAWR -20,0
190 DRAWR 12,-12
200 DRAWR 24,0
210 RETURN
```

EDIT Znr

Systembefehl: Listet die durch Znr angegebene Programmzeile auf und setzt den Cursor auf die erste Stelle dieser Zeile. Die Übernahme der Zeile durch den Copy-Cursor ist nicht notwendig. Verwendet man EDIT im Programm, so wird das Programm an dieser Stelle unterbrochen.

Verwendungsmöglichkeit: Verändern bestehender Programmzeilen, Kopieren von Programmzeilen durch Änderung der Zeilennummer.

EI

Befehl: Gibt einen durch DI geschützten Zeilenbereich für Unterbrechungen frei. Da der Schneider sich »merkt«, wie oft ein Unterprogramm wegen DI nicht angesprungen wurde, wird er in dem Augenblick, wenn EI eingegeben wird, alle aufgelaufenen Arbeiten erledigen und dann erst im Programm weitermachen. Um das zu verhindern, muß man nach DI den entsprechenden Zeitgeber mit REMAIN (gA) ausschalten. Das führt

allerdings dazu, daß der EVERY-Befehl wiederholt werden muß. Geben Sie die folgenden Programmzeilen in der abgedruckten Reihenfolge ein und testen Sie das Programm nach jeder Zeile mit RUN. Beginnen Sie mit den Zeilen 10, 40, 80, 90 und 100:

```
10 EVERY 30 GOSUB 90

40 PRINT"ERSTE SCHLEIFE":FOR I=1 TO 5000:NEXT I

80 END

90 A=A+1:PRINT A

100 RETURN

RUN

20 DI

RUN

50 EI

RUN

30 B=REMAIN(0)

RUN

70 PRINT"ZWEITE SCHLEIFE":FOR I=1 TO 5000:NEXT I

RUN

60 EVERY 30 GOSUB 90

RUN
```

Befehl: Das laufende Programm wird beendet; es kann allerdings mit CONT an genau dieser Stelle wieder gestartet werden. Im Gegensatz zu STOP gibt END nicht die Programmzeile aus, in der das Programm unterbrochen wurde.

Verwendungsmöglichkeiten: In Programmen, deren Steuerung ganz am Anfang steht und in denen die einzelnen Programmteile als Unterprogramme am physischen Ende des Programmes stehen, wird END unter Umständen als fünfte Programmzeile stehen.

```
10 GOSUB 1000
20 GOSUB 2000
30 ON wahl GOSUB 3000,4000,5000,6000
40 IF ende=1 THEN END
50 GOTO 20
1000 'Vorroutinen
1010 RETURN
2000 'Hauptmenue
2010 CLS
2020 PRINT"<1> = Laden"
2030 PRINT"<2> = Speichern"
2040 PRINT"<3> = Listen
2050 PRINT"<4> = Ende"
2060 INPUT"Ihre Wahl <1> bis <4>";wahl$
2070 wahl=VAL(wahl$)
2080 IF wahl<1 OR wahl>4 THEN GOTO 2010
2090 RETURN
3000 'Laden
3010 RETURN
4000 'Speichern
4010 RETURN
5000 'Listen
5010 RETURN
6000 'Ende
6010 CLS
6020 INPUT"Sind sie wirklich sicher <j>/
<n>";wahl$
6030 IF wahl$<>"j" AND wahl$<>"n" THEN G
OTO 6010
6040 IF wahl$="j" THEN ende = 1
6050 RETURN
```

ENT

Soundbefehl: Siehe S. 81

ENV

Soundbefehl: Siehe S. 77

EOF

Systemvariable: Diese Variable enthält normalerweise den Wert −1. Nur wenn eine Kassettendatei geladen wird, springt sie auf 0. Sobald das Ende der Datei erreicht ist, hat EOF wieder den Wert −1. Diesen Wert kann man abfragen, um Fehler wegen unerwarteten Dateiendes zu vermeiden.

Übrigens: EOF ist die Abkürzung für End Of File (Ende der Datei).

Mit folgendem Programm erstellen wir eine Testdatei:

```
10 OPENOUT"test"
20 FOR i=1 TO 100
30 PRINT#9,i
40 PRINT i;EOF;
50 NEXT
60 CLOSEOUT
70 PRINT EOF
80 END
```

Danach wollen wir sie wieder einlesen:

```
10 OPENIN"TEST"
20 PRINT EOF
30 INPUT#9,a
40 PRINT EOF;a;
50 GOTO 30
```

Das Programm bricht mit einer Fehlermeldung: »EOF met in 30« ab. Ändern Sie jetzt die Zeile 50 um:

```
50 IF EOF=0 THEN GOTO 30
```

oder

```
50 IF NOT EOF THEN GOTO 30
```

Danach wird nur solange von der Kassette gelesen, wie Daten vorhanden sind.

ERASE Liste von Variablennamen

Befehl: Löscht Tabellen (!) aus dem Speicher. Im Gegensatz zum Handbuch können keine einzelnen Variablen gelöscht werden!

```
10 a$="Schneider"
20 PRINT a$
30 ERASE a$
40 PRINT a$
```

Das Programm bricht ab mit »Improper Argument in 30«. Und dann versuchen Sie mal folgende Version:

```
10 a$(1)="Schneider"
20 PRINT a$(1)
30 ERASE a$
40 PRINT a$(1)
```

Verwendungsmöglichkeiten: Neben der Möglichkeit, mit ERASE Speicherplatz zu sparen, bietet dieser Befehl einen bequemen Weg, alle Werte einer Tabelle auf Null bzw. einen Leerstring zu setzen. Man löscht die entsprechende Tabelle und dimensioniert sie neu. Eine Anwendung finden Sie bei CLEAR (S. 122).

ERL

Systemvariable: Enthält die Zeilennummer, in der der letzte Fehler auftrat. Fehler im Direktmodus erzeugen die Fehlerzeile 0. Lesefehler bei Kassettenoperationen erzeugen unsinnige Fehlerzeilennummern.

ERR

Systemvariable: Enthält die Nummer des letzten Fehlers. Schreib/Lesefehler werden nicht durch ERR behandelt. Der Programmierer kann ERR durch ERROR füllen; Beispielprogramme siehe ON ERROR GOTO

ERROR gA

Befehl: Erzeugt einen Fehler, der bestimmt durch gA wird. Wenn zu diesem Fehler keine Fehlerroutine programmiert wurde, bricht das Programm mit der entsprechenden Fehlermeldung ab.

Verwendungsmöglichkeit: Da die von Basic benützten Fehler nur von 0 bis 30 numeriert sind, kann man durch selbst definierte Fehler alle möglichen Falscheingaben abfangen, indem man Fehlernummern definiert.

```
10 ON ERROR GOTO 90
20 INPUT"bitte eine Zahl von 1 bis 10";a
$
30 IF a$="" THEN ERROR 31
40 FOR i=1 TO LEN(a$):IF MID$(a$,i,1)>"9
" OR MID$(a$,i,1)<"0" THEN ERROR 34
50 NEXT
60 IF VAL(a$)<1 OR VAL(a$)>10 THEN ERROR
 32
70 IF VAL(a$)<>INT(VAL(a$)) THEN ERROR 3
3
80 GOTO 20
90 IF ERR=32 THEN PRINT"Zahl ausserhalb
des gueltigen Bereiches!":RESUME 20
100 IF ERR=33 THEN PRINT"Bitte nur ganze
 Zahlen":RESUME 20
110 IF ERR=31 THEN PRINT"Ohne Eingabe ke
ine Ausgabe!":RESUME 20
120 IF ERR=34 THEN PRINT"Keine Buchstabe
n!":RESUME 20
130 PRINT ERR,ERL
140 LIST
```

EVERY gA1 ⟨,gA2⟩ **GOSUB** Znr

Befehl: Springt in regelmäßigen Abständen in das Unterprogramm, das ab Znr beginnt. gA1 gibt dabei die Abstände, gemessen in 60stel Sekunden, an. Negative Zahlen bei gA1 erzeugen »Improper argument«, Zahlen größer als 32767 »Overflow«. gA2 gibt den Zeitgeber an, der bei einem bestimmten Wert den Sprung nach Znr verursachen soll. Dabei gilt: Je höher die Nummer des Zeitgebers ist, desto größere Priorität hat er.

Verwendungsmöglichkeiten: Alle Aufgaben, die in regelmäßigen Abständen erledigt werden müssen, können auf diese Art gesteuert werden. Nimmt man die Zeitabstände allerdings zu kurz, so wird das Hauptprogramm nie erreicht, weil der Schneider versucht, liegengelassene Arbeiten aus seinen Zeitgeberprogrammen aufzuarbeiten. Das beste Beispiel für eine regelmäßige Arbeit ist eine Uhr.

```
10 INPUT"Stunde";st
20 INPUT"Minuten";mn
30 INPUT"Sekunden";sec
40 EVERY 60,1 GOSUB 1000
50 EVERY 3600,2 GOSUB 2000
60 EVERY 21600,3 GOSUB 3000
70 WINDOW #1,1,12,1,1
80 WINDOW 1,80,2,24
90 PRINT"Schneider ";
100 GOTO 90
1000 sec=sec+1:IF sec=60 THEN sec=0
1010 LOCATE#1,1,1:PRINT#1,USING"## ## ##
";st,mn,sec;
1020 RETURN
2000 mn=mn+1:IF mn=60 THEN mn=0
2010 RETURN
3000 st=st+1:IF st=24 THEN st=0
3010 RETURN
```

Die CLEAR-Anweisung löscht alle EVERY-Anweisungen. Ebenso werden durch den Gebrauch der REMAIN-Variablen bestimmte EVERY-Befehle außer Kraft gesetzt; genauere Informationen finden Sie dort (S. 196).

EXP (nA)

Gleitkommafunktion: Der natürliche Logarithmus von 1 potenziert mit nA. Folgende Programme liefern die gleiche Zahl bei gleicher Eingabe:

```
10 a=2.7182818281828

20 INPUT"Eine Zahl zwischen -89 und +88";z

30 PRINT "e hoch";z;"=";a^z
```

```
10 INPUT"Eine Zahl zwischen -89 und +88";z

20 PRINT "e hoch";z;"=";EXP(z)
```

Die Beschränkung in der Eingabe ist notwendig, um keinen »Overflow« zu erhalten; ein solcher Fehler tritt ab dem Argument 88.0296919 auf.

Verwendungsmöglichkeiten: Diese Funktion wird bei der Berechnung des natürlichen Wachstums benötigt. Ein Kapital erhält seinen Zinszuschlag nur einmal pro Jahr, während eine Pflanze ununterbrochen wächst. Wollte man seine Spareinlagen so verzinsen lassen, wären alle Banken blitzartig pleite; das würde nämlich einen sofortigen Zinszuschlag bedeuten.

FIX (nA)

Gleitkommafunktion: Liefert den ganzzahligen Anteil einer Zahl. Im Gegensatz zu INT, das immer die nächstkleinere Ganzzahl liefert, und CINT, das rundet, werden bei dieser Funktion nur die Nachkommastellen abgeschnitten.

Verwendungsmöglichkeiten: Um bei Divisionen im negativen Bereich auch dann den korrekten ganzzahligen Anteil zu erhalten, wenn die Modula-Operatoren nicht einsetzbar sind, weil die Argumente zu groß sind, muß man die Fix-Funktion benützen:

```
10 INPUT"Zaehler";z
20 INPUT"Nenner";n
30 PRINT z;" geteilt durch";n;"ergibt "F
IX(z/n);" Rest "n*(z/n-FIX(z/n))
40 PRINT z;" geteilt durch";n;"ergibt "I
NT(z/n);" Rest "n*(z/n-INT(z/n))
```

FOR numerische var = nA1 **TO** nA2 ⟨**STEP** nA3⟩
NEXT ⟨var⟩

nA1 = Anfangswert
nA2 = Endwert
nA3 = Schrittweite

Befehl: Setzt bei Beginn der Schleife, die durch die Befehle FOR und NEXT gebildet wird, die Variable »var« auf den Anfangswert, läuft danach alle Befehle bis zum NEXT durch, dann wird die Schrittweise auf »var« aufaddiert und zum FOR-Befehl gesprungen. Dort wird »var« mit dem Endwert verglichen. Sobald »var« größer (bzw. kleiner bei negativer Schrittweite) als der Endwert ist, wird die nächste Anweisung nach NEXT ausgeführt.

Im Gegensatz zu vielen anderen BASIC-Versionen werden beim Schneider unlogische Schleifen nicht durchlaufen. Zur Verdeutlichung der Logik hilft folgendes kleine Programm:

```
10 INPUT"Anfangswert ";na1
20 INPUT"Endwert     ";na2
30 INPUT"Schrittweite";na3
40 FOR i=na1 TO na2 STEP na3
50 a=a+1
60 PRINT"Der Wert von 'i' beim "a". Durc
hlauf betraegt: "i
70 NEXT i
80 PRINT"Endlich fertig"
90 END
```

(Für Fachleute: der Schneider benutzt bei der FOR...NEXT...Schleife eine kopfgesteuerte Schleife. Sehr praktisch!)

Der Schneider ist sehr gut in der Lage, wackelige Konstruktionen in Schleifen zu erkennen. Beim Aufbau einer Schleife untersucht er zuerst die Konstruktion; unlogischer Aufbau wird sofort mit einer Fehlermeldung quittiert, ohne daß die Schleife auch nur einmal durchlaufen wird. Da der Schneider durch den Aufbau genau weiß, welches NEXT zu welchem FOR gehört, ist die Angabe der Variablen nach NEXT nur noch für den Programmierer von Bedeutung. Mir ist es nicht gelungen, eine fehlerhafte Konstruktion zu erreichen. Bei meinen Versuchen, den Schneider auf's Glatteis zu führen, ist mir noch etwas aufgefallen: Schleifen, die ausschließlich mit Ganzzahlen arbeiten, sollten auch Integervariablen verwenden. Vergleichen Sie den Zeitbedarf bei den beiden folgenden Programmen:

```
10 c=TIME
20 FOR i=1 TO 1000
30 NEXT
40 PRINT TIME - c
```

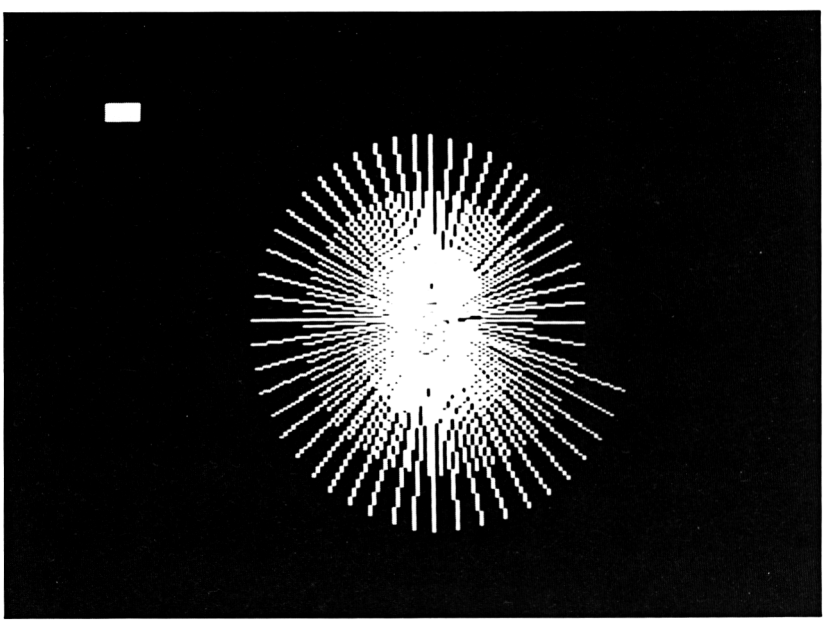
und

```
10 c=TIME
20 FOR i%=1 TO 1000
30 NEXT
40 PRINT TIME - c
```

Das erste endete bei mir mit 327, das zweite mit 176. Rechnet man die Zeitersparnis auf größere Programme um, so erhält man bei Sortierroutinen sehr schnell Minutenbeträge – vielleicht eine Anregung zum sinnvollen Gebrauch der Variablentypen.

Verwendungsmöglichkeiten: Sämtliche Arbeiten, die wiederholt werden müssen und deren Anzahl von Wiederholungen vorher feststeht. Es gibt nur wenige Programme, die ohne FOR..NEXT.. Schleifen auskommen. Als ein Beispiel von unendlich vielen hier ein kleines Graphikprogramm:

```
10 ON ERROR GOTO 140
20 DEG
30 MODE 0
40 ORIGIN 320,200
50 b=1
60 FOR i%=0 TO 2000 STEP 7
70 a=a+1
80 IF a/52=a\52 THEN b=b+1
90 MOVE 0,0
100 DRAW a*COS(i%),a*SIN(i%),b
110 NEXT
120 b=b+1
130 GOTO 60
140 a=0
150 b=0
160 RESUME
```



FRE (Ausdruck)

Ganzzahlfunktion:

a. FRE (nA) gibt die Anzahl der nicht benützten Speicherplätze zurück.
b. FRE (sA) führt erst eine Garbage Collection durch, bevor die Anzahl der freien Speicherplätze zurückgegeben wird.

Zum Verständnis der zweiten Möglichkeit eine Demonstration:

```
20 MODE 2

30 FOR i=1 TO 170

40 a$=STRING$(255,"*")

50 PRINT i,fre(0),

60 NEXT i
```

Der Schneider zeigt Ihnen einen ständig schwindenden freien Speicherplatz an, obwohl Sie immer dieselbe Stringvariable benützen. Ihr Programm endet mit der Zahl 98. Der Computer legt nämlich bei jeder Zuweisung auf eine Stringvariable diese neu an. Erst wenn ihm der Kragen zu eng wird, d. h. der Speicherplatz rar wird, wird er den nicht mehr benötigten Speicherplatz als frei markieren. Diesen Vorgang nennt man »Garbage Collection«. Mit FRE (sA) wird diese Sammlung vom Programmierer veranlaßt. Da eine Garbage Collection relativ viel Zeit benötigt, sollte man sie durchführen, wenn Nachrichten an den Benutzer des Programmes ausgegeben werden. Erweitern Sie das obige Programm um die Zeilen 10 und 70:

```
10 c=TIME

70 PRINT TIME-c
```

Wenn Sie das Programm laufen lassen, erhalten Sie als letzte Zahl die benötigte Zeit, gemessen mit der internen Uhr. Ändern Sie jetzt die Zeile 50 in:

```
50 PRINT i,FRE(0),FRE(""),
```

und starten Sie das Programm erneut. Zwei Dinge fallen auf: FRE (0) liefert kleinere Zahlen als FRE ("") – und das Programm braucht wesentlich mehr Zeit.

Verwendungsmöglichkeiten: Wenn Sie mit Tabellen arbeiten, kann eine ungewollte Garbage Collection zu Pausen von mehreren Minuten führen. Führen Sie diese deshalb selbst durch, und warnen Sie den Benutzer rechtzeitig, wenn der Speicherplatz eng wird.

```
10 DIM a$(500)
20 FOR i=1 TO 500
30 fp=FRE("")
40 PRINT"Freier Speicherplatz"fp
50 IF fp<255 THEN PRINT"Datei ist voll!"
:GOTO 80
60 PRINT"Satz Nr."i;
70 INPUT a$(i)
80 NEXT
90 END
```

GOSUB znr

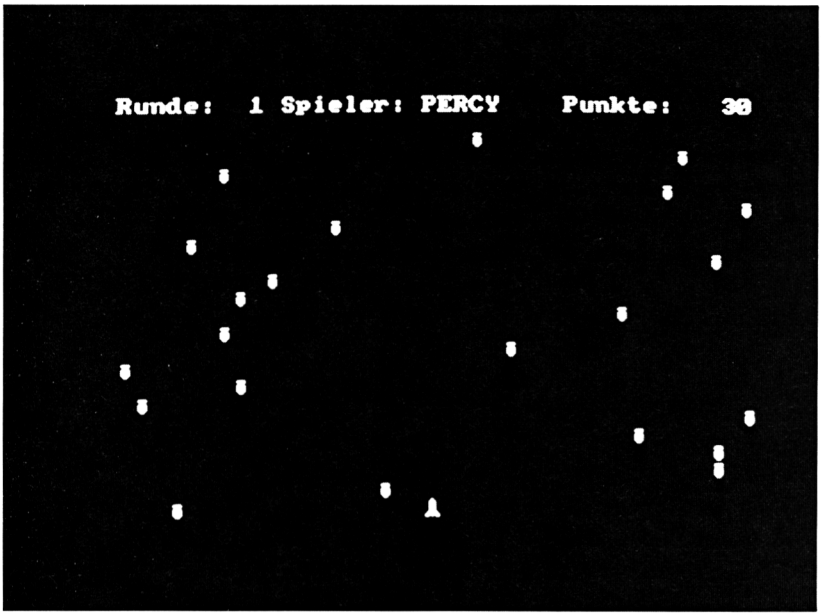
Befehl: Verzweigt zu der angegebenen Programmzeile und fährt dort mit der Arbeit fort, bis ein RETURN angetroffen wird. Danach erfolgt ein Rücksprung auf den Befehl, der dem GOSUB folgt.

Ein Sprung in eine nicht vorhandene Zeilennummer verursacht eine »Line does not exist«-Meldung. Variablen statt Zeilennummern sind nicht erlaubt. Ein rekursiver Aufruf – i. e. ein Aufruf des Unterprogrammes, in dem man sich befindet – führt zu einer »Memory full«-Meldung.

```
10 A=A+1:PRINT A;

100 GOSUB 10
```

Da der Computer bei jedem GOSUB-Befehl sich die Stelle merken muß, an die er bei dem dazugehörigen RETURN zurückspringt, und nur eine bestimmte Anzahl dieser Rücksprungstellen in seinem Speicher Platz hat, kommt es zu dieser Fehlermeldung. Die Zahl der erlaubten Unterprogrammebenen ist auch abhängig von offenen Schleifen.

Verwendungsmöglichkeiten: In fast jedem Programm gibt es Routinen, die mehrmals angesprochen werden (Zulässigkeit von Datumseingaben, Eingaberoutinen, Bildschirmaufbau, usw.). Statt diese Routinen jedesmal neu zu programmieren, kann man sich als Programmierer eine kleine Bibliothek von Programmbausteinen erstellen, die man zu Beginn der Arbeit an einem neuen Programm lädt und deren Teile man mit GOSUB anspringt. Der Befehl CHAIN MERGE bietet beim Schneider bereits die beste Möglichkeit, solche Bibliotheken zusammenzustellen. Ein weiterer Vorteil des GOSUB ist die Möglichkeit, Programme übersichtlicher zu gestalten. Sobald sich der Programmierer im Klaren darüber ist, wie sein Programm aufgebaut sein soll, hat er die halbe Arbeit schon geleistet.




```
10 '*************************************
20 '              Invaders
30 '          A. Werminghoff
40 '*************************************
50 GOSUB 150:'Vorbereitung
60 FOR i=1 TO ru
70 FOR j=1 TO sp
80 GOSUB 330:'Bildschirmaufbau
90 GOSUB 410:'Eingabe
100 NEXT j
110 GOSUB 540:'Rundenende
120 NEXT i
130 GOSUB 650:'Spielende
140 END
150 '*********************************
160 '           Vorbereitung
170 '*********************************
180 DIM na$(6),wert(6)
190 MODE 1
200 INPUT"Wieviel Spieler";sp
```

```
210 IF sp<1 OR sp>6 THEN 190
220 FOR i=1 TO sp
230 PRINT
240 PRINT"Spieler Nr."i","
250 INPUT"wie heisst Du";na$(i)
260 IF na$(i)="" THEN PRINT STRING$(3,11
):GOTO 240
270 NEXT
280 INPUT"Wieviel Runden";ru
290 WINDOW#1,1,40,1,2
300 WINDOW#2,1,40,3,24
310 ORIGIN 6,18
320 RETURN
330 '**********************************
340 '           Bildschirm
350 '**********************************
360 CLS#1
370 CLS#2
380 PRINT#1,USING"Runde: ## Spieler: \
    \ Punkte: ####";i,na$(j),wert(j);
390 GOSUB 880
400 RETURN
410 '**********************************
420 '             Eingabe
430 '**********************************
440 FOR k%=1 TO 200
450 LOCATE#2,1+RND*40,129
460 PRINT#2,CHR$(252);
470 a$=INKEY$
480 IF a$=CHR$(242) AND x>0 THEN x=x-16
490 IF a$=CHR$(243) AND x<640 THEN x=x+1
6
500 IF TEST(x,22)<>0 THEN wert(j)=wert(j
)+10:LOCATE#1,37,1:PRINT#1,USING"####";w
ert(j)
510 LOCATE #2,x/16+1,21:PRINT#2,CHR$(239
);
520 NEXT
530 RETURN
540 '**********************************
```

```
550 '           Rundenende
560 '*********************************
570 CLS
580 PRINT"Stand nach der "i". Runde"
590 PRINT
600 FOR k=1 TO sp
610 PRINT USING"\         \  ###";na$(k);
wert(k)
620 NEXT
630 GOSUB 880
640 RETURN
650 '*********************************
660 '           Spielende
670 '*********************************
680 CLS
690 PRINT"Endauswertung"
700 PRINT
710 format$="#. \         \  ###"
720 FOR i=1 TO sp
730 FOR j=1 TO sp
740 IF wert(j)>so(i) THEN so(i)=wert(j)
750 NEXT j
760 FOR j=1 TO sp
770 IF so(i)=wert(j) THEN wert(j)=0:so$(
i)=na$(j):GOTO 790
780 NEXT j
790 NEXT i
800 FOR i=1 TO sp
810 PRINT USING format$;i,so$(i),so(i)
820 NEXT
830 PRINT"noch ein Spiel <j>/<n>"
840 a$=INKEY$:IF a$="" THEN 840
850 IF a$="j" THEN RUN
860 IF a$="n" THEN RETURN
870 GOTO 840
880 LOCATE 14,12:PRINT"Weiter = RETURN"
890 IF INKEY$<>CHR$(13) THEN 890
900 CLS#2
910 RETURN
```

GOTO znr

Befehl: Verursacht einen Sprung zu der Programmzeile mit der angegebenen znr.

Da der Computer normalerweise die Programmzeilen in der Reihenfolge ihrer Zeilennummern verarbeitet, müßte man Anweisungen, die 10 mal verarbeitet werden sollen, auch 10 mal programmieren. Der GOTO-Befehl erspart uns diese Arbeit. Die größte Gefahr besteht allerdings in der Programmierung von Endlosschleifen. Aus folgendem Programm kommen Sie nur durch Ausschalten wieder heraus:

```
10 KEY DEF 66,0,0,0,0

20 PRINT "Schneider CPC 464";

30 GOTO 10
```

HEX$ (gA1 ⟨, gA2⟩)

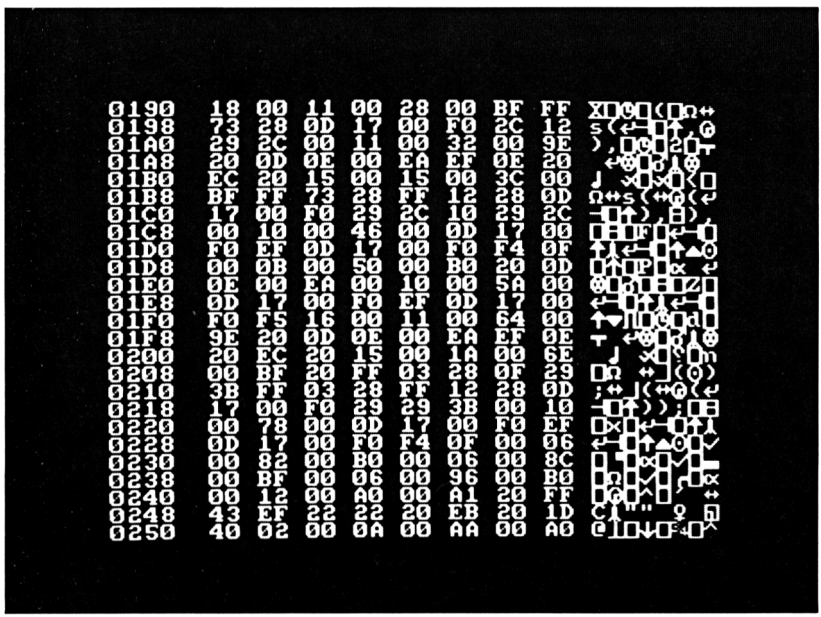
Stringfunktion: Liefert als String in der Länge gA2 die hexadezimale Entsprechung von gA1.

Achtung: Auch die Längenangabe muß in der Klammer stehen. Leider ist im Handbuch die falsche Syntax angegeben. gA1 darf nur im Bereich von −32768 bis 65535 liegen. Das heißt, daß maximal eine vierstellige Hexzahl geliefert wird. Trotzdem kann man den String durch führende Nullen auf die Länge gA2 bringen. Wenn die Hexzahl von gA1 länger ist als gA2, wird gA2 ignoriert.

Verwendungsmöglichkeit: Viele Programmierer schätzen das Hexadezimale Zahlensystem, weil es ein genaueres Bild vom Inhalt einer Speicherstelle vermittelt als das Dezimalsystem. Deshalb wird diese Funktion meistens dazu benutzt, Speicherinhalte auf dem Bildschirm darzustellen. Ein Programm, das dazu in der Lage ist, nennt man Monitor. Hier eine Miniaturversion:

```
10 MODE 1
20 ZONE 3
30 FOR i=0 TO 24
40 PRINT HEX$(p,4),
50 FOR j=0 TO 7
60 PRINT HEX$(PEEK(p),2),
70 p=p+1
80 NEXT j
90 p=p-8
100 FOR j=0 TO 7
```

```
110 PRINT CHR$(1);CHR$(PEEK(p));
120 p=p+1
130 NEXT
140 PRINT
150 NEXT
160 IF INKEY$="" THEN 160
170 GOTO 30
```



HIMEM

Systemvariable: Enthält die Basic-Obergrenze, also die Adresse des höchsten zur Verfügung stehenden Speicherplatzes. HIMEM kann nicht direkt geändert werden. Benutzen Sie dafür den Befehl MEMORY. Beim Einschalten des Schneiders enthält HIMEM die Adresse 43903.

Da HIMEM durch SYMBOL AFTER verändert wird, empfiehlt es sich, vor HIMEM die SYMBOL AFTER-Anweisung zu setzen, damit nicht der benutzereigene Zeichensatz Maschinenroutinen zerstört.

IF Bedingung THEN Befehl ⟨skette⟩1 ⟨ELSE Befehl⟨skette⟩2⟩

Befehl: Führt Befehlskette 1 aus, wenn die Bedingung als wahr erkannt wird, also einen Wert ungleich Null ergibt (siehe S. 115). Wenn die Bedingung Null ergibt, wird Befehlskette 2 ausgeführt, wenn die ELSE-Option genutzt wurde; sonst wird direkt in der nächsten Programmzeile fortgefahren.

Die IF-Anweisung ist im Programmaufbau einer der wichtigsten Befehle. Ein Großteil der Steuerung wird über diesen Befehl durchgeführt. Sie hat nur einen Nachteil: Jede IF-Abfrage kostet Zeit – und zwar viel Zeit. Da immer erst der Wert der Bedingung ausgewertet werden muß, ist es, wenn man eine Bedingung mehrmals überprüfen muß, vernünftiger, diese einer Variablen zuzuordnen. Als Demonstration dieses Beispiel:

```
10 c=TIME
20 FOR i=1 TO 300
30 IF i=150 THEN PRINT"150"
40 IF i=150 THEN PRINT"Schleifendurchlae
ufe"
50 IF i=150 THEN PRINT"hab ich schon!"
60 NEXT
70 PRINT TIME-c
```

Sie erhalten als gemessene Zeit einen Wert über 600. Wenn Sie Ihr Programm ändern und die Zeile 25 hinzufügen, könnte das Programm durch die zusätzliche Zeile langsamer werden. Statt dessen sparen Sie 50 Zeiteinheiten.

```
10 c=TIME
20 FOR i=1 TO 300
25 b%=(i=150)
30 IF b% THEN PRINT"150"
40 IF b% THEN PRINT"Schleifendurchlaeufe
"
50 IF b% THEN PRINT"hab ich schon!"
60 NEXT
70 PRINT TIME-c
```

Im Gegensatz zu der Beschränkung, die im Handbuch steht, kann auch nach THEN und ELSE ein weiterer IF-Befehl stehen.

```
10 CLS
20 DATA Inge,w,blau,rot
30 DATA Helga,w,blau,schwarz
```

```
40 DATA Max,m,gruen,grau
50 DATA Heinz,m,braun,blond
60 DATA x,x,x,x
70 INPUT"Suchen Sie eine Frau";fr$
80 IF fr$="ja" THEN kenn$="w":GOTO 120
90 INPUT"Suchen Sie einen Mann";fr$
100 IF fr$<>"ja" THEN PRINT"Tschuess..."
:END
110 kenn$="m"
120 INPUT"Augenfarbe";auge$
130 INPUT"Haarfarbe";haar$
140 READ na$,ke$,au$,ha$
150 h%=(ha$=haar$)
160 a%=(au$=auge$)
170 k%=(ke$=kenn$)
180 IF na$="x" THEN PRINT"Tut mir leid,
sonst kenne ich niemanden":END
190 IF NOT k% THEN GOTO 210 ELSE IF a% T
HEN PRINT"Die Augenfarbe stimmt!":IF h%
THEN PRINT"Ihr Wunschpartner heisst ";na
$:END ELSE PRINT"Leider nicht die Haarfa
rbe!"
200 IF a% AND NOT h% THEN INPUT"Soll ich
 weitersuchen";e$:IF e$="nein" THEN END
210 GOTO 140
```

INK gA1,gA2 ⟨,gA3⟩

Befehl: Weist der Farbnummer gA1 (0–16) eine oder zwei der 26 verfügbaren Farben zu (gA2). Wenn mit gA3 eine zweite Farbe gewählt wird, so bestimmt die Anweisung SPEED INK die Blinkdauer. Die Zuordnung gilt für alle Bildschirmmodi bis zur nächsten Änderung. Allerdings sind je nach gewähltem MODE nur vier oder zwei Farben wählbar: in MODE 1 die Farben 0 bis 3 und in MODE 2 die Farben 0 und 1.

Verwendungsmöglichkeiten: Zusätzlich zu der normalen Verwendung kann man mit der INK-Anweisung erreichen, daß erst ein komplett aufgebauter Bildschirm sichtbar wird. Man setzt mit INK Hintergrund und Zeichenfarbe auf den gleichen Wert, zeichnet dann das gewünschte Bild und schaltet erst dann mit INK die Zeichenfarbe sichtbar ein.

```
10 MODE 1
20 DEG
```

```
30 ORIGIN 320,200
40 INK 0,1
50 INK 1,1
60 PLOT 0,200,1
70 FOR i=0 TO 360 STEP 5
80 DRAW 320*SIN(120+i),200*COS(120+i),1
90 DRAW 320*SIN(240+i),200*COS(240+i),1
100 DRAW 320*SIN(i),200*COS(i),1
110 NEXT
120 INK 1,24
```

INKEY (gA)

Ganzzahlfunktion: Gibt als Zahlenwert zurück, ob eine Taste gedrückt wurde. Je nach dem zurückgegebenen Zahlenwert kann man erkennen, ob zusammen mit der Taste SHIFT, CTRL oder beides gedrückt wurde. Die Abfrage dieser Funktion ist etwas problematisch, weil eine gedrückte Taste den Wert 0 liefert, wenn weder SHIFT noch CTRL gedrückt wurde. Ein solcher Wert wird aber vom Computer als unwahr erkannt. Wenn die Taste nicht gedrückt ist, liefert INKEY minus 1, was normalerweise der Wert für »wahr« ist. Zur Lösung gibt es zwei Möglichkeiten:

```
10 PRINT"Noch ein Spiel J/N"
20 IF NOT INKEY(45) THEN RUN
30 IF NOT INKEY(46) THEN END
40 GOTO 20
```

und

```
10 PRINT"Noch ein Spiel J/N"
20 IF INKEY(45) THEN ELSE RUN
30 IF INKEY(46) THEN ELSE END
40 GOTO 20
```

Im Gegensatz zu INKEY$ nimmt INKEY kein Zeichen aus dem Tastaturpuffer. Die folgende Zeile wartet auf das Drücken von »J«; jedes andere Zeichen wird in den Tastaturpuffer übernommen:

```
10 IF INKEY(45) THEN GOTO 10
```

Wenn Sie diese Zeile benutzen, müssen Sie hinterher den Tastaturpuffer leerräumen:

```
20 IF INKEY$<>"" THEN 20
```

INKEY ist außerdem eine Möglichkeit, das alleinige Drücken von SHIFT und CTRL abzufragen, da diese Tasten bekanntlich kein Zeichen erzeugen.

INKEY$
Stringfunktion: Liefert das erste Zeichen aus dem Tastaturpuffer, wartet aber nicht auf einen Tastendruck. Dafür muß der Programmierer durch eine Schleife selbst sorgen.
Neben dem Warten auf das Drücken einer bestimmten Taste durch:

```
10 a$=INKEY$:if a$<>"j" THEN GOTO 10
```

kann man mit INKEY$ eine Eingaberoutine formulieren, die aus einem Unterprogramm einen String bestimmter Länge, bestehend aus bestimmten Zeichen, übergibt. So erspart man sich die lästige Fehlermeldung von INPUT und vermeidet außerdem, daß Steuerzeichen eingegeben werden, die bei einem späteren PRINT-Befehl den Bildschirmaufbau zerstören.

Die Variablen im Hauptprogramm:
l=maximale Länge des Eingabestrings
u=ASCII-Code des niedrigsten CHR$
o=ASCII-Code des höchsten CHR$
zahl=enthält nach Sprung ins Unterprogramm die eingegebene Zahl

Die Variablen im Unterprogramm:
a$=eingegebenes Zeichen
b$=aus den eingegebenen Zeichen zusammengesetzter String
cu$=Stern als simulierter Cursor plus Schritt nach links

```
10 l=4
20 u=48
30 o=57
40 PRINT"Bitte eine Zahl bis 9999 ";
50 GOSUB 50000
60 PRINT"Die eingegebene Zahl ist: "b$
70 zahl=VAL(b$)
80 END
50000 a$="":b$="":cu$="*"+CHR$(8)
50010 PRINT cu$;
50020 a$=INKEY$
50030 IF a$=CHR$(13) THEN 50110
50040 IF a$=CHR$(127) THEN 50120
```

```
50050 IF a$<CHR$(u) OR a$>CHR$(o) THEN G
OTO 50020
50060 IF LEN(b$)>1-1 THEN GOTO 50020
50070 IF LEN(b$)>1-2 THEN cu$=""
50080 PRINT a$;cu$;
50090 b$=b$+a$
50100 GOTO 50020
50110 PRINT:RETURN
50120 IF LEN(b$)=0 THEN GOTO 50020
50130 IF LEN(b$)<1 THEN PRINT CHR$(32);C
HR$(8);CHR$(8);cu$;
50140 IF LEN(b$)=1 THEN PRINT CHR$(8);:c
u$="*"+CHR$(8):PRINT cu$;
50150 b$=LEFT$(b$,LEN(b$)-1)
50160 GOTO 50020
```

INP(gA)

Ganzzahlfunktion: Liefert den Wert, der an der durch gA angegebenen Schnittstellenadresse liegt.

INPUT

Befehl: Füllt über ein Eingabegerät eine oder mehrere Variablen.

Da beim Schneider der Inputbefehl sehr viele verschiedene Möglichkeiten bietet, fangen wir am besten klein an.

```
10 INPUT a
```

Diese Programmzeile macht bei der Ausführung folgendes: Es wird ein Fragezeichen ausgedruckt und solange gewartet, bis der Benutzer die ENTER-Taste drückt. Erst danach wird überprüft, ob das, was der Benutzer zwischen dem Fragezeichen und dem Drücken der ENTER-Taste getippt hat, auch in die Variable a paßt. Wenn das aus irgendwelchen Gründen nicht geht (weil der Benutzer z. B. statt der Zahl Eintausend die Kleinbuchstaben looo eingegeben hat), erscheint eine Meldung an den Benutzer »Redo from start« (frei übersetzt: Das Ganze noch einmal) und es erscheint ein neues Fragezeichen.

```
10 INPUT"Wie heißt Du";a$
```

Damit der Benutzer weiß, was er eingeben soll, hat der Programmierer die Möglichkeit, es ihm in einem Text mitzuteilen. Dieser Text muß in Anführungsstrichen eingeschlossen werden – allerdings darf er alle Steuerzeichen enthalten, die über die Tastatur erreichbar sind. So kann man zum Beispiel den Text an eine bestimmte Stelle des Bildschirms bringen.

```
10 INPUT"XXXXWie heißt Du";a$
```

Für das erste X drücken Sie ⟨CTRL⟩ und ⟨L⟩, für das zweite ⟨CTRL⟩ und ⟨SHIFT⟩ und Null, für das dritte X ⟨CTRL⟩ und ⟨J⟩ und für das vierte ⟨CTRL⟩ und ⟨L⟩.

Es gibt Fälle, da stört das obligatorische Fragezeichen von INPUT gewaltig. Um dieses Problem zu lösen, setzen Sie statt des Semikolons (;) ein Komma:

```
10 INPUT"Bitte eine Zahl von 1 bis 1000",a
```

Wenn Sie glauben, daß damit die Möglichkeiten des Schneider erschöpft sind, liegen Sie falsch. Sie können mit einem INPUT-Befehl mehrere Variablen zu füllen.

```
10 INPUT"Bitte ein Koordinatenpaar,
    durch Komma getrennt ",x,y
```

Diese Schreibweise hat allerdings einen Nachteil. Wenn der Benutzer die Daten eingibt, und sich bei der letzten Information verschreibt, meldet sich der Computer mit »Redo« und der Arme darf von vorne anfangen. Die einzelnen Daten müssen durch Kommata getrennt werden. Leerzeichen oder ⟨ENTER⟩ sind nicht erlaubt.

Eine weitere Besonderheit bietet INPUT bei der Eingabe von Zeichenketten. Da Kommas normalerweise als Trennzeichen interpretiert werden, sie aber ab und zu auch in Strings stehen müssen, kann man bei der Eingabe von Strings die Anführungszeichen wie beim Programmieren verwenden.

```
10 INPUT"Nachname,Vorname";a$

20 PRINT a$
```

Versuchen Sie die Variable a$ mit Meier, Anton zu füllen. Sie werden vom Schneider mit »Redo« ermahnt. Geben Sie dann "Meier, Anton" ein und jeder ist zufrieden. Damit aber nicht genug: Das ⟨ENTER⟩ nach einer Eingabe sorgt normalerweise dafür, daß die nächste Druckausgabe eine Zeile tiefer rutscht. Auch das kann man verhindern:

```
10 MODE 2

20 INPUT;"Eine negative Zahl ",z
```

```
30  IF z>0 THEN PRINT TAB(40);
                "NEGATIV SAGTE ICH!"+CHR$(13)
                :GOTO 10 ELSE PRINT TAB(40);SPACE$(18)
```

Das Semikolon nach dem Wort INPUT sorgt dafür, daß der Cursor nicht in die nächste Zeile rutscht.

Der INPUT-Befehl beim Schneider hat einen kleinen Schönheitsfehler: Bei Stringeingaben werden sowohl führende als auch folgende Leerstellen »vergessen«.

```
10 INPUT "Test",a$

20 PRINT LEN(a$)
```

Geben Sie ein: ****ABCD****, ersetzen Sie aber bei der Eingabe das Sternchen durch Leerstellen. Statt der zu erwartenden Länge 8 oder 12 erhalten Sie 4. Das kann, wenn man nicht darauf vorbereitet ist, bei der Erstellung von Suchroutinen zu Problemen führen.

Bis hierhin haben wir uns um die Eingabe von der Tastatur gekümmert. Setzen Sie hinter INPUT noch das »#« und eine Zahl von 0 bis 9, dann haben Sie zusätzliche Möglichkeiten. Die Zahlen von 0 bis 7 geben das WINDOW an, über das Sie die Eingabe haben möchten. Für diese gilt alles, was oben gesagt wurde. Die Zahl 8 leitet die Eingabe auf die parallele Schnittstelle, an der normalerweise der Drucker hängt. Da der Drucker aber eine denkbar schlechte Eingabeeinheit ist, wird der Schneider lange auf sein ⟨ENTER⟩ warten müssen. Die Zahl 9 legt die Eingabe auf den Kassettenrecorder. Die Benutzung dieser Zahl führt zu »EOF met«, wenn Sie die Datei nicht geöffnet haben. Damit Sie sich den Unterschied zu Tastatureingaben klarmachen können, errichten wir eine Testdatei auf Kassette und lesen sie nachher wieder ein.

```
10 MODE 2
20 a$="Andreas "
30 b$="Werminghoff"
40 OPENOUT"TEST"
50 FOR i=1 TO 10
60 PRINT#9,i
70 NEXT i
80 FOR i=1 TO 10
90 PRINT#9,i;
100 NEXT i
110 PRINT#9,"Schneider CPC 464";
```

```
120 PRINT#9,"DuMont creativ"
130 PRINT#9,"Computerbuecher"
140 PRINT#9,a$,b$
150 PRINT#9,a$;",";b$
160 CLOSEOUT
170 PRINT"Bitte spulen Sie die Kassette
zurueck und druecken Sie dann auf ENTER"
180 DIM i$(30)
190 i=1
200 IF INKEY$<>CHR$(13) THEN GOTO 200
210 OPENIN"Test"
220 CLS
230 INPUT#9,i$(i)
240 PRINT i;"Eingabe: ";i$(i)
250 i=i+1
260 IF NOT EOF THEN GOTO 230
270 CLOSEIN
280 PRINT"Da sind Sie platt, was?!"
```

In den Zahlen 50–150 wird dem Computer insgesamt 25 mal ein PRINT-Befehl für die Kassette gegeben. Das erste, was beim Einlesen auffällt, ist die Tatsache, daß nur 15 Strings wieder eingelesen werden. Schauen wir uns diese Strings etwas genauer an. Die ersten 10 sind verständlich; sie wurden durch die Schleife der Zeilen 50 bis 70 erzeugt. Vollkommenes Unverständnis stellt sich beim 11. String ein. Erst die 10 Zahlen aus den Zeilen 80–100, dann die Strings aus Zeile 110 und 120. Erinnern wir uns: Eine Eingabe bei INPUT wird durch CHR$(13) (ENTER) abgeschlossen. Das Semikolon in Zeile 90 und 110 verhindert aber, daß der Computer nach dem Drucken ein CHR$(13) erzeugt. Wenn der Computer bei einem INPUT nach einem ENTER sucht, findet er das erste nach »creativ«. Alles vorher muß also zu einem String gehören. String Nr. 12 ist normal. Nach dem entsprechenden PRINT-Befehl stand kein Semikolon – also wurde CHR$(13) erzeugt. Der String Nr. 13 sieht etwas seltsam aus: Anscheinend hat der Schneider das Komma in Zeile 140 nicht als Trennungszeichen interpretiert, sondern als Anweisung, a$ auf eine bestimmte Länge zu bringen und dann b$ anzuhängen. Anders ist String 13 nicht zu erklären. Nun aber zu String 14 und 15. Diese beiden wurden durch einen PRINT-Befehl erzeugt. Zwischen den beiden stand aber als String(!) ein Komma. Nur solch ein Komma wirkt bei einer Ausgabe auf die Kassette als Trennungszeichen.

Also: Ausgaben auf Kassette werden begrenzt durch die Strings CHR$(13) und CHR$(44)! Leerzeilen sind keine (!) gültigen Trennungszeichen.

INSTR (⟨gA,⟩,sA1, sA2)

Ganzzahlfunktion: Liefert die Position von sA2 in sA1. Wird gA verwendet, so beginnt die Suche bei Position gA. Allerdings wird auch dann die Position relativ zum Beginn von sA ermittelt. Ist sA2 nicht in sA1 enthalten, so liefert INSTR den Wert Null.

gA darf nur Werte von 1 bis 255 enthalten. Alle anderen erzeugen »Improper argument«. Problematisch wird die Verwendung von INSTR, wenn sA2 ein Leerstring ist, also nichts enthält (""). Dann liefert INSTR nämlich die Position, an der mit der Suche begonnen wurde.

```
PRINT INSTR(3,"Schneider","")

 3

Ready
```

Man muß also vorher sA2 auf Gültigkeit überprüfen.

Verwendungsmöglichkeiten: Obwohl selten gebraucht, ist INSTR eine praktische Funktion bei effektiver Programmierung. Ist man zum Beispiel gezwungen, aus Platzgründen mehrere Strings in einen zu verwandeln, so hilft die INSTR-Funktion, diesen wieder auseinander zu pflücken.

```
10 DATA Anton,Mueller,Weizenstr. 15,7000
,Stuttgart 41,0700/1234567
20 trenn$=CHR$(144)
30 FOR i=1 TO 6
40 READ a$
50 b$=b$+a$+trenn$
60 NEXT
70 PRINT b$
80 vn=INSTR(b$,trenn$)
90 nn=INSTR(vn+1,b$,trenn$)
100 st=INSTR(nn+1,b$,trenn$)
110 pl=INSTR(st+1,b$,trenn$)
120 ot=INSTR(pl+1,b$,trenn$)
130 te=INSTR(ot+1,b$,trenn$)
140 PRINT vn,nn,st,pl,ot,te
150 PRINT"Vorname:   ";MID$(b$,1,vn-1)
160 PRINT"Nachname: ";MID$(b$,vn+1,nn-vn
-1)
```

```
170 PRINT"Strasse :  ";MID$(b$,nn+1,st-nn
-1)
180 PRINT"PLZ:       ";MID$(b$,st+1,pl-st
-1)
190 PRINT"Ort:       ";MID$(b$,pl+1,ot-pl
-1)
200 PRINT"Telefon:   ";MID$(b$,ot+1,te-ot
-1)
```

In diesem Beispiel würde b$ als ein langer String vom Kassettenrecorder eingelesen.

Eine weitere gute Möglichkeit, INSTR einzusetzen, ergibt sich aus der Beschränkung der Eingabe auf bestimmte Zeichen. Als Beispiel eine Routine, in der man als erstes Zeichen ein Vorzeichen eingeben kann, danach nur noch Zahlen und ⟨ENTER⟩.

```
10 MODE 1
20 pr$="+-1234567890"+CHR$(13):e=13
30 PRINT"Bitte eine Zahl ";
40 a$=INKEY$:IF a$="" THEN 40
50 a=INSTR(pr$,a$)
60 IF a=0 THEN 40
70 IF a=e THEN END
80 IF LEN(pr$)=13 THEN pr$=RIGHT$(pr$,11
):e=11
90 PRINT a$;
100 GOTO 40
```

INT (nA)

Gleitkommafunktion: Liefert bei einem nicht ganzzahligen nA die nächst kleinere Ganzzahl:

In Basic die gebräuchlichste Funktion zur Ermittlung eines ganzzahligen Anteils. Allerdings bietet FIX in den meisten Fällen die bessere Funktion.

JOY (gA)

Ganzzahlfunktion: Gibt als Wert zwischen 0 und 255 den Zustand des Joysticks 0 oder 1 zurück. Die einzelnen Bits des Joystickports geben den Zustand der Joystickschalter zurück.

Verwendungsmöglichkeiten: Alle Programmteile, die vom Benutzer eine Entscheidung verlangen, können auch über eine Joystick-Abfrage gesteuert werden. Hier eine schnelle Joystickabfrage:

```
10 a=JOY(0)
20 ON INSTR(BIN$(a,6),"1") GOSUB 600,500
,400,300,200,100
30 GOTO 10
100 PRINT CHR$(8);CHR$(11);CHR$(240);
199 RETURN
200 PRINT CHR$(8);CHR$(10);CHR$(241);
299 RETURN
300 PRINT CHR$(8);CHR$(8);CHR$(242);
399 RETURN
400 PRINT CHR$(243);
499 RETURN
500 PRINT CHR$(8);CHR$(238);
599 RETURN
600 PRINT CHR$(30);
699 RETURN
```

Der Wert von JOY(0) wird in eine Binärzahl umgewandelt und dann nach dem Auftreten der ersten 1 gefragt. Sie können das nachprüfen, indem Sie eine Zeile 15 einfügen:

```
15 PRINT BIN$(a,6);" ";:GOTO 10
```

Aus diesem Programm können Sie ein nettes Spiel machen, wenn Sie ein Bild vorgeben, und dieses mit dem Joystick nachfahren müssen.

KEY gA, sA

Befehl: Belegt eine Taste des Zehnerblocks mit sA. Dabei gibt gA die Taste an.

Sie können maximal 109 Zeichen zur Belegung der Tasten verwenden. Dabei ist die Verteilung auf die einzelnen Tasten abhängig von der Zahl der belegten Tasten: Sie können also alle Zeichen auf eine Taste legen oder auf jede 9 Zeichen. Um die normale Funktion der Tasten nicht zu zerstören, empfehle ich eine Umbelegung des ASCII-Codes der Tasten durch KEY-DEF, so daß die Steuerfunktionen durch die Kombination von CTRL und der entsprechenden Taste erreicht werden.

```
10 KEY DEF 13,0,129,129,141
20 KEY DEF 14,0,130,130,142
```

```
30 KEY DEF  5,0,131,131,143
40 KEY DEF 20,0,132,132,144
50 KEY DEF 12,0,133,133,145
60 KEY DEF  4,0,134,134,146
70 KEY DEF 10,0,135,135,147
80 KEY DEF 11,0,136,136,148
90 KEY DEF 3,0,137,137,149
100 help$=CHR$(24)+CHR$(4)+CHR$(2)+CHR$(
31)+CHR$(1)+CHR$(26)
110 help$=help$+" 1 RUN"+CHR$(1)+CHR$(13
)+"  2 LIST  3 LOAD"+CHR$(34)
120 help$=help$+"  4 SAVE"+CHR$(34)+"  5
 AUTO  6 EDIT  7 CLS"+CHR$(1)
130 help$=help$+CHR$(13)+"  8 MODE 2"+CH
R$(1)+CHR$(13)+"  9 HELP"+CHR$(1)+CHR$(1
3)
140 help$=help$+CHR$(24)+CHR$(26)+CHR$(0
)+CHR$(79)+CHR$(0)+CHR$(23)
150 KEY 141,"RUN"+CHR$(13)
160 KEY 142,"LIST "
170 KEY 143,"LOAD"+CHR$(34)
180 KEY 144,"SAVE"+CHR$(34)
190 KEY 145,"AUTO "
200 KEY 146,"EDIT "
210 KEY 147,"CLS"+CHR$(13)
220 KEY 148,"MODE 2"+CHR$(13)
230 KEY 149,"PRINT help$"+CHR$(13)
```

Wenn Sie aus diesem Programm die Zeilen 100–150 umnumerieren auf 65530 bis 65535, haben Sie auch nach dem Löschen des Hauptprogramms die Möglichkeit, Ihre Tastenbelegung auf dem Bildschirm zu sehen.

KEY DEF gA1, gA2⟨,gA3⟨,gA4⟨,gA5⟩⟩⟩

Befehl: Belegt eine Taste des Schneiders mit einem neuen ASCII-Code. Dabei haben die Zahlen folgende Bedeutung: gA1: Tastennummer (siehe Anhang III S.16 des Handbuches)
gA2: Wiederholung (0 ohne Wiederholfunktion, 1 mit Wiederholfunktion)
gA3: ASCII-Code bei Drücken der Taste alleine

gA4: ASCII-Code bei Drücken der Taste mit SHIFT
gA5: ASCII-Code bei Drücken der Taste mit CTRL oder mit SHIFT und CTRL

Verwendungsmöglichkeiten: In der Praxis wird der KEY DEF-Befehl in zwei Zusammenhängen benutzt. Einmal, um das Unterbrechen des Programms durch die Break-Taste zu verhindern. (übrigens kann man sie wieder zurückstellen mit: KEY DEF 66,0,252,252,252):

```
10 KEY DEF 66,0,0,0,0
```

Zum zweiten, um einen mit SYMBOL erstellten Zeichensatz auf die englische Tastatur zu legen. Ein ausführliches Beispiel sehen Sie bei SYMBOL. Hier sei nur die Vertauschung von Z und Y gezeigt.

```
10 KEY DEF 71,1,&79,&59,&19

20 KEY DEF 43,1,&7A,&5A,&1A
```

LEFT$ (sA, gA)

Stringfunktion: Liefert von sA die ersten durch gA bestimmten Zeichen. Ist sA kürzer, als gA verlangt, wird der gesamte sA ausgegeben. (Für Programmierer mit Kenntnissen in anderen Programmiersprachen: Es wird nicht mit Leerstellen aufgefüllt!)

Verwendungsmöglichkeiten: Häufig wird LEFT$ dazu benützt, um von einem vorhandenen String, dessen Länge nicht bekannt ist, das letzte Zeichen zu entfernen. Wenn Sie sich die Eingaberoutine bei INKEY$ ansehen, stellen Sie fest, daß beim Drücken der DEL-Taste (CHR$(127)) von dem Eingabestring das letzte Zeichen abgeschnitten wird. Wichtiger zur Manipulation von Strings sind RIGHT$ und MID$.

LEN (sA)

Ganzzahlfunktion: Ermittelt die Länge eines Strings. Falls der String ein Leerstring ist, ergibt LEN Null.

Verwendungsmöglichkeit: Meistens benutzt man LEN, um einen String auf seine zulässige Länge zu überprüfen. Viele Funktionen erlauben keinen Leerstring als Argument; wenn man das vorher abfängt, erspart man sich Ärger.

```
10 INPUT"Wie heisst Du";a$

20 IF LEN(a$)=0 then ?"Namenlos?":GOTO 10

30 PRINT ASC(a$)
```

Die Zeile 30 erzeugt ein »Improper Argument«, wenn die Zeile 20 fehlt und der Benutzer nur auf die ENTER-Taste drückt.

Eine weitere Verwendung ergibt sich beim Formatieren von Strings; im Schneider-BASIC ist es leider nicht möglich, eine Zahl in der gleichen Form auf einen String zuzuweisen, die sie bei PRINT USING hat. Da man so etwas bei Dateiverwaltung jedoch ab und zu benötigt, benutze ich folgende Routine:
l ist die Länge des auszugehenden Strings
nk die Anzahl der Nachkommastellen
z die zu formatierende Zahl
zz$ die formatierende Zahl als String
statt des Sternchens nimmt man besser ein Leerzeichen, damit man mit der Zahl auch rechnen kann.

```
10 INPUT"Irgendeine Zahl";z
20 l=10
30 nk=3
40 GOSUB 100
50 PRINT zz$
60 GOTO 10
100 zz=ROUND(z,nk):zv=SGN(zz):za=ABS(zz)
105 IF LEN(STR$(ABS(zz)*10^nk))>l THEN P
RINT"Zahl zu gross!":GOTO 170
110 zf$="*"
120 zz$=MID$(STR$(za),2)
130 IF zz=INT(zz) THEN zz$=zz$+"."
140 IF ASC(RIGHT$(zz$,nk+1))<>46 THEN zz
$=zz$+"0":GOTO 140
150 IF zv=-1 THEN zf$=zf$+"-"
160 IF LEN(zz$)<l THEN zz$=zf$+zz$:GOTO
160
170 RETURN
```

⟨LET⟩ variable = Ausdruck

Befehl: Weist der Variablen den ausgewerteten Ausdruck zu. Bei der Verarbeitung einer Zuweisung geht der Computer immer folgendermaßen vor: Erst wird der Ausdruck ausgewertet, dann der ermittelte »Wert« in die Variable links vom Gleichheitszeichen gefüllt. Das Gleichheitszeichen hat in BASIC zwei Bedeutungen: In einer Zuweisung erfüllt es die Aufgabe eines Befehls, in einem Vergleich seine ursprüngliche Aufgabe.

LINE INPUT

Befehl: Funktioniert genauso wie der normale INPUT-Befehl – mit zwei Unterschieden: Erstens werden keine Zeichen außer ⟨ENTER⟩ als Trennungszeichen erkannt, und zweitens wird standardmäßig kein Fragezeichen ausgedruckt. LINE INPUT ist nur mit Stringvariablen zu verwenden. Eine Miniaturtextverarbeitung als Beispiel.

```
10 CLS
20 DIM a$(30)
30 LINE INPUT a$(i)
40 IF a$(i)="ende" THEN i=i-1:GOTO 70
50 i=i+1
60 GOTO 30
70 CLS
80 FOR j=0 TO i
90 l=1
100 IF l>195 THEN k=255:GOTO 150
110 FOR k=l+60 TO l+40 STEP-1
120 e$=MID$(a$(j),k,1)
130 IF e$=" " OR e$="," OR e$="." THEN 1
50
140 NEXT k
150 PRINT#0,MID$(a$(j),l,k-l+1)
160 l=k+1
170 IF l<LEN(a$(j)) THEN 100
180 NEXT j
```

LIST

Systembefehl: Listet die angegebenen Programmzeilen auf das angegebene Ausgabegerät. Folgende Formate sind zulässig:

LIST : Listet das gesamte Programm;
LIST znr : Listet die angegebene Programmzeile;
LIST – znr : Listet bis zur angegebenen Programmzeile;
LIST znr – : Listet ab der angegebenen Programmzeile;
LIST znr1 – znr2 : Listet von znr1 bis znr2.

Die Nummer 0 bis 7 für das Ausgabegerät setzt die Liste auf ein Window, die Nummer 8 schickt die Ausgabe auf den Drucker. Auf den Kassettenrecorder kann man auf zweierlei Arten listen: Das ganze Programm mit SAVE„name",A oder Programmteile mit

```
OPENOUT"name"

LIST 100-200,#9

CLOSEOUT
```

Wenn Sie ein Programm so gelistet haben, können Sie es mit LOAD ganz normal laden.

LOAD

Systembefehl: Lädt ein Programm vom Kassettenrecorder in den Speicher. Der Adreßausdruck beim Laden von Binärdateien gibt an, ab welcher Adresse ein Programm geladen werden soll.

LOCATE ⟨# Ausgabegerät,⟩ gA1,gA2

Befehl: Setzt die Position auf dem Ausgabegerät fest, an der der nächste Druckvorgang stattfinden soll. Dabei setzt gA1 die X-Koordinate fest und gA2 die Y-Koordinate.

Koordinaten, die außerhalb des Bildschirms liegen, haben unterschiedliche Wirkungen. So wird bei einer X-Koordinate zwischen rechtem Bildrand und 255 am linken oder rechten Rand der nächsten Zeile gedruckt. Y-Koordinaten zwischen 0 und 128 lassen den Bildschirm vorm Drucken eine Zeile aufwärts scrollen, ab 129 wird der Bildschirm nach unten gescrollt und danach wird in der obersten Zeile ausgedruckt. Koordinaten über 255 erzeugen ein Improper Argument.

Verwendungsmöglichkeiten: LOCATE bietet die Möglichkeit, erst den Bildschirm aufzubauen und dann die Eingaben zu erwarten.

```
10 MODE 2
20 DIM fe$(6),in$(6)
30 FOR i=1 TO 6
```

```
40 READ fe$(i)
50 NEXT
60 DATA Anrede,Vorname,Nachname,Strasse,
PLZ Ort,Telefon
70 FOR i=1 TO 6
80 LOCATE 10,3+2*i
90 PRINT fe$(i)
100 NEXT
110 FOR i=1 TO 6
120 LOCATE 30,25:PRINT SPACE$(20);
130 LOCATE 20,3+2*i:PRINT SPACE$(60)
140 LOCATE 20,3+2*i
150 INPUT in$(i)
160 IF LEN(in$(i))>20 THEN LOCATE 30,25:
PRINT"EINGABE ZU LANG!":GOTO 130
170 NEXT
```

LOG (nA)

Gleitkommafunktion: Ergibt den natürlichen Logarithmus des ausgewerteten numerischen Ausdrucks.

```
10 CLS
20 a=EXP(1)
30 b=LOG(10)
40 c=a^b
50 PRINT"Die Eulersche Zahl (e) lautet:"
a
60 PRINT"Der natuerliche Logarithmus von
 10 ist:"b
70 PRINT"e hoch dem nat. Logarithmus von
 10 ist:"c
```

LOG10 (nA)

Gleitkommafunktion: Gibt an, mit welcher Zahl man 10 potenzieren muß, um den numerischen Ausdruck zu erhalten. Es gilt: 10^2 ist gleich 100; also ist LOG10(100) gleich 2.

LOWER$ (sA)

Stringfunktion: Liefert einen String, der identisch ist mit dem ausgewerteten Stringausdruck, nur daß alle Großbuchstaben durch Kleinbuchstaben ersetzt werden. Achtung! Bei der Umformung ändert sich natürlich auch der ASCII-Code.

Verwendungsmöglichkeiten: Bei der Tastaturabfrage ist es möglich, daß der Benutzer die CAPS LOCK-Taste gedrückt hat. Um sich umständliche Abfragen der verschiedenen Buchstaben zu ersparen, benutzt man LOWER$ oder UPPER$

```
10 a$=LOWER$(INKEY$)

20 IF a$="n" THEN END

30 IF a$="j" THEN RUN

40 GOTO 10
```

Eine weitere Verwendung ergibt sich beim Suchen:

```
10 DATA Mueller,Anton,Meier,Ralph
20 DATA Schneider,Ilse,Schmitz,Erika
30 DATA Schultz,Gustav,Diebold,Percy
40 DATA Neuner,Albert,Mayer,Ida
50 DATA x,x
60 INPUT"Nach welchem Namen suchen Sie";
n$
70 RESTORE
80 su$=LOWER$(n$)
90 READ a$,b$
100 IF a$="x" THEN PRINT"Dateiende errei
cht!":GOTO 60
110 IF su$=LOWER$(a$) THEN PRINT"Vorname
 :";a$:PRINT"Familienname: ";b$
120 GOTO 90
```

MAX (nA1⟨,nA2,nA3....⟩)

Gleitkommafunktion: Liefert aus der Liste numerischer Ausdrücke den Wert des größten.

Verwendungsmöglichkeiten: Sortierroutinen und Ermittlung des größten Werts aus einer Tabelle.

```
10 DIM a(100)
20 FOR i=1 TO 100
30 a(i)=INT(1000*RND)+1
40 PRINT a(i);
50 NEXT
60 FOR i%=1 TO 100
70 a=MAX(a,a(i%))
80 NEXT
90 PRINT" Die groesste Zahl war";a
```

MEMORY gA

Befehl: Setzt die für BASIC benutzbare Speicherobergrenze hinunter, um Platz für Maschinenroutinen und Daten zu sichern.

Vorsicht! Die Speicherobergrenze wird von BASIC durch SYMBOL AFTER wieder verändert. Es ist nicht möglich, die Speichergrenze in den vom BASIC-Interpreter geschützten RAM-Bereich zu verlegen. Spielereien mit dem MEMORY-Befehl können seltsame Folgen zeigen:

```
10 MEMORY 4000

20 SYMBOL AFTER 127
```

MERGE sA

Systembefehl: Lädt ein Programm mit dem Namen sA von der Kassette zu dem im Speicher. Sollten sowohl im vorhandenen Programm wie im nachzuladenen Programm die gleichen Zeilennummern vorhanden sein, so überschreibt das neue Programm die Zeilennummern des alten. Abgesehen davon wirkt es genau wie LOAD. MERGE kann nicht im Programm angewendet werden, da es den Programmablauf unterbricht.

MID$ (sA,gA1⟨,gA2⟩)

Stringfunktion: Ermittelt aus sA einen String, der bei gA1 beginnt und die Länge gA2 hat. Wird gA2 weggelassen, so liefert MID$ den gesamten Rest des Strings, beginnend bei gA1.

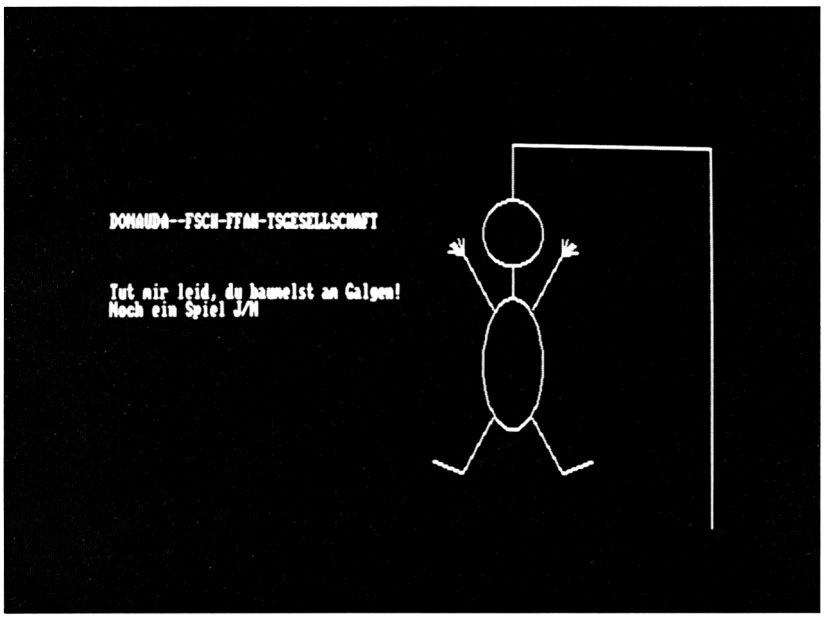
Sollten gA1 oder gA2 die Länge des Stringausdrucks überschreiten, so liefert MID$ einen Leerstring. Wenn Sie gA1 gleich Null setzen, so erhalten Sie ein »Improper argument«.

Verwendungsmöglichkeiten: Wenn in einem langen String nach einem Buchstaben oder einer Buchstabenkombination gesucht wird, die an einer beliebigen Stelle stehen kann, so können Sie MID$ verwenden.

MID$ (sA1,gA1⟨,gA2⟩)=sA2

Befehl: Ersetzt in sA1 ab gA1 alle Zeichen durch die ersten gA2 Zeichen von sA2. Es werden nie mehr Zeichen ersetzt, als sA1 lang ist.

```
10 'An den Galgen!
20 MODE 2
30 DATA Donaudampfschiffahrtsgesellschaf
t
40 DATA x
50 FOR i=1 TO RND*100
60 READ a$:IF a$="x" THEN RESTORE
70 NEXT
```



```
80 IF a$="x" THEN 50
90 LOCATE 1,8
100 FOR i=1 TO LEN(a$)
110 PRINT"-";:v$=v$+" "
120 NEXT
130 PRINT
140 FOR i=1 TO 15
150 LOCATE 1,12
160 PRINT"Welchen Buchstaben nimmst Du j
etzt?"
170 LOCATE 1,8
180 i$=UPPER$(INKEY$)
190 IF i$<"A" OR i$>"Z" THEN 180
200 FOR j=1 TO LEN(a$)
210 IF i$=UPPER$(MID$(a$,j,1)) THEN MID$
(v$,j,1)=i$:s=1:LOCATE j,8:PRINT i$;
220 NEXT
230 IF UPPER$(a$)=v$ THEN GOTO 770
240 IF s=1 THEN s=0:GOTO 150
250 ON i GOSUB 280,310,330,350,400,430,4
70,500,530,580,630,660,690,720,750
260 PRINT
270 NEXT
280 ORIGIN 0,50
290 MOVE 600,-50:DRAW 600,300
300 RETURN
310 DRAWR -200,0
320 RETURN
330 DRAWR 0,-50
340 RETURN
350 DEG:
360 FOR j=0 TO 360 STEP 5
370 DRAW 400+30*SIN(j),220+30*COS(j)
380 NEXT
390 RETURN
400 MOVE 400,190
410 DRAW 400,170
420 RETURN
430 FOR j=0 TO 360 STEP 5
440 DRAW 400+30*SIN(j),100+60*COS(j)
```

```
450 NEXT
460 RETURN
470 MOVE 420,150
480 DRAW 450,200
490 RETURN
500 MOVE 380,150
510 DRAW 350,200
520 RETURN
530 FOR j=-280 TO -360 STEP -20
540 MOVE 450,200
550 DRAW 450+15*SIN(j),200+15*COS(j)
560 NEXT
570 RETURN
580 FOR j=280 TO 360 STEP 20
590 MOVE 350,200
600 DRAW 350+15*SIN(j),200+15*COS(j)
610 NEXT
620 RETURN
630 MOVE 420,50
640 DRAW 450,0
650 RETURN
660 MOVE 380,50
670 DRAW 350,0
680 RETURN
690 MOVE 450,0
700 DRAW 480,10
710 RETURN
720 MOVE 350,0
730 DRAW 320,10
740 RETURN
750 LOCATE 1,12:PRINT"Tut mir leid, du b
aumelst am Galgen!"
760 GOTO 780
770 LOCATE 1,12:PRINT"Du hast Dich vorm
Galgen gerettet!  "
780 PRINT"Noch ein Spiel J/N"
790 i$=UPPER$(INKEY$)
800 IF i$="N" THEN CLS:END
810 IF i$="J" THEN RUN
820 GOTO 790
```

MIN (nA1 ⟨,nA2,..⟩)

Gleitkommafunktion: Ermittelt aus einer Liste von numerischen Ausdrücken den Ausdruck mit dem kleinsten Wert. Umkehrfunktion von MAX.

MOD

Ganzzahloperator: Zwei Zahlen, durch MOD verknüpft, liefern den ganzzahligen Rest der Division. Sie kennen vielleicht noch aus den Anfängen Ihrer Schulzeit das Teilen von zwei Zahlen: 7 durch 4 ergibt 1, Rest 3; der MOD-Operator liefert die 3.

Die Operanden bei MOD dürfen nur im Bereich von −32768 bis +32767 liegen. Alle anderen Werte werden mit »Overflow« abgewiesen.

Verwendungsmöglichkeiten: Außer bei mathematischen Aufgaben hat MOD auch Bedeutung, wenn man die Parameter einer Funktion in gültigen Grenzen halten will, obwohl man nicht weiß, welchen Wert ein Argument annehmen kann.

```
10 FOR i=0 TO 15
20 INK i,i
30 NEXT
40 MODE 0
50 FOR k=0 TO 400 STEP 64
60 FOR i=0 TO 640 STEP 64
70 FOR j=0 TO 63 STEP 4
80 b=b+1
90 a=b MOD 16
100 PLOT i+j,k,a
110 DRAW i+j,k+j
120 DRAW i,k+j
130 PLOT i+j,k+j+2
140 DRAW i,k+j+2
150 NEXT
160 NEXT
170 NEXT
180 WHILE INKEY$="":WEND
190 INK 1,24
```

MODE gA

Befehl: Setzt durch gA die graphische Auflösung und die Zahl der verfügbaren Farben fest. Dabei erzeugt gA bei

MODE	Auflösung X×Y	Zeichen/Zeile	Verfügbare Farben
0	160×200	20	16
1	320×200	40	4
2	640×200	80	2

MODE setzt alle WINDOWs und die ORIGIN-Werte zurück; der Bildschirmanfang wird auf &C000 gesetzt und der Bildschirm wird gelöscht.

Die MODE-Anweisung ändert nicht die Bildschirm-Koordinaten, sondern nur die Auflösung. Deshalb sollte man in den niedrigen Bildschirmmodi beim Zeichnen von senkrechten Linien mit größeren Schrittweiten arbeiten.

```
10 FOR i=0 TO 2
20 MODE i:DEG
30 WINDOW#1,1,9*2^i,1,10
40 WINDOW#2,1,9,14,24
50 PAPER#2,1:CLS#2
60 PAPER#1,1:CLS#1
70 PEN#1,0
80 PRINT"Schneider"
90 ORIGIN 320,194
100 DRAW 320,-200
110 FOR j=0 TO 360
120 PLOT 160+70*COS(j),120+70*SIN(j)
130 NEXT
140 FOR j=1 TO 4
150 FOR k=-320 TO 320 STEP j
160 FOR m=0 TO 2
170 PLOT k,j*8+m
180 NEXT
190 NEXT
200 NEXT
210 NEXT
```

MOVE gA1,gA2

Graphikbefehl: Setzt den Graphik-Cursor auf die durch gA1 und gA2 angegebenen X-Y-Koordinaten. Die tatsächliche Position auf dem Bildschirm wird bestimmt durch eine eventuell gegebene ORIGIN-Anweisung. Wenn gA1 und gA2 die Grenzen des Bildschirms verlassen, erscheint keine Fehlermeldung, dafür wird die Verarbeitung von graphischen Befehlen unerträglich langsam.

Da der MOVE – Befehl keine sichtbare Wirkung hat, vergleichen Sie die Beispielprogramme der anderen Graphikbefehle.

MOVER gA1,gA2

Graphikbefehl: Bewegt den Graphik-Cursor relativ zu seiner letzten Position um gA1 Einheiten in der X-Richtung und gA2 Einheiten in der Y-Richtung.

NEW

Systembefehl: Löscht den Beginn des BASIC-Programms im Speicher, setzt die meisten internen Zeiger wieder zurück. MEMORY- und SYMBOL-Anweisungen bleiben erhalten.

NEXT ⟨numerische Variable⟩

Befehl: Siehe FOR.

ON gA **GOTO** znr1⟨,znr2,...⟩

Befehl: Abhängig vom Wert gA wird in die erste, zweite, dritte, usw... Zeilennummer verzweigt. Werte unter 0 oder über 255 werden als »Improper argument« abgewiesen. Wenn weniger Zeilennummern vorhanden sind, als der Wert von gA angibt, oder gA den Wert 0 hat, wird in die nächste Programmzeile verzweigt.

ON gA **GOSUB** znr1⟨,znr2,...⟩

Befehl: Funktioniert genau wie der ON..GOTO-Befehl, nur daß hier in ein Unterprogramm verzweigt wird, das bei der entsprechenden Zeilennummer beginnt. Die Beschränkungen, denen gA unterliegt, sind auch wie bei ON...GOTO...

Verwendungsmöglichkeiten: Der ON..GOSUB..Befehl wird vor allen Dingen zur Menuesteuerung benützt. Die Rückkehradresse aus dem Unterprogramm ist der Befehl, der dem ON...GOSUB.. folgt, also auch der Befehl, der ausgeführt wird, wenn gA gleich 0 ist oder zu wenig Zeilennummern angegeben wurden. Man sollte also den Wert von gA prüfen, bevor man einen ON...GOSUB.. ausführen läßt. Vorsicht! Dem BASIC-Neuling gelingt es häufig, bei ON..GOSUB.. ein »Memory full« zu erzeugen! Die Begründung sehen Sie bitte bei GOSUB nach. Programmbeispiele sehen Sie bei JOY und vielen anderen Befehlen.

ON BREAK GOSUB znr

Befehl: Verzweigt das Programm zur angegebenen Zeilennummer, wenn der Computer zweimal das Zeichen CHR$(252) von der Tastatur erhält; das heißt normalerweise, daß zweimal ESC gedrückt wurde.

Achtung! ON BREAK GOSUB ist die einfachste Möglichkeit, ein Programm aufzuhängen, also den Computer soweit zu bringen, daß nur noch Abschalten hilft. Deshalb vergewissern Sie sich, daß nichts wichtiges im Speicher ist, bevor Sie das nächste Programm abtippen!

```
10 ON BREAK GOSUB 100
20 PRINT"Schneider";
30 GOTO 20
100 RETURN
```

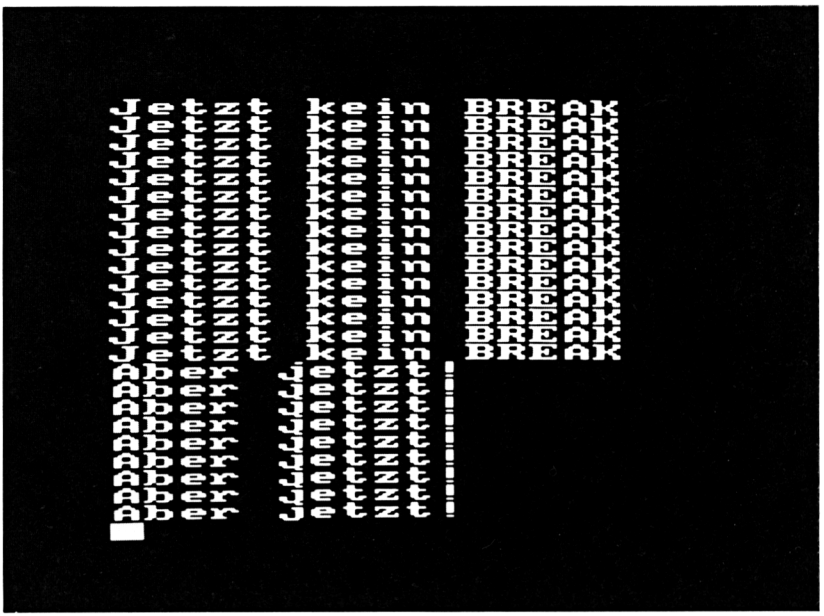
Erst, wenn Sie ein Programm ein END programmiert haben oder es mit der letzten Zeile aufhört, sollten Sie ON BREAK benutzen. Keiner arbeitet gerne vergeblich. Wenn Sie nur einen bestimmten Programmteil mit ON BREAK GOSUB schützen wollen, können Sie danach mit ON BREAK STOP der ESC-Taste die normale Funktion wiedergeben.

ON BREAK STOP

Befehl: Sorgt dafür, daß ein Programm mit ESC wieder unterbrochen werden kann, wenn vorher diese Funktion unterbrochen wurde. Eine Veränderung der Tastatur durch KEY DEF wird nicht aufgehoben.

Ohne vorherigen ON BREAK GOSUB hat ON BREAK STOP keine Funktion.

```
10 ON BREAK GOSUB 100
20 FOR i=1 TO 100
30 PRINT"Jetzt kein BREAK"
40 NEXT
50 ON BREAK STOP
60 FOR i=1 TO 100
70 PRINT"Aber jetzt!"
80 NEXT
90 GOTO 10
100 PRINT"Aetsch..."
110 FOR j=1 TO 500:NEXT
120 RETURN
```



ON ERROR GOTO znr

Befehl: Verzweigt zu der Zeilennummer nach GOTO, sobald der Compiler einen Fehler entdeckt oder der Programmierer durch ERROR einen Fehler hervorruft. Es ist gefährlich, eine ON ERROR-ROUTINE zu früh einzubauen – es sei denn, daß Sie in der ERROR-Routine eine Meldung ausgeben, die Ihnen die fehlerhafte Zeile und die Art des Fehlers zeigt. Ein Beispielprogramm finden Sie bei ERROR.

ON SQ GOSUB znr

Befehl: Wird bei der Soundausgabe gebraucht. Eine Beschreibung finden Sie im Kapitel über die Musik.

OPENIN sA

Befehl: Öffnet eine sequentielle Datei mit dem Namen sA auf der Kassette zum Lesen. Ist der Stringausdruck länger als 16 Zeichen, werden nur die ersten 16 Zeichen als Datei-

name gelesen. Beginnt er mit einem Ausrufezeichen, so werden Meldungen des Betriebssystems beim Laden der Datei unterdrückt. Das ist bei Textverarbeitung und anderen Programmen unbedingt notwendig, wenn man ohne WINDOWs arbeitet, weil sonst nach jedem Block eine Meldung irgendwo auf dem Bildschirm erscheint: »Loading NAME block n«. Versuchen Sie folgendes Programm.

```
10 OPENOUT"test2"
20 FOR i=1 TO 500
30 PRINT#9,i
40 NEXT
50 CLOSEOUT
60 OPENIN"test2"
70 INPUT#9,a
80 PRINT a
90 IF NOT EOF THEN GOTO 70
100 CLOSEIN
```

Ändern Sie jetzt die Zeile 70 in:

```
70 OPENIN"!test2"
```

und Sie stellen fest, daß die Druckausgabe nicht mehr durch Meldungen unterbrochen wird.

OPENOUT sA

Befehl: Öffnet eine sequentielle Datei mit dem Namen sA auf der Kassette zum Schreiben. Beginnt der Stringausdruck mit einem Ausrufezeichen, so werden Meldungen des Betriebssystems beim Laden der Datei unterdrückt. Sie haben so die Möglichkeit, Ihre eigenen Meldungen zu formulieren. Ein '!' bei OPENOUT unterdrückt nicht die Meldungen von OPENIN.

```
10 INPUT"Bitte Dateinamen eingeben (max.
 16 Zeichen.)";na$
20 IF LEN(na$)>16 THEN PRINT CHR$(11);:G
OTO 10
30 PRINT "Bitte druecken Sie REC und PLA
Y und danach ENTER"
40 IF INKEY$<>"" THEN GOTO 50
50 IF INKEY$<>CHR$(13) THEN GOTO 50
60 PRINT"Ihre Datei wird gespeichert!"
```

```
70 OPENOUT"!"+na$
80 FOR i=1 TO 10
90 PRINT#9,"Schneider CPC 464"
100 NEXT i
110 CLOSEOUT
120 PRINT"Fertig....."
```

ausdruck1 **OR** ausdruck2 ⟨**OR** ausdruck3....⟩

Logischer Operator: Sobald einer der durch OR verbundenen Ausdrücke einen Wert verschieden von Null hat, liefert die Verknüpfung OR einen von Null verschiedenen Wert und wird so in einer IF-Abfrage als wahr erkannt.

Die Auswertung der Ausdrücke darf die Grenzen, die für Ganzzahloperanden gelten, nicht überschreiten.

Verwendungsmöglichkeiten: Eine OR Verknüpfung wird in zwei Zusammenhängen häufig gebraucht: Sobald eine von mehreren Bedingungen reicht, um eine Aktion herbei zu führen, so wird OR benützt.

```
10 INPUT"Scheint draussen die Sonne";a$
20 a$=UPPER$(a$)
30 INPUT"Hast Du einen Regenschirm";b$
40 b$=UPPER$(b$)
50 IF a$="JA" OR b$="JA" THEN PRINT"Waru
m gehst Du nicht spazieren?":GOTO 70
60 PRINT"Bei dem Wetter kann Dir nur noc
h dein Computer helfen"
70 END
```

Die zweite Art, OR zu gebrauchen ergibt sich bei Bitmanipulationen: Mit OR ist man in der Lage, ein einzelnes Bit eines Bytes auf 1 zu setzen und alle anderen unverändert zu lassen.

```
10 MODE 2
20 POKE 160,0
30 PRINT"Das sind die Bits von Byte Nr.
160:  ";BIN$(PEEK(160),8);PEEK(160)
40 PRINT"                          Bit Nr:
      76543210"
50 INPUT"Welches wollen Sie auf 1 setzen
 (0-7)";bi
60 IF bi<0 OR bi>7 THEN GOTO 10
```

```
70 POKE 160,PEEK(160)OR 2^bi
80 PRINT"Und so sieht Byte Nr. 160 jetzt
 aus: ";BIN$(PEEK(160),8);PEEK(160)
90 GOTO 40
```

ORIGIN gA1,gA2(,gA3,gA4,gA5,gA6)

Graphikbefehl: Legt den Kreuzungspunkt der X- und Y-Koordinaten für Graphikanweisungen fest. Gleichzeitig bietet dieser Befehl die Option, ein Fenster (WINDOW) für Graphik zu definieren. Alle Parameter beziehen sich auf die ursprünglichen Nullpositionen von X und Y, die beide in der linken unteren Ecke des Bildschirms liegen. Die Parameter haben folgende Bedeutungen:

gA1: X-Nullpunkt (Waagerechte Angabe)
gA2: Y-Nullpunkt (Senkrechte Angabe)
gA3: Linker Rand des Graphikfensters
gA4: Rechter Rand des Graphikfensters
gA5: Oberer Rand des Graphikfensters
gA6: Unterer Rand des Graphikfensters

Das Graphikfenster umfaßt normalerweise den gesamten Bildschirm. Definiert man es mit ORIGIN um, so wird nur noch innerhalb dieses Fensters gezeichnet. Alle Zeichenvorgänge außerhalb werden unterdrückt. Da der Schneider aber die Zulässigkeit der Koordinaten überprüfen muß, wird das Zeichnen wesentlich langsamer. Zusätzlich muß man beachten, daß die X-Koordinaten, also der rechte und linke Rand, immer auf den Beginn eine Bildschirmbytes gesetzt werden. Das heißt im Klartext, daß Sie für den Rand nicht 640 Möglichkeiten, sondern nur 80 haben. Graphikfenster schmaler als 8 Punkte sind also nicht möglich.

Verwendungsmöglichkeiten: Da ORIGIN erlaubt, den Bezugspunkt einer Graphik selbst zu wählen, kann man eine feststehende Figur einmal (bezogen auf 0) entwerfen und sie dann überall auf dem Bildschirm zeichnen, indem man den Nullpunkt verlegt.

```
10 e=INT(RND*100)+10
20 CLS
30 DEG
40 ORIGIN 320,200
50 PLOT 0,100
60 FOR i=0 TO 360 STEP e
70 DRAW 100*SIN(i),100*COS(i)
80 NEXT
```

```
90 FOR i=0 TO 360 STEP e
100 ORIGIN 320+100*SIN(i),200+100*COS(i)
110 PLOT 0,100
120 FOR j=0 TO 360 STEP e
130 DRAW 100*SIN(j),100*COS(j)
140 NEXT
150 NEXT
160 FOR i=1 TO 1000
170 NEXT
180 GOTO 10
```

OUT gA1,gA2

Befehl: Sendet an das Interface mit der Adresse gA1 den Wert gA2 (max. 255). Mit diesem Befehl können Sie Aufgaben des Betriebssystems selbst programmieren. OUT &c200,255 schaltet den Motor des Kassettenrecorders ein, OUT &c300,255 schaltet ihn wieder ab.

PAPER ⟨#gA1,⟩gA2

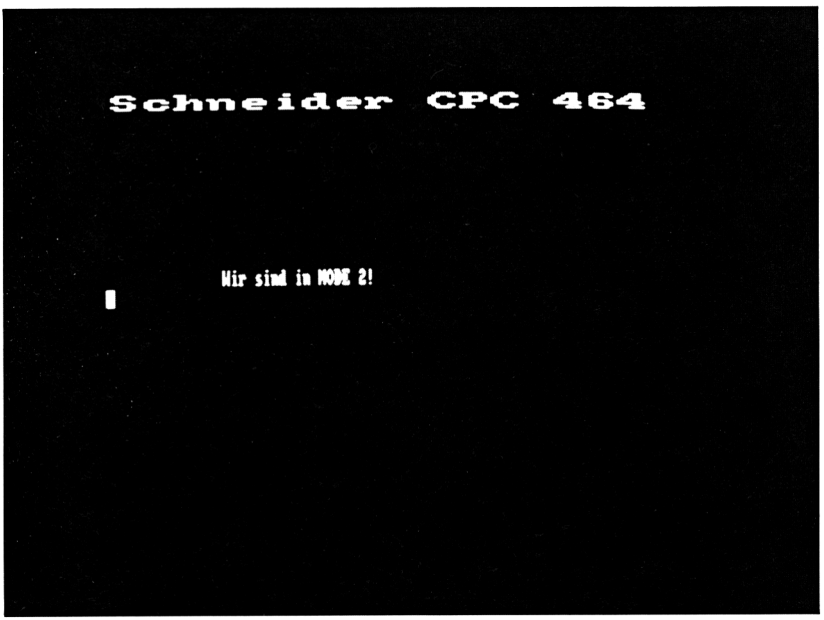
Befehl: Setzt die Hintergrundfarbe für PRINT-Befehle fest. Ein nachfolgendes CLS löscht den gesamten Ausgabebereich, der durch gA1 festgelegt ist, mit dieser Farbe. Um nur einen bestimmten Schriftzug zur Betonung einzufärben, muß man diesen Schriftzug mit Semikolon (;) abschließen und mit PAPER 0 wieder auf die normale Bildschirmfarbe zurückkehren. Abhängig vom Bildschirmmodus haben Sie 16, 4 oder 2 Farben zur Verfügung. Die Farbnummern, die nicht in Ihrem aktuellen Modus zur Verfügung stehen, werden bei MODE 1 mit Farbstiftnummer MOD 4 ersetzt und bei MODE 2 mit dem Farbstift MOD 2.

```
10 ON BREAK GOSUB 190
20 MODE 0
30 FOR i=0 TO 7
40 INK i,i
50 INK 15-i,26-i
60 NEXT i
70 CLS
80 i=INT(RND*15)+1
90 x=INT(RND*9)+1
100 y=INT(RND*23)+1
```

```
110 PAPER i:PEN 15-i
120 LOCATE x,y
130 PRINT"              ";
140 LOCATE x,y+1
150 PRINT" Schneider ";
160 LOCATE x,y+2
170 PRINT"              ";
180 GOTO 80
190 INK 1,24
200 END
```

PEEK(gA)

Ganzzahlfunktion: Liefert den Inhalt der Speicherstelle gA. Das Ergebnis ist immer eine Zahl zwischen 0 und 255. Ohne Umwege ist es nicht möglich, den Inhalt von Speicherstellen aus dem ROM auszulesen. Dafür kann man PEEK sehr gut dazu benutzen, um verschiedene Schriftgrößen vom Bildschirm in eine Tabelle zu lesen und in einem anderen Modus wieder darzustellen.



```
10 DEFINT a-z
20 DIM a(80,8)
30 MODE 0
40 PRINT"Schneider CPC 464"
50 FOR i=0 TO 7
60 FOR j=0 TO 79
70 b=&C000+i*&800+j
80 c=PEEK(b)
90 d=((c AND 128)/128)*240
100 e=((c AND 64)/64)*15
110 a(j,i)=d+e
120 NEXT
130 NEXT
140 MODE 2
150 FOR i=0 TO 7
160 FOR j=0 TO 79
170 POKE &C000+i*&800+j,a(j,i)
180 NEXT
190 NEXT
200 PRINT:PRINT
```

PEN ⟨#gA1,⟩gA2

Befehl: Setzt die Schreibfarbe für PRINT-Anweisungen fest. gA2 ist die Farbe, die durch ein vorheriges INK-Kommando zugeordnet wurde. Die Beschränkungen, denen gA2 unterliegt, sowie ein Beispielprogramm finden Sie bei PAPER. Wenn die Werte für gA1 7 überschreiten, erhalten Sie ein »Improper argument«.

PI

Systemkonstante: Liefert das Verhältnis von Umfang und Durchmesser eines beliebigen Kreises. Da der Schneider von sich aus alle Winkelfunktionen im Bogenmaß errechnet, kann es beim Errechnen von Winkelfunktionen Unterschiede geben, wie das folgende Programm beweist. Diese Unterschiede liegen aber in den Grenzen, die man von BASIC gewöhnt ist.

```
5 DEFINT i
10 FOR i=0 TO 360
```

```
20 DEG
30 a=SIN(i)
40 RAD
50 b=SIN(i*((2*PI)/360))
60 IF b<>a THEN PRINT i;TAB(10);a;TAB(25
);b;TAB(50);a-b
70 NEXT
```

PLOT gA1,gA2⟨,gA3⟩

Befehl: Setzt einen Punkt an die durch gA1-bestimmte X-Koordinate und die durch gA2 bestimmte Y-Koordinate, bei Verwendung von gA3 in der durch gA3 bestimmten INK-Farbe. Ist keine Farbe angegeben, wird INK 1 verwendet.

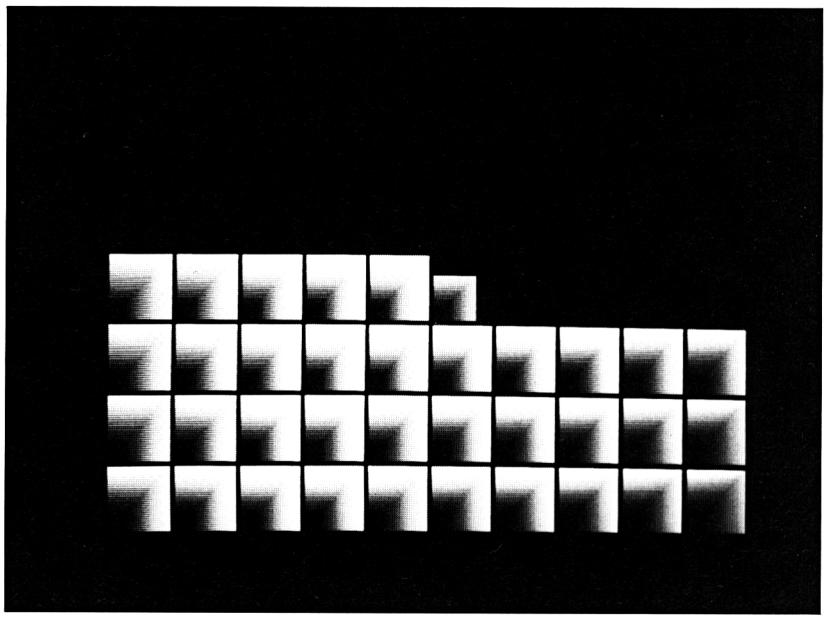
```
10 MODE 2
20 ORIGIN 0,200
30 f=-1
40 f1=ABS(f)
50 FOR i=1 TO 600
60 a=RND*200
70 PLOT i,-a,f1
80 DRAW i,a
90 NEXT
100 f=NOT f
110 GOTO 40
```

PLOTR gA1,gA2⟨,gA3⟩

Befehl: Setzt einen Punkt an eine Position, deren X- und Y-Position sich aus der letzten X-Position des Graphik-Cursors plus gA1 und der letzten Y-Position plus gA2 errechnet. Für die Farbangabe gilt das gleiche wie bei PLOT.

Verwendungsmöglichkeiten: Ergeben sich sowohl in der graphischen Gestaltung wie bei der Auswertung von Daten.

```
10 DATA 10,12,15,13,16,20,23,17,15,18
20 DATA 11,13,16,14,17,21,24,18,16,19
30 DATA 12,14,17,15
40 ORIGIN 0,0
```



```
50 MOVE 0,0
60 MODE 1
70 DIM a(24)
80 b=100
90 FOR i=1 TO 24
100 READ a(i)
110 a=MAX(a,a(i))
120 NEXT
130 a1=400/a
140 x1=300/24
150 FOR i=0 TO 299 STEP x1
160 z=z+1
170 DRAW i,a(z)*a1,1
180 b=b+a(z)
190 MOVE i,0
200 PLOTR 0,b/2,2
210 MOVE i,a(z)*a1
220 NEXT
```

POKE gA1,gA2

Befehl: Füllt die durch gA1 angegebene Speicherstelle mit dem Wert von gA2. Da ein Byte maximal einen Wert von 255 haben kann, wird gA2 mit »Improper argument« abgewiesen, sobald es größer ist. POKE bietet bei vielen Gelegenheiten die einzige Möglichkeit, etwas zu erreichen. Das folgende Programm gibt Ihnen die Ein-Byte-Verschlüsselungen der BASIC-Befehle heraus (die Tokens). Achten Sie beim Abtippen darauf, die Zeile 10 ohne Fehler abzuschreiben. Starten Sie es einige Male hintereinander. Die zweite Version liefert Ihnen die Tokens der Funktionen. Bevor Sie es laufen lassen, geben Sie im Direktmodus ein:

```
POKE &177,0

10 'AUTO
20 POKE &176,PEEK(&176)+1
30 PRINT PEEK(&176)
40 LIST

10 'AA
20 POKE &176,255
30 POKE &177,PEEK(&177)+1
40 PRINT PEEK(&177)
50 LIST
```

POS (#gA)

Ganzzahlfunktion: Liefert die augenblickliche X-Position des Cursors, bezogen auf den linken Rand der jeweiligen durch gA bestimmten Ausgabeeinheit – also Window oder Drucker.

Verwendungsmöglichkeiten: POS wird benützt, wenn man Daten in bestimmten Formaten ausdrucken will. Im folgenden Programm sorgt POS dafür, daß die Ausgabe eine Spalte des Bildschirms nicht überschreitet.

```
10 FOR i=1 TO 100
20 PRINT USING"####";i;
30 IF POS(#0)>20 THEN PRINT
40 NEXT
```

PRINT

Befehl: Gibt die nach PRINT folgenden Daten aus. Da PRINT, ähnlich wie INPUT, ein sehr vielseitiger Befehl ist, machen wir uns mit den Möglichkeiten von PRINT schrittweise vertraut.

PRINT

PRINT alleine sorgt dafür, daß der Cursor auf den Beginn der nächsten Zeile gesetzt wird. Wenn er vorher am Beginn einer Zeile stand, wird also eine Leerzeile ausgedruckt.

PRINT Druckliste

Eine Druckliste besteht aus Konstanten wie »Computer« oder -12 und/oder Variablen, wie a$ oder wert. Diese Teile können wahllos durcheinander gedruckt werden, solange sie in der Druckliste irgendwie getrennt sind. Als gültige Trennzeichen werden Leerstelle, Semikolon und Komma interpretiert (das ist auch der Grund dafür, daß Variablennamen keine Leerstellen enthalten dürfen). Die Druckliste kann beliebig lang sein. Bei der Ausführung wird von links beginnend jeder Teil der Druckliste hintereinander ausgedruckt, bis der nächste Teil nicht mehr in die aktuelle Zeile passen würden. Danach wird in die nächste Zeile gesprungen und dort weiter gedruckt. Nur wenn ein Teil der Druckliste länger als eine Zeile ist, wird mitten in diesem Teil auf die nächste Zeile gesprungen.

```
10 CLS

20 WINDOW 1,5,1,15

30 PRINT"Schneider Computer"

40 WINDOW 1,80,1,25
```

PRINT Druckliste Trennzeichen

Es gibt drei verschiedene Trennzeichen für den PRINT-Befehl. Diese Zeichen haben außerdem teilweise noch eine verschiedene Wirkung, je nachdem ob sie am Ende oder in der Mitte der Druckliste stehen. Beginnen wir mit der Leerstelle.

```
10 PRINT "Schneider"

20 GOTO 10
```

In der Zeile sorgt sie nur dafür, daß zwei Informationen von einander getrennt werden, die sonst nicht unterschieden werden könnten. Der Computer »sieht« an der Variablenkennzeichnung, daß eine Information zuende ist. Genauso merkt er, daß die zweiten Anführungszeichen das Ende einer Zeichenkette bedeuten. Das einzige, womit er Schwierigkeiten hat, ist die Unterscheidung von Variablen und Zahlenkonstanten: Diese müssen durch das Semikolon getrennt werden. Am Ende einer Druckliste wird automatisch in die nächste Zeile gesprungen – es sei denn, daß ein Komma oder Semikolon dort steht.

```
10 PRINT "Schneider";

20 GOTO 10
```

Das Semikolon sorgt in der Zeile dafür, daß die Teile der Druckliste voneinander getrennt werden. Am Ende der Zeile hält es den Cursor fest, d. h. der nächste PRINT-Befehl wird genau an dieser Stelle ausgeführt. Das Komma in der Druckliste und am Ende der Zeile läßt den Cursor an die nächste Tabulatorstelle springen. Wo die ist, kann man durch ZONE feststellen.

```
10 PRINT "Schneider",

20 GOTO 10
```

PRINT #gA, Druckliste Trennzeichen

Wenn man hinter PRINT mit #gA eine Ausgabeeinheit festsetzt, geht der Druckvorgang auf die damit bezeichnete Einheit. Hat gA Werte zwischen 0 und 7, so sind damit Fenster des Bildschirms gemeint. Diese verhalten sich genau wie kleine Bildschirme. Trennzeichen und Zeilenumbruch folgen den gleichen Gesetzen. Genauso ist es beim Ausgabegerät #8, dem Drucker. Mit WIDTH definieren Sie die Breite des »Fensters« – alles andere macht Ihr Schneider wie nach Maß. Etwas problematischer wird es beim Kassettenrecorder. Da jede Information, die Sie auf Kassette schreiben, auch wieder in den Computer gelesen werden muß, und Sie dafür nur den INPUT-Befehl haben, müssen Sie die PRINT-Befehle so formulieren, daß sie bei einem späteren Einlesen auch als das gelesen werden, als was sie auf das Band geschrieben wurden. Als erstes: Die Trennzeichen haben eine andere Bedeutung. Alles das, was durch das Semikolon bei dem PRINT#8 getrennt wurde, wird beim Einlesen verknüpft. Versuchen Sie dieses Beispiel:

```
10 OPENOUT"test"
20 FOR i=1 TO 100
30 PRINT#9,STR$(i);
```

```
40 NEXT
50 CLOSEOUT
60 PRINT"Vergessen Sie nicht, zurueckzus
pulen!"
70 OPENIN"Test"
80 INPUT#9,a$
90 PRINT a$
100 IF EOF THEN CLOSEIN:END
110 GOTO 80
```

Sie merken beim Einlesen, daß der Schneider das Trennungszeichen erst dann als solches interpretiert, wenn die maximale Länge des Strings erreicht ist. Ändern Sie die Zeile 80 und 90 in:

```
80 INPUT#9,a

90 PRINT a
```

An der Ausgabe sehen Sie, daß der Computer keinen Unterschied macht, ob Sie Zahlen als Zahlen oder als die entsprechende Zeichenkette aufzeichnen. Nur beim Einlesen ist es ein Unterschied. Das Leerzeichen gilt bei Strings nicht als Trennungszeichen, sondern nur bei Zahlen. Gültige Trennungszeichen sind bei Strings nur das Komma (und auch das nur, wenn es in Anführungszeichen steht) und die ENTER-Taste. Sie können diese ENTER-Taste mit der Funktion CHR$(13) einfügen oder die Arbeit Ihrem Schneider überlassen, indem Sie einen neuen PRINT-Befehl für die nächste Ausgabe wählen, denn Sie wissen ja: Am Ende eines PRINT-Befehls geht der Cursor in die nächste Zeile und das erreicht er durch ein CHR$(13) + CHR$(10).

PRINT ⟨#gA⟩SPC(gA1);druckliste

Setzt vor dem Ausdrucken der Druckliste den Cursor um soviele Stellen weiter, wie gA1 angibt. Falls der Zwischenraum beschrieben ist, wird er mit Leerstellen überschrieben.

```
10 MODE 2

20 PRINT"AAAAAAAAAAAAAAAAAAAAAAAAA"

40 PRINT SPC(10);"BBBBB"
```

Wenn Sie dieses Beispiel versucht haben, fügen Sie eine Zeile 30 hinzu:

```
30 LOCATE 1,1
```

PRINT ⟨#gA,⟩**TAB**(gA1); druckliste

Druckt die Druckliste ab Position gA1 aus; gezählt wird beginnend beim linken Rand der Ausgabeeinheit. Ist diese Position schon überschritten, so wird vor dem Ausdruck ein Zeilenvorschub ausgelöst.

PRINT ⟨# gA⟩,**USING** formatstring; ausdruck ⟨,ausdruck,ausdruck,...⟩

Druckt auf die durch gA bestimmte Ausgabeeinheit die Ausdrücke entsprechend den Festsetzungen im Formatstring aus. Zeichen innerhalb des Formatstrings bestimmen die Position und die Formatierung der Ausdrücke innerhalb der auszugebenden Druckzeile.

Leider kann man USING nicht benutzen, um eine Liste von formatierten Ausdrücken einer Stringvariablen zuzuordnen; ausschließlich die Benutzung im Zusammenhang mit PRINT ist erlaubt. Zur Demonstration der Möglichkeiten des Formatstrings diese Tabelle:

Ausdruck	Formatstring	Druckergebnis	Kommentar
		0123456789	
1	####	1	Gibt Länge des auszugebenden Strings incl. Leerstellen an.
1.2	####	1	
1.7	####	2	
−1.7	####	−2	
1	#.##	1.00	Bestimmt die Lage des Dezimalpunktes und rundet kaufmännisch.
1.2	#.##	1.20	
−1	#.##	%−1.00	
1	##.#	1.0	
−1.95	##.#	−2.0	
1	+###	+1	Gibt vor allen Zahlen Vorzeichen mit aus. Gibt nach allen Zahlen Vorzeichen aus.
−10	+###	−10	
10	###+	10+	
−10	###+	10−	
10	−###	−10 (!)	Nicht benutzen! Negative Zahlen immer mit Vorzeichen. Gibt das Vorzeichen negativer Zahlen danach aus.
−10	−###	−−10 (!)	
10	###−	10	
−10	###−	10−	

10	$####	$ 10	Dollarzeichen immer an der gleichen Stelle.
10	$$###	$10 (!)	Dollarzeichen nach dem Vorzeichen, aber vor der Zahl.
1000	$$###	$1000	
-10	$$###	-$10	
10	*####	* 10	
10	**###	***10	Statt führender Leerstelle Sternchen.
1	****#	*1* (!)	Unvorhersehbare Ergebnisse, wenn Zahl > 9!
10	****#	10*	
100000	,#######	, 100000	Ein Komma zwischen erstem # und '.' erzeugt Dreiergruppen.
100000	#.######	100,000	
100000	######,#	100.000	
1.1234	#,#,####	1.1,	
100000	#^^^^	1E+05	Eponentialschreibweise; richtet sich nach Vorzeichen und Vorkommastellen.
100000	###^^^^	10E+04 (!)	
0.00001	###^^^^	10E-06 !)	
abcdefghijkl	!	a	Das erste Zeichen des Strings wird gedruckt.
abcdefghijkl	\ \	ab	Es werden soviel Zeichen ausgedruckt, wie Leerstellen zwischen den beiden \ vorhanden sind plus 2 weitere Zeichen. Überzählige Zeichen werden unterdrückt, fehlende durch Leerzeichen ersetzt.
abcdefghijkl	\ \	abcd	
abcdefghijkl	&	abcdefghijkl	Der gesamte Stringausdruck wird ausgegeben; alle weiteren Positionen richten sich nach der Länge des Ausdrucks.

Da der Schneider durch die Angabe der Ausgabeeinheit die gleiche Ausgabe sowohl auf den Bildschirm wie auf den Drucker schicken kann, sollten Sie Ihre mit PRINT USING formatierte Ausgabe erst prüfen, bevor Sie Druckerpapier verschwenden.
Achtung: nicht in Formatstring verwenden: CHR$ (31) und CHR$ (92). Ergebnisse nicht vorhersehbar.

RAD

Befehl: Schaltet die Berechnung von Winkeln und Winkelfunktionen auf Bogenmaß um.

Da in der Mathematik heute meistens im Bogenmaß gerechnet wird, ist der Schneider beim Einschalten im Bogenmaßmodus. In diesem Modus entspricht ein Winkel von 180 Grad einem Bogen von PI, ist also der Sinus von 180 Grad derselbe Wert wie der Sinus von PI.

RANDOMIZE ⟨gkA⟩

Befehl: Ausgehend von Null oder dem Wert von gkA werden die Zufallszahlen bei der nächsten RND-Funktion berechnet.

Negative Werte als Argument erzeugen die gleichen Zufallszahlen wie ihre positiven Entsprechungen. Je näher zwei Zahlen bei einander liegen, desto schneller werden die Zufallszahlen auf die gleiche Zahlenfolge kommen. Gleiche gkA erzeugen gleiche Zahlenfolgen. Um auf wirklich jedesmal neue Zahlenfolgen zu kommen, benutzen Sie am besten den RANDOMIZE-Befehl mit der Systemvariablen TIME und das jedesmal, bevor Sie eine Zufallszahl erzeugen.

```
10 RANDOMIZE TIME

20 a=RND*6
```

Achtung! Nur das Vorzeichen des im Handbuch in Klammern angegebenen Arguments hat eine Funktion! Versuchen Sie folgendes Programm:

```
10 RANDOMIZE

20 PRINT RND(6)

30 RANDOMIZE

40 PRINT RND(100)
```

Auf Ihrem Bildschirm steht zweimal die gleiche Zahl.

READ var1 ⟨,var2, var3,...⟩

Befehl: Füllt die nach READ angegebenen Variablen mit den nächsten Konstanten in den DATA Zeilen. Der Lesezeiger wird auf den Beginn der nächsten, noch nicht gelesenen Konstanten gesetzt.

Wenn die Zuweisung zu einem »Type mismatch« führen würde, wird das Programm abgebrochen – leider mit einem »Syntax error in ...« der DATA Zeile, in gerade gelesen wurde. Es kann ziemlich schwierig sein, die dazugehörige READ-Anweisung zu finden. Deshalb ein Tip: Falls Sie in einem Programm mehrere READ-Anweisungen haben, bei denen numerische und Stringdaten gemischt sind, testen Sie die DATA-Zeilen einzeln durch, bevor Sie die nächste READ-Anweisung verwenden.

Verwendungsmöglichkeiten: Siehe DATA.

REM

Befehl: Deklariert den Rest der Zeile als Kommentar. Der Text danach oder nach einem Apostroph (') erscheint ausschließlich beim Listen des Programmes.

Verwendungsmöglichkeiten: die REM-Zeile hat in der Praxis in drei Zusammenhängen Bedeutung erlangt.

1. Aufnahme von Kommentaren für den Programmierer und Copyright-Vermerke. Da es schwierig genug ist, sich während der Erstellung eines Programmes in diesem zurecht zu finden, benötigt man nach Monaten, wenn man dieses Programm verändern will, eine Gedankenstütze, die zeigt, was man sich damals bei der Erstellung gedacht hat. Kurze Kommentare in REM-Zeilen helfen da weiter.

2. Testhilfe beim Programmieren. Oft kommt es vor, daß der Programmierer nicht recht weiß, ob eine Zeile richtig oder falsch ist. Um sich die Arbeit zu ersparen, die alte Version einer Zeile nach einer Änderung doch noch einmal eintippen zu müssen, versieht man sie am Beginn mit REM und läßt sie für den Computer nicht verarbeitbar stehen. Ist die neue Version besser – na gut – raus damit; wenn nicht – auch gut – dann wird halt die neue Zeile gelöscht und aus der alten das REM entfernt.

3. (Nur empfehlenswert für den erfahrenen Programmierer) Da eine Programmzeile beim Schneider inclusive Zeilennummer 255 Zeichen lang sein darf, eignet sich die REM-Zeile besonders gut dafür, kurze Maschinenprogramme aufzunehmen, die man später mit CALL aufruft. Voraussetzung dafür ist eine genaue Kenntnis über die Lage der Zeile im Speicher. Meiner Ansicht nach sind die ersten Zeilen dafür prädestiniert, weil man deren Lage am besten überprüfen kann.

REMAIN gA

Ganzzahlfunktion: Liefert die Zeit, die dem durch gA bestimmten Zeitgeber bis zum nächsten Aufruf verbleibt und schaltet ihn und alle anderen mit einer höheren Nummer ab. Sofern der Zeitgeber nicht durch EVERY oder AFTER eingeschaltet war, liefert REMAIN Null.

Verwendungsmöglichkeiten: Da es unmöglich ist, den Wert eines Zeitgebers abzufragen, ohne ihn zurückzusetzen, kann REMAIN nicht in Vergleichen verwendet werden. Die zweite Arbeit, die REMAIN verrichtet, bietet bessere Möglichkeiten.

```
10 PRINT"Bevor Sie Ihre Eingabe machen,
 bitte ich um etwas Pause."
20 PRINT"Bitte eine Zahl von 0-9"
30 EVERY 100 GOSUB 90
40 a$=INKEY$:IF a$="" THEN 40
50 re=REMAIN(0)
60 IF a$<"0" OR a$>"9" THEN GOTO 30
70 PRINT"Na also, es geht ja!"
80 END
90 PRINT"Wie lange soll ich noch warten?
"
100 RETURN
```

RENUM znr1⟨,znr2⟨,schrittweise⟩⟩

Systembefehl: Numeriert, beginnend bei znr2, das Programm um. Dabei werden die neuen Zeilennummern so verteilt, daß die alte znr2 nur neuen znr1 wird. Die nächste Zeilennummer ist um die Schrittweite größer usw.

Falls die Zeilennummer in einem Sprungbefehl nicht vorhanden ist, erhalten Sie nach dem Umnumerieren eine Fehlermeldung: Undefined line 600 in ... Ab da ist Vorsicht geboten. Wenn nämlich die unbekannte Zeile innerhalb des neuen Zeilenbereiches liegt, so wird dieser Fehler beim nächsten RENUM nicht mehr gemeldet. Numerieren Sie das folgende Nonsensprogramm zweimal um:

```
10 GOTO 100

20 GOTO 200

30 GOTO 300

40 GOTO 400
```

Benützen Sie zuerst den Befehl RENUM 100,10,100. Sie werden für jede Zeile eine Meldung erhalten. Danach tippen Sie RENUM ohne Argumente. Damit erreichen Sie das gleiche, als ob Sie eintippten RENUM 10,1,10. Eine Fehlermeldung wird jetzt nicht ausgegeben, da die alten unbekannten Zeilennummern jetzt im Programm liegen.

RESTORE ⟨znr⟩

Befehl: Setzt den Lesezeiger für die nächste READ-Anweisung auf den Beginn des Programmes oder – wenn znr benutzt wurde – auf die entsprechende Zeilennummer. Eine nicht vorhandene Zeilennummer wird mit »Line does not exist in ...« abgewiesen.

Verwendungsmöglichkeiten: Der RESTORE-Befehl ist vor allen in seiner zweiten Form ein nützliches Werkzeug beim Umgang mit DATA-Zeilen. Baut man mit DATA und READ eine kleine Dateiverwaltung auf, so hat man oft Informationen, die nur am Beginn des Programmes eingelesen werden müssen. Die Daten in den restlichen DATA-Zeilen allerdings müssen jedesmal beim Suchen, Drucken und ähnlichen Aufgaben gelesen werden. In anderen BASIC-Versionen muß man oft eine Routine schreiben, die dafür sorgt, daß die nicht benötigten DATAs nach jedem RESTORE überlesen werden; beim Schneider ist das glücklicherweise besser geregelt.

```
10 DATA Vorname
20 DATA Name
30 DATA Strasse
40 DATA PLZ
50 DATA Ort
60 DATA Telefon
70 DATA Adam,Aragon,Eifelstr.18,5000,Koe
ln 1,0221/12345
80 DATA Berta,Berend,Bahnweg 7,2000,Hamb
urg 71,040/9876543
90 DATA Caesar,Campino,Ginsterallee 15,4
000,Duesseldorf,0211/555555
900 DATA x,x,x,x,x,x
910 DIM fe$(6),i$(6)
920 FOR i=1 TO 6:READ fe$(i):NEXT i
1000 'Hauptmenue
1010 CLS
1020 PRINT"  Hauptmenue"
1030 PRINT:PRINT
1040 PRINT"<1> Suchen"
1050 PRINT"<2> Listen"
1060 PRINT"<3> Ende"
1070 PRINT:PRINT
1080 PRINT"Ihre Wahl";
1090 wahl$=INKEY$:IF wahl$<"1" OR wahl$>
"3" THEN GOTO 1090
1100 RESTORE 70
```

```
1110 ON VAL(wahl$) GOSUB 2000,3000,4000
1120 GOTO 1010
2000 'Suchen
2010 CLS
2020 PRINT"Suchen"
2030 PRINT:PRINT
2040 PRINT"Nach was suchen Sie? ";
2050 PRINT"Bitte waehlen Sie die Zahl"
2060 FOR i=1 TO 6
2070 PRINT i;fe$(i)
2080 NEXT i
2090 a$=INKEY$:IF a$<"1" OR a$>"6" THEN
GOTO 2090
2100 a=VAL(a$):PRINT"...und was suchen S
ie in ";fe$(a);"?"
2110 INPUT su$
2120 FOR i=1 TO 6
2130 READ i$(i)
2140 NEXT i
2150 IF i$(1)="x" THEN PRINT"Mehr wurde
nicht gefunden!":WHILE INKEY$="":WEND:GO
TO 2230
2160 IF i$(a)<>su$ THEN GOTO 2120
2170 FOR i=1 TO 6
2180 PRINT USING"\            \:  &";fe$(
i);i$(i)
2190 NEXT i
2200 PRINT"Weitersuchen j/n"
2210 a$=INKEY$:IF a$<>"j" AND a$<>"n" TH
EN GOTO 2210
2220 IF a$="j" THEN GOTO 2120
2230 RETURN
3000 'Listen
3010 CLS:FOR i=1 TO 6
3020 READ i$(i)
3030 NEXT i
3040 IF i$(1)="x" THEN PRINT"Dateiende e
rreicht!":FOR i=1 TO 800:NEXT:RETURN
3050 FOR i=1 TO 6
3060 PRINT USING"\            \:  &";fe$(
```

```
i);i$(i)
3070 NEXT i
3080 PRINT"Weiterlisten j/n"
3090 a$=INKEY$:IF a$<>"j" AND a$<>"n" TH
EN GOTO 3090
3100 IF a$="j" THEN GOTO 3010
3110 RETURN
4000 'Ende
4010 CLS
4020 PRINT"Sind Sie sicher j/n"
4030 a$=INKEY$:IF a$="" THEN 4030
4040 IF a$="j" THEN CLS:PRINT"Tschuess..
.":END
4050 RETURN
```

RESUME ⟨znr⟩

Befehl: Beendet eine durch ON ERROR GOTO aufgerufene Fehlerbehandlung und verzweigt in die durch znr bestimmte Zeilennummer.

Achtung! Tritt ein Fehler bei abgeschaltetem BREAK auf, so kann RESUME dazu führen, daß der Computer eine Endlosschleife läuft, aus der Sie nur durch Ausschalten oder komplettes RESET wieder herauskommen.

```
10 ON BREAK GOSUB 80
20 ON ERROR GOTO 60
30 prunt"adfg"
40 PRINT"wwwwwwwww"
50 END
60 PRINT ERR,ERL
70 RESUME
80 RETURN
```

RETURN

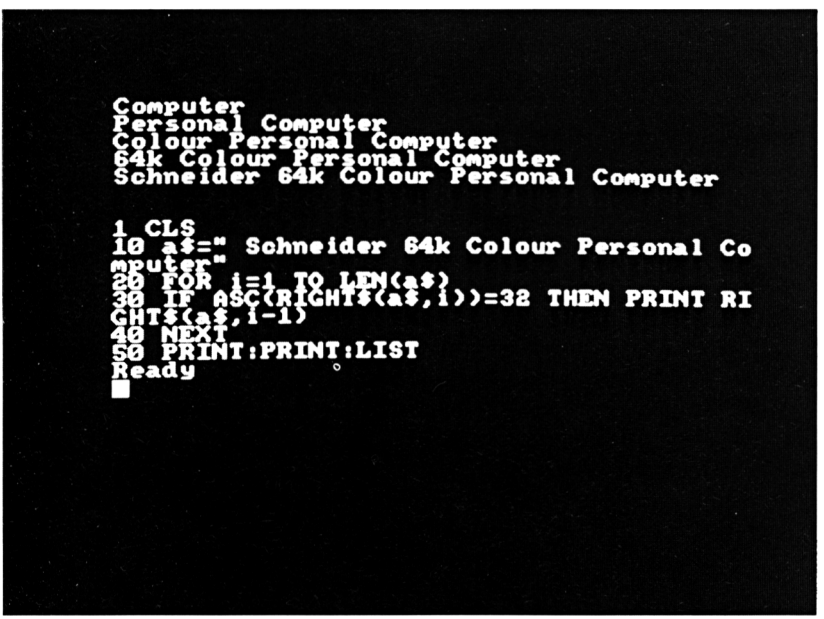
Befehl: Beendet ein durch GOSUB aufgerufenes Unterprogramm und kehrt auf den Befehl zurück, der entweder dem GOSUB folgt oder vor dessen Bearbeitung eine Unterbrechung durch einen Zeitgeber stattfand.

Verwendungsmöglichkeiten: siehe GOSUB.

RIGHT$ (sA, gA)

Stringfunktion: Liefert die letzten Zeichen aus sA. Die Zahl der Zeichen wird durch gA bestimmt. Ist sA kürzer als gA, wird der gesamte Stringausdruck zurückgegeben. gA darf sich im Bereich 0 bis 255 bewegen. Außerhalb dieses Bereiches wird ein »Improper argument« angezeigt.

Verwendungsmöglichkeiten: Sind beschränkt, weil man bei RIGHT$ genau die Länge des Strings wissen muß, um an der richtigen Stelle zum Abtrennen anzusetzen. Diese Länge läßt sich durch die Funktion LEN herausfinden. Das folgende Beispiel sucht nach einem Leerzeichen und gibt ab dem Leerzeichen den Rest des Strings aus.



RND⟨gkA⟩

Gleitkommafunktion: Liefert eine Zufallszahl im Bereich 0 bis 1.

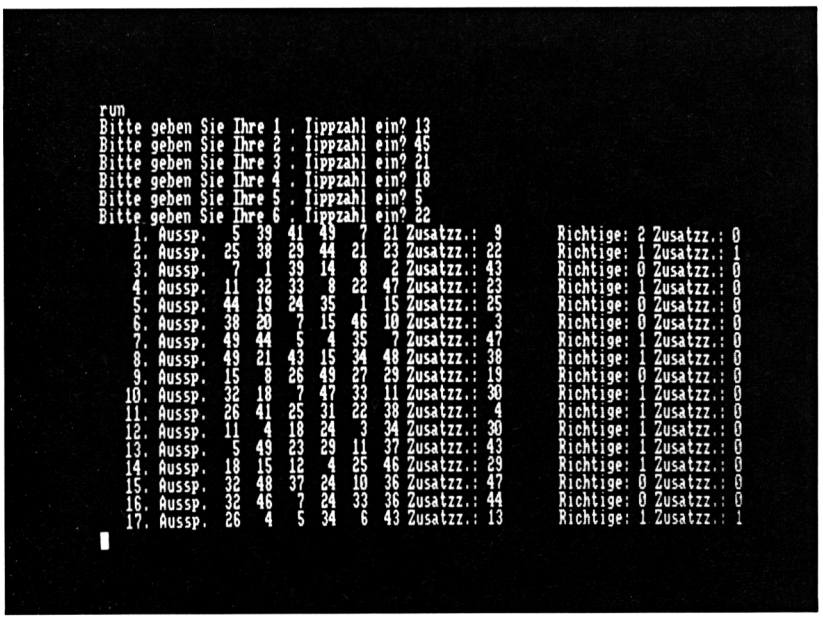
Ein gkA größer als Null liefert eine Folge von Zufallszahlen, die nicht vorhersehbar sind. Wenn gkA gleich Null ist, liefert RND die Zufallszahl, die beim letzten Gebrauch von RND erzeugt wurde. Ist gkA negativ, so wird abhängig von gkA immer die gleiche Zufallszahl erzeugt. Die Auswirkungen machen Sie sich mit folgendem Programm deutlich:

```
10 PRINT:INPUT a
20 IF INKEY$="" THEN PRINT INT(RND(a)*10
);:GOTO 20
30 GOTO 10
```

Verwendungsmöglichkeiten: RND wird nicht nur zur Erzeugung von Zufallszahlen in Spielen, sondern auch in Simulationen benötigt, da man mit RND sehr bequem die Wahrscheinlichkeit eines Ereignisses erzeugen kann. Das nächste Programm macht Ihnen klar, warum Sie noch nicht sechs Richtige im Lotto gewonnen haben, selbst wenn Sie seit zehn Jahren immer die gleichen Zahlen tippen.

```
10 DEFINT a-z
20 DIM tip(7),erg(7)
30 FOR i=1 TO 6
40 PRINT"Bitte geben Sie Ihre"i". Tippza
hl ein";
50 INPUT tip(i)
60 IF INT(tip(i))<>tip(i) OR tip(i)<1 OR
 tip(i)>49 THEN PRINT"Versuchen Sie doch
 einfach mal gueltige Zahlen":GOTO 40
70 FOR j=i-1 TO 1 STEP-1
80 IF tip(i)=tip(j) THEN PRINT"Was denn?
! Die gleiche Zahl zweimal?":GOTO 40
90 NEXT j
100 NEXT i
110 a=1
120 richt=0:zusatz=0
130 FOR i=1 TO 7
140 erg(i)=0:pic(i)=0
150 NEXT i
160 FOR i=1 TO 7
170 erg(i)=INT(RND(TIME)*49)+1
180 FOR j=i-1 TO 1 STEP-1
190 IF erg(i)=erg(j) THEN 170
200 NEXT j
210 pic(i)=erg(i)
220 NEXT i
230 FOR i=1 TO 6
240 FOR j=1 TO 6
250 IF tip(i)=erg(j) THEN richt=richt+1:
erg(j)=0:GOTO 270
```

```
260 NEXT j
270 NEXT i
280 FOR i=1 TO 6
290 IF tip(i)=erg(7) THEN zusatz=1
300 NEXT i
310 PRINT USING"#####. Aussp.  ##  ##  #
#  ##  ##  ## Zusatzz.: ##       Richtig
e: # Zusatzz.: #";a,pic(1),pic(2),pic(3)
,pic(4),pic(5),pic(6),pic(7),richt,zusat
z;
320 a=a+1:GOTO 120
```



ROUND (gkA(,gA))

Gleitkommafunktion: Ermittelt aus gkA einen gerundeten Wert; wie gerundet wird, ist abhängig von gA. Als positiver Wert ergibt gA die Zahl der Nachkommastellen an. Es wird dann mit der letzten verlorenen Stelle gerundet. Ein negativer Wert von gA ersetzt – beginnend bei der ersten Vorkommastelle – soviel Stellen durch Null, wie der Absolut-

wert von gA angibt. Die nächst höhere Stelle wird mit der letzten weggefallenen gerundet.

Es ist sinnlos, für gA größere Zahlen als acht einzusetzen, da der Schneider, wenn keine Exponentialschreibweise gewählt wird, maximal auf 9 Stellen genau rechnet. Sobald der Schneider diesen Modus wählt, ist der sinnvolle Einsatz von gA abhängig vom Exponenten:

```
PRINT ROUND(123456789E+27,-32)
```

Verwendungsmöglichkeiten: Die meisten Aufgaben für ROUND werden sich im kaufmännischen Bereich ergeben. Hier ein Beispiel, das die Mehrwertsteuer von einem Betrag errechnet – sowohl, wenn sie in dem Betrag enthalten ist, als auch, wenn sie herausgezogen wird.

```
10 INPUT"Betrag";Betrag
20 Steuerin=ROUND((Betrag/114)*14,2)
30 Steuerex=ROUND((Betrag/100)*14,2)
40 PRINT"Wenn Steuer im Betrag enthalten
, MWSt=";Steuerin;" Nettobetrag=";Betrag
-Steuerin
50 PRINT"Wenn Steuer nicht enthalten
  MWSt=";Steuerex;" Bruttobetrag=";Betra
g+Steuerex
```

In diesem Fall könnte man das Problem mit PRINT USING genauso gut oder sogar besser lösen. Da man aber die gerundeten Werte von PRINT USING nicht weitergeben kann, muß man Zahlen, mit denen man weiterrechnen will, erst auf den gleichen Wert bringen, der mit PRINT USING ausgedruckt wird.

```
10 INPUT"Bitte eine Zahl eingeben";zahl
20 b=zahl/3
30 c=b^2
40 PRINT USING"Ein Drittel von ### ist #
##.##, das Quadrat des Drittels ist ####
#.##";zahl,b,c
50 d=ROUND(zahl/3,2)
60 e=ROUND(d^2,2)
70 PRINT USING"Ein Drittel von ### ist #
##.##, das Quadrat des Drittels ist ####
#.##";zahl,d,e
```

RUN

Systembefehl: 1. Mit der Syntax RUN znr wird ein im BASIC-Speicher enthaltenes Programm gestartet – entweder bei der niedrigsten Zeilennummer – oder bei znr. Alle Variablen und Tabellen werden gelöscht und alle BASIC-Pointer zurückgesetzt (außer HIMEM!).

2. Mit der Syntax RUN sA wird ein Programm von Kassette geladen und danach gestartet. Durch SAVE »name«,P geschützte Programme können ausschließlich so geladen werden.

SAVE

Systembefehl: Speichert ein Programm aus dem Speicher auf Kassette. Sie haben beim Gebrauch von SAVE folgende Möglichkeiten:

SAVE sA

Das Basicprogramm im Speicher wird in der Form auf Kassette gespeichert, in der es im Speicher vorliegt, also die Basic-Wörter als ein oder zwei Byte lange Tokens.

SAVE sA, A

Das Basicprogramm im Speicher wird als Listing im ASCII-Code gespeichert. Das Programm ist normal ladbar und ablaufbar, hat allerdings den Vorteil, daß es in einem Textverarbeitungsprogramm verändert werden kann. Zu Dokumentationszwecken ist diese Speicherungsform sehr gut geeignet.

SAVE sA,P

Das Basicprogramm im Speicher wird mit einem Listschutz versehen und auf Kassette gespeichert. Ein LIST nach dem Laden ist nicht mehr möglich.

SAVE sA, B, gA1, gA2

Ein Speicherbereich, der bestimmt wird durch die Anfangsadresse gA1 und die Länge gA2 wird Byte für Byte übertragen. Bei einem späteren LOAD wird dieser Bereich genau

wieder an diese Adresse übertragen, es sei denn, Sie geben durch die Parameter nach LOAD etwas anderes an. Diese Form eignet sich neben dem Speichern von Maschinenprogrammen hervorragend dazu, komplette Bildschirmseiten zu speichern. Ein kurzes Ladeprogramm vor dem Hauptprogramm lädt dann erst das Titelbild und dann das Hauptprogramm. So nimmt die Erzeugung des Titels keinen Speicherplatz weg.

SGN (gkA)

Ganzzahlfunktion: Ermittelt das Vorzeichen des gkA. Ist gkA negativ, so liefert SGN den Wert −1, bei einem positiven Argument den Wert +1 und 0, wenn auch gkA gleich Null ist.

Verwendungsmöglichkeiten: Außer der bekannten Funktion, abhängig vom Vorzeichen einer Zahl eine bestimmte Aktion herbeizuführen, ist SGN die einfachste Möglichkeit, den Inhalt einer numerischen Variablen auf einen von Null verschiedenen Wert zu prüfen.

```
10 INPUT"Bitte eine Zahl ungleich Null "
,a
20 IF SGN(a) THEN GOTO 50
30 PRINT"Ich hab' gesagt "CHR$(24)"ungle
ich Null!"CHR$(24)
40 GOTO 10
50 PRINT 5;"/";a;"=";5/a
```

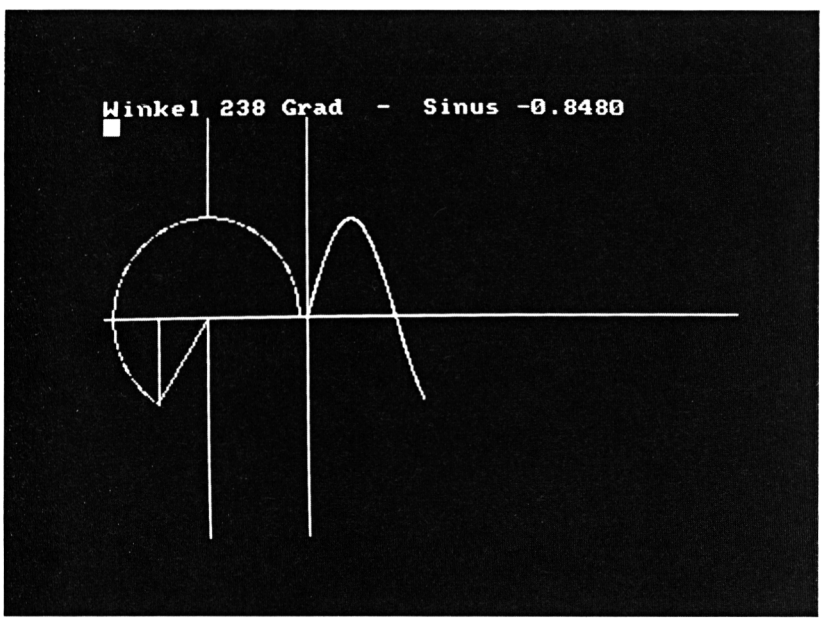
SIN (gkA)

Gleitkommafunktion: Liefert den Sinus eines Bogens im Einheitskreis. Wenn Sie vorher den Befehl DEG benützen, wird der Sinus von einem Winkel berechnet.

Sinus gibt das Verhältnis von Gegenkathete zur Hypothenuse an. Der Sinus eines Winkels kann Werte zwischen −1 (270 Grad) und +1 (90 Grad) annehmen.

```
10 DEFINT i
20 MODE 1
30 ORIGIN 100,200
40 DRAWR-100,0,1
50 DRAWR+640,0
60 MOVE 0,200
70 DRAW 0,-200
80 MOVE 100,200
```

```
90 DRAW 100,-200
100 DEG
110 FOR i=0 TO 720
120 PLOT 90*COS(i-1),0,0
130 DRAW 90*COS(i-1),90*SIN(i-1),0
140 DRAW 0,0,0
150 DRAW 90*COS(i-1),0,0
160 PLOT 92*COS(i),92*SIN(i),1
170 PLOT 90*COS(i),0,1
180 DRAW 90*COS(i),90*SIN(i),1
190 DRAW 0,0,1
200 DRAW 90*COS(i-1),0,1
210 PLOT 100+i/2,90*SIN(i),1
220 LOCATE 1,1:PRINT USING"Winkel ### Gr
ad  -  Sinus ##.#### ";i,SIN(i)
230 NEXT i
```



SOUND

Soundbefehl: Siehe im Kapitel über Musik.

SPACE$(gA)

Stringfunktion: Liefert einen String in der Länge gA, der aus Leerzeichen besteht (CHR$(32)).

Verwendungsmöglichkeiten: SPACE$ kann zum Löschen von Falscheingaben und zum Formatieren der Druckausgabe verwendet werden.

```
10 MODE 2
20 fr$="Bitte geben Sie Ihren Namen ein
(max. 8 Buchstaben)"
30 LOCATE 1,12
40 PRINT fr$;SPACE$(80-LEN(fr$))
50 LOCATE LEN(fr$)+2,12
60 INPUT"",na$
70 PRINT SPACE$(80)+CHR$(11)+CHR$(11)
80 IF LEN(na$)>8 THEN PRINT"Zu lang!":GO
TO 30
90 FOR i=1 TO 72
100 PRINT SPACE$(i);na$
110 NEXT i
```

SPEED INK gA1,gA2

Befehl: Setzt die Dauer, für die bei blinkenden INK-Anweisungen die beiden Farben auf dem Monitor sichtbar sind. gA1 gibt die Dauer der ersten Farbe, gA2 die Dauer der zweiten Farbe an.

Sowohl gA1 als auch gA2 dürfen nicht größer als 255 sein – »Improper argument« wäre die Folge. Eine SPEED INK-Anweisung wird erst ausgeführt, wenn der nächste Farbwechsel fällig wäre. Es gilt immer der letzte SPEED INK-Befehl. Die Dauer der Farben kann durch andere Anweisungen geändert werden. Verwendungsmöglichkeiten liegen hauptsächlich im graphischen Bereich und bei der Programmierung von Spielen. Um den passenden Bildhintergrund für eine Explosion zu erreichen, benötigen Sie nur noch die INK-Anweisung und SPEED INK:

```
10 MODE 0
20 BORDER 26,0
30 INK 0,0,26
40 INK 1,13
50 LOCATE 9,12
60 PRINT"BUMM"
```

```
70 SPEED INK 5,5
80 WHILE INKEY$=""
90 WEND
100 INK 0,1
110 BORDER 1
120 INK 1,24
```

SPC

Siehe PRINT SPC(x)

SPEED WRITE gA

Befehl: Gibt die Ausgabegeschwindigkeit für Kassettenoperationen vor. gA darf nur die Werte 0 oder 1 annehmen. Ist gA gleich 0, dann beträgt die Ausgabegeschwindigkeit 1000 Baud, das heißt circa 125 Byte pro Sekunde, bei einem Wert 1 genau die doppelte Geschwindigkeit, also circa 250 Zeichen pro Sekunde (2000 Baud).

Beim Einlesen von Informationen von Kassette erkennt der »Schneider« selbsttätig, welche Aufzeichnungsgeschwindigkeit gewählt wurde. Das Lesen einer Information mit hoher Aufzeichnungsgeschwindigkeit hat keinen Einfluß auf den Schalter SPEED WRITE.

SQ (gA)

Systemvariable: Siehe im Kapitel über Musik.

SQR (gkA)

Gleitkommafunktion: Ermittelt die Quadratwurzel aus dem numerischen Ausdruck.

Der Benutzer muß selbst überprüfen, ob sich gkA in den erlaubten Grenzen bewegt. Ein negativer gkA ergibt ein »Improper argument«. Die Funktion wird relativ selten benutzt, da der Operator (^) wesentlich flexibler ist.

STOP

Befehl: Unterbricht den Ablauf eines Programmes mit Angabe der Zeilennummer, in der der STOP-Befehl steht.

Verwendungsmöglichkeiten: STOP wird meistens beim Austesten eines Programmes eingesetzt. Will man überprüfen, ob eine Variable zu einem bestimmten Zeitpunkt einen bestimmten Wert angenommen hat, so setzt man an die Stelle im Programmtext ein STOP.

STR$ (nA)

Stringfunktion: Liefert einen String aus dem dezimalen Wert des nA. Dieser String hat die gleiche Form, die das Ausdrucken des nA auf dem Bildschirm bewirken würde.

Verwendungsmöglichkeiten: Diese Funktion benutzt man, um Texte und Zahlenwerte zu einem String zusammenzuketten. Leider hat man mit STR$ nicht die gleichen Möglichkeiten wie bei PRINT USING. Eine Formatierung muß der Programmierer selbst durchführen. Ein weiteres Problem ergibt sich durch die Tatsache, daß positive Zahlen im Gegensatz zu negativen mit einer führenden Leerstelle ausgedruckt werden.

```
10 INPUT"Bitte eine Zahl";z
20 a$=STR$(z)
30 IF ASC(a$)=32 THEN a$=MID$(a$,2)
40 PRINT z
50 PRINT STR$(z)
60 PRINT a$
```

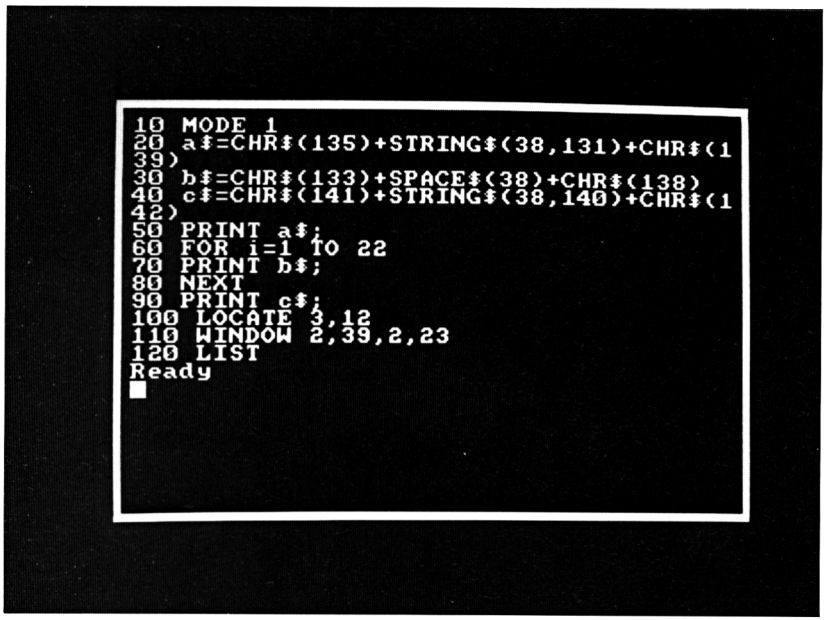
STRING$ (gA,gA2)

STRING$ (gA,sA)

Stringfunktion: Diese Funktion liefert, abhängig von der verwendeten Syntax, einen String, dessen Länge sich ergibt aus gA und der aus lauter Zeichen mit dem ASCII-Code gA2 besteht. Die zweite mögliche Syntax liefert einen String aus lauter Zeichen, die gleich dem ersten Zeichen von sA sind.

PRINT STRING$ (40,65) und PRINT STRING$ (40,"Andreas") liefern beide einen String auf dem Bildschirm, der aus 40 großen A's besteht. Sowohl gA als auch gA2 dürfen nur Werte von 0 bis 255 annehmen. Alle anderen Werte werden mit »Improper argument« abgewiesen.

Verwendungsmöglichkeiten: Es haben sich im Laufe der Zeit zwei Arten herausgestellt, STRING$ im Programm einzusetzen. Häufig kommt es vor, daß einem String ein Anfangswert zugewiesen werden muß; STRING$ bietet dazu die Möglichkeit. Der zweite Einsatz ergibt sich beim Aufbau eines Bildschirms.



SYMBOL gA1,gA2,gA3,gA4,gA5,gA6,gA7,gA8,gA9

Befehl: Ordnet dem Zeichen, das durch gA1 bestimmt wird, ein neues Aussehen zu. Das Aussehen wird bestimmt durch gA2 bis gA9. Mindestens ein gA muß nach dem ASCII-Code gA1 eingesetzt werden.

Beim Einsatz von SYMBOL unterliegt man Beschränkungen. Will man zum Beispiel das Zeichen mit einem ASCII-Code geringer als 240 verändern, so muß man die Anweisung SYMBOL AFTER benutzen. Alle gAs dürfen nur Werte zwischen 0 und 255 enthalten. gA2 bis gA9 bestimmen das Bit-Muster, das das Zeichen auf dem Bildschirm erzeugt. Es empfiehlt sich, beim Entwerfen eines Zeichens folgendes Musterblatt zu benutzen:

128	64	32	16	8	4	2	1	
								1
								2
								3
								4
								5
								6
								7
								8

Verwendungsmöglichkeiten: SYMBOL ist die optimale Lösung zur Herstellung eines benutzerdefinierten Zeichensatzes. Als Beispiel eine deutsche Tastatur:

```
10 KEY DEF 71,1,121,89,25
20 KEY DEF 43,1,122,90,26
30 KEY DEF 57,1,51,64,0
40 KEY DEF 32,1,48,61
50 KEY DEF 25,1,126,63
60 KEY DEF 26,1,125,93
70 KEY DEF 29,1,124,92
80 KEY DEF 28,1,123,91
90 KEY DEF 17,1,43,42
100 KEY DEF 19,1,35,94
110 KEY DEF 39,1,44,59
120 KEY DEF 31,1,46,58
130 KEY DEF 30,1,45,95
140 KEY DEF 22,1,60,62
150 SYMBOL AFTER 64
160 SYMBOL 64,60,96,60,102,102,60,6,60
170 SYMBOL 91,198,56,108,198,254,198,198
,0
180 SYMBOL 92,198,56,108,198,198,108,56,
0
190 SYMBOL 93,198,0,198,198,198,198,124,
0
200 SYMBOL 123,102,0,120,12,124,204,118,
0
210 SYMBOL 124,102,0,60,102,102,102,60,0
220 SYMBOL 125,102,0,0,102,102,102,62,0
230 SYMBOL 126,60,102,102,108,102,102,12
4,96
```

SYMBOL AFTER gA

Befehl: Durch gA wird bestimmt, ab welchem ASCII-Code ein benutzerdefinierter Zeichensatz erstellt werden kann.

gA darf Werte zwischen 0 und 255 annehmen, alles andere wird durch »Improper argument« abgewiesen.

Verwendungsmöglichkeit: Will man auch das letzte Byte im Speicher ausnutzen, so kann man durch die Anweisung SYMBOL AFTER 255 ungefähr noch 100 weitere Speicherplätze freimachen:

```
10 PRINT HIMEM

20 SYMBOL AFTER 255

30 PRINT HIMEM
```

TAB

Siehe PRINT TAB (x)

TAG ⟨#gA⟩

Befehl: Setzt den nächsten Druckvorgang mit PRINT auf die aktuelle Graphik-Cursorposition. Alle Steuerzeichen (ASCII-Codes von 0 bis 31) werden durch ihre graphische Entsprechung dargestellt. Die Angabe eines WINDOWS durch gA hat – im Gegensatz zu der Behauptung im Handbuch – keinen Einfluß auf die Gestaltung des Bildschirms. Den Beweis bietet folgendes Programm:

```
10 WINDOW#1,1,10,1,10

20 TAG#1

30 PRINT#1,"SCHNEIDER";
```

Obwohl TAG die Möglichkeit bietet, an jeder beliebigen Stelle des Bildschirms drucken zu können, sind die Einsatzmöglichkeiten für die Ausgabe längerer Texte beschränkt, da man für eine neue Zeile den Graphik-Cursor auf eine neue Position setzen muß. Die linke obere Ecke des ersten Zeichens, das ausgedruckt werden soll, wird genau auf der Graphik-Cursorposition ausgegeben.

Verwendungsmöglichkeiten: Bei der Beschriftung von Graphiken und bei der Erstellung von Titeln bietet TAG unschätzbare Hilfe.

```
10 MODE 0
20 ORIGIN 50,0
30 FOR i=0 TO 300 STEP 8
```

```
40 MOVE 100,i
50 TAG
60 PRINT "Schneider";
70 MOVE 100,i-15
80 PRINT"          ";
90 NEXT
100 FOR i=-300 TO 90 STEP 4
110 MOVE i,280
120 PRINT" Computer";
130 NEXT
```

TAGOFF ⟨gA⟩

Befehl: Schaltet die Druckausgabe auf Graphik-Cursorposition wieder ab. Die nächste Druckausgabe erfolgt an der Stelle, die durch den TAG-Befehl verlassen wurde. Ein Programmende verursacht automatisch ein TAGOFF.

```
10 MODE 2
20 ORIGIN 0,200
30 FOR i=1 TO 30
40 IF i/2=INT(i/2) THEN TAG
50 PRINT"Schneider"
60 TAGOFF
70 NEXT
```

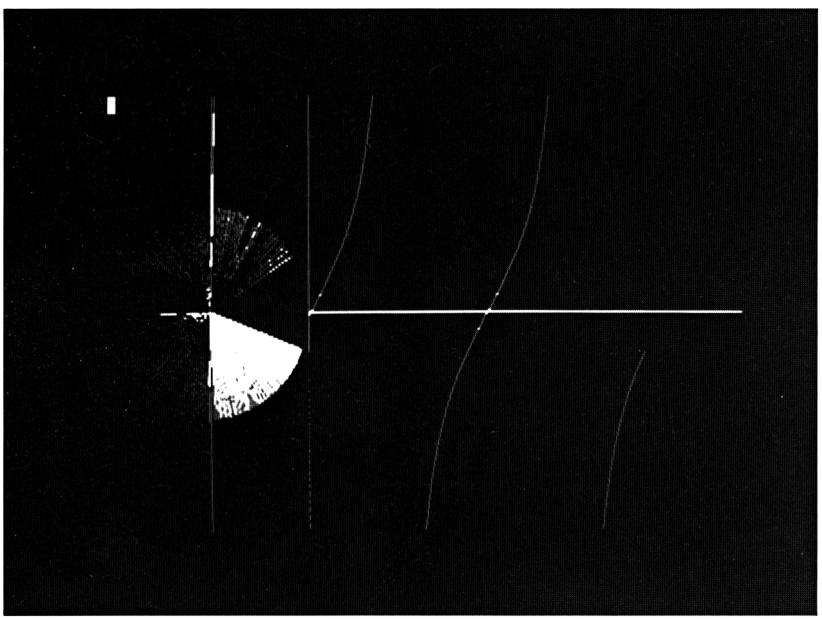
TAN (nA)

Gleitkommafunktion: Ermittelt den Tangens des nA im Bogenmaß. nA darf, wenn im Bogenmaß gerechnet wird, zwischen −205884 und 205884 liegen. Wenn durch die Anweisung DEG auf Winkelmaß umgeschaltet wurde, darf nA zwischen −11796299 und 11796299 liegen. Werte, die jenseits dieser Grenzen sind, werden durch »Improper argument« abgewiesen.

Der Tangens gibt an, in welchem Verhältnis die Gegenkathete und Ankathete eines rechtwinkligen Dreiecks zueinander stehen. Er kann an der Tangente im Einheitskreis abgelesen werden. Zur Demonstration ein kleines Programm:

```
10 MODE 2
20 ORIGIN 100,200
30 DEG
```

```
40 FOR i=0 TO 360
50 PLOT 98*SIN(i),98*COS(i)
60 NEXT
70 PLOT -100,0
80 DRAW 540,0
90 PLOT 0,-200
100 DRAW 0,200
110 FOR i=0 TO 360
120 IF i>89 AND i<92 OR i>269 AND i<272
THEN 220
130 PLOT 0,0
140 DRAW 98*COS(i),98*SIN(i)
150 MOVE 98*COS(i-1),98*SIN(i-1)
160 DRAW 98,TAN(i-1)*98,0
170 PLOT 98,0,1
180 DRAW 98,TAN(i)*98
190 DRAW 98*COS(i),98*SIN(i),1
200 MOVE i+99,TAN(i-1)*98
210 DRAW i+100,TAN(i)*98
220 NEXT
```



TEST (gA1,gA2)

Ganzzahlfunktion: Liefert die Farbstiftnummer, die an der durch die x-Koordinate gA1 und die y-Koordinate gA2 bestimmten Positionen verwendet wurde. Dabei werden gA1 und gA2 auf die durch eine vorherige ORIGIN-Anweisung festgelegten Bildschirmkoordinaten bezogen. Liegen gA1 oder gA2 außerhalb des sichtbaren Bildschirms, liefert TEST immer 0, selbst wenn eine PLOT- oder DRAW-Anweisung auf diese Stelle gezeigt hat. Es kann auf diese Art und Weise also nur der sichtbare Bildschirm untersucht werden.

Verwendungsmöglichkeit: TEST kann bei der Programmierung von Spielen und Graphikprogrammen dazu dienen, bestimmte Bereiche auf dem Bildschirm abzufragen. Als Beispiel ein Programm, das im MODE 2 Buchstaben vergrößert:

```
10 DEFINT a-z
20 MODE 2
30 INPUT"Nachricht";a$
40 INPUT"Vergroesserung x-Achse";x
50 INPUT"Vergroesserung y-Achse";y
60 y=2*y
70 CLS
80 PRINT a$
90 FOR i=0 TO LEN(a$)*8
100 FOR j=0 TO 7
110 IF TEST(i,400-2*(j+1))<>1 THEN 210
120 links=i*x
130 rechts=i*x+x
140 oben=380-j*y
150 unten=380-j*y-y
160 FOR l=links TO rechts
170 FOR m=oben TO unten STEP-2
180 PLOT l,m
190 NEXT m
200 NEXT l
210 NEXT j
220 NEXT i
230 LOCATE 1,1
240 PRINT SPACE$(LEN(a$))
250 IF INKEY$="" THEN 250
```

TESTR (gA1,gA2)

Ganzzahlfunktion: Liefert die Farbstiftnummer. die an einer Stelle verwendet wurde, die bestimmt ist durch die aktuelle x-Position des Graphik-Cursors plus gA1 und die aktuelle y-Position plus gA2.

TIME

Systemvariable: In TIME ist die Zeit gespeichert, die seit dem Einschalten des Gerätes vergangen ist. Nicht gezählt wird die Zeit, die für Ein- und Ausgabeoperationen auf Kassette verbraucht wird.

Teilt man den aktuellen Wert von Time durch 30, so erhält man die Arbeitszeit in Sekunden. Folgendes kleine Programm kann die Variable TIME wieder auf 0 setzen.

```
10 PRINT TIME
20 POKE 45447,0
30 POKE 45448,0
40 POKE 45449,0
50 POKE 45450,0
60 PRINT TIME
```

TROFF

Befehl: Schaltet die Ausgabe von Programmzeilennummern auf dem Bildschirm wieder ab.

TROFF kann im Programm dazu verwendet werden, vor Warteschleifen (zum Beispiel INKEY$) die TRON-Option abzuschalten. Damit wird vermieden, daß die Zeilennummern den gesamten Bildschirm zerstören. Nach Ende der Warteschleife kann auch das TRON im Programm eingesetzt werden.

TRON

Befehl: Veranlaßt den Computer, bei der Verarbeitung von Programmzeilen die jeweilige Zeilennummer in eckigen Klammern auf dem Bildschirm auszugeben.

Warteschleifen, zum Beispiel bei Tastaturabfragen, können dazu führen, daß von dem Originalbildschirm nichts mehr zu sehen ist.

```
10 TRON

20 PRINT"Schneider"
```

```
30 TROFF

40 a$=INKEY$

50 IF a$="" THEN GOTO 10

60 TRON

70 PRINT"Computer"
```

UNT(gA)

Ganzzahlfunktion: Falls gA einen Wert zwischen −32768 und 32767 hat, liefert UNT genau diese Zahl zurück. Hat gA einen Wert zwischen 32768 und 65535, so liefert UNT diesen Wert minus 65536. Jenseits dieser Grenzen erzeugt UNT einen »Overflow«.

Da viele Ganzzahlfunktionen an bestimmte Parameter gebunden sind, was den Zahlenbereich angeht, hilft die Funktion UNT dabei, Rechenergebnisse, die jenseits von 32767 sind, in das entsprechende Format umzuwandeln. Nähere Angaben über die Logik, die dahinter steht, finden Sie ab Seite 49.

UPPER$(sA)

Stringfunktion: Ermittelt aus sA alle Buchstaben und liefert diese als Großbuchstaben zurück.

Verwendungsmöglichkeiten: Bei Sortier- und Vergleichsroutinen bietet UPPER$ die Möglichkeit, Groß- und Kleinschreibung zu ignorieren. Auch einzelne Tastenabfragen können auf diese Weise wesentlich präziser formuliert werden.

```
10 a$="Der Schneider Personal Computer i
st ein vielseitiger Partner fuer Buero u
nd Hobby"
20 INPUT"Nach welcher Buchstabenkombinat
ion suchen Sie";su$
30 PRINT"Klein/Grossbuchstaben ignoriere
n J/N"
40 i$=UPPER$(INKEY$)
50 IF i$="J" THEN b$=UPPER$(a$):su$=UPPE
R$(su$):GOTO 80
```

```
60 IF i$="N" THEN b$=a$:GOTO 80
70 GOTO 40
80 FOR i=1 TO LEN(a$)
90 IF MID$(b$,i,LEN(su$))=su$ THEN a=a+1
100 NEXT
110 PRINT "Die gesuchte Kombination kam"
a"mal vor."
120 a=0
130 GOTO 20
```

VAL(sA)

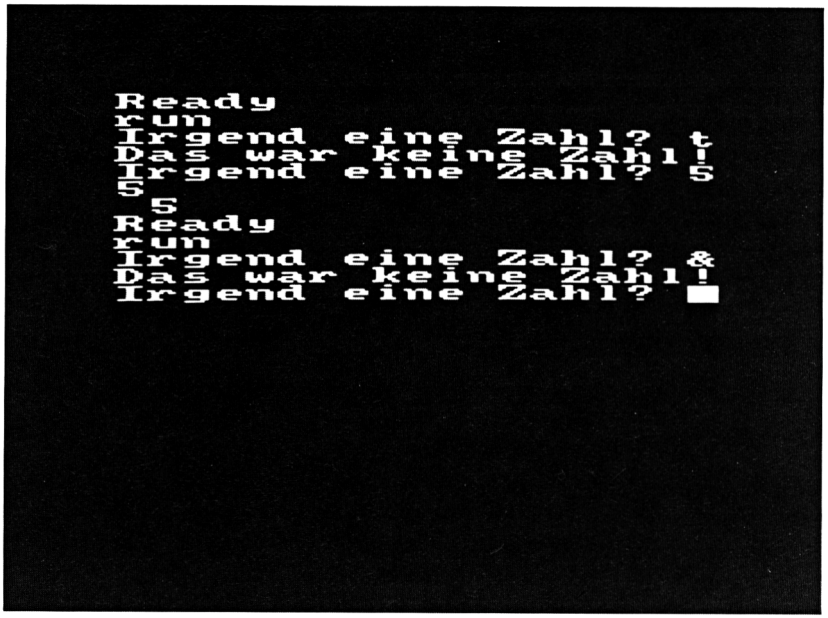
Gleitkommafunktion: Liefert den numerischen Wert von sA, solange dieser numerischen Inhalt hat. Beim ersten nicht numerischen Zeichen wird die Verarbeitung abgebrochen.

Leider ist bei der Programmierung des BASIC-Interpreters der Firma Amstrad ein kleiner Fehler unterlaufen. Obwohl die einzige Fehlermeldung, die VAL bei richtiger Anwendung erzeugen könnte, ein »Overflow« wäre, haben sich in diese Funktion einige Unstimmigkeiten eingeschlichen. Enthält der sA ausschließlich einen der folgenden Strings: &,+,−,&×,., dann bricht die Verarbeitung mit einem »Type mismatch« ab. Man kann sich helfen, indem man die VAL-Funktion erst dann einsetzt, wenn man diese Fehlermöglichkeit ausgeschlossen hat:

```
10 INPUT a$

20 IF INSTR("+-.&x",a$)>0 THEN PRINT "Fehler!"

   :GOTO 10

20 PRINT VAL(a$)

30 GOTO 10
```

Verwendungsmöglichkeit: Um die lästige Redo-Meldung bei einem INPUT zu vermeiden, kann man mit VAL die Eingabe eines Strings in eine Zahl umwandeln.

```
10 INPUT"Irgend eine Zahl";a$
20 p=0:s=0
30 beginn=1
```




```
40 IF LEFT$(a$,1)="-" OR LEFT$(a$,1)="+"
 THEN beginn=2
50 FOR i=beginn TO LEN(a$)
60 b$=MID$(a$,i,1)
70 IF b$="." AND p=0 THEN p=1:GOTO 100
80 IF b$<"0" OR b$>"9" THEN GOSUB 150:GO
TO 10
90 s=1
100 NEXT i
110 IF s=0 THEN GOSUB 150:GOTO 10
120 zahl=VAL(a$)
130 PRINT a$,zahl
140 END
150 PRINT"Das war keine Zahl!"
160 RETURN
```

VPOS (#gA)

Ganzzahlfunktion: Ermittelt die Bildschirmzeile, in der der Cursor sich befindet. Bezugspunkt ist immer die linke obere Ecke des durch gA angesprochenen Fensters. Wurde

kein Fenster definiert, so benutzt man für gA den Wert 0. gA darf sich im Bereich von 0 bis 7 bewegen, alles andere ergibt ein »Improper argument«

Verwendungsmöglichkeit: Im Zusammenhang mit LOCATE und POS liefert VPOS die Möglichkeit zur optimalen Bildschirmgestaltung innerhalb eines Fensters.

```
10 ' Zeichnen mit den Cursortasten und B
uchstaben
20 MODE 1
30 b$=CHR$(&F1)+CHR$(&F0)+CHR$(&F3)+CHR$
(&F2)
40 x=POS(#0)
50 y=VPOS(#0)
60 a$=INKEY$
70 IF a$="" THEN 60
80 IF x=2 AND cu$=CHR$(8)+CHR$(8)THEN 19
0
90 IF x=37 AND cu$="" THEN 210
100 IF y=1 AND cu$=CHR$(11)+CHR$(8)THEN
150
110 IF y=24 AND cu$=CHR$(10)+CHR$(8)THEN
 170
120 ON INSTR(b$,a$) GOTO 150,170,190,210
130 PRINT cu$;a$;
140 GOTO 40
150 IF y<24 THEN cu$=CHR$(10)+CHR$(8)
160 GOTO 220
170 IF y>1 THEN cu$=CHR$(11)+CHR$(8)
180 GOTO 220
190 IF x<37 THEN cu$=""
200 GOTO 220
210 IF x>2 THEN cu$=CHR$(8)+CHR$(8)
220 a$=""
230 GOTO 40
```

WAIT gA1,gA2 ⟨,gA3⟩

Befehl: Ein selten benützter Befehl, der nur eingesetzt werden sollte, wenn der mögliche Inhalt einer Adresse genau bekannt ist. Das Programm hält solange an, bis der Inhalt von gA1 identisch mit gA2 ist. Vorher kann man noch den Inhalt von gA1 durch ein XOR mit gA3 verknüpfen, um auf diese Art bestimmte Bits herauszufiltern. Es können aus-

schließlich Interfaceadressen auf diese Weise getestet werden, weil sich Ihr Schneider sonst aufhängt.

WEND

Befehl: Beendet eine WHILE-Schleife; siehe WHILE.

WHILE

Befehl: Leitet eine Schleife ein, die bei einem WEND wieder aufhört und solange durchlaufen wird, bis die Aussage, die hinter WHILE stehen muß, einen Wert 0 ergibt. Ergibt die Auswertung des logischen Ausdrucks schon bei der ersten Überprüfung einen Wert 0, so wird die WHILE..WEND-Schleife übersprungen.

Die Logik hinter WHILE entspricht dem menschlichen Denken wesentlich eher als die komplizierte FOR..NEXT Logik. Solange die Aussage hinter WHILE zutrifft, wird all das ausgeführt, was bis zum WEND an Programmzeilen existiert. Wichtig ist noch die Feststellung, daß die Überprüfung des Wahrheitsgehaltes der Aussage beim WHILE durchgeführt wird, und nicht beim WEND. Das kann dazu führen, daß eine Änderung im Wahrheitsgehalt in der ersten Zeile nach WHILE die gesamte Schleife noch einmal durchlaufen läßt. Der große Vorteil ist, daß Bedingungen, die bis jetzt nur in verschiedenen Programmzeilen oder verwickelter Logik programmiert werden konnten, nun innerhalb einer Zeile ohne Sprungfehler formuliert werden können. Also

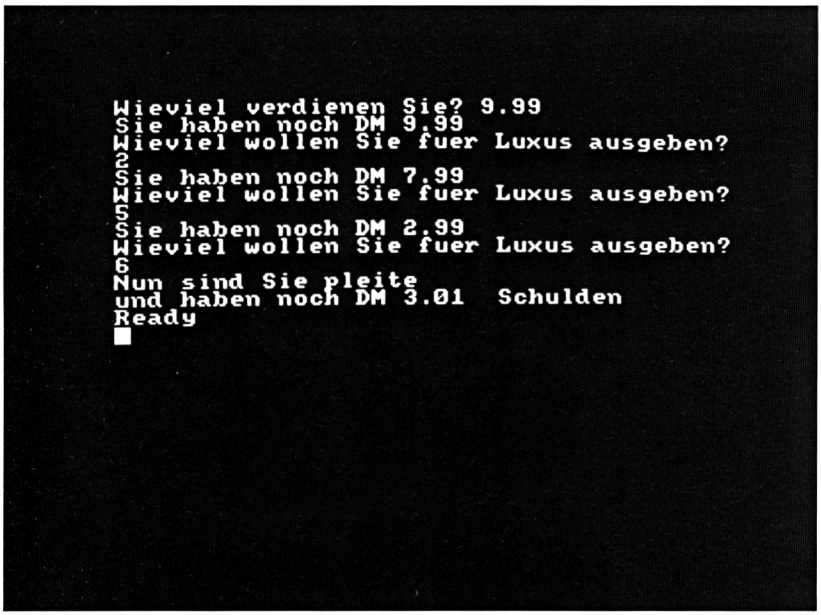
```
10 WHILE INKEY$ = "":WEND
```

statt

```
10 IF INKEY$ = "" THEN GOTO 10
```

Ein weiterer großer Vorteil ergibt sich aus der Tatsache, daß Zählschleifen programmiert werden können, in denen im Gegensatz zu FOR..NEXT die Schrittweite nicht konstant sein muß.

```
10 CLS
20 INPUT"Wieviel verdienen Sie";a
30 WHILE a>0
40 PRINT "Sie haben noch DM"a
50 INPUT"Wieviel wollen Sie fuer Luxus a
usgeben";b
60 a=a-b
```




```
70 WEND
80 PRINT"Nun sind Sie pleite"
90 IF a<>0 THEN PRINT"und haben noch DM"
ABS(a)" Schulden"
```

WIDTH gA

Befehl: Gibt für die Ausgabe auf dem Drucker die Zeichen pro Zeile vor. gA darf im Bereich von 1 bis 255 liegen. Normale Breite wäre 80.

Da die meisten Drucker am Ende einer Zeile automatisch einen Carriage Return einfügen, kann eine zu groß gewählte Zahl zum Überdrucken von Zeilen führen. Erreicht der Druckkopf die horizontale Position gA, so fügt BASIC ein Line Feed und ein Carriage Return ein.

WINDOW(#gA1,)gA2,gA3,gA4,gA5

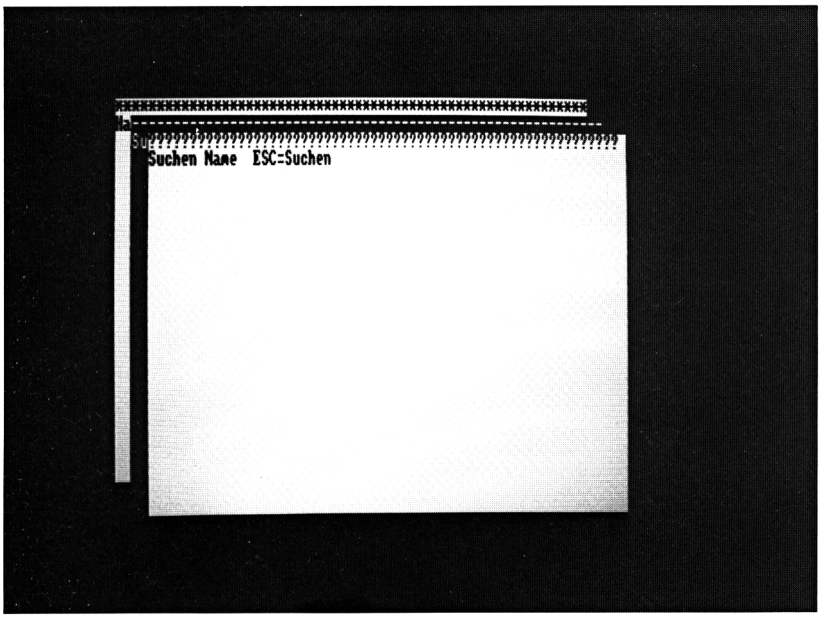
Befehl: Definiert einen Bereich des Bildschirms, dessen Grenzen durch gA2 bis gA5 bestimmt sind, als Fenster. Dieses Fenster kann bei PRINT, INPUT, POS, VPOS, LOCATE

und CLS als gesonderte Ausgabeeinheit mit seiner Nummer, die durch gA1 bestimmt wird, angesprochen werden.

gA2 gibt den linken Rand, gA3 den rechten, gA4 den oberen und gA5 den unteren Rand an. Diese Ausdrücke können Werte zwischen 1 und 255 annehmen; sobald aber die Grenzen des durch MODE festgelegten Bildschirmformats verlassen werden, sind die Ergebnisse bei Ausgaben unvorhersehbar.

Verwendungsmöglichkeit: WINDOW bietet dem Benutzer bei geschickter Anwendung immer die Möglichkeit, zu sehen, auf welcher Stufe eines menuegesteuerten Programmes er sich befindet.

```
10 KEY DEF 66,0,27,252,252
20 MODE 2
30 WINDOW#1,1,60,1,22:PEN#1,0:PAPER#1,1
40 WINDOW#2,3,63,2,23:PEN#2,1:PAPER#2,0
50 WINDOW#3,5,65,3,24:PEN#3,0:PAPER#3,1
60 CLS#1
70 PRINT#1,STRING$(60,"*");
80 PRINT#1,"Hauptmenue  ESC=Ende"
90 LOCATE#1,3,3
100 PRINT#1,"<1>= SUCHEN"
110 a$=INKEY$:IF a$="1" THEN GOSUB 140:C
LS#1:GOTO 70
120 IF a$=CHR$(27) THEN CLS:END
130 GOTO 110
140 CLS#2
150 PRINT#2,STRING$(60,"-");
160 PRINT#2,"Suchen  ESC=Hauptmenue"
170 LOCATE#2,3,3
180 PRINT#2,"<1> Name"
190 a$=INKEY$:IF a$="1" THEN GOSUB 220:G
OTO 140
200 IF a$=CHR$(27) THEN RETURN
210 GOTO 190
220 CLS#3
230 PRINT#3,STRING$(60,"?");
240 PRINT#3,"Suchen Name  ESC=Suchen"
250 a$=INKEY$:IF a$<>CHR$(27) THEN 250
260 RETURN
```



WINDOW SWAP gA1, gA2

Befehl: Vertauscht die Grenzen und andere Parameter, die für 2 WINDOWS gelten, miteinander.

Ist eines dieser beiden WINDOWS nicht definiert, werden die Originalgrenzen des Bildschirms benutzt.

Verwendungsmöglichkeit: Benutzt man ein Programm, das mehrere Funktionen ausführen kann, so bietet WINDOW SWAP einen Weg, dem Benutzer zu zeigen, welchen Programmteil er benutzt, und welche anderen er zur Verfügung hat.

WRITE⟨#gA1,⟩Druckliste

Befehl: Schreibt die Druckliste auf das durch gA1 angegebene Ausgabegerät. Die einzelnen Teile der Druckliste werden, durch Kommata getrennt, ausgegeben. Unerheblich ist, ob in der Druckliste die einzelnen Teile durch Komma oder Semikolon getrennt werden.

Verwendungsmöglichkeit: Vor allen Dingen bei der Benutzung des Kassettenrecorders im Zusammenhang mit sequentiellen Dateien gibt uns WRITE die Möglichkeit,

Trennungszeichen einzufügen, ohne sie jedesmal ausdrücklich als String ausgeben zu müssen. Wichtig in diesem Zusammenhang ist noch die Tatsache, daß numerische Daten ohne führende Leerstellen ausgegeben werden.

XOR

Operator: Verknüpft zwei Ausdrücke so miteinander, daß das Ergebnis dann wahr ist, wenn nicht beide richtig oder nicht beide falsch sind.

In der Praxis wird XOR meistens zu Bitmanipulationen benutzt; siehe auch CHR$(23).

XPOS

Ganzzahlfunktion: Liefert die augenblickliche x-Koordinate des Graphik-Cursors.

YPOS

Ganzzahlfunktion: Liefert die augenblickliche Y-Koordinate des Graphik-Cursors.

ZONE gA

Befehl: Setzt die Tabulatorbreite für den Printbefehl; gA darf Werte zwischen 1 und 255 annehmen. Ein Wert, der größer ist als die Anzahl der verfügbaren Zeichen pro Zeile, wird ignoriert; stattdessen wird an seiner Stelle ein Line Feed eingefügt, wenn das PRINT-Kommando ein Komma enthält. Bei einem PRINT-Befehl, der mit einem Komma abgeschlossen wird, wird der Cursor auf die zur Zeit gültige Tabulatorposition gesetzt. Erst das nächste Komma wird durch die geänderte ZONE betroffen.

BASIC-Vergleichstabelle

Schneider CPC464	apple II	commodore 64	Typ
ABS(nA)	ABS(nA)	ABS(nA)	FKF
AFTER	n.v.	n.v.	B
AND	AND	AND	O
ASC(sA)	ASC(sA)	ASC(sA)	GZF
ATN(nA)	ATN(nA)	ATN(na)	FKF
--	AT(siehe VLIN und HLIN)	--	
AUTO	AUTO(nur in Integer B.)	n.v.	SB
BIN$	n.v.	n.v.	SF
BORDER	n.v.	POKE 53280,x	B
CALL	CALL	SYS	B
CAT	n.v.	n.v.	SB
CHAIN	n.v.	n.v.	SB
CHR$(nA)	CHR$(nA)	CHR$(nA)	SF
CINT	(INT(ABS(nA)+0.5)×SGN(nA))		GZF
CLEAR	CLEAR	CLR	B
CLG	(GR,HGR)	n.v.	B
CLOSEIN	n.v.	CLOSE lfn	B
CLOSEOUT	n.v.	CLOSE lfn	B
CLS	HOME	Print CHR$(147);	B
--	PR#	CMD lfn	IOB
(setzt die Ausgabe auf ein anderes Gerät, fallweise durch LIST#, bzw. PRINT# ersetzen)			
CONT	CONT,CON	CONT	SB
--	COLOR=	--	GB
(Setzt die Standardfarbe für Graphikbefehle, ersetzen durch Angabe des Farbparameters bei PLOT und DRAW)			
COS	COS	COS	FKF
CREAL	n.v.	n.v.	FKF
DATA	DATA	DATA	B
DEF FN	DEF FN	DEF FN	B
DEFINT	n.v.	n.v.	B
DEFREAL	n.v.	n.v.	B
DEFSTR	n.v.	n.v.	B
DEG	n.v.	n.v.	FKF
	(Fallweise von Grad in Bogenmaß umrechnen)		
DELETE	DEL	n.v.	SB

Schneider CPC464	apple II	commodore 64	Typ
DI	n.v.	n.v.	B
	(Zurücksetzen des Interruptvektors)		
DIM	DIM	DIM	B
DRAW	PLOT,HPLOT	n.v.	GB
DRAWR	(PLOT,HPLOT)	n.v.	GB
EDIT	n.v.	n.v.	SB
EI	n.v.	n.v.	B
	(Interruptvektor Änderungen zugänglich machen)		
ELSE	n.v.	n.v.	B
END	END	END	B
ENT	n.v.	n.v.	B
	(Klanghüllkurve setzen)		
ENV	n.v.	n.v.	B
	(Lautstärkenhüllkurve setzen)		
EOF	n.v.	(IF ST=64 THEN...)	GZF
ERASE	(CLR)	(CLR)	B
ERL	PEEK-	n.v.	GZF
	(218)+256×PEEK(219)		
ERROR	n.v.	n.v.	SB
EVERY	n.v.	n.v.	B
	(Nur möglich durch verändern des Interrupts)		
EXP	EXP	EXP	FKF
FIX	n.v.	n.v.	FKF
	(Ersetzt INT(ABS(nA))×SGN(nA)		
n.v.	FLASH	n.v.	B
(Läßt Schrift blinken; zu ersetzen durch INK mit zwei Farbangaben)			
FN	FN	FN	FKF
FOR	FOR	FOR	B
FRE	FRE	FRE	FKF
(siehe INKEY$)	GET	GET	B
GOSUB	GOSUB	GOSUB	B
GOTO	GOTO	GOTO	B
n.v.	GR	n.v.	B
(Schaltet den Bildschirm auf graphische Auflösung 40×40)			
n.v.	HCOLOR=	n.v.	B
(Setzt Farbe für nächsten Graphikbefehl in hochauflösender Graphik)			
n.v.	HGR	n.v.	B
(Schaltet den Bildschirm auf graphische Auflösung 280×160)			
n.v.	HGR2	n.v.	B

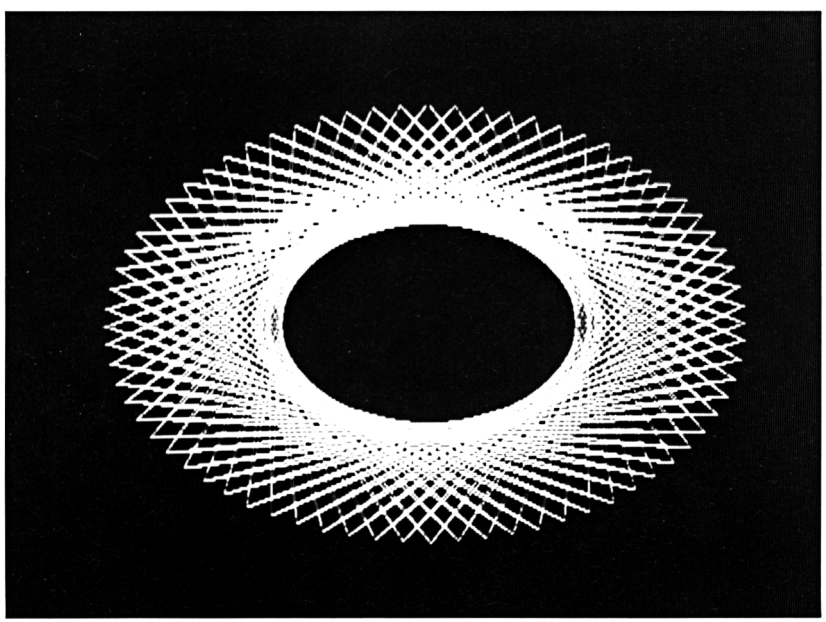
Schneider CPC464	apple II	commodore 64	Typ
(Schaltet den Bildschirm auf graphische Auflösung 280×192)			
HEX$	n.v.	n.v.	SF
HIMEM	PEEK-(116)×256+PEEK(115)	PEEK-(56)×256+PEEK(55)	FKF
(MEMORY)	HIMEM:	n.v.	B
(DRAW)	HLIN	n.v.	GB
(Zeichnet waagerechte Linie in niedrigauflösender Graphik)			
CLS	HOME	PRINT CHR$(147);	B
(PLOT,DRAW)	HPLOT	n.v.	GB
(Verschiedene Wirkungen je nach Parameterangabe – (Immer hochauflösende Graphik): HPLOT nA1,nA2 zeichnet Punkt an Koordinaten x,y HPLOT TO nA1,nA2 zeichnet Linie Position des Graphikcursors nach nA1, nA2 HPLOT nA1,nA2 TO nA3,nA4 zeichnet Linie von nA1,nA2 bis nA3,nA4)			
(siehe LOCATE)	HTAB	n.v.	B
IF	IF	IF	B
INK	n.v.	n.v.	B
INKEY	n.v.	n.v.	GZF
INKEY$	n.v.	n.v.	SF
(A$=INKEY$ hat die gleiche Wirkung wie GET A$ bei apple und commodore			
INP	PEEK(Adresse des I/O = Ports)		GZF
INPUT	INPUT	INPUT	B
INPUT#	n.v.	INPUT#	B
INSTR	n.v.	n.v.	GZF
(zu ersetzen durch	10 FOR I=1 TO LEN (A$) 20 IF MID$(A$,I,LEN(B$))=B$ THEN GOTO 40 30 NEXT I:I=0 40 REM in I ist jetzt die Position des ersten Auftretens von B$ in A$		
INT	INT	INT	FKF
(PRINT CHR$(24);)	INVERSE	(PRINT CHR$(41);)	B
n.v.	IN#	n.v.	B
(Setzt das Standard – Eingabegerät fest)			
JOY	(PDL)	10 POKE 56322,224 20 J=PEEK(56320)	GZF
KEY	n.v.	n.v.	B
KEY DEF	n.v.	n.v.	B
LEFT$	LEFT$	LEFT$	B
LEN	LEN	LEN	B
LET	LET	LET	B

Schneider CPC464	apple II	commodore 64	Typ
LINE INPUT	n.v.	n.v.	B
LIST	LIST	LIST	SB
LOAD	LOAD	LOAD	SB
LOCATE x,y	VTAB x HTAB y	10 POKE 211,x 20 POKE 214,y 30 SYS 58732	B
LOG	LOG	LOG	FKF
LOG10	n.v.	n.v.	FKF
(n.v. (Setzt BASIC-Untergrenze)	LOMEM:	10 a=Basicuntergrenze 20 POKE 44,a–256×INT(a/256) 30 POKE 43,a/256 40 POKE a,0:NEW	B
LOWER$	n.v.	n.v.	SF
MAX	n.v.	n.v.	FKF
MEMORY	HIMEM:	10 a=Basicobergrenze 20 POKE 56,a–256×INT(a/256) 30 POKE 55,a/256 40 CLR	B
MERGE	n.v.	n.v.	SB
MID$	MID$	MID$	SF
MIN	n.v.	n.v.	FKF
MOD	n.v.	n.v.	O
MODE	(PR#3 = MODE 2)	n.v.	B
MOVE	(Entspricht HPLOT TO, wenn HCOLOR=0)	n.v.	GB
MOVER	n.v.	n.v.	GB
NEW	NEW	NEW	SB
NEXT	NEXT	NEXT	B
PRINT CHR$(24);	NORMAL (REVERSE oder FLASH aus)	PRINT CHR$(146);	B
NOT	NOT	NOT	O
TROFF	NOTRACE	n.v.	SB
ON var GOTO	ON var GOTO	ON var GOTO	B
ON var GOSUB	ON var GOSUB	ON var GOSUB	B
ON BREAK GOSUB	n.v.	n.v.	B
ON ERROR GOTO	ON ERR GOTO	n.v.	B
ON SQ GOSUB	n.v.	n.v.	MB
OPENIN	(siehe IN#)	OPEN 1,1,0,„name“	IOB

Schneider CPC464	apple II	commodore 64	Typ
OPENOUT	(siehe PR#)	OPEN 1,1,1,„name"	IOB
OR	OR	OR	O
ORIGIN	n.v.	n.v.	GB
OUT	n.v.	n.v.	IOB
PAPER	n.v.	POKE 53281,farbe	B
(siehe JOY)	PDL	n.v.	GZF
PEEK	PEEK	PEEK	GZF
(Beim apple kann PEEK(x) Einfluß auf den Inhalt einer Speicherstelle haben)			
PEN	n.v.	(siehe Anm. 1)	B
PLOT	PLOT (niedrige Aufl.) HPLOT (hohe Auflösung)	n.v.	GB
PLOTR	n.v.	n.v.	GB
POKE	POKE	POKE	B
n.v.	POP	n.v.	B
(Nimmt eine Rücksprungadresse vom RETURN-Speicher)			
POS	POS	POS	GZF
PRINT	PRINT	PRINT	B
PRINT#	n.v.	PRINT#	IOB
n.v.	PR#	(CMD dateinummer)	IOB
RAD	n.v.	n.v.	B
RANDOMIZE	RND(neg. Zahl)	RND(neg. Zahl)	B
READ	READ	READ	B
RELEASE	n.v.	n.v.	MB
REM	REM	REM	B
REMAIN	n.v.	n.v.	MB
RENUM	n.v.	n.v.	SB
RESTORE	RESTORE	RESTORE	B
(Bei commodore und apple kann der DATA – Pointer nicht auf eine Zeile setzen)			
RESUME	RESUME	n.v.	B
RETURN	RETURN	RETURN	B
RIGHT$	RIGHT$	RIGHT$	SF
RND	RND	RND	FKF
n.v.	ROT = nA	n.v.	GB
(Setzt Winkeldrehung für Form, die mit DRAW und XDRAW gezeichnet wird)			
RUN	RUN	RUN	SB
RUN„name"	RUN name (nur DOS)	n.v.	SB
SAVE	SAVE	SAVE	SB
n.v.	SCALE	n.v.	GB
(Setzt Maßstab für Form, die mit DRAW und XDRAW gezeichnet wird)			

Schneider CPC464	apple II	commodore 64	Typ
(TEST)	SCRN	n.v.	GZF
(Gibt Farbstiftnummer für eine Position bei niedriger Auflösung)			
SGN	SGN	SGN	GZF
n.v.	SLOAD	n.v.	SB
(Lädt die Daten zu einer Form von Kassette unterhalb des Interpreters; zu ersetzen durch Kombination von LOAD und MEMORY)			
SIN	SIN	SIN	FKF
SOUND	n.v.	n.v.	MB
SPACE$	n.v.	n.v.	SF
	(FOR I=1 TO 10:A$=A$+CHR$(32):NEXT I)		
SPC	SPC	SPC	SF
SPEED INK	n.v.	n.v.	B
SPEED KEY	n.v.	n.v.	B
SPEED WRITE	n.v.	n.v.	SB
n.v.	SPEED=	n.v.	SB
(Setzt Ausgabegeschwindigkeit für Bildschirm fest)			
SQ	n.v.	n.v.	GZF
SQR	SQR	SQR	FKF
n.v.	n.v.	STATUS	GZF
(Systemvariable, in der der Status von I/O Operationen steht)			
STOP	STOP	STOP	B
n.v.	STORE	n.v.	IOB
(Speichert Inhalt einer Tabelle auf Band)			
STR$	STR$	STR$	SF
STRING$(a,b)	(FOR i=1 TO a:a$=a$+CHR$(b):NEXT)		SF
SYMBOL	n.v.	n.v.	B
SYMBOL AFTER	n.v.	n.v.	B
CALL	SYS	CALL	B
TAB	TAB	TAB	SF
TAG	n.v.	n.v.	GB
TAGOFF	n.v.	n.v.	GB
TAN	TAN	TAN	FKF
TEST	n.v.	n.v.	GZF
TESTR	n.v.	n.v.	GZF
n.v.	TEXT	n.v.	GB
(Schaltet den Bildschirm auf Textmodus)			
TIME	n.v.	TIME	FKF
n.v.	n.v.	TIME$	SF
(Gibt die Laufzeit seit Anschalten in SSMMSS [std,min,sec] an)			

Schneider CPC464	apple II	commodore 64	Typ
TRON	TRACE	n.v.	SB
TROFF	NOTRACE	n.v.	SB
UNT	n.v.	n.v.	GZF
UPPER$	n.v.	n.v.	SF
n.v.	USR	USR	FKF
(Übergibt einer Maschinenroutine einen Parameter. Die Auswertung des Parameters ist danach im Fließkomma-Akkumulator enthalten.)			
VAL	VAL	VAL	FKF
(CAT)	n.v.	VERIFY	SB
(Vergleicht den Speicherinhalt mit der Kassette)			
n.v.	VLIN a,b AT c	n.v.	GB
(Zeichnet in der niedrigen Auflösung eine Senkrechte in Spalte c von a bis b)			
OS	PEEK(37)	PEEK(201)	GZF
siehe LOCATE)	VTAB	n.v.	B
AIT	WAIT	WAIT	B
n.v.	XDRAW	n.v.	GB
(Zeichnet eine Form an den angegebenen Koordinaten)			



Erklärung der Abkürzungen:

B	Befehl
FKF	Fließkommafunktion
GB	Graphikbefehl
GKF	Ganzzahlfunktion
IOB	Ein-/Ausgabebefehl
MB	Musikbefehl
O	Operator
SB	Systembefehl
SF	Stringfunktion

Farbtabellen

Apple (niedrigauflösende Graphik)

0 schwarz	4 dunkelgrün	8 braun	12 grün
1 magenta	5 grau	9 orange	13 gelb
2 dunkelblau	6 mittelblau	10 grau	14 blaugrün
3 purpur	7 hellblau	11 rosa	15 weiß

Apple (hochauflösende Graphik)

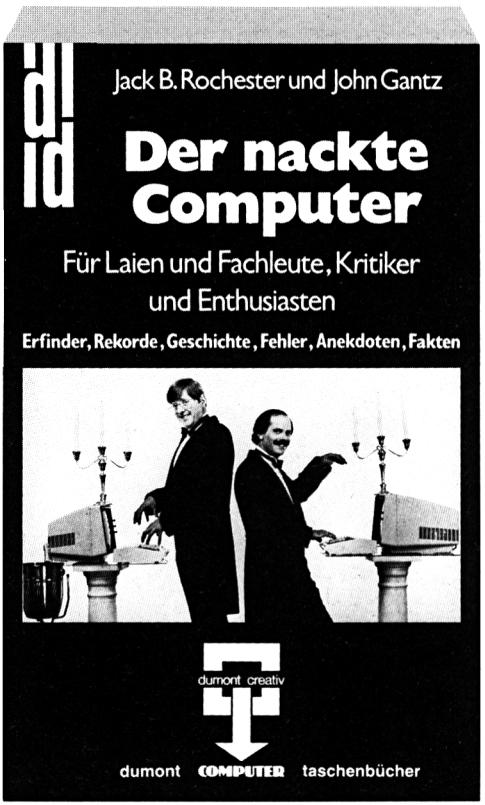
0 schwarz	2 violett	4 schwarz	6 blau
1 grün	3 weiß	5 orange	7 weiß

Commodore

0 schwarz	4 violett	8 orange	12 grau 2
1 weiß	5 grün	9 braun	13 hellgrün
2 rot	6 blau	10 hellrot	14 hellblau
3 türkis	7 gelb	11 grau 1	15 grau 3

Die Traum-Maschine

Eine Einführung in das Wunderland der Computer
Vom Spatzenhirn zur Großrechenanlage. Harte Chips und softe Snacks. Für Laien und Fachleute, Liebhaber und Super-Profis
Herausgegeben von Steve Ditlea. 382 Seiten mit 286 einfarbigen Abbildungen und vielen Listings, kartoniert
Dieses Buch, in Amerika inzwischen zum Computer-Cult-Buch avanciert, präsentiert alle erdenklichen Aspekte jener ›Traum-Maschinen‹ – von den Ursprüngen des Computers bis zu den Micros von morgen. Die Top-Journalisten der Branche melden sich zu Wort. Von den Veteranen des Silicon-Valley bis zu den ›Wunderkindern‹ der neuesten Generation – Exotisches und Spektakuläres –, aber auch handfeste Informationen . . .



Der nackte Computer

Die »Wunderwelt« der großen und kleinen Superhirne
Für Laien und Fachleute, Kritiker und Enthusiasten
Von Jack B. Rochester und John Gantz. 384 Seiten mit 40 einfarbigen Abbildungen, Tabellen, Übersichten, Index, kartoniert (DuMont Taschenbücher, Band 158)

»Es macht Spaß, soviel Klatsch und Tratsch um eine eigentlich seelenlose Maschine zu lesen, ein Buch für Neugierige, nicht nur für Computer-Freaks.« *Buchmarkt*

»Die beiden Autoren haben ein Buch zusammengestellt, das auf unterhaltsame Weise die Geschichte des Computers erzählt.« *Wetzlarer Neue Zeitung*

DuMont's Basic Kochbuch

Ein Lehr- und Lernbuch der Computerprogrammierung für alle Computer von Apple bis Commodore, von Dragon bis IBM

Von Andreas Werminghoff. 160 Seiten mit zahlreichen einfarbigen Abbildungen, Listings, Lehr- und Fehlerprogrammen, kartoniert

»Bedienungstips und Programmierungsbeispiele werden so verständlich dargeboten, daß selbst ein computerunerfahrener Leser den Eindruck gewinnt, alles das sei erlernbar. Zumal die Zwischentexte ausgesprochen witzig formuliert sind.« *Rheinischer Merkur*

»... logischer Aufbau, begreiflich auch für den technischen ›Idioten‹; läßt erkennen, daß ›Computern‹ faszinierend ist.« *Schöner Wohnen*

DuMont's Basic Kramkiste

Nützlich für Apple, Commodore 64, Dragon 32, IBM PC/Peanut und verwandte Basic-Dialekte

Von Andreas Werminghoff und Christoph Silex. 139 Seiten mit zahlreichen einfarbigen Abbildungen, Listings, kartoniert

Das Buch enthält Computerprogramme für Spiele und Abenteuer, Superprogramme, strategische Sandkastenspiele für Business und Vergnügen, Listings und Utilities, Kurzroutinen für Graphik, Musik und Geräusche sowie den Schriftgenerator. Alle Programme sind an den entsprechenden Stellen mit alternativen Befehlen anderer Basic-Versionen versehen.

Der im Herbst 1984 der Öffentlichkeit vorgestellte Heimcomputer Schneider CPC 464 hat größtes Aufsehen erregt. Zum erstenmal gibt es einen Computer mit diesen Fähigkeiten zu einem erschwinglichen Preis. Seine Freundlichkeit und die Leichtigkeit seiner Bedienung laden gleichermaßen zum Spielen wie zum Arbeiten ein: Allerdings ist das Angebot an fertigen Programmen so gering, daß der Käufer zum großen Teil auf eigene Programme angewiesen ist. Wie aber soll der stolze Besitzer und Benutzer eigene Programme schreiben, wenn die beigelegte Bedienungsanleitung nur rudimentäre Informationen über den Computer liefert? Wer dann noch Schwierigkeiten mit der Programmiersprache Basic hat, kann nur noch mit hungrigen Augen um seinen Computer herumstehen, aber nichts damit anfangen.
Um ihm dieses Schicksal zu ersparen und dem Leser eine vergnügliche Einführung in das »Wesen« des Schneiders zu geben, hat sich der Autor liebevoll mit dem Gerät auseinandergesetzt: »DuMont's Handbuch zum Schneider CPC« bringt dem Anfänger wie dem Profi den Schneider nahe und zeigt, daß dieser Computer mehr Möglichkeiten bietet, als seine Betriebsanleitung auch nur ahnen läßt.
Der Autor, geübt durch seine langjährige Lehrerfahrung, zeigt auf leicht verständliche Weise, was der Benutzer mit diesem Computer anfangen kann; dabei führt er anhand von Programmbeispielen in die Eigenheiten des Computers ein. Die Neuheiten des Schneider-Basics, die die Fachwelt erstaunten, liegen vor allen Dingen im Bereich der Zeitgeber-Steuerung und der Tonausgabe. Die entscheidend wichtige Möglichkeit, an einem Computer dieser Preisklasse mehrere Arbeiten gleichzeitig erledigen zu können (Multitasking) sowie die erstaunliche Graphik können erst dann ausgenützt werden, wenn man die in diesem Buch präzise dargestellten Begriffe wie »Interrupt-Steuerung« und »Logische Verknüpfung« versteht und begreift, welche Vorgänge im Computer sich dahinter verstecken. Im Anhang finden sich dann *Programmierhilfen* für die eigene Arbeit und Basic-Vergleichstabellen zu Apple und Commodore: eine nützliche und gebrauchsfreundliche Einrichtung, die erlaubt, durch eigene Programmübersetzungen das geringe Software-Angebot des Schneiders auszugleichen
So ist »DuMont's Handbuch zum Schneider CPC« zugleich eine *leichtverständliche zusammenfassende Einführung,* ermöglicht darüber hinaus aber, die verblüffenden Fähigkeiten und Kapazitäten dieser Maschine überhaupt zu begreifen und zu nutzen.

Andreas Werminghoff, geboren 1945 in Dresden, studierte in Köln Jura, Afrikanistik, Völkerkunde und Musikwissenschaft. Nach einem Abstecher in die freie Kunst (Konstruktivismus) Ausbildung als Elektroniker in Hamburg; Programmierschulung. Danach Mathematikdozent an einer Privatschule. Gründete mit anderen die Basic-Schule VRIL und ist dort seitdem als EDV-Pädagoge und Programmierer tätig.

Bei DUMONT ist von Andreas Werminghoff erschienen:
DuMont's Basic Kochbuch (16127)
und zusammen mit Christoph Silex:
DuMont's Basic Kramkiste (16135)
DuMont's Commodore 64 Kramkiste (16631)

ISB N 3-7701-1724-7

DuMont's Handbuch zum ›Schneider CPC‹

Andreas Werminghoff

COMPUTER dumont creativ

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/