

KLEINCOMPUTER



KC 85/3

BASIC - Handbuch

KLEINCOMPUTER

KC 85/3

BASIC – Handbuch

veb mikroelektronik
»wilhelm pieck«
mühlhausen

im veb kombinat mikroelektronik

Ri 3/88 WV/6/1-10 71506
Gesamtherstellung: Druckerei August Bebel Gotha

veb mikroelektronik „wilhelm pieck“ mühlhausen

Ohne Genehmigung des Herausgebers ist es nicht gestattet, das Buch oder Teile daraus nachzudrucken oder auf fotomechanischem Wege zu vervielfältigen.

INHALTSVERZEICHNIS

Zur Einführung	Seite 5
Grundlagen	
Kapitel 1 : Start des Basic-Interpreters (CAOS-Anweisungen BASIC, REBASIC, Anweisung BYE)	6
Kapitel 2 : Der KC 85/3 als Taschenrechner (Anweisung PRINT, LET, CLEAR, CLS)	10
Kapitel 3 : Wir programmieren schon 19 (Anweisungen RUN, GOTO, LIST, CONT)	
Kapitel 4 : Kleine Rechenprogramme (Anweisungen REM, NEW, INPUT, LINES)	25
Kapitel 5 : FOR-Schleifen und die große PRINT- Zusammenfassung (Anweisung FOR...TO...STEP...NEXT...)	30
Kapitel 6 : Vergleichsoperationen (Anweisungen IF...THEN..., ELSE, END, logische Operatoren)	35
Erweiterungen	
Kapitel 7 : Ein Kapitel Mathematik (mathematische Funktionen, Anweisungen DEF, RANDOMIZE, RND-Funktion)	41
Kapitel 8 : Weitere Programmierhilfen (Anweisungen EDIT, DELETE, KEY, KEYSLIST, AUTO, RENUMBER)	48
Kapitel 9 : Retten und Laden (Anweisungen CLOAD, CSAVE, BLOAD)	55
Kapitel 10 : Farbe (Anweisungen PAPER, INK, COLOR)	60
Kapitel 11 : Graphik (Anweisungen PSET, PRESET, CIRCLE, LINE, Funktion PTEST)	63
Kapitel 12 : Bildgestaltung (Anweisungen WINDOW, LOCATE, WIDTH, Funktionen AT, TAB, SPC, CSRLIN)	71

INHALTSVERZEICHNIS

	Seite
Kapitel 13 : Zeichenvorrat (Funktionen ASC, CHR\$)	78
Kapitel 14 : Strings (Stringoperationen, Stringfunktionen LEN, VAL, STR\$, LEFT\$, MID\$, RIGTH\$, STRING\$, VGET\$, INSTR, INKEY\$)	81
Kapitel 15 : Pause (Anweisung PAUSE)	85
Kapitel 16 : Unterprogrammtechnik (Anweisungen GOSUB, RETURN)	87
Kapitel 17 : Mehrfache Programmverzweigungen (Anweisung ON . . . GOTO, ON . . . GOSUB)	90
Kapitel 18 : Eingabe von mehreren Daten (Anweisungen DATA, READ, RESTORE)	94
Die letzten Tricks	
Kapitel 19 : Programmablaufschwierigkeiten? (Anweisungen TRON, TROFF, STOP)	98
Kapitel 20 : Musik (Anweisungen SOUND, BEEP)	104
Kapitel 21 : Variablenfelder (Anweisungen DIM, CSAVE*, CLOAD*)	109
Kapitel 22 : Innenleben des Computers (Anweisungen FRE, CLEAR, PEEK, DEEK, POKE, DOKE, VPEEK, VPOKE, CALL, SWITCH, Funktion USR)	113
Kapitel 23 : Arbeit mit der Peripherie (Anweisungen OPEN, CLOSE, PRINT#, LIST#, INPUT#, LOAD#, NULL, INP, OUT, WAIT, JOYST, Funktion POS)	123
Kapitel 24 : Lösungen	129
Sachwortregister	133

ZUR EINFÜHRUNG

Ihr BASIC-Interpreter ermöglicht es Ihnen, sich mit Ihrem KC 85/3 fast umgangssprachlich zu unterhalten. Er realisiert also ein Dolmetscher-Programm zwischen der Maschinensprache, die der Computer versteht, und BASIC, einer Sprache, die sich stark an Englisch anlehnt.

Warum programmieren wir nicht gleich in Maschinensprache? Die Antwort ist einfach. Wir programmieren in einer höheren Programmiersprache, nämlich konkret in BASIC, weil diese für uns viel leichter verständlich ist. BASIC-Anweisungen, wie z.B. PRINT, INPUT, NEXT oder GOTO, sind doch entschieden einprägsamer als Maschinencodes, wie z.B. E5, C9, 7F oder 88.

BASIC beherrschen über 70% der in der Welt vorhandenen Mikrocomputer, d.h. Sie können alle BASIC geschriebenen Programme in Ihrem Computer mit nur geringen Veränderungen eingeben und nutzen. Der Name BASIC ist die Abkürzung für "Beginners All-Purpose Symbolic Instruction Code", übersetzt eine "Universelle Programmiersprache für Anfänger". Das "Anfänger" bezieht sich jedoch nur auf die leichte Erlernbarkeit der Sprache, denn mit den komfortablen BASIC-Anweisungen können nicht nur Anfänger etwas anfangen.

Mit Hilfe des BASIC-Handbuches erlernt man die Sprache sehr leicht. Die Übersichten enthalten auch eine Zusammenfassung der BASIC-Anweisungen und dienen als Nachschlagewerk im täglichen Umgang mit dem Computer.

GRUNDLAGEN

KAPITEL 1

Start des BASIC-Interpreters
(CAOS-Anweisungen BASIC, REBASIC,
Anweisung BYE)

KALTSTART/WARMSTART

Wie Sie bereits erfahren haben, wird uns der BASIC-Interpreter das Programmieren stark erleichtern. Um den Interpreter zu nutzen, müssen wir diesen starten.

Es stehen zwei Möglichkeiten zur Verfügung. Der Kaltstart wird durch die CAOS-Anweisung BASIC und der Warmstart durch die CAOS-Anweisung REBASIC ausgeführt. Beide Anweisungen finden wir im Anweisungs Menü des Betriebssystems. Setzen wir nun den Cursor z.B. auf die Anweisung BASIC und betätigen die ENTER-Taste, so erfolgt der Kaltstart des BASIC-Interpreters. Das Betriebssystem wird verlassen und der Interpreter meldet sich mit folgendem Bild:

```
HC-BASIC
MEMORY END?:
```

Nun können Sie den zur Verfügung stehenden Speicherbereich durch Vorgabe der dezimalen Speicherendadresse begrenzen. Wird die Speicherendadresse nicht vorgegeben, so wird der größtmögliche Speicherbereich genutzt. Durch Druck auf die ENTER-Taste wird die Größe des vom Anwender begrenzten oder nicht begrenzten Arbeitsspeichers für BASIC-Programme z.B. in Form von

```
15086 BYTES FREE
```

angezeigt und der Interpreter meldet sich mit den Zeilen

```
OK
>
```

betriebsbereit.

Probieren Sie das erst einmal aus!

15086 BYTES FREE bedeutet, daß uns ein Arbeitsspeicher, der RAM also, von 15086 Byte zur Verfügung steht. Bei diesem Kaltstart werden alle BASIC-Programme und -Daten gelöscht. Man muß jedoch nicht das Anweisungswort BASIC im Menü mit dem Cursor anwählen. Es ist auch möglich, daß Anweisungswort (für den Kaltstart des BASIC-Interpreters BASIC) unterhalb des Menüs noch einmal einzugeben und mit Drücken der ENTER-Taste auszuführen.

Der Warmstart des BASIC-Interpreters kann in gleicher Weise mit Hilfe der Anweisung REBASIC ausgeführt werden. Hierbei bleiben jedoch die BASIC-Programme und -Daten im Speicher erhalten. Diese Anweisung sollte also nur benutzt werden, wenn bereits ein BASIC-Programm im Arbeitsspeicher ist,

welches nicht gelöscht werden soll. Beim Warmstart meldet sich der Interpreter mit:

```
OK
>
```

DIE TASTATUR UND DIE ANWEISUNG BYE

Die wichtigste Taste ist auch beim Betrieb des BASIC-Interpreters die ENTER-Taste (↵) ganz rechts unten auf der Tastatur. Mit dieser Taste werden eingegebene Kommandos ausgeführt und Programmanweisungen gespeichert.

Die Tasten INS, DEL, (←), (→) und (⏪) sind nur bei der Anweisungseingabe und mit Hilfe des Befehls EDIT funktionsfähig. In Kapitel 6 werden wir ausführlich auf diese Problematik eingehen.

Der BASIC-Interpreter besitzt folgende Editiermöglichkeiten:

Tastenbetätigung	Funktion
(←)	Cursor nach links
(→)	Cursor nach rechts
(⏪) (←)	Cursor auf den Zeilenanfang
(⏩) (→)	Cursor an das Zeilenende
(⏪) DEL	Zeile löschen
(⏪) HOME	Bildschirm löschen
DEL	Zeichen löschen
INS	Zeichen einfügen

Nachdem wir den Interpreter gestartet haben, wollen wir uns der Vollständigkeit halber gleich darüber informieren, wie wir diesen gegebenenfalls auch wieder verlassen können. Mit Hilfe der BASIC-Anweisung BYE verabschieden wir unseren Interpreter und befinden uns nach Ausführung des Kommandos wieder im Betriebssystem CAOS. Geben Sie ein:

BYE

Betätigen Sie nun die ENTER-Taste (↵), damit der Befehl ausgeführt wird. Es meldet sich die Kommandoingabe des Betriebssystems.

Nun machen Sie bitte zur Übung einen "Warmstart" aus dem Betriebssystem in den BASIC-Interpreter, wie oben beschrieben.

Sollten Sie einmal eine fehlerhafte Eingabe mit dem KC 85/3-BASIC-Interpreter zur Ausführung gebracht haben, so erscheint auf dem Bildschirm eine

Fehlermeldung (z.B. ?SN ERROR) und es wird vom eingebauten Piezosummer sowie über FBAS- oder RGB- angeschlossene Fernsehgeräte ein akustisches Signal ausgegeben. Die Erklärung zu den Fehlermeldungen, die Sie in der BASIC-Übersicht Teil 7 finden, werden Ihnen eine große Hilfe sein.

WAS MACHEN WIR, WENN WIR UNS VERSCHRIEBEN HABEN ODER "ES EINFACH NICHT WEITERGEHT"?

1. Wir setzen den Cursor zurück und überschreiben bzw. korrigieren mit den bereits erwähnten Editiermöglichkeiten die falsche Stelle oder
2. wir drücken ENTER-Taste.
Führt das nicht zum gewünschten Erfolg, d.h. der Computer reagiert nicht auf Eingabe, so können wir
3. die RESET-Taste drücken (nicht auf der Tastatur, direkt am Computer) und mit einem "Warmstart" (Anweisung "REBASIC") aus dem Betriebssystem in den BASIC-Interpreter zurückkehren. Dabei wird das BASIC-Programm nicht zerstört. Führt das auch nicht zum Erfolg, so werden wir
4. nach nochmaliger Betätigung der RESET-Taste den BASIC-Interpreter mit der Anweisung "BASIC" erneut starten. Hierbei wird jedoch das eventuell vorhandene BASIC-Programm zerstört. Arbeitet der BASIC-Interpreter immer noch nicht einwandfrei, so ist es ratsam,
5. das Gerät aus- und einzuschalten.

Merke:

- "Kaltstart" des BASIC-Interpreters mit der Anweisung " BASIC "
- "Warmstart" des BASIC -Interpreters mit der Anweisung "REBASIC" (Hier bleiben bereits eingegebene Programme erhalten.)
- **Anweisung:** BYE
Format : BYE
Bemerkung : BYE gibt die Steuerung an das Betriebssystem zurück
Beispiel: BYE
- Es ist unmöglich, den KC 85/3 durch falsche Eingaben zu beschädigen. Schlimmstenfalls müssen Sie den Computer erneut einschalten.


KAPITEL 2

Der KC 85/3 als Taschenrechner
(Anweisungen PRINT, LET, CLEAR, CLS)

DIE ENTERTASTE UND DIE PRINT-ANWEISUNG

Unser KC 85/3 kann mehr als wir uns im ersten Moment vorstellen können. Er wartet nur darauf, das auszuführen, was wir ihm anweisen. Das erste und wichtigste Kommando, das wir lernen wollen, ist die Anweisung etwas auf dem Bildschirm darzustellen. Geben Sie ein:

PRINT 2

Nun hat der Computer die Anweisung bekommen, eine "2" auf den Bildschirm zu schreiben und wartet auf den Hinweis, daß er die Anweisung ausführen soll. Diese, wie auch jede andere Anweisung wird durch Betätigen der ENTER-Taste  ausgeführt. Überzeugen Sie sich mit einem Druck auf die ENTER-Taste. Sie sehen, der Befehl wird ausgeführt. Es erscheint die Zahl 2. Geben Sie ein:

PRINT 7+9

(Anweisungsausführung durch ENTER-Taste nicht vergessen!)
Sie erhalten das Ergebnis. Probieren Sie nun selbstständig weiter:

PRINT 7—58

und

PRINT 355+48—66

Vergessen Sie bitte nicht, nach jeder Anweisung die ENTER-Taste zu drücken!

Unser KC 85/3 als Taschenrechner besitzt jedoch auch einige Besonderheiten gegenüber der üblichen mathematischen Schreibweise, auf die wir jetzt näher eingehen wollen.

Wir müssen stets den Buchstaben O und die Ziffer 0 (NULL) unterscheiden. Aus diesem Grund wird die Null durchgestrichen.

Führende Nullen vor dem Komma können weggelassen werden.

Da BASIC sehr viele englische Begriffe enthält, wird auch bei der Zahlendarstellung die englische Schreibweise gewählt. Wir schreiben also einen Dezimalpunkt statt eines Dezimalkommas. Der KC 85/3 arbeitet mit ganzen und reellen Zahlen. Hierbei müssen wir jedoch beachten, daß wir bei der Exponentendarstellung statt „mal zehn hoch" einfach E schreiben.

Nun einige Beispiele:

Mathematische Schreibweise

Computerschreibweise

2,7

2,7

0,16

.16

3,84.10⁻²⁰

3.84E-20

9,02.10¹³

9.02E13

VARIABLEN UND KONSTANTEN

Das Verständnis dieser beiden Begriffe ist die wichtigste Voraussetzung zum Erlernen einer Programmiersprache. Deshalb werden wir uns jetzt anschaulich mit diesen Begriffen vertraut machen.

Konstanten sind festgelegte, unveränderliche Werte. Der KC 85/3-BASIC-Interpreter verarbeitet Zahlen und Zeichenketten als Konstanten. Als Zahlen (numerische Konstanten) stehen ganze Zahlen von -999999 bis +999999 sowie reelle Zahlen mit einem Betrag zwischen $9.40396E-39$ bis $1.70141E+38$ und Null zur Verfügung. Die Zeichenketten-Konstanten, eine Aneinanderreihung beliebiger Zeichen, nennen wir Strings. Sie werden stets von Anführungszeichen eingeschlossen und können höchstens 255 Zeichen lang sein (z.B. "LUTZ").

Variablen, das sagt schon der Name, sind veränderliche Größen. Zum besseren Verständnis stellen wir uns vor, unser Computer besitzt eine Menge von Fächern, die Zahlen oder Worte enthalten können. Den Fächern können wir Namen geben. Diese Namen kennzeichnen unsere Variablen. Der Inhalt eines Faches entspricht dem Wert dieser Variablen.

Die Namen der Variablen stehen also stellvertretend für den momentan in dem Fach vorhandenen Wert.

Veranschaulichen wir uns dies im Bild 1:

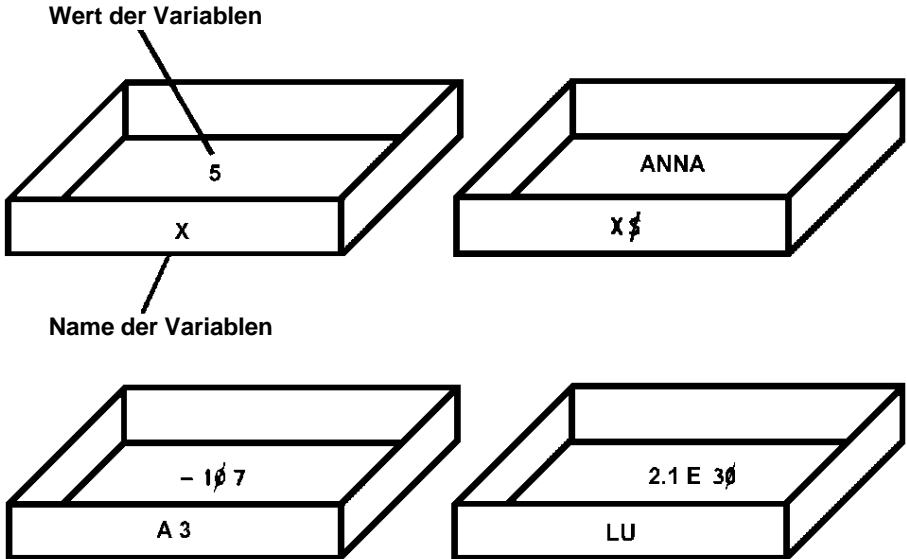


Bild 1 Veranschaulichung des "Gedächtnisses" unseres Computers durch eine Reihe von Fächern

Zum betrachteten Zeitpunkt besitzt

- die Variable X den Wert .5
- die Variable X\$ den Wert "ANNA"
- die Variable A3 den Wert -10.7
- die Variable LU den Wert 2.1E30

Wie wir sehen, können die Werte einer Variablen sowohl Zahlen als auch Strings sein. Deshalb unterscheiden wir numerische Variablen und Stringvariablen.

Der Variablenname ist unter Berücksichtigung folgender Regeln frei wählbar:

1. Ein Variablenname muß immer mit einem Buchstaben beginnen.
z.B. ANNA, LUTZ, WERT, X, A, B7, Typ
2. Am Ende des Namens einer Stringvariablen steht das \$-Zeichen.
z.B. X\$, TYP\$

3. Variablennamen dürfen keine Worte enthalten, die schon mit einer festen Bedeutung in BASIC benutzt werden. Der Variablenname LETTER ist z.B. nicht erlaubt, da er die BASIC-Anweisung LET erhält.
4. Obwohl eine Variable beliebig lang sein kann, unterscheidet der Computer nur die ersten beiden Zeichen einer Variablen. So kann er z. B. die Variablen ROHR und ROSE nicht unterscheiden.

WERTZUWEISUNG MIT LET UND DIE ANWEISUNG CLEAR

Eine Variable können wir mit einem Speicher in einem Taschenrechner vergleichen. Unser Computer bietet jedoch den Vorteil, daß wir sehr viele solcher Speicher anlegen können. Wir geben Ihnen, wie oben beschrieben, Variablennamen.

Mit Hilfe der Anweisung LET schaffen wir uns eine solche Variable und weisen ihr einen Wert zu. Geben Sie ein:

```
LET WURST =3.57
```

(ENTER-Taste nicht vergessen!)

Der Computer hat sich nun ein Fach (besser eine Variable) angelegt, die den Namen WU trägt und dieser den Wert 3.57 zugeordnet. Wir können uns vorstellen, daß dies in der Praxis z.B. die Abspeicherung des Preises für ein Kilogramm Wurst sein kann. Nun löschen wir den Bildschirm mit der Anweisung CLS.

Geben Sie ein:

```
CLS
```

(ENTER-Taste!)

Jetzt ist der Wurstpreis nicht mehr zu sehen, aber wir möchten ihn wieder erfahren. Geben Sie ein:

```
PRINT WURST
```

Die Ausführung dieser Anweisung bewirkt, daß wir den unter der Variablen WU abgelegten Wert, also den Wurstpreis, angezeigt bekommen. Wenn Sie z.B. den Preis von 3,5 Kg Wurst erfahren wollen, so geben Sie ein:

```
PRINT 3,5 * WURST
```

Wie sie bemerken, ist das Zeichen "*" das Multiplikationszeichen. Das Zeichen für die Division ist der Schrägstrich "/". So ermitteln wir den Preis für 1/2 kg Wurst mit der Anweisung:

```
PRINT WURST/2
```

bzw.

```
PRINT .5 * WURST
```

Nun probieren Sie das auch mit anderen Mengen Wurst!

Angenommen, diese Wurst ist ausverkauft und dafür wird eine andere Wurstsorte angeboten. Dann geben wir z.B. ein:

```
LET WURST=3.10
```

Wenn Sie nun schreiben:

```
PRINT WURST
```

So erfahren Sie den neuen Preis.

Da Wertzuweisungen sehr häufig vorkommen, kann die Anweisung LET auch weggelassen werden. Wir schreiben:

```
WURST=3.10
```

Darüber hinaus können wir, wie oben bereits erwähnt, auch andere Variablen festlegen, z.B.

```
W2=2 * WURST
```

```
BROT=0.93
```

```
F6=3.20
```

```
U4=111 * F6
```

(Nach jeder Anweisung die ENTER-Taste drücken!)

Das in den Anweisungen auftretende Gleichheitszeichen ist jedoch inhaltlich nicht dem uns aus der Mathematik bekannten identisch. Schreiben wir z.B. $F6=3.20$, so bedeutet das: Die links stehende festzulegende Variable F6 bekommt den rechts stehenden Wert von 3,20 zugewiesen.

Diese Wertzuweisung darf nie seitenvertauscht erfolgen.

Beim Beginn von Rechnungen ist es üblich, alle nachfolgend verwendeten Variablen zu löschen, d.h. ihnen wird der Wert 0 zugewiesen. An Ihrem Taschenrechner können Sie alle Speicher löschen, indem Sie die C-Taste drücken. Selbstverständlich bietet uns der KC 85/3 den gleichen Komfort. Wir verwenden den Befehl CLEAR. Geben Sie ein:

```
CLEAR
```


Nun überzeugen Sie sich von der Wirkungsweise der Anweisung, indem Sie sich die Werte der vorhin festgelegten Variablen anzeigen lassen.

```
PRINT BROT
```

usw.

Sie werden feststellen, daß die Variablen alle den Wert 0 enthalten.

DIE GRUNDRECHENARTEN

Mit Zahlen und den von Ihnen definierten Variablen können Sie nun nach Belieben rechnen. Geben Sie z.B. ein:

```
PRINT 5 * F6+BROT+WURST/3+15.76
```

Falls Sie die Werte für F6, BROT und WURST noch nicht wieder neu festgelegt haben, werden sie eben als Ergebnis 15.76 erhalten haben. Dies ist auch ganz klar, da wir die Variablen mit der Anweisung CLEAR gelöscht haben.

Weisen Sie nun den Variablen neue Werte zu und berechnen Sie die Aufgabe nochmals. Sie erhalten das richtige Ergebnis. Bei allen Rechnungen ist zu beachten, daß der BASIC-Interpreter ihm übertragene Aufgaben in einer genau festgelegten Reihenfolge verarbeitet, d. h. er gibt bestimmte Operationen Vorrang vor anderen. So weiß er natürlich auch daß Punkt- vor Strichrechnung geht. Das bedeutet, daß er, wie z.B. in der letzten Aufgabe, erst die Multiplikation $5 * F6$ und die Division $WURST/3$ ausführt. Danach wird von links nach rechts addiert bzw. subtrahiert.

Natürlich können Sie bestimmte Operationen durch Klammern der Vorrang geben, z.B.:

```
PRINT 5 * F6+(BROT+WURST)/3+15.76
```

Nun werden Sie ein anderes Ergebnis als vorhin erhalten, da wir Klammern gesetzt haben. Unsere Aufgabe wurde nämlich wie folgt abgearbeitet:

1. Klammerinhalt berechnen
2. Multiplikation und Division ausführen
3. Von links nach rechts addieren bzw. subtrahieren

Um sich zu überzeugen, geben Sie noch folgendes Beispiel ein:

```
A=4+3 * 2:B=(4+3) * 2: PRINT A, B
```

Bei diesem Beispiel sind zwei neue Möglichkeiten angewendet worden: Mit dem Doppelpunkt können mehrere Anweisungen getrennt werden, d.h. es können somit mehrere Anweisungen auf einer Zeile geschrieben werden. Durch die Angabe von mehreren Variablen in der PRINT-Anweisung können mehrere Werte mit einer Anweisung dargestellt werden; es entsteht damit eine sogenannte PRINT-Liste.

Beachten Sie, daß das Multiplikationszeichen "*" auch bei Klammerrechnungen und bei Variablen angegeben werden muß.

Beispiele: $C=3 * (5+7 * A)$ richtig
 $C=3 (5+7 * A)$ falsch
 $C=3 * (5+7A)$ falsch

Merke:

- Anweisung: PRINT
- Besonderheiten der Computerschreibweise
- Variablen, Konstanten, Strings
- Trennen von Anweisungen auf einer Zeile durch Doppelpunkt
- Mit der PRINT-Anweisung können mehrere Werte dargestellt werden
- **Anweisung:** LET
 Format: LET Variable = Ausdruck
 Bemerkung: Die Anweisung weist einer Variablen einen Wert zu. Die Anweisung kann auch weggelassen werden.
 Beispiel: LET X=3 * 7+5 * A
 oder
 $Y=3 * (7+5 * A)$
 oder
 $X\$="ABC 123"$
- **Anweisung:** CLEAR
 Format: CLEAR
 Bemerkung: CLEAR löscht den Variablenspeicher. Weitere Funktionen der Anweisung werden später erläutert
 Beispiel: LET A=4
 PRINT A
 CLEAR
 PRINT A

- **Anweisung:** CLS
- Format: CLS
- Bemerkung: CLS löscht den Bildschirm
- Beispiel: CLS

Übungen

1. Geben Sie ein:

```
PRINT 4.8E0
PRINT 4.8E1
PRINT 4.8E2
.
.
.
PRINT 4.8E9
```

Wir erkennen, daß Zahlen, die länger als sechs Ziffern sind, in wissenschaftlicher Notation (mit Exponenten) dargestellt werden. Probieren Sie gleiches mit negativen Zahlen!

2. Geben Sie ein:

```
CLS
S51=3210
PRINT S51
CLEAR
PRINT S51
```

Beantworten Sie anhand der Übung folgende Fragen:

Was bewirken die Anweisungen CLS und LET?

Worin unterscheidet sich das Gleichheitszeichen einer LET-Anweisung vom mathematischen "="?

Was bewirkt die Anweisung CLEAR?

3. Warum werden die beiden Variablen S51 und S52 in BASIC nicht unterschieden?
4. Geben Sie ein:

```
PRINT 6 * 4 + 5
PRINT 5 + 4 * 6
```

Warum gelangt der Computer (im Gegensatz zu vielen Taschenrechnern) bei beiden Aufgaben zum gleichen Ergebnis?

KAPITEL 3

Wir programmieren schon
(Anweisungen RUN, GOTO, CONT)

TEXT SCHREIBEN

Ein großer Vorteil unseres KC 85/3 gegenüber einem Taschenrechner ist die Möglichkeit der Textverarbeitung.

Geben Sie ein:

```
PRINT "KLEINCOMPUTER AUS MUEHLHAUSEN"
```

ENTER-Taste drücken!

Sie sehen, die Zeichenkette innerhalb der Anführungszeichen wird auf den Bildschirm geschrieben. So können wir beliebige Zeichenketten mit Hilfe der PRINT-Anweisung in Verbindung mit den Anführungszeichen ausdrucken lassen.

DAS PROGRAMM

Wollen wir den obigen Text wiederholt ausgeben lassen, so müssen wir nach eben beschriebenen Verfahren die Anweisung jeweils neu eingeben. Hier kommt uns die Möglichkeit der BASIC-Programmierung entgegen. Schreiben wir also ein Programm, das den obigen Text ausdrückt. Geben Sie ein:

```
10 PRINT "KLEINCOMPUTER AUS MUEHLHAUSEN"
```

Drücken Sie die ENTER-Taste!

Der Computer hat die Anweisung offensichtlich nicht ausgeführt. Das ist richtig, denn mit Betätigen der ENTER-Taste hat er die Anweisung als eine Programmzeile unter der Zeilennummer 10 abgespeichert. Das ist die zweite wesentliche Funktion der ENTER-Taste (Kommandos werden mit Hilfe der ENTER-Taste ausgeführt; Programmzeilen jedoch nur abgespeichert).

Nun wollen wir unser Programm, das aus einer Programmzeile mit einer Anweisung besteht, abarbeiten. Geben Sie ein:

```
RUN
```

Vergessen Sie nicht, die ENTER-Taste zu drücken!

Es erscheint der Text auf dem Bildschirm; der Computer meldet sich mit OK und das Programm ist abgearbeitet. Dieses Programm können Sie nun beliebig oft mit der Anweisung RUN starten und ablaufen lassen. Probieren Sie !

Die Anweisung RUN bewirkt also die Abarbeitung eines Programms. Beginnend mit der niedrigsten Zeilennummer werden die Programmzeilen in aufsteigender Nummerierung abgearbeitet. Hierbei gibt es jedoch Ausnahmen. Eine solche ist z.B. die Anweisung GOTO. Geben Sie ein:

```
20 GOTO 10
```

(ENTER-Taste !)

Und nun lassen wir unser um die Zeile 20 erweitertes Programm mit der Anweisung RUN ablaufen. Der Bildschirm wird vollgeschrieben und danach der Bildschirminhalt von unten nach oben geschoben, wenn Sie den Scroll-Modus gewählt haben. Haben Sie genug davon, drücken Sie die BRK-Taste. Dadurch wird der Programmablauf unterbrochen und der BASIC-Interpreter meldet sich mit

```
BREAK IN 10  
OK  
>
```

(oder BREAK IN 20, je nach dem in welcher Programmeile die Abarbeitung unterbrochen wurde).

Nun löschen wir den Bildschirm mit dem Kommando CLS

Wir möchten uns das Programm wieder ansehen. Geben Sie ein:

```
LIST
```

(ENTER-Taste drücken!)

Wir schauen uns anhand unseres Programms noch einmal an, was eben auf dem Bildschirm passierte. Anweisungszeile 10 wurde wie gehabt ausgeführt. Danach erfolgte die Abarbeitung der Zeile 20. Die Anweisung auf dieser Zeile (GOTO 10) ist eine Sprunganweisung zur Zeile 10. Die Ausführung dieser Anweisung bewirkt, daß der Computer mit der Abarbeitung der Programmzeile 10 fortfährt. Nun wird wieder unser Text ausgedruckt und die nächste Anweisung auf Programmzeile 20, wie eben geschildert, abgearbeitet usw.

Wir wollen unser Programm weiter komplettieren, indem wir den Bildschirm zuerst löschen lassen. Wir geben ein:

```
5 CLS
```

Starten Sie das Programm mit RUN und unterbrechen Sie es nach einer ge-

wissen Zeit durch Druck auf die BRK-Taste. Danach können Sie das Programm fortsetzen. Geben Sie ein:

```
CONT
```

(continue, übersetzt fortsetzen)

Sie merken, das Programm arbeitet weiter. Eine andere Möglichkeit den Programmablauf einmal kurz anzuhalten, um sich die Kontrolle der Aufschrift auf dem Bildschirm anzusehen, ist durch Drücken der STOP-Taste gegeben. Wollen Sie das Programm wieder starten, so betätigen Sie die Taste **⏏** und das Programm wird an gleicher Stelle fortgesetzt, an der es unterbrochen wurde; oder Sie betätigen die BRK-Taste und der BASIC-Interpreter meldet sich mit:

```
BREAK IN . . .  
OK  
>
```

Nun lassen Sie sich das Programm mit Hilfe des Kommandos LIST wieder anzeigen. Sie sehen, daß die Programmzeile 5 entsprechend der Zeilennummer ins Programm eingeordnet wurde. Wir können also nachträglich Programmzeilen einfügen. Es wird deshalb empfohlen, die Programmzeilen in Zehnerschritten zu nummerieren. Sie können dann, wenn es sich später als erforderlich erweist, auch nachträglich noch jeweils bis zu neun Programmzeilen einfügen.


Bei der Anweisung RUN sucht der Computer die Programmzeile mit der niedrigsten Zeilennummer und beginnt dort mit der Abarbeitung des Programms. Es gibt aber auch die Möglichkeit, das Programm auf einer anderen als der niedrigsten Zeilennummer zu starten. Angenommen, wir möchten unser Programm nicht auf der Programmzeile 5, sondern erst auf der Programmzeile 20 starten, so geben wir ein:

```
RUN 20
```

Wir sehen, der Bildschirm wird nicht gelöscht, denn der BASIC-Interpreter arbeitet niemals die Programmzeile 5 ab.

Verschaffen wir uns erst einmal einen Überblick über unsere neuen Erkenntnisse:

- Ein Programm besteht aus Programmzeilen mit Zeilennummern.
- Eine Programmzeile wird durch Betätigen der ENTER-Taste gespeichert.
- Die Programmzeilen werden vom Computer entsprechend ihrer Zeilennummer aufgelistet und abgearbeitet.


- Die Programmzeilen können wir korrigieren, indem wir die entsprechenden Stellen überschreiben oder die Zeile neu eingeben.
- Programmzeilen können wir löschen, in dem wir die Zeilennummer eingeben und die ENTER-Taste drücken
- Ein laufendes Programm können wir durch Betätigen der BRK-Taste unterbrechen, um z.B. irgendwelche Rechnungen durchzuführen, und, sofern wir nichts in unserem Programm geändert haben, mit der Anweisung CONT fortsetzen.
- Ein laufendes Programm können wir auch durch Betätigen der STOP-Taste anhalten, um z.B. einen Ausdruck länger anzusehen zu können, und durch Betätigen der Taste  fortsetzen.

Beachten Sie bitte noch folgende Hinweise:

Die Zeilennummer einer Programmzeile können Sie zwischen einschließlich 0 und 65529 frei wählen. Es besteht auch die Möglichkeit, mehrere durch Doppelpunkt getrennte Anweisungen auf eine Zeile zu schreiben. Eine Zeile darf dabei nur maximal 72 Zeichen lang sein. Da aber nur 40 Zeichen einschließlich der Zeilennummer auf eine Bildschirmzeile passen, wird automatisch ein Überlauf in die nächste Bildschirmzeile realisiert.

Solche Überläufe stören aber das Schriftbild erheblich und insbesondere dann, wenn man in einem Programm nach Fehlern suchen muß. Es ist daher besser, wenn wir eine BASIC-Zeile einer Bildschirmzeile anpassen und nicht mehr als 40 Zeichen pro Zeile verwenden.

Merke:

- Anweisung: PRINT in Verbindung mit einem String bewirkt die Ausgabe des Strings
- Ein Programm besteht aus Programmzeilen.
- Eine Programmzeile besteht aus Zeilennummern (n zwischen 0 und 65529), der Anweisung (spezielle BASIC-Anweisung) und aus eventuell noch weiteren Anweisungen, die durch einen Doppelpunkt voneinander getrennt sind.
- Mit Betätigen der ENTER-Taste werden die Anweisungen der Programmzeilen nicht ausgeführt, sondern die Programmzeilen gespeichert.
- Mit Betätigen der STOP-Taste werden BASIC-Programme angehalten. Sie können an gleicher Stelle mit der Taste  fortgesetzt werden.

- Ein Druck auf die BRK-Taste unterbricht jedes laufende Programm. Die Programmzeile, in der die Unterbrechung stattfand, wird angezeigt.
Beispiel: BREAK IN 20
- **Anweisung:** CONT
Format: CONT
Bemerkung: CONT setzt ein mit BRK-Taste unterbrochenes Programm an gleicher Stelle fort.
Beispiel: BRK-Taste betätigen
:
BREAK IN 20
OK
>
CONT
:
- **Anweisung:** RUN
Format: RUN ENTER-Taste
Bemerkung: RUN startet die Programmausführung bei der niedrigsten oder angegebenen Zeile
Beispiel: RUN oder RUN 20
- **Anweisung:** LIST
Format: LIST [Zeilennummer]
Bemerkung: Das im Speicher befindliche Programm wird beginnend mit der niedrigsten oder der angegebenen Zeilennummer aufgelistet. Die Anzahl der mit einer LIST-Anweisung ausgegebenen Zeilen beträgt vorläufig 10, sie kann auch mit dem LINES-Kommando verändert werden. Das Listen kann man durch Drücken der BRK-Taste unterbrechen.
Beispiel: LIST oder LIST 100
- **Anweisung::** GOTO
Format: GOTO Zeilennummer
Bemerkung: GOTO ist eine unbedingte Sprunganweisung zur angegebenen Zeilennummer
Beispiel: 10 PRINT "HALLO"
20 GOTO 10

Übungen

1. Welche zwei wesentlichen Aufgaben der ENTER-Taste sind zu unterscheiden?
2. Wie viele Anweisungen kann man in eine Programmzeile schreiben?
3. Wie kann man ein BASIC-Programm unterbrechen und wieder fortsetzen?

KAPITEL 4

kleine Rechenprogramme
(Anweisungen REM, NEW, INPUT, LINES)

DIE ANWEISUNGEN REM, NEW, INPUT UND LINES

In diesem Kapitel lernen wir weitere Anweisungen kennen. Danach sind Sie in der Lage, kleine zuverlässige Rechenprogramme selbst zu erstellen. Die ersten drei Anweisungen sind schnell erklärt.

Die Anweisung REM ist abgeleitet vom englischen Wort "remark", das übersetzt Bemerkung heißt. Hinter der Anweisung REM können wir für uns zur Information einen beliebigen Kommentar schreiben. Dieser wird gespeichert und mit der Anweisung LIST auch immer ausgegeben. Im Programmablauf werden die Kommentare jedoch ignoriert, d.h. sie haben nicht den geringsten Einfluß auf die Programmausführung. Die Anweisung ist also nur eine Gedächtnisstütze für den Programmierer.

Ein Beispiel:

```
10 REM DIESES PROGRAMM MACHT NICHTS
20 PRINT "NICHTS"
```

Insbesondere in größeren Programmen ist das Einfügen von Kommentaren sehr nützlich und wird oft angewendet. Sie können für REM auch ein Ausrufezeichen benutzen:

```
10 ! KOMMENTAR
```

Lassen Sie nun das Programm mit dem Kommando RUN ablaufen und sehen Sie es sich mit Hilfe des LIST-Kommandos wieder an!

Geben Sie ein:

```
NEW
```

NEW löscht alle im Speicher befindlichen Programme und Variablen. Der Computer befindet sich nun im gleichen Zustand wie nach dem BASIC-Kaltstart. Überzeugen Sie sich mit Hilfe des Kommandos LIST. Sehen Sie: Keine einzige Programmzeile ist mehr im Speicher zu finden.

RECHENPROGRAMME

Geben Sie bitte ein:

```
10 REM DAS PROGRAMM BERECHNET DAS QUADRAT EINER ZAHL
20 INPUT ZAHL
30 PRINT ZAHL * ZAHL
```

Starten Sie das Programm mit RUN!

Kommt ein Fragezeichen auf Ihrem Bildschirm zum Vorschein, so ist dies keine Fehlermeldung, sondern der Computer ist, nachdem er die Zeile 10 ignoriert hat, zur Abarbeitung der Anweisung INPUT gelangt.

Die Anweisung INPUT unterbricht den Ablauf des Programms und wartet auf eine Eingabe. Geben wir z.B. ein:

3

Mit dem Druck auf die ENTER-Taste wird die Eingabe beendet und der Computer legt den eingegebenen Wert (3) unter der hinter INPUT stehenden Variablen (ZAHL) ab. Von nun kann man den eingegebenen Wert im Programm nach Belieben durch Benutzen der Variablen zu Rechnungen verwenden. So wird in unserem Beispiel in Zeile 30 das Quadrat des eingegebenen Wertes berechnet und ausgegeben.

Schauen wir uns nun ein Programm zur Kreisberechnung an, das nach Eingabe des Radius R den Umfang U und die Fläche F eines Kreises ausgibt. Dazu müssen wir noch wissen, daß in unserem BASIC-Interpreter die Konstante $PI = 3.14159$ bereits gespeichert ist.

Löschen Sie vorher das alte Programm mit NEW und den Bildschirm mit CLS.

```
10 REM KREISBERECHNUNG
20 CLS
30 INPUT R
40 U=2 * PI * R
50 F=PI * R * R
60 PRINT U:PRINT F
```

Probieren Sie das Programm aus!

Nun lassen Sie sich bitte das Programm mit der Anweisung LIST auflisten. Sie können aber auch anders listen. Geben Sie ein:

LINES 3

Nun werden Sie bei der Anweisung LIST nur drei Zeilen erhalten.

Leider können wir im Programmablauf nicht feststellen, für welche Variable wir einen Wert eingeben. Günstiger ist es, wenn sich unser Computer während des Programmablaufes bei der INPUT-Anweisung mit einer Eingabeaufforderung meldet. Geben Sie die Zeile 30 neu ein:

```
30 INPUT "R=";R
```

Lassen Sie nun das Programm ablaufen. Sie sehen, ein in Anführungszeichen gesetzter String direkt nach der INPUT-Anweisung wird im Programmablauf entsprechend ausgedruckt. Dieser nach Belieben einzufügender String ist von der folgenden Variablen in jedem Fall durch ein ";" abzutrennen. Weiterhin ist es auch möglich, mit einer INPUT-Anweisung mehrere, durch Komma getrennte Variablen einzulesen. Auch unsere Ergebnisanzeige können wir noch komfortabler gestalten. Dazu sollten wir wissen, daß mit einer PRINT-Anweisung mehrere, durch Semikolon getrennte Variablen, Zahlen und Strings ausgegeben werden können.

```
60 PRINT "U=";U:PRINT "F=";F
```

Probieren Sie das verbesserte Programm aus!

Analysieren wir wiederholend die Zeile 60.

Der in Anführungszeichen stehende String U= wird unverändert ausgegeben. Das darauf folgende Semikolon bewirkt in Verbindung mit der PRINT-Anweisung, daß die nächste PRINT-Anweisung auf dem Bildschirm unmittelbar folgt. U ist eine Variable, also wird der Wert von U ausgegeben. Der Doppelpunkt trennt die erste von der zweiten PRINT-Anweisung.

Merke:

- Werden die hinter PRINT stehenden Variablen, Zahlen oder Strings durch ";" voneinander getrennt, so erfolgt Ausgabe auf Ausgabe ohne Zwischenraum (bzw. bei Zahlen mit je einem Leerzeichen Abstand hinter der Zahl und einem Vorzeichenfeld vor der Zahl) auf dem Bildschirm.
- Eine Trennung durch Komma bewirkt dagegen ein tabelliertes Ausgabeformat.

● **Anweisung:** REM
 Format: REM Kommentar
 Bemerkung: REM ist eine Kommentarkennzeichnung und wird im Programmablauf ignoriert. Statt REM kann auch "!" verwendet werden
 Beispiel: 10 ! /COM

● **Anweisung:** NEW
 Format: NEW
 Bemerkung: NEW löscht alle im Speicher befindlichen Programme und Variablen
 Beispiel: NEW

- **Anweisung:** INPUT
 Format: INPUT ["String";] Variable [,Variable . . .]
 Bemerkung: INPUT bewirkt eine Anforderung von Daten für das Programm. Es ist möglich, Werte mehreren, durch Kommata voneinander getrennten Variablen mit einer INPUT-Anweisung zuzuordnen. Wird in die INPUT-Anweisung eine Stringkonstante aufgenommen, so wird diese bei der Eingabeanforderung mit ausgegeben; ansonsten erscheint ein Fragezeichen. Erfolgen nicht die vereinbarten Eingaben (z.B. String anstatt numerischer Werte), so meldet sich der Computer "? REDO FROM START" und die Eingabe muß wiederholt werden. Werden zu wenige Daten eingegeben, erfolgt eine wiederholte Eingabeanforderung "??". Werden zu viele Daten eingegeben, erfolgt die Ausgabe "EXTRA IGNORED" und die Programmabarbeitung wird mit den benötigten Werten fortgesetzt.
 Beispiel: 10 REM UMFANG EINES RECHTECKES
 20 INPUT "SEITENLAENGEN"; A,B
 30 U=2 * (A+B)
 40 PRINT U
- **Anweisung:** LINES
 Format: LINES Zahl
 Bemerkung: LINES legt die Anzahl der auf einmal auszugebenden Zeilen beim Listen fest
 Beispiel: LINES 25

Übungen

1. Übernehmen Sie das Beispielprogramm aus der Zusammenfassung zur Anweisung INPUT. Verbessern Sie die Ausgabefähigkeit der Ausgabe, indem Sie die Möglichkeiten einer Stringvariable in Verbindung mit der PRINT-Anweisung nutzen.
2. Erweitern Sie das Programm von Übung 1 so, daß auch die Fläche des Rechtecks ermittelt und ausgegeben wird.
3. Erstellen Sie ein Programm, das aus dem Grundwert und dem Prozentsatz den Prozentwert ermittelt.
 (Prozentwert/Prozentsatz = Grundwert/100)
4. Erweitern Sie das Programm von Übung 3 so, daß auch die Summe vom Grundwert und Prozentwert ausgegeben wird.

KAPITEL 5

FOR-Schleifen
und die große PRINT-Zusammenfassung
(Anweisung FOR. . .TO. . .STEP. . .NEXT)

DIE FOR-NEXT-SCHLEIFE UND EIN NEUER PRINT-KNIFF

Es besteht in der Praxis oft die Notwendigkeit, Programmteile mehrfach, z.B. für die Ermittlung mehrerer Funktionswerte, abzuarbeiten. Das würde sich auf Grund der uns bisher bekannten Anweisungen sehr mühsam gestalten. Wir müßten mit der INPUT-Anweisung Wert für Wert eingeben und das Programm immer neu ablaufen lassen. Für solche Probleme können wir jedoch die FOR-NEXT-Schleife benutzen.

Angenommen, wir möchten uns die Werte der Quadratzahlen von 1 bis 25 ausgeben lassen. Wir schreiben folgendes Programm:

```
10 FOR I=1 TO 25
20 PRINT I*I
30 NEXT I
```

Im einzelnen bedeutet das:

In der Zeile 10 wird eine Variable I vereinbart, die als „Laufvariable“ dient und Werte von 1 bis 25 annimmt.

In der Zeile 20 steht die Druckanweisung für die Größe I^2 .

Der Anweisungsteil in Zeile 30 schließt den Zyklus ab. Falls I kleiner ist als 25, so wird die Laufvariable I jeweils um 1 erhöht. Für $I=25$ bewirkt dieser Befehl ein Verlassen der FOR-NEXT-Schleife.

Die Abarbeitung des Programms wird in der nächsten Zeile fortgesetzt. Lassen Sie das Programm ablaufen!

Das "I" in Zeile 30 können wir auch weglassen. Mit der Anweisung NEXT sucht sich der Computer immer die letzte FOR-Anweisung. Die FOR-NEXT-Schleife erweist sich also als recht praktisch. Wie ist das aber, wenn wir nicht nur ganze Zahlen berechnen wollen, sondern z.B. auch das Quadrat von 1.5, 2.5 usw. benötigen?

Wir verändern die Zeile 10 wie folgt:

```
10 FOR I=1 TO 25 STEP 0.5
```

Hier wurde zusätzlich eine Schrittweite vereinbart. D.h., die Variable I wird nun pro Schleifendurchlauf nicht mehr um 1, sondern um die angegebene Schrittweite 0.5 erhöht. Probieren Sie das Programm mit der veränderten Zeile 10 aus!

Sie werden enttäuscht feststellen, daß die Werte nicht alle auf den Bildschirm passen. Hier gibt es eine kleine Hilfe. Verändern Sie die Zeile 20 wie folgt:

```
20 PRINT I * I,
```

Starten Sie das Programm ! Sie sehen, das Komma tabelliert die Ausgaben in 3 Bereiche zu je 13 Zeichen.

Wir können auch zwei oder mehrere Schleifen ineinander verschachteln. Probieren Sie folgendes Beispiel aus:

```
10 FOR A=1 TO 3
20 FOR I=1 TO 4
30 PRINT "AEUSSERE SCHLEIFE";A;"INNERE SCHLEIFE";I
40 NEXT I
50 NEXT A
```

Die Befehlszeilen 40 und 50 kann man auch zusammenfassen zu:

```
40 NEXT I,A
```

Nach dieser Vereinbarung ist jedoch die Zeile 50 zu löschen! Achten Sie unbedingt auf die Reihenfolge der Variablen – die Variable der innersten Schleife steht als erste hinter NEXT usw.

Merke:

- **Anweisung:** PRINT
 - Format: PRINT [Ausdruck] [Trennzeichen] ...
 - Bemerkung: PRINT ist eine Ausgabeanweisung. PRINT allein bewirkt die Ausgabe einer Leerzeile. Als Ausdruck können der Anweisung folgen:
 1. Konstanten
 2. Variablen
 3. Ausdrücke (Verknüpfungen von Konstanten und Variablen)
 4. Stringkonstanten

PRINT bewirkt die Ausgabe dieser Konstanten, Variablen, Ergebnisse der Ausdrücke und der Strings. Man kann mehrere Ausdrücke hinter einen PRINT-Befehl schreiben. Diese müssen, mit Ausnahme der String-Konstanten, durch Trennzeichen getrennt sein. Die Trennzeichen Komma und Semikolon haben eine Formatierungsfunktion. Gleiches gilt auch für die Platzierung der Ausgabe einer PRINT-Anweisung, wenn die vorhergehende mit einem Komma bzw. Semikolon abgeschlossen wurde. Steht am Ende der zuletzt abgearbeiteten PRINT-Anweisung kein Trennzeichen, so erfolgt die Ausgabe der aktuellen PRINT-Anweisung in der nächsten Zeile. Steht zwischen zwei Ausdrücken ein ",", so erfolgt die Ausgabe tabelliert in 3 Bereiche mit je 13 Zeichen. Sind die Ausdrücke durch ein ";" voneinander getrennt, so folgt Ausgabe auf Ausgabe hintereinander. Handelt es sich bei der Ausgabe um eine Zahl, so wird hinter dieser ein Leerzeichen Abstand zur nächsten Ausgabe gelassen. Vor jeder Zahl finden wir das Vorzeichenfeld. Ist die Zahl positiv, so wird das Vorzeichen wie gewohnt weggelassen und es erscheint somit ein weiteres Leerzeichen vor der Zahl. Handelt es sich bei den durch Semikolon getrennten Ausdrücken um Strings, so werden diese ohne Abstand hintereinander ausgegeben.

Das Anweisungswort PRINT kann auch durch ein "?" eingegeben werden. Dieses wird beim nächsten Auflisten des Programms durch "PRINT" ersetzt.

Beispiel: PRINT 1;—2; "D"; "3"

1Ø INPUT X

2Ø PRINT "DAS QUADRAT VON";X; "IST"; X * X

- **Anweisung:** FOR...TO...STEP...NEXT...
- Format: FOR Variable=Anfangswert TO Endwert STEP Schrittweite
- Bemerkung: Die Anweisung FOR...TO...NEXT erzeugt eine Programmschleife, in der die zwischen diesen Anweisungsteilen stehenden Programmzeilen mehrfach abgearbeitet werden. Beim ersten Durchlauf erhält die Laufvariable den Anfangswert. Nach Abarbeitung des NEXT-Befehls wird der Wert der Laufvariablen um die Schrittweite erhöht. Das wiederholt sich solange, bis der Endwert erreicht ist. Die Schrittweite kann sowohl positiv als auch negativ sein.

Wird keine Schrittweite angegeben, so wird diese automatisch zu +1 gesetzt. In der NEXT-Anweisung kann der Variablenname auch weggelassen werden, da sich diese Anweisung immer auf die davor liegende FOR-Schleife bezieht. Der NEXT-Abschluß mehrerer ineinandergeschachtelter FOR-

Schleifen kann in einer NEXT-Anweisung zusammengefaßt werden. Dabei folgen der Anweisung NEXT von links nach rechts die durch Komma voneinander getrennten Variablen der inneren bis zur äußeren Schleife.

Beispiel: 10 FOR A=8 TO 3 STEP -0.5
 20 PRINT A
 30 NEXT

Übungen

1. Geben Sie ein:

```
10 FOR I=0 TO 25
20 PRINT I
30 NEXT
```

Wie würde sich die Ausgabe verändern, wenn Sie die Programmzeile 20 mit einem "," oder einem ";" beenden würden?

Überprüfen Sie Ihre Antwort am KC 85/3.

2. Verändern Sie die Zeile 20 von Übung 1 wie folgt:

```
20 PRINT I; " " ;
```

(Geben Sie zwischen den beiden Anführungszeichen vier Leerzeichen ein.) Erklären Sie die veränderte Ausgabeformatierung!

3. Verändern Sie das Programm der Übung 2 so, daß es nicht mehr in Einzelschritten sondern in 1/4 Schritten von 0 bis 25 zählt !

4. Probieren Sie folgendes Programm aus:

```
10 FOR I=20 TO 40
20 PRINT "I=";I, "I * I="; I * I
30 NEXT
```

Was würde passieren, wenn wir das Komma in der Zeile 20 durch ein Semikolon ersetzen?

5. Verändern Sie das in Übung 4 aufgelistete Programm so, daß es I^3 in der gleichen Weise ausgibt wie I und I^2 !

KAPITEL 6

Vergleichsoperationen
(Anweisungen IF, THEN, ELSE, END
Logische Operatoren

DIE IF-ANWEISUNG UND DIE LOGISCHEN OPERATOREN

Der KC 85/3 kann für Sie auch Entscheidungen treffen. Selbstverständlich müssen Sie ihm dazu erst die Entscheidungskriterien, d.h. in welchem Fall er was zu tun hat, mitteilen. Dies machen Sie im Programm mit Hilfe der IF-Anweisung. Schauen wir uns dazu ein kleines Beispiel an. Hier soll der Computer eine erteilte Note, die in das Programm eingegeben wird, mit gut oder schlecht bewerten. Dabei werden die Noten 1 und 2 pauschal mit gut und die anderen mit schlecht bewertet. Geben Sie ein:

```

10 INPUT A
20 IF A=1 OR A=2 GOTO 50
30 PRINT "SCHLECHT"
40 END
50 PRINT "GUT"
60 END

```

Mit folgendem Kommentar ist das Programm sofort verständlich.

```

Zeile 20      IF A =1      OR A =2      GOTO      50

```

Übersetzung: Wenn A gleich 1 oder 2 ist, so gehe zur Zeile 50, ansonsten arbeite die nächste Zeile ab.

Testen Sie das Programm!

In diesem Programm wird erstmals die Anweisung END verwendet. Sie schließt den Programmablauf ab und kann prinzipiell an jeder Stelle des Programms stehen. END am Ende eines Programms ist nicht erforderlich, da sich nach Abarbeitung der letzten Programmzeile BASIC automatisch mit "OK" meldet. Die IF-GOTO-Anweisung funktioniert verallgemeinert wie folgt:

```
IF x GOTO n
```

Wenn der Ausdruck x wahr ist, so gehe zur Programmzeile n, ansonsten arbeite nächste Zeile ab.

Nebenbei haben Sie eben Bekanntschaft mit einem der Booleschen Operatoren, dem logischen ODER (Anweisung OR) gemacht. Darüber hinaus verfügt unser BASIC-Interpreter über zwei weitere logische Operationen, das Und und die Negation (Anweisung AND und NOT). Durch geeignete Kombinationen dieser Operatoren können wir weitestgehend alle Probleme der Booleschen Algebra bewältigen. Wir können diese Operatoren nach logischem Ermessen analog der Umgangssprache einsetzen. Hierzu merken wir uns folgende einfache Regel:

Ein Ausdruck der aus einer logischen Verknüpfung (in unserem Beispiel OR) zweier Bedingungen (in unserem Beispiel $A=1$, $A=2$) besteht, ist in folgenden Fällen wahr:

Operator	Übersetzung/Wahrheitskriterium
AND	UND/Beide Bedingungen treffen gleichzeitig zu.
OR	ODER/Mindestens eine von beiden Bedingungen trifft zu.

NOT bezieht sich nur auf einen Ausdruck bzw. auf eine Bedingung, die Negation NOT ist genau dann wahr, wenn dieser Ausdruck falsch ist bzw. die Bedingung nicht zutrifft.

Es gibt auch noch eine zweite Variante der IF-Anweisung. Diese sieht verallgemeinert wie folgt aus:

IF x THEN Anweisung

Das bedeutet inhaltlich: Wenn der Ausdruck x wahr ist, dann führe die hinter THEN stehende Anweisung aus, ansonsten ignoriere die Anweisung und fahre mit der Abarbeitung der nächsten Zeile fort. Statt einer Anweisung kann auch eine Zeilennummer stehen, zu der bei erfüllter Bedingung gesprungen wird. GOTO kann hierbei entfallen. Der Anweisung nach THEN können weitere durch Doppelpunkt getrennte Anweisungen folgen. Diese werden jedoch nur dann ausgeführt, wenn die vorstehende Bedingung erfüllt wurde.

Wir schreiben nun unser Beispielprogramm so um, daß die zweite Variante der IF Anweisung zum Einsatz kommt. Geben Sie ein:

```
10 INPUT A
20 IF A=1 OR A=2 THEN PRINT "GUT"
30 IF NOT (A=1 OR A=2) THEN PRINT "SCHLECHT"
```

Beachten Sie am Beispiel den logisch einfachen, für die Programmierung sehr wichtigen Einsatz der Booleschen Operatoren. Weiterhin haben wir noch die Möglichkeit, eine alternative Anweisung in die IF-Anweisung einzubauen. Dies geschieht mit Hilfe des Anweisungsteils ELSE (übersetzt: ansonsten), der an die uns schon bekannten Formen der IF-Anweisung angehängt wird.

Damit können wir unser Beispielprogramm wie folgt weiter vereinfachen:

```
10 INPUT A
20 IF A=1 OR A=2 THEN PRINT "GUT":ELSE PRINT "SCHLECHT"
```

Beachten Sie, daß vor dem Anweisungsteil ELSE ein Doppelpunkt zu setzen ist.

DIE VERGLEICHSOPERATOREN

Der Interpretier verfügt über folgende Vergleichsoperatoren:

- = gleich
- < kleiner als
- > größer als
- <= kleiner oder gleich
- >= größer oder gleich
- <> ungleich

Die einfache Handhabung dieser Vergleiche veranschaulichen wir uns am besten wieder an einem Beispielprogramm.

Dieses Programm bewertet den Benzinverbrauch von PKW wie folgt:

Benzinverbrauch/100 km Bewertung

weniger als 3l oder mehr als 13l unmöglich

3l und mehr, aber weniger als 8l gut

Von 8l bis 13l schlecht



Bild 2

Geben Sie nun das Programm ein:

```

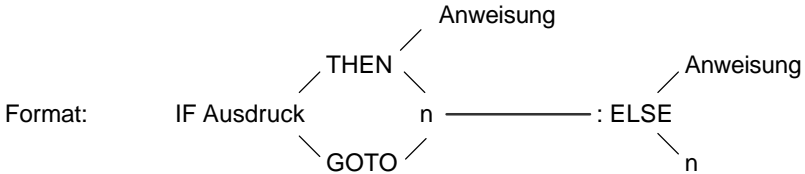
10 INPUT "BEZINVERBRAUCH IN L/100 KM=";A
20 PRINT "BEWERTUNG:";
30 IF A >= 3 AND < 8 GOTO 100
40 IF A >= 8 AND A <= 13 GOTO 200
50 PRINT "UNMOEGLICH":END
100 PRINT "GUT":END
200 PRINT "SCHLECHT"

```

* richtig: 3 AND A < 8 ...

Merke:

- **Anweisung:** IF, THEN
oder
IF, GOTO



Bemerkung: IF ist eine bedingte Anweisung. Ist der Ausdruck wahr bzw. das Resultat des Ausdrucks nicht Null, so wird die Anweisung THEN oder GOTO ausgeführt. Steht hinter THEN eine Anweisung, so wird diese ausgeführt. Steht hinter dem Ausdruck THEN n oder GOTO n, so wird das Programm bei der angegebenen Zeilennummer n fortgesetzt.

Ist der Ausdruck jedoch falsch bzw. das Resultat des Ausdrucks Null, so werden die mit GOTO oder THEN verbundenen Anweisungen ignoriert und die ELSE-Anweisung, falls angegeben, ausgeführt. Andernfalls fährt das Programm mit der Abarbeitung der nächsten Zeile fort. Der Ausdruck kann eine Zeichenkette, eine Variable oder ein arithmetischer Ausdruck sein.

Beispiel:

```

:
.
110 INPUT A$
120 IF A$ = "JA" THEN P = P+2
:
.
    
```

- **Anweisung:** END
Format: END
Bemerkung: Die Anweisung kann an beliebigen Stellen des Programms abgespeichert werden und beendet die Programmabarbeitung.

- Für **Vergleiche** stehen folgende Operatoren zur Verfügung :

```

=          gleich
<         kleiner als
>         größer als
<= bzw. =<  kleiner oder gleich
>= bzw. =>  größer oder gleich
<>        ungleich
    
```


- Die **logischen** oder **Boolschen Operatoren** AND, OR und NOT:

Bei Verknüpfung der Ausdrücke A und B mit den Boolschen Operatoren ergeben sich neue Ausdrücke mit folgenden Wahrheitswerten (wahr=1; falsch=0):

A	B	NOT A	A OR B	A AND B
0	0	1	0	0
0	1	1	1	0
1	0	0	1	0
1	1	0	1	1

Übungen

- Formen Sie das Benzinverbrauchsprogramm analog dem Noten-Programm so um, daß die zweite Variante der IF-Anweisung (IF . . . THEN-Anweisung) zur Anwendung kommt! (Lösung Kapitel 24)
- Wieso ergeben sich für die Ausdrücke A und NOT NOT A die gleichen Wahrheitswerte?
- Wieso ergeben sich für die Ausdrücke NOT (A OR B) und (NOT A) AND (NOT B) die gleichen Wahrheitswerte?
- Die IF-Anweisung wird bei iterativen Näherungsverfahren zum Test der Abbruchbedingungen benutzt.

Schreiben Sie ein Programm zur Berechnung der Quadratwurzel nach Newton. Hierbei ergibt sich

$$X_{i+1} = x_i - \frac{x_i^2 - a}{2x_i}$$

Die Abbruchbedingung lautet:

$$|x_{i+1} - x_i| < 10^{-3}$$

(Lösung Kapitel 24)

ERWEITERUNGEN

KAPITEL 7

Ein Kapitel Mathematik
(mathematische Funktionen, Anweisungen DEF,
RND, RANDOMIZE)

DIE OPERATOREN IN IHRER HIERARCHIE

Als letzte Operation sei die Potenzierung $a = b^e$, in der Computerschreibweise $A = B \wedge C$, genannt. Damit haben wir alle Operatoren unseres BASIC-Interpreters kennengelernt. Beim Umgang mit diesen müssen wir die Reihenfolge ihrer Abarbeitung, d.h. die Hierarchie, in der die Operatoren geordnet sind, beachten. In diese hierarchische Ordnung sind nicht nur, wie in Kapitel 4 beschrieben, die vier Grundrechenarten, sondern alle Operatoren einbezogen. Stehen mehrere Operatoren in einem Ausdruck, so ist die Reihenfolge ihrer Abarbeitung wie folgt hierarchisch geordnet :

1. Klammern
2. Exponenten \wedge
3. Negative Vorzeichen
4. $*$, $/$
5. $+$, $-$
6. Vergleichsoperatoren: $=$, $<$, $>$, $<=$, $>=$, $<>$
7. NOT
8. AND
9. OR

Gleichartige Operatoren werden von links nach rechts abgearbeitet. Da wir diese hierarchische Abarbeitung von außen nicht verfolgen können, versetzen wir uns gedanklich in den Computer und verfolgen an einer Beispielaufgabe die einzelnen Teilschritte der Bearbeitung im Zeitlupentempo.

Aufgabe:	Arbeitsschritte:
3.1 E2 * 2+(6 -5) * 7 -12/3	{ Klammerinhalt ermitteln Exponenten berechnen
3.1 E2 * 2+ 1 * 7 -12/3	
310 * 2+ 1 * 7 -12/3	{ Multiplikation und Division werden von links nach rechts ausgeführt
620 + 1 * 7 -12/3	
620 + 7 -12/3	{ Addition und Subtraktion ebenfalls von links nach rechts
620 + 7 - 4	
627 - 4	
623	

DIE MATHEMATISCHEN FUNKTIONEN

Wie bereits im Kapitel 2 erwähnt, besitzen wir durch den BASIC-Interpreter mit unserem KC 85/3 einen wissenschaftlichen Rechner, der über alle einschlägigen Standard-Funktionen verfügt. Bevor wir zu den konkreten mathemati-

schen Funktionen des Computers kommen, werden wir kurz, aber anschaulich den Begriffs der mathematischen Funktion wiederholen.

Aufbau als Gleichung:

$$y = f(x)$$

Funktionswert
Funktion
Argument

Eine Funktion f ist die Zuordnungsvorschrift, die dem Argument x eindeutig einen Funktionswert y zuordnet.

Am Beispiel: $y = f(x) = \sqrt{x}$
 $f(4) = \sqrt{4} = 2$

$\left| \begin{array}{c} \text{---} \\ \text{---} \end{array} \right.$
 Argument

$\left| \begin{array}{c} \text{---} \\ \text{---} \end{array} \right.$
 Funktionswert

Dabei sind insbesondere immer Beschränkungen des Definitionsbereiches (Bereich der Argumente) und des Wertebereichs (Bereich der Funktionswerte) zu beachten. Diese Beschränkungen resultieren einmal aus der Definition der Funktion selber, zum anderen aus rechentechnischen Gegebenheiten des Computers. Die computerspezifischen Einschränkungen sind in folgender Vorstellung der mathematischen Funktionen des KC 85/3 vermerkt:

Mathematische Standardfunktionen des KC 85/3	Entsprechende mathematische Funktionen	Bemerkung
ABS(X)	x	Betrag von x
ATN(X)	arctan x	Resultat im Bogenmaß
COS(X)	cos x	X im Bogenmaß
EXP(X)	e ^x	X ≤ 87.3365
INT(X)	ganzer Teil von x	
LN(X)	ln x	X > 0
SGN(X)	Signumfunktion	$SGN(x) = \begin{cases} -1 & \text{für } X < 0 \\ 0 & \text{für } X = 0 \\ 1 & \text{für } X > 0 \end{cases}$
SIN(X)	sin x	X im Bogenmaß
SQR(X)	\sqrt{x}	X ≥ 0
TAN(X)	tan x	X im Bogenmaß

Das Argument der Funktion muß stets in Klammern geschrieben werden. Testen Sie nun die uns zur Verfügung stehenden Funktionen ! Beginnen Sie mit dem folgenden einfachen Beispielen:

```
PRINT SQR(81)
```

oder

```
10 INPUT A
20 PRINT SQR(A)
```

etc.

Durch geeignete Kombinationen der Standardfunktionen können Sie alle geläufigen mathematischen Funktionen realisieren. So ergibt sich z.B. der Sekans aus:

$$\sec x = 1/\cos x$$

In der BASIC-Übersicht, im Teil F, finden Sie eine Liste solcher hergeleiteten mathematischen Funktionen.

Der BASIC-Interpreter verfügt über die Konstante mit dem Wert 3.14159. Die Konstante kann durch die Eingabe "PI" genutzt werden:

```
10 INPUT R
20 PRINT "A";PI * R * R
```

FUNKTIONEN DEFINIEREN

Mit der Anweisung DEF FN können wir umfangreiche Formeln (z.B. die eben beschriebenen hergeleiteten Funktionen) als Funktionen definieren und unter einem selbst zu wählenden Funktionsnamen wieder aufrufen. Dies erleichtert die Schreibearbeit insbesondere bei längeren Formeln, die häufig im Programm vorkommen. Der Name der zu definierenden Funktion ist FN, gefolgt von einer zulässigen Variablenbezeichnung. Machen Sie sich anhand des folgenden kleinen Programms mit der Wirkungsweise der Anweisung vertraut :

```
10 DEF FNQ(X)=2 * X * X -X/2
20 INPUT X
30 PRINT FNQ(X)
40 END
```

Hier haben wir die Funktion FNQ (x)=2x² -x/2 definiert. Das Programm berechnet den Funktionswert des einzelnen Argumentes.

ZUFALLSZAHLENBERECHNUNG

Die Funktion RND (X) liefert eine Zufallszahl im Bereich $0 \leq \text{RND}(X) < 1$. Für $X > 0$ erscheinen völlig zufällige Zahlen.

Für $X = 0$ wird der Wert der letzten Zufallszahl noch einmal ausgegeben.

Für $X < 0$ gibt es für jedes X eine bestimmte Zufallszahl.

Außerdem wird der Zufallsgenerator neu initialisiert. Überzeugen wir uns am Beispiel.

Das folgende Programm gibt nach den letzten Erläuterungen also "völlig" zufällige Zahlen aus. Lassen Sie es mehrfach ablaufen !

```
10 FOR I = 1 TO 5
20 PRINT RND(1)
30 NEXT
```

Diese Funktion können wir zum "Würfeln" benutzen. Probieren Sie das Programm einmal mit folgender Veränderung aus.

```
20 PRINT 6 * RND(1)
```

Die fünf Würfelerggebnisse haben nun aber alle die unschöne Eigenschaft, Dezimalstellen hinter dem Komma zu besitzen. Diese können wir beseitigen, indem wir nur den ganzen Teil der Zahl verwenden. Dazu bedienen wir uns der Funktion INT:

```
20 PRINT INT(6 * RND(1))
```

Nach einigen Versuchen ist Ihnen sicher aufgefallen, daß bei unseren Würfelerggebnissen nie eine 6 "gewürfelt" wurde. Verändern Sie deshalb die Zeile 20 ein letztes Mal wie folgt:

```
20 PRINT INT(6 * RND(1))+1
```

Spielen Sie dieses oder anderes Zufallspiel öfter, werden Sie bemerken, daß nach dem Start des BASIC-Interpreters jeweils die gleiche Folge von Zufallszahlen erscheint. Dies liegt in der Funktionsweise des Zufallsgenerators begründet. Beim Start des BASIC-Interpreters erfolgt die Initialisierung des Zufallsgenerators immer mit dem gleichen Wert. Die folgenden Zufallswerte werden ausgehend von diesem Wert stets in gleicher Folge berechnet. Die Wiederholung der stets gleichen Folge von Zufallszahlen können wir durch Einfügen der Programmzeile

```
5 RANDOMIZE
```

beheben. Durch die Anweisung RANDOMIZE erfolgt eine erneute Initialisierung des Zufallsgenerators mit einem zufälligen Anfangswert. Entsprechend den verschiedenen Anfangswerten liefert die RND-Funktion nun auch verschiedene Folgen zufälliger Zahlen.

Merke:

- Stehen mehrere Operatoren in einem Ausdruck, so werden diese der Reihenfolge nach, entsprechend der im Kapitel beschriebenen Hierarchie abgearbeitet.
- Das KC 85/3-BASIC verfügt über folgende mathematische Standardfunktionen: ABS, ATN, COS, EXP, INT, LN, SGN, SIN, SQR, TAN. Das Argument mit einer Funktion muß stets in Klammern stehen.
- Die Konstante ist mit $PI=3.14159$ gespeichert.
- Die Funktion RND erzeugt Zufallszahlen: $0 \leq x < 1$
- **Anweisung:** DEF FN
 Format: DEF FN Name (Verwendetes Argument) Î Ausdruck
 Bemerkung: DEF FN definiert eine vom Anwender bestimmte Funktion. Der Funktionsname besteht aus FN, gefolgt von einem zulässigen Variablennamen. Dem Funktionsnamen folgt in Klammern das Argument der Funktion.
 Beispiel: 10 DEF FNA(Z)=Z*Z-SIN(Z)
 20 INPUT Z
 30 PRINT FNA (Z)
- **Anweisung:** RANDOMIZE
 Format: RANDOMIZE
 Bemerkung: Mit Hilfe der RANDOMIZE-Anweisung wird der Zufallsgenerator neu initialisiert. Wiederholungen von Zufallszahlenfolgen sind damit weitestgehend ausgeschlossen.
 Beispiel: 10 RANDOMIZE
 20 FOR I=1 TO 5
 30 PRINT INT(36*RND(1))+ 1
 40 NEXT I

Übungen

1. Warum hat unser Würfelspiel nach der zweiten Änderung keine 6 "gewürfelt"?
2. Schreiben Sie ein Programm zur Berechnung der längeren Seite c aus den beiden kürzeren Seiten a und b eines rechtwinkligen Dreiecks ABC . (Es gilt der Satz des Pythagoras $c^2 = a^2 + b^2$.) (Lösung im Kapitel 24.)
3. Schreiben Sie ein Programm zur Berechnung der Lösungen x_1 und x_2 der gegebenen quadratischen Gleichung $x^2 + px + q = 0$.

Lösungsvorschrift:
$$x_{1/2} = \frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q}$$

(Lösung Kapitel 24.)

KAPITEL 8

Weitere Programmierhilfen
Anweisungen EDIT, DELETE, KEY,
KEYLIST, AUTO, RENUMBER)

EDIT

Die Anweisung EDIT wird uns in Zukunft eine große Hilfe bei der Korrektur von Programmen sein. Sie haben in Kapitel 5 erfahren, wie man fehlerhafte Programmzeilen überschreiben kann bzw. durch Eingabe der Zeilennummer löscht.

Mit Hilfe der Anweisung EDIT können wir nicht nur die gerade in der Eingabe befindliche Programmzeile, sondern auch bereits abgespeicherte Programmzeilen mit Hilfe der Tasten \leftarrow , \rightarrow sowie der INS und DEL-Taste korrigieren. Probieren Sie das erst einmal an folgendem Beispiel aus:

```
10 INPUT X
20 PRINT "X/2=";X/2
30 PRINT RND(2)
40 PRINT "ENDE"
```

Nach dem Sie das Programm ausprobiert und sicherlich schnell verstanden haben, geben Sie ein:

```
EDIT 20
```

Nun können Sie, wie vorhin beschrieben, innerhalb der Zeile 20 beliebig mit dem Cursor nach links und rechts fahren. Mit Hilfe der Zweitbelegung der Tasten \leftarrow und \rightarrow läßt sich der Cursor am Zeilenanfang bzw. am Zeilenende plazieren. Wir werden die Ausgabe von Zeile 20 noch etwas aussagekräftiger machen, d. h. wir werden die Zeile korrigieren.

... "X/2=" ... wollen wir ändern zu ... "DIE HAELFTE VON X BETRAEGT" ...

Dazu fahren wir mit dem Cursor auf das erste X der Zeile 20 und betätigen 4 mal die DEL-Taste. Wir sehen, das X und die drei nach X befindlichen Zeichen sind gelöscht. Nun geben wir mit Hilfe der INS-Taste so viele Leerzeichen ein, wie Sie zur angegebenen Korrektur Zeichen benötigen. Danach tippen Sie den neuen Text ein. Mit einem Druck auf die ENTER-Taste ist die korrigierte Zeile gespeichert. Betätigen wir dagegen die Taste \leftarrow , so wird die Korrektur nicht übernommen.

In beiden Fällen befinden wir uns nun jedoch in der nächsten Programmzeile, die wir ebenfalls, wie eben beschrieben, korrigieren können. Hier könnten wir z.B. RND(2) in RND(-1) ändern. Der EDIT-Modus wird durch Betätigung der BRK-Taste oder nach Übernahme der letzten Programmzeile verlassen.

Weiterhin können wir mit der Eingabe

```
DELETE x, y
```

Alle Programmzeilen eines Programms von Zeile X bis Zeile Y löschen. Überzeugen Sie sich, indem Sie eingeben:


```
DELETE 20, 30
```

Mit LIST und RUN können Sie nun die Wirkung der Anweisung DELETE 20, 30 kontrollieren.

FUNKTIONSTASTENBELEGUNG

Mit Hilfe der Funktionstasten können wir einzelne Anweisungen oder eine ganze Folge von Anweisungen mit einem Tastendruck eingeben und auch ausführen. Dazu müssen die Funktionstasten jedoch erst mit Hilfe der Anweisung KEY wie folgt belegt werden:

1. Eingabe
KEY Funktionstastenummer
2. ENTER-Taste drücken
3. Eingabe
Tastenfolge, mit welcher die Funktionstaste belegt werden soll
4. STOP-Taste drücken


Als Beispiel werden wir die Taste F1 mit der Anweisung RUN und  belegen, so daß ein im Computer befindliches Programm durch Betätigen dieser Taste gestartet werden kann. Dazu sind folgende Eingaben erforderlich:

```
KEY 1
```

(ENTER -Taste drücken!)

```
RUN 
```

(STOP-Taste drücken!)

Das Steuerzeichen der ENTER -Taste ist im KEY-Eingabemodus einfach durch Drücken der ENTER -Taste einzugeben. Mit Hilfe der eben programmierten Funktionstaste F1 können wir jetzt ein im Speicher befindliches BASIC-Programm mit einem Druck auf dieselbe starten. Damit erreichen wir im Programmierbetrieb eine Arbeitserleichterung von vorher vier (RUN ENTER) zu jetzt einer F1 Tastenbetätigung. Mit Ausnahme der Umschaltfunktion SHIFT , der Editierfunktion CLR und dem Steuerfunktion STOP können wir alle auf der Tastatur in BASIC verfügbaren Steuer- und Editier-funktionen wie am Beispiel der ENTER-Funktion beschrieben, durch die Funktionstasten ausführen. Das

heißt, wir können diese durch Zeichen im KEY-Eingabemodus als Tastenbelegung speichern und später durch Druck auf die entsprechende Funktionstaste ausführen. Die Summe aller Tastenbelegungen darf 143 Zeichen nicht übersteigen.

Nur mit der CLR-Taste kann während der Eingabe der Tastenbelegung korrigiert werden. Bei Betätigung dieser Taste rückt der Cursor um eine Position nach links und löscht das überschriebene Zeichen. Um die Funktionstastenbelegung nicht nach jedem Einschalten des Gerätes erneut eingeben zu müssen, können wir diese auf Magnetband speichern. (SAVE B900 B99B; siehe Systemhandbuch Kapitel 4). Damit steht uns die Funktionstastenbelegung jederzeit nachladbar als Maschinenprogramm zur Verfügung.

Möchten wir uns einen Überblick über alle möglichen Funktionstastenbelegungen verschaffen, so geben wir einfach die Anweisung

KEYLIST

ein und führen sie aus.

Der daraufhin auf dem Bildschirm erscheinenden Liste der Funktionstasten entnehmen wir, daß es 12 mögliche Funktionstastenbelegungen gibt. Nach der oben beschriebenen Eingabe ist die Taste F1 mit "RUN" belegt und die Tasten F2 bis FC sind noch nicht belegt. Die Nummerierung, der letzten drei Funktionstastenbelegungen erfolgt hexadezimal (A, B, C hexadezimal entsprechen 10, 11, 12 dezimal).

Sollen diese Tasten jedoch belegt werden, so sind sie dezimal, also mit "KEY10", "KEY11", bzw. "KEY12" aufzurufen.

Die Ausgabe bzw. Ausführung der Funktionstastenbelegung erfolgt von F1 bis F6 durch Betätigen der entsprechenden Taste bzw. von F7 bis F12 durch die Zweitbelegung der Tasten F1 bis F6.

ZEILENNUMERIERUNG

Beim Programmieren ist es notwendig, vor jede Programmzeile eine Zeilennummer zu schreiben. Mit dem Kommando AUTO kann man den Computer dazu bringen, die Zeilennummerierung selbst durchzuführen, so daß nur noch die Anweisungen selbst eingegeben werden müssen. Der AUTO -Modus wird durch Betätigen der BRK-Taste beendet. Wir können das Kommando AUTO auch durch die Form

AUTO i, j

konkretisieren. Hierbei ist i die erste zu erzeugende Zeilennummer und j legt den Zeilennummernabstand fest.

Testen Sie diese Programmierhilfe an einem kleinen Programm mit mehreren Zeilen!

Nach dem Sie sich nun über die Funktionsweise des Kommandos AUTO am Beispiel informiert haben, geben Sie bitte in das bestehende Programm weitere Programmzeilen ein, so daß möglichst eine "bunte, ungeordnete" Nummerierung entsteht. Eine solche "ungeordnete" Zeilennummerierung ergibt sich in der Praxis oft nach einer notwendigen Korrektur. Hier können wir leicht mit dem Kommando RENUMBER Ordnung schaffen. Geben Sie ein:

RENUMBER

Sie sehen, dieses Kommando bewirkt eine neue Numerierung der Zeilen des Programms in 10er Schritten. Weitere Anwendungsmöglichkeiten des Kommandos RENUMBER entnehmen Sie bitte dem Abschnitt "Merke" dieses Kapitels.


Merke:

- **Anweisung:** EDIT
 Format: EDIT Zeilennummer
 Bemerkung: EDIT bringt die im Anweisungsaufwurf benannte Zeile zur Anzeige und versetzt den Computer in einen Korrekturmodus. In diesem ist es möglich, innerhalb der aufgerufenen Zeile mit Hilfe der Tasten \leftarrow und \rightarrow den Cursor beliebig zu bewegen. Weiterhin kann man durch die Tasten DEL bzw. INS in der Programmzeile Zeichen lückenlos löschen bzw. Leerzeichen, welche man anschließend neu überschreiben kann, einfügen. Mit Hilfe der Taste \downarrow gelangt der Cursor in die folgende Programmzeile und man kann nun in dieser, wie eben beschrieben, korrigieren. Die zu korrigierende Zeilen werden durch Betätigen der ENTER-Taste übernommen. Den EDIT- Modus verlässt man durch die BRK-Taste.

Beispiel: EDIT 20

- **Anweisung:** DELETE
 Format: DELETE Anfangszeile [,Endzeile]
 Bemerkung: Löschen aller Zeilen zwischen einschließlich Anfangs- und Endzeile
 Beispiel 1: DELETE 30
 Beispiel 2: DELETE 30, 100

- **Anweisung:** AUTO
 - Format: AUTO [Zeilennummer] [,Schrittgröße]
 - Bemerkung: AUTO erzeugt automatisch nach jedem Betätigen der ENTER-Taste eine Zeilennummer. Es wird mit der angegebenen Zeilennummer begonnen und diese jeweils um die angegebene Schrittgröße erhöht.
Werden keine Schrittgröße und keine Zeilennummer angegeben, so beginnt AUTO mit der Zeilennummer 10 und nummeriert im 10er Abstand. Stimmt eine erzeugt Zeilennummer mit einer schon im Programm vorhandenen überein, so wird dies durch ein *-Zeichen hinter der Zeilennummer gekennzeichnet.
AUTO wird durch Betätigen der BRK-Taste beendet.
 - Beispiel: AUTO 30,5 erzeugt die Zeilennummern 30, 35, 40, 45, 50 etc.

- **Anweisung:** KEY
 - Format: KEY Funktionstastennummer
 - Bemerkung: Bei Ausführung der KEY-Anweisung im angegebenen Format (Funktionsnummern von 1 bis 12 möglich) wird der Computer in einen Eingabemodus versetzt, in welchem der einzugebende String für die in der KEY-Anweisung benannte Funktionstaste definiert wird. In diesem Eingabemodus können, neben den üblichen alphanumerischen und Sonderzeichen, auch Zeichen für die auf der Tastatur verfügbaren Editier- und Steuerfunktionen durch Druck auf die jeweilige Editier- und Steuertaste eingegeben werden. Die einzige Editiermöglichkeit besteht in diesem Modus in der Betätigung der CLR-Taste. Dabei rückt der Cursor eine Position nach links und das falsche Zeichen kann neu überschrieben werden.
Der Eingabemodus wird durch Betätigen der STOP-Taste beendet.
Mit einem Druck auf die betreffende Funktionstaste wird nun stets der im KEY-Eingabemodus festgelegte String ausgegeben bzw. die darin als Steuer- und Editierzeichen enthaltenen Funktionen ausgeführt.
 - Beispiel: KEY 2 (ENTER-Taste drücken)
WINDOW:COLOR 7,1:CLS:LIST  (STOP-Taste drücken)
OK
>

- **Anweisung:** KEYLIST
 Format: KEYLIST
 Bemerkung: Mit Hilfe dieser Anweisung kann man die Funktionstastenbelegung auflisten.

Beispiel: KEYLIST
 F1 : RUN
 F2 : WINDOW:COLOR7,2:CLS:LIST
 F3 : EDIT
 F4 : INPUT
 F5 : FOR I=
 F6 : NEXT
 F7 : CLS
 F8 : GOTO
 F9 : GOSUB
 FA : RETURN
 FB : INKEY\$
 FC : ?CHR\$

- **Anweisung:** RENUMBER
 Format: RENUMBER [Ab-alte-Zn] [,Bis-alte-Zn] [,Ab-neue-Zn] [Schrittgröße] Zu . . . Zeilennummer
 Bemerkung: RENUMBER numeriert die in einem Programm vorkommenden Zeilen neu. Wird RENUMBER ohne Parameterangabe ausgeführt, so erfolgt die Zeilennummerierung beginnend mit der kleinsten Programmzeilennummer in 10er Schritten. RENUMBER darf nur in Verbindung mit einem BASIC-Programm verwendet werden, da es sonst zu einem modifizierten Zustand des Computers führt. Neustart erfolgt über RESET.

Beispiel: RENUMBER 12, 110, 20, 5
 Numeriert Zeilen 12 bis 110 des alten Programms neu. Dabei wird die Zeile 12 zur angegebenen niedrigsten Zeile 20 der neuen Nummerierung und die weiteren Zeilennummern folgen in der angegebenen Schrittgröße 5.

Übungen

Keine, statt dessen ein Hinweis:

Nutzen Sie diese Programmierhilfe im weiteren konsequent bei jeder sich bietenden Gelegenheit. Nach einer kurzen Eingewöhnungsphase möchten Sie den dadurch gebotenen Komfort bestimmt nicht mehr missen.

KAPITEL 9

Retten und Laden
(Anweisungen CLOAD, CSAVE, BLOAD)

SPEICHERMEDIUM TONBAND

So, wie wir in Maschinensprache geschriebene Programme vom Recorder in den Computer laden, können wir auch in BASIC geschriebene Programme mit Hilfe des Interpreters durch die BASIC-Anweisung CLOAD in den Computer laden.

Da wir nicht nur daran interessiert sind, im Handel erworbene Programme, sondern auch selbsterstellte wiederholt zu nutzen, müssen wir diese vor dem Ausschalten des Gerätes auf unseren Speicher, das Tonband, "retten" (Englisch: save). Die Anweisung, die diesen Vorgang auslöst, heißt deshalb auch CSAVE.

WIR RETTEN EIN PROGRAMM

Die Funktionsweise dieser beiden Anweisungen verdeutlichen wir uns am besten an einem Beispiel. Schreiben Sie ein kleines Programm, das wir dann als Testprogramm retten und laden wollen. Dieses Programm legen wir unter einem selbst gewählten Name auf der Kassette ab. Der Computer unterscheidet die letzten 8 Buchstaben des Namens. Nennen wir unser Programm der Einfachheit halber TEST.

Bevor wir mit der eigentlichen Aufnahme beginnen, ist es ratsam, sich den Zählerstand des Recorders aufzuschreiben oder den Programmnamen mit Mikrofon auf die Kassette zu sprechen. Dies erleichtert das Suchen des Programms erheblich.

Geben Sie nun ein: (Aber noch nicht die ENTER-Taste drücken!)

```
CSAVE "TEST"
```

Schalten Sie den Recorder auf Aufnahme.

Drücken Sie nun die ENTER-Taste.

Nun wird das im Computer gespeicherte BASIC -Programm auf Magnetband aufgezeichnet. Das Auslesen des Programms erfolgt in Blöcken zu je 128 Bytes. Die angezeigten zweistelligen Zahlen stellen die Blocknummern der eingelesenen Blöcke dar. Damit ergibt sich die Länge der Aufnahme, also die Anzahl der eingelesenen Blöcke, aus der Länge des BASIC- Programms.

Nach dem Retten besteht die Möglichkeit des Vergleichs der Magnetbandaufzeichnung mit dem im Computer befindlichen Programm. Dazu erfolgt nach Übernahme des letzten Blockes die Ausschrift:

VERIFY?(Y):

Mit dieser Ausgabe ist die Aufnahme beendet und Sie können den Recorder ausschalten. Wenn Sie den Vergleich durchführen möchten, spulen Sie die Kassette auf den Programmanfang zurück, schalten den Recorder zur Wiedergabe ein und betätigen während des Pfeiftones die Taste "Y". Die eingelesenen Datenblöcke werden nun mit dem Speicherinhalt verglichen. Dabei werden der Programmname und die Nummern der Datenblöcke ausgegeben. Hinter jeder Daten-blocknummer erscheint ein ">". Dieses Zeichen bestätigt die fehlerfreie Übernahme des Blockes.

Erscheint jedoch hinter der Blocknummer ein "?", so ist dieses als Fehlermeldung zu werten. Es trat also ein Fehler bei der Datenübernahme auf.

Bei einem solchen Fehler können Sie nach dem Zurückspulen des Bandes den fehlerhaft gelesenen Block erneut einlesen. Dabei wird der Computer unverändert im VERIFY-Modus belassen. Liest man beim erneuten Lesen Blöcke ein, die vor dem fehlerhaften eingelesenen Block bereits richtig gelesen wurden, so wird dies durch ein "*" verdeutlicht. Dabei entsteht jedoch kein Schaden.

Wird ein Datenblock wiederholt falsch eingelesen, so ist ein Datenaufzeichnungsfehler zu vermuten, der eventuell durch stellenweise unzureichende Qualität des Magnetbandes verursacht wurde.

In einem solchen Fall wiederholen Sie die Aufzeichnung auf einem anderen Magnetbandabschnitt.

Hinweis:

Magnetbänder mit schadhafte Stellen wie z. B. Knitter- oder Klebestellen sind nicht zu verwenden !

Der Vergleich ist mit der Ausschrift OK abgeschlossen und der Recorder kann ausgeschaltet werden. Möchten Sie das gerettete Programm nicht überprüfen, wird nach Erscheinen der Abfrage VERIFY?(Y): die ENTER-Taste betätigt, wobei als Enderkennung OK ausgeschrieben wird. Nun hören Sie sich die Aufnahme am besten einmal über Lautsprecher an. Spulen Sie das Band zurück und spielen Sie es ab. Am Anfang werden Sie gegebenenfalls den von Ihnen gesprochenen Programmnamen hören. Danach hören Sie einen Pfeifton. Das ist der Vorton. Spätestens jetzt müssten Sie, wenn Sie das Programm wieder vom Recorder in den Computer laden wollten, die ENTER-Taste drücken. An den nachfolgenden schrillen Tönen erkennen Sie das eigentliche Programm.

WIR LADEN EIN PROGRAMM

Löschen Sie nun den Arbeitsspeicher mit NEW und überzeugen Sie sich mit LIST, daß unser Testprogramm nicht mehr im Computer gespeichert ist. Spulen Sie die Kassette an den Programmanfang zurück und geben Sie in den Computer ein: (ENTER-Taste noch nicht drücken !)

```
CLOAD "TEST"
```

Schalten Sie nun den Recorder zur Wiedergabe ein. Sobald die Ansage beendet ist oder der pfeifende Vorton beginnt, drücken Sie auf die ENTER-Taste. Nun wird das Programm von der Kassette in den Computer geladen. Angezeigt werden wieder die Blocknummern. Bei fehlerhaftem Lesen eines Blockes – "?" hinter der Blocknummer – ist ein erneutes Einlesen des fehlerhaften Blockes erforderlich. Nach Beendigung des Ladevorganges meldet sich der Computer mit:

```
FILE FOUND
```

Nun ist das Programm wieder vollständig im Arbeitsspeicher des Computers vorhanden. Überzeugen Sie sich !

Sollte sich beim Einlesen ein Fehler eingeschlichen haben, so wiederholen Sie bitte den Ladevorgang. Beachten Sie auch, daß bei einigen Recordertypen ein einwandfreies Abspeichern und Laden von der richtigen Einstellung des Aufnahme- und Lautstärkereglers abhängig ist.

Mit der Anweisung BLOAD haben wir darüberhinaus die Möglichkeit, Maschinenprogramme (z.B. eine Zeichenbildtabelle) vom BASIC-Interpreter aus zu laden.

Die Anweisung ist wie die Systemanweisung LOAD (siehe Systemhandbuch Kapitel 2) zu handhaben.

Merke:

- **Anweisung:** CSAVE
Format: CSAVE "NAME"
Bemerkung: CSAVE "NAME" speichert das im Arbeitsspeicher befindliche Programm unter dem angegebenen Namen auf der Kassette ab. Es sind maximal die letzten 8 Zeichen des Namens signifikant.
Beispiel: CSAVE "TEST"

- **Anweisung:** CLOAD
Format: CLOAD "NAME"
Bemerkung: Die Anweisung lädt das angegebene Programm von der Kassette und legt es hinter dem schon im Speicher befindlichen Programm ab.
Beispiel: CLOAD "TEST"
- **Anweisung:** BLOAD
Format: BLOAD
Bemerkung: Die Anweisung ruft das Betriebssystemprogramm LOAD zum Einlesen von Maschinenprogrammen während der Arbeit des BASIC-Interpreters auf.
Beispiel: BLOAD
- **Retten eines BASIC-Programmes:**
 1. Programmanfang durch Zählerstand am Recorder merken oder durch akustisches Signal kennzeichnen.
 2. Geben Sie ein:
CSAVE "NAME"
 3. Schalten Sie den Recorder zur Aufnahme ein.
 4. Drücken Sie die ENTER-Taste.
- **Laden eines BASIC- bzw. eines Maschinenprogrammes**
 1. Spulen Sie die Kassette vor den Anfang des zu ladenden Programmes.
 2. Geben Sie ein:
CLOAD "NAME"
bzw. für Maschinenprogramme:
BLOAD
 3. Schalten Sie den Recorder zur Wiedergabe ein.
 4. Drücken Sie vor oder spätestens während des pfeifenden Vortons die ENTER-Taste.

Übungen

Schreiben Sie noch ein oder zwei weitere Programme, die Sie retten und wieder aufnehmen. Testen Sie mit ihrer so entstandenen Mini-Programmbibliothek alle im Kapitel behandelten Anweisungen.

KAPITEL 10

Farbe
(Anweisungen PAPER, INK, COLOR)

FARBEN

Mit Hilfe des KC 85/3 können Sie auf Ihrem Farbfernsehgerät 24 Farben darstellen. Das sind im einzelnen 8 Hintergrundfarben und 16 Vordergrundfarben. Unter der Vordergrundfarben verstehen wir die Farbe der Buchstaben bzw. bei der Graphik die Farbe der gezeichneten Linien und Punkte.

Jeder Vordergrund- und Hintergrundfarbe wird zur Programmierung ein Farbcode in Form einer Zahl zugeordnet. Die konkrete Zuordnung ist der folgenden Farbcode-Tabelle zu entnehmen. Die Vordergrundfarbe können wir blinkend darstellen, indem wir zum entsprechenden Vordergrundfarbcode die Zahl 16 addieren. Dadurch ergeben sich insgesamt 8 Hintergrundfarbcodes (0 bis 7) und 32 Vordergrundfarbcodes (0 bis 31).

Die Farben sind wie folgt codiert:

Vordergrundfarbe	Nummer	Hintergrundfarbe
Schwarz	0	Schwarz
Blau	1	Blau
Rot	2	Rot
Purpur	3	Purpur
Grün	4	Grün
Türkis	5	Türkis
Gelb	6	Gelb
Weiß	7	Grau
Schwarz	8	
Violett	9	
Orange	10	
Purpurrot	11	
Grünblau	12	
Blaugrün	13	
Gelbgrün	14	
Weiß	15	

Dabei sind die Vordergrundfarben deutlich heller als die Hintergrundfarben.

Mit Hilfe der drei Anweisungen PAPER (für Hintergrundfarbe), INK (für Vordergrundfarbe) und COLOR (für Vorder- und Hintergrundfarbe) sowie den Farbcodes lassen sich die umfangreichen Farbmöglichkeiten des KC 85/3 unkompliziert nutzen. Die genaue Handhabung der eben genannten Befehle entnehmen Sie bitte dem folgenden Abschnitt "Merke".

Merke:

- **Anweisung:** PAPER
Format: PAPER h
Bemerkung: Einstellen der Hintergrundfarbe; h . . . Hintergrundfarbcode entsprechend obiger Übersicht
Beispiel: PAPER 3
- **Anweisung:** INK
Format: INK v
Bemerkung: Einstellen der Vordergrundfarbe; v . . . Vordergrundfarbcode entsprechend obiger Übersicht
Beispiel: INK 0
- **Anweisung:** COLOR
Format: COLOR v,h
Bemerkung: Einstellen der Vorder- und Hintergrundfarbe
Beispiel: 250 VF=7:HF=2
260 COLOR VF, HF

● **Farbanweisung innerhalb einer PRINT-Anweisung**

Werden die Farbanweisungen in eine PRINT-Anweisung eingefügt, so bezieht sich die Farbeinstellung nur auf die Ausgabe derselben PRINT-Anweisung. Danach ist der vorherige Farbzustand wieder hergestellt.

Das Einbinden einer Farbanweisung in eine PRINT-Anweisung geschieht in folgender Form:

PRINT Farbanweisung; Ausdruck

Beispiel:

20 INPUT A\$
30 PRINT COLOR 0,4;A\$

- Die **Farbparameter** kann man auch den Übersichten Punkt 6 entnehmen.

Übung

Erstellen Sie ein Programm, das alle möglichen Farbkombinationen von den Hintergrundfarben und den blinkenden Vordergrundfarben ausgibt.

KAPITEL 11

Graphik
(Anweisungen PSET, PRESET, CIRCLE,
LINE, Funktion PTEST)

PUNKTE

Der KC 85/3 bietet Ihnen einen für einen Kleincomputer beispielhaften Graphik-Komfort. Wir sind in der Lage, das Fernsehbild in 320 Punkte (in der Waagerechten) mal 256 Punkten (in der Senkrechten) aufzulösen. Damit stehen uns insgesamt 81 920 Graphikpunkte zur Verfügung. Etwas anschaulicher bedeutet das z.B., daß wir jedes Buchstabenfeld in 64 (8X8) Punkte auflösen können. Dadurch sind Sie in der Lage, mathematische Funktionen oder vom Computer überwachte Prozesse graphisch exakt in geschlossenen Kurvenzügen darzustellen. Des weiteren können Sie Bilder zur Unterstützung selbst-erstellter Programme, Diagramme, Muster und Figuren entwerfen.

Wie zeichnen wir nun konkret?

Jeder einzelne Punkt ist ansprechbar durch seine Koordinaten. Die X-Koordinate gibt den Abstand vom linken Bildschirmrand (von links nach rechts von 0 bis 319) und die Y-Koordinate den Abstand vom unteren Bildschirmrand (von unten nach oben von 0 bis 255) des jeweiligen Punktes an. Vergleichen Sie hierzu bitte Bild 3 auf der nächsten Seite.

Wollen wir nun einen Punkt mit der Koordinate X und Y in einer bestimmten Vordergrundfarbe zeichnen, so geben wir dies in Form der

PSET x-Koordinate, y-Koordinate, Farbcode

ein. Als Farbcode wird der im letzten Kapitel beschriebene Vordergrundfarbcode (0 bis 31) verwendet.

Zeichnen Sie einen weißen Punkt mit den Koordinaten $x=20$ und $y=15$, indem Sie eingeben:

PSET 20, 15, 7

Zeichnen Sie nach Belieben noch weitere Punkte ein.

So, wie wir einen bestimmten Punkt setzen können, ist es auch möglich, diesen mit der Anweisung PRESET zu löschen. Geben Sie ein:

PRESET 20, 15

Und der von Ihnen vorhin gesetzte Punkt ist wieder gelöscht. Probieren Sie beide Anweisungen ruhig eine Weile aus!

Folgendes Programm erzeugt einen kleinen Sternenhimmelausschnitt:

```
10 CLS
20 FOR I=0 TO 30
30 PSET RND(1) * 100=100, RND (1) * 100=100,7
40 NEXT
```

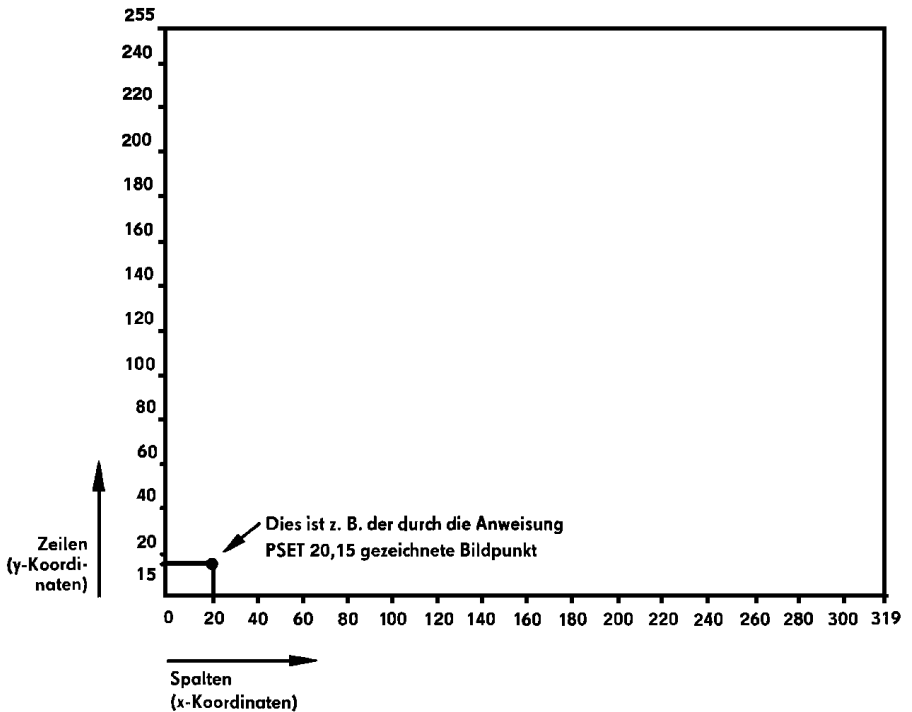


Bild 3: Vollgraphik; Das Anzeigebild setzt sich aus 320×256 Bildpunkten Zusammen

Nun schauen wir uns noch ein kleines mathematisches Beispiel an.
Folgendes Beispiel zeichnet die Sinusfunktion im Bereich von 0 bis 2π .

```

10 PAPER 1: CLS
20 FOR X=0 TO 319
30 Y=128 + 50 * SIN(X/159.5 * PI)
40 PSET X, Y, 7
50 NEXT

```

LINIEN UND KREISE

Wir können jedoch nicht nur einige Punkte setzen oder löschen. Der BASIC-Interpreter verfügt auch über Graphikanweisungen, die es ermöglichen, unkompliziert geometrische Grundfiguren zu zeichnen.

Die Anweisung CIRCLE in der Form

CIRCLE x-Koordinate, y-Koordinate, Radius, Farbcode

zeichnet auf dem Bildschirm einen Kreis mit dem angegebenen Radius, um den Mittelpunkt der angegebenen Koordinaten x und y in der durch den Farbcode festgelegten Vordergrundfarbe. Möchten wir z.B. einen weißen Kreis mit einem Radius von 50 Bildpunkten in die linke untere Ecke des Bildschirms zeichnen, so geben wir ein

CIRCLE 49, 49,50,7

Die Angabe der Mittelpunkts-Koordinaten orientiert sich ebenfalls an der im Bild 3 beschriebenen Graphikpunkte-Gliederung.

Programmieren Sie nach Belieben noch einige weitere Kreise.

Mit der Anweisung LINE in der Form

LINE x_a, y_a, x_e, y_e , Farbcode

Können wir eine Linie auf dem Bildschirm zeichnen. Dabei bezeichnen die Koordinaten x_a und y_a den Anfangs- und die Koordinaten x_e und y_e den Endpunkt der Linie. Der letzte Parameter legt wie bei der Anweisung CIRCLE die gewünschte Vordergrundfarbe fest. Die Anweisung

LINE 0,0,319,255,7

Zeichnet z.B. eine Bildschirmdiagonale, beginnend in der linken unteren Ecke. Mit Hilfe des folgenden kleinen Programms können Sie die Anweisung LINE unkompliziert probieren:

```
10 INPUT "XA,YA,XE,YE,F";XA,YA,XE,YE,F
20 LINEXA,YA,XE,YE,F:GOTO10
```

Als letztes lernen wir in diesem Kapitel die Graphik-Funktion PTEST kennen. Mit Hilfe der Funktion kann man an einem beliebigen Bildpunkt testen, ob dieser die Vorder- oder Hintergrundfarbe trägt.

Diese Funktion ist ebenfalls nur in Verbindung mit einer Anweisung sinnvoll zu gebrauchen.

Als Argument wird die x-Koordinate des zu testenden Bildpunktes in Klammern hinter die Funktion geschrieben. Als aktuelle y-Koordinate setzt der Computer die zuletzt in einer der vier Graphik-Anweisungen erwähnte y-Koordinate ein. Probieren Sie folgendes Beispiel:

```
10 CLS:LINE 50,30,50,100,7
20 PRINT PTEST (50)
```

Nach Ausführung des Programmes werden Sie auf dem Bildschirm, neben der in Zeile 10 festgelegten Strecke als Ergebnis der Anweisung in Zeile 20 die Zahl 1 ablesen können.

In unserem Programm wurde der Punkt (50,100) getestet. Die y-Koordinate 100 ergibt sich, da sie die letzterwähnte y-Koordinate einer Graphik-Anweisung (in Zeile 10) ist. Dieser Punkt ist der oberste der durch das Programm erzeugten Linie und erscheint damit in der Vordergrundfarbe weiß. Ist die Vordergrundfarbe gesetzt, so liefert die Funktion wie wir am Beispiel sehen können den Wert 1. Ist jedoch die Hintergrundfarbe gesetzt, so liefert die Funktion den Funktionswert 0.

Überzeugen Sie sich an folgendem Beispiel:

```
PRINT PTEST (51)
```

Soeben haben wir den Bildpunkt 51,100 rechts neben dem obersten unserer Gerade getestet. Da dieser in der Hintergrundfarbe erscheint, lautet das Ergebnis der letzten Anweisung 0.

Zeichnen Sie sich nun mit folgendem Programm Ihre eigene aufgehende Computer-Sonne:

```
10 COLOR7,1:CLS
20 X=220:Y=50
30 CIRCLE X,Y,30,2
40 FORB=1TO79:PRESET0,B:A=220
50 IF PTEST(A)=0 THEN PSETA,B,2:A=A-1:GOTO50
60 A=221
70 IF PTEST(A)=0 THEN PSETA,B,2:A=A+1:GOTO70
80 NEXT
90 LINEX,Y,0,150,2
100 LINEX,Y,0,65
110 LINEX,Y,0,247
120 LINEX,Y,110,255
130 LINEX,Y,220,255
140 LINEX,Y,319,255
```

```
150 LINEX,Y,319,146
160 LINEX,Y,319,100
170 LINEX,Y,319,58
180 FORY=0TO51
190 LINE0,Y,319,Y,9
200 NEXT
210 GOTO210
```

Lassen Sie das Programm mit der Anweisung RUN abarbeiten.

Möchten Sie Ihren Sonnenauf- oder Untergang beenden, so drücken Sie die BRK-Taste! Das Programm wird unterbrochen und der Computer meldet sich mit dem Cursor eingabebereit. Löschen Sie das Programm bitte noch nicht, da wir es im nächsten Abschnitt noch einmal verwenden werden! Selbstverständlich können Sie es auch auf Magnetband speichern.

Sollten nach der Ausführung der Anweisung CLS noch Reste der zuvor gezeichneten Graphik auf dem oberen und unteren Bildschirmrand verbleiben, so stören Sie sich bitte nicht daran.

Im nächsten Kapitel erfahren Sie, wie diese beseitigt werden.

GRAPHIK UND FARBE

Lassen Sie das "Sonnenbild" nach Abbruch unseres letzten Programmes stehen und geben Sie ein:

```
PSET 50,25,6
```

Sicher hatten Sie angenommen, daß sich nun der in unserem "blauen Meer" befindliche Bildpunkt (50,25) gelb färbt. Statt dessen wurde jedoch, wie Sie sehen können, ein Feld von 8×4 Punkten gelb gefärbt.

Diese Erscheinung ist schnell geklärt. Für die Punkte eines Feldes von waagrecht 8 mal senkrecht 4 Punkten ist im Computer jeweils nur eine Farbcodierung, welche die Vordergrund- und die Hintergrundfarbe festlegt, gespeichert. Damit erscheinen innerhalb dieses Feldes alle Punkte, die eine Vordergrundfarbe tragen in der gleichen Vordergrundfarbe und alle Hintergrundfarbbildpunkte in der gleichen Hintergrundfarbe.

In unserem Beispiel tragen alle Punkte des 8×4 -Feldes, in dem der Punkt (50,25) liegt, die Vordergrundfarbe Violett (Farbcode 9; siehe letzte Programmzeile 190). Da der veränderte Vordergrundfarbcode 6 nicht nur für den

Punkt (50,25) maßgebend ist, nahmen alle 32 Bildpunkte, die ihre Farbwerte aus der gleichen Speicherstelle wie der Punkt (50,25) beziehen, die neu programmierte Vordergrundfarbe gelb an.

Folgende Eingabe auf unser "Sonnenbild" verdeutlicht das eben Gesagte noch einmal am Beispiel:

```
LINE 120,0,319,120,0
```

Merke:

- Der Bildschirm gliedert sich von links nach rechts in die Spalten 0 bis 319 (x-Koordinate) und von unten nach oben in die Zeilen 0 bis 255 (y-Koordinate). Der Schnittpunkt jeder Zeile und Spalte ist ein Punkt, den wir graphisch darstellen können.
- **Anweisung:** PSET
 Format: PSET x,y[, f]
 Bemerkung: Die Anweisung setzt einen Punkt auf dem Bildschirm durch Angabe der Parameter
 x (Horizontalkoordinate $0 \leq x \leq 319$),
 y (Vertikalkoordinate $0 \leq y \leq 255$) und
 f (Bildpunktfarbe $0 \leq f \leq 31$).
 Wird der Parameter f weggelassen, so erscheint der Punkt in der zuletzt gewählten Graphikfarbe.
 Beispiel:

```
10 F=7:CLS
20 FOR P=0 TO 2 * PI STEP 0.01
30 X=159+100 * SIN(P * 3)
40 Y=127+100 * SIN(P * 4)
50 PSET X, Y, F
60 NEXT
```
- **Anweisung:** PRESET
 Format: PRESET X, Y
 Bemerkung: Löschen eines Punktes auf dem Bildschirm. Anwendung wie PSET
- **Anweisung:** LINE
 Format: LINE x_a, y_a, x_e, y_e [, f]
 Bemerkung: Die Anweisung zeichnet eine Linie auf dem Bildschirm, die im Punkt mit dem Koordinaten x_a und y_a beginnt und im Punkt mit den Koordinaten x_e und y_e endet.

Die Farbe der Geraden wird durch den Vordergrundfarbcode f ($0 \leq f \leq 31$) festgelegt. Wird der Parameter f weggelassen, so erscheint die Gerade in der zuletzt gewählten Graphikfarbe.

Beispiel: LINE 10,100,300,100,26

- **Anweisung:** CIRCLE
- Format: CIRCLE $x_m, y_m, r [, f]$
- Bemerkung: Die Anweisung zeichnet einen Kreis mit dem Mittelpunktskordinaten x_m, y_m und dem Radius r auf dem Bildschirm. Die Farbe der Kreislinie wird durch den Vordergrundfarbcode f ($0 \leq f \leq 31$) festgelegt. Wird der Parameter f weggelassen, so erscheint die Kreislinie in der zuletzt gewählten Graphikfarbe.

Beispiel: 10 CLS
20 FOR R=1 TO 100
30 CIRCLE 159,127,R,16 * RND(1)
40 NEXT

- **Anweisung:** PTEST
- Format: PTEST (X)
- Bemerkung: Die Funktion testet, ob in einem Bildpunkt die Vordergrundfarbe gesetzt ist. Die x -Koordinate des zu testenden Punktes wird als Parameter in Klammern hinter die Funktion geschrieben. Als y -Koordinate benutzt der Computer die zuletzt genannte y -Koordinate einer Graphikanweisung. Ist die Vordergrundfarbe gesetzt, liefert die Funktion als Ergebnis den Wert 1; anderenfalls den Wert 0.

Beispiel: :
.
50 PRESET 0,56
55 A= PTEST(70)
60 PRINT A

Übungen

1. Erstellen Sie ein Programm, das die Sinusfunktion von 0 bis 2π und das dazugehörige Achsenkreuz des Koordinatensystems zeichnet! (Lösung Kapitel 24)
2. Erklären Sie das Auftreten farbiger Rechtecke bei der Abarbeitung des Beispiel-Programmes zur Anweisung CIRCLE!

KAPITEL 12

Bildgestaltung
(Anweisungen WINDOWS, LOCATE, WIDTH,
Funktionen AT, TAB, SCP, CSRLIN)

FENSTER

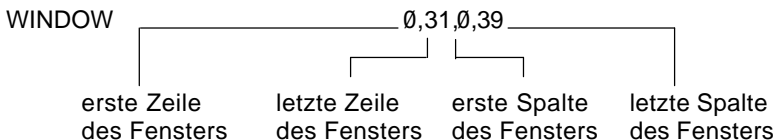
Vielleicht haben Sie es schon einmal ausgezählt; unser Bildschirm gliedert sich in 30 Schriftzeilen mit je 40 Zeichen. Dies ist das "Standard-Format", welches stets nach dem Einschalten unseres Computers eingestellt ist.

Darüber hinaus können wir aber noch die verschiedensten Bildausschnitte, wir sagen in Zukunft Fenster, einstellen. Der Wirkungsbereich der Steuerzeichen beschränkt sich auf das jeweils eingestellte Fenster. Nur hier kann geschrieben werden. Der restliche Bildschirm außerhalb des Fensters bleibt bei der Computerarbeit mit zwei Ausnahmen unbeeinflusst.

Die eine Ausnahme ist die PRINT-Anweisung AT. Diese Funktion werden wir im nächsten Abschnitt kennenlernen.

Die andere Ausnahme sind die Graphikanweisungen.

Als größtmöglichstes Fenster lässt sich der Bildschirm auf 32 Zeilen und 40 Spalten einstellen. Dies geschieht in folgender Form:



Die Numerierung der einzelnen Zeichenzeilen und Zeichenspalten ersehen Sie aus dem Bild 4.

Nun sind Ihnen bestimmt auch die nicht gelöschten Bildränder unseres "Sonnenprogrammes" aus dem letzten Kapitel erklärlich. Es war das Standardfenster (WINDOW 1,30,0,39) eingestellt. Die Ausführung der Graphik-Anweisungen ist unabhängig vom eingestellten Fenster

Dadurch wurde auch auf dem Bildschirm außerhalb des Fensters gezeichnet. Da die Steuerzeichen jedoch nur innerhalb des Fensters wirken, wurde die Graphik außerhalb des Fensters nicht mit gelöscht.

Durch die Ausführung der WINDOW-Anweisung ohne Parameter wird das Standard-Fenster mit 30 Zeilen und 40 Spalten eingestellt. Veranschaulichen Sie sich die Funktionsweise der Anweisung am Beispiel des folgenden Programms!

```

10 WINDOW0,31,0,39:COLOR0,6:CLS
20 PRINT"GROESSTMÖGLICHES FENSTER"
30 WINDOW15,25,2,38:COLOR7,2:CLS
40 PRINT"LISTING IM KLEINEN FENSTER"
50 LIST

```

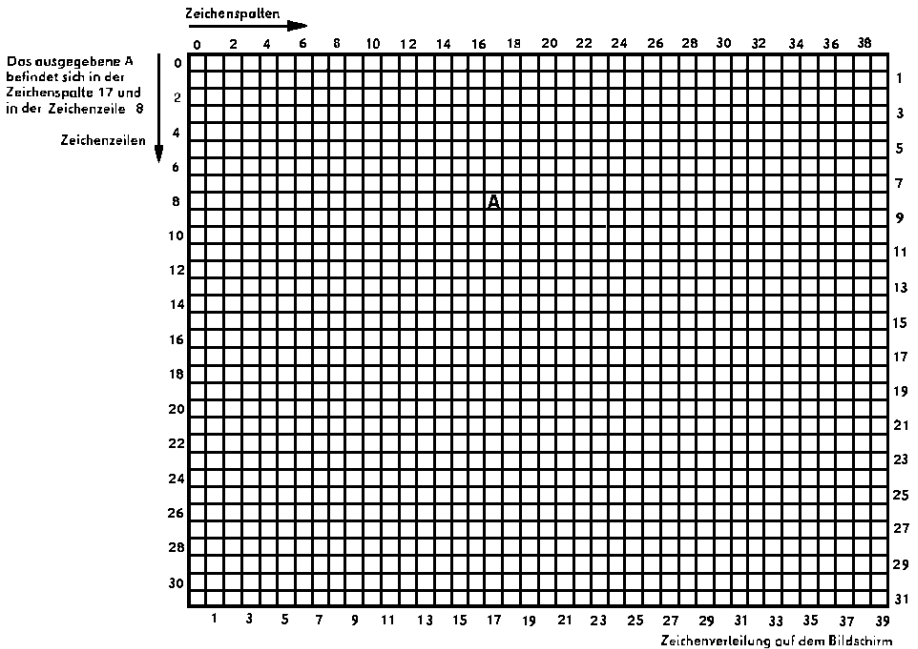


Bild 4: Einteilung des Bildschirms in Zeichenzeilen und Zeichenspalten

PLAZIEREN UND FORMATIEREN

Unabhängig vom eingestellten Fenster können wir mit der PRINT-Funktion AT unsere Ausgaben an jeder beliebigen Stelle des Bildschirms plazieren. Dabei ist folgende Form zu beachten:

PRINT AT(Zeile,Spalte);Ausdruck

Möchten wir also z.B. in die Mitte der vorletzten Zeile das Wort ENDE schreiben, so geben wir ein:

PRINT AT(30,20); "ENDE"

In folgender Weise kann man in die PRINT AT-Anweisung eine Farbanweisung einfügen:

PRINT AT(Zeile,Spalte);Farbanweisung;Ausdruck

Verändern wir unser Beispiel und lassen das Wort ENDE blau auf gelb blinken:

```
PRINT AT(30,20);COLOR17,6;"ENDE"
```

Es ist jedoch stets darauf zu achten, daß die eingefügte COLOR-Anweisung beide Farbparameter enthält.

In gleicher Weise können auch anstelle der COLOR-Anweisung die Farb-Anweisungen INK und PAPER eingefügt werden.

Um unsere Ergebnisse in leicht überschaubarer, tabellierter Form darstellen zu können, stehen uns darüber hinaus die Formatierungsfunktionen TAB und SPC zur Verfügung. Mit Hilfe dieser Funktionen können wir eine Ausgabe exakt in einer Spalte plazieren. So druckt z.B. die Anweisung

```
PRINT TAB(11);"C"
```

ein C ins 12. Buchstabenfeld vom linken Fensterrand aus. Das Argument von TAB gibt also den Abstand der nach dem Semikolon folgenden Ausgabe zum linken Bildschirmrand in Zeichenfeldern an. Das Argument der Formatierungsfunktion SPC gibt dagegen die Anzahl der zu erzeugenden Leerzeichen zwischen der Ausgabe und dem linken Fensterrand bzw. der letzten bereits in der Zeile erfolgten Ausgabe an. Verdeutlichen Sie sich dies an folgendem Beispiel:

```
10 PRINT TAB(4);"X";TAB(7);"Y"
20 PRINT SPC(4);"X";SPC(7);"Y"
```

Die Angaben der TAB- und der SPC-Funktionen beziehen sich auf das jeweils eingestellte Fenster. Alle drei Formatierungsfunktionen können nur in Verbindung mit einer PRINT-Anweisung wirksam werden!

Mit Hilfe der Anweisung LOCATE können wir den Cursor an einer beliebigen Stelle innerhalb des Fensters plazieren. Dies erweist sich als sehr vorteilhaft, wenn z.B. auch die Eingabe in eine übersichtliche Bildgestaltung mit einbezogen werden sollen:

```
10 LOCATE 20,30
20 INPUT"A=";A
.
..
.
```

Der erste Parameter hinter LOCATE gibt die Zeile und der zweite die Spalte an, in welcher der Cursor zu plazieren ist.

Um die Zeile zu ermitteln, in welcher sich der Cursor befindet, verfügt der Interpreter über die Funktion CSRLIN(n).

Ist das Argument n gleich Null, so erhalten wir die Zeilennummer der Zeile, in der sich der Cursor befindet, bezüglich dem maximalen Bildschirmfenster (32 Zeilen). Im Falle von $n > 0$, bezieht sich die Zeilenangabe auf das aktuell eingestellte Fenster.

Testen Sie folgendes Beispiel:

```
10 WINDOW 10,30,0,39:PAPER3:CLS
20 PRINT CSRLIN(0),CSRLIN(44)
```

Normalerweise beträgt die Länge einer Ausgabezeile auf dem Bildschirm 40 Zeichen. Mit der Anweisung WIDTH können wir diese jedoch verändern. Schauen wir uns die Wirkung der Anweisung an folgendem Beispiel an. Geben Sie ein:

```
10 PRINT "ABCDEFGHIJKLMN OPQRSTUVWXYZ"
```

Lassen sie das Programm abarbeiten. Geben Sie weiter ein:

```
WIDTH 16
```

Beim erneuten Abarbeiten des Programms erkennen Sie nun deutlich, daß die Länge einer Ausgabezeile auf 16 Zeichen begrenzt wurde. Gleiches gilt auch für die Ausgabe auf Drucker.

Merke:

- **Anweisung:** WINDOW
- Format: WINDOW z_a, z_e, s_a, s_e
- Bemerkung: Die Anweisung ermöglicht es, bestimmte Bildausschnitte, sogenannte Fenster, auf dem Bildschirm festzulegen. Dabei besitzen die der Anweisung folgenden Parameter folgende Bedeutung:
 - z_a – erste Zeile des Fensters
 - z_e – letzte Zeile des Fensters
 - s_a – erste Spalte des Fensters
 - s_e – letzte Spalte des Fensters
 Entsprechend der Zeichenzeile- und -Spalteneinteilung (siehe Bild 4) gilt $0 \leq z_a \leq z_e \leq 31$ und $0 \leq s_a \leq s_e \leq 39$. Bei Ausführung der Anweisung ohne Parameterangabe wird das Standardfenster (WINDOW 1,30,0,39), welches auch stets nach dem Einschalten des Computers wirksam ist, eingestellt.
- Beispiel:


```
10 ZA=2:ZE=30
20 SA=1:SE=38
30 WINDOW  $z_a, z_e, s_a, s_e$ 
40 CLS
```

- **Funktion:** AT
 Format: PRINT AT(z,s); [COLOR-Anweisung;] PRINT-Liste
 Bemerkung: Die Ausgabefunktion AT plziert die nächste Ausgabe beginnend auf dem Zeichenfeld z und in der Spalte s. Anstelle der COLOR-Anweisung kann in den PRINT-Ausdruck in gleicher Weise eine INK- oder PAPER-Anweisung eingefügt werden.
 Beispiel: PRINT AT(158,16); COLOR 23,2; "ACHTUNG!"

- **Funktion:** TAB
 und
 SPC
 Format: PRINT TAB(i);Ausdruck
 und
 PRINT SPC(i);Ausdruck
 Bemerkung: Die Ausgabefunktionen TAB und SPC plazieren eine Ausgabe in einer bestimmten Zeichenspalte. Das Argument i von TAB gibt den Abstand der Ausgabe zum linken Fensterrand in Zeichenfeldern an. Das Argument i von SPC gibt die Anzahl der zu erzeugenden Leerzeichen zwischen der Ausgabe und dem linken Fensterrand bzw. der letzten bereits in der Zeile erfolgten Ausgabe an. i ist in beiden Fällen eine ganze Zahl zwischen 0 und 255.
 Die Funktionen sind nur in Verbindung mit der Anweisung PRINT wirksam.
 Beispiel:


```
5 PRINT"X";TAB(13);"SINX";TAB(26);"COSX"
10 FOR I=0 TO 7 STEP PI/10
20 PRINT I;TAB(13);SIN(I);TAB(26);COS(I)
30 NEXT
```

- **Anweisung:** LOCATE
 Format: LOCATE z,s
 Bemerkung: LOCATE platziert den Cursor auf die angegebene Spalte der Zeile innerhalb des festgelegten Fensters. Dabei gilt für den die Zeile festlegenden Parameter z $0 \leq z \leq z_e - z_a$ mit z_a – erste Zeile des Fensters und z_e – letzte Zeile des Fensters. Ferner gilt für den Parameter s $0 \leq s \leq s_e - s_a$ mit s_a – erste Spalte des Fensters und s_e – letzte Spalte des Fensters.

Beispiel: 10 CZ=3: CS=5
20 LOCATE CZ, CS

.
.
.

- **Funktion:** CSRLIN
Format: CSRLIN (n)
Bemerkung: CSRLIN (n) liefert als Funktionswert die Nummer der Zeile, in welcher sich der Cursor befindet.

n=0 – Zeilennummer bezüglich des maximalen Fenster
n>0 – Zeilennummer bezüglich des aktuellen Fenster
n<0 – nicht definiert

Beispiel: 10 WINDOW0,31, 0,39:CLS
15 PRINTCHR\$(17)
20 RANDOMIZE
30 FOR I=1TO100
40 INK(16 * RND(1))
50 PRINTAT(32 * RND(1),38 * RND(1));CSRLIN(0)
60 NEXT
70 INK 7: PRINTCHR\$(18)

- **Anweisung:** WIDTH
Format: WIDTH Zeichenanzahl
Bemerkung: WIDTH legt die Länge einer Ausgabezeile für Drucker oder Bildschirm fest. Im Standardfall beträgt WIDTH 0, d. h. es ist nicht begrenzt.
Beispiel: WIDTH 15

Übungen

1. Erläutern Sie den Unterschied der Funktionen TAB und SPC!
2. Welche Werte können die Parameter der Anweisung LOCATE annehmen?

KAPITEL 13

Zeichenvorrat
(Funktionen ASC, CHR\$)

DIE FUNKTIONEN ASC UND CHR\$

Alle Zeichen, wie Buchstaben, Zahlen, Satzzeichen oder Steuerzeichen (z.B. Cursor links), die wir über die Tastatur eingeben können, sind im Computer unter einem bestimmten Code abgelegt. Die Codes sind Zahlen zwischen 0 und 255. Wollen Sie nun wissen, mit welchem Code der Buchstabe Z gespeichert wird, so bedienen Sie sich der Funktion ASC:

```
PRINT ASC ("Z")
```

liefert die Zahl 90. Diese Zahl ist der Code des Zeichens Z. umgekehrt können Sie auch mit Hilfe der Funktion CHR\$ zu einem Code das dazugehörige Zeichen ermitteln. Geben Sie ein:

```
PRINT CHR$(90)
```

Da CHR\$ sozusagen die Umkehrfunktion zu ASC ist, erhalten wir nun das Zeichen Z.

Beachten Sie bitte, daß die Argumente der Funktion (bei ASC ein String und bei CHR\$ ein Zahlen-Code) in Klammern zu setzen sind.

In den Übersichten Punkt 9 finden Sie eine Auflistung des gesamten Zeichenvorrates mit dem dazugehörigen Code. Mit Hilfe dieser Funktion können wir uns ohne viel Aufwand den gesamten Zeichenvorrat unseres Computers ausgeben lassen:

```
10 FOR I=32 TO 255
20 PRINT CHR$(I);
30 NEXT
```

Im Kapitel 22 werden wir erfahren, wie wir diesen Zeichenvorrat noch ergänzen und verändern können.

Unter den ersten 32 Codes (0-31) finden wir die so genannten nichtdruckenden Steuerzeichen. Durch diese sind wir nun auch in der Lage, Steuerfunktionen auszuführen, zu denen man vom BASIC aus keinen direkten Zugriff hat. So können wir zum Beispiel mit

```
PRINT CHR$(18)
```

den Scrolling-Modus einstellen, der eine Rollen des Bildes bei Überlauf organisiert bzw. mit

```
PRINT CHR$(17)
```

den Page-Modus, der ein Überschreiben des Bildes bewirkt.

Merke:

- **Anweisung:** ASC (String)
Bemerkung: Die Funktion ASC ermittelt den ASCII-Code des ersten Zeichens des Strings.
Beispiel: PRINT ASC("ZIGARRE")
90
- **Anweisung:** CHR\$ (Zahl)
Bemerkung: Die Funktion CHR\$ weist der Zahl (ASCII-Code) das entsprechende Zeichen zu
Beispiel: PRINT CHR\$(77)
M

Übung

1. Erklären Sie die Wirkung der Anweisung PRINT CHR\$(16)!

KAPITEL 14

Strings

(Stringoperationen, Stringfunktionen LEN,
VAL, STR\$, LEFT\$, MID\$, RIGHT\$, STRING\$,
VGET\$, INSTR, INKEY\$)

ZUR WIEDERHOLUNG

Wir wissen, daß unser Computer Zeichenketten, sogenannte Strings, verarbeitet. Diese Strings können wir auch unter Stringvariablen, die am Ende durch ein \$ zu kennzeichnen sind, ablegen. Es gibt mehrere Möglichkeiten, Strings in einem Programm festzulegen. Hier einige Beispiele:

```
LET A$="BAUM"
INPUT B3$
R$=INKEY$
```

Die Ausgabe erfolgt mit Hilfe der Anweisung PRINT, z.B. wie folgt:

```
PRINT "ANTWORT: ";G$
```

STRING-OPERATIONEN

Wir können auch mit Strings in gewissem Umfang operieren. So ist es zum Beispiel möglich, Strings durch das Operationszeichen "+" miteinander zu verknüpfen. Geben Sie z.B. ein:

```
PRINT "AUTO"+"MOBIL"
```

Selbstverständlich können Sie die Strings auch unter Variablen ablegen und diese dann miteinander oder mit Stringoperationen verknüpfen.

Weiterhin lassen sich die Strings durch alle in Kapitel 6 behandelten Vergleichsoperationen miteinander vergleichen. Ein String ist "kleiner" als ein anderer, wenn er im Alphabet vorher steht; also z.B.:

```
"SCHMIDT" < "SCHMITT"
"BAUM" < "GEDANKE"
"APFEL" < "APFELKUCHEN"
```

DIE STRINGFUNKTIONEN LEN, VAL, STR\$, LEFT\$, MID\$, RIGHT\$, STRING\$, VGET\$ UND INSTR

LEN (String) gibt die Anzahl der Zeichen des Strings aus. Geben Sie folgende Beispiele ein:

```
PRINT LEN ("OTTO")
PRINT LEN ("234")
```

VAL (String) gibt den numerischen Wert des Strings an. Ist das erste Zeichen des Strings kein +, -, % und keine Zahl, so ist VAL (String) = 0.

Testen Sie folgendes Programm:

```
10 A$="22"
20 B$="33"
40 PRINT A$+B$
50 PRINT VAL (A$)+VAL (B$)
```

STR\$(Zahl) wandelt die Zahl zu einem String (die Ziffernfolge, die die Zahl darstellt) um. Beispiel:

```
PRINT STR$(5672)
```

Die folgenden Stringfunktionen bilden einen neuen String, der im Falle:

LEFT\$(String, x) aus x Zeichen von links besteht;
 RIGHT\$(String, x) aus x Zeichen von rechts besteht;
 MID\$(String, x) aus dem Zeichen ab der x. Stelle besteht;
 MID\$(String, x,y) aus dem y Zeichen ab der x. Stelle besteht.

Folgendes Programm demonstriert die Wirkungsweise dieser Funktionen an einem Beispiel:

```
10 A$="PAPAGEI"
20 PRINT LEFT$( A$,4)
30 PRINT RIGHT$( A$,2)
40 PRINT MID$( A$,7)
50 PRINT MID$( A$,3,4)
```

Die Funktion STRING\$ in der Form

```
STRING$( n, String)
```

vervielfältigt den in der Klammern stehenden String n-mal.

```
Beispiel: 10 A$="KC 85/3":N=5
          20 B$= STRING$( N,A$)
          30 PRINT B$
```

Die Funktion VGET\$ liefert den Inhalt der Cursorposition als String. Beispiel:

```
10 CLS
20 PRINT"TEXT"
30 LOCATE0,1:A$=VGET$
40 PRINT AT (10, 10);A$
50 LOCATE1,0
```

INSTR (A\$,B\$) ermittelt die Position, ab welcher A\$ in der Zeichenkette B\$ enthalten ist. Beispiel:

```
10 A$="PFERDE":B$="BLUMENTOPFERDE"
20 PRINT INSTR (A$,B$)
```

EINGABE MIT INKEY\$

Zur Vereinfachung der Programmbedienung sowie zur Durchführung von Reaktionstests ist die Funktion INKEY\$ gleichermaßen gut geeignet.

INKEY\$ holt während des Programmablaufs eine Information von der Tastatur, ohne daß das Programm angehalten oder die ENTER-Taste gedrückt werden muß. Dabei ist es jedoch nur möglich, ein Tastenzeichen als String einzulesen. So kann man mit INKEY\$ den Dialogbetrieb einfacher gestalten. Wenn wir mit INPUT eine Eingabe realisieren, so müssen wir nach der Eingabe immer noch die ENTER-Taste drücken. Dies können wir mit der Anweisung INKEY\$ wie folgt umgehen:

```
1130 A$=INKEY$ : IF A$=""GOTO 1130
```

Das Programm bleibt solange in der Zeile 1 130 bis der Anwender eine Taste drückt. Dieses wird dann als String unter der Variablen A\$ abgelegt und steht somit der weiteren Auswertung zur Verfügung.

Merke:

- Strings können durch das Zeichen "Í" miteinander **verknüpft** werden.
- Strings kann man miteinander **vergleichen**. Ein String ist "kleiner" als ein anderer, wenn er im Alphabet vorher steht.
- Der BASIC-Interpreter verfügt über folgende **Stringfunktionen**: LEN, VAL, STR\$, MID\$, LEFT\$, RIGHT\$, STRING\$, VGET\$, INSTR

Das Argument der Funktion muß stets in Klammern stehen.

- **Funktion:** INKEY\$
Format: INKEY\$
Bemerkung: INKEY\$ liest von der Tastatur ohne Programmunterbrechung und ohne Betätigung der ENTER-Taste genau ein Zeichen ein.
Beispiel: 130 LET B\$= INKEY\$:IF B\$=""GOTO 130

Übung

Schreiben Sie ein Programm, das zwei einzugebende Strings alphabetisch ordnet. (Lösung Kapitel 24).

KAPITEL 15

Pause
(Anweisung Pause)

DIE ANWEISUNG PAUSE

Sollen Computerbilder eine bestimmte oder eine beliebige Zeit während des Programmablaufes erhalten bleiben, so können wir die Anweisung PAUSE sehr vorteilhaft einsetzen.

```

10 CLS:PRINT"GEDAECHTNISTRAINING"
20 PRINT "BITTE ZIFFERNFOLGE MERKEN UND BEI"
30 PRINT "ABFRAGE '?' EINGEBEN."
40 PRINT CHR$(17)
50 COLOR0,7:WINDOW13,13,15,25:CLS
60 FOR I=1TO5
70 CLS
80 A=INT(RND(1)*1E6):IFA<111111 GOTO 80
90 PRINTA:PAUSE25:CLS
100 INPUTB:IFA=BGOTO120
110 CLS:PRINT"FALSCH":PAUSE30:GOTO130
120 CLS:PRINT "RICHTIG": PAUSE 30
130 NEXT
140 WINDOW:COLOR7,1:CLS

```

Wie Sie beim Test dieses kleinen Beispielprogrammes gemerkt haben werden, unterbricht die Anweisung PAUSE die Abarbeitung des Programmes für einen bestimmten Zeitraum. Dieser Zeitraum wird mit Hilfe des Parameters (t) der Anweisung festgelegt. Für eine PAUSE von einer Sekunde gilt dabei der Parameter $t=10$. So bewirkt die Anweisung PAUSE 30 z. B. eine Programmunterbrechung von etwa 3 Sekunden.

Erfolgt keine Parameterangabe, so wird das Programm analog der Tastenfunktion STOP bis zum Bestätigen der Taste **⏏** unterbrochen. Arbeiten Sie mit einzeiligen Fenster, so vergessen Sie bitte nie vorher den Page-Modus einzustellen (Zeile 40 des letzten Programmes). Ist der Scroll-Modus eingestellt, so wird jede Ausgabe sofort aus dem einzeiligen Fenster "gerollt".

- **Anweisung:** PAUSE
- Format: PAUSE [t]
- Bemerkung: Mit der PAUSE-Anweisung wird die Programmabarbeitung für die Zeit von $t * 0,1$ Sekunden unterbrochen ($1 \leq t \leq 255$). PAUSE ohne Argument entspricht der Taste STOP.

Beispiel:

```

.
.
.
.
50 PAUSE 250
.
.

```

KAPITEL 16

Unterprogrammtechnik
(Anweisungen GOSUB, RETURN)

DIE ANWEISUNGEN GOSUB UND RETURN

Wird ein bestimmter Programmabschnitt mehrmals im Programm benötigt, so ist es effektiv, diesen als Unterprogramm zu schreiben. Mit der Anweisung GOSUB *n* wird zu dem auf der Programmzeile *n* beginnenden Unterprogramm gesprungen und dieses abgearbeitet. Nach der Abarbeitung des Unterprogrammes, das immer mit der Anweisung RETURN abzuschließen ist, "springt" der Computer zur Aufrufstelle des Hauptprogramms zurück und führt die Arbeit mit der folgenden Anweisung fort.

Die Funktionsweise dieser beiden Anweisungen werden wir uns an folgendem Programm veranschaulichen :

```

10 PRINT "DAS IST DAS HAUPTPROGRAMM HP"
20 GOSUB 100
30 PRINT "WIEDER IM HP"
40 GOSUB 200
50 PRINT "SCHON WIEDER IM HP"
60 GOSUB 100
70 PRINT "SCHLUSS"
80 END
100 PRINT "HIER IST UNTERPROGRAMM 1"
110 RETURN
200 PRINT "HIER IST UNTERPROGRAMM 2"
210 RETURN

```

} Hauptprogramm
 } Unterprogramm 1
 } Unterprogramm 2

Das Beispiel unterstreicht die bereits erwähnte Bedeutung der Anweisung END. Fehlt diese Anweisung am Schluss des Programms, so wird der Computer nachfolgende Unterprogramme oder Programme als zum Hauptprogramm gehörend erkennen und weiter abarbeiten. Es kommt dabei jedoch zu Fehlermeldungen (Rücksprung durch RETURN nicht definiert).

Merke:

- **Anweisung:** GOSUB
und
RETURN
- Format: GOSUB Zeilennummer
und
RETURN

Bemerkung: GOSUB ruft ein BASIC-Unterprogramm auf, welches in der angegebenen Zeilennummer beginnt. Jedes BASIC -Unterprogramm ist mit RETURN abzuschließen.

Beispiel:

```
10 INPUT X
20 GOSUB 300
30 PRINT E
40 END
300 E=(X+365)/52
300 RETURN
```

KAPITEL 17

Mehrfache Programmverzweigungen
(Anweisungen ON. . .GOTO, ON. . .GOSUB)

ON...GOTO...

Haben Sie in einem Programm mehr als zwei Fälle zu unterscheiden, so können Sie die Anweisung ON GOTO sehr vorteilhaft verwenden.

Die Anweisung ist wie folgt aufgebaut:

ON i GOTO Liste von Zeilennummern

Der Sprungbefehl verzweigt im konkreten Fall zu der Zeilennummer, die als i-te in der Liste steht.

Veranschaulichen wir uns das an unserem Benzinverbrauch-Bewertungsprogramm aus Kapitel 6. Diesmal soll die Bewertung wie folgt sein:

Benzinverbrauch/100km	Bewertung
≤ 3	unmöglich
3– 6	sehr gut
6– 9	gut
9– 12	schlecht
> 12	unmöglich

```

10 INPUT "BENZINVERBRAUCH IN L/100 KM =";B
20 A = INT(B/3)=1
30 ON A GOTO 40, 50, 60, 70
40 PRINT "UNMOEGlich" :END
50 PRINT "SEHR GUT" :END
60 PRINT "GUT" : END
70 PRINT "SCHLECHT" :END

```

Ist nun z.B. $B = 8.5$, so errechnet der Computer $A = 3$, für dieses spezielle A bewirkt der Befehl in Zeile 30 einen Sprung zur Zeile 60, weil diese Zeilennummer in der Liste der Zeilennummern in Zeile 30 an dritter Stelle steht.

ON...GOSUB...

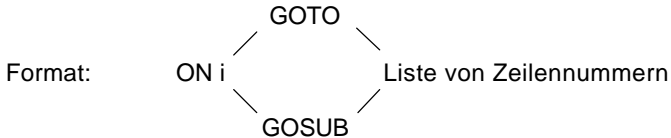
Die Anweisung in der Form

ON i GOSUB Liste von Zeilennummern

wirkt wie ON...GOTO...mit der Ausnahme, daß bei der Programmverzweigung Unterprogramme angesprungen werden und nach der Abarbeitung dieser das Programm in der der ON...GOSUB...-Anweisung folgenden Zeile fortgesetzt wird.

Merke:

- **Anweisung:** ON...GOTO
und
ON...GOSUB...



Bemerkung: Die Anweisungen verzweigen das Programm zu der Zeilennummer, die als i-te in der Liste steht. Im Fall ON...GOSUB sind die Zeilennummern die Adresse der aufzurufenden Unterprogramme. Die Zeilennummern werden voneinander durch Kommata getrennt. Der Ausdruck i wird stets auf eine ganze Zahl abgerundet.

Ist $i = 0$ oder größer als die Anzahl der Listenelemente, wird der Befehl ignoriert und die nächste Zeile ausgeführt. Der gerundete Wert von i muß im Bereich von 0 bis 255 liegen.

Beispiel:

```

PROGRAMM SCHERE-STEIN-PAPIER
10 RANDOMIZE
20 WINDOW0,31,0,39:COLOR7,1:CLS:WINDOW
30 A$="SCHERE":B$="STEIN":C$="PAPIER"
40 PRINT A$,B$,C$
50 PRINT:PRINT
60 PRINT"SPIELREGELN":PRINT
70 PRINT"-SCHERE SCHNEIDET PAPIER":PRINT
80 PRINT"-STEIN SCHLEIFT SCHERE":PRINT
90 PRINT"-PAPIER WICKELT STEIN EIN":PRINT:PRINT
100 INPUT"WAS WAEHLST DU?";S$
110 IF S$=A$ THEN S=3:PRINT
120 IF S$=B$ THEN S=6:PRINT
130 IF S$=C$ THEN S=9:PRINT
140 IF S=0 THEN PRINT"FALSCH EINGABE":GOTO100
150 WINDOW15,18,0,39
160 COLOR0,5:CLS:PRINT"ICH WAEHLE";
170 C=INT(3*RND(1))
180 ONC+S GOTO20,20,190,200,210,220,230,240,250,260,
270
190 PRINTA$:GOSUB460:GOTO280
  
```

```
200 PRINTB$:GOSUB410:GOTO280
210 PRINTC$:GOSUB360:GOTO280
220 PRINTA$:GOSUB360:GOTO280
230 PRINTB$:GOSUB460:GOTO280
240 PRINTC$:GOSUB410:GOTO280
250 PRINTA$:GOSUB410:GOTO280
260 PRINTB$:GOSUB360:GOTO280
270 PRINTC$:GOSUB460
280 COLOR7,1
290 WINDOW25,27,0,39
300 PRINT"EIN NEUES SPIEL? (J/N)"
310 X$=INKEY$:IFX$=""GOTO310
320 IFX$="N" THEN WINDOW:CLS:END
330 IFX$="J"GOTO300
340 C=0:S=0:SS="" :X$=" "
350 GOTO20
360 PRINT
370 PRINT "GRATULIERE! DU HAST GEWONNEN."
380 SP=1:CP=0
390 GOSUB510
400 RETURN
410 PRINT
420 PRINT"PECH FUER DICH! ICH HABE GEWONNEN."
430 SP=0:CP=1
440 GOSUB510
450 RETURN
460 PRINT
470 PRINT"UNENTSCIEDEN."
480 SP=0:CP=0
490 GOSUB510
500 RETURN
510 SSP=SSP+SP
520 SCP=SCP+CP
530 COLOR25,6
540 WINDOW21,23, 0,39:CLS
550 PRINT"SPIELER","COMPUTER"
560 PRINTSSP,SCP
570 RETURN
```

KAPITEL 18

Eingabe von mehreren Daten
(Anweisungen DATA, READ, RESTORE)

DATA, READ UND RESTORE

Bisher haben wir zwei Möglichkeiten kennen gelernt, einer Variablen einen Wert zuzuweisen. Einmal können wir dies durch eine direkte Wertzuweisung (z.B. $V = 1.2$), zum anderen haben wir die Möglichkeit, die Wertzuweisung mit Hilfe des INPUT-Befehls zu realisieren. Müssen wir nun größere Datenmengen verarbeiten, so erweisen sich beide Methoden aufgrund der immer wiederkehrenden Eingabe als langwierig. Abhilfe schaffen hier die drei oben genannten Anweisungen. Hinter die Anweisung DATA sind die Daten, also die Werte, die wir verarbeiten wollen, zu schreiben. Diese Werte können sowohl numerische als auch String-Konstanten sein.

Mit der Anweisung

```
READ Variable (,Variable . . .)
```

weisen wir den hinter READ stehenden Variablen die hinter DATA stehenden Werte zu.

Testen Sie dazu folgendes Programm:

```
10 DATA33,2,3,55,84, -9.5723
15 PRINT "NUN STEHT DER DATA-ZEIGER AUF DEMERSTEN WERT"
20 FOR I=1 TO 6
30 READ A
40 PRINT "A HAT JETZT DEN WERT: ";A
45 PRINT "NUN WIRD DER DATA-ZEIGER AUF DEN NAECHSTEN
    WERT GESETZT"
50 NEXT
```

Sicher haben Sie an diesem Beispiel die Funktionsweise der beiden neuen Anweisungen schnell verstanden.

Damit der Computer sich merken kann, welchen der Werte er schon ausgelesen hat und welcher somit der nächste ist, gibt es einen internen DATA-Zeiger, der jeweils auf den nächsten abzuarbeitenden Wert der DATA -Liste zeigt. Würden Sie das Programm verändern und die Daten in der Zeile 10 z.B. noch einmal auslesen wollen, so käme dort die Fehlermeldung OD (Out of DATA), was bedeutet, daß nicht genügend Daten zum Auslesen bereitstehen.

Diesen Fehler können Sie beheben, indem Sie die fehlenden Daten in Zeile 10 anfügen oder eine neue Programmzeile (z.B. 12) mit DATA und den Werten hinzufügen.

Benötigen Sie jedoch die gleichen Werte in der gleichen Reihenfolge noch einmal, so brauchen Sie nur den DATA -Zeiger auf den ersten Wert zurücksetzen. Dies können Sie ganz einfach mit der Anweisung RESTORE bewerkstelligen. In unserem erwähnten Beispiel würden Sie dann folgende Programmzeile einfügen:

```
95 RESTORE
```

Eine einfache Möglichkeit der geschilderten Programmerweiterung könnte dann wie folgt aussehen:

```
95 RESTORE
100 FOR I = 1 TO 6
110 READ A
120 PRINT "A HALBE HAT JETZT DEN WERT"; A/2
130 NEXT
```

Abschließend soll anhand unseres Sonnenprogrammes aus Kapitel 11 demonstriert werden, wie Sie mit Hilfe der genannten Anweisungen Programme elegant und wirksam verkürzen können.

Gleichzeitig wurde im Programm die Darstellung der Meeresfläche vereinfacht über die WINDOW-Anweisung abgefahren und die Sonne durch die CIRCLE-Anweisung innerhalb einer FOR-Schleife kürzer und schneller realisiert. Dabei wurden zwei Sonnen um einen Graphikpunkt versetzt gezeichnet, um die Fläche lückenlos mit Vordergrundfarbpunkten "dicht" zu bekommen.

```
10 COLOR7,1:CLS
20 X=220:Y=50
30 FOR R=1TO30
40 CIRCLEX,Y,R,2: CIRCLE,49,R,2
50 NEXT
60 FORI=1TO9
70 READC,D
80 LINEX,Y,C,D,2
90 NEXT
100 WINDOW26,31,0,39:PAPER5:CLS:WINDOW
110 GOTO110
120 DATA 0, 150, 0, 65, 0, 247, 110, 255, 220
130 DATA 255,319,255,319,146,319,100,319,58
```

Merke:

- **Anweisung:** DATA
Format: DATA Konstantenliste
Bemerkung: DATA kennzeichnet die Liste von numerischen und String-Konstanten, welche durch die READ -Anweisung den einzelnen Variablen zugeordnet werden. Die Listenelemente werden durch Kommata voneinander getrennt. Beginnt eine Stringkonstante der Liste mit einem oder mehreren Leerzeichen, so ist diese konstant in Anführungszeichen zu setzen. Bei mehreren DATA -Listen liest die READ-Anweisung in der Reihenfolge der Zeilennummern aus.

Beispiel: 10 FOR I=1 TO 3
20 READ X\$
30 PRINT "X\$ HAT JETZT DEN WERT"; X\$
40 NEXT
50 DATA SO, FUNKTIONIERT, DAS
60 END

- **Anweisung:** READ
Format: READ Variablenliste
Bemerkung: READ liest die mit DATA vereinbarten Werte aus und weist sie den Variablen der Liste in der Reihenfolge der Vereinbarung zu. Die Listenelemente werden durch Kommata voneinander getrennt.

Beispiel: siehe Anweisung DATA

- **Anweisung:** RESTORE
Format: RESTORE [Zeilennummer]
Bemerkung: RESTORE setzt den DATA-Zeiger auf den ersten Wert der DATA -Liste mit der niedrigsten oder der vereinbarten Zeilennummer, so daß die in der DATA -Anweisung vereinbarten Werte erneut gelesen werden können!

Beispiel: Siehe dieses Kapitel

DIE LETZTEN TRICKS

KAPITEL 19

Programmschwierigkeiten
(Anweisungen, TRON, TROFF, STOP)

Sicher ist es Ihnen schon öfter passiert, daß ein Programm bei seinem ersten Lauf Fehler aufwies. Nicht immer waren es nur Schreibfehler. Diese grundlegenden Fehler resultieren meistens aus einer fehlerhaften Problemanalyse oder einer nicht korrekten Umsetzung Ihrer Ideen in BASIC.

Diese Fehler passieren auch Experten immer wieder. Es ist sogar recht ungewöhnlich, daß ein Programm auf Anhieb fehlerfrei läuft. In unseren bisherigen kleinen Programmen sind die Fehler oft, insbesondere auch durch die Fehlermeldungen, recht schnell auszumachen. Je umfangreicher die Programme sind, um so leichter schleichen sich Fehler ein und um so schwieriger sind diese dann zu finden.

Dieses Kapitel möchte Ihnen ein paar Tips zur Vermeidung und zum schnellen Finden solcher Fehler geben.

PROGRAMMERSTELLUNG

Die meisten Fehler können wir durch eine gründliche Programmvorbereitung vermeiden. Wie sollte eine solche Vorbereitung nun konkret aussehen?

Am Anfang eines Programms steht meistens ein Problem aus der Praxis. Dieses Problem gilt es als erstes genau zu erörtern, d.h. wir überlegen uns, welche Größen uns bekannt sind, in das Programm eingehen und welche Größen oder Antworten ausgegeben werden sollen. Nun denken wir uns noch eine prinzipielle Lösungsmöglichkeit aus und gliedern diese grob in einzelne Schritte. Diese Schritte übertragen wir dann in einen Programm-Ablauf-Plan, kurz PAP genannt. Ein PAP legt die Struktur eines Programmes sehr übersichtlich dar, so daß grundlegende Fehler sofort ins Auge fallen. Als Beispiel wollen wir uns diese und die weiteren Schritte zum Erstellen eines lauffähigen Programmes anhand eines PAP anschauen.

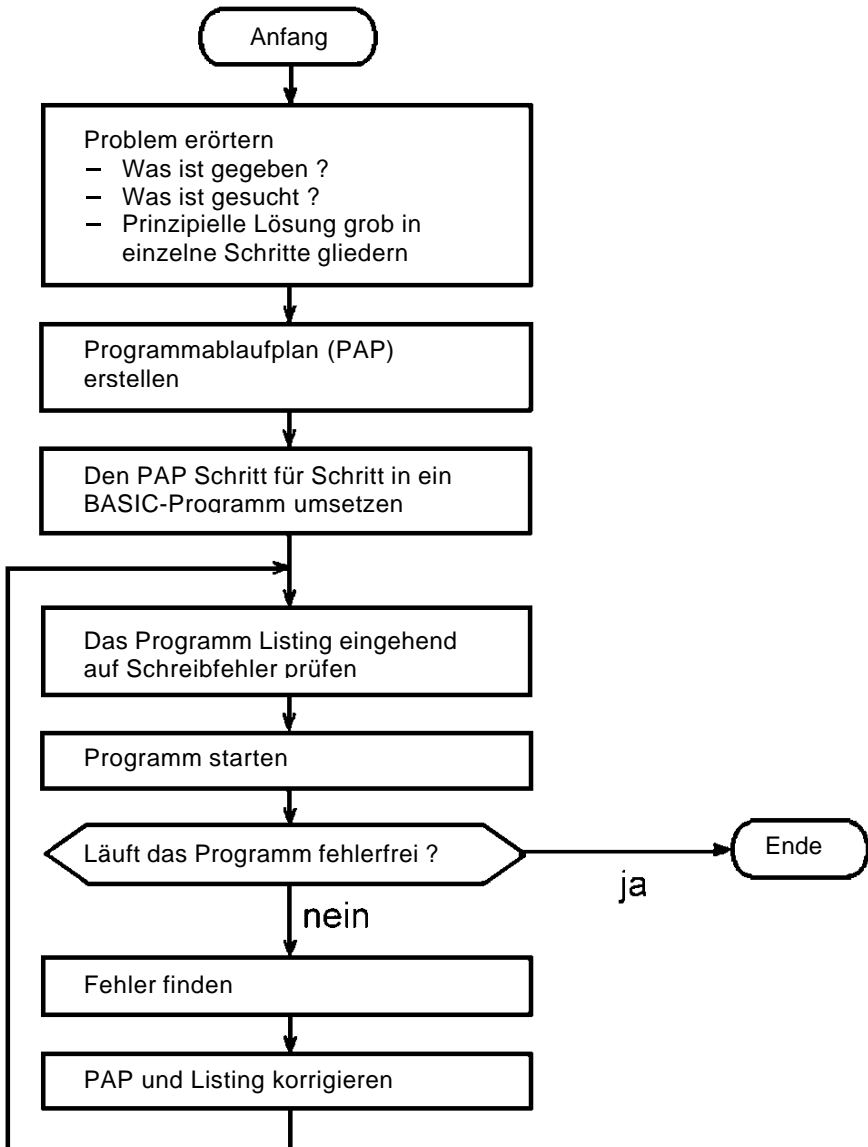





Bild 5 Programmablaufplan zum Erstellen eines lauffähigen Programms

In einem PAP symbolisiert

ein  den Anfang oder das Ende eines Programms

ein  eine oder mehrere Anweisungen und

ein  eine antwortbedingte Verzweigung des Programms

Anschließend schauen wir uns unser ursprüngliches Noten-Bewertungs-Programm aus Kapitel 8 im PAP an.

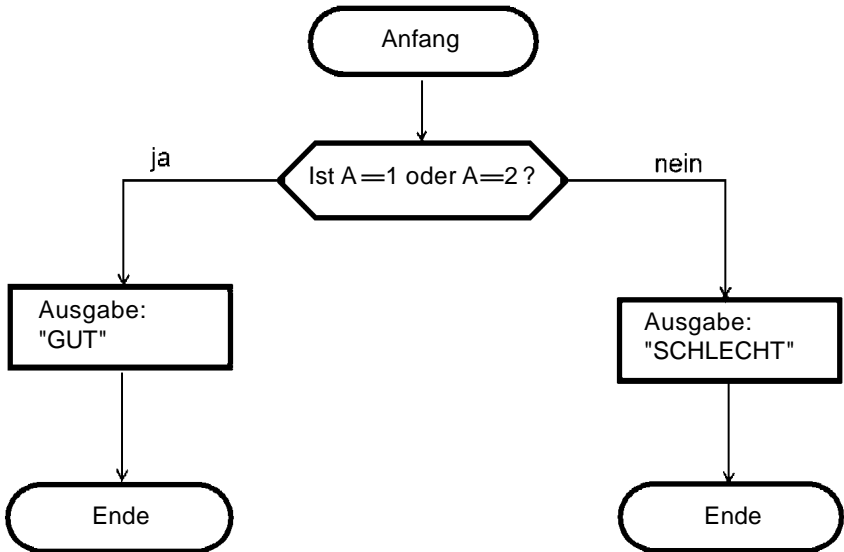


Bild 6 Symbolische Darstellung

DAS FINDEN VON FEHLERN

Trotz aller Bemühungen lassen sich Fehler oft nicht vermeiden. Deshalb wollen wir jetzt auf das kleine Anweisungskästchen unseres ersten PAP, das da "Fehler finden" heißt, eingehen. Den ersten Hinweis auf unseren Fehler bekommen wir durch die Fehlermeldung bei Abbruch des Programms. Mit dieser Information und einer nochmaligen Prüfung finden wir viele Fehler im Listing.

Oft jedoch wird das Programm gar nicht unterbrochen, sondern der Computer macht einfach etwas anderes, uns völlig Unverständliches. In diesem Fall können wir die Anweisung TRON zur besseren Verfolgung des Programmablaufs einsetzen. Die Anweisung bewirkt, daß bei Abarbeitung eines Programms die Nummern der Zeilen in der Reihenfolge ihrer Abarbeitung angezeigt werden. Geben Sie ein:

```
10 FOR I=1 TO 5
20 PRINT I
30 NEXT
TRON
```

Mit dem Kommando TRON haben Sie den eben beschriebenen Anzeigemodus eingestellt. Bei Ablauf des eingegebenen Programms gelangen die Nummern der abgearbeiteten Zeilen zur Anzeige. Der Modus wird durch das Kommando TROFF wieder ausgeschaltet.

Führt auch das nicht zum Erfolg, so können wir die Anweisung STOP in das Programm einfügen und so an jeder beliebigen Stelle anhalten. Nach einer Programmunterbrechung mit STOP meldet sich der Computer wie bei einer Unterbrechung durch die BRK-Taste, z.B. mit

```
BREAK IN 30
```

Durch Eingabe der Anweisung CONT und das Betätigen der ENTER-Taste können Sie das Programm fortsetzen. Testen Sie die Anweisung, indem Sie sie in unser Programm wie folgt einfügen :

```
25 STOP
```

Darüber hinaus können Sie das Programm auch direkt während des Programmablaufs durch Drücken der STOP-Taste anhalten. Hier wird das Programm, wie bereits in Kapitel 5 beschrieben, durch Betätigen der ⏏-Taste fortgesetzt.

Nun noch ein letzter Tipp zur Fehlersuche:

Lassen Sie sich durch Einfügen der PRINT-Befehls die Variablen vor und nach den kritischen Punkten zur Kontrolle ausgeben.

Merke:

- **Anweisung:** TRON
und
TROFF
Format: TRON
und
TROFF
Bemerkung: TRON bewirkt, daß bei Abarbeitung eines Programms die Nummern der Zeilen in der Reihenfolge ihrer Abarbeitung zur Anzeige gebracht werden.
TROFF schaltet diesen Anzeigemodus wieder aus.
Beispiel: Siehe dieses Kapitel

- **Anweisung:** STOP
Format: STOP
Bemerkung: STOP unterbricht gezielt ein Programm. Der Programmablauf kann durch Ausführung der Anweisung CONT fortgesetzt werden.
Beispiel:
10 X=3
15 STOP
20 PRINT X
30 END

Übung

Zeichnen Sie zu dem Benzin-Verbrauch-Bewertungsprogramm von Kapitel 19 einen PAP.

KAPITEL 20

Musik
(Anweisungen SOUND, BEEP)

TONAUSGABE

Mit dem KC 85/3 können Sie Töne monophon über das Fernsehgerät bei FBAS- oder RGB-Anschluß zu Gehör bringen. Dabei ist die Lautstärke in 32 Stufen regelbar.

Darüber hinaus erfolgt die Tonausgabe auch stereophon mittels einer Stereoanlage oder eines anderen geeigneten Musikwiedergabegerätes. Hierbei wird der Computeranschluß TAPE über das Diodenkabel mit dem Wiedergabegerät verbunden.

Im einfachsten Fall drücken Sie die Schnellstop-Taste Ihres Recorders und schalten diesen auf Aufnahme. Nun funktioniert der Recorder nicht mehr als Speichereinheit, sondern als Musikwiedergabeeinheit des Computers. Achten Sie jedoch bitte in Ihrem eigenen Interesse darauf, daß keine auf dem Band gespeicherten Programme gelöscht werden!

Auch das bereits in Kapitel 1 erwähnte Fehlersignal kann nun über die eben beschriebenen Wege realisiert werden.

Technische Angaben zur Tonausgabe finden Sie im System-Handbuch Kapitel 1, 6 und 7.

TÖNE

Töne sind die Bausteine der Musik und etwas weiter gesehen aller Geräusche. Sie unterscheiden sich durch die Tonhöhe, die Tondauer und die Lautstärke.

Der KC 85/3 verfügt über zwei Kanäle, über einen Tonhöhenumfang von 7 Oktaven und ist in 32 Lautstärkestufen programmierbar. Wollen wir nun einen bestimmten Ton erzeugen, so müssen wir demzufolge angeben

- auf welchem Kanal
 - wie lange
 - in welcher Tonhöhe
 - in welcher Lautstärke
- die Tonausgabe erfolgen soll.

Diese Angaben erfolgen mit Hilfe der Anweisung SOUND in der Form:

SOUND $z_1, v_1, z_2, v_2, l_s, t_d$

Dabei legen die Parameter z_1 und v_1 die Tonhöhe des Kanals 1 und die Parameter z_2 und v_2 die Tonhöhe des Kanals 2 entsprechend der folgenden Tonwerttabelle fest. Mit den Parametern l_s und t_d werden die Lautstärke ($0 \leq l_s \leq 31$) und die Tondauer ($0 \leq t_d \leq 255$) bestimmt.

Beachten Sie folgende Sonderfälle:

- werden t_1 und t_2 nicht angegeben, so bleiben die vorherigen Werte erhalten
- ist die Lautstärke = 0, so erfolgt keine Tonausgabe über das Fernsehgerät, jedoch weiterhin über die Diodenbuchse;
- ist $z_1 = 0$ oder $z_2 = 0$, so wird kein Ton ausgegeben;
- ist die Tondauer = 0, so wird bis zum nächsten Aufruf ein Dauerton abgegeben

Mit dem folgenden Programm soll ein Beispiel aus der Fülle von Anwendungsmöglichkeiten der akustischen Möglichkeiten des Computers demonstriert werden. Im Programm wird die Tastatur als Klaviatur genutzt. Durch das Drücken einer der Zifferntasten 1 bis 8 wird ein Ton der C-Dur-Tonreihe erzeugt. Die Zuordnung der Töne zu den Tasten ist dem Bildschirm zu entnehmen. Die Tonausgabe wird durch Drücken der Leertaste beendet. Das Programm kann nur durch die BRK-Taste beendet werden.

```

10 WINDOW:COLOR7,1:CLS
20 PRINTAT (14,12); "cdefgahc"
30 PRINTAT (16,12); COLOR0,7; "1■2■3■4■5■6■7■8"
40 A$=INKEY$:IFA$=""GOTO40
50 IFASC(A$)=32THENX=0:GOTO100
60 Z=VAL(A$):IFZ<1 OR Z>8 GOTO40
70 FOR I=1 TO Z
80 READX
90 NEXT
100 SOUNDX,0,0,0,31,0
110 RESTORE:CLEAR:GOTO40
120 DATA216, 192, 171, 162, 144, 128, 114, 108

```

Darüber hinaus können Tonfolgen oder Lieder zur Unterstützung der Aussagekraft fest programmiert werden. Aber es ist z.B. auch möglich, den Computer selbstständig mit Hilfe der RND-Funktion improvisieren zu lassen. Wie bereits gesagt, es gibt viele Anwendungsmöglichkeiten. Um ein einfaches akustisches Signal zu erzeugen, benutzt man am besten die Anweisung BEEP. Probieren Sie aus:

```
BEEP
```

Der bei der Ausführung der Anweisung erzeugte Ton entspricht einer SOUND-Anweisung mit dem Vorteiler $v = 0$ und der Zeitkonstante $z = 48$. Es ist der gleiche Ton, der auch bei den Fehlermeldungen ausgegeben wird.

Die Zuordnung der Töne zu den Zeitkonstanten bzw. Vorteilerstufen kann nach folgender Tabelle erhalten werden :

Ton	c	cis	d	dis	e	f	fis	g	gis	a	ais	h	c
Oktave													
1	V	1	1	1	1	1	1	1	1	1	1	1	1
	Z	204	199	196	181	173	160	152	144	137	127	118	102
2	V	1	1	1	1	1	1	1	1	1	1	1	1
	Z	102	97	95	90	86	81	75	72	68	64	60	54
3	V	1	1	1	1	1	1	1	1	1	1	1	1
	Z	54	50	48	45	43	40	38	36	34	32	30	27
4	V	1	1	1	1	1	1	1	1	1	1	1	1
	Z	27	25	24	23	21	20	19	18	17	16	15	216
5	V	0	0	0	0	0	0	0	0	0	0	0	0
	Z	216	204	192	182	171	162	153	144	136	128	120	108
6	V	0	0	0	0	0	0	0	0	0	0	0	0
	Z	108	102	96	91	86	81	76	72	68	64	60	54
7	V	0	0	0	0	0	0	0	0	0	0	0	0
	Z	54	51	48	45	43	40	38	36	34	32	30	27

Anmerkung: Nicht jeder Ton wird nach dieser Tabelle die exakte Frequenz erhalten. Sie werden selbst feststellen, daß damit eigene Kompositionen auch ganz gut klingen.

Mit einem der Anweisung folgenden Parameter können Sie festlegen, wie oft das Signal ausgegeben werden soll. Testen Sie es selbst:

BEEP 4

Merke:

- **Anweisung:** BEEP
 Format: BEEP [n]
 Bemerkung: Mit Hilfe der Anweisung BEEP werden akustische Signale mit festgelegter Länge erzeugt. Der Parameter n bestimmt die Anzahl der Signale ($1 \leq n \leq 255$)
 Entfällt der Parameter, so gilt $n = 1$.
 Beispiel:
 - .
 - .
 - .
 - 130 BEEP
 - .
 - .
 - .

- **Anweisung:** SOUND
 Format: SOUND $z_1, v_1, z_2, v_2 [l_s] [t_d]$
 Bemerkung: Die Anweisung SOUND gestattet die Ausgabe von Tönen mit steuerbarer Tonhöhe, Tonlänge und Lautstärke.
 Parameterfestlegung:
 z_1 – Zeitkonstante für CTC Kanal 1
 v_1 – Vorteiler für CTC Kanal 1
 z_2 – Zeitkonstante für CTC Kanal 2
 v_2 – Vorteiler für CTC Kanal 2
 l_s – Lautstärke
 t_d – Tondauer
 mit $(1 \leq z_i \leq 255)$ ($z_i = \emptyset$: Ton ausgeschaltet)

$$v_i = \left\{ \begin{array}{l} \emptyset \\ 1 \end{array} \right\}$$
 $1 \leq l_s \leq 31$ ($l_s = \emptyset$: Tonausgabe über TV gesperrt)
 $1 \leq t_d \leq 255$ ($t_d = \emptyset$: Dauerton)
 Beispiel: SOUND 15,1,16,1,31,100

KAPITEL 21

Variablenfelder
(Anweisungen DIM, CSAVE *, CLOAD *)

DIMENSIONIERUNG

Bisher haben wir Variablen der Form X, H3, J\$ oder R benutzt. Treten jedoch größere Variablenmengen auf, so bedient man sich in der Mathematik indizierter Variablen, also solcher Variablen, die sich durch Indizes (kleine, am Fuß der Variablen befindliche Zahlen, wie z.B. x_1 , x_2) unterscheiden. Diese Möglichkeit bietet selbstverständlich auch unserer Computer.

Eine Gruppe von Variablen, die sich nur durch ihre Indizes unterscheiden, wird dabei ein Variablenfeld genannt. Der Index der Variablen kommt nicht nach unten versetzt "an den Fuß" der Variablen, sondern wird in Klammern hinter die Variable geschrieben (also $X(\emptyset)$, $X(1)$, $X(2)$ usw.).

Dieses Variablenfeld $X(i)$ ist eindimensional, da es nur einen Index enthält. Wir können jedoch auch 2-dimensionale oder n-dimensionale Variablenfelder vereinbaren. Diese besitzen dann 2 bzw. n-Indizes. Bei einem zweidimensionalen Variablenfeld wird auch der Name verständlicher. Unser oben angeführtes eindimensionales Feld könnte z.B. eine Reihe i Pflanzen darstellen. Nehmen wir jetzt jedoch einen zweiten Index für die Anzahl der Reihen dazu, so ergibt sich ein Feld auch anschaulich. Die Feldfestlegung oder besser Dimensionierung der Variablenfelder erfolgt mit Hilfe der Anweisung DIM. Das mit der Anweisung

```
10 DIM X(2,3)
```

dimensionierte zweidimensionale Variablenfeld besteht konkret aus folgenden Variablen:

```
X(0,0), X(0,1), X(0,2), X(0,3)
X(1,0), X(1,1), X(1,2), X(1,3)
X(2,0), X(2,1), X(2,2), X(2,3)
```

Die durch die Anweisung reservierten Speicherplätze der 12 Variablen sind alle auf \emptyset gesetzt.

Im weiteren Verlauf des Programms können wir jede einzelne Variable mit einem Wert belegen und mit diesen Variablen wie gewohnt operieren. Dies probieren Sie am besten einmal selbst aus, in dem Sie die obenstehende Programmzeile 10 zu einem Testprogramm ergänzen:

```
20 X(1,2)=8
30 FOR I=0 TO 2
40 FOR J=0 TO 3
50 PRINT X(I,J);
60 NEXT J
70 PRINT
80 NEXT I
```

Sie sehen, lediglich die Variable X(1,2) ist ungleich \emptyset , da ihr in der Zeile 20 unseres Programmes der Wert 8 zugewiesen wurde. So wie wir bisher numerische Variablenfelder betrachtet haben, können wir selbstverständlich auch Stringvariablenfelder anlegen. Das nächste Programm demonstriert am Beispiel eines eindimensionalen, aus 12 Elementen bestehenden Stringvariablenfeldeseinfache Möglichkeiten zur effektiven Nutzung von Variablenfeldern. Das Programm gibt Ihnen über Menü-Technik wahlweise die Möglichkeit, die Elemente des Feldes über die Tastatur einzugeben, sie als Datei vom Recorder einzulesen, die Elemente aufzulisten oder sie als Datei auf Magnetband zu speichern. Dabei lernen wir zwei uns bereits bekannte Anweisungen in einer neuen Form kennen:

```
CSAVE * "Name";Feldname
CLOAD * "Name";Feldname
```

Mit diesen Anweisungen kann das bezeichnete Variablenfeld (in unserem Beispiel A\$) auf Magnetband als Datei gespeichert und von dort auch wieder in den Computer geladen werden.

Die Vorgänge des Rettens und Ladens werden wie in Kapitel 9 für Programme beschrieben, ausgeführt. Wie das Beispiel zeigt, können diese Anweisungen nicht nur als Kommando, sondern auch geschickt als Programmanweisung eingesetzt werden.

Vor dem Laden eines Variablenfeldes ist dieses stets mit der Anweisung DIM festzulegen.

```
10 DIMA$(11)
20 CLS:PRINT "ARBEIT MIT STRINVARIABLENFELDERN":PRINT
30 PRINT "1 VARIABLEN EINGEBEN"
40 PRINT "2 VARIABLEN EINLESEN"
50 PRINT "3 VARIABLEN LISTEN"
60 PRINT "4 VARIABLEN RETTEN"
70 B$=INKEY$:IFB$=""GOTO70
80 B=VAL(B$):PRINTB
90 IF B=1 OR B=2 OR B=3 OR B=4 GOTO100:ELSEGOTO70
100 ON B GOTO110,140,170,220,
110 FOR I=0 TO 11
120 INPUTA$(I):NEXT
130 GOTO20
140 PRINT "RECORDER EINSCHALTEN !"
150 CLOAD * "TESTFELD";A$
160 GOTO20
170 FOR I=0 TO 11
180 PRINT "A$("I")=";A$(I):NEXT
```



```

190 PRINT:PRINT"EINGABE M FUER MENU"
200 M$=INKEY$:IFM$<>"M" GOTO200:ELSEGOTO20
210 GOTO20
220 PRINT"RECORDEREINSCHALTEN!":PAUSE50
230 CSAVE * "TESTFELD";A$
240 GOTO20

```

Insbesondere bei der Arbeit mit Variablenfeldern wird der vorhandene Speicherplatz schnell überschritten. Dabei wird für numerische Variablen der Speicherbereich nur durch die verfügbare Arbeitsspeicherkapazität begrenzt. Hier hilft dann nur noch eine Systemerweiterung durch RAM-Module.

Für Stringvariablen ist nach dem Start des BASIC-Interpreters ein Speicherbereich von 256 Bytes reserviert. Reicht die nicht aus, so können wir den Stringspeicherbereich leicht erweitern. Dazu erfahren Sie jedoch im nächsten Kapitel mehr.

Merke:

- **Anweisung:** DIM
 Format: DIM Variable (Index, . . . , . . .)
 Bemerkung: DIM reserviert Speicherplatz für Feldvariablen und setzt diese gleich Null. Die Liste der Indizes kann maximal eine Programmzeile lang sein. Der Wert jedes Ausdrucks gibt den größten zulässigen Index der Dimension an. Der kleinstmögliche ist 0. Es sind beliebig viele Indizes zulässig. Wird keine Dimension vereinbart, wird für jede angegebene Dimension ein größtmöglicher Index von 10 angenommen. Die Indizes sind durch Kommata getrennt in Klammern zu setzen.
 Beispiel: 10 DIM X(5,10)
- **Anweisung:** CSAVE *
 Format: CSAVE * "Name"; Feldname
 Bemerkung: Die Anweisung speichert das angegebene Feld auf Kassette ab. Sie kann auch als Programmanweisung verwendet werden.
 Beispiel: CSAVE * "DATEN";X
- **Anweisung:** CLOAD *
 Format: CLOAD * "Name"; Feldname
 Bemerkung: Die Anweisung lädt das angegebene Feld von der Kassette. Sie kann auch als Programmanweisung verwendet werden.
 Beispiel: CLOAD * "DATEN";X

KAPITEL 22

Inneneleben des Computers
(Anweisungen SWITCH, FRE, CLEAR, PEEK,
DEEK, POKE, CALL, USR)

Wenn wir uns mit dem Innenleben des Computers etwas vertraut machen wollen, müssen wir uns darüber im klaren sein, daß wir bis jetzt bei unserer BASIC-Programmierung noch gar nicht direkt mit dem "Gehirn" des Computers, dem U880D "gesprochen" haben. Dieser Mikroprozessorschaltkreis versteht nämlich nur seinen U880-Maschinencode. Der BASIC-Interpreter stellt das Bindeglied zwischen uns und dem Maschinencode dar. Für größere Programme und sehr schnelle Programme gibt es in BASIC jedoch einige Anweisungen, deren Verständnis ohne ein Mindestmaß an Kenntnissen über das Computer-Innenleben nicht möglich ist.

Der Computer benutzt intern nicht nur eine andere Sprache, sondern er zählt und rechnet auch anders. Dies liegt in seiner dualen Grundstruktur begründet.

Die kleinste Informationseinheit, die der Computer kennt, ist ein Bit. Ein Bit kann nur eine von zwei möglichen Informationen tragen. Diese Informationen können wir auch als Zahlen ansehen, also 1 oder 0. Zwei Bit können demnach 4 Zahlen zusammen darstellen, nämlich: 00, 01, 10, 11. Überlegen wir uns anhand des folgenden Schemas, wie diese Entwicklung weiter geht:

1 Bit	kann	2 ¹ =	2 Zahlen darstellen
2 Bit	können	2 ² =	4 Zahlen darstellen
3 Bit	können	2 ³ =	8 Zahlen darstellen
4 Bit	können	2 ⁴ =	16 Zahlen darstellen
8 Bit	können	2 ⁸ =	256 Zahlen darstellen

Dabei kommt den 4 Bit und 8 Bit als Einheit eine besondere Bedeutung zu. Jeweils 4 Bit faßt der Computer als eine Ziffer auf. Damit stehen dem Computer nicht nur 10 Ziffern wie üblich von 0 bis 9 zur Verfügung, sondern 16.

Diese heißen der Reihe nach:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Das interne Zahlensystem des Computers baut also auf die Basis-Zahl 16 auf. Es heißt deswegen, im Gegensatz zu unserem dezimalen oder 10-er Zahlensystem, hexadezimales Zahlensystem.

Hexadezimale Zahlen werden mit einem H hinter der Zahl gekennzeichnet. Beispiel:

$$23H = 2 \cdot 16^1 + 3 \cdot 16^0 = 32 + 3 = 35$$

Zwei mal vier Bit oder zwei hexadezimale Zahlen sind 8 Bit oder 1Byte. Unser Computerspeicher ist in jeweils 8 Bit, also in Bytes aufgeteilt. Jedes Byte besitzt eine eigene Adresse. Die Adressen bestehen aus hexadezimalen Zahlen. Der Prozessor kann direkt 216 Byte = 26 mal 2¹⁰ Byte = 64 KByte zu je 8 Bit adressieren (1 KByte = 1024 Byte). Die Adressen laufen dabei von 0000 bis FFFFH.

Mit folgenden kleinen Programmen können wir dezimale in hexadezimale bzw. hexadezimale in dezimale Zahlen umwandeln.

```

10 A$="0123456789ABCDEF"
20 Z$=""
30 INPUT"DEZIMALZAHL ?";D
40 R=D-16 * INT(D/16)
50 Z$=MID$(A$,R+1,1)+Z$
60 IFD=0THEN90
70 D=INT(D/16)
80 GOTO40
90 PRINT"HEXADEZIMALZAHL:";RIGHT$(Z$,LEN(Z$)-1)
100 GOTO20

10 INPUT"HEXADEZIMALZAHL?";Z$
20 L=LEN(Z$)
30 D=0
40 FOR I=1 TO L
50 T$=MID$(Z$,I,1)
60 IFT$>="A" AND T$<="F" THEN U=ASC(T$)-55:GOTO90
70 IF T$>="0" AND T$<="9" THEN U=VAL(T$):GOTO90
80 PRINT"KEINE HEXADEZIMALZAHL!":GOTO10
90 D=U * 16^(L-I)+D
100 NEXT
110 PRINT"DEZIMALZAHL !";D
120 GOTO10

```

Greifen Sie jedoch vom BASIC-Interpreter direkt zum Speicher, müssen Sie nicht unbedingt die hexadezimale Adresse angeben. Die Speicherübersicht des Buches enthält sowohl die dezimalen als auch die hexadezimalen Adressen.

Der Speicher selbst besteht aus zwei Arten, dem ROM und dem RAM. Aus dem ROM, dem Festwertspeicher, können wir nur Informationen auslesen, aber nichts hineinschreiben. Er enthält das Betriebssystem und den BASIC-Interpreter. Unter dem Betriebssystem verstehen wir die Gesamtheit der Grundprogramme, die nach dem Einschalten des Computers sofort selbstständig starten (z.B. Programme zur Tastaturabfrage, Programm zum Aussenden des Kontrollbildes u. ä.) bzw. verfügbar sind (z.B. Programme zur Zusammenarbeit mit dem Magnetbandgerät: SAVE , LOAD , VERIFY). Dieser Festspeicher hat im KC 85/3 einen Umfang von 16 KByte (10 KByte BASIC-Interpreter, 6 KByte Betriebssystem).

Im RAM, dem Arbeitsspeicher, befinden sich alle Programme, die wir mit dem Kassettengerät oder der Tastatur eingeben. Aus diesem RAM können wir sowohl lesen als auch Informationen hineinschreiben. Beim Ausschalten des Computers gehen jedoch sämtliche Programme dieses Speicherbereiches ver-

loren. Der RAM hat einen Umfang von 32 KByte. Diese 32 KByte stehen uns jedoch nicht uneingeschränkt zur Verfügung, da wir auch RAM-Speicherplatz für den Bildwiederholtspeicher und für die Grundprogramme benötigen. Aber dazu erfahren wir mehr im nächsten Abschnitt.

SPEICHERVERWALTUNG

Der U880D kann direkt einen Speicherbereich von 64 KByte adressieren. Die 16 KByte ROM und 32 KByte RAM der Grundausrüstung sind dabei wie folgt verteilt:

Speicheradressen Speicherbelegung

0000H – 3FFFH	RAM (Arbeits-RAM)
8000 – BFFFH	RAM (IRM ; Bildwiederholtspeicher)
C000H – FFFFH	ROM

Der erste 16 KByte RAM-Block, also der Arbeits-RAM, steht Ihnen für Ihre speziellen Programme fast vollständig frei zur Verfügung. Der zweite 16Kbyte-RAM-Block ist reserviert für den Bildwiederholtspeicher. Dieser enthält die Informationen für das Anzeigebild und ermöglicht die Vollgraphik von 320X256 Bildpunkten.

Der ROM enthält das Betriebssystem und den BASIC-Interpreter.

Im Bild 7 ist die Speicherverwaltung übersichtlich dargestellt.

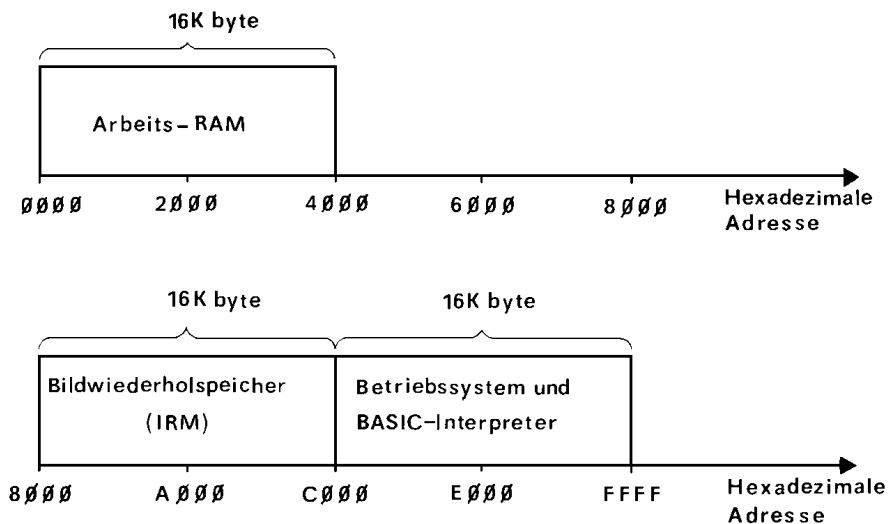


Bild 7: Graphische Darstellung der Speicheranordnung im KC 85/3

Die nicht belegten Speicheradressen sind für mögliche Speichererweiterungen vorgesehen.

Diese, wie auch die im Grundgeräte bereits enthaltenen Speicherblöcke können durch die Anweisung

SWITCH m, k

zu- oder abgeschaltet und schreibgeschützt werden. Die Parameter m (Steckplatzadresse) und k (Steuerbyte) sind, wie im System-Handbuch Kapitel 5 beschrieben, zu bilden und in dezimaler Form durch Komma voneinander getrennt einzugeben. Eine detaillierte Speicherbeschreibung finden Sie im System-Handbuch Kapitel 10.

Wie bereits dargelegt, müssen wir beim Erstellen größerer Programme oder bei der Verarbeitung großer Datenmengen mit dem uns zur Verfügung stehenden Speicherplatz haushalten. Dieses Bemühen unterstützt insbesondere die Funktion FRE. Mit Hilfe der Funktion FRE können wir zu jeder Zeit die Anzahl der noch freien Speicherplätze im RAM oder die Anzahl der freien Bytes im String-Speicherbereich erfahren. Erweist sich der Stringspeicherbereich als zu klein für ein geplantes Programm, so können wir diesen durch die Anweisung

Ausdruck, Ausdruck

neu festlegen. Dabei legt der Wert des ersten Ausdrucks fest, wieviel Bytes für Strings reserviert werden sollen. Mit dem zweiten Ausdruck kann die zugewiesene Grenze des oberen RAM-Bereiches neu bestimmt werden. Die beiden erwähnten Speicher-Anweisungen werden im letzten Abschnitt des Kapitels ausführlich beschrieben.

MASCHINENCODE UND BASIC

Aus der Speicheraufteilung ergeben sich auch die Inhalte. So sind z. B. im Bildwiederholtspeicher auf genau bestimmten Speicherplätzen die Farb-, Vordergrund- und Hintergrund-Information abgelegt. Diese Informationen bestehen in der Grundform, wie auch die Informationen der Programme, aus zweistelligen Hexadezimalzahlen. Sind die zu speichernden Werte oder Anweisungen länger als ein Byte bzw. zwei Hexadezimale Zahlen, so werden diese über mehrere Speicheradressen abgelegt.

Eine Einführung in die Arbeit mit dem hexadezimalen Maschinencode würde den Rahmen dieses Buches sprengen. Dazu verwenden Sie bitte folgende weiterführende Literatur:

- Kieser, H. ; Meder, M.: Mikroprozessortechnik
Aufbau und Anwendung des Mikroprozessorsystem U880
VEB Verlag Technik Berlin, 1982
- Cläßen:
Programmierung des Mikroprozessorsystems U880 – K1520
VEB Verlag Technik Berlin, 1981
(Reihe Automatisierungstechnik Band 192)
- Roth, M.: Mikroprozessoren
Wissenschaftliche Zeitschrift KdT.
Hochschulsektion TH Ilmenau 1979

Aber wir können auch ohne diese Spezialkenntnisse direkt aus dem Speicher lesen und eingeben. Die Werte werden mittels der Anweisungen PEEK und DEEK aus dem Speicher gelesen. Durch die Anweisungen POKE und DOKE werden Werte in die Speicherzellen eingelesen. Sowohl die Speicheradressen als auch die Speicherinhalte sind dabei in gewohnter dezimaler Schreibweise einzugeben bzw. werden in dieser ausgegeben. Mit Hilfe dieser Anweisungen kann man somit kleine Maschinenprogramme vom BASIC-Interpreter aus in den dafür geeigneten Speicherbereich schreiben.

Nicht geeignet ist der Bildwiederholungspeicherblock, da dieser bei der Arbeit des BASIC-Interpreters abgeschaltet wird. Hier können wir jedoch mit Hilfe der Anweisungen VPEEK Zellen aus dem Speicherinhalt im Bereich von 8000H bis BFFFH lesen und durch VPOKE Werte in diesem Adressbereich eintragen. Bei Verwendung dieser Anweisungen wird der Adressbereich des Bildwiederholungspeichers auf 0000H bis 3FFFH (0 bis 16383 dezimal) gelegt. Deshalb sind in den Anweisungen nicht die tatsächlichen Adressen, sondern die angesprochenen Adressen abzüglich – 8000H anzugeben. Da die Adressangaben dezimal erfolgen, müssen wir also von der dezimalen Adresse 32768 subtrahieren. Möchten wir z.B. den Inhalt der Speicherzelle BA00H (47616 dezimal) erfahren, so müssen wir von der dezimalen Adresse 47616 die oben genannten 32768 subtrahieren. Die in der Anweisung PEEK zu verwendende Adresse ergibt sich zu 14848.

Die Anweisung lautet:

```
PRINT PEEK(14848)
```

Mit Hilfe der eben genannten sechs Anweisungen sind wir also in der Lage, kleine Maschinenprogramme, Zeichenbildtabellen oder andere Dateien vom BASIC-Interpreter aus zu erstellen.

Solche Maschinenprogramme kann man dann auch innerhalb eines BASIC-Programms durch die Anweisung CALL oder die Funktion URS aufrufen und abarbeiten lassen.

Die Funktion URS (x) ruft ein Maschinenprogramm mit dem Argument x auf.

Dabei sind folgende Schritte erforderlich:

1. Speichern der Anfangsadresse des Maschinenprogramms auf Adresse 304H (L-Teil) und 305H (H-Teil)
(L-Teil... niederwertiger Teil, z.B. 80 bei 3080; H-Teil... höherwertiger Teil, z. B. 30 bei 3080)
2. Aufruf der URS-Funktion

Beispiel: Ein Maschinenprogramm wird mit dem MODIFY-Kommando des Betriebssystems KC 85/3-CAOS auf eine freie Adresse (z.B. 0H) eingegeben:

Adresse	MC	Anweisung	Bemerkung
0000	CD6FC9	START: CALL CPRVL3	; BASIC-UP zur Pa- ; rameterausgabe in ; das Register DE
0003	13	INC DE	; Von BASIC übergebene ; Parameter
0004	7A	LD A,D	; Reg. A, B-Funktions-
0005	43	LD B,E	; werückgabe
0006	CDB1D0	CALL FRE3	; BASIC-UPzur Funktions- ; wertzweisung an ; BASIC-Variable
0009	C9	RET	; Rücksprung in BASIC

Die Eingabe kann aber auch vom BASIC-Programm erfolgen:

```
10 DATA 205,111,201;REM Maschinencode
12 DATA 19,122,67;REM als Dezimalzahlen
14 DATA 205,117,13
16 DATA 201
20 AD=0:REM Anfangsadresse des Maschinenprogramms
30 FOR I=0TO9
40 READ B:POKE AD + I,B
50 NEXT
60 REM Speichern der Anfangsadresse auf 2A04H
70 DOKE 772, AD
80 REM Aufruf des Maschinenprogrammes
90 FOR I=250 TO 260
100 K=USR(I)
110 PRINT "I=";I;"USR(I)="; K
120 NEXT
```

Dieses Beispiel Maschinencode-Unterprogramm erhöht den eingegebenen Parameter um 1 (INC DE) und gibt diesen Wert als Funktionswert zurück.

Dieses Maschinencode-Unterprogramm funktioniert zunächst unabhängig von der Tatsache, daß der IRM beim Zugriff auf den BASIC-Arbeitsspeicher abgeschaltet wird. Verallgemeinert ergeben sich aus dieser Arbeitsweise des BASIC-Interpreters jedoch folgende Konsequenzen für den Aufruf von Maschinenprogrammen vom BASIC aus.

- Liegt ein Maschinenprogramm im IRM, so muß vor seinem Aufruf der IRM zugeschaltet werden. Dies kann durch ein weiteres Maschinenunterprogramm geschehen, das aber im Adreßbereich 0000H bis 7FFFH liegen muß.
- Soll von einem Maschinenunterprogramm aus, das im BASIC-Arbeitsspeicher liegt, auf den IRM zugegriffen werden, z.B. zur Realisierung von Bildschirm-ausschriften (Text und/oder Graphik), muß der IRM ebenfalls zugeschaltet werden.
- Vor der Rückkehr ins BASIC muß der IRM wieder abgeschaltet werden.

Die folgende Befehlsfolge realisiert das Ein- und Abschalten des IRM mit Hilfe spezieller Unterprogramme des Betriebssystems, die außerdem den STACK des BASIC-Interpreters verlegen und den Inhalt des BC-Registers zerstören.

CALL . 0F018H CD18F0 ; Einschalten des IRM

- . Zugriff zum IRM ist möglich
- . z.B. zum Maschinenunterprogramm dort bzw. zur Realisierung von
- . Bildschirmausschriften

CALL 0F01BH CD1BF0 ; Abschalten des IRM
RET C9 ; Rückkehr ins BASIC

Beispiel: Das folgende Beispiel ist ein Rahmenprogramm, mit dem es möglich ist, ein Maschinenunterprogramm im IRM aufzurufen. Es wird auf die freien Speicherplätze unterhalb des BASIC-Arbeitsspeichers gelegt. Das Maschinenunterprogramm beginnt auf Adresse 0BE00H im IRM.

Adresse	MC	Anweisung	Bemerkung
0000	CD6FC9	CALL CPRVL3	; UP zur Parameter- ; übergabe vom BASIC ; an das Maschinen- ; programm
0003	CD18F0	CALL 0F018H	; Zuschalten des IRM
0006	CD00BE	CALL 0BE00H	; Ruf des Anwender- ; Maschinenunterprogramms
0009	CD1BF0	CALL 0F01BH	; Abschalten des IRM
000C	7A	LD A,D	; In den Registern A und B
000D	43	LD B,E	; erfolgt die ; Parameter-Rückgabe
000E	CDB1D0	CALL FRE3	; UP zur Funktionswert- ; zuweisung an die ; BASIC-Variable
0011	C9	RET	; Rückkehr ins BASIC

Das Maschinenunterprogramm, das auf der Adresse 0BE00H beginnt, übernimmt den Parameter X der USR (X)-Funktion im DE-Register. Im obigen Beispiel wird der berechnete Funktionswert auch im DE-Register an das Rahmenprogramm zurückgegeben. Hierfür kann jedes Registerpaar, außer dem BC-Register, verwendet werden, da das BC-Register durch das Unterprogramm zum Abschalten des IRM zerstört wird. Das Maschinenunterprogramm ab Adresse 0BE00H realisiert hier die gleiche Funktion wie das bereits besprochene Beispiel: Es erhöht den eingegebenen Parameter um 1 (INC DE) und gibt diesen Wert als Funktionswert zurück.

Adresse	MC	Anweisung	Bemerkung
BE00	13	INC DE	; Von BASIC überge- ; bener Parameter ; wird um 1 erhöht
BE01		C9 RET	; Rückkehr zum ; Rahmenprogramm

Die Eingabe des Rahmenprogramms sowie des Maschinenunterprogramms kann vom Betriebssystem-Niveau aus mit MODIFY erfolgen.

Die Eingabe kann auch per BASIC-Programm geschehen was im folgenden dargestellt wird

```

10 REM Maschinencode des Rahmenprogramms
12 DATA 205,111,201,205,24,240,205, 0,190
14 DATA 205,27,240,122,67,205,177,208,201
20 REM Maschinencode des Maschinenunterprogramms
22 DATA 19,201
30 REM AD = Anfangsadresse des Rahmenprogramms
32 AD =0
34 REM MP = Anfangsadresse des Maschinenunterprogramms
  minus 8000H
36 MP = 15872
40 REM Eingabe des Rahmenprogramms über POKE
42 FOR I=0 TO 17
44 READ B: POKE AD + I, B
46 NEXT
50 REM Eingabe des Maschinenunterprogramms über VPOKE
52 READ B: VPOKE MP, B: READ B: VPOKE MP+1,B
60 REM Speichern der Anfangsadresse auf 304H
70 DOKE 772, AD
80 REM Aufruf des Maschinenunterprogramms über das Rahmen-
  programm
90 FOR I=250 TO 260
100 K = USR(I)
110 PRINT "I";I;"USR(I) =" ;K
120 NEXT

```

Statt der einfachen Erhöhung des Parameters I um 1 kann ab Adresse 0BE00H jedes beliebige andere Maschinenprogramm laufen. Dazu müßten die Zeilen 22 und 52 entsprechend geändert werden.

Achtung: Die angegebenen Adressen 304H, 305H, C96FH (UP CPRVL3) und D0B1H (UP FRE3) sind nicht für die Magnetbandversion des BASIC-Interpreters zum KC 85/3 gültig.

Merke:

- **Anweisung:** SWITCH
 Format: SWITCH m, k
 Bemerkung: Die BASIC-Anweisung SWITCH ermöglicht das Schalten von Modulen auch innerhalb eines Programms. In folgenden Angaben unterscheidet sich die SWITCH-BASIC-Anweisung von der SWITCH-System-Anweisung (vgl. System-Handbuch Kapitel 5):

	Systemanweisung	BASIC-Anweisung
SWITCH	<ul style="list-style-type: none"> - (%) Promptzeichen des Systems - Eingabe des Parameters m - Status lesen und schalten - Durch Leerzeichen getrennt - Modulsteckplatzadresse Angabe hexadezimal 	<ul style="list-style-type: none"> - (>) Promptzeichen von BASIC - Eingabepflicht der Parameter m und k - Nur schalten - Parameter durch Komma trennte Eingaben getrennt - Modulsteckplatzadresse Angabe dezimal

Beispiel:

Schalten des V24-Moduls, siehe M003 und C0171 Beschreibung SWITCH C1 0C EE 01 Drucken des Namen Emil 10 SWITCH 12,1 20 PRINT #2 "EMIL" 30 SWITCH 12,0	SWITCH 12,1 zuweisen V24-Modul Druckanweisung abschalten V24-Modul
--	---

- **Funktion:** FRE
 Format: FRE (v)
 Bemerkung: Die Funktion FRE gibt die Anzahl der noch freien Speicherplätze im RAM an. Ist v eine Stringvariable, wird die Anzahl der freien Bytes im Stringspeicherbereich angegeben.
 Beispiel: PRINT FRE (A\$)

- **Anweisung:** CLEAR
Format: CLEAR [Ausdruck] [,Ausdruck]
Bemerkung: CLEAR löscht den Variablenspeicher. Zusätzlich wird mit dem Wert des ersten Ausdrucks festgelegt, wieviel Speicherplätze für Strings reserviert werden sollen. Mit dem zweiten Ausdruck kann die obere Grenze des zugewiesenen RAM-Bereiches neu bestimmt werden. Wird kein Ausdruck angegeben, bleibt die Größe des Stringspeichers unverändert.
Beispiel: CLEAR 1000

- **Anweisung:** PEEK
Format: PEEK (i)
Bemerkung: PEEK (i) liest ein Byte von der Speicherstelle i
Beispiel: 10 E = PEEK (128)

- **Anweisung:** POKE
Format: POKE i, j
Bemerkung: Die Anweisung speichert das Byte j in der Speicherzelle i. Ist i negativ, wird es als 65536 + i interpretiert.
Beispiel: POKE 2,15

- **Anweisung:** DEEK
Format: DEEK (i)
Bemerkung: DEEK (i) liest je ein Byte von den Speicherzellen i und i+1.
Beispiel: 10 PRINT DEEK(218)

- **Anweisung:** DOKE
Format: DOKE i, j
Bemerkung: Die Anweisung speichert den Wert j in den Speicherzellen i und i + 1. sind i und j negativ, so werden sie als 65536+i bzw. 65636+i+1 interpretiert.
Beispiel: DOKE 0, 302

- **Anweisung:** CALL
Format: CALL Dezimaladresse
oder
CALL * Hexadezimaladresse
Bemerkung: CALL i ruft ein Maschinenprogramm mit der Startadresse i auf. Das Maschinenprogramm muß mit RETURN (0C9H) abgeschlossen sein und darf keine Register verändern. CALL *

wirkt wie CALL. Die Startadresse ist dabei jedoch hexadezimal anzugeben. Bei Verwendung von CALL muß gegebenenfalls beachtet werden, daß der Bildwiederholpeicher beim Erreichen des Maschinenprogramms abgeschaltet ist.

Beispiel: CALL * 30A0

● **Anweisungen:** VPEEK und VPOKE

Format: VPEEK (Adresse) und
VPOKE Adresse, Wert

Bemerkung: VPEEK gewährleistet das Lesen des Inhalts einer Speicherzelle im Bildwiederholpeicher und VPOKE das Beschreiben einer Speicherzelle im Bildwiederholpeicher. VPEEK und VPOKE sind erforderlich, um Speicherzellen im Bildwiederholpeicher zu lesen bzw. zu beschreiben, da dieser bei der Arbeit des BASIC-Interpreters abgeschaltet wird. Der Anfang des Bildwiederholspeichers liegt dabei auf Adresse 0 (vgl. PEEK, POKE).

Beispiel: 10 W = VPEEK (12)
20 PRINT W
30 VPOKE 25, W

● **Funktion:** USR

Format: USR (X)

Bemerkung: Die Funktion ruft ein Maschinenprogramm mit dem Argument X auf. Die Anfangsadresse des Maschinenprogramms ist auf die Speicherplätze 304H und 305H (772 und 773 dezimal), z.B. mit der Anweisung DOKE, zu speichern. Die Parameterübergabe vom BASIC-Programm zum Maschinenprogramm wird mit dem Unterprogramm CPRVL3 (Adresse C96FH) und die Parameterübergabe vom Maschinenprogramm mit dem Unterprogramm FRE3 (Adresse D0B1H) des BASIC-Interpreters realisiert.

Der Parameter vom BASIC wird in das Register DE übernommen und in den Registern A (H-Teil) und B (L-Teil) zum BASIC übergeben. Bei der Verwendung von USR muß gegebenenfalls beachtet werden, daß der Bildwiederholpeicher beim Erreichen des Maschinenprogramms abgeschaltet ist.

Beispiel: POKE773,38
K=USR(X)

KAPITEL 23

Arbeit mit der Peripherie

(Anweisungen OPEN, CLOSE, PRINT#, LIST#,
INPUT#, LOAD#, NULL, INP, OUT,
WAIT, JOYST, Funktion POS)

ZUSATZGERÄTE UND BASIC

Mit BASIC-Programmen können wir mit unserem KC 85/3 auch Zusatzgeräte, die Peripherie, ansteuern.

Eine Peripherie erhöht die Anwendungsbreite und den Wert des Computersystems enorm. So besitzen Sie z.B. mit einem anschließbaren Drucker und dem KC 85/3 eine elektronische Schreibmaschine. Zu den vom Hersteller angebotenen Zusatzgeräten wird stets eine ausführliche Beschreibung und Bedienungsanleitung mitgeliefert.

Im Gegensatz zu den im Kapitel 9 beschriebenen Anweisungen CSAVE und CLOAD, die speziell die periphere Einheit Kassettenrecorder ansprechen und eine rechnerinterne (übersetzte) Zeichendarstellung übermitteln, werden in diesem Kapitel allgemeingültige Anweisungen zum Betreiben von Zusatzgeräten beschrieben.

Mit Hilfe der Anweisung

```
LIST#n "Name"
```

lassen sich Programmlisten Zeichen für Zeichen auf ein durch die Zahl n bezeichnetes, frei wählbares Peripheriegerät (z.B. Drucker für $n=2$) ausgeben. Liegt nun ein auf diese Weise gespeichertes Programm z.B. auf Magnetband vor (LIST#1 "Name"), so können wir diese mit der Anweisung

```
LOAD#1 "Name"
```

wieder Zeichen für Zeichen in den Computer lesen. Dabei ergibt sich im Gegensatz zur Anweisung CLOAD der Vorteil, daß beim Nachladen von Programmen das zu ladende Programm entsprechend den Zeilennummern richtig bezüglich des bereits gespeicherten Programmes einsortiert wird.

Auch Daten können wir auf ein Peripheriegerät ausgeben und von dort wieder laden. Hierzu benötigen wir die Anweisungen OPEN, CLOSE, PRINT# und INPUT#. Durch die Anweisung

```
OPEN r#n "Name"
```

wird der Kanal zu dem durch n bezeichneten Peripheriegerät zur Datenausgabe oder zur Dateneingabe geöffnet. Das r ist im konkreten Fall durch ein O zur Datenausgabe bzw. durch ein I zur Dateneingabe zu ersetzen.

Ist der Kanal geöffnet, kann die Datenausgabe durch

```
PRINT#n Daten
```


oder die Dateneingabe durch

INPUT#n Daten

erfolgen. Ist die Datenübertragung beendet, wird der Kanal mit

CLOSE r#n

wieder geschlossen. Betrachten Sie dazu bitte noch das Beispiel im Abschnitt "Merke" dieses Kapitels.

Das in den Anweisungen enthaltene, zur Gerätezuweisung bestimmte, n ist wie folgt festgelegt:

0	Bildschirm und Tastatur
1	Kassettenrecorder
2 } 3 }	frei für weitere Anwendungen (Drucker, Plotter, Floppy, Lochstreifen- einheit usw.)

Zur Synchronisation einer langsamer arbeitenden peripheren Einheit kann vor der Datenausgabe eine NULL-Anweisung z.B.

NULL 60

eingegeben werden. Dadurch werden an jeder Zeile 60 bzw. die angegebene Anzahl "NULL-" oder auch "Dummyzeichen" angehängt. Für die Grundeinstellung sind 10 Dummyzeichen festgelegt. Diese bedeutungslosen Zeichen ermöglichen die synchrone Zusammenarbeit zwischen dem KC 85/3 und der peripheren Einheit. Die Anzahl der zu vereinbarenden Dummyzeichen ist abhängig vom Peripheriegerät.

Darüber hinaus stehen uns in BASIC zur Ansteuerung und Überwachung peripherer Einheiten die Anweisungen INP, OUT, WAIT, JOYST sowie die Funktion POS zur Verfügung. Die Handhabung dieser Anweisungen entnehmen Sie bitte dem folgenden Abschnitt "Merke".

Merke:

- **Anweisung:** OPEN
 Format: OPEN r#n "Name"
 Bemerkung: Open eröffnet eine Kanaloperation mit der Namensübergabe auf den Kanal n für dateifähige Geräte (z.B. Floppy) oder Geräte mit Namensverwaltung. Der Parameter r legt fest, ob es sich um eine Aus- oder Eingabe handelt:
 r = I Eingabe
 r = O Ausgabe
 Beispiel: siehe Anweisung PRINT#, INPUT#

- **Anweisung:** CLOSE
 Format: CLOSE r#n
 Bemerkung: CLOSE schließt den Kanal n (nach Dateneingabe r=I oder nach Datenausgabe r=O). CLOSE führt der BASIC-Interpreter automatisch bei Programmende oder -abbruch (Taste oder Error) aus.
 Beispiel: siehe Anweisungen PRINT#, INPUT#

- **Anweisung:** PRINT#, INPUT#
 Format: PRINT#n Daten
 INPUT#n Daten
 Bemerkung: PRINT# ermöglicht die Ausgabe von Daten auf einen durch n definierten Kanal.
 INPUT# ermöglicht die Eingabe der mit PRINT ausgegebenen Daten von einem durch n wählbaren Peripheriegerät. Beide Anweisungen sind nur in Verbindung mit OPEN und CLOSE (Übergabe des Dateinamens) zu verwenden. Ein Lesen der mit PRINT# ausgegebenen Daten ist nur durch die Anweisungen OPEN und CLOSE möglich. Sollen Daten jedoch nur ausgegeben werden (z.B. auf Drucker), können sie auch mit der Anweisung PRINT# ohne OPEN ausgegeben werden.
 Beispiel:
 1. 10 CLS:DIMA\$(4)
 20 FOR I=0TO4:INPUTA\$(I):NEXT
 30 PRINT"RECORDER EINSCHALTEN!"
 40 OPEN O#1 "BEISPIEL"
 50 FOR I=0TO4:PRINT#1 A\$(I):NEXT
 60 CLOSE O#1
 2. 10 CLS:DIMA\$(4)
 20 PRINT"RECORDER EINSCHALTEN!"
 30 OPEN I#1 "BEISPIEL"
 40 FOR I=0TO4:INPUT#1 A\$(I):NEXT
 50 CLOSE I#1

- **Anweisungen:** LIST#, LOAD#
 Format: LIST#n "Programmname"
 LOAD#n "Programmname"
 Bemerkung: LIST# ermöglicht die Ausgabe eines Programmlistings,(ASCII-Zeichen für ASCII-Zeichen) auf ein durch n wählbares Peripheriegerät.

Mit LOAD# können die mit LIST# ausgegebenen Programme wieder eingegeben werden. Durch n wird wiederum das Peripheriegerät festgelegt.

Beispiel: LIST#2 "KALKU" (Ausgabe des Programmlistings
KALKU auf Drucker)
LOAD#1 "TEST" Einlesen des mit LIST#1 "TEST"
auf Magnetband ausgegebenen
Programmlisting

- **Peripheriecodierung n** für die Anweisungen OPEN, CLOSE, PRINT# INPUT#, LIST# und LOAD#

n Peripherie

∅ Bildschirm und Tastatur
1 Kassettenrecorder
2 } frei für weitere Anwendungen (Drucker, Plotter, Floppy,
3 } Lochstreifeneinheit usw.)

- **Anweisung:** NULL
Format: NULL Zahl
Bemerkung: NULL legt die Anzahl der am Ende einer Zeile auszugebenden ASCII-Nullcodes (sogenannte Dummyzeichen, die keinen Einfluß auf das Programm haben) fest. Damit wird eine Synchronisation von Rechner und Peripherie erreicht.
Beispiel: NULL4∅
- **Anweisung:** INP
Format: INP(i)
Bemerkung: INP(i) liefert das aus dem Port i gelesene Byte ($0 \leq i \leq 255$)
Beispiel: A=INP(255)
- **Anweisung:** OUT
Format: OUT i,j
Bemerkung: Die Anweisung gibt das Byte j aus dem Port i aus. ($0 \leq i, j \leq 255$)
Beispiel: OUT 32, 1∅∅

- **Anweisung:** WAIT
Format: WAIT i, j, (k)
Bemerkung: WAIT hält den Programmablauf in einer Warteschleife bis am Port i das erwartete Bitmuster erscheint. Der eingelesene Wert wird exklusiv-oder-verknüpft mit k und anschließend und-verknüpft mit j. Ist das Resultat 0, bleibt das Programm in der Warteschleife, ansonsten wird es fortgesetzt. Wird k nicht angegeben, so ist k gleich 0.
Beispiel: WAIT 32, 2

- **Anweisung:** JOYST
Bemerkung: Der BASIC-Interpreter ist für diese Anweisung vorbereitet. Nähere Informationen zur Anweisung JOYST erhalten Sie in der Dokumentation zur geplanten Zusatzeinheit Joystick.

- **Funktion:** POS
Format: POS (i)
Bemerkung: Die Funktion gibt die aktuelle Schreibposition in der Zeile an. Die Position links außen ist gleich 0. Die Zahl i ist dabei ein Dummyargument (d.h. der Wert des Arguments hat keinen Einfluß auf die Funktion).
Beispiel: 1180 IF POS(2) = 30 GOTO 50

KAPITEL 24

Lösungen

In diesem Kapitel wird jeweils eine Lösung der in den Übungen zur Aufgabe gestellten BASIC-Programme vorgestellt. Sollten Sie ein anderes Programm als Lösung erarbeitet haben, welches die Aufgabe ebenfalls erfüllt, so ist dieses genauso richtig wie die hier aufgeführten Programme. Denn es gibt fast immer eine Vielzahl von Lösungsmöglichkeiten.

Kapitel 6 Übung 1

```
10 INPUT"BENZINVERBRAUCH IN L/KM=" ;A
20 PRINT"BEWERTUNG:" ;
30 IF A>=3 AND A<8 THEN PRINT "GUT"
40 IF A>=8 AND A<=13 THEN PRINT "SCHLECHT"
50 IF A<3 OR A>13 THEN PRINT "UNMOEGLICH"
```

Kapitel 6 Übung 4

```
10 CLS:PRINT"NAEHERUNGSVERFAHREN ZUR BERECHNUNG"
20 PRINT"DER QUADRATWURZEL NACH NEWTON" :PRINT
30 INPUT"ZAHL:" ;A
40 INPUT"GESCHAETZTE NAEHERUNG:" ;X
50 PRINT"NAEHERUNGEN : "
60 N=X-(X * X-A)/(2 * X)
70 IFABS (N-X) < 1E-3THEN90
80 X=N :PRINTN:GOTO60
90 PRINT :PRINT"DIE QUADRATWURZEL DER ZAHL"A
100PRINT"IST"N". "
```

Kapitel 7 Übung 2

```
10 ! DREIECKSBERECHNUNG
20 INPUT"A,B:" ;A,B
30 A=A * A:B=B * B
40 C=SQR(A+B)
50 PRINT"C=" ;C
60 END
```

Kapitel 7 Übung 3

```
10 ! QUADRATISCHE GLEICHUNG
20 INPUT"P,Q:" ;P,Q
30 S=-P/2
40 W=S * S-Q
50 IF W<0 THEN GOTO 100
60 W=SQR(W)
```

```
70 X1=S+W:X2=S-W
80 PRINT"X1=" ;X1
90 PRINT" X2=" ;X2:GOTO 110
100 PRINT"GLEICHUNG IM REELLEN ZAHLENBEREICH NICHT LOESBAR"
110 END
```

Kapitel 11 Übung 1

```
10 PAPER1:CLS
12 LINE0,128,319,128,7
14 LINE0,0,0,255
20 FOR X=0 TO 319
30 Y=128=50*SIN(X/159.5*PI)
40 PSETX,Y,7
50 NEXT
```

Kapitel 14 Übung 1

```
10 INPUTA$
20 INPUTB$
30 IF A$<=B$ GOTO 70
40 W$=A$
50 A$=B$
60 B$=W$
70 PRINTA$
80 PRINTB$
```

SACHWORTREGISTER

A

Adressen 116
AND 36
Arbeitsspeicher s. RAM
Aufnahmen eines BASIC
Programmes 56

B

BASIC 5, 7
Bildwiederholpeicher 116
Bit 114
BRK-Taste 21
Byte 114

C

Cursor-Tasten 8

D

DEL-Taste 8
Dimensionierung 110

E

Entertaste 8

F

Farbe 61
Fenster 72
Festwertspeicher s. ROM 116
Funktion
– mathematische 43
– RND 45
– String 82

G

Graphik 64
Grundrechenarten 16

H

Hexadezimaales Zahlen-
System 114
Hierarchie 42

I

Index 110
INS-Taste 8
Interpreter 7

K

Kaltstart 7
Kassettenrecorder 56
Konstante 12

L

Laden eines BASIC-
Programmes 58
Lautstärke 105

M

Maschinencode 117
Mehrfachprogramm-
Verzweigung 91
Mikroprozessor 114
Musik 105

N

NOT 37

O

Operatoren 36
OR 36

P

Programmablaufplan 100
Programmieren 99

R

RAM 115
REBASIC 7
RESET-Taste 9
Retten eines BASIC
Programmes 56
ROM 115

SACHWORTREGISTER

S

Scrolling-Modus
Speicher 116
– gliederung 116
STOP-Taste
String 82
– Operationen
– Speicher

T

Tabellen 107
Ton 105

U

Unterprogramm 88

V

Variable 12
Variablenfeld 110
Variablenname 12
Vergleichsoperatoren 38

W

Warmstart 7
Wert der Variablen 13

Z

Zeichenvorrat 79
Zufallszahlen 45