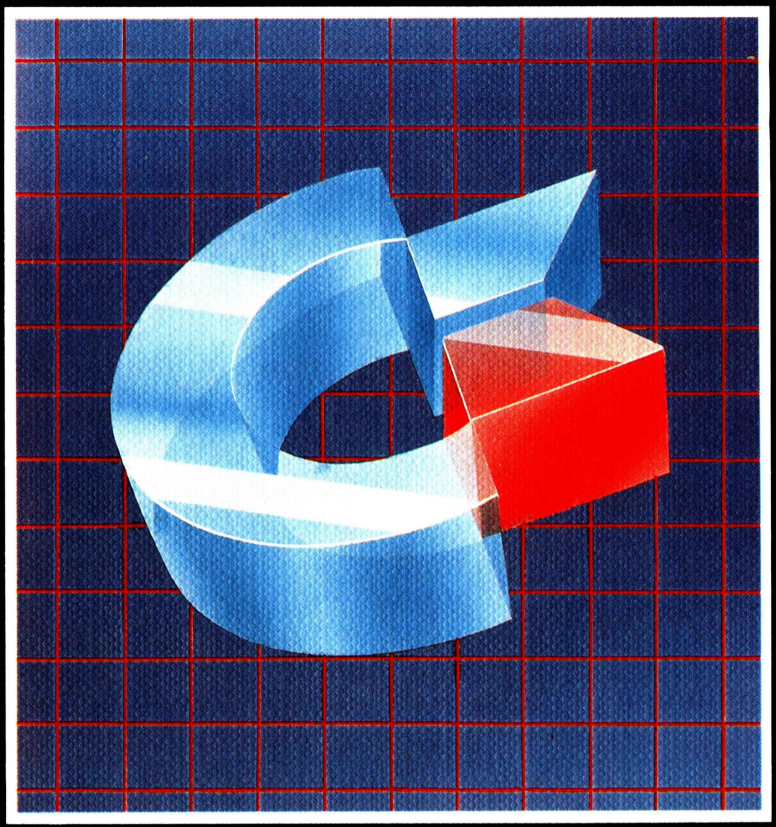
SYBEX

# Das große Commodore BASIC Handbuch



Michael Orkim

# Das große Commodore BASIC-Handbuch

# Das große Commodore BASIC-Handbuch

Michael Orkim


Düsseldorf · Berkeley · Paris

Satz: SYBEX-Verlag GmbH, Düsseldorf
Umschlaggestaltung: Daniel Boucherie/tgr - typo-grafik-repro gmbh., remscheid
Gesamtherstellung: Boss-Druck und Verlag, Kleve

Die Reprovorlagen für dieses Buch wurden auf einem Apple Macintosh erstellt und mit einem Apple LaserWriter ausgegeben.



ISBN 3-88745-615-7
1. Auflage 1986
2. Auflage 1986

# Inhaltsverzeichnis

# Einleitung

Dieses Buch ist für alle gedacht, die sich näher mit dem BASIC ihres Commodore-Rechners beschäftigen wollen. Ausgehend von der sehr umfassenden Version 7.0, die Commodore mit dem C128 zur Verfügung stellt, wird in die BASIC-Programmierung für alle Rechner dieses Herstellers eingeführt, wobei jeweils auf Unterschiede und Gemeinsamkeiten der einzelnen BASIC-Versionen detailliert eingegangen wird.

Wir wollen zeigen, wie leistungsfähig BASIC sein kann und daß es durchaus zu einem klaren und strukturierten Programmaufbau verwendet werden kann. Im vorliegenden Buch wird weitgehend auf PEEK- und POKE-Befehle verzichtet und demonstriert, daß die meisten maschinennahen Befehle elegant und sauber durch reine BASIC-Programmierung, frei von schwer verständlichen Tricks und Umwegen, ersetzt werden können.

Das Buch wendet sich daher an frischgebackene Besitzer der Rechner C128 mit BASIC 7.0 sowie C16, C116 und Plus/4 mit BASIC 3.5, die die beachtlichen Möglichkeiten ihrer Rechner durch eigene Programme für sich erschließen wollen. Und weiterhin stellt es eine wertvolle Informationsquelle für alle diejenigen dar, die Programme von anderen Commodore-Rechnern auf ihrem Rechner anpassen wollen. Es wendet sich außerdem noch an erfahrene Anwender, die auf neuere und leistungsfähigere Commodore-Geräte umsteigen wollen.

Trotz aller Unterschiede, die durch Hardware oder Betriebssystem bedingt sind, und trotz der ständigen Weiterentwicklung der Systeme seit 1979 kann man aufgrund der Gemeinsamkeiten im BASIC-Bereich und der Aufwärtsverträglichkeit von reinen BASIC-Programmen alle Commodore-Rechner als eine Familie von Rechnern betrachten. Ausnahmen bilden hier natürlich die IBM-kompatiblen Geräte PC10 und PC20, der 9000, der Amiga und der Micro Mainframe (MMF).

Den BASIC-Stammbaum der Commodore-Rechnerfamilie zeigt Abb. 1.

Wie aus der Abbildung ersichtlich wird, lassen sich BASIC-Programme in der Regel von kleineren oder älteren Systemen auf größere oder neuere übertragen. Unser Buch soll Ihnen auch dabei helfen. Sie finden in dem vorliegenden Buch eine Fülle von Informationen, die über die Handbücher hinausgehen, und viele Beispiele, die Sie vergessen lassen, daß Sie systematisch BASIC lernen.

| BASIC | 1.0 | 2.0 | 4.0 | 4.0+ | 3.5 | 7.0 |
|---|---|---|---|---|---|---|
| Jahr 1977 ↓ | PET2000 | CBM3000<br>VC20<br>C64<br>SX64<br>C4064 | CBM4000<br>CBM8000<br>CBM8296 | CBM600<br>CBM700 | C116<br>C16<br>Plus/4 | C128 |

*Abb. 1: Der Stammbaum der Commodore-Computer*

Kapitel 1 ermöglicht einen Schnelleinstieg in BASIC, der Sie mit einem Grundwissen für erste Programmierversuche ausstattet.

Kapitel 2 stellt Ihnen die Funktionsweise des BASIC-Sprachübersetzers und sein Zusammenspiel mit der Rechner-Hardware und den anderen Teilen der Betriebs-Software Ihres Computers vor.

Kapitel 3 führt Sie in lockerer Form anhand zahlreicher Beispiele in die Sprachregeln von BASIC ein.

Kapitel 4 rüstet Sie mit dem notwendigen Wissen für den Einsatz von BASIC-Befehlen für die Ein- und Ausgabe aus.

Kapitel 5 macht Sie damit vertraut, wie Sie die eingebauten Funktionen von BASIC und selbsterstellte Funktionen als leistungsfähige Sprachelemente innerhalb Ihrer Programme einsetzen können.

Kapitel 6 zeigt Ihnen, wie Sie das Kommando über Ihren Rechner übernehmen können und wie Sie wichtige Rechnerfunktionen direkt ansprechen. Sie lernen, in interaktiver Weise mit Ihrem Rechner zu arbeiten.

Kapitel 7 demonstriert, wie Sie unter Einsatz von externen Speichermedien die volle Leistungskraft Ihres Rechners für wirklich professionelle Programme zur Entfaltung bringen können.

Kapitel 8 erschließt Ihnen den gesamten Sprachumfang von BASIC durch detaillierte Erläuterungen und interessante Anwendungen jedes einzelnen BASIC-Befehls. Es dient Ihnen gleichzeitig als Nachschlagewerk und Fundgrube für zahlreiche Programmideen.

Kapitel 9 verhilft Ihnen schließlich zu einem guten Programmierstil, indem vorgestellt wird, wie man Programme strukturiert und wie man den Programmablauf steuern kann.

Kapitel 10 unterstützt Sie beim systematischen Entwickeln und Austesten Ihrer neuen Programme.

Kapitel 11 zeigt Ihnen, was Sie alles beachten müssen, wenn Sie Programme von einem System in ein anderes übernehmen wollen.

Die Anhänge A bis D bieten den kompletten alphabetisch gegliederten Befehlssatz der BASIC-Erweiterungen: Simon's BASIC, EXBASIC LEVEL II, Honey.Aid und Super-Expander. In weiteren Anhängen finden Sie eine Crossrefenz-Tabelle der BASIC-Versionen 2.0 bis 7.0 und ein Verzeichnis aller BASIC- Schlüsselwörter, gegliedert nach ihren Aufgaben.

## Kapitel 1

# Ein erstes einfaches Programm

Wir wollen nun gleich zur Praxis übergehen und ein einfaches Programm schreiben, um das BASIC in unserem Rechner kennenzulernen. Leser, die schon erste Erfahrungen mit BASIC gemacht haben, können den Rest des ersten Kapitels überschlagen.

Wir nehmen an, Sie haben Ihren Rechner eingeschaltet und sehen das Bereitschaftszeichen READY. bzw. ready. auf dem Bildschirm. Wir werden dem Rechner nun unseren ersten BASIC-Befehl erteilen. Tippen Sie

```
PRINT "HALLO"
```

ein. Es passiert zunächst nichts. Damit der Rechner diesen Befehl ausführt, müssen Sie die Taste drücken, die mit RETURN beschriftet ist. Nun reagiert er mit der Meldung:

```
HALLO

READY.
```

Die Anweisung, die wir als erstes ausprobiert haben, bestand aus zwei Teilen: aus dem BASIC-Schlüsselwort PRINT und aus der Mitteilung "HALLO", die in Anführungszeichen eingeschlossen war. Probieren Sie doch jetzt einmal eine andere Mitteilung auszugeben.

Anstatt immer einzelne Befehle oder Anweisungen auszuführen, was man auch Arbeiten im Direktmodus nennen kann, schreiben wir jetzt unser erstes Programm, was dann analog Arbeiten im Programmodus heißt. Dabei muß jede Zeile mit einer Zeilennummer anfangen, die zwischen 0 und 63999 liegen darf:

```
10 PRINT "HALLO"<RETURN>
20 PRINT "WIE GEHT ES?"<RETURN>
30 END<RETURN>
```

Obwohl wir hinter jeder Zeile die Taste RETURN gedrückt haben, was durch die Angabe von RETURN in eckigen Klammern symbolisiert sein soll, er-

scheint keine Ausgabe. Das liegt daran, daß ein Programm erst zu arbeiten beginnt, wenn es mit dem Befehl RUN gestartet wird. Tippen Sie nun

```
RUN<RETURN>
```

und es erscheint die Ausgabe:

```
HALLO
WIE GEHT ES?

READY.
```

Das eingegebene Programm wurde vom Rechner Zeile für Zeile in der Reihenfolge der Zeilennummern abgearbeitet.

Das kleine Programm befindet sich nach dem Eintippen im sogenannten Arbeitsspeicher des Rechners. Es kann nun beliebig oft ausgeführt, erweitert oder verändert werden. Zeilen, die Sie nachträglich einfügen wollen, brauchen nur eine geeignete Zeilennummer aufzuweisen. Der Rechner ordnet sie dann entsprechend ein. Geben Sie beispielsweise die Zeile

```
15 PRINT "GUTEN MORGEN"<RETURN>
```

ein, und sehen Sie sich das Ergebnis an.

Wir können das Programm immer wieder mit dem Befehl

```
LIST<RETURN>
```

auf dem Bildschirm anzeigen lassen. Es bleibt so lange im Rechnerspeicher stehen, bis wir den Computer ausschalten oder den Speicher mit

```
NEW<RETURN>
```

für ein neues Programm frei machen.

In unserem neuen Programm soll der Computer mit Zahlen rechnen, was ihm natürlich keine Probleme bereitet. Er kann addieren, subtrahieren, multiplizieren, dividieren, potenzieren u. v. m. Das Programm dazu könnte folgendermaßen aussehen:

```
10 PRINT "8 plus 4 =";8+4
20 PRINT "8 minus 4 =";8-4
30 PRINT "8 mal 4 =";8*4
40 PRINT "8 durch 4 =";8/4
50 PRINT "8 hoch 4 =";8↑4
60 END
```

Die Ergebnisse hätten wir natürlich zur Not auch im Kopf errechnen können. Nach dem Text, der in Anführungszeichen steht und der die Rechenaufgabe auf dem Bildschirm präsentieren soll, folgt ein Semikolon. Dieses bewirkt, daß der PRINT-Befehl mit seiner Ausgabe in derselben Zeile unmittelbar hinter dem Text fortfahren soll. Ein Komma anstelle des Semikolons würde einen Tabulatorsprung veranlassen. Hinter dem Semikolon wird die eigentliche Rechenaufgabe gelöst.

Das gerade besprochene Programm ist sehr starr und unflexibel, da es nur die 5 fest programmierten Aufgaben lösen kann. Wir könnten vielleicht auf die Idee kommen, daß wir lieber ein Programm hätten, das mit verschiedenen Zahlen, die wir ihm von Fall zu Fall vorgeben, rechnen soll.

Zur Lösung dieses Problems bietet sich in BASIC neben anderen der Befehl INPUT an, der vom Anwender eine Eingabe erwartet. Diese Eingabe, die je nachdem ein Text oder eine Zahl sein kann, wird in einer sogenannten Variablen abgelegt, die im Programm eine Platzhalterfunktion hat.

```
10 INPUT A
20 PRINT A;2*A;3*A
30 END
```

Wenn Sie das Programm mit RUN starten, erscheint auf dem Schirm ein Fragezeichen als Zeichen dafür, daß vom Benutzer eine Eingabe, in diesem Fall eine Zahl, erwartet wird. Diese Zahl wird in die Variable mit dem Namen A eingelesen, die fortan im Programm bis zu einer etwaigen Änderung mit diesem Wert behaftet bleibt. Den Wert einer Variablen kann man durch eine einfache Zuweisung ändern, die so aussehen kann:

```
25 A=8
```

oder komplizierter:

```
26 A=(599-67)/7*15
```

oder, auf den ersten Blick etwas verwirrend, so:

```
27 A=A+1
```

was bedeutet, daß der Wert von A um 1 erhöht werden soll.

Wir haben vorhin gesagt, daß vom Benutzer die Eingabe einer Zahl erwartet wird. Das liegt daran, daß die Variable, die wir gewählt haben, einen entsprechenden Namen hat, nämlich A. Wenn der Name einer Variablen mit einem Dollarzeichen endet, wie z. B. A$ oder ERNA$, dann kann sie Zeichenketten

und nur diese aufnehmen. Über gültige Variablennamen informieren Sie sich bitte in Kapitel 3.

Zur Verbesserung des Programms möchten wir nun 2 Zahlen eingeben und diese miteinander multiplizieren. Wir löschen das alte Programm mit NEW und fangen unsere Programmnumerierung zur Abwechslung einmal mit 200 an:

```
200 INPUT A,B
210 PRINT "ERSTE ZAHL:";A
220 PRINT "ZWEITE ZAHL:";B
230 PRINT A;"*";B;"=";A*B
240 END
```

Die beiden Zahlen, die in den Variablen A und B gespeichert werden sollen, werden nach dem Fragezeichen, durch ein Komma getrennt, eingegeben.

Eine Weiterentwicklung dieses kleinen Multiplikationsprogramms könnte der Einbau einer sogenannten Schleife sein, die es uns ermöglicht, die gleichen Anweisungen immer wieder, so oft, wie wir es wünschen, zu durchlaufen.

Wir beschließen, daß die Multiplikation von zwei Zahlen zehnmal durchgeführt werden soll und geben dafür zwei weitere Programmzeilen ein:

```
190 FOR I=1 TO 10
```

und

```
235 NEXT I
```

In diesem Zusammenhang spricht man von der Variablen I als von einer Lauf- oder Zählvariablen, die in unserem Beispiel nacheinander die Werte 1 bis 10 annimmt. Alle Anweisungen zwischen der FOR-Anweisung und dem zugehörigen NEXT werden so oft wiederholt ausgeführt, bis I die Grenze, in diesem Fall 10, überschritten hat. Dann fährt das Programm die Bearbeitung mit der Zeile fort, die auf das NEXT folgt.

Im besonderen Fall kann sich in einer Schleife wieder eine Schleife befinden. Dann spricht man von verschachtelten Schleifen. Dabei ist unbedingt auf die richtige Reihenfolge der NEXT-Anweisungen zu achten. Die innere Schleife muß als erste durch NEXT abgeschlossen werden, etwa so:

```
100 FOR I=1 TO 10
110 : PRINT "AEUSSERE SCHLEIFE"
120 : FOR J=1 TO 10
```

```
130 :   PRINT "INNERE SCHLEIFE"
140 : NEXT J
150 NEXT I
160 END
```

Die Doppelpunkte in den Zeilen 110 bis 140 haben dabei keine Bedeutung. Sie verhindern nur das gleichmäßige Einrücken aller Zeilen und dienen so der besseren Demonstration der Schleifenkonstruktion.

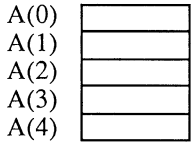
Eine Schleife kann auch sehr nützlich sein, wenn man in einem Programm zunächst viele Eingabewerte einlesen möchte, die erst später verarbeitet werden sollen. Dabei könnte man natürlich jedem Eingabewert eine Variable mit einem anderen Namen zuordnen: A, B, C, D usw. Das ist aber, wie man leicht einsieht, nicht besonders praktisch. BASIC stellt uns hierfür die indizierten Variablen zur Verfügung:

```
100 DIM A(20)
110 FOR I=1 TO 20
120 : INPUT"GEBEN SIE EINEN WERT EIN";A(I)
130 NEXT I
```

In der in Zeile 100 mit 20 dimensionierten Variablen A haben nun 20 Werte Platz, die über die Namen A(1), A(2), A(3), ..., A(20) angesprochen werden können. So können wir die eingegebenen Werte beispielsweise verdoppeln und auch wieder ausdrucken:

```
140 FOR I=1 TO 20
150 : PRINT 2*A(I)
160 NEXT I
170 END
```

Die dimensionierte Variable A(I) in unserem Beispiel, auch Feldvariable genannt, ist eine eindimensionale Variable. Man kann sie sich auch als Liste vorstellen, z.B. DIM A(4):

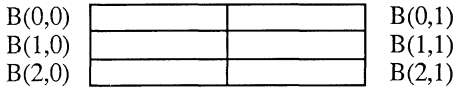



Die Anzahl der Feldelemente ist immer um 1 höher als der höchste Index. Wir haben in unserem Beispiel also 5 Elemente.

Außer den eindimensionalen Feldern gibt es noch mehrdimensionale Felder. Ein zweidimensionales Feld wird z. B. folgendermaßen definiert:

```
DIM B(2,3)
```

Dieses zweidimensionale Feld kann man sich als Tabelle mit 2 Spalten und 3 Zeilen vorstellen:




Felder mit 3 Dimensionen kann man sich noch räumlich vorstellen, aber bei 4 oder mehr Dimensionen hört die Vorstellungskraft auf.

Wir haben vorhin erwähnt, daß es außer dem Befehl INPUT für das Einlesen von Daten noch andere Befehle gibt. Ein anderer, der allerdings ganz anders arbeitet als der INPUT-Befehl, ist READ. Mit READ können Daten gelesen werden, die sich im Programm selbst in DATA-Zeilen befinden:

```
100 FOR I=1 TO 20
110 : READ A(I)
120 NEXT I
500 DATA 1,2,3,4,5,6,7,8,9,10
510 DATA 11,12,13,14,15,16,17,18,19,20
```

Dabei ist es gleichgültig, in welcher Zeile die DATA-Werte stehen. Das Programm sucht immer den ersten und liest dann einen nach dem anderen. Es muß allerdings gewährleistet sein, daß sich genauso viele Angaben in den DATA-Zeilen befinden, wie der READ-Befehl lesen will. Selbstverständlich können wir die so eingelesenen Daten auch wieder auf dem Bildschirm drucken:

```
130 FOR I=1 TO 20
140 : PRINT A(I)
150 NEXT I
160 END
```

Die READ/DATA-Konstruktion bietet eine simple Möglichkeit, wenn man ein Programm zusammen mit seinen Daten abspeichern will. Denn der Wunsch, ein Programm dauerhaft auf einem Datenträger abzulegen, wird bei Ihnen vielleicht auch allmählich gewachsen sein.

Je nachdem, ob Sie an Ihrem Commodore-Computer einen Kassettenrecorder oder ein Diskettenlaufwerk angeschlossen haben, gibt es zum Speichern oder Laden von Programmen verschiedene Befehle. Für das Speichern eines Programms mit dem Kassettenrecorder gibt es bei allen Commodore-Rechnern den Befehl SAVE:

```
SAVE "Programmname"
```

Unter seinem Namen kann das Programm dann auf der Kassette auch wieder gefunden werden. Achten Sie nur darauf, daß die Kassette mindestens bis zum Programmanfang wieder zurückgespult ist. Dann laden Sie das Programm mit dem Befehl

```
LOAD "Programmname"
```

wieder in den Rechnerspeicher.

Der Befehl zum Speichern von Programmen auf Diskette lautet außer in BASIC 2.0:

```
DSAVE "Programmname"
```

und zum Laden:

```
DLOAD "Programmname"
```

Dabei muß man beachten, daß eine neue Diskette zunächst mit dem Befehl HEADER formatiert werden muß:

```
HEADER "Diskettenname",IKennung
```

Der Diskettenname von maximal 16 Zeichen Länge kann dabei frei vergeben werden, und die Kennung muß aus zwei Zeichen bestehen.

Zum Schluß kann es noch wichtig sein, das Inhaltsverzeichnis einer Diskette einzusehen. Das bewerkstelligt der Befehl:

```
DIRECTORY
```

Bei den Rechnern C64 und VC20 mit BASIC 2.0 sehen diese Diskettenbefehle etwas anders aus. Der Befehl zum Speichern eines Programms lautet da:

```
SAVE "Programmname",8
```

und zum Laden entsprechend:

```
LOAD "Programmname",8
```

Das Inhaltsverzeichnis kann nur mit

```
LOAD"$",8
LIST
```

angesehen werden, und die Routine zum Formatieren einer leeren Diskette wird durch den Befehl

```
OPEN 1,8,15,"N:Diskettenname,Kennung"
```

ausgelöst.

Wir haben nun mit nur ein paar BASIC-Wörtern funktionsfähige Programme geschrieben und vielleicht sogar auf Kassette oder Diskette gespeichert. So schnell geht das in BASIC. Natürlich sind die Möglichkeiten noch wesentlich vielfältiger, als wir das hier für den Anfang zeigen wollten. Was Sie sich auch ausdenken, für (fast) jedes Problem bietet BASIC eine Lösung. Der Befehlsvorrat des Commodore-BASIC ist so umfangreich, daß Sie ihn wohl schwerlich intensiv ausschöpfen werden. Um ihn und seine Funktionen zu beschreiben und an Beispielen zu demonstrieren, wurde dieses Buch geschrieben, das Sie von den Programmieranfängen bis zu professionellen Lösungen begleiten soll.

Kapitel 2

# Grundlagen für Neugierige - Zusammenspiel von Interpreter, Betriebssystem und Hardware

Der eilige Leser kann dieses Kapitel überschlagen und erst bei späterem Anlaß darauf zurückkommen.

## Platzverteilung im Rechner

Die Commodore-Rechner zeichnen sich dadurch aus, daß sie das Betriebssystem und das BASIC komplett im Festwertspeicher enthalten. Daher ist praktisch sofort nach dem Einschalten der Rechner betriebsbereit. Das ist nicht bei allen Rechnern so. Es gibt einige, bei denen man zunächst das Betriebssystem laden muß, und bei manchen muß auch noch das BASIC geladen werden.

Da Ihr Rechner oder genauer der in ihm enthaltene Prozessor nur eine begrenzte Anzahl von Speicherzellen adressieren kann, verringert sich natürlich durch die permanent im Speicher vorhandene Software der für die BASIC-Programme zur Verfügung stehende Platz entsprechend.

Eine Speicherstelle Ihres Commodore-Rechners kann genau eine achtstellige Binärzahl, also 0000 0000 bis 1111 1111, aufnehmen, d. h. sie kann als größte Zahl dezimal ausgedrückt 255 enthalten. Eine Informationseinheit in computergerechter Darstellung mit 8 Binärstellen nennt man auch Byte. Eine einzelne Stelle eines Bytes wird als Bit bezeichnet. Ein Bit kann stets nur die Werte 0 oder 1, ein Byte nur Werte zwischen 0 und 255 enthalten.

Binärzahlen können auch als Summe von Potenzen der Zahl 2 dargestellt werden, so wie man Dezimalzahlen als Summe der Potenzen von 10 aufschlüsseln kann.

Die Dezimalzahl 1 entspricht der Binärzahl:

0000 0001 - 0*2↑7+0*2↑6+0*2↑5+0*2↑4+0*2↑3+0*2↑2+0*2↑1+1*2↑0

Die Dezimalzahl 255 entspricht der Binärzahl:

1111 1111 - 1*2↑7+1*2↑6+1*2↑5+1*2↑4+1*2↑3+1*2↑2+1*2↑1+1*2↑0

Wie viele Bytes für Ihre Programme übrigbleiben, zeigt Ihnen der Rechner mit der Meldung 122365 BYTES FREE beim C128 oder z. B. 38911 BYTES FREE beim C64 gleich nach dem Einschalten an. Den verfügbaren Arbeitsspeicher (RAM - Random Access Memory) können Sie auch jederzeit mit der Abfrage

```
PRINT FRE(0) <RETURN>
```

feststellen. Falls aber der Wert für den Speicherplatz größer als 32768 ist, die größte mit 2↑15 darstellbare Zahl, so wird die vorderste Stelle, nämlich 2↑16, einfach weggelassen. Um die richtige Anzahl verfügbarer Bytes zu erfahren, sollten Sie deshalb

```
PRINT FRE(0)+2↑16 <RETURN>
```

eingeben.

*Hinweis:* Der Befehl PRINT kann außer als Bestandteil von `PRINT#` immer als Fragezeichen eingegeben werden. Das ist kürzer und bequemer:

```
? FRE(0)+2↑16 <RETURN>
```

Auf diese Weise können Sie also immer feststellen, wieviel Speicher Sie noch zur Verfügung haben. Natürlich verringert sich der verfügbare Speicher mit der Zeit, denn alle Programme, die Sie selbst eingeben oder von einem peripheren Speichergerät wie Diskettenlaufwerk oder Kassettenrecorder einlesen, belegen ebenfalls einen Teil des Arbeitsspeichers.

Sie können leicht überprüfen, daß Sie vor dem ersten Lauf eines Programms mehr Speicherplatz haben als hinterher. Er hat sich während des Programmlaufs verringert. Es liegt in der Funktion eines Programms, daß es Informationen in Form von Daten in irgendeiner Form verarbeitet. Um diese Daten zu lesen oder zu verändern, muß das Programm sie zumindest zeitweise im Arbeitsspeicher ablegen, was teilweise drastische Anforderungen an den noch freien Platz stellt.

Selbstverständlich benötigen auch das Betriebssystem und das BASIC Datenbereiche im Speicher, um Informationen zwischenzuspeichern und zu verarbeiten. Dieser Speicherbereich ist zwar relativ klein und von einer fixen Größe, wird dafür aber, wie Sie sich denken können, intensiv angesprochen.

Sie brauchen nur zu bedenken, daß alle gewünschten Aktionen Ihres BASIC-Programms für den Prozessor erst verständlich werden, nachdem sie durch das eingebaute BASIC interpretiert wurden und dann zur eigentlichen Ausführung an Routinen des Betriebssystems übergeben werden.

Diese Betriebssystem-Routinen sind Gruppen von Anweisungen in der Sprache des Prozessors. Dieser versteht nichts anderes als Folgen von 0 und 1, die verschlüsselte Befehle darstellen. Anweisungen und auch Daten müssen ihm in dieser Form präsentiert werden.

Natürlich ist der BASIC-Interpreter selbst auch ein solches Maschinenprogramm, das unmittelbar vom Prozessor ausführbar ist.

Sie sehen also, daß BASIC-Interpreter und Betriebssystem äußerst nützliche Schwerarbeiter sind, die Sie erfolgreich vom komplizierten Innenleben Ihres Rechners abschirmen und es Ihnen ermöglichen, sich bei der Entwicklung von Programmen auf die Aufgabenstellung aus Anwendungssicht zu konzentrieren.

## Maschinenunabhängigkeit

Die tatsächlichen Abläufe in Ihrem Rechner sind jedoch erheblich komplizierter, als wir es hier betrachten wollen. Sie brauchen sich jedoch, solange Sie in BASIC oder einer anderen sogenannten höheren Programmiersprache wie z. B. Logo, Pascal, Pilot oder Comal programmieren, nicht darum zu kümmern.

Sie sollten es im Gegenteil sogar vermeiden, auf spezifische interne Gegebenheiten Ihres Commodore-Rechners in Ihren Programmen explizit einzugehen. BASIC bietet Ihnen zwar auch die Möglichkeit, mit den maschinennahen Befehlen WAIT, PEEK und POKE die Spezifika Ihres Rechners voll auszureizen, Sie erkaufen sich diesen Vorteil jedoch dadurch, daß Ihre Programme in hohem Maße von einem bestimmten Rechner abhängig werden. Außerdem werden sie schwer verständlich, und Sie benötigen bedeutend mehr Zeit für die Entwicklung und Veränderung Ihrer Programme.

BASIC 7.0 und die verfügbaren Erweiterungen für die älteren Commodore-BASIC-Versionen stellen Ihnen jedenfalls einen so umfangreichen Befehlsvorrat zur Verfügung, daß Sie Ihre Vorstellungen auch ohne große Tricks realisieren können. Aus diesem Grunde verzichten wir auch in diesem Buch auf solche Verrenkungen und zeigen Ihnen dafür, wie Sie die gleichen Effekte in BASIC erzielen können.

Man muß allerdings erwähnen, daß Programme, die in BASIC oder einer anderen höheren Programmiersprache geschrieben sind, aus den gerade erläu-

terten Gründen in der Ausführung langsamer sind als Programme, die in einer maschinennahen Sprache geschrieben sind.

Deshalb kann es für sehr anspruchsvolle und zeitkritische Anwendungen sinnvoll sein, Maschinenprogramme, die aus einem BASIC-Programm aufgerufen werden können, für bestimmte Teilaufgaben einzusetzen. Hierfür stellt BASIC die Befehle USR und SYS sowie BLOAD und BSAVE zur Verfügung. Wir empfehlen Ihnen jedoch unbedingt, einen guten Makro-Assembler einzusetzen.

## Interpretation und Kompilierung

Falls primär Geschwindigkeitsüberlegungen der Grund für Teillösungen in Maschinensprache sind, sollte der Einsatz eines BASIC-Compilers, wie z. B. Austro Comp, Petspeed oder DTL, erwogen werden. Diese Compiler verringern nämlich den Zeitaufwand, der zur Abarbeitung der Anweisungen eines BASIC-Programms benötigt wird, erheblich.

Sie müssen bedenken, daß der eingebaute BASIC-Interpreter jedesmal Zeile für Zeile und Anweisung nach Anweisung übersetzt, wenn Sie Ihr Programm ausführen lassen. Dieser Zeitaufwand fällt bei einem reinen Compiler jedoch praktisch weg, da dieser Ihr Programm in einem einmaligen Übersetzungslauf in eine durch den Prozessor Ihres Rechners direkt ausführbare Form umwandelt. Das Programm können Sie dann als Maschinenprogramm abspeichern und in dieser Form jederzeit wie jedes andere Maschinenprogramm mit SYS starten.

Der BASIC-Interpreter arbeitet jedoch nicht ganz so ineffizient, wie wir das vereinfachend dargestellt haben, und leider sind nicht alle Compiler vollwertige Übersetzer.

Der Editor-Teil des eingebauten Interpreters erkennt nämlich schon bei der Eingabe eines Programms die Befehlswörter von BASIC und wandelt sie in eine intern verschlüsselte Form, in sogenannte Tokens, um. Genau wie allen Zeichen des Commodore-Zeichensatzes ein bestimmter Byte-Wert, d. h. genau ein Muster von 8 Bits, als rechnerinterne Repräsentation zugeordnet ist, gibt es auch für alle BASIC-Befehle und andere BASIC-Schlüsselwörter eine eindeutige maschineninterne Codierung als Bitfolge.

Für alle Commodore-Rechner bis auf den C128 reicht dafür ein Byte aus. Der C128 aber benötigt aufgrund seines umfangreichen BASIC-Sprachschatzes und der Tatsache, daß Bytewerte, die kleiner sind als 128, zur Codierung der Zeichen des CBM-ASCII-Codes reserviert sind, für einige Schlüsselwörter

zwei Bytes. Dies ist übrigens auch bei einigen Spracherweiterungen für das Commodore-BASIC der Fall.

In dieser verkürzten Form werden die Programme auch abgespeichert, so daß der reine Programmtext in abgespeicherter Form wesentlich kompakter ist, als er auf dem Bildschirm angezeigt wird. Es werden beispielsweise auch keine Leerstellen, die sich am Ende von Programmzeilen befinden, abgespeichert.

Für die Anzeige auf dem Bildschirm oder bei der Ausgabe auf dem Drucker stellt BASIC den LIST-Befehl zur Verfügung, der die Tokens wieder in die Langform der Befehlswörter zurückverwandelt.

Die meisten BASIC-Befehle können abgekürzt durch Eingabe ihres ersten und des geshifteten zweiten (SHIFT + Buchstabe) Buchstabens oder durch Eingabe der ersten zwei und des geshifteten dritten eingegeben werden:

```
directory
```

ist gleichbedeutend mit

```
diR
```

Diese zulässigen Abkürzungen werden selbstverständlich intern in dieselben Tokens umgewandelt wie die äquivalente ausführliche Schreibweise. Das erklärt somit, warum die mit Abkürzungen eingegebenen Befehle nach dem Aufruf des Programms mit LIST wieder in der Langform präsentiert werden.

Auch bei den BASIC-Erweiterungen werden die neu hinzukommenden Befehle in Token abgelegt. Leider ist die Token-Verschlüsselung bei gleichen Befehlswörtern von verschiedenen Erweiterungen und zum Teil auch zwischen verschiedenen Commodore-Rechnern unterschiedlich. Daher müssen die Tokens bei der Übernahme von Programmen auf andere Rechner entsprechend angepaßt werden.

Einige der angebotenen Compiler führen die Idee der Verdichtung von BASIC-Programmen konsequent weiter und fassen ganze Gruppen von Befehlen zusammen. Sie verschlüsseln das Programm in eine noch kompaktere Form, die allerdings dann keine Rückwandlung in den ursprünglichen Programmtext mehr erlaubt. Diese so erzeugte Form des Programms ist jedoch noch nicht unmittelbar durch den Prozessor ausführbar, sondern muß jeweils zur Ausführungszeit des Programms noch einmal durch ein spezielles sogenanntes Laufzeitsystem, das Bestandteil des Compiler-Paketes ist, umgewandelt werden.

Dieser Vorgang läuft jedoch viel schneller ab als die Interpretation reiner BASIC-Programme. Diese verdichtete Form wird oft als Zwischencode oder

auch als Speedcode bezeichnet. Der Nachteil gegenüber echter Kompilierung ist jedoch die deutlich längere Ausführungszeit und der zusätzliche Platzbedarf, der bei kleinerer und mittlerer BASIC-Programmgröße ins Gewicht fällt. Nur bei sehr großen Programmen ist dieses Verfahren speicherplatzgünstiger als ein reines BASIC-Programm oder die Verwendung echter kompilierter Programme.

Bei der Anschaffung eines Compilers sollten Sie zusätzlich noch die folgenden Kriterien berücksichtigen:

- Versteht der Compiler alle Befehle Ihrer BASIC-Version und Ihrer BASIC-Erweiterungen und führt sie genauso aus?

- Unterstützt der Compiler Overlays, d. h. das Nachladen von Programmteilen?

- Erlaubt der Compiler das nachträgliche Zusammenfügen von getrennt übersetzten Programmteilen und/oder mit anderen BASIC-Programmen und/oder Maschinenprogrammen?

- Erstellt er Crossreferenzen und Speicherbelegungsübersichten?

- Gibt er Fehlermeldungen mit Bezug auf die ursprüngliche BASIC-Programmzeile und den verursachenden BASIC-Befehl heraus?

## Dem Interpreter auf die Finger geschaut

### Zusammenspiel von Betriebssystem und BASIC-Interpreter bei der Programmeingabe

Jeder Rechner, auch wenn er nur für Spiele eingesetzt wird, benötigt für die Ausführung selbst einfachster Aufgabenstellungen mindestens ein Ausgabemedium für Ergebnisse und eine Eingabemöglichkeit für Befehle und Daten.

Eins der häufigsten Eingabegeräte ist die Tastatur. Bei der Erstellung eines BASIC-Programms wird jedes Zeichen, das Sie über Tastatur eingeben, zunächst in einem Zwischenspeicher, dem Tastaturpuffer, abgelegt. Solange keine BASIC-Programme oder -Befehle abgearbeitet werden, befindet sich Ihr Commodore-Rechner immer im Eingabemodus. Das heißt, er wartet auf die Eingabe von Zeichen und zeigt sofort nach Betätigen einer Taste eine Reaktion.

Jeder erkannte Tastenanschlag führt nämlich außer zur Ablage im Tastaturpuffer auch zum Abspeichern im Bildschirmspeicher, der mit Ausnahme des

80-Zeichen-Modus beim C128 einen Teilbereich des normalen Arbeitsspeichers darstellt. Das Zeichen erscheint also unmittelbar nach Drücken der Taste auf dem Bildschirm, da der Inhalt des Bildschirmspeichers sozusagen identisch mit der Bildschirmanzeige ist.

Der Tastaturpuffer ist in der Größe beschränkt, deshalb leert ihn das Betriebssystem fast gleichzeitig mit der Anzeige eines Zeichens auf dem Bildschirm und übergibt das Zeichen an den BASIC-Eingabepuffer. Dort wird so lange Zeichen an Zeichen gereiht, bis Sie die RETURN-Taste betätigen.

Dann stoppt das Betriebssystem den Transfer von weiteren Zeichen aus dem Tastaturpuffer, was Sie aber nicht daran zu hindern braucht, dort weitere Zeichen einzutasten. Solange dabei der Puffer nicht überläuft, gehen Ihre Eingaben nicht verloren, sondern werden später übernommen.

Der BASIC-Interpreter überprüft nun, ob es sich bei der durch Drücken von RETURN abgeschlossenen Eingabe um eine durch eine führende Programmzeilennummer erkenntliche Programmzeile oder aber um einen oder mehrere durch Doppelpunkte abgetrennte Direktbefehle handelt. Unabhängig davon, ob nun eine Programmzeile oder ein Befehl im Direktmodus vorliegt, werden alle erkannten Befehlswörter in Token umgesetzt (siehe auch Token-Tabelle unter dem Stichwort POKE).

Sowohl die Zeichenleseroutine als auch die Token-Tabelle mit der entsprechenden Tabelle für die Zeichenkette mit der Befehlslangform bieten im Commodore-BASIC gute Einbindungsmöglichkeiten zur Erweiterung des BASIC-Befehlsvorrates.

Durch Abfangen bestimmter Sonderzeichen, die selbstimplementierten Befehlswörtern vorangestellt sind, durch Erweiterung der Token-Tabelle und der Befehlsnamentabelle und Verzweigung zu zugehörigen Interpretations- und Ausführungsroutinen in Maschinensprache kann BASIC relativ einfach erweitert werden. Das zeigen auch die zahlreichen kommerziellen Erweiterungen.

Wenn der Direktmodus vorliegt, analysiert der Interpreter die einzelnen Befehle und führt sie im Fall der Fehlerfreiheit sofort aus.

Findet der Interpreter jedoch als erstes eine gültige Zeilennummer, wandelt er diese zunächst in eine interne, 16 Binärstellen umfassende Darstellung um und fügt sie dann an der richtigen Stelle in die Abfolge der bereits im Speicher befindlichen Programmzeilen ein.

Die 16stellige Binärverschlüsselung der Zeilennummer läßt insgesamt nur 64000 mögliche Programmzeilen mit den Nummern von 0 bis 63999 zu.

Die automatisch richtige Einordnung der Programmzeile entsprechend ihrer Zeilennummer erlaubt Ihnen, völlig unabhängig vom gerade eingegebenen oder angezeigten Kontext neue Zeilen einzugeben. Falls die Zeilennummer dieser neu eingegebenen Zeile allerdings schon existiert, wird die alte Zeile mit dieser Nummer überschrieben. Das kann praktisch sein, ist aber auch gefährlich. Das Einfügen bringt natürlich auch ein Verschieben von Teilen des schon im Speicher befindlichen Programms mit sich.

Der BASIC-Interpreter überprüft Programmzeilen bei der Eingabe nicht auf Fehler im Hinblick auf die Regeln für die Formulierung von BASIC-Programmzeilen. Diese erkennt er erst, wenn er das Programm ausführen soll.

Er meldet aber, wenn er eine Programmzeile nicht in das Programm einfügen kann, weil die eingegebene Zeile mehr Zeichen enthält, als erlaubt ist (maximal 254), oder eine ungültige Zeilennummer besitzt (kleiner als 0 oder größer als 63999).

### Zusammenspiel von Betriebssystem und BASIC-Interpreter bei der Programmausführung

Wenn Sie nun das Programm starten wollen, können Sie das über verschiedene Befehle in Gang setzen:

| | |
|---|---|
| `RUN` | startet das Programm von Beginn an. |
| `RUN Zeilennummer` | startet das Programm von der Zeilennummer an. |
| `GOTO Zeilennummer` | startet das Programm von der Zeilennummer an. Dabei werden bereits vorhandene Variablenwerte nicht gelöscht. |
| `GOSUB Zeilennummer` | startet das Programm von der Zeilennummer an. Bereits vorhandene Variablenwerte werden nicht gelöscht. Als Abschluß der Programmroutine wird RETURN erwartet. |

Findet nun der Interpreter bei der Analyse der Eingaben in seinem Eingabepuffer die Schlüsselwörter RUN, GOTO oder GOSUB, so prüft er, ob eventuell noch die Angabe einer Zeilennummer folgt (Beispiel: RUN 10). Er holt sich dann entweder ab dieser Zeilennummer oder bei ihrem Fehlen die verschlüsselten Programmzeilen des im Speicher befindlichen Programms von Anfang an statt weiterer Zeichen aus dem Tastaturpuffer.

Der BASIC-Interpreter legt die Zeilennummer der gerade in den BASIC-Eingabepuffer eingelesenen Programmzeile an einer besonderen Stelle im

Arbeitsspeicher ab. Er prüft außerdem ständig, ob nicht die RUN/STOP-Taste ausgelöst wurde, was einen sofortigen Abbruch der Programmausführung zur Folge hat.

*Hinweis:* Der RESET-Knopf, den es bei einigen Commodore-Modellen gibt, hat eine ähnliche Wirkung. Nur führt das Drücken des RESET-Knopfs im Gegensatz zur RUN/STOP-Taste zu einer Herstellung des Rechnergrundzustands wie unmittelbar nach dem Einschalten. Dabei geht auch das im Arbeitsspeicher befindliche Programm verloren.

Die dabei ausgelöste Initialisierungsroutine ändert alle Speicherstellen, die Betriebssystem und Interpreter benötigen, um Informationen abzulegen, wieder auf ihre Anfangswerte im Grundzustand. So wird auch der Zeiger auf den Anfang und das Ende unseres BASIC-Programms wieder zurückgesetzt, und wir können nicht mehr ohne weiteres darauf zugreifen, obwohl es sich nach wie vor im Speicher befindet.

Solange also weder RUN/STOP noch RESET gedrückt werden, analysiert der Interpreter unser bereits tokenisiertes Programm. Wenn er ein Token, das ein gültiges BASIC-Schlüsselwort repräsentiert, findet, prüft er zuerst, ob es sich um ein primäres Schlüsselwort handelt.

Es gibt außer primären noch sekundäre Schlüsselwörter im BASIC-Wortschatz, die stets nur im Zusammenhang mit bestimmten primären Schlüsselwörtern zulässig sind und außerdem nur hinter diesen als weiteres Wort auftreten dürfen. Dabei kann vom Vorhandensein sekundärer Schlüsselwörter durchaus noch das obligatorische oder fakultative Auftreten weiterer nachgeordneter Schlüsselwörter abhängen, die erst zur Anerkennung als sprachlich richtige Formulierung in BASIC seitens des Interpreters führen.

Betrachten wir einmal das folgende Programmsegment:

```
100 INPUT "GEBEN SIE EINE ZAHL EIN";A
110 IF A>0 THEN BEGIN
120 :              B=SQR(A)
130 :              PRINT "DIE WURZEL VON";A;"IST";B
140 :              GOTO 180
150 :            BEND
160 PRINT "DIE ZAHL MUSS GROESSER ALS 0 SEIN"
170 GOTO 100
180 END
```

Außer in der Zeile 110 steht in jeder Programmzeile nur ein BASIC- Schlüsselwort, nämlich INPUT, SQR, PRINT, GOTO, BEND, PRINT, GOTO und END in dieser Reihenfolge. In Zeile 110 stehen 3 Schlüsselwörter:

```
110 IF A>0 THEN BEGIN
```

nämlich IF, THEN und BEGIN. Nach dem, was wir vorher gesagt haben und was Sie den einzelnen Erläuterungen im alphabetisch gegliederten Befehlsteil dieses Buchs entnehmen können, sehen wir nun, daß IF ein primäres Schlüsselwort ist, das obligatorisch THEN nach sich zieht. IF darf ohne THEN nicht verwendet werden und THEN nicht ohne IF.

Dann folgt nach THEN ein Anweisungsblock, der durch das Schlüsselwort BEGIN eingeleitet wird. BEGIN kann nicht ohne THEN angewendet werden, muß aber nicht unbedingt nach THEN kommen, ist also ein fakultatives tertiäres Schlüsselwort.

Dieser durch BEGIN eingeleitete Anweisungsblock wird zu guter Letzt durch das Schlüsselwort BEND in der Zeile 150 beendet:

```
150 :              BEND
```

BEND muß immer nach BEGIN folgen. Hier haben wir also sogar ein Beispiel für ein obligatorisches quartiäres Schlüsselwort.

*Hinweis:* Die Doppelpunkte in den Zeilen 120 bis 150 sind überflüssig, gewährleisten aber, daß die Einrückungen vom Interpreter nicht wieder rückgängig gemacht werden und Sie die Programmstrukturierung so besser erkennen können.

Zwischen IF und THEN und zwischen BEGIN und BEND können, obwohl sie voneinander abhängig sind und zur Bildung eines insgesamt zulässigen BASIC-Anweisungsblocks alleine schon ausreichen, durchaus noch weitere BASIC-Befehle oder sogar Gruppen von BASIC-Befehlen auftreten. Andernfalls sind BEGIN und BEND ja überflüssig, denn ein nacktes THEN reicht aus, um kundzutun, daß im Falle der Bedingungserfüllung keine besondere Aktion erfolgen soll.

So ist es in unserem Beispiel, das zur bedingten Ausführung von Anweisungen dient, die zwischen BEGIN und BEND aufgeführt werden, zwingend erforderlich, zwischen den Schlüsselwörtern IF und THEN noch die Bedingung anzugeben, bei deren Erfüllung die Anweisungsgruppe nach BEGIN ausgeführt werden soll:

```
IF Bedingung THEN BEGIN Anweisungsgruppe BEND
```

Die Bedingung muß natürlich ebenfalls entsprechend den Regeln von BASIC formuliert sein.

Der Interpreter prüft also, nachdem er beispielsweise das Schlüsselwort IF erkannt hat, ob anschließend eine BASIC-Variable folgt. Diese könnte durch irgendeine beliebige Folge von Zeichen aus dem Alphabet spezifiziert sein. Dabei achtet der Interpreter unter anderem auf folgende Einschränkungen:

- Die Variable darf nicht mit einem Sonderzeichen oder einer Ziffer beginnen (&MEIER ist falsch).

- Der Variablenname darf kein BASIC-Schlüsselwort sein oder als Teil enthalten (FIFFI ist falsch).

Wenn der Variablenname für gültig befunden wurde, wird der Ausdruck weiter überprüft. Es darf sich an diese Variable einer der in BASIC erlaubten Vergleichsoperatoren wie das Gleichheitszeichen (=), Größer- (>) oder Kleinerzeichen (<) anschließen.

Auch logische Operatoren (AND, OR, XOR, NOT) können in einem Ausdruck vorkommen. Nach dem Operator kann wieder eine Variable oder ein Ausdruck folgen. Arithmetische Operatoren (+, -, *, /) können diese Variablen verbinden.

Ebenfalls dürfen in BASIC eingebaute oder durch den Programmierer definierte Funktionen eingesetzt werden. Auf der rechten Seite des Vergleichsoperators können außer Variablen auch Zeichenkettenkonstanten (z. B. "AHAH") oder Zahlen (z. B. 4711) auftreten.

Der Interpreter prüft also, ob der Programmierer diesen Ausdruck korrekt formuliert hat. Es ist beispielsweise wie auch in der Mathematik nicht erlaubt, Äpfel und Birnen zusammenzuzählen. Deshalb reagiert der BASIC-Interpreter genau wie ein Mathematiklehrer ungehalten mit einer Fehlermeldung, wenn man etwa versucht, ein Zeichen und eine Zahl zu addieren:

```
PRINT A$+B
```

Hat der Interpreter den Ausdruck nun als richtig akzeptiert, prüft er, ob der Ausdruck wahr ist, oder anders ausgedrückt, ob die Bedingung für die Ausführung der Anweisungsgruppe erfüllt ist. Ist sie nämlich nicht erfüllt, beschäftigt er sich gar nicht erst mit den auf THEN folgenden und zwischen BEGIN und BEND eingeschlossenen Anweisungen, sondern sucht in der Zeile unmittelbar hinter BEND das Schlüsselwort ELSE. ELSE gibt an, welche Anweisungen in dem Fall ausgeführt werden sollen, wenn die Bedingung nicht erfüllt ist.

Findet er im Fall einer unwahren, also nicht erfüllten Bedingung keine ELSE-Anweisungsgruppe, dann holt der Interpreter den nächsten Befehl oder die

nächste Programmzeile aus dem Puffer bzw. aus dem BASIC-Programmspeicherbereich und fährt mit der normalen, Zeile für Zeile voranschreitenden Programmausführung fort.

Die Anweisungen zwischen BEGIN und BEND werden, falls die Bedingung hinter IF erfüllt war, eine nach der anderen ausgeführt. Dabei muß auch geprüft werden, ob hinter einer Anweisung eine weitere innerhalb derselben Programmzeile folgt.

Diese muß dann durch einen Doppelpunkt von der vorhergehenden Anweisung abgetrennt sein, sonst erfolgt eine Fehlermeldung. Wenn in derselben Zeile keine weitere Anweisung mehr folgt, abgesehen vom Schlüsselwort REM als Anzeige für einen Kommentartext, geht der Interpreter zur Bearbeitung der ersten Anweisung in der folgenden Programmzeile über.

Sobald der Interpreter jedoch auf BEND stößt, prüft er, ob als nächstes Schlüsselwort ELSE folgt. Da die Bedingung hinter IF erfüllt war und folgerichtig alle Anweisungen des THEN-Blocks ausgeführt wurden, muß die alternative Anweisungsgruppe hinter ELSE nun ignoriert werden. Die nächstfolgende Programmzeile wird in den BASIC-Eingabepuffer geholt.

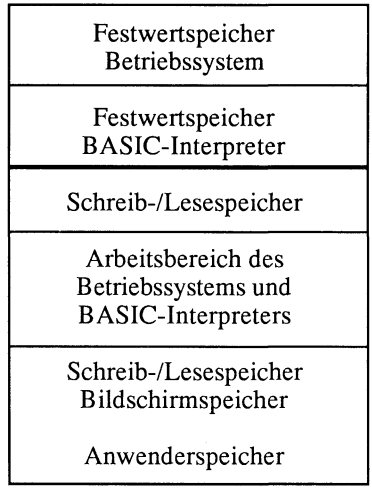
Wie der Doppelpunkt als Trennzeichen zwischen mehreren Anweisungen in einer Zeile haben auch das Komma und das Semikolon ihre eigene Bedeutung in den BASIC-Sprachregeln. Falscher Gebrauch führt wie immer zu einer Fehlermeldung. Ein syntaktischer Fehler (?SYNTAX ERROR) wird auch hervorgerufen, wenn in einem Ausdruck die Anzahl der öffnenden Klammern nicht gleich der Anzahl der schließenden ist oder zu viele oder zu wenige Anführungszeichen vorhanden sind.

In BASIC 7.0 und bei einigen Spracherweiterungen zeigt der Interpreter nach der Eingabe von "HELP" oder nach Drücken der entsprechend belegten Funktionstaste sogar die fehlerhafte Zeile zur Sofortkorrektur an, zumindest gibt er Fehlermeldungen in Klartext, wenn auch in Englisch, aus.

Machen Sie ruhig einmal absichtlich ein paar Fehler, und sehen Sie sich die Fehlermeldungen danach an. Sonst verpassen Sie vielleicht die Chance, aus Fehlern zu lernen.

## Speicherverwaltung und Interpreter

Zu Beginn dieses Kapitels haben wir bereits schon einmal kurz die generelle Aufteilung des Arbeitsspeichers skizziert, so wie er sich uns aus der Sicht des BASIC-Programmierers darstellt.




Diese Darstellung ist zwar, wie Sie beim Vergleich mit Ihrem Rechnerhandbuch feststellen werden, eine Vereinfachung, gibt jedoch ein brauchbares Modell der Verhältnisse für den reinen BASIC-Programmierer ab. Dieser braucht sich nämlich gar nicht um die recht komplexe Speicherarchitektur, die sich aus der mehrfachen Überlagerung von ROM- und RAM-Speicherbereichen und der Einbettung von Registern der Ein- und Ausgabebausteine in den Adreßraum wie beim C128 oder C64 ergibt, zu kümmern.

*Hinweis:* ROM (Read Only Memory) ist der Nur-Lese-Speicher. RAM (Random Access Memory) ist der Speicher mit wahlfreiem (Lesen und Schreiben) Zugriff, hier auch Arbeitsspeicher genannt.

Die mit der Überlagerung verbundenen Vorgänge beim Zugriff auf bestimmte Speicherbereiche werden vom Betriebssystem und vom BASIC-Interpreter gesteuert. Dem Programmierer stehen komfortable Befehle zur Verfügung, falls er für sehr anspruchsvolle Anwendungen besondere Techniken der Speicherverwaltung wie Overlays, Bankswitching oder RAM-Disk realisieren will.

Wird beispielsweise eine BASIC-Spracherweiterung als ROM-Modul in einen Commodore-Rechner eingesetzt, wird in der Regel automatisch der Speicher so neu konfiguriert, daß diese neuen Befehle ohne weiteres angesprochen werden können. Der BASIC-Programmierer muß nur berücksichtigen, daß für seine BASIC-Programme etwas weniger Arbeitsspeicher vorhanden ist.

Für den Programmierer eines C128, der im 128-Modus Software entwickeln will, ergeben sich gegenüber den anderen Commodore-Rechnern vier Änderungen, die jedoch bei reiner BASIC-Programmierung nicht ins Gewicht fallen:

- Der Bildschirmspeicher des 80-Zeichen-Schirms ist nicht Teil des direkt zugreifbaren Arbeitsspeichers.

- Der Anwenderspeicher gliedert sich zwar auch wie bei allen anderen Commodore-Rechnern in die beiden Teile, BASIC-Programmspeicher und Variablenspeicher. Diese sind jedoch im Gegensatz zu den bisherigen Commodore-Rechnern als parallele Speicherbereiche realisiert.

- Weitere parallele Speicherbereiche können bei Aufrüstung des Rechners zusätzlich Daten in binärer Form oder Maschinenprogramme aufnehmen.

- Es gibt eine noch höhere Mehrfachverwendung der ROM-Adreßbereiche für die System-Software und der System-RAM-Arbeitsbereiche.

Beim C128 hat der BASIC-Programmierer keinen direkten Einfluß auf die Verwaltung der ersten beiden RAM-Speicherbänke durch das Betriebssystem und den BASIC-Interpreter. Er hat stets für die BASIC-Programme höchstens 60 KBytes in der ersten Bank, und zwar nur in dieser, und höchstens weitere 63 KBytes in der zweiten zur Verfügung.

Logisch gesehen kann man sich die beiden Bänke auch hintereinanderliegend vorstellen, da man ja nicht über direkte Adressierung auf diese Bereiche zugreift, so daß man gegenüber dem bisherigen Anwenderspeicher-Modell nur den Unterschied bemerkt, daß mehr Anwenderspeicher zur Disposition steht. Dieser ist hier lediglich in einen Daten- und einen Programmbereich aufgeteilt.

Im unteren Teil dieser beiden RAM-Bänke ist jeweils der Arbeitsbereich des Betriebssystems eingeblendet, der also nur einmal vorhanden und allen Bänken gemeinsam ist. Zusätzlich können im C128 noch bis zu 16 KBytes vom Programmierer als weiterer gemeinsamer Bereich für alle Speicherbänke durch entsprechende Parametereinstellung des Speicherverwaltungsbausteins eingerichtet werden.

Der Programmspeicher wird bei der Erweiterung eines im Speicher befindlichen Programms in Richtung aufsteigender Speicheradressen aufgefüllt. Ein Einfügen oder Anfügen oder Erweitern einer Zeile schob also bei den bisherigen Commodore-Rechnern das Programmende und damit auch den Variablenbereich in die Richtung der größeren Adressen.

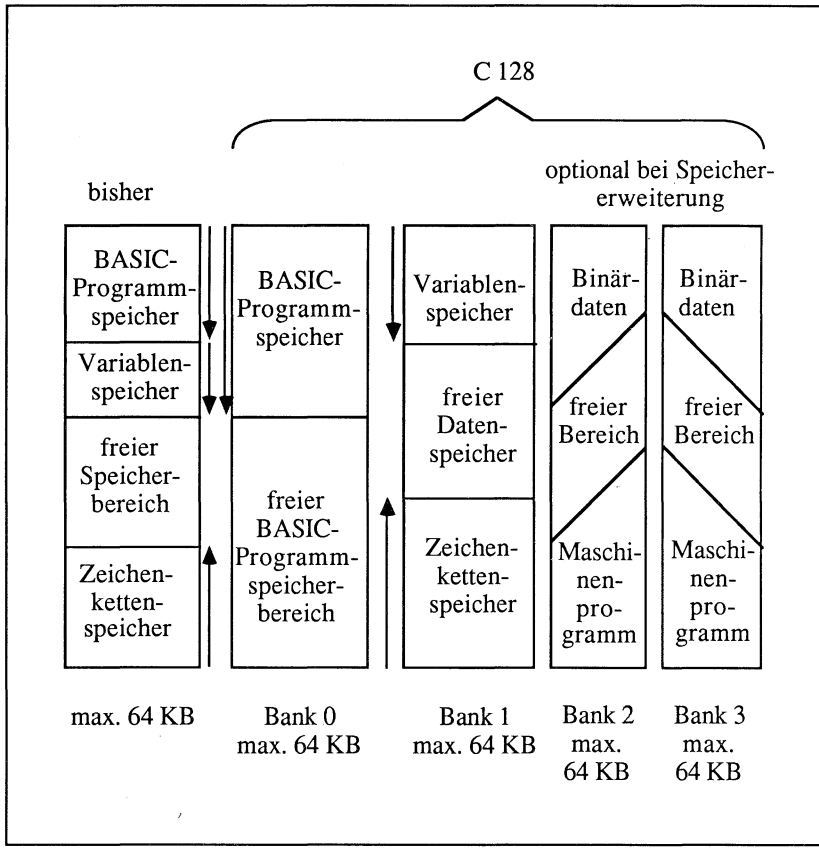



*Abb. 2.1: Die RAM-Bänke des C128*

Beim C128 bleibt jetzt der Variablenbereich von einer solchen Verschiebung unberührt. Dies hat zur Folge, daß der Fehler ?OUT OF MEMORY bei vollem Speicher also nicht mehr allein dadurch gekennzeichnet ist, daß sich die höchste Adresse des nach oben wandernden Variablenbereichs mit der niedrigsten Adresse des nach unten aufgefüllten Zeichenkettenbereichs trifft. Beim C128 kann eine weitere Ursache das Erreichen der 64 KByte-Grenze im Programmspeicher sein.

Die dynamische Verwaltung des Speichers durch das Betriebssystem und den Interpreter ermöglicht Ihnen, relativ freizügig Daten und Programme zu verändern, zu erweitern oder zu löschen, ohne daß Sie wissen müssen, wo sich

diese Objekte Ihrer Aktion nun gerade befinden. Das Betriebssystem versucht die Speicherverwaltung so durchzuführen, daß kein Speicherplatz verschwendet wird und daß sich die verschiedenen Bereiche gegenseitig nicht stören und etwa Daten überschrieben werden.

Wenn Sie beispielsweise mit Zeichenkettenvariablen arbeiten, kann es durchaus vorkommen, daß diesen einmal Zeichenketten maximaler Länge (254 Zeichen) als Wert zugewiesen und anschließend durch Zuweisen von leeren Zeichenketten wieder gelöscht wurden. Eine leere Zeichenkette ist eine Zeichenkette der Länge null, die kein Zeichen enthält:

```
A$=""
```

Frei werdende Zeichenketten-Speicherbereiche kann das Betriebssystem nur für Zeichenketten wiederverwenden, die in solche Lücken hineinpassen. Bereinigt Ihr Betriebssystem nicht von Zeit zu Zeit die so entstandenen Lücken im Zeichenkettenspeicher durch Zusammenschieben dieses Speicherbereichs, kann in kurzer Zeit der verfügbare Platz im Arbeitsspeicher durch eine derartige Verschwendung aufgezehrt werden.

Dieser Vorgang des Speicheraufräumens, der auf amerikanisch treffend als Garbage Collection bezeichnet wird, wird aufgrund verschiedener Kriterien vom Betriebssystem von Zeit zu Zeit ausgelöst und kann bei älteren Commodore-Modellen unter ungünstigen Umständen zu Wartezeiten bis in den Minutenbereich führen.

Sie können diesen Effekt durch einen entsprechend abgestimmten, bewußten Einsatz von Zeichenkettenoperationen beeinflussen, oder Sie können auch aktiv durch den BASIC-Befehl FRE(0) die Garbage Collection gezielt auslösen.

Das Resultat einer Verknüpfung von Zeichenketten zu neuen Zeichenketten speichert das Betriebssystem erst einmal temporär in einem speziellen Arbeitsbereich. Erst durch eine explizite Zuweisung wird das Resultat an eine schon bestehende oder zu verändernde Stelle im dezidierten Speicherbereich für Zeichenkettenvariablen abgelegt.

**Beispiel**

| | |
|---|---|
| `PRINT "COM"+"MODORE"` | nur im Arbeitsbereich |
| `CM$="COM"+"MODORE"` | führt zur Speicherung im Zeichenkettenspeicher |

Wenn Ihre Anwendung es erlaubt, können Sie auch mit der Anweisung CLR im Programm den gesamten Datenbereich freigeben, wobei natürlich alle

Variableninhalte verlorengehen. Für ein gezieltes Löschen genau definierter Gruppen von Programmen bietet das Commodore-BASIC leider keinen Befehl.

Sollten Sie mit einem Ihrer Programme auf Speicherengpässe stoßen, können Sie sich dadurch behelfen, daß Sie alle erhaltenswerten Variableninhalte in Form von Datensätzen kurz vor dem erwarteten Auftreten von Engpässen auf eine Diskette schreiben und CLR eingeben. Alle Variablen, die jetzt noch im Speicher sind, werden gelöscht. Hinterher werden die geretteten Werte wieder von der Diskette eingelesen und den entsprechenden Variablen zugewiesen.

Eine weitaus bessere Lösung als diese Notlösung, wenn Speicherengpässe zu erwarten sind, ist, vom Programmdesign her nur die jeweils benötigten Daten im Speicher zu halten. Alle anderen Daten werden ansonsten prinzipiell nur auf der Diskette geführt, bei Bedarf eingelesen und im Anschluß an die aktuelle Verwendung wieder auf die Diskette zurückgeschrieben.

Neben der optimalen Speicherausnutzung spielt auch der für die Speicherverwaltung erforderliche Anfwand und das resultierende Zeitverhalten bei der Organisation des Anwenderspeichers durch den BASIC-Interpreter eine Rolle. Die Kenntnis der Strategie des BASIC-Interpreters für die Speicherverwaltung kann für Sie nützlich sein, wenn Sie zeitkritische Anwendungen erstellen oder mit der Geschwindigkeit Ihres Programms unzufrieden sind.

Am schnellsten kann der Interpreter auf einfache Variablen zugreifen, da diese in einem festen 7-Byte-Raster abgelegt sind, das nur von den Gleitkommavariablen voll ausgeschöpft wird. Der den Variablen zugewiesene Wert wird im Variablenbereich zusammen mit dem Variablennamen innerhalb dieses Rasters abgespeichert.

Damit die Variableninhalte dort noch Platz haben, ist übrigens die signifikante Länge eines Variablennamens auf zwei Stellen beschränkt. Das heißt, der Interpreter betrachtet immer nur die beiden ersten Stellen eines Variablennamens, der Rest wird ignoriert. Das führt häufig zu Fehlern, da für ihn z. B.

```
TEXT1$
```

und

```
TEXT2$
```

die gleiche Variable ist.

Bei Zeichenkettenvariablen ist allerdings nur der Variablenname, die Länge der Zeichenkette und ein Zeiger auf die zugehörige Zeichenkette enthalten, die

aufgrund ihrer möglichen Länge von bis zu 255 Zeichen nur in variabel großen Feldern in einem separaten Zeichenketten-Speicherbereich abgelegt wird.

Die Speicherorganisation für indizierte Variablen ist wesentlich komplizierter als bei den einfachen Variablen, nutzt dafür aber den Speicherplatz bis auf das letzte Byte aus. Die Verwaltung und der Zugriff auf diese Daten dauert allerdings etwas länger. Da der Interpreter für indizierte Variablen, die nicht explizit mit der DIM-Anweisung dimensioniert wurden, automatisch beim ersten Ansprechen der Variablen dieser den Wert 10 zuweist, sollten Sie, um Speicherplatz zu sparen, besonders auch kleine Feldvariablen dimensionieren:

```
100 DIM A$(30),B(4)
```

Um unnötige Verschiebeoperationen des Variablen- und des Zeichenkettenspeichers während des Programmlaufs zu vermeiden und auch der Übersichtlichkeit wegen, ist es also wünschenswert, allen Variablen zu Beginn des Programms Anfangswerte zuzuweisen und alle Feldvariablen zu dimensionieren. Das sollte nach Möglichkeit in der folgenden Reihenfolge erfolgen, was zu optimalem Zeitverhalten führt:

1. Gleitkommavariablen
2. Integervariablen
3. Zeichenkettenvariablen
4. Gleitkommafelder
5. Integerfelder
6. Zeichenkettenfelder

## Platz für Maschinenprogramme

Der BASIC-Interpreter nimmt bei seiner Speicherverwaltung natürlich keine Rücksicht auf Daten oder Maschinenprogramme, die der Anwender an eine ungeschützte Stelle in den Speicher geschrieben hat.

Für C128-Programmierer besteht die Möglichkeit, Maschinenprogramme sicher in eine der beiden Speicherbänke 2 oder 3 abzulegen.

Besitzer aller Commodore-Systeme können bestimmte geschützte Speicherbereiche, die der Interpreter nicht benutzt und respektiert, für ihre Zwecke umfunktionieren. Am beliebtesten sind hierfür die Kassettenpuffer. Das sind die Speicherbereiche, die Ihr Rechner als Zwischenspeicher in ähnlicher Weise wie den Tastaturpuffer für die Ein- und Ausgabe von Daten auf den Kassetten-

recorder verwendet. Hier können Sie nun kleinere Maschinenprogramme ablegen, natürlich nur, wenn Sie nicht vorhaben, den Kassettenrecorder von Ihrem Programm aus anzusprechen.

Bei größeren Commodore-Rechnern wird auch häufig ein Teil des Bildschirmspeichers, der für die Bildschirmanzeige nicht benötigt wird, dafür verwendet. Dieser Bereich ist ebenfalls vor dem Überschreiben seitens des Interpreters geschützt.

Falls Sie über Honey.Aid oder EXBASIC LEVEL II verfügen, können Sie jedoch auch vom Anwenderspeicher, der sich unter der Kontrolle des BASIC-Interpreters befindet, ein Stück abzweigen und für größere Maschinenprogramme reservieren. Honey.Aid bietet hierfür den Befehl HMEM an (HIMEM bei EXBASIC LEVEL II), der den Zeiger verändert, der auf das obere Ende des Anwenderspeichers zeigt und sich im Arbeitsbereich des Betriebssystems und des Interpreters befindet.

Dieser Adreßzeiger in der sogenannten Zeropage (Seite 0 des Speichers) wird entsprechend der Angabe nach dem HIMEM-Befehl heruntergesetzt. Für den Interpreter ist jetzt der Anwenderspeicher an dieser Stelle zu Ende, und er legt ab dort in Richtung fallender Adressen den Zeichenkettenspeicher an. Auf diese Weise wird im obersten Adreßbereich eine sichere Ablage für Maschinenprogramme oder Datenbereiche, die nicht vom BASIC-Interpreter verwaltet werden sollen, geschaffen.

Beim C64, C16, C116, Plus/4 und C128 können auch die Puffer für die RS232-Ein-/Ausgabe, falls diese nicht benötigt werden, oder nicht benutzte RAM-Adreßbereiche, die wegen eventueller Überlagerung durch ROM-Modulbereiche normalerweise frei bleiben, verwendet werden.

## Der BASIC-Interpreter und die Ein-/Ausgabesteuerung

Neben der schon besprochenen Eingabe von Programmen über die Tastatur und der schon erwähnten Anzeige von Zeichen muß der Interpreter dem BASIC-Programmierer für einen sinnvollen Einsatz des Rechners natürlich noch weitere Ein-/Ausgabefunktionen bieten. Für einen professionellen Gebrauch eines Rechners ist die Ein- und Ausgabe von Daten und Programmen auf periphere Speichermedien wie Kassette oder Diskette unerläßlich. Nur so können große Datenvolumina und umfangreiche Programmsequenzen sicher gehandhabt werden.

| | Richtung | VC20 | C64 | 3000 | 4000 | 8032 |
|---|---|---|---|---|---|---|
| **Anschluß Standardperipherie** | | | | | | |
| 1 externer Busanschluß | B | serieller Bus | serieller Bus oder als RS232 verwendbar | IEC-Bus | IEC-Bus | IEC-Bus |
| 1-2 Kassetten-Ausgänge | B | 1 | 1 | 1 | 2 | 2 |
| 0-2 Steuerausgänge (1-2 Steuerknüppel oder 1-4 Drehregler) | E | 1 | 2 | 0 | 0 | 0 |
| **Anschluß Spezialperipherie** | | | | | | |
| 0-1 8-bit-parallel-Ausgänge (Userport) | B | 1 | 1 | 1 | 1 | 1 |
| 1 interner Rechner-Busanschluß | B | 1 | 1 | 1 | 1 | 1 |
| **Anschluß Bildschirm und Tastatur** | | | | | | |
| 1 Tastaturanschluß | E | i | i | i | i | i/e |
| 1-2 Bildschirmanschlüsse | A | e | e | i | i | i |

*Abb. 2.2 (Teil 1): Die Anschlüsse der Commodore-Rechner*
*A: Ausgabe E: Eingabe B: beides i: intern e: extern*

Für Profis ist ferner die Auflistung von Programmen und die Präsentation von Daten in jedweder Form auf Papier ein unbedingtes Muß, was bedeutet, daß ein Drucker angeschlossen werden soll.

Technische Anwendungen dieser Art erfordern eine gezielte Abfrage oder Ausgabe von Signalen, die an einzelnen Leitungen anliegen (Userport).

Auch für spezielle Peripheriegeräte wie Steuerknüppel (Joystick), Lichtstift (Lightpen), Drehregler (Paddles), Zeichenbretter zur Digitalisierung (Grafik-

| | Richtung | 600/700 | C128/+4 | C16/116 |
|---|---|---|---|---|
| **Anschluß Standardperipherie** | | | | |
| 1 externer Busanschluß | B | IEC- | Serieller Bus oder als RS232 verwendbar | |
| 1-2 Kassetten-Ausgänge | B | 1 | 1 | 1 |
| 0-2 Steuerausgänge { 1-2 Steuerknüppel oder 1-4 Drehregler | E | 0 | 2 | 2 |
| **Anschluß Spezialperipherie** | | | | |
| 0-1 8-bit-parallel-Ausgänge (Userport) | B | 1 | 1 | 0 |
| 1 interner Rechner-Busanschluß | B | 1 | 1 | 1 |
| **Anschluß Bildschirm und Tastatur** | | | | |
| 1 Tastaturanschluß | E | i/e | i | i |
| 1-2 Bildschirmanschlüsse | A | i/e | 2*e | e |

*Abb. 2.2 (Teil 2): Die Anschlüsse der Commodore-Rechner*
*A: Ausgabe E: Eingabe B: beides i: intern e: extern*

tablett) sowie spezielle rechnerinterne Prozessoren zur Ton- und Grafikerzeugung muß der BASIC-Interpreter Ein-/Ausgabemöglichkeiten bereitstellen.

Selbst exotische Aufgabenstellungen, wie z. B. Sprachein-/-ausgabe, sind in BASIC programmierbar. Hierfür sind allerdings spezielle BASIC-Erweiterungen notwendig, die aber in der Regel mit der erforderlichen Zusatz- Hardware mitgeliefert werden.

Von der Rechner-Hardware her stehen die in Abb. 2.2 gezeigten Anschlußmöglichkeiten zur Verfügung.

Der wichtigste und vielseitigste Anschluß für Commodore-Rechner ist der externe Bus, sei es der langsamere serielle Bus oder der schnellere parallele

IEC-Bus. Das Betriebssystem erlaubt theoretisch, über diesen Eingang bis zu 10 Geräte gleichzeitig anzuschließen. Dabei sind allerdings die Positionen 0 bis 3 schon für unmittelbar dem Rechner über spezielle andere Verbindungen zugeordnete Ein-/Ausgabegeräte wie Tastatur, Bildschirm und Kassettenport belegt.

Aus praktischen Gründen wird wohl kaum jemand mehr als zwei Printer/ Plotter, zwei Diskettenlaufwerke und eine Harddisk (Festplatte) an den externen Bus anschließen wollen, da nur jeweils zwei Geräte zum gleichen Zeitpunkt auf diesem externen Bus miteinander kommunizieren können. Daher ist die mögliche Anzahl der an diesem Eingang anschließbaren Geräte mehr als ausreichend.

Damit nun gewährleistet ist, daß sich nur jeweils zwei Geräte miteinander "unterhalten", übernimmt der Rechner als wichtigstes am externen Bus angeschlossenes Gerät eine diesbezügliche Kontrollfunktion und regelt, wer mit wem sprechen darf.

BASIC bietet für die Verbindung vom Rechner zu den Peripheriegeräten den OPEN-Befehl, mit dem über Angabe einer sogenannten Kanalnummer eine logische Verbindung zwischen den Geräten hergestellt wird:

```
OPEN 1,1,0,"DATEI1"
```

öffnet eine Kassettendatei zum Lesen.

Zwischen dem Rechner und den Peripheriegeräten können zu einem Zeitpunkt nur maximal 10 logische Kanäle existieren, wobei aber der Rechner durchaus mehrere Kanalverbindungen zu ein und demselben Peripheriegerät unterhalten kann.

Der Rechner als Koordinator achtet ferner darauf, daß nicht zwei Geräte gleichzeitig Daten senden oder das eine vergeblich auf Daten wartet. Er sorgt dafür, daß ein geordneter Dialog zwischen Sender und Empfänger stattfindet.

Für das Senden und Empfangen von Daten von und zu Peripheriegeräten haben wir in BASIC die Befehle PRINT#, INPUT# und GET#. Außerdem gibt es die Möglichkeit, eingebettet in sogenannte Kommando-Strings, über spezielle Befehlssteuerzeichen bestimmte elementare Funktionen der einzelnen Peripheriegeräte anzusprechen.

Dabei ist natürlich nicht jede Befehls- und Gerätekombination sinnvoll. Es macht sicher wenig Sinn, Daten von einem Drucker einlesen zu wollen, es sei denn, er wäre zu einem optischen Lesegerät umgerüstet worden. Andererseits

kann ein Diskettenlaufwerk nichts mit speziellen Drucker-Steuerbefehlen, wie z. B. dem Seitenvorschub, anfangen.

Bei der Kommunikation mit Peripheriegeräten muß der Interpreter in der Regel stets mit zwei Betriebssystemen kooperieren, da zumindest alle über den externen Bus anschließbaren Peripheriegeräte von Commodore über eine eigene Intelligenz und Steuersoftware verfügen. Das können Sie einfach daran erkennen, daß nach der Eingabe bestimmter Kommandos an das Diskettenlaufwerk dieses selbständig den Befehl ausführt, ohne Ihren Rechner zu blockieren. Das gilt in gleicher Weise für den Drucker, sofern er über einen ausreichen großen internen Puffer verfügt.

Um Geschwindigkeitsunterschiede auszugleichen und um eine gewisse Überlappung von Aktivitäten der einzelnen Geräte zu ermöglichen, werden die Daten auf beiden Seiten jeweils in einem Pufferspeicher zwischengespeichert, der meist mindestens 80 Zeichen umfaßt.

Neben der reinen Transportfunktion bereitet der Interpreter die Daten für das empfangende Gerät noch geeignet auf. So werden z. B. Programme für das Abspeichern auf Kassettenrecorder in eine spezielle Aufzeichnungsform gebracht, die ein besseres Erkennen von Informationsverfälschungen durch Übertragungsfehler oder fehlerhafte Speichermedien erlaubt.

Ferner erkennt die System-Software fehlerhafte Betriebszustände von Peripheriegeräten und meldet diese dem Benutzer oder ermöglicht eine gezielte Abfrage derartiger Statusmeldungen über reservierte Variablen wie DS$, DS, ST.

Für die Abfrage von Impulsen, die von Eingabegeräten wie Steuerknüppel, Drehregler usw. kommen, stellen BASIC 3.5 und 7.0 spezielle Befehle bereit.

Die Ansteuerung oder Abfrage des Userports und des Systembusses ist über die Befehle POKE, PEEK und WAIT möglich, wobei hierfür aber oft Maschinenprogramme eingesetzt werden, die über die Befehle SYS oder USR aktiviert werden können.

## Der Bildschirm-Editor – Teil des Interpreters

Unmittelbar nach dem Einschalten meldet sich Ihr Commodore-Rechner mit der Anzeige Ready. bzw. READY. Dieses Bereitschaftszeichen, auch Prompt genannt, bedeutet, daß Ihr Rechner zur Entgegennahme von Systemkommandos und BASIC-Befehlen im Direktmodus bereit ist.

Sie können prinzipiell alle reinen BASIC-Befehle direkt am Bildschirm eingeben und ihre sofortige Ausführung durch die Betätigung der RETURN-Taste veranlassen. So können sogar kleinere Programme entwickelt werden, die von der Größe her noch bequem auf den Bildschirm passen. Bei der Erstellung dieser Programme tritt der im folgenden besprochene Bildschirm-Editor in Aktion. Ein Abspeichern ist jedoch in dieser Form nicht möglich, so daß direkte BASIC-Programme hier nur für Ad-Hoc-Aufgaben eingesetzt werden.

Direkte BASIC-Befehle sind eigentlich nur wie Taschenrechner-Funktionen zu verwenden, können aber zur Verbindung und Steuerung von Gruppen von Systemkommandos dienen.

Es ist natürlich möglich, solche Programme in Direktmodus-Form durch nachträgliches Einfügen von Zeilennummern in speicherfähige Programme umzuwandeln.

## Der Bildschirm-Editor

Das Prompt-Zeichen des Rechners zeigt jedoch nicht nur seine Bereitschaft an, unsere Anweisungen auszuführen. Immer dann, wenn Sie das blinkende Feld, den Cursor, der uns die jeweilige Schreibposition für mögliche Eingaben anzeigt, auf dem Bildschirm sehen, steht uns der Bildschirm-Editor zur Verfügung.

In dem Zusammenhang wollen wir zunächst den Begriff Editieren erläutern. Wir editieren ein Programm, wenn wir es verändern. Man kann Zeilen oder Zeichen einfügen oder löschen, duplizieren, auseinanderschieben usw., bis ein Programm die Form hat, die wir uns wünschen.

Der Begriff Editieren bezieht sich aber nicht nur auf die Programmgestaltung. Man kann auch einen durch ein Programm erzeugten Output auf dem Bildschirm so modifizieren, wie man ihn beispielsweise für einen Hardcopy-Ausdruck haben möchte.

Bei diesen ganzen Arbeiten unterstützt uns der bei Commodore besonders komfortable ROM-resistente Bildschirm-Editor. Seine Möglichkeiten und Funktionen wollen wir im folgenden betrachten.

## Prinzipielle Arbeitsweise des Editors

Eins der einfachsten Hilfsmittel für das Editieren ist der Cursor. Mit Hilfe der Cursor-Steuertasten kann er zeilenweise nach oben oder unten bzw. spaltenweise nach links oder rechts sowie in die Ausgangsstellung - obere linke Ecke -

des Bildschirms (mit <HOME>) bewegt werden. Zweimaliges Drücken von HOME bewirkt bei allen Commodore-Rechnern mit Fensterunterstützung das Zurücksetzen eines gegebenenfalls definierten Bildschirmfensters.

Als Bildschirmfenster wird ein Teilbereich der insgesamt für Datenein- und -ausgaben zur Verfügung stehenden Anzeigeoberfläche Ihres Bildschirmgerätes bezeichnet. Dabei werden alle Ein- und Ausgaben nach Aktivieren eines Bildschirmfensters auf diesen Bereich beschränkt. Alle anderen auf der restliche Fläche des physischen Bildschirms stehenden Zeichen und Grafiken bleiben erhalten und werden, solange das eingerichtete Fenster aktiv bleibt, nicht verändert.

Das gleichzeitige Drücken der Tasten <SHIFT> und <RETURN> setzt den Cursor an den Beginn der folgenden Zeile nach der augenblicklichen Cursorposition. Beim C128 und den großen Commodore-Rechnern kann er zusätzlich über eine Tabulatortaste analog einer Schreibmaschine positioniert werden. Diese voreingestellten Tabulatorstops befinden sich in jeder achten Spalte und können auch beim C128 durch gleichzeitiges Drücken der Tasten <SHIFT> und <TAB> an der jeweiligen Cursorposition um weitere ergänzt bzw. wieder gelöscht werden.

Die Bildschirmorientierung des Editors erlaubt es, den Cursor frei auf dem Bildschirm zu bewegen und an beliebiger Stelle Zeichen einzufügen, zu löschen oder zu ändern. Diese Änderung wird jedoch nur durch eine anschließende Betätigung der RETURN-Taste wirksam, d. h. in das momentan bearbeitete BASIC-Programm übernommen. Dadurch wird aber immer nur diejenige Programmzeile übernommen, in der sich der Cursor gerade befindet.

Insofern ist der Editor kein echter Fullscreen-Editor, der etwa durch einmaliges Drücken von RETURN eine ganze Bildschirmseite übernehmen würde.

An dieser Stelle muß darauf hingewiesen werden, daß der Bildschirm-Editor für die meisten Commodore-Systeme Programmzeilen von 79 bis maximal 160 Zeichen Länge zuläßt, der BASIC-Interpreter erlaubt jedoch Zeilenlängen bis 255 Zeichen. Als einziges System erlaubt es der C128, bei der Eingabe diese Zeilenlänge mit 160 Zeichen langen Eingaben voll auszuschöpfen. Da alle Befehle mit zwei oder drei Buchstaben abgekürzt werden können, ist es auch möglich, wesentlich längere Zeilen zu erzeugen. Jedoch gibt es bei Änderungen in diesen Zeilen dann Probleme, da der Editor nur 79 (bzw. 160) Zeichen pro Zeile in das editierte Programm übernimmt und den Rest ignoriert oder mit einer Fehlermeldung abweist.

Die Anzahl der physisch auf dem Bildschirm dargestellten Zeichen pro Zeile ist je nach System verschieden und reicht von 23 Zeichen für den VC20 bis zu 80 Zeichen für den C128 und die Geräte der 8000- und 700-Serie. Das führt

dazu, daß sich eine Programmzeile über mehrere physische Bildschirmzeilen erstrecken kann. Da der Bildschirm bei den Systemen mit BASIC-Versionen ab 3.5 logisch in kleinere Teilbildschirme aufgegliedert werden kann (WINDOWS), kann sich im Extremfall eine Programmzeile über mehr als 25 physikalische Bildschirmzeilen erstrecken.

Angenommen, Sie haben folgende Programmzeile auf einem normal breiten Bildschirm eingegeben:

```
10000 PRINT"BITTE EINE ZAHL EINGEBEN"
```

Nun definieren Sie ein Fenster, das 2 Spalten breit ist. Ihre Programmzeile sieht nun so aus:

```
10
00
0
PR
IN
T"
BI
TT
E
EI
NE
 Z
AH
L
EI
NG
EB
EN
"
```

In der Praxis ist jedoch leicht zu unterscheiden, ob eine Bildschirmzeile den Beginn einer neuen Programmzeile enthält oder nur die Fortsetzung der vorigen Zeile. Jede neue Programmzeile muß nämlich stets durch eine gültige BASIC-Zeilennummer (0 bis 63999) eingeleitet werden, und zwischen dem Beginn der Bildschirmzeile und dieser Zeilennummer setzt der Editor genau ein Leerzeichen ein, unabhängig davon, wie viele Leerzeichen Sie freigelassen haben. Bei einigen Commodore-Systemen, z. B. bei Rechnern der 8000-Serie, fügt der Editor nach der Eingabe außerdem noch ein Leerzeichen zwischen linkem Bildschirmrand und der Zeilennummer ein. Dies sollten Sie bei der Eingabe sehr langer Programmzeilen oder auch bei der optischen Gestaltung von Programmen bedenken.

*Anmerkung:* Wenn eine Programmzeile länger ist als die Bildschirmbreite bzw. Fensterbreite, sie sich also über 2 oder mehr Zeilen erstreckt, wird dieses Prinzip auch Wrap around genannt. Bei diesem Wrap around können zwei Situationen auftreten, bei denen Programmfehler entstehen können, die man nur sehr schwer finden kann.

Angenommen Sie haben eine solche Programmzeile, die sich über 2 Zeilen erstreckt, in Ihrem Programm:

```
10000 IF A<ANZAHL AND B>SUMME THEN I=1:C
IRCLE 1,160,100,100,,S(I),S(I+1)
10010 PRINT ...
10020 .
10030 .
```

Im Laufe der Programmentwicklung beschließen Sie nun, den CIRCLE- Befehl in Zeile 10000 zu löschen, und fahren entsprechend mit der Leertaste über den Befehl, drücken aber hinterher nicht RETURN. Das Listing sieht dann wie folgt aus:

```
10000 IF A<ANZAHL AND B>SUMME THEN I=1

10010 PRINT ...
10020 .
10030 .
```

Da Sie die freie Zeile nun gut gebrauchen können, um noch die Zeile 10005 einzufügen, sieht das Ganze dann so aus:

```
10000 IF A<ANZAHL AND B>SUMME THEN I=1
10005 IF B<SUMME THEN I=2
10010 PRINT ...
10020 .
10030 .
```

Optisch stellt sich das Programm nun so dar, wie Sie es haben wollten, aber in Wirklichkeit gibt es die Zeile 10005 gar nicht. Die Zahl 10005 ist in dem Fall keine Zeilennummer, sondern Bestandteil der Zeile 10000, da diese sich immer noch über zwei Zeilen erstreckt. Fatal an diesem Fehler ist, daß das Programm, auch wenn Sie es erneut listen, optisch einwandfrei aussieht. Feststellen können Sie den Irrtum nur, wenn Sie in der Zeile 10000 mit der Taste INST/DEL etwas einfügen oder löschen. Dann bemerken Sie, daß sich auch die vermeintliche Zeile 10005 mit verschiebt.

Es gibt aber noch eine zweite Möglichkeit, mit dem Zeilenumlauf einen Fehler in Ihr Programm einzubauen. Wir betrachten das folgende Programmstück:

```
20000 AN=0:SUMME=SUMME+(COS(I)*3.1415)

20010 PRINT ...
```

Bei der Programmentwicklung stellen Sie fest, daß Sie die Summenbildung in Zeile 20000 unabhängig von der Wertzuweisung für AN anspringen wollen, und Sie beschließen, die SUMME in Zeile 20005 zu berechnen. Weil Sie aber die Formel nicht noch einmal eingeben wollen, fahren Sie mit dem Cursor auf das S der ersten Variable SUMME und schieben den Rest der Zeile so weit hinaus, daß er in die nächste Zeile rutscht:

```
20000 AN=0:
      SUMME=SUMME+(COS(I)*3.1415)
20010 PRINT ...
```

Nun fehlt nur noch die Zeilennummer. Die setzen Sie davor:

```
20000 AN=0:
20005 SUMME=SUMME+(COS(I)*3.1415)
20010 PRINT ...
```

Hier nützt es Ihnen auch nichts, wenn Sie Ihre Eingaben mit RETURN abschließen - die Zeile 20005 existiert nicht. Das werden Sie spätestens dann feststellen, wenn Sie die Zeile 20005 anspringen wollen.

Wie Sie es richtig machen, wenn Sie eine Zeile oder einen komplizierten Ausdruck nicht noch einmal neu tippen wollen, erfahren Sie im weiteren Verlauf dieses Kapitels.

### Globales Editieren

Die Zeilennummern werden vom Editor und auch von Programmierhilfen wie POWER, Toolkit, SM-Kit oder auch von entsprechenden Befehlen in Spracherweiterungen wie Honey.Aid zum großmaßstäblichen Editieren herangezogen.

Durch die Eingabe einer Zeilennummer, gefolgt von mindestens einem Zeichen außer dem Leerzeichen, und anschließendes Betätigen der RETURN-Taste wird eine neue BASIC-Zeile angelegt oder, falls unter dieser Nummer bereits eine Zeile existiert, wird diese mit der neuen Eingabe überschrieben.

Diese Eigenschaft des Editors kann sehr geschickt dazu ausgenutzt werden, um in bestehenden Programmen größere Änderungen vorzunehmen.

Mit dem Befehl

```
LIST Zeilennummer1 - Zeilennummer2 <RETURN>
```

wird die Anzeige eines Programmteils auf dem Bildschirm veranlaßt. Um eine Programmzeile zu löschen, brauchen wir nur in einer leeren Bildschirmzeile ihre Zeilennummer einzugeben und die RETURN-Taste zu betätigen. Ein erneutes Auflisten des Programmstücks läßt erkennen, daß die betreffende Programmzeile fehlt.

Auf ähnliche Weise lassen sich Programmzeilen kopieren. Der Cursor wird auf die Zeilennummer der zu kopierenden Zeile positioniert. Dann wird diese Zeilennummer mit der für die Kopie gewünschten Nummer überschrieben und die RETURN-Taste ausgelöst. Der Befehl LIST mit Angabe der neuen Zeilennummer zeigt, daß die Zeile erfolgreich kopiert wurde. Zur zusätzlichen Kontrolle kann mit LIST und Angabe der alten Zeilennummer überprüft werden, daß die Originalzeile weiterhin existiert.

Eine Kombination von Kopieren einer Programmzeile mit anschließendem Löschen der Ursprungzeile kann dazu verwendet werden, um Zeilen von einem Teil des Programms in einen andern zu verschieben.

Diese Vorgehensweise ist bei einer größeren Anzahl von Programmzeilen allerdings ziemlich mühsam. BASIC 3.5 und 7.0 stellen hierfür den RENUMBER-Befehl bereit, der aber auch für die anderen BASIC-Versionen in Form von Toolkits (Programmierhilfen) oder als Bestandteil von BASIC-Erweiterungen verfügbar ist.

Mit RENUMBER in der Commodore-Version können allerdings nicht ohne weiteres Programmteile innerhalb des Programms verschoben werden, weil zwar eine Startadresse für die erste umzunumerierende Zeile aber keine Endadresse für den RENUMBER-Prozeß spezifiziert werden kann, wie dies bei manchen RENUMBER-Befehlen vom Fremdherstellern möglich ist. Die RENUMBER-Version von Commodore ist also primär zur vollständigen Neunumerierung aller Zeilen eines Programms gedacht.

Es gibt jedoch eine elegante Ersatzlösung. Bei vernünftiger Programmierung sollte der zu verschiebende Bereich meist kleiner als 20 Bildschirmzeilen sein. Der in BASIC 3.5 und 7.0 und in fast allen Spracherweiterungen vorhandene Befehl

```
DELETE Zeilennummer1 - Zeilennummer2
```

ermöglicht das einfache Löschen von ganzen Gruppen von Programmzeilen.

Zunächst bringen wir aber das zu verschiebende Programmstück mit dem Befehl

```
LIST Zeilennummer1 - Zeilennummer2 <RETURN>
```

auf den Schirm. Dann geben wir

```
DELETE Zeilennummer1 - Zeilennummer2 <RETURN>
```

unmittelbar hinter der letzten Programmzeile ein. Nun bewegen wir den Cursor wieder hinter die letzte Programmzeile und überschreiben den DELETE-Befehl mit

```
RENUMBER,20
```

und betätigen die RETURN-Taste. Nun haben wir unser Programm in Zwanzigerschritten organisiert, was zwischen zwei Programmzeilen Platz läßt für jeweils 19 zusätzliche Zeilen.

Die gelöschten Zeilen, die wir ja verschieben wollten, müssen wir nun wieder in das Programm aufnehmen. Sie stehen ja noch auf dem Bildschirm und müssen nur noch mit den Zeilennummern versehen werden, die in den neuen Bereich hineinpassen. Vergessen Sie nicht, jede Eingabe durch <RETURN> zu quittieren.

Abschließend können Sie das ganze Programm mit der einfachen Angabe

```
RENUMBER
```

wieder in Zehnerschritten, was am praktischsten ist, durchnumerieren.

Damit ist die Verschiebeoperation beendet, wie Sie mit erneutem Auflisten des Programms leicht überprüfen können.

An dieser Stelle muß allerdings die dringende Empfehlung ausgesprochen werden, vor solch umfangreichen Änderungen prinzipiell immer eine temporäre Sicherungskopie der aktuellen Programmversion auf Band oder Diskette zurückschreiben. Wählen Sie daher Ihre Programmnamen immer so, daß Sie hinten noch Platz für das Anfügen einer mehrstelligen Versionsnummer haben. Genauso wichtig ist es, hin und wieder diese temporären Sicherungskopien zu löschen, wenn sie nicht mehr benötigt werden, bevor Sie den Überblick verlieren.

Der RENUMBER-Befehl kann auch eine Hilfe sein, wenn man beim Programmieren feststellt, daß irgendwo noch Zeilen dazwischengeschoben werden

müssen, der Platz aber nicht mehr ausreicht. Bei sorgfältiger Planung und vorherigem Entwurf des Programms auf Papier sowie einer vernünftigen Strukturierung der Zeilennummern mit ausreichender Schrittweite sollte die Notwendigkeit, RENUMBER für solche Fälle einzusetzen, beschränkt bleiben.

Ein sinnvoller Einsatz für RENUMBER ist aber sicher bei der Übernahme von nützlichen Routinen aus verschiedensten Quellen gegeben, die in den seltensten Fällen numerierungsmäßig in die eigenen Programme passen.

Hierbei ist man natürlich nicht auf Programmteile in der Größenordnung von Bildschirmseiten beschränkt.

Für die Einbindung solcher Programmstücke benötigen Sie allerdings dann ein spezielles Hilfsprogramm, oder Sie verfügen, was noch besser ist, über eine Spracherweiterung, die den MERGE-Befehl oder wie Honey.Aid den APPEND-Befehl enthält.

Obwohl es für den C128 zur Zeit solche Erweiterungen noch nicht gibt, besteht auch für C128-Besitzer die Möglichkeit, kommerzielle Programmierhilfen und Spracherweiterungen, die für den C64-Modus erhältlich sind, einzusetzen (siehe auch Kapitel 11). Diese Programme, die im 64-Modus mit MERGE zusammengebunden und dann wieder abgespeichert wurden, sind selbstverständlich im 128-Modus wieder lauffähig. Hierbei kann es allerdings bei bestimmten Realisierungsformen der MERGE-Funktion insofern Einschränkungen geben, als daß nur dem C64 bekannte Befehle in den Programmen enthalten sein dürfen.

Auch der DELETE-Befehl findet sein Haupteinsatzgebiet bei der Übernahme von eigenen oder fremden Programmen aus ähnlichen Aufgabenstellungen für neue Programmierprojekte. Daher sollten Routinen für formatierte Ausgabe, Datenverwaltung, Kalenderrechnung, Datenerfassung und Menüsteuerung so geschrieben werden, daß sie wiederverwendbar sind.

Für die Eingabe einer größeren Anzahl neuer Programmzeilen stellen BASIC 3.5 und 7.0 den Befehl AUTO zur Verfügung, der automatisch eine neue Zeilennummer nach dem Abschluß einer Programmzeileneingabe mit <RETURN> für die folgende neue Programmzeile vorgibt. Dabei ist die Schrittweite frei wählbar:

```
AUTO 100
```

schaltet beispielsweise eine automatische Zeilennummernvorgabe mit der Schrittweite 100 ein.

Aber auch ohne den AUTO-Befehl können sich die Benutzer von Commodore-Rechnern das Eintippen von Zeilennummern ersparen. Geben Sie bitte folgende BASIC-Zeile ohne zusätzliche Leerzeichen in Ihren Rechner ein:

```
0 Z=Z+100*0↑S:S=10:FORZN=ZTOZ+S*19STEPS:
?ZN:NEXT:?"Z=ZN    :S=S :GOTO 0":END
```

Wenn Sie nun RUN eintippen, werden 20 Zeilennummern vorgegeben, die Sie für die Eingabe Ihrer ersten Programmzeilen benutzen können. Sie beginnen mit 100 und haben eine Schrittweite von 10.

Nachdem Sie die 20 Programmzeilen eingegeben haben, die Sie jedesmal mit RETURN abschließen, steht der Cursor auf der Bildschirmzeile, die

```
Z=ZN    :S=S :GOTO 0
```

enthält.

Drücken Sie nun einfach die RETURN-Taste, und es werden die nächsten 20 Zeilennummern vorgegeben.

Dieses Programm ist natürlich ziemlich unflexibel und nicht besonders komfortabel. Dafür ist es auch nur ein Einzeiler.

Das Ganze kann man natürlich noch sehr viel aufwendiger realisieren. Dabei bleibt das Prinzip das gleiche: Das Programm arbeitet nur im Bildschirm-Editor. Professioneller sieht es dann also so aus:

```
0  REM**********************************
   *   BASIC-EDITOR ALS MINITEXTSYSTEM *
1  REM**********************************
12 IF ABS(ZE+SW-ZN)>0 THEN GOSUB 90
                         :GOTO 15
13 REM                 ELSE
14 :                        ZE=ZN
15 IF TE<ZA            THEN TE=ZE
16 IF BE$="+"          THEN 40
17 IF BE$="A" OR ZA=ZI OR ZN-ZA<100
                       THEN ZE=ZI
18 IF BE$="-" AND ZE>ZI+D2*SW AND ZA>ZI
              OR ZN<0 THEN ZE=ZE-D4*SW
19 IF BE$="E"          THEN ZE=TE-DP*SW
20 IF BE$="P"          THEN STOP
40 FK=0↑SW: IF FK      THEN GOSUB 90
45 ZA=ABS(ZE+ZI*FK):ZE=ZA+DP*SW
```

```
50 PRINT TP$=RV$+"LIST"+STR$(ZA)+" -"
                +STR$(ZE)+":"+MS$+D$
60 FOR ZN=ZA TO ZE STEP SW
62 : PRINT STR$(ZN)+HV$
68 NEXT ZN
70 IF BE$="-" THEN A$="-"
71 IF BE$="+" OR BE$="A" OR BE$="E"
              THEN A$="+"
73 PRINT D6$+RV$+"BEFEHL$="+HK$+A$+HK$;
  ":ZN="+ZN$;":SW="+"+SW ";":GOTO0:"+U6$
74 PRINT D$+RV$+"VERSION$="+HK$+STR$(N)+
HK$;":   TEXTNAME$="+HK$+"MTEXT"+HK$+":
75 PRINT"+ / -   VOR-/RUECKWAERTS      A
  ANFANG"
76 PRINT"D DRUCKEN S SPEICHERN L LADEN E
  ENDE "
77 PRINT"VERSIONSNUMMER 3 STELLEN TESTNAME 5"
   +U$:END
90 REM*********************************
   *          INITIALISIERUNG          *
91 REM*********************************
92 DP=06:D2=DP+2:D4=2*DP+2:ZE=ZN-SI:
   SI=10:SW=ABS(SW)+SI*FK
93 ZI=ABS(ZN+100-ZI*(ZN+1)/ABS(ZN+1))
94 RA$=CHR$(146):RV$=CHR$(18):BK$="
                                 "
95 BS$=CHR$(160):TP$=CHR$(147):BE$="A"
   :ZN$="-.+ZN  ":REM "-."=0 DA "."=0
96 MS$="BITTE BEI EINGABE        ZEILEN
  MIT RETURN ABSCHLIESSEN!":IF FK THEN N=1
97 HK$=CHR$(34):HV$=HK$+BK$:A$="+":
   U$=CHR$(145):D$=CHR$(17):L$=CHR$(157)
98 D6$=D$+D$+D$+D$+D$+D$:
   U6$=U$+U$+U$+U$+U$+U$+U$
99 RETURN:STOP
```

Wenn Sie keine Fehler mehr feststellen, können Sie die letzte Version des AUTO-Programms mit dem Befehl

```
SAVE "AUTONUM",8
```

auf einer formatierten Diskette oder mit

```
SAVE "AUTONUM"
```

auf einer leeren Kassette abspeichern.

Bevor Sie nun mit der Eingabe eines neuen Programms beginnen, laden Sie immer zuerst AUTONUM in Ihren Rechner und geben <GOTO 0>, gefolgt von <RETURN>, ein.

Auf dem Bildschirm werden als Vorgabe immer 7 Zeilennummern vorgegeben, die Sie für die Eingabe Ihrer Programme verwenden können. Nach dem Abschluß der Eingabe stellen Sie den Cursor auf die letzte Zeile und drücken RETURN. Dann erhalten Sie weitere 7 Zeilennummern.

Wenn Sie eine andere Startadresse für die Zeilennumerierung ZN oder eine andere Schrittweite SW vorziehen, so ist dies jederzeit möglich. Ändern Sie einfach ZN oder SW auf den gewünschten Wert und drücken RETURN.

Achten Sie dabei aber darauf, daß Sie keine bereits vorhandenen Zeilennummern wählen. Dies gilt auch für Anwender des AUTO-Kommandos.

Wenn Sie den vorgegebenen Zeilennummernbereich nur teilweise ausschöpfen, können Sie neue Zeilennummern am schnellsten durch Betätigen von <HOME> + <RETURN> anfordern. Der Bildschirm wird hierbei nicht gelöscht.

Wenn eine Zeile fehlen sollte, haben Sie sicher vergessen, bei der Eingabe RETURN zu drücken, oder Sie haben nicht immer die Zeilennummer geändert.

Wenn Sie versehentlich zwei Zeilen vertauscht haben, wie im Beispielprogramm die Zeilen 1030 und 1040:

```
1000 FOR I=1 TO 20
1010 : FOR J=1 TO 10
1020 :   PRINT I,J
1030 : NEXT I
1040 NEXT J
```

geben Sie einfach <LIST 1030-1040> ein, ändern 1030 in 1040, drücken RETURN, ändern 1040 in 1030, drücken RETURN, und der Fehler ist behoben.

In diesem speziellen Beispiel kann man natürlich auch in den Zeilen 1030 und 1040 einfach I und J vertauschen.

Sie können sich dies zunutze machen, wenn sehr viele ähnliche Programmzeilen eingegeben werden müssen. Sie brauchen nur die Zeilennummer mit

der nächsten zu überschreiben und gegebenenfalls die Änderungen in der Zeile auszuführen.

Angenommen Sie haben bereits

```
1000 DATA ANTON,WENZEL,BONN,RADWEG 13
```

eingegeben. Wenn Sie jetzt noch ERNA, GOTTFRIED und PETER WENZEL mit derselben Adresse als Daten erfassen wollen, tippen Sie einfach <LIST 1000><RETURN>. Es erscheint:

```
1000 DATA ANTON,WENZEL,BONN,RADWEG 13
```

Sie ändern nun 1000 in 1010 und ANTON in ERNA und drücken RETURN. Dann überschreiben Sie 1010 mit 1020, geben für ERNA GOTTFRIED ein und drücken wieder RETURN usw. Wenn Sie jetzt die Zeilen ab 1000 listen, werden alle Namen angezeigt.

### Spezielle Editor-Befehle

Natürlich gibt es noch eine ganze Reihe anderer Editor-Funktionen als die bis hierhin beschriebenen. Einen großen Teil dieser Funktionen erreicht man über die sogenannten Escape-Sequenzen und weitere des C128 dann über die CTRL- und Commodore-Taste.

Zunächst wollen wir jedoch die Escape-Sequenzen betrachten. Sie werden so genannt, weil man bei jeder dieser Funktionen zunächst die ESC-Taste einmal drücken muß. Danach drückt man den angegebenen Buchstaben, um die gewünschte Funktion einzuschalten. Da die ESC/Buchstaben-Kombinationen nicht bei allen Commodore-Rechnern mit den gleichen Funktionen belegt sind, müssen wir hier wieder differenzieren:

In der Spalte mit der Überschrift Rechner steht die Ziffer 1 für den C128 und die Ziffer 2 für den C116:

| Tasten | Rechner | Funktion |
|---|---|---|
| ESC + A | 1,2 | Automatisch einfügen. |
| ESC + B | 1,2 | (Set Bottom) Fixiert an der gegenwärtigen Cursorposition die rechte, untere Fensterecke. |
| ESC + C | 1,2 | (Clear auto insert) Hebt automatisches Einfügen auf. |

| Tasten | Rechner | Funktion |
|---|---|---|
| ESC + D | 1,2 | (Delete) Löscht die Zeile, in der der Cursor steht. Die Zeile wird dabei jedoch nicht aus dem BASIC-Programm entfernt. Alle Zeilen unterhalb der gelöschten werden um eine Bildschirmzeile hochgeschoben. |
| ESC + E | 1 | Cursor-Blinken wird ausgeschaltet. |
| ESC + F | 1 | Cursor-Blinken wird eingeschaltet. |
| ESC + G | 1 | Läßt mit CTRL-G akustisches Signal ertönen. |
| ESC + H | 1 | Verhindert akustisches Signal mit CTRL-G. |
| ESC + I | 1,2 | (Insert) Fügt eine Zeile über der Cursorposition ein. |
| ESC + J | 1,2 | Cursor wird an den Anfang der Cursor-Positionszeile gesetzt. |
| ESC + K | 1,2 | Cursor wird rechts neben das letzte gedruckte Zeichen der Cursor-Positionszeile gesetzt. |
| ESC + L | 1,2 | Schaltet Bildschirm-Rollen wieder ein. |
| ESC + M | 1,2 | Verhindert Bildschirm-Rollen. |
| ESC + N | 1 | Schaltet den Bildschirm von invers auf normal zurück (im 80-Zeichen-Modus). |
| | 2 | Schaltet zur normalen Bildschirmgröße zurück und löscht den Bildschirm. |
| ESC + O | 1,2 | (Off) Hebt Einfüge-, Anführungszeichen-, Invers- und Blink-Modus wieder auf. |
| ESC + P | 1,2 | Löscht Bildschirmzeile vom Anfang bis zur Cursorposition. Erst durch Eingabe der gekürzten Zeile mit <RETURN> wird diese Änderung in das zu editierende Programm übernommen. |
| ESC + Q | 1,2 | Löscht Bildschirmzeile ab der Cursor-Position bis zum Ende. Erst durch Eingabe der gekürzten Zeile mit <RETURN> wird diese Änderung in das zu editierende Programm übernommen. |
| ESC + R | 1 | Schaltet den Bildschirm von normal auf invers (im 80-Zeichen-Modus). |
| | 2 | Verkleinert die Anzeigefläche auf dem physikalischen Bildschirm, kann zur Anpassung an bestimmte Fernseher oder Monitoren verwendet werden. |
| ESC + S | 1 | Schaltet Cursor in inversen Block um (Größe 8 x 8 Pixels). |

| Tasten | Rechner | Funktion |
|---|---|---|
| ESC + T | 1,2 | (Set Top) Fixiert an der gegenwärtigen Cursorposition die linke, obere Fensterecke. |
| ESC + U | 1 | Schaltet Cursor in Unterstrich um (im 80-Zeichen-Modus). |
| ESC + V | 1,2 | Rollen des Bildschirminhalts um eine Zeile nach oben. |
| ESC + W | 1,2 | Rollen des Bildschirminhalts um eine Zeile nach unten. |
| ESC + X | 1 | Schaltet den Bildschirm vom 40-Zeichen-Modus in den 80-Zeichen-Modus und umgekehrt. |
| | 2 | (Exit ESC) Befreit Sie aus dem Escape-Modus nach versehentlicher Betätigung der <ESC>-Taste. |
| ESC + Y | 1 | Schaltet die voreingestellten Tabulatorstops wieder ein. |
| ESC + Z | 1 | Schaltet die voreingestellten Tabulatorstops aus. |
| ESC + @ | 1 | Löscht den ganzen Bildschirm ab der Cursorposition. |
| HOME | 1,2 | Setzen des Cursors in die linke obere Ecke des Bildschirmfensters. |
| 2 x HOME | 1,2 | Aufheben des Bildschirmfensters und Positonierung des Cursors in die linke obere Ecke des Bildschirms. |
| SHIFT+CLR | 1,2 | Löschen des Bildschirmfenster-Inhalts und Setzen des Cursors in die linke obere Ecke des Fensters. |
| SHIFT+TAB | 1 | Tabulatorstop an Cursorposition setzen oder löschen (analog einer Schreibmaschine). |
| TAB | 1 | Cursor auf nächste Tabulatorposition setzen. |
| SHIFT+C= | alle | Umschalten zwischen Block-/Grafikmodus (Großbuchstaben umd Grafikzeichen) und Textmodus (Groß-/Kleinschreibung). |
| ASCII/DIN | 1 | Umschalten zwischen deutschem und amerikanischem Zeichensatz. |

### Verwendung von Editor-Befehlen in Programmen

Editor-Befehle können in zweierlei Hinsicht in Programmablauf eine Rolle spielen. Zunächst einmal kann die Mehrzahl von ihnen bei der Gestaltung von Ausgaben auf dem Bildschirm sinnvoll eingesetzt werden. Außerdem können Editor-Befehle wie INST/DEL bei Eingaben während des Programmablaufs verwendet werden, falls sie nicht explizit durch das Programm deaktiviert wurden, was auch möglich ist.

Die Verwendung von Editor-Befehlen ist recht einfach. Wir wollen uns als Beispiel die Definition eines Fensters als Eingabefeld ansehen. Dieses Programm enthält keine POKEs oder unleserlichen Grafikzeichen, so daß es leicht nachvollziehbar ist. In BASIC 7.0 könnte man diese Anwendung auch mit Hilfe des WINDOW-Befehls realisieren:

```
100 OPEN 1,0
110 ESC$=CHR$(27)
120 EINGABEFELD$="FALSCH"
130 FENSTER$="ANTWORT? "+ESC$+"M"+ESC$+"
T"+EINGABEFELD$+ESC$+"B"
140 PRINT FENSTER$;CHR$(19):INPUT#1,EINGABE$
150 PRINT ESC$+"L"+CHR$(19)+CHR$(19)+CHR
$(147)+EINGABE$
160 CLOSE 1
```

In Zeile 100 wird der Bildschirm als Datei eröffnet. Das ist deshalb notwendig, damit in Zeile 140 der INPUT#1-Befehl für die Eingabe verwendet werden kann, der im Gegensatz zum INPUT-Befehl kein überflüssiges Fragezeichen erzeugt.

Der ASCII-Code 27 für die ESC-Taste wird der Variablen ESC$ in Zeile 120 zugewiesen. Das Eingabefeld mit dem gleichlautenden Namen wird in Zeile 110 mit dem Wort FALSCH vorbesetzt. An diese Stelle können Sie auch irgendeinen anderen kürzeren oder längeren Text setzen. Dementsprechend wird auch die Größe des Eingabefensters ausfallen.

Zeile 130 speichert in der Variablen FENSTER$ nun zunächst die Ausgabe des Wortes ANTWORT?, ESC + M für das Verhindern des Bildschirmrollens und die Escape-Sequenzen mit T und B für das Fixieren des Eingabefensters in der Länge von EINGABEFELD$.

In Zeile 140 wird nun die Variable FENSTER$ ausgegeben, der Cursor mit CHR$(19) an den Anfang des Fensters gesetzt und eine Eingabe des Benutzers in der Variablen EINGABE$ gespeichert. Diese Eingabe wird in Zeile 150, nachdem das Bildschirmrollen wieder eingeschaltet wurde und mit zweimal

CHR$(19) (HOME) das Fenster aufgehoben und mit CHR$(147) (SHIFT/CLR) der Bildschirm gelöscht wurde, oben links auf dem Bildschirm ausgegeben.

Für die Rechner der 8000-, 500-, 600- und 700-Serien müssen Sie das Programm leicht abwandeln:

```
100 open 1,0
110 eingabefeld$="falsch"
120 fenster$="antwort? "+chr$(15)+eingabefeld$
+chr$(143)
130 print fenster$;chr$(19);:input#1,eingabe$
140 print chr$(19)+chr$(19)+chr$(147)+eingabe$
150 close 1
```

Auf allen anderen Rechnern kann man mit den Befehlen GET und POS() ein Eingabefenster simulieren.

## Fensterbefehle beim C128

Wie oben schon erwähnt, ist die ganze Fenstertechnik beim C128, C16, C116 und Plus/4 wesentlich leichter zu realisieren als bei den anderen Commodore-Rechnern, da uns hier ein entsprechender Befehl dafür zur Verfügung steht:

```
WINDOW SOL,ZOL,SUR,ZUR,0/1
```

Dieser Befehl definiert ein Bildschirmfenster mit der oberen linken Ecke an der Position SOL (Spalte oben links) und ZOL (Zeile oben links) und der unteren rechten Ecke an der Position SUR (Spalte unten rechts) und ZUR (Zeile unten rechts).

Falls als 5. Parameter 1 angegeben wird, wird automatisch das Fenster vorher gelöscht. Dabei muß man die Maximalangaben berücksichtigen, die man beim Arbeiten im 40- bzw. 80-Zeichen-Bildschirm nicht überschreiten darf. Der Zeilenbereich liegt bei beiden zwischen 0 und 24, während der Spaltenbereich einmal die Spalten 0 bis 39 und im 80-Zeichen-Modus die Spalten 0 bis 79 umfaßt.

Alle Ein- und Ausgaben auf dem Bildschirm werden bis zum Aufheben des Bildschirmfensters nur innerhalb dieses Fensters abgewickelt. Zum Beispiel kann es benutzt werden, um die Inhaltsverzeichnisse zweier Disketten zu vergleichen. Zunächst läßt man sich das Inhaltsverzeichnis der ersten Diskette zeigen. Dann definiert man die rechte Bildschirmhälfte als Fenster und ruft darin das Inhaltverzeichnis der zweiten Diskette auf.

Im 40-Zeichen-Modus passen allerdings nicht zwei Verzeichnisse nebeneinander auf den Bildschirm, da bereits ein Inhaltsverzeichnis mit allen seinen Angaben eine Breite von 27 Zeichen einnimmt. Bei der Anzeige des ersten Verzeichnisses spielt das zwar keine Rolle, da hierfür ja noch der gesamte Bildschirm zur Verfügung steht, aber beim zweiten laufen dann die Zeilen, die zu lang sind, um und nehmen pro Programm zwei Zeilen in Anspruch, so daß kein direkter Vergleich der beiden Inhaltsverzeichnisse mehr möglich ist.

Deshalb definieren wir für das erste Verzeichnis die linke Bildschirmhälfte als Fenster, damit die Anzeigen dann gleich aussehen und auf gleicher Ebene verglichen werden können.

Der WINDOW-Befehl, den wir dafür benötigen, lautet:

```
WINDOW 0,0,20,24,1
```

Nun rufen wir mit DIRECTORY das Inhaltsverzeichnis der ersten Diskette auf. Danach definieren wir das zweite Fenster mit

```
WINDOW 20,0,39,24,1
```

und lassen uns hier das Inhaltsverzeichnis der zweiten Diskette zeigen. Die Anzeige im ersten Fenster wird dabei nicht überschrieben, da sie sich nun außerhalb des aktuellen Fensters befindet. Der Bildschirm könnte dann folgendermaßen aussehen:

```
6    "KREIS9"          1     "TORUS"
   PRG                     PRG
7     "KREIS10"        8     "TEST"
   PRG                     PRG
12    "FENSTER9"       11    "FENSTER11"
   PRG                     PRG
16    "FENSTER20"      17    "FENSTER21"
   PRG                     PRG
```

Im 80-Zeichen-Modus braucht man lediglich ein Fenster für die zweite Anzeige zu definieren:

```
WINDOW 40,0,79,24,1
```

Der WINDOW-Befehl kann auch verwendet werden, um durch eine Art Überlagerung Programmzeileninhalte zu mischen oder Bilder und Texte in Programme zu übernehmen.

**Beispiel:**

Definieren Sie zunächst ein Fenster mit dem Befehl

```
WINDOW 10,0,39,24,1
```

Ihr Bildschirm ist nun im Verhältnis 1:3 aufgeteilt, und der Cursor befindet sich in der rechten Hälfte. Dorthinein laden Sie jetzt mit

```
diR <RETURN>
```

das Inhaltsverzeichnis einer Diskette. Drücken Sie nun zweimal hintereinander die HOME-Taste, um das Fenster zu verlassen. Nun können Sie beispielsweise vor die Programmnamen folgendes schreiben, um das Inhaltsverzeichnis in ein Auswahlmenü zu übernehmen:

```
1 PRINT"1 ...
2 PRINT"2 ...
3 PRINT"3 ...
```

Dabei stehen für die Pünktchen bereits die richtigen Programmnamen von der Diskette da. Die Typbezeichnungen der Programme kann jeweils mit ESC + Q am Ende der Zeile gelöscht werden.

*Hinweis:* Um Fenster deutlicher voneinander abzusetzen, sollten Sie mit <CONTROL> + Farbtaste für jedes Fenster eine eigene Cursor- und somit Schriftfarbe wählen.

Mit der Anweisung RWINDOW können wir die Parameter unseres Fensters erfragen, und zwar:

```
PRINT RWINDOW(0) - Anzahl der Zeilen im Fenster

PRINT RWINDOW(1) - Anzahl der Spalten im Fenster

PRINT RWINDOW(2) - 40 (im 40-Zeichen-Modus)
                   80 (im 80-Zeichen-Modus)
```

Wenn kein Fenster definiert ist, wird immer der ganze Bildschirm als Fenster betrachtet, und wir erhalten im 40-Zeichen-Modus nacheinander die Werte 24, 39 und 40.

## Externe Editoren

Besitzer älterer Commodore-Rechner und auch des C64, die nicht über die sehr komfortablen Editorfunktionen der Rechner der 8000- und 7000-Serie, des C16, C116, Plus/4 oder C128 verfügen, können neben Spracherweiterungen wie Honey.Aid (enthalten im SYBEX BASIC-Kurspaket) auch einen großen Teil der mittlerweile recht leistungsfähigen Textsysteme für Ihre Rechner einsetzen. Diese gestatten es oft auch, Programme zu bearbeiten und stellen, wie z. B. StarTexter, der im Funktionsvorrat in die Leistungsklasse von Profi-Software für die großen Personal Computer heranreicht, ein recht komfortables Instrumentarium für das Editieren von Programmen bereit.

Um eine Tokenisierung der BASIC-Schlüsselwörter von so erstellten, montierten oder modifizierten Programmen zu bewirken, reicht ein einfaches BASIC-Programm, das das editierte Programm von einer Diskette zeilenweise einliest, diese Zeilen in den Tastaturpuffer bringt und jeweils am Ende CHR$(34) für RETURN einfügt, so daß sie der ROM-residente BASIC-Editor wie normal eingegebene Zeilen akzeptiert und das Programm nach und nach in der tokenisierten Form aufbaut.

Alternativ ist auch eine zeilenweise Übernahme über die Bildschirmanzeige und manuelles RETURN für kleinere Programme denkbar.

# Kapitel 3
# Bestandteile von BASIC

## Generelle Eigenschaften des Commodore-BASIC

Für die Entwicklung und Anwendung von Programmen werden von den Commodore-Rechnern eine Reihe von Befehlen zur Verfügung gestellt. Bei den älteren Commodore-Systemen und auch noch beim VC20 und C64 reichte jedoch der Satz der Systembefehle nicht aus. Daher gibt es für diese Rechner eine große Zahl sogenannter Toolkits und BASIC-Erweiterungen, die hier Abhilfe schaffen.

BASIC 7.0 und auch in eingeschränktem Maße BASIC 3.5 machen auf dem Gebiet einen großen Schritt nach vorn. Die Liste der Befehle für die BASIC-Programmerstellung, das Testen von Programmen und die Steuerung von Diskettenlaufwerken ist wesentlich länger geworden, läßt jedoch immer noch Raum für Ergänzungen.

Durch die enge Verzahnung von Betriebssystem und BASIC-Interpreter und die Möglichkeit, die meisten BASIC-Befehle bis auf z. B. INPUT, INPUT#, GET, GET#, GETKEY, DEF FN, DATA im Direktmodus auszuführen, andererseits auch durch die teilweise Verwendbarkeit von Systemkommandos in BASIC-Programmen, z. B. RUN, sind die Grenzen zwischen Systemkommandos und BASIC-Befehlen fließend.

Wichtig ist jedoch zu erkennen, welche vielfältigen Möglichkeiten der Benutzer eines Commodore-Systems hat, durch direkte Anweisungen den Rechner und den Ablauf von Programmen zu steuern.

Auch wenn man nicht selbst programmieren will, wird man beispielsweise die Kommandos zur Steuerung der Peripheriegeräte häufig verwenden. Hier bieten BASIC 7.0 und 3.5 den gleichen Komfort wie BASIC 4.0. Commodore-Rechner mit BASIC 2.0 stellen zwar prinzipiell die gleichen Funktionen zur Verfügung, wobei allerdings die Handhabung im Direktmodus umständlich ist. Das liegt daran, daß die Peripheriekommandos nicht im Rechner ausgeführt werden, sondern von diesem zur Ausführung an das Peripheriegerät in Form von Zeichenketten übergeben werden.

Die Commodore-Peripherie besteht aus intelligenten Einheiten mit eigenen Prozessoren, Speichern und Betriebssystemen, in denen die jeweils benötigten

Funktionen implementiert sind. Daher ist klar, daß die Bereitstellung dieser Funktionen völlig unabhängig vom angeschlossenen Rechner und dessen BASIC-Version ist.

Die in der Regel am häufigsten verwendeten Befehle sind im übrigen auf den Rechnern mit BASIC 3.5 und 7.0 durch Funktionstasten auszulösen. Diese Zuordnung von Kommandos auf den 8 Funktionstasten können Sie sich anzeigen lassen und, falls gewünscht, ändern. Um die Standardbelegung nach dem Einschalten zu sehen, tippen Sie einfach:

```
KEY <RETURN>
```

und beim C128 erscheint folgende Liste:

```
KEY 1,"GRAPHIC"
KEY 2,"DLOAD"+CHR$(34)
KEY 3,"DIRECTORY"+CHR$(13)
KEY 4,"SCNCLR"+CHR$(13)
KEY 5,"DSAVE"+CHR$(34)
KEY 6,"RUN"+CHR$(13)
KEY 7,"LIST"+CHR$(13)
KEY 8,"MONITOR"+CHR$(13)
```

Beim C16 und C116 ist lediglich die Taste F8 anders belegt, und zwar:

```
KEY 8,"HELP"+CHR$(13)
```

und beim Plus/4 zusätzlich die Taste F1:

```
KEY 1,"SYS1525: 3-PLUS-1"
```

sonst stimmt die Belegung überein. Dabei ist CHR$(13) der Code für <RETURN>, und CHR$(34) liefert ein Anführungszeichen.

BASIC ist bewußt als eine einfache fast wie Englisch wirkende Sprache konzipiert worden. Dennoch müssen Sie einige Regeln beachten, wenn Sie aus einzelnen Befehlen Programme zusammensetzen wollen. Selbst im Direktmodus versteht Sie der BASIC-Interpreter nur, wenn Sie korrekt BASIC mit ihm sprechen. Sonst reagiert er mit Fehlermeldungen, anstatt die gewünschte Aktion auszuführen.

BASIC ist zum Leidwesen von Informatiklehrern keine so strenge Sprache wie Pascal oder MODULA, sondern erlaubt eine etwas weniger stark strukturierte Formulierung. Das erschwert natürlich andererseits eine knappe und präzise Beschreibung der BASIC-Syntax (Sprachregeln). Ganz im Sinne von BASIC

soll daher hier nur eine mehr umschreibende kurze Charakterisierung der wesentlichen Spracheigenschaften erfolgen.

Der "Direktmodus" und der "Programmodus" von BASIC unterscheiden sich bezüglich der Sprachregeln nur durch die Verwendung von Zeilennummern und Befehlen, die auf diese Bezug nehmen:

```
100 PRINT "DAS IST PROGRAMMODUS"
```

und

```
PRINT "DAS IST DIREKTMODUS"
```

## BASIC-Sprachelemente

### Anweisungen

Die kleinste ausführbare Einheit für den BASIC-Interpreter ist eine BASIC-Anweisung. Sie enthält mindestens

- eine Zuweisung oder
- neben anderen BASIC-Sprachelementen genau ein primäres BASIC-Befehlswort.

Sie ist entweder durch einen Doppelpunkt in derselben Zeile von anderen Anweisungen getrennt oder durch eine führende Zeilennummer in der nächsten Zeile gekennzeichnet (Ausnahme IF).

Eine einzige BASIC-Anweisung kann sich also nicht über mehrere Programmzeilen, wohl aber über mehrere Bildschirmzeilen erstrecken. Sie darf nicht mehr als ein primäres BASIC-Schlüsselwort enthalten.

### Beispiele

| | |
|---|---|
| `10 ABC` | Falsch, die Anweisung enthält kein primäres BASIC-Schlüsselwort. |
| `10 PRINT"ABC"` | Richtig. |
| `10 PRINT PRINT"ABC"` | Falsch, die Anweisung enthält mehr als ein primäres BASIC-Schlüsselwort. |

```
10 PRINT:PRINT"ABC"
```
Richtig, die Zeile enthält zwei zulässige BASIC-Anweisungen.

```
10 PRINT
20 PRINT"ABC"
```
Richtig, beide Anweisungen sind voneinander durch den Beginn einer neuen Programmzeile getrennt.

```
10 PRINT
20 PRINT
30 "ABC"
```
Falsch, die Anweisung in Zeile 20 kann sich nicht bis in die Zeile 30 erstrecken. Zeile 30 wird als selbständige Anweisung gewertet, enthält aber kein gültiges Schlüsselwort.

```
10 PRINT
   "ABC"
20 PRINT
```
Richtig, die Anweisung in Zeile 10 erstreckt sich über zwei Bildschirmzeilen.

## Datentypen und Zuweisungen

Eine Zuweisung ist eine spezielle Anweisung der Form:

```
LET Variable=Ausdruck
```

Dabei kann das Schlüsselwort LET wahlweise entfallen.

Zuweisungen dürfen immer nur typweise vorgenommen werden, also Zeichenkettenvariablen dürfen nur Zeichenketten zugewiesen bekommen, Integervariablen dürfen nur ganze Zahlen zugewiesen bekommen und Gleitkommavariablen nur Zahlen. Dabei zeichnen sich Zeichenkettenvariablen durch ein angehängtes Dollarzeichen (A$) und Integervariablen durch ein angehängtes Prozentzeichen (I%) aus.

### Beispiele

```
10 X="FALSCH"
```
Falsch, der Gleitkommavariablen X darf keine Zeichenkette zugewiesen werden.

```
10 X$="RICHITG"
```
Richtig, X$ ist eine Zeichenkettenvariable.

```
10 X=VAL("RICHTIG")
```
Richtig, die Funktion VAL liefert eine Zahl, die der Gleitkommavariablen X zugewiesen wird.

| | |
|---|---|
| `10 X$=STR$(10)` | Richtig, die in die Zeichenkette "10" umgewandelte Zahl 10 darf der Zeichenkettenvariablen X$ zugewiesen . |
| `10 X$=0` | Falsch, die Zahl 0 darf nicht der Zeichenkettenvariablen X$ zugeordnet . |
| `10 X%=0.5` | Wenn die Gleitkommazahl 0.5 der Integervariablen X% zugewiesen wird, werden die Nachkommastellen abgeschnitten. |
| `10 X%=INT(0.5)` | Richtig, das ganzzahlige Ergebnis der Funktion INT kann der Integervariablen X% zugewiesen werden. |

Eine BASIC-Zuweisung darf links vom Gleichheitszeichen nur eine Variable enthalten. Rechts vom Gleichheitszeichen muß ein zulässiger BASIC-Ausdruck stehen. Eine Zuweisung kann in BASIC wahlweise durch das BASIC-Schlüsselwort LET eingeleitet werden.

**Beispiele**

| | |
|---|---|
| `10 X+X=1` | Falsch, auf der linken Seite darf nur genau eine Variable stehen. |
| `10 1=X+2` | Falsch, auf der linken Seite muß eine Variable stehen. |
| `10 X+1=2` | Falsch, auf der linken Seite muß genau eine Variable stehen. |
| `10 X=1+2` | Richtig. |
| `10 LET X=1+2` | Richtig. |
| `10 X=1+2*X` | Richtig. |
| `10 X=1+2*COS(X)` | Richtig. |
| `10 X$="ABC"` | Richtig. |

Einer Zeichenkettenvariablen darf eine Zeichenkette von maximal 255 Zeichen Länge zugewiesen werden. Eine Integervariable darf nur Werte zwi-

schen -32767 und +32767 enthalten. Der Gleitkommavariablen dürfen nur Werte zwischen ±1.7*10↑37 und ±2.9*10↑-39 zugeordnet werden.

**Beispiele**

```
10 X=-2.9*10↑38
```
Falsch.

```
10 X= 1.7*10↑40
```
Falsch.

```
10 X=-1.7*10↑39
```
Richtig.

```
10 X%=40000
```
Falsch.

```
10 X%=-31000
```
Richtig.

```
10 X$="AAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAA"
```
Falsch, mehr als 255 Zeichen.

```
10 X$=""
```
Richtig.

Für Feldvariablen gelten für jedes Feld die gleichen Regeln wie für einfache Variablen gleichen Typs.

**Beispiele**

```
10 X%(11)=-40000
```
Falsch, die zugewiesene Zahl ist kleiner als -32767.

```
10 X(2)=1.7*10↑37
```
Richtig.

## Zeichen und Codes

Eine BASIC-Anweisung darf nicht π oder ein Zeichen mit einem internen Zeichencode, der kleiner als 129 ist, vor dem Auftreten des primären BASIC-Befehlswortes enthalten. Eine Ausnahme bilden dabei der Doppelpunkt und das Leerzeichen.

### Beispiele

| | |
|---|---|
| `10 A PRINT"BC"` | Falsch, A hat den Zeichencode 65. |
| `10 : PRINT"ABC"` | Richtig, der Doppelpunkt ist erlaubt. |
| `10 :  PRINT"ABC"` | Richtig, Doppelpunkt und Leerzeichen sind erlaubt. |
| `10  PRINT"ABC"` | Richtig, Leerzeichen sind gestattet, werden beim Listen aber bis auf eins weggelassen. |
| `10 : PPRINT"ABC"` | Falsch, P vor PRINT ist nicht erlaubt, da der Zeichencode von P kleiner als 129 ist. |
| `10 π PRINT` | Falsch, π vor PRINT ist nicht erlaubt. |
| `10 _  PRINT` | Richtig, der Zeichencode von _ ist größer als 128. |

## Primäre und sekundäre Schlüsselwörter

Eine BASIC-Anweisung darf hinter einem primären BASIC-Befehlswort nur zulässige sekundäre Befehlswörter enthalten.

### Beispiele

| | |
|---|---|
| `10 PRINT TAB(10) "ABC"` | Richtig, TAB ist nach PRINT ein zulässiges sekundäres Schlüsselwort. |
| `10 PRINT THEN "ABC"` | Falsch, THEN ist hinter PRINT kein zulässiges sekundäres Schlüsselwort. |

Hinter einem BASIC-Schlüsselwort dürfen nur die jeweils dort zulässigen Trennzeichen wie : , ; auftauchen.

**Beispiele**

| | |
|---|---|
| `10 PRINT;` | Richtig, das Semikolon ist hinter PRINT erlaubt. |
| `10 PRINT,` | Richtig, das Komma ist hinter PRINT erlaubt. |
| `10 PRINT:` | Richtig, der Doppelpunkt ist hinter PRINT erlaubt. |
| `10 LET;` | Falsch, das Semikolon ist hinter LET nicht gestattet. |
| `10 LET,` | Falsch, das Komma ist hinter LET nicht gestattet. |
| `10 IF : 1<2 THEN` | Falsch, der Doppelpunkt ist hinter IF nicht gestattet. |

Nach einem BASIC-Schlüsselwort dürfen Anführungszeichen (") und Klammern () nur verwendet werden, wenn sie dort zulässig sind. Dabei muß die Anzahl der öffnenden und schließenden Klammern übereinstimmen. Schließende Klammern dürfen nur im Anschluß an öffnende Klammern auftreten.

**Beispiele**

| | |
|---|---|
| `10 PRINT"  "` | Richtig, Anführungszeichen sind nach PRINT erlaubt. |
| `10 PRINT"ABC` | Zulässig, ein Anführungszeichen genügt, ist aber unschön. |
| `10 PRINT"ABC:PRINT` | Zulässig, wird aber anders als geplant ausgeführt. Der gesamte Rest hinter dem Anführungszeichen wird nach PRINT als Zeichenkette ausgegeben. |
| `10 PRINT (` | Falsch, öffnende Klammer nach PRINT ist nicht gestattet. |

| | |
|---|---|
| `10 PRINT TAB (` | Falsch, nach der öffnenden muß eine schließende Klammer folgen. |

Zwischen zwei Klammern muß ein für das vorhergehende Schlüsselwort zulässiger Parameter stehen.

**Beispiele**

| | |
|---|---|
| `10 PRINT TAB()` | Falsch, das Leerzeichen ist bei TAB als Argument nicht zulässig. |
| `10 PRINT TAB($)` | Falsch, das Dollarzeichen ist bei TAB als Argument nicht zulässig. |
| `10 PRINT TAB(0)` | Richtig, die Null ist bei TAB als Argument gestattet. |
| `10 PRINT TAB(512)` | Falsch, 512 ist für TAB als Argument zu groß. |

Je nach vorangehendem Befehlswort und nach Typ des folgenden sekundären Schlüsselwortes darf dieses in einer Anweisung mehrfach auftreten.

**Beispiele**

| | |
|---|---|
| `10 IF 1<2 THEN THEN` | Falsch, THEN darf nicht unmittelbar nach THEN auftreten. |
| `10 IF 1<2 THEN IF 2<3 THEN` | Richtig, IF darf nach IF erneut auftreten, wenn THEN dazwischenliegt. THEN darf nach THEN erneut auftreten, wenn IF dazwischenliegt. |
| `10 PRINT TAB(0);TAB(10)` | Richtig, TAB kann nach PRINT mehrfach auftreten. |

Bestimmte sekundäre Schlüsselwörter erfordern zwingend das vorherige Auftreten bestimmter anderer Schlüsselwörter in einer zulässigen Reihenfolge.

**Beispiele**

| | |
|---|---|
| `10 IF BEGIN` | Falsch, BEGIN nach IF ohne vorheriges Auftreten von THEN ist nicht zulässig. |
| `10 IF 1<2 THEN BEND` | Falsch, BEND nach THEN ohne vorheriges Auftreten von BEGIN ist nicht zulässig. |
| `10 IF 1<2 THEN BEGIN:BEND` | Richtig, die Reihenfolge der Schlüsselwörter nach IF ist korrekt. |

## Ausdrücke

Ausdrücke, die in BASIC-Anweisungen auftreten, müssen zulässig sein. Sie dürfen nur erlaubte Zeichen aus dem BASIC-Zeichenvorrat und gültige BASIC-Operatoren und BASIC-Funktionen sowie zulässige BASIC-Variablennamen enthalten. Dabei dürfen sie keine BASIC-Befehlswörter oder sekundären Schlüsselwörter enthalten.

Zulässige Zeichen, die als Konstanten in BASIC-Ausdrücken auftreten können, sind alle Buchstaben und Sonderzeichen des Alphabets.

**Beispiele**

| | |
|---|---|
| `10 IF 1<2 THEN` | Richtig, 1 und 2 sind zulässige Konstanten. |
| `10 IF WAIT 32,2 THEN` | Falsch, in einem Ausdruck darf kein BASIC-Befehlswort auftreten. |

Ein Ausdruck kann in seiner Minimalform aus nur einer Zahl oder Variablen bestehen.

**Beispiele**

| | |
|---|---|
| `10 IF 1 THEN` | Richtig, 1 ist ein gültiger Ausdruck. |
| `10 IF X THEN` | Richtig, X ist ein gültiger Ausdruck. |

*Anmerkung:* Die als Bedingungsklausel in einer IF-Anweisung auftretenden Ausdrücke werden von BASIC ohnehin durch Auswerten ihres Wahrheitsgehaltes auf eine einzige Zahl reduziert. Dabei repräsentiert -1 einen wahren Ausdruck, und alle anderen Zahlen stehen für eine nicht erfüllte Bedingungsklausel.

Sie können dies in einfacher Weise überprüfen. Geben Sie

```
PRINT 1<2 <RETURN>
```

ein. Die Ausgabe ist

```
-1
```

denn 1<2 ist offensichtlich wahr. Tippen Sie nun

```
PRINT 1>2 <RETURN>
```

ein, und es wird

```
0
```

ausgegeben, denn 1>2 ist offensichtlich falsch.

In Ausdrücken können auch aus mehreren Ziffern oder Zeichen zusammengesetzte Konstanten auftreten.

**Beispiele**

```
10 11=11
```
Richtig.

```
10 "AA"="AA"
```
Richtig.

## Variablennamen

Es dürfen immer nur zulässige Variablennamen verwendet werden. Das sind Variablennamen, die mit einem Buchstaben beginnen und kein BASIC-Schlüsselwort sind oder als Teil enthalten. Ferner dürfen sie nur aus Buchstaben des Alphabets und Ziffern bestehen. Am Ende eines Variablennamens kann entweder ein Buchstabe, eine Ziffer, ein Prozentzeichen (Integervariablen) oder ein Dollarzeichen (Zeichenkettenvariablen) stehen. Als Ausnahme müssen hier die indizierten oder Feldvariablen genannt werden, Sie enden mit einer schließenden Klammer (A$(I)).

Die Regeln für die Namenbildung von Feldvariablen entsprechen denen für die einfachen Variablen im vorderen mnemonischen Teil (mnemonisch - sinngebend abgekürzt). An diesen schließt sich immer zusätzlich ein Klammerausdruck an, der aus einer öffnenden und einer schließenden Klammer gebildet sein muß. Zwischen den beiden Klammern dürfen bis zu 255 ganze Zahlen, durch Kommas getrennt, stehen. Jede einzelne Zahl muß zwischen 0 und 32767 liegen. Anstelle von Zahlen können auch numerische BASIC-Ausdrücke eingesetzt werden, deren Auswertung eine ganze Zahl im zulässigen Wertebereich ergibt.

**Beispiele**

| | |
|---|---|
| `10 DIM AUTO$(5)` | Falsch, die Zeichenkettenvariable enthält das Schlüsselwort TO. |
| `10 DIM 1A$(100)` | Falsch, der Variablenname beginnt mit einer Ziffer. |
| `10 DIM A$(40000,2)` | Falsch, zu großer Dimensionierungsfaktor. |
| `10 DIM A#(3)` | Falsch, der Variablenname endet mit einem unzulässigen Zeichen. |
| `10 DIM A%(1,2,1)` | Richtig. |
| `10 AAAA%=INT(A)` | Richtig. |
| `10 X1=0.5` | Richtig. |

## Operatoren

Zulässige Operatoren in BASIC sind:

```
+, *, -, /, ↑, |, AND, OR, XOR, NOT, >, <, ><, <>,
<=, =<, >=,=>, =
```

XOR gibt es nur im BASIC 7.0. Bis auf NOT sind alle Operatoren zweistellig, das bedeutet, daß jeweils links und rechts von einem Operator ein Ausdruck steht. NOT darf nur links von einem Ausdruck auftreten:

```
PRINT NOT (A+B)
```

und bei XOR werden die Ausdrücke in Klammern dahinter gesetzt:

```
PRINT XOR (A,B)
```

### Beispiele

| | |
|---|---|
| `10 PRINT -1 NOT` | Falsch, NOT muß links vor dem Ausdruck stehen. |
| `10 PRINT NOT 1` | Richtig, NOT steht links. |
| `10 PRINT TRUE = 1<2` | Richtig, 1<2 ist ein gültiger Ausdruck, ebenso wie TRUE, also ergibt auch die Verknüpfung beider Ausdrücke einen gültigen Ausdruck. |

Bei der Anwendung der Operatoren sind selbstverständlich die Regeln der Mathematik und der Boolschen Logik anzuwenden. Es ist daher erlaubt, die aus der Schule bekannten Klammerungsregeln anzuwenden. Ansonsten gelten die Prioritätsbestimmungen für Operatoren.

Beispielsweise ist

`2+3*3` nicht gleich `(2+3)*3`

oder es ist

`2↑3+3` nicht gleich `2↑(3+3)`

Auch die Division durch null ist wie in der Mathematik verboten. Falls sie unbeabsichtigt, etwa bei der Division durch eine Variable, die 0 wird, durchgeführt wird, führt sie zu einer Fehlermeldung und zum Programmabbruch:

```
PRINT 5/N

?DIVISION BY ZERO ERROR
READY.
```

Für rein numerische Ausdrücke, d. h. Ausdrücke, bei denen nur Zahlen, Variablen und Ausdrücke, die bei ihrer Auswertung eine Zahl ergeben, auftreten, dürfen als Operatoren alle, die im Commodore-BASIC enthalten sind, eingesetzt werden.

Für Zeichenkettenausdrücke, d. h. Ausdrücke, die nur Zeichen aus dem ASCII-Zeichensatz, Zeichenkettenvariablen und Zeichenkettenausdrücke, die bei ihrer Auswertung eine Zeichenkette ergeben, sind alle Operatoren bis auf *, /, ↑, -, AND, OR, NOT und XOR zulässig.

Das Pluszeichen ist erlaubt, hat allerdings für Zeichenketten eine andere Bedeutung als für Zahlen. Der Operator + bewirkt, daß die rechts vom Operator stehende Zeichenkette an die links vom Operator stehende Zeichenkette hinten angehängt wird.

**Beispiele**

```
PRINT "SY"+"BEX"

C128$="C"+"128"
```

Insbesondere können auch die Vergleichsoperatoren angewendet werden.

Die Vergleichsoperatoren sind:

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

Wenn Vergleichsoperatoren auf Zeichenkettenausdrücke angewendet werden, findet die Auswertung der Zeichenketten von links nach rechts statt. Dabei wird zeichenweise der ASCII-Code beider Zeichenketten verglichen. Beim ersten Auftreten einer Code-Gleichheit wird die Zeichenkette, deren gerade untersuchtes Zeichen einen kleineren ASCII-Code als das entsprechende Zeichen der anderen Zeichenkette aufweist, als die kleinere Zeichenkette identifiziert. Also die Zeichenkette "KLEIN" ist kleiner als die Zeichenkette "KLUG". Beim Vergleich der Zeichenketten werden auch führende und nachfolgende Leerstellen mit ausgewertet. Dabei muß man wissen, daß ein Leerzeichen den ASCII-Code 32 hat.

Eine Verknüpfung von Zeichenkettenausdrücken mit numerischen Ausdrücken ist mit keinem der Operatoren zulässig.

**Beispiele**

`PRINT"C"+128` Falsch, Verknüpfung des Zeichenket-

tenausdrucks "C" mit dem numerischen Ausdruck 128.

`IF C<"1" THEN PRINT"JA"` Falsch, Vergleich eines Zeichenkettenausdrucks mit einem numerischen Ausdruck.

Unter Verwendung von Konvertierungsroutinen wie VAL, ASC oder STR$ ist eine Verknüpfung ungleichartiger Ausdrücke möglich.

**Beispiele**

`PRINT"C"+STR$(128)` Richtig, STR$ ergibt einen Zeichenkettenausdruck, nämlich "128".

`IF C<VAL("1") THEN PRINT"JA"` Richtig, VAL ergibt einen numerischen Ausdruck, nämlich die Zahl 1.

Beachten Sie bei der Verwendung der Boolschen Operatoren AND, OR, NOT und XOR, daß Sie Boolsche Ausdrücke mit Hilfe der Morganschen Regeln vereinfachen können. Der Ausdruck

```
(W AND Z) OR (W AND V) OR (W AND X)
```

entspricht dem Ausdruck

```
W AND (Z OR V OR X)
```

Die konsequente Anwendung der Morganschen Regeln erlaubt es oft, komplexe Ausdrücke zu vereinfachen und sie dadurch überschaubarer und für den Interpreter schneller auswertbar zu machen. Reaktivieren Sie also bitte Ihre alte Formelsammlung!

Die folgenden sieben Variationen eines kleinen Programms zur Erzeugung von sogenannten Wahrheitstabellen sollen eine Anregung sein, wie man auch schwierige logische Ausdrücke mit Rechnerhilfe leicht analysieren kann. Sie brauchen dafür dieses Programm nur entsprechend zu ändern. Die einzelnen Operanden können beispielsweise durch größere Ausdrücke ersetzt werden.

**1. Beispiel:** Wahrheitstabelle für AND

```
39000 REM*******************************
39010 REM*       WAHRHEITSTABELLE       *
```

```
39020 REM*******************************
39030 DIM WF(16)
39040 T$(1)="OPERAND1 "
39050 T$(5)="   OPERAND2 "
39060 T$(9)="        AND "
39070 WF$(1)="W":WF$(0)="F"
39080 WF(1)=-1:WF(2)=-1:WF(3)=0:WF(4)=0
39090 WF(5)=-1:WF(6)=0:WF(7)=-1:WF(8)=0
39100 SCNCLR
39110 PRINT"      WAHRHEITSTABELLE"
39120 PRINT
39130 PRINT"     OPERAND1 AND OPERAND2"
39140 PRINT
39150 PRINT
39160 WF(09)=    (WF(1) AND WF(5))
39170 WF(10)=    (WF(2) AND WF(6))
39180 WF(11)=    (WF(3) AND WF(7))
39190 WF(12)=    (WF(4) AND WF(8))
39200 FOR J=1 TO 9 STEP 4
39210 : PRINT TAB(0) T$(J);
39220 : FOR I=J TO J+3
39230 :   PRINT "    ";
39240 :   PRINT WF$(ABS(WF(I)));"  ";
39250 NEXT I,J
```

Wenn Sie das Programm laufenlassen, erhalten Sie die folgende Ausgabe:

```
       WAHRHEITSTABELLE

     OPERAND1 AND OPERAND2

OPERAND1     W       W       F       F
OPERAND2     W       F       W       F
    AND      W       F       F       F
READY.
```

Als zweites Beispiel wollen wir die Wahrheitstabelle für OR erstellen.

**2. Beispiel:** Wahrheitstabelle für OR

```
39000 REM*******************************
39010 REM*      WAHRHEITSTABELLE       *
39020 REM*******************************
39030 DIM WF(16)
```

```
39040 T$(1)="OPERAND1 "
39050 T$(5)="   OPERAND2 "
39060 T$(9)="          OR "
39070 WF$(1)="W":WF$(0)="F"
39080 WF(1)=-1:WF(2)=-1:WF(3)=0:WF(4)=0
39090 WF(5)=-1:WF(6)=0:WF(7)=-1:WF(8)=0
39100 SCNCLR
39110 PRINT"       WAHRHEITSTABELLE"
39120 PRINT
39130 PRINT"     OPERAND1 OR OPERAND2"
39140 PRINT
39150 PRINT
39160 WF(09)=     (WF(1) OR WF(5))
39170 WF(10)=     (WF(2) OR WF(6))
39180 WF(11)=     (WF(3) OR WF(7))
39190 WF(12)=     (WF(4) OR WF(8))
39200 FOR J=1 TO 9 STEP 4
39210 : PRINT TAB(0) T$(J);
39220 : FOR I=J TO J+3
39230 :   PRINT "    ";
39240 :   PRINT WF$(ABS(WF(I)));"   ";
39250 NEXT I,J
```

Wenn Sie das Programm laufenlassen, erhalten Sie die folgende Ausgabe:

```
       WAHRHEITSTABELLE

     OPERAND1 OR OPERAND2

OPERAND1      W       W       F       F
OPERAND2      W       F       W       F
      OR      W       W       W       F
READY.
```

Sie können dieses zweite Beispiel ganz schnell aus dem ersten entwickeln, wenn Sie nur das Wort AND in den Zeilen 39060, 39130 und 39160 bis 39190 durch das Wort OR ersetzen.

### 3. Beispiel: Wahrheitstabelle für NOT AND

Auch für dieses Beispiel verwenden Sie wieder das bereits existierende Wahrheitstabellenprogramm. Sorgen Sie nun dafür, daß folgende Zeilen wie gezeigt geändert werden:

```
39060 T$(9)="   NOT( AND )"
39130 PRINT " NOT( OPERAND1 AND OPERAND2 )"
39160 WF(09)= NOT(WF(1) AND WF(5))
39170 WF(10)= NOT(WF(2) AND WF(6))
39180 WF(11)= NOT(WF(3) AND WF(7))
39190 WF(12)= NOT(WF(4) AND WF(8))
```

Das Ergebnis des Programmlaufs sieht so aus:

```
      WAHRHEITSTABELLE

 NOT( OPERAND1 AND OPERAND2 )

OPERAND1      W       W       F       F
OPERAND2      W       F       W       F
NOT( AND )    F       W       W       W
READY.
```

**4. Beispiel:** Wahrheitstabelle für NOT OR

Hier müssen Sie gegenüber dem Beispiel 3 nur wieder das Wort AND durch das Wort OR austauschen. Der Programmlauf ergibt:

```
      WAHRHEITSTABELLE

 NOT( OPERAND1 OR OPERAND2 )

OPERAND1      W       W       F       F
OPERAND2      W       F       W       F
NOT( OR )     F       F       F       W
READY.
```

**5. Beispiel:** Wahrheitstabelle für AND NOT

Ändern Sie für dieses Beispiel die folgenden Zeilen:

```
39060 T$(9)="   AND NOT()"
39130 PRINT "OPERAND1 AND NOT( OPERAND2 )"
39160 WF(09)= (WF(1) AND NOT WF(5))
39170 WF(10)= (WF(2) AND NOT WF(6))
39180 WF(11)= (WF(3) AND NOT WF(7))
39190 WF(12)= (WF(4) AND NOT WF(8))
```

Das Ergebnis des Programmlaufs sieht so aus:

```
        WAHRHEITSTABELLE

OPERAND1 AND NOT( OPERAND2 )

OPERAND1      W       W       F       F
OPERAND2      W       F       W       F
AND NOT()     F       W       F       F
READY.
```

**6. Beispiel:** Wahrheitstabelle für OR NOT

Hierbei muß wieder nur gegenüber dem vorigen Beispiel 5 das Wort AND gegen das Wort OR ausgetauscht werden, und das Programmergebnis ist:

```
        WAHRHEITSTABELLE

OPERAND1 OR NOT( OPERAND2 )

OPERAND1      W       W       F       F
OPERAND2      W       F       W       F
OR NOT()      W       W       F       W
READY.
```

**7. Beispiel:** Wahrheitstabelle für OR 0

Tauschen Sie nun im Beispiel 6 das Wort NOT überall durch den Ausdruck 0 = aus, und Sie erhalten die folgende Beziehung:

```
        WAHRHEITSTABELLE

OPERAND1 OR 0 =( OPERAND2 )

OPERAND1      W       W       F       F
OPERAND2      W       F       W       F
OR 0 =()      W       W       F       W
READY.
```

Den logischen Operatoren AND, OR, NOT und XOR kommt im Zusammenhang mit Zahlen, insbesondere positiven, ganzen Zahlen, die im Wertebereich von 0 bis 15 liegen, also Byte-Werte repräsentieren, eine besondere Bedeutung bei der Manipulation einzelner Bits eines Bytes zu.

**Beispiele**

| Dezimalzahl Ergebnis | Operanden | Operator | Binärdarstellung Operanden | Ergebnis |
|---|---|---|---|---|
| | 2 | | 0010 | |
| 0 | | AND | | 0000 |
| | 8 | | 1000 | |
| | 15 | | 1111 | |
| 2 | | AND | | 0010 |
| | 2 | | 0010 | |
| | 10 | | 1010 | |
| 2 | | AND | | 0010 |
| | 6 | | 0110 | |
| | 10 | | 1010 | |
| 14 | | OR | | 1110 |
| | 6 | | 0110 | |
| | 12 | | 1100 | |
| 15 | | OR | | 1111 |
| | 3 | | 0011 | |
| | 10 | | 1010 | |
| 4 | | XOR(,) | | 0100 |
| | 14 | | 1110 | |

Wir sehen also, daß diese Operatoren und die Funktion XOR jeweils die Bits beider Zahlen pro Bitposition paarweise logisch verknüpfen. Beide Zahlen werden als jeweils 16stellige Bitzeichenkette betrachtet, auf die 16 elementare Bitverknüpfungen durchgeführt werden. In unserem Beispiel haben wir die Zahlen verkürzt dargestellt. Sie müssen sich vor jede Zahl so viele führenden Nullen denken, bis 16 Stellen voll sind.

Von diesen Operationen wird also auch das Vorzeichen betroffen, da die negativen Zahlen im sogenannten Zweierkomplement vorliegen, wobei eine 1 im höchstwertigen linken Bit auf eine negative und eine 0 dort auf eine positive Zahl hinweist.

*Anmerkung:* Sie sollten Bit-Operationen nur sehr zurückhaltend einsetzen und nur dort, wo sie z. B. zur Steuerung externer Schaltungen im Zusammenhang

mit den Befehlen WAIT, PEEK und POKE wirklich gebraucht werden. Verwenden Sie auf keinen Fall einzelne Bitpositionen zur Steuerung des Programmablaufs in Form von Bitleisten. Das führt leicht zu nicht mehr transparenten Programmen.

## Funktionen

In einem BASIC-Ausdruck dürfen auch die eingebauten Funktionen auftreten. Bei Zuweisungen hingegen können sie nur auf der rechten Seite verwendet werden.

### Beispiele

| | |
|---|---|
| `10 MID$("AAA",1,1)="A"` | Falsch, Funktionen dürfen bei Zuweisungen nicht auf der linken Seite vorkommen. |
| `10 IF LEN("AAA")=3 THEN` | Richtig, in Ausdrücken dürfen Funktionen auf beiden Seiten von Vergleichsoperatoren stehen. |

Dabei muß das Argument der Funktion natürlich dem zulässigen Typ entsprechen und innerhalb des jeweils erlaubten Wertebereichs liegen. Das Argument muß in Klammern stehen und kann selbst wieder ein BASIC-Ausdruck sein, dessen Auswertung durch den Interpreter ein zulässiges Argument für diese Funktion ergibt.

Die Zeichenkettenfunktionen STR$, VAL, DEC, RIGHT$, LEN, ASC, LEFT$, MID$ und INSTR akzeptieren als Argumente Zeichenketten bis zu 255 Zeichen Länge. Einige Funktionen wie MID$, INSTR, LEFT$ und RIGHT$ erfordern neben Zeichenketten noch weitere Argumente:

```
10 A$=MID$("ABC",2,1)

10 PO(A)=INSTR("ABC",MID$("ABC",2,1),1)
```

Die DEC-Funktion allerdings akzeptiert nur Zeichenketten der Länge 4, die auch nur die Ziffern 0 bis 9 sowie die Buchstaben A, B, C, D, E und F enthalten dürfen:

```
10 HX=DEC(LEFT$("FFFFGGGG",4))
```

Die VAL-Funktion läßt nur Zeichenketten, die aus Ziffern aufgebaut sind, als Argument zu. Zusätzlich kann man dabei noch + oder - als Vorzeichen sowie das E-Zeichen für die wissenschaftliche Darstellung von Gleitkommazahlen angeben:

```
10 PRINT VAL(STR$(99999))
```

Die numerischen Funktionen im Commodore-BASIC sind:

```
ATN, BUMP, COS, ERR$, EXP, FRE, HEX$, INT, JOY, LOG,
PEEK, PEN, POT, POS, POT, RCLR, RDOT, RGR, RND,
RSPCOLOR, RSPPOS, RSPRITE, RWINDOW, SGN, SIN, SPC, SQR,
TAB, TAN, XOR
```

Diese erwarten als Argumente in der Regel ganze Zahlen, teilweise sind aber auch Gleitkommazahlen zugelassen. Manchmal ist der erlaubte Wertebereich nur eine Teilmenge des für den jeweiligen Zahlentyp prinzipiell in BASIC vorgesehenen Bereichs.

### Beispiele

```
10 IF ER=255 THEN PRINT ERR$(ER)
```

Falsch, das Argument von ERR$ muß kleiner als 128 sein.

```
10 IF JOY(3)=135 THEN GOTO 99
```

Falsch, das Argument von JOY muß entweder 1 oder 2 sein.

Zusätzlich zu den schon eingebauten numerischen Funktionen können in Ausdrücken und Zuweisungen auch durch den Programmierer selbst definierte Funktionen Verwendung finden. Für den frei wählbaren Funktionsnamen gelten hierbei dieselben Regeln wie für Variablennamen. Jede benutzerdefinierte Funktion muß vor ihrem ersten Aufruf im Programm durch eine DEF FN-Anweisung definiert worden sein.

### Beispiel

```
10 MF%=FN MY(99):DEF FN MY(X)=INT(X/100)
```

Falsch, die Funktion MY muß zuerst definiert werden.

## Reservierte Wörter in BASIC 7.0

Reservierte Wörter stellen Zeichenkombinationen dar, die vom Programmierer nicht zur Bildung von Variablennamen verwendet werden dürfen. Zu diesen reservierten Wörtern gehören auch die Systemvariablen EL, ER, ERR$, DS, DS$, ST, TI und TI$, die alle bis auf TI und TI$ der Fehlerüberwachung dienen.

Weiterhin sind alle primären und die sekundären Schlüsselwörter reserviert und dürfen nicht zur Bildung von Variablennamen herangezogen werden.

Ferner dürfen nicht die BASIC-Operatoren, das #-Zeichen, π und die Separatoren " , ; : . ( ) [] in Variablennamen auftauchen.

Wir geben nun alphabetische Übersichten über die verschiedenen Typen von primären Befehlswörtern.

Es folgen primäre BASIC-Befehlswörter, die nicht innerhalb eines Programms, sondern nur im Direktmodus Verwendung finden dürfen. Im Programmodus erzeugen sie die Fehlermeldung:

```
?DIRECT MODE ONLY ERROR IN Zeilennummer
```

Die Befehle sind die folgenden:

AUTO, CONT (nur C64: im Programmodus Endlosschleife), DELETE, RENUMBER

Primäre BASIC-Befehlswörter, die keinen Parameter haben und zu denen auch keine sekundären Schlüsselwörter gehören:

CLR, CONT, END, FAST, GO 64, HELP, MONITOR, NEW, RESTORE, SLOW, SPRDEF, STOP, TROFF, TRON

Primäre BASIC-Befehlswörter mit Parameter:

ABS, APPEND, ASC, ATN, BACKUP, BANK, *BEGIN, *BEND, BLOAD, BOOT, BOX, BSAVE, BUMP, CATALOG, CHAR, CHR$, CIRCLE, CLOSE, CMD, COLLECT, COLLISION, COLOR, CONCAT, COPY, COS, *DATA, DCLEAR, DEC, *DEF, DIM, DIRECTORY, DLOAD, *DO, DOPEN, DRAW, DSAVE, DVERIFY, *ELSE, ENVELOPE, *EXIT, EXP, FILTER, *FOR, FRE, GET, GET#, GETKEY, *GOSUB, GOTO, GRAPHIC, GSHAPE, HEADER, HEX$, *IF, INPUT, INPUT#, INSTR, INT, JOY, KEY, LEFT$, LEN,

```
LET, LIST, LOAD, LOCATE, LOG, *LOOP, MID$, MOVSPR,
*NEXT, OPEN, PAINT, PEEK, PEN, PLAY, POINTER, POKE,
POS, POT, PRINT, PRINT#, PUDEF, RCLR, RDOT, *READ,
RECORD, REM, RENAME, *RESUME, *RETURN, RGR, RIGHT$,
RLUM, RND, RREG, RSPCOLOR, RSPPOS, RSPRITE, RUN,
RWINDOW, SAVE, SCALE, SCNCLR, SCRATCH, SGN, SIN, SLEEP,
SOUND, SPRCOLOR, SPRITE, SPRSAV, SQR, SSHAPE, STASH,
STR$, SWAP, SYS, TAN, TEMPO, TRAP, USR, VAL, VERIFY,
VOL, WAIT, WIDTH, WINDOW, XOR
```

Die mit einem Stern * gekennzeichneten primären Befehlswörter erfordern entweder nachfolgende sekundäre BASIC-Schlüsselwörter, oder sie korrespondieren zu einem oder mehreren anderen primären Befehlswörtern, die sich in einer anderen Anweisung befinden müssen, damit das Auftreten dieses primären Schlüsselwortes zulässig wird.

Solche Befehlspaare sind beispielsweise:

```
FOR    und NEXT
DO     und LOOP
READ   und DATA und DATA und DATA...
GOSUB  und RETURN
IF     und ELSE
BEGIN  und BEND
IF     und THEN
TRAP   und RESUME
```

Manche Befehlspaare, wie z. B. FOR und NEXT müssen immer zusammen auftreten, bei anderen, wie etwa bei TRAP und RESUME, ist es so, daß wohl einer von beiden alleine vorkommen kann (TRAP), der andere jedoch nicht ohne ihn (RESUME).

Bei den paarweise auftretenden Befehlen ist es natürlich erforderlich, daß die jeweilige Anzahl beider Befehlswörter gleich ist und daß sie in der richtigen Reihenfolge auftreten.

### Beispiele

```
10 FOR I=1 TO 9
20 : FOR J=1 TO 9
30 :   FOR K=1 TO 9
40 :     REM
50 :   NEXT K
60 : NEXT J
70 NEXT I
```

Richtig, die Anzahl von FOR und NEXT stimmt überein, und sie korrespondieren richtig.

```
10 FOR I=1 TO 9
20 : FOR J=1 TO 9
30 :   FOR K=1 TO 9
40 :     REM
50 : NEXT J
60 NEXT I
```

Falsch, die Anzahl von FOR und NEXT stimmt nicht überein.

```
10 FOR I=1 TO 9
20 : FOR J=1 TO 9
30 :   FOR K=1 TO 9
40 :     REM
50 NEXT K,J,I
```

Richtig, es ist erlaubt, mehrere FOR mit einem NEXT zu beenden. Dabei muß auf die korrekte Reihenfolge der Laufvariablen geachtet werden.

BASIC-Schlüsselwörter, die nicht selbständig in einer Anweisung auftreten dürfen, sind:

```
FN, ON, SPC, STEP, TAB, THEN, TO, UNTIL, USING, WHILE
```

Das Schlüsselwort ON nimmt hierbei eine Sonderstellung ein, weil es als einziges sekundäres Schlüsselwort vor dem primären Befehlswort in einer Anweisung stehen muß.

**Beispiele**

```
10 ON INT(X/100) GOSUB 500,600,700
```

Richtig, ON tritt vor GOSUB in einer Anweisung auf.

```
10 ON INT(X/100):GOSUB 600,700
```

Falsch, ON darf nicht auftreten, ohne daß in derselben Anweisung entweder GOSUB oder GOTO folgt.

Die restlichen sekundären Schlüsselwörter dürfen nur hinter dem zugehörigen primären Befehlswort in derselben BASIC-Anweisung wie dieses auftreten.

Diese Schlüsselwörter zerfallen noch einmal in zwei Gruppen:

- Wörter, die im Zusammenhang mit dem PRINT-Befehl auftreten;
- Wörter, die im Zusammenhang mit Sprachelementen zur Ablaufsteuerung von BASIC-Programmen auftreten.

Zu dieser zweiten Gruppe gehört auch das Schlüsselwort ON.

Nur das Schlüsselwort FN paßt in keine der beiden Gruppen. Es darf nur nach dem Befehlswort DEF innerhalb derselben Anweisung folgen.

### Beispiele

```
10 DEF FN MY(X)=INT(X/100*LOG(X))
```
Richtig.

```
10 DEF:FN MY(X)=(X/2)*4
```
Falsch, FN muß sich in derselben Anweisung wie DEF befinden.

Die Schlüsselwörter TAB und SPC dienen zur Steuerung der Schreibposition während der Druckausgabe mit dem PRINT-Befehl, wohingegen USING dazu dient, das vorangehende PRINT-Befehlswort als eine besondere Form des Druckbefehls zu kennzeichnen. Die PRINT USING-Variante des PRINT-Befehls erlaubt es, mit Hilfe der auf das sekundäre Schlüsselwort USING folgenden Druckmaske formatierte Ausgaben zu erzeugen.

### Beispiel

```
10 PRINT#2 USING MD$;TAB(10);SPC(3);M(1),M(2),M(3);
```

Richtig.

Die Schlüsselwörter TO und STEP dürfen nur hinter dem BASIC-Schlüsselwort FOR innerhalb derselben Anweisung auftreten, wobei das sekundäre Schlüsselwort TO obligatorisch und STEP wahlweise hinter TO auftreten kann.

### Beispiele

```
10 FOR I=0 STEP TRUE TO TRUE
```
Falsch, STEP tritt in der FOR-Anweisung vor TO auf.

```
10 FOR I=0 TO TRUE STEP TRUE
```
Richtig, STEP und TO kommen in derselben Anweisung wie der FOR-Befehl und außerdem in der richtigen Reihenfolge vor.

```
10 FOR I=0 TO TRUE:STEP TRUE
```

Falsch, STEP steht in einer anderen Anweisung als der zugehörige FOR-Befehl.

Die Schlüsselwörter UNTIL und WHILE, die im Zusammenhang mit den BASIC-Befehlswörtern DO und LOOP bei der Konstruktion von Wiederholungsschleifen auftreten, erlauben eine gewisse Flexibilität bei ihrem Einsatz. Sie können nämlich entweder in derselben Anweisung wie das DO-Befehlswort oder in derselben Anweisung wie das LOOP-Befehlswort erscheinen. Das ist insofern von Bedeutung, da dann die Gruppe von Anweisungen, die von der DO- und der LOOP-Anweisung eingeschlossen wird, mindestens einmal durchlaufen wird.

### Beispiele

```
10 DO
20 : PRINT"EIN DURCHLAUF"
30 : EXIT
40 LOOP WHILE 0
```

Richtig, aber die Schleife ist eigentlich überflüssig, da sie aufgrund der konstant falschen Bedingung nur einmal durchlaufen wird.

```
10 DO WHILE 0
20 : PRINT"NIE"
30 : EXIT
40 LOOP
```

Richtig, aber die Schleife wird nie durchlaufen, weil die Bedingung immer falsch ist.

```
10 UNTIL X=-1 DO
20 : PRINT"FALSCH"
30 : EXIT
40 LOOP
```

Falsch, UNTIL muß hinter DO stehen.

```
10 DO WHILE -1
20 : PRINT"EIN DURCHLAUF"
30 : EXIT
40 LOOP UNTIL 1=1
```

Richtig, aber die Bedingung in der LOOP-Anweisung behält die Oberhand.

```
10 DO
20 : DO WHILE -1
30 :   PRINT"U.S.W."
40 : LOOP
50 LOOP UNTIL 1=1
```

Richtig, aber die Schleife läuft endlos weiter, da die Bedingung konstant richtig ist.

```
10 DO
20 : PRINT"SCHLEIFE"
30 : GET A$
40 : IF A$<>"" THEN EXIT
50 LOOP
```

Richtig, Sie können die DO-Schleife auch ohne WHILE und UNTIL verwenden. Sie müssen aber für eine ordnungsgemäße Beendigung sorgen.

Das Befehlswortpaar READ und DATA sowie der TRAP-Befehl weisen eine Besonderheit auf. TRAP, der Befehl zum Abfangen von Fehlerbedingungen, kann irgendwo im Programm stehen. Wenn der Interpreter in einem ganz anderen Programmteil bei der Programmausführung auf einen Fehler stößt, springt er dennoch sofort an die durch TRAP spezifizierte Zeilennummer, wo dann die Fehlerbehandlungsroutinen stehen können.

Ein ähnlicher Fall liegt beim READ-Befehl vor. Trifft der Interpreter auf das Schlüsselwort READ, sucht er die nächste zu lesende DATA-Zeile, die an beliebiger Stelle im Programm stehen kann. Bei der nächsten Ausführung des READ-Befehls wird der Interpreter das nächste DATA-Element, vom Programmanfang gerechnet, hinter dem gerade gelesenen suchen und auswerten. Die DATA-Zeilen brauchen auch nicht hintereinander im Programm zu stehen. Es ist jedoch übersichtlicher, stets die zugehörigen Gruppen von DATA-Zeilen unmittelbar hinter dem jeweiligen READ-Befehl aufzuführen.

## Leerstellen

Zwischen den Schlüsselwörtern, den Parametern und Anweisungen brauchen keine Leerstellen zu stehen, da der Interpreter diese in der Regel auch so erkennt. Aus Gründen der Lesbarkeit sollte man jedoch recht großzügig mit Leerstellen umgehen, auch wenn das etwas mehr Speicherbedarf und Rechenzeit zur Folge hat.

Im Zusammenhang mit den Systemvariablen EL, ERR$, ER, DS$, DS, ST, TI, TI$ können aber auch für den Interpreter ohne trennende Blanks mehrdeutige Situationen entstehen, die ihn zu einer falschen Interpretation oder auch zu einer Fehlermeldung bewegen.

Angenommen Sie meinen:

```
IF F OR T THEN PRINT
```

und schreiben:

```
IF FORT THEN PRINT
```

dann führt das zu einer Fehlermeldung.

Kapitel 4

# Ein- und Ausgabe in BASIC

## Schnittstellen

Ihr Commodore-Rechner ist von Hause aus ein recht kommunikativer Typ, verfügt er doch in der Regel über mindestens 5 externe Anschlußmöglichkeiten, die man auch Schnittstellen nennt, von denen einige sogar mehrfach genutzt werden können. Das ist auch gut so, denn das schont Ihren Geldbeutel und gestattet schon in der Grundausstattung des Rechners vielfältige Einsatzmöglichkeiten. Denn erst die Ein- und Ausgabe von Daten, d. h. maschinenlesbar verschlüsselter Informationen, seien es Zahlen, Texte, Grafiken, Bilder, Töne oder Steuersignale, ermöglicht es, den Computer Aufgaben von praktischer Bedeutung erledigen zu lassen.

Die Vielfalt der auch an Heim- und Personalcomputer mittlerweile anschließbaren Zusatzgeräte bedingt es, daß verschiedene Anforderungen an die Anschlüsse des Rechners gestellt werden. Die Hauptmerkmale eines Anschlusses sind die Anzahl der Leitungen, die für Daten und Steuersignale zur Verfügung stehen, die erreichbare Übertragungsgeschwindigkeit und die Regeln, nach denen sich der Computer und das Peripheriegerät unterhalten müssen, damit keine Verständigungsschwierigkeiten auftreten.

Natürlich gibt es noch weitere Unterscheidungskriterien wie die elektrischen Anschlußwerte und Signalpegel und so banale wie die Form der Stecker. Damit hat so mancher Anwender schon seine traurigen Erfahrungen hinter sich, weil er aufgrund eines falschen Anschlusses seinen Rechner beschädigt hat oder weil ein von ihm gekauftes Zusatzgerät einen nicht passenden Anschlußstecker hat.

Die beiden leistungsfähigsten Schnittstellen sind der Userport und der Peripheriegeräte-Bus. Beide sind für relativ hohe Datendurchsätze angelegt, und an beide können mehrere Geräte angeschlossen werden.

Der Userport ist einerseits ein sehr flexibler und offener Anschluß, wird aber durch BASIC, außer durch die Befehle PEEK, POKE und WAIT, nicht weiter unterstützt. Allerdings gibt es bei den neueren Commodore-Rechnern rudimentäre Betriebssystem-Routinen, die einen Einsatz dieser Schnittstelle für Akustikkoppler oder als parallelen Druckeranschluß erlauben.

Der Peripheriegeräte-Bus wird ab BASIC 3.5 und 4.0 sehr komfortabel unterstützt, ist aber für den C128 und C16, C116, C64 und Plus/4 nur in einer wenig leistungsfähigen seriellen Form vorhanden. Das führt nicht nur zu einer geringeren Geschwindigkeit bei diesen Rechnermodellen, sondern schränkt auch deren Möglichkeiten, die sehr interessanten, leistungsfähigeren Peripheriegeräte einzusetzen, die für diese parallele Busform zur Verfügung stehen. Sie sind nicht nur von Commodore sondern aufgrund der Verträglichkeit dieses Anschlusses mit dem Industrie-Standardbus IEEE488 auch von anderen Herstellern erhältlich. Allerdings kann man die serielle Busform auch durch Zusatz-Hardware auf die parallele Form hochrüsten.

Die Bezeichnung Bus deutet an, daß mehrere Geräte gleichzeitig an diese Schnittstelle angeschlossen werden können und daß eine komplexe Steuerlogik notwendig ist, damit eine geordnete Datenübermittlung zwischen diesen Geräten erfolgen kann.

Der Peripheriebus dient primär dem Anschluß von Diskettenlaufwerken und Druckern. Beide Anschlüsse befinden sich in der Regel auf der Rückseite der Commodore-Rechner, wo auch der interne Rechnerbus über einen Anschluß erreichbar ist. Dieser Anschluß wird meist für externe RAM- oder ROM-Erweiterungen verwendet und spielt im Zusammenhang mit BASIC-Ein-/Ausgabeprogrammierung keine große Rolle, da auch der eventuelle Anschluß von Spezial-Hardware dort höchstens mit PEEK und POKE sowie WAIT Software-mäßig unterstützt wird.

Die Commodore-Rechner verfügen traditionell alle über mindestens einen Kassettenrecorder-Anschluß, der zwar nur für relativ geringe Übertragungsleistungen ausgelegt ist. Dafür gewährleistet er aber eine sichere Datenübertragung und wird auch ausreichend komfortabel durch einen speziellen Satz von BASIC-Befehlen unterstützt. Commodore verwendet für die Datenübertragung über diesen Anschluß ein ähnlich firmenspezifisches Protokoll wie beim seriellen Bus, was etwas hemmend auf das Angebot von Fremdgeräten wirkt. Dieser Anschluß wird gelegentlich auch für andere Peripherie zweckentfremdet, kann aber dann von BASIC aus nicht angesprochen werden.

Relativ interessant sind noch die rein analogen Anschlüsse für Steuerung und Eingabegeräte, die praktisch Analog-/Digitalwandler darstellen und sich zumeist an der rechten Seite des Rechners befinden. Primär ist dieser Anschluß, der meist doppelt vorhanden ist, für spezielle Eingabegeräte wie Lichtgriffel, Drehregler und Steuerknüppel gedacht und wird in BASIC 3.5 und 7.0 durch die Abfrage-Funktionen PEN, POT und JOY unterstützt. Er kann aber auch z. B. für Grafik-Tabletts, Thermometer und Spracheingabe verwendet werden. Über diesen Anschluß werden also keine direkten Datenströme gesendet, sondern der Rechner fragt anliegende Werte ab, deren fortlaufende Änderung mit einer Bedeutung belegt werden kann.

Der sicherlich nicht unwichtige externe Anschluß für Bildschirmgeräte, von denen der C128 gleich drei besitzt, wobei allerdings nur zwei sinnvoll gleichzeitig verwendbar sind, ist nicht direkt von BASIC aus zu beeinflussen. Ebenso ist der Tastaturanschluß, der bei den meisten Commodore-Rechnern intern realisiert ist, zwar nicht von BASIC aus ansprechbar, aber die Möglichkeit, Zusatz-Hardware anzuschließen, wie z. B. weitere Funktionstasten, kann Einfluß auf die Programmgestaltung im Zusammenhang mit der Tastaturabfrage haben, da so bisher freie Plätze der Tastaturmatrix auf einmal belegt sein können.

Das Betriebssystem unterscheidet die analogen Schnittstellen wie Steuerknüppel- und Bildschirmeingang grundlegend von den anderen Schnittstellen für die digitale Datenübertragung. Prinzipiell behandelt das Betriebssystem die letzteren alle ähnlich und stellt für sie gleichartige Befehle mit ähnlichem Aufbau bereit.

Zwar gibt es für die Tastatur, den Kassettenrecorder und den Bildschirm spezielle Varianten der allgemeinen dem IEEE-Protokoll angepaßten Befehlsform, doch lassen sich die letzteren auch für diese drei Sonder-Ein-/Ausgabegeräte verwenden. Diese einheitliche Sprachform bietet den großen Vorteil, leicht durch Änderung eines Befehlsparameters eine Ein- oder Ausgabe umzulenken oder von einem anderen als dem gewohnten Gerät entgegenzunehmen. So ist es durchaus möglich, den Tastatur-Input durch Eingabewerte vom Diskettenlaufwerk zu ersetzen oder Ausgaben wahlweise auf Bildschirm, Drucker oder Diskette zu lenken.

Der wichtigste Befehl bei der Ein- und Ausgabe ist sicherlich der OPEN-Befehl, der allen Ein-/Ausgaben vorangehen muß. Bei BASIC 4.0, 3.5 und 7.0 enthalten allerdings einige Befehle, wie z. B. DIRECTORY, bereits implizit den OPEN-Befehl. OPEN und die anderen Ein-/Ausgabebefehle besitzen eine etwas kompliziertere parametrische Struktur, als sonst in BASIC allgemein üblich ist. Außerdem sind diese Parameter nicht positionsgebunden, sondern sie werden aufgrund ihrer Kennzeichnung durch führende Symbole vom Interpreter erkannt.

Die Parameterlisten sind jedoch überwiegend gleichartig aufgebaut und enthalten folgende Befehlsparameter:

- KN: Kanalnummer oder logische Adresse oder Dateinummer
- GN: Gerätenummer
- HN: Hilfskanalnummer oder Sekundäradresse
- DN: Dateiname

Außer der Kanalnummer können die Parameter wahlweise weggelassen werden. Dann werden automatisch die folgenden Standardwerte angenommen:

- GN: 1, für Kassettenrecorder
- HN: 0, für Normalverarbeitung
- DN: "", für Dateinamen

Durch die Kanalnummer wird der Bezug zwischen dem mit GN angegebenen Gerät und den Ein-/Ausgabebefehlen hergestellt, die explizit diese Kanalnummer enthalten müssen. Ein in Ausführung befindliches Programm kann durchaus mehrere logische Verbindungen zu einem Gerät gleichzeitig aufrechterhalten. Das ist z. B. für das abwechselnde Lesen oder Schreiben in verschiedene Dateien auf der gleichen Diskette unbedingt erforderlich, um die einzelnen Ein-/Ausgabevorgänge aufgrund der Kanalnummer sauber auseinanderhalten zu können. Man kann sagen, daß also immer genau eine Beziehung zwischen einer Kanalnummer und einer logischen Datei besteht. Deshalb werden diese beiden Begriffe oft als Synonym verwendet. Zusätzlich wird auch der Begriff "logische Adresse" hierfür gebraucht.

Die Kanalnummer hat auch insofern eine ganz konkrete Bedeutung, als über sie auch für diese logische Verbindung zwischen Programm und logischer Datei/Gerät ein spezieller Ein-/Ausgabebereich, Puffer genannt, im Rechner reserviert wird. Daher ist es zwingend, daß eine Datei zu einer Zeit nur mit einer bestimmten logischen Kanalnummer geöffnet sein kann, da aufgrund der Pufferung die vermeintliche Ausgabe in einem Programm erst zu einem späteren Zeitpunkt zu einer tatsächlichen Ausgabe auf das Gerät führt. Das könnte bei mehreren Puffern zu einem unkontrollierten Überholen einer Ausgabe durch einen anderen Ausgabeprozeß führen, weil die Puffer aufgrund unterschiedlicher Satzlängen verschieden schnell geleert und gefüllt werden.

Ein Versuch, eine schon geöffnete Datei, d. h. eine, zu der schon eine Kanalverbindung besteht, erneut zu öffnen, führt daher zu einer Fehlermeldung.

Eine weitere Restriktion, noch aus der Zeit von knappen Hauptspeichergrößen stammend, läßt nur maximal 10 gleichzeitig offene Dateien zu.

Als Wertebereich für die logische Kanalnummer sind die Zahlen 1 bis 255 zugelassen, wobei bezüglich der Ausgabe auf Druckern wichtig ist, daß für logische Adressen 128 bis 255 Wagenrücklauf und Zeilenvorschub nach einem PRINT#-Befehl ausgegeben wird und bei 1 bis 127 nur Wagenrücklauf. Das gilt allerdings nur für PRINT-Befehle, die nicht mit einem Semikolon abgeschlossen werden. Für Diskettenausgabe ist jedoch unbedingt ein Wert unter 127 zu wählen.

Folgende Gerätenummern sind fest vergeben:

| | |
|---|---|
| - Tastatur: | 0 |
| - Kassettenrecorder 1: | 1 |

| | |
|---|---|
| – Kassettenrecorder 2: | 2 |
| – Bildschirm: | 3 |

Folgende Gerätenummern werden üblicherweise verwendet:

| | |
|---|---|
| Drucker 1: | 4 |
| Plotter oder Drucker 2: | 5 |
| Plotter oder Drucker 3: | 6 |
| Plotter oder Drucker 4: | 7 |
| Disketteneinheit 1: | 8 |
| Festplatte oder Disketteneinheit 2: | 9 |
| Festplatte oder Disketteneinheit 3: | 10 |
| Sonderperipherie: z. B. Meßgeräte | 11 ff. bis 31 |

Angaben über 31 sind zwar möglich, werden aber nicht ordnungsgemäß verwaltet. Im Normalfall ist man mit 31 angeschlossenen Geräten auch gut bedient.

Der Hilfskanal wird bei der Diskette für unterschiedliche Datenkanäle und beim Drucker zur Betriebsmodus-Steuerung verwendet. Beim Kassettenrecorder wird die Datenübertragungsrichtung sowie das Schreiben der Datei-Endemarke über den Hilfskanal gesteuert.

Der Dateiname muß nur bei der Ein-/Ausgabe auf Diskette angegeben werden und mindestens ein Zeichen lang sein. Das ist jedoch auch für die Ein-/Ausgabe auf Kassettenrecorder zu empfehlen.

Bei der Diskettenein- und -ausgabe gibt es außerdem noch drei Parameter mit folgender Bedeutung:

- Diskettenidentität (ID) - Hierbei handelt es sich um eine zweistellige Zahl, die dem Benutzer zur zusätzlichen Diskettenkennzeichnung dient, vom Betriebssystem des Diskettenlaufwerks aber nicht weiter verwendet wird.

- Diskettenname (FN) - Auf den Diskettennamen trifft sinngemäß die gleiche Aussage wie bei der Diskettenidentität zu. Er darf maximal 16 Zeichen lang sein.

- Laufwerknummer (LN) - Die Laufwerknummer ist jedoch für das Diskettenlaufwerk eine signifikante Information, die angibt, auf welchem Laufwerk die zugehörige Diskettenoperation ausgeführt werden soll. Teilweise ist die Angabe eines Laufwerks zwingend, ansonsten ist das Laufwerk D0 voreingestellt. Es sind nur die Angaben D0 oder D1 möglich.

## Die Tastatur

Die Tastatur ist sicherlich für die meisten Commodore-Besitzer weiterhin noch das wichtigste Eingabegerät neben dem Steuerknüppel oder Joystick. Alle Informationen müssen in der Regel erst einmal über die Tastatur erfaßt werden, und fast alle Systemkommandos werden primär über die Tastatur eingegeben und ausgelöst.

Die Tastatur wird vom Betriebssystem als Gerät mit der Nummer 0 betrachtet, d. h. wir könnten den vom Betriebssystem dafür vorgesehenen OPEN-Befehl verwenden, um eine spezielle Verbindung zwischen dem eigentlichen Rechner und der Tastatur herzustellen. Das ist allerdings normalerweise nicht notwendig, da BASIC schon gleich nach der Systeminitialisierung eine logische Verbindung für die Datenübermittlung von der Tastatur zum Rechner herstellt. Dafür braucht vom Bediener und Programmierer kein spezielles Kommando eingegeben zu werden.

Die Verbindung Tastatur/Rechner wird von BASIC durch drei spezielle Formen der generellen Eingabebefehle INPUT# und GET# unterstützt. Es handelt sich um die Anweisungen GET, GETKEY und INPUT, bei denen keine Kanalnummer angegeben werden muß und die nur im Programmodus verwendet werden dürfen.

GET und GETKEY lesen die Tastatur zeichenweise, d. h. sie werten jeden Tastendruck sofort aus, wohingegen INPUT eine ganze Zeichenkette in eine Variable übernimmt. Diese Zeichenkette kann je nach Rechner 78 (C64) bis 160 (C128) Zeichen vom Bildschirm übernehmen. Zusammen mit dem Fragezeichen, das der INPUT-Anweisung als Eingabeaufforderung an den Benutzer dient, kann ein Hinweistext ausgegeben werden, der hinter dem Befehlwort als Textkonstante, nicht als Zeichenkettenvariable, aufgeführt wird:

```
100 INPUT"WIE FUEHLEN SIE SICH";F$
```

Es können dem Benutzer mit einer INPUT-Anweisung auch mehrere Eingaben abverlangt werden. Dazu wird hinter dem Semikolon eine Variablenliste geführt, und die Eingaben, durch Kommas getrennt, werden den Variablen eine nach der anderen zugeordnet. Dabei muß streng darauf geachtet werden, daß die Datentypen übereinstimmen:

```
100 INPUT"GEBEN SIE NAMEN, ALTER, GROESSE UND
GEWICHT AN";N$,A,GR,GE
```

Bei GET und GETKEY sind ebenfalls Variablenlisten zugelassen, und den Variablen wird jeweils entsprechend ihrer Abfolge genau ein Zeichen in der

Reihenfolge der Tastendrücke zugewiesen. Auch dabei müssen die Datentypen übereinstimmen.

GET und GETKEY, das es nur in BASIC 3.5 und 7.0 gibt, unterscheiden sich dadurch, daß bei GETKEY der Computer so lange auf die Eingabe wartet, bis sie erfolgt, während es bei GET genau darauf ankommt, eine Taste zu drücken, wenn gerade die Tastatur abgefragt wird.

**Beispiel**

```
100 REM REAKTIONSTEST
110 A$=""
120 T$="NIETE"
130 GET A$
140 IF A$<>"" THEN T$="TREFFER"
150 PRINT T$
160 END
```

Lassen Sie das Programm mehrmals laufen. Wenn Sie mehrere Treffer erzielen, sind Sie wirklich gut. Besser klappt es mit folgender Änderung:

```
130 GET A$:IF A$="" THEN 130
```

oder einfacher:

```
130 GETKEY A$
```

GET und GETKEY eignen sich gut für Abfragen auf J(a) oder N(ein) oder zur Zifferneingabe, aber kaum zur Eingabe von längeren Wörtern oder größeren Zahlen.

Die INPUT-Anweisung kann das besser. Ein kleiner Trick läßt das Fragezeichen, falls es einmal unerwünscht sein sollte, verschwinden. Eröffnen Sie den Bildschirm als Eingabedatei:

```
100 OPEN 1,0,99,"DUMMY"
110 PRINT"GEBEN SIE IHREN WEIHNACHTSWUNSCH AN:"
120 INPUT#1,W$
130 PRINT:PRINT"AHA, EIN(E) ";W$
140 END
```

## Ausgabe auf den Bildschirm

Für die Ausgabe auf den Bildschirm gibt es ähnlich wie bei der Tastatur eine einfachere Form des PRINT#-Befehls, die keine Kanalnummer benötigt, da diese spezielle logische Verbindung beim Einschalten des Rechners schon automatisch vom Betriebssystem vorgenommen wurde.

Der PRINT- und der CHAR-Befehl können Zeichenketten bis zu einer Länge von 255 Zeichen drucken. Dabei erfolgt die Ausgabe beim PRINT-Befehl Zeile für Zeile, vom oberen Bildschirmrand fortschreitend, während mit dem CHAR-Befehl gezielt auf eine bestimmte Bildschirmposition gedruckt wird.

Normalerweise wird nach jedem PRINT-Befehl ein Zeilenvorschub ausgelöst. Ein fortlaufendes bündiges Drucken in derselben Zeile kann beim PRINT-Befehl durch Anhängen eines Semikolons erreicht werden, und das Anhängen eines Kommas steuert die nächste voreingestellte Tabulatorposition an. Ferner können die Positionierungsfunktionen POS, SPC und TAB mit PRINT verwendet werden. Mit Hilfe des Schlüsselwortes USING und der Angabe einer speziellen Druckmaske für den formatierten Ausdruck können PRINT und PRINT# sehr flexibel eingesetzt werden. Es besteht allerdings die Einschränkung, daß PRINT nur im Text- und im Blockgrafikmodus eingesetzt werden kann.

Sowohl im Text- als auch im hochauflösenden Grafikmodus kann der CHAR-Befehl für die Ausgabe von Zeichen auf den Bildschirm verwendet werden. Die Bildschirm-Steuercodes (siehe unter dem Stichwort CHR$), die innerhalb der zu druckenden Zeichenketten eingebettet werden können, wirken bei dem CHAR-Befehl jedoch auch nur im Textmodus.

Da der Bildschirmspeicher mit Ausnahme des 80-Zeichen-Bildschirmspeichers des C128 bei allen Commodore-Rechnern innerhalb der ersten Bank des Arbeitsspeichers liegt, ist es zudem auch möglich, mit POKE unmittelbar dort hineinzuschreiben. Dies führt aber in der Regel nicht zu guter Programmierung und wird daher nicht empfohlen.

Commodore-Rechner haben generell mindestens zwei Darstellungsarten auf dem Bildschirm:

- den Groß-/Kleinschriftmodus;
- den Großschrift-/Blockgrafikmodus;

sowie zusätzlich noch

- den DIN-Modus, den es nur beim C128, 8032 SK, 8296 und den 700-Rechnern gibt und der durch Drücken der Taste ASCII/DIN die deutsche Tastaturbelegung verfügbar macht.

Die Modi unterscheiden sich durch die jeweils verwendeten Zeichensätze, die zwar nicht mischbar, aber jederzeit im Direktmodus oder unter Programmsteuerung umschaltbar sind. Dabei zeigt der erste Modus die Buchstaben in Groß- und Kleinschrift und verfügt über einen eingeschränkten Blockgrafiksatz. Der zweite Modus besteht aus Großbuchstaben mit einem erweiterten Halbgrafiksatz. Der dritte Zeichensatz ist mit dem ersten bis auf die Umlaute und einige wenige Sonderzeichen identisch, verwendet allerdings einen völlig anderen Zeichengeneratorsatz, dessen Buchstaben aus wesentlich dünneren Linien bestehen und der sich mehr für den 80-Zeichen-Schirm als für den 40-Zeichen-Schirm eignet.

Zwischen dem Groß-/Kleinschriftmodus und dem Großschrift-/Grafikmodus kann direkt durch Drücken der Commodore- und der SHIFT-Taste hin- und hergeschaltet werden. Im Programmodus oder auch direkt kann dazu der Befehl PRINT CHR$(14) bzw. PRINT CHR$(142) benutzt werden.

Zusätzlich ist der C128 noch von einem 80-Zeichen-Monitor auf eine 40-Zeichen-Schirm hin- und herschaltbar, wobei beide Bildschirme aufgrund getrennter Bildschirmspeicher gleichzeitig anzeigen können. Wenn Sie ein Fernsehgerät als Monitor betreiben, bleibt Ihnen allerdings der 80-Zeichen-Modus verschlossen.

Außer durch die Taste mit der Aufschrift 40/80 DISPLAY können Sie auch vom Programm aus mit Hilfe des Befehl PRINT CHR$(27)+"X" oder durch Drücken der Tasten ESC und danach X zwischen dem 40- und dem 80-Zeichen-Bildschirm hin- und herschalten.

Alle neueren Commodore-Rechner sind zudem farb- und bis zu mittlerer Auflösung grafikfähig. Das heißt, in allen Modi können die gedruckten Zeichen noch verschieden gefärbt sein.

Im Gegensatz zum zeichenorientierten Drucken mit PRINT und CHAR werden in den Grafikmodi, die in BASIC 3.5 und 7.0 mit dem GRAPHIC-Befehl eingestellt werden können, einzelne Punkte (Pixels) auf dem Bildschirm direkt angesteuert, und zwar:

- hochauflösender Grafikmodus - 320 * 200 Punkte
- Grafik/Text gemischt - 320 * 160 Punkte plus 5 Textzeilen
- Mehrfarbenmodus - 160 * 200 Punkte mehrfarbig
- Grafik/Text gemischt im Mehrfarbenmodus - 160 * 160 Punkte mehrfarbig plus 5 Textzeilen

Für diese vier Grafikmodi gibt es einen speziellen Satz von Grafikbefehlen, die in allen diesen Modi einsetzbar sind: BOX, CIRCLE, COLOR, DRAW, GRAPHIC, LOCATE, PAINT, RCLR, RDOT, RLUM, SCALE und WIDTH.

Zusätzlich kann hier noch der Befehl CHAR zur Beschriftung von Grafiken verwendet werden, da dieser in allen Bildschirmmodi einsetzbar ist.

Die Befehle COLOR und GRAPHIC dienen dazu, bestimmte Bildschirmattribute wie Farben, Auflösung und Aufteilung festzulegen. Diese Festlegungen werden in der Regel von den anderen Befehlen bis auf die Zeichenfarbe nicht verändert.

COLOR erlaubt die Färbung von Bildschirmrahmen, Hintergrund- und Vordergrundfarbe, und der GRAPHIC-Befehl ermöglicht es, den Rechner in einen der vier oben aufgeführten Grafikmodi zu versetzen.

Die Befehle SCALE und WIDTH sind ebenfalls Befehle ohne direkte Ausgabe. Diese Hilfskommandos beeinflussen den Zeichenmaßstab und die Strichstärke für Linien. SCALE gestattet es dem Programmierer, logisch auf einen fast 900 x 900 Zeichen großen Schirm quasi Millimeterraster zu plazieren. Durch Veränderung des Skalierungsfaktors für aufeinanderfolgende Ausgaben sind Zoom-Effekte realisierbar.

WIDTH, der weniger spektakuläre Befehl von diesen beiden, verändert die Strichstärke von eine auf zwei Pixellinien und umgekehrt.

CIRCLE, BOX und DRAW sind die Ausgabebefehle für den Grafikmodus. Sie erlauben aufgrund ihrer Parametervielfalt das Zeichnen einer unglaublichen Fülle verschiedener elementarer grafischer Gebilde schon mit einem Ausgabebefehl.

DRAW, der Befehl zum Zeichnen von Linien, ist noch der simpelste und am einfachsten verständliche Befehl, da er entweder Linien zwischen zwei explizit angegebenen Punkten oder von der aktuellen Cursorposition zu einem bestimmten Punkt zieht. Das Zeichnen einer Strecke bestimmter Länge in einem bestimmten Winkel ist allerdings mit DRAW nicht möglich. Das kann man jedoch selbst mit elementaren geometrischen Umformungen erreichen.

**Beispiel**

```
100 PRINT CHR$(147)
110 INPUT"GEBEN SIE DIE LAENGE EIN";L
120 IF L<0 OR L>319 THEN 110
130 GRAPHIC 1,1
140 DRAW 1,0,100 TO L,100
150 GETKEY T$
160 GRAPHIC 0,1
```

Prinzipiell ist der DRAW-Befehl schon völlig ausreichend, um jede beliebige grafische Ausgabe, insbesondere jede grafische Figur, zu zeichnen, da sich diese alle durch eine Vielzahl, wenn auch teilweise sehr kleine, Linienstücke darstellen lassen. Das würde jedoch einen sehr großen Programmieraufwand und enorme Rechenzeiten mit sich bringen.

Glücklicherweise ist Commodore mit den BASIC-Versionen 3.5 und 7.0 diesmal nicht auf halbem Wege stehengeblieben, sondern hat mit BOX und CIRCLE zwei weitere Befehle zur Verfügung gestellt, die die Erzeugung praktisch jeder elementaren zweidimensionalen Grundfigur erlauben.

BOX plaziert ein Viereck mit beliebigen Proportionen an der angegebenen Stelle auf dem Bildschirm. Um Ihnen und dem Rechner die komplizierten Umrechnungen bei nicht rechtwinkligen Bildaufbauten zu ersparen, können diese Rechtecke noch um einen beliebigen Winkel gedreht werden. Ein weiterer Parameter des BOX-Befehls macht es möglich, das Viereck direkt in einer von drei vom Hintergrund verschiedenen Farben auszufüllen.

Der leistungsfähigste und aufgrund seiner Komplexität anfangs vielleicht etwas schwer durchschaubare Befehl ist CIRCLE, mit dem beliebige Ellipsen, Kreise und N-Ecke auf den Schirm gezaubert werden können. Aufgrund der Vielfalt der übrigen Parameter ist allerdings das Spezifizieren einer Füllfarbe nicht vorgesehen. Sie können jedoch den Mittelpunkt, die Radien, den Drehungsfaktor sowie die Darstellung nur eines Segmentes, also eines Teilstücks der eigentlich definierten geometrischen Figur, frei wählen. Für Segmente ist zusätzlich noch die Angabe eines Drehwinkels möglich.

Um so erstellte Ellipsen, Vielecke und Segmente auszumalen, besitzen BASIC 3.5 und 7.0 den PAINT-Befehl. Wichtig dabei ist allerdings, daß die auszufüllende Fläche wirklich vollständig von einem Linienzug umgeben ist. Ansonsten läuft nämlich sozusagen die Farbe aus und färbt den ganzen Bildschirmhintergrund.

Ein auf den ersten Blick unscheinbarer Befehl LOCATE leistet hierbei wichtige Dienste. Der LOCATE-Befehl ist deshalb so unscheinbar, weil der durch ihn positionierte grafische Cursor im Gegensatz zum Cursor im Textmodus nicht sichtbar ist und LOCATE selbst keine Ausgaben erzeugt. Er gestattet es jedoch, die aktuelle Schreibstelle, die sich ja zumeist genau auf der umgrenzenden Linie der zuletzt gezeichneten Figur befindet, in das Innere derselben zu bewegen, damit sie mit PAINT gefärbt werden kann. Selbstverständlich kann es auch sinnvoll sein, statt des Figurinneren die Umgebung mit einer der von der Hintergrundfarbe verschiedenen Tönungen, die der PAINT-Befehl erlaubt, auszufüllen.

Die Funktionen RDOT, RCLR, RGR, sowie die nur in BASIC 3.5 verfügbare Funktion RLUM (aufgrund der etwas leistungsfähigeren Farbunterstützung

durch die Hardware) ermöglichen es dem Benutzer, dem Programmierer und auch dem Programm, bei all diesen Aktionen nicht den Überblick zu verlieren. Fast alle aktivierten Attribute des aktuellen Bildschirmcodes sowie die aktuelle Grafik-Cursorposition können mit ihnen ermittelt werden.

Ein effektives Arbeiten mit dem LOCATE-Befehl kommt beispielsweise erst durch Hinzunahme der RDOT-Funktion zustande. Sie ermittelt im Gegensatz zur POS-Funktion des Textmodus sowohl die X- als auch die Y-Koordinate getrennt und gleichzeitig, ob an der aktuellen Cursorposition ein Punkt gesetzt ist oder nicht. So kann durch Abfahren von Bildschirmteilen mit dem Cursor über den LOCATE-Befehl unter gleichzeitiger Abfrage mit RDOT eine gezielte Analyse von Bildschirmsegmenten zur Vorbereitung einer weiteren Bildschirmausgabe vorgenommen werden. Das ermöglicht kontextsensitive Grafikoperationen.

Die ständige Verfügbarkeit von Informationen über die aktuelle Cursorposition befreit den Programmierer von der mühseligen Aufgabe, diese selbst zu verwalten und ständig nachzuführen.

Mit RGR kann der eingestellte Grafikmodus erfragt werden, damit z. B. entsprechende Formatierungen von Ausgaben und Prüfungen auf Einhaltung von Koordinatengrenzen oder entsprechende Skalierungen abgeleitet werden können.

RCLR und RLUM ermöglichen die Abfrage der Farbtönungen aller Elemente des Grafikbildschirms, wie Rahmen, Hintergrund etc., und darauf abgestimmt die Parameter des folgenden COLOR-Befehls oder die Farbwahlparameter der übrigen Befehle entsprechend zu setzen.

Last not least wollen wir hier noch den SCNCLR-Befehl erwähnen, der es erlaubt, vor der Erstellung neuer Grafiken sozusagen den Schirm sauberzuputzen oder nach Beendigung von Ausgaben wieder einen definierten Grundzustand herzustellen. Eine Tatsache, die Anfänger zuerst etwas verwirren mag, ist, daß SCNCLR nur die Grafikausgaben löscht, aber ansonsten alle Grafikattribute, die mit GRAPHIC oder COLOR gesetzt wurden, völlig unberührt läßt. Das ist besonders zu beachten, wenn nach Beendigung einer Grafikausgabe in den Textmodus zurückgeschaltet wird und die normalen Textfarben gewünscht werden. Man sollte es sich zur Gewohnheit machen, jedesmal mit COLOR, GRAPHIC und SCNCLR wieder einen sauberen Grundzustand herzustellen, wobei nach Möglichkeit diese Reihenfolge einzuhalten ist.

Bei Ihrem C128 erhalten Sie die gewohnte Farbeinstellung mit

```
COLOR 0,12 :REM HINTERGRUNDFARBE
COLOR 4,14 :REM RAHMENFARBE
COLOR 5,14 :REM ZEICHENFARBE
```

Ganz besondere Grafikbefehle stehen für die Erzeugung, Steuerung und Überwachung von beweglichen Grafikobjekten, den sogenannten Sprites, zur Verfügung. Es handelt sich um die Befehle und Funktionen: BUMP, COLLISION, MOVSPR, RSPCOLOR, RSPPOS, RSPRITE, SPRCOLOR, SPRDEF, SPRITE und SPRSAV, die ebenfalls ausführlich im alphabetischen Teil besprochen werden.

Die Hard- und Software-technische Unterstützung animierbarer grafischer Objekte ist sozusagen eine Spezialität der Atari- und Commodore-Heimcomputer. In der DV-Technik zwar allgemein als MOBs - Movable OBjects - bekannt, haben sie sich hier unter dem Namen Sprites bei den Freunden von Grafikspielen beliebt gemacht. Beim C64 mußte man bisher noch auf die Eigenerstellung von Sprites und die Programmierung ihrer Steuerung aufgrund fehlender Software-Unterstützung verzichten. Es sei denn, man verfügte über den im SYBEX BASIC-Kurs enthaltenen Multicolor-Sprite-Editor oder gleichwertige Programmierhilfen.

Mit dem eingebauten Sprite-Editor und den zusätzlichen Befehlen für Hardware-unterstützte MOBs, den Sprites, und für Software-unterstützte MOBs, den Shapes, muß man nicht unbedingt Maschinenprogrammierung lernen, um ein Sprite zum Laufen zu bringen oder gar ein komplettes Grafikprogramm zu entwickeln.

Sie brauchen nur SPRDEF einzugeben, und schon befinden Sie sich im Sprite-Editor, der Ihnen komfortablerweise erlaubt, Sprites direkt auf dem Bildschirm in achtfacher Vergrößerung zu zeichnen und gleichzeitig eine Anzeige des echten Sprite-Bildes auf der rechten Bildschirmhälfte zu verfolgen.

Mit dem SPRITE-Befehl können Sie Ihre Kreationen dann aktivieren, färben, ihnen eine Priorität zuordnen, sie vergrößern oder in den Mehrfarbenmodus überführen.

Mit MOVSPR kann dem Sprite eine Richtung und eine bestimmte Geschwindigkeit zugewiesen werden, mit der es sich dann automatisch fortbewegt, bis es explizit gestoppt wird.

Eine weitere Möglichkeit, Sprites zu erstellen, ist, im hochauflösenden Grafikmodus unter Verwendung aller dort verfügbarer Befehle oder mit einem speziellen Hilfsprogramm eine Grafik zu zeichnen und diese mit SSHAPE in eine Variable zu übertragen. Diese Variable kann mit Hilfe der POINTER-Funktion und dem BSAVE-Befehl auf Diskette abgespeichert werden. Dann können Sie dieses Shape mit BLOAD jederzeit wieder laden und mit SPRSAV in ein Sprite umwandeln oder mit dem Befehl GSHAPE als binäre Bildinformation in den Grafikspeicher bringen.

Selbstverständlich können Sie Ihre Shape-Variableninhalte auch ganz normal Satz für Satz in eine sequentielle Datei auf Diskette oder Kassette ablegen.

Wenn sich nun die Sprites auf dem Bildschirm bewegen, kann es z. B. sein, daß man feststellen will, ob etwa zwei Sprites zusammengestoßen sind, wie es bei einem Treffer in einem Grafikspiel der Fall sein könnte. Dafür bieten sich die Befehle BUMP und COLLISION an. Dabei erlaubt der COLLISION-Befehl, ähnlich wie bei der Fehlerbehandlung, ohne fortgesetztes Abfragen dennoch, bei einer Sprite-Kollision zu einem speziellen Unterprogramm zu verzweigen. Diese Verzweigung ist unabhängig davon, welche Anweisung im Programm gerade ausgeführt wird. Mit der Funktion BUMP läßt sich dabei feststellen, welche Sprites betroffen waren.

Im Zusammenhang mit Ausgaben auf den Bildschirm kann man noch über die Musikbefehle sprechen, bei denen es sich allerdings mehr um akkustische Ausgaben handelt.

Die Befehle PLAY, SOUND, TEMPO, ENVELOPE und FILTER, die mit ausführlichen Beispielen in der alphabetischen Übersicht in Kapitel 8 dargestellt werden, erlauben eine Programmierung von Sound- und Musikeffekten in einer viel natürlicheren Weise, als dies mit direkter Manipulation der Register des Sound Interface Chips, kurz SID genannt, möglich wäre.

Mit PLAY kann man direkt Melodien durch einfaches Aneinanderreihen von einzelnen Tönen in einer Zeichenkette angeben und abspielen. Dabei kann man 3 Stimmen und 6 Oktaven ertönen lassen.

Außerdem stehen 10 sogenannte Hüllkurven zur Verfügung die eine Klangcharakteristik dieser Töne in der Art erlauben, daß sie einigermaßen ähnlich wie bestimmte Instrumente, z. B. Klavier, Flöte, Gitarre, Trompete oder Orgel, erklingen. Wenn Sie mit diesen vordefinierten Instrumenten nicht zufrieden sind, können Sie mit den Befehlen ENVELOPE und FILTER neue Instrumente oder Klangquellen nachbilden oder erfinden.

Für weniger musikalische Geräusche wie Explosionen, Piepsen oder Schußgeräusche gibt es noch den Befehl SOUND.

TEMPO und VOL schließlich legen die Geschwindigkeit und die Lautstärke, mit der Ihre musikalischen Innovationen gespielt werden sollen, fest und verändern sie nach Belieben.

## Ausgabe vom Bildschirm

Wenn keine hochauflösende Grafik benutzt wird, ist das Abspeichern von Daten, die auf dem Bildschirm stehen, recht einfach. Beim C64 sind es bei-

spielsweise 40 x 25 Zeichen, die im Speicher ab der Adresse 1024 stehen. Diese 1000 Zeichen können mit unserer unter dem Stichwort DEF selbst definierten Funktion BS einzeln ausgelesen und weggeschrieben werden. Unter dem Stichwort PEEK haben wir eine Hardcopy-Routine aufgeführt, die diese Bildschirm-Codes in ASCII-Zeichen umwandelt und auf den Drucker ausgibt.

Will man die Zeichen aber wieder auf den Bildschirm ausgeben, so ist diese Umwandlung von Bildschirm-Codes in ASCII-Zeichen unnötig, da wir für die Ausgabe auf den Bildschirm mit POKE auch nur Bildschirm-Codes gebrauchen können. Wir wollen sie deshalb in einem Vektor ablegen und hinterher ausnahmsweise mit dem POKE-Befehl wieder auf den Bildschirm bringen, nachdem wir diesen zwischendurch gelöscht haben. Wir wollen damit zeigen, wie man den POKE-Befehl, falls er unumgänglich sein sollte, am besten in Form eines Unterprogramms verwendet, damit nur an einer Stelle im Programm POKE-Befehle auftreten. Mit Hilfe von Kommentaren kann dann ein solches Programm, in dem POKE-Befehle sind, übersichtlich und lesbar gemacht werden. Das wird auch an unserem kleinen Beispiel klar.

```
100 BA=1023 :REM BILDSCHIRMANFANG-1
110 DIM Z(1000)
120 DEF FNBP(I)=PEEK(BA+I)
130 REM LESEN DER BILDSCHIRMZEICHEN
140 FOR I=1 TO 999
150 : Z(I)=FNBP(I)
160 NEXT I
170 REM BILDSCHIRM LOESCHEN
180 PRINT CHR$(147)
190 REM SCHREIBEN DER BILDSCHIRMZEICHEN
200 FOR I=1 TO 999
210 : GOSUB 900 SCHREIBE ZEICHEN
220 NEXT I
230 END
900 REM SCHREIBEN IN DEN BILDSCHIRMSPEICHER
910 : POKE BA+I,Z(I)
920 RETURN
```

Möchte man jetzt diese Zeichen beispielsweise auf einer Diskette zwischenspeichern, so muß man eine sequentielle Datei eröffnen, diese beschreiben und dann wieder schließen.

Das Programm dazu sieht das so aus:

```
100 OPEN 1,8,2,"ZEICHEN,S,W"
110 DEF FNBP(I)=PEEK(1023+I)
```

```
120 REM LESEN DER ZEICHEN VOM BILDSCHIRM
130 FOR I=1 TO 999
140 : PRINT#1,FNBP(I)
150 NEXT I
160 CLOSE 1
170 END
```

Nun wollen wir uns ansehen, ob es auch geklappt hat. Wenn wir das Inhaltsverzeichnis der Diskette betrachten, sehen wir, daß die sequentielle Datei mit dem Namen ZEICHEN dazugekommen ist. Um diese Datei wieder auf den Bildschirm zu bringen, benutzen wir im folgenden Programm den POKE-Befehl:

```
100 OPEN 1,8,2,"ZEICHEN,S,R"
110 FOR I=1 TO 999
120 : INPUT#1,Z
130 : POKE 1023+I,Z
140 NEXT I
150 CLOSE 1
160 END
```

Das Ausdrucken von hochauflösenden Grafikbildschirmen kann allerdings in reinem BASIC aus Zeitgründen nicht sinnvoll durchgeführt werden, da dazu sowohl die einzelnen Punkte des Bildschirmspeichers als auch die des Farbspeichers für den Bildschirmausdruck gewertet werden müssen. Es ist daher ratsam, dafür ein spezielles Maschinenprogramm zu schreiben, oder noch besser, eine BASIC-Erweiterung einzusetzen, die über eine Hardcopy-Funktion für mehrfarbige Grafikbilder verfügt.

## Ausgabe auf den Drucker

Die Standard-Schnittstelle für Drucker ist bei den Commodore-Rechnern der Peripheriebus. Daran lassen sich selbstverständlich alle Commodore-Drucker ohne weiteres anschließen. Viele andere Drucker sind jedoch nicht mit einer seriellen sondern mit einer Centronics- oder RS232-Schnittstelle ausgerüstet. Dazu gehören beispielsweise auch die weit verbreiteten Epson-Drucker.

Es ist in jedem Fall möglich, diese Drucker über ein jeweils spezielles Interface anzuschließen. Diese Interface-Karten werden von verschiedenen Firmen angeboten und ermöglichen dann auch das Drucken aller Grafikzeichen.

Für die Ausgabe auf den Drucker gibt es außer PRINT# noch den CMD-Befehl. Beiden muß ein entsprechender OPEN-Befehl vorangehen, der die logische Kanalnummer zuteilt.

Beispiel:

```
OPEN 1,4
```

Der Unterschied zwischen CMD und PRINT# besteht darin, daß mit CMD alle folgenden Ausgaben mit LIST oder PRINT auf das entsprechende Gerät, hier den Drucker, umgelenkt werden, während mit PRINT# jeweils nur eine Zeile übergeben wird. Außerdem ist es empfehlenswert, nach der Ausgabe mit CMD mit PRINT# mindestens eine Leerzeile zu drucken, damit der Puffer vollständig geleert wird:

```
100 OPEN 1,4
110 CMD1
120 LIST
130 PRINT#1
140 CLOSE 1
150 END
```

Wenn der OPEN-Befehl bei der Herstellung der Verbindung zwischen Rechner und Drucker mit der Sekundäradresse 7 angegeben wird:

```
OPEN 1,4,7
```

ist beim Drucker der Groß-/Kleinschriftmodus angesprochen. Ein Weglassen der Sekundäradresse ist gleichbedeutend mit der Sekundäradresse 0:

```
OPEN 1,4,0
```

oder

```
OPEN 1,4
```

und bedeutet, daß die Ausgabe auf dem Drucker im Großschrift-/Grafikmodus erfolgen soll. Während der Druckausgabe können einzelne Zeichen oder Zeilen in dem jeweils anderen Modus gedruckt werden. Die Umschaltung muß dann mit PRINT# und der Angabe CHR$(17) bzw. CHR$(145) erfolgen. Weitere brauchbare Steuerzeichen, die ebenfalls mit PRINT# gesendet werden können, sind:

| | |
|---|---|
| CHR$(10) | Zeilenvorschub |
| CHR$(14) | doppelt breit drucken |
| CHR$(15) | Ende doppelt breit drucken |
| CHR$(17) | in Groß-/Kleinschrift drucken |
| CHR$(145) | in Großschrift/Grafik drucken |
| CHR$(18) | revers drucken |
| CHR$(146) | Ende revers drucken |

Es ist auf jeden Fall ratsam, sämtliche Druckausgaben zunächst auf dem Schirm zu testen. Dazu geben Sie im OPEN-Befehl einfach 3 als Geräteadresse an:

```
OPEN 1,3
```

und schon erscheint die ganze Ausgabe auf dem Bildschirm.

## Ein- und Ausgabe von Kassette

Das Laden eines Programms von Kassette erfolgt mit dem Befehl

```
LOAD
```

Es wird das nächste auf der Kassette befindliche Programm geladen. Sucht man nach einem bestimmten Programm, dann muß man auch den entsprechenden Namen mit angeben.

```
LOAD "Dateiname"
```

Wenn beim Laden eines Programms keine Gerätenummer angegeben wird, wird automatisch von der Kassette geladen, also die Gerätenummer 1 angenommen.

Sobald der Ladevorgang beendet ist, kann das Programm mit RUN gestartet werden.

Schneller geht die ganze Prozedur, wenn man die beiden Tasten SHIFT und RUN/STOP gleichzeitig drückt. Damit wird das nächste Programm von der Diskette bzw. Kassette geladen und automatisch gestartet.

Um eine Programmdatei auf Kassette zu speichern, benutzt man den Befehl

```
SAVE"Dateiname"
```

Der Dateiname kann zwar wahlweise entfallen, es ist jedoch ratsam, einen Namen zu vergeben, da das Programm dann sicherer wieder aufgefunden werden kann. Es ist selbstverständlich, daß die Kassette zum Wiederauffinden einer Datei bis mindestens vor deren Anfang zurückgespult werden muß. Das und die langsame Aufzeichnungs- und Lesegeschwindigkeit sowie Platzprobleme beim Überschreiben von zu löschenden oder alten Programmversionen machen das Arbeiten mit dem Kassettenrecorder wenig komfortabel.

Für das Schreiben und Lesen von sequentiellen Dateien werden auch hier die Befehle PRINT#, INPUT# und GET# verwendet.

Leider lassen sich Programme, die mit einem VC20 abgespeichert wurden, auf dem Commodore 64 nicht mehr einlesen, da sie mit einer höheren Aufzeichnungsdichte gespeichert wurden, als dieser es erwartet. Es erscheint die Fehlermeldung "Out of Memory". Wenn man kein Diskettenlaufwerk hat, muß man die Programme noch einmal eintippen.

## Ein- und Ausgabe von der Diskette

Die Gerätenummern, die für die Diskettenlaufwerke voreingestellt sind, sind die Nummern 8 und 9 (für ein eventuelles zweites Laufwerk). Beim OPEN-Befehl können für die Verbindung zwischen Rechner und Diskettenlaufwerk außerdem noch die Sekundäradressen 0 bis 15 angegeben werden. Diese Adressen sind mit der folgenden Bedeutung belegt:

| | |
|---|---|
| 0 | Laden von Programmen |
| 1 | Speichern von Programmen |
| 2 bis 14 | zur freien Verfügung |
| 15 | Öffnen des Kommando- oder Fehlerkanals |

Um eine Programmdatei von Diskette zu laden oder auf Diskette zu speichern, ist zunächst jedoch kein OPEN-Befehl notwendig. In BASIC 2.0 muß beim Laden eines Programms die Gerätenummer 8 mit angegeben werden:

```
LOAD "Dateiname",8
```

Bei den anderen BASIC-Versionen heißt es:

```
DLOAD "Dateiname"
```

Wenn man nicht mehr genau weiß, wie man die Datei genannt hat oder wenn der Dateiname sehr lang ist, dann lohnt es sich, zunächst das Inhaltsverzeichnis der Diskette aufzurufen:

```
LOAD "$",8          beim C64
```

oder

```
DIRECTORY           bei allen anderen
```

außerdem

```
CATALOG
```
beim C128

Dann fährt man mit dem Cursor zu dem entsprechenden Programmnamen hoch, tippt DLOAD davor und löscht den Rest der Zeile nach dem Dateinamen. Beim C64 und VC20 setzt man LOAD davor und ,8: hinter die zweiten Anführungszeichen. Wenn nun der Cursor hinter dem Doppelpunkt blinkt, kann man SHIFT und RUN/STOP gleichzeitig drücken, und das Programm wird, nachdem es in den Arbeitsspeicher geladen worden ist, automatisch gestartet. Falls kein automatischer Start gewünscht wird, drückt man einfach RETURN.

Der Dateiname muß nicht immer vollständig angegeben werden, er kann durch sogenannte Joker ergänzt werden. Die Angabe eines Fragezeichens ersetzt dabei ein einzelnes Zeichen und ein Stern * das ganze Ende eines Namens. So kann z. B. "ST??W*" für die Namen STARWARS1, STARWARS2 und STRUWELPETER stehen.

Selbstverständlich können Sie den LOAD-Befehl auch in den BASIC-Versionen 4.0, 3.5 und 7.0 zum Laden des Inhaltsverzeichnisses einer Diskette verwenden. Bei dieser Form des Ladens wird ein im Arbeitsspeicher befindliches Programm überschrieben. Damit es nicht verlorengeht, sollte es vorher auf Diskette abgespeichert worden sein.

Dafür kann man das so geladene Inhaltsverzeichnis dann wie eine Programmdatei behandeln und z. B. wiederholt auflisten.

Außerdem sind auch hier wieder besondere Qualifizierungen durch Joker zur Selektion bestimmter Datei- oder Programmgruppen möglich. Will man beispielsweise nur sequentielle Dateien für die Datenverwaltung ansehen, die alle mit dem Präfix DATVER beginnen, so gibt man:

```
LOAD"$0:DATVER*=S",8
```

ein. Für relative Dateien der Datenverwaltung heißt es:

```
LOAD"$0:DATVER*=R",8
```

und für die Programme

```
LOAD"$0:DATVER*=P",8
```

Der DIRECTORY-Befehl bietet diese Möglichkeiten nicht so ohne weiteres. Dafür läßt er ein im Speicher befindliches Programm unversehrt, weil er

direkt von der Diskette in den Bildschirmspeicher schreibt. Deshalb läuft auch das Diskettenlaufwerk so lange weiter, bis das gesamte Verzeichnis aufgelistet ist, also auch, wenn mit der NO SCROLL-Taste beim C128 die Ausgabe auf dem Bildschirm angehalten wird. Die Ausgabe kann durch erneutes Drücken dieser Taste fortgesetzt werden.

Um eine Programmdatei abzuspeichern, heißt es in BASIC 2.0:

```
SAVE"Dateiname",8
```

und in den anderen BASIC-Versionen:

```
DSAVE"Dateiname"
```

Dabei ist darauf zu achten, daß fabrikneue Disketten zuvor formatiert werden, was in BASIC 2.0 mit dem Befehl:

```
OPEN 1,8,15,"N:Diskettenname,Kennung"
```

und in den anderen BASIC-Versionen mit:

```
HEADER"Diskettenname",IKennung
```

Es kann zuweilen vorkommen, daß Sie ein Programm unter einem bereits vergebenen Namen speichern. Obwohl keine Warnung und keine Fehlermeldung erfolgt, ist dieses neue Programm nicht gespeichert. Sie könnten höchstens am blinkenden roten Lämpchen des Diskettenlaufwerks merken, daß es nicht geklappt hat, wenn Sie darauf achten. Also achten Sie bei der Namensvergabe darauf, daß Sie stets einen neuen Namen wählen.

Wenn Sie hingegen absichtlich einen schon vergebenen Namen wählen, um die betreffende Datei zu überschreiben, müssen Sie im SAVE-Befehl den sogenannten Klammeraffen einsetzen:

```
SAVE"@:Dateiname",8
```
in BASIC 2.0

und

```
DSAVE"@Dateiname"
```
in den anderen BASIC-Versionen

Ein äußerst nützlicher Befehl, wenn auch sträflicherweise selten gebraucht, ist VERIFY bzw. DVERIFY. Er dient dazu, das Programm im Arbeitsspeicher mit dem Programm, dessen Name im Befehl angegeben wird, auf der Kassette bzw. Diskette zu vergleichen. Diesen Befehl sollte man zur Sicherheit nach jedem Speichern mit SAVE verwenden.

# Kapitel 5

# Funktionen

Die eingebauten Funktionen im Commodore-BASIC können Sie als Start für eine kleine Bibliothek von gebrauchsfertigen Unterprogrammen betrachten, die Sie durch Definition eigener Funktionen nach und nach ausbauen können, indem Sie beispielsweise die vorhandenen miteinander kombinieren.

In BASIC 3.5 und 7.0 sind gegenüber den früheren BASIC-Versionen ein paar elementare Funktionen für Zahlen- und Zeichenkettenverarbeitung hinzugekommen. Jedoch die Funktionen für Grafik, Sprites, Farb-, Ton- und Bildschirmverwaltung und Rechnersteuerung sind wesentlich zahlreicher geworden als im BASIC 2.0 oder 4.0.

Kennzeichnend ist jedoch auch für diese neuen Funktionen weiterhin, daß sie in der Regel nur ein Argument haben, nur einen Wert zurückliefern und nur in Ausdrücken, jedoch niemals auf der rechten Seite einer Zuweisung, vorkommen dürfen. Als weitere Einschränkung gilt, daß benutzerdefinierte Funktionen lediglich arithmetische Funktionen sein können.

In BASIC, insbesondere im Commodore-BASIC, sind die Grenzen zwischen Operatoren und Funktionen im Laufe der Weiterentwicklung der Sprache noch stärker verwischt worden, als sie es ohnehin schon waren. Wenn wir beispielsweise das Potenzieren und die Umkehrung dazu, das Wurzelziehen, betrachten, wird diese Aussage deutlich: Die erste Operation ist als Operator (↑), die andere als Funktion (SQR) verfügbar.

Die meisten logischen Verknüpfungen sind Operatoren, das exklusive Oder (XOR) jedoch ist als Funktion implementiert.

Ebenso hätte manche Funktion im Bereich Bild-, Farb-, Grafik-, Ton- und Fenstersteuerung genauso gut als Befehl mit entsprechenden Parametern eingerichtet werden können. Auch umgekehrt ist dies der Fall.

Lassen Sie sich also dadurch nicht verwirren, und betrachten Sie die entsprechenden Funktionen, Operatoren und auch Befehle jeweils im Zusammenhang. Eine Funktion erkennen Sie stets daran, daß die Angabe eines Argumentes in Klammern notwendig ist und daß die Funktion ein Ergebnis liefert. Außerdem können alle Funktionen in Ausdrücken, also auch selbst wieder als Funktionsargumente und bei Befehlsparametern sowie auf beiden Seiten von verknüpf-

ten Ausdrücken, auftreten. Das ist ein Grund für ihre enorme Vielseitigkeit und Leistungsfähigkeit.

## Gliederung nach Funktionstypen

Am übersichtlichsten können die Funktionen nach den verschiedenen Typen von Ausdrücken klassifiziert werden, bei denen sie auftreten können.

Die Ausdrücke selbst haben wir in Kapitel 3 nach ihren Ergebnissen unterschieden und dabei festgestellt, daß es

- numerische Ausdrücke und
- Zeichenkettenausdrücke

gibt.

Numerische Ausdrücke kann man jedoch noch feiner nach den Wertebereichen ihrer Ergebnisse differenzieren. Wir unterscheiden:

| | |
|---|---|
| – Gleitkomma-Ausdrücke | Wertebereich +/-2.9E-39 bis +/-1.7E+38 |
| – Integer-Ausdrücke | Wertebereich -32768 bis +32767 |
| – Byte-Ausdrücke | Wertebereich 0 bis 15 |

Im Zusammenhang mit Bedingungen und anderen Logik-Fragestellungen, wo das Ergebnis nur insofern interessiert, ob es -1 oder irgendeine andere Zahl, also wahr oder unwahr ist, haben wir es mit Ausdrücken zu tun, deren Bereich vom Interpreter nur als zweiwertig aufgefaßt wird. Man kann diese Ausdrücke auch als logische Ausdrücke bezeichnen.

Funktionen können also, je nachdem, welche Typen von Ausdrücken sie als Argumente akzeptieren und welcher Datentyp als Ergebnis verwendet wird, in folgende Kategorien eingeordnet werden:

- Gleitkomma-Funktionen
- Integer-Funktionen
- Byte-Funktionen
- logische Funktionen
- Zeichenkettenfunktionen

Manche Funktionen gehören, je nach Betrachtungsweise, in zwei Bereiche hinein. Das sind die Funktionen, die als Argument einen anderen Datentyp verlangen als der für das Ergebnis verwendete darstellt. Dies gilt insbesondere für Funktionen, die primär zur Umwandlung von Daten von einem Datentyp in einen anderen eingesetzt werden.

Wir wollen im folgenden die Funktionen mehr praxisorientiert nach ihrer Funktionalität innerhalb der folgenden Grobeinteilung betrachten:

- mathematische Funktionen
- Zeichenkettenfunktionen
- Systemfunktionen

## Mathematische Funktionen

### Mathematische Funktionen zur Zahlenumwandlung

Die elementarsten arithmetischen Funktionen stellen INT und ABS dar. Sie sind auf ganze Zahlen anwendbar, die im EDV-Sprachgebrauch auch Integer-Zahlen genannt werden, und auf Dezimalzahlen, die man auch Gleitkommazahlen nennt. Der zulässige Wertebereich für das Argument von INT reicht für ganzzahlige Argumente von -32768 bis +32767 und für Gleitkomma-Argumente von +/-2.9E-39 bis +/-1.7E38. Also gilt für INT uneingeschränkt der gesamte Wertebereich der Zahlen.

INT liefert als Ergebnis die nächste ganze Zahl, die kleiner ist als das Argument.

```
PRINT INT(1.7*10↑37)
```

ergibt als Resultat 1.7E+37, was eine ganze Zahl ist.

ABS wandelt alle Zahlen in ihr positives Äquivalent um, d. h. das Ergebnis eines positiven oder negativen Arguments ist stets der Absolutbetrag, also die positive Zahl, der angegebenen Größe. Für ABS gilt ebenfalls der gesamte Wertebereich der Zahlen.

Eine Funktion, die ganze Zahlen in Gleitkommazahlen umwandelt, ist überflüssig, da der Interpreter automatisch jede Zahl in die Gleitkommadarstellung umwandelt und eine ganze Zahl ja auch nichts anderes ist als eine spezielle Gleitkommazahl.

Der Rechner verfügt ohnehin intern nur über Gleitkomma-Arithmetikroutinen, so daß eine gezielte Verwendung von ganzen Zahlen keine Geschwindigkeitsvorteile bringt. Einige Compiler bieten aber eine reine Integer-Arithmetikoption an, die eine beträchtliche Geschwindigkeitssteigerung für Anwendungen, die nur mit ganzen Zahlen auskommen können, darstellt.

Die INT-Funktion wird hauptsächlich verwendet, um Argumente für Funktionen oder Befehle, die nur ganzzahlige Werte als Parameter akzeptieren, in eine entsprechende Form zu bringen.

**Beispiele**

```
10 ON INT(A) GOSUB

10 IF -9=INT(-8.4) THEN PRINT"-9<-8.4"

10 IF 1.7*10E4=INT(1.7*10E4) THEN PRINT"RICHTIG"
```

Eine weitere Anwendung ist das gezielte Trennen des Vor- und Nachkommabereichs einer Zahl:

```
10 INPUT A
20 VK=INT(A)
30 NK=A-VK
40 PRINT"ZAHL VOR DEM KOMMA";VK
50 PRINT"ZAHL NACH DEM KOMMA";NK
60 END
```

Bei negativen Zahlen muß allerdings im Vorkommabereich eine 1 dazuaddiert werden, da die INT-Funktion immer die nächstkleinere Zahl als Ergebnis liefert, also für -6.3 den Wert -7. Aus diesem Grund fügen wir die Zeile 25 ein:

```
25 IF VK<0 THEN VK=VK+1
```

Eine weitere Möglichkeit, die Funktion INT einzusetzen, könnte sein, wenn man eine Zahl als Summe von Zehnerpotenzen aufschlüsseln möchte:

```
10 INPUT"GEBEN SIE EINE ZAHL EIN";A
20 FOR Z=1 TO 20
30 : NK=INT((A/10↑Z-INT(A/10↑Z))*10)
40 : PRINT"FAKTOR FUER 10 HOCH";Z;"=";NK
50 : A=A-NK*10↑(I-1)
60 NEXT Z
```

Das Programm funktioniert nur für positive ganze Zahlen mit maximal 20 Ziffern. Hin und wieder treten Rundungsfehler auf. Probieren Sie beispielsweise, die Zahl 253 einzugeben.

Analog können beliebige andere Faktorzerlegungen, etwa für Zweierpotenzen, für Primzahlfaktoren oder Modulo-Rechnungen einfach durchgeführt werden.

*Hinweis:* Eine Umwandlung von ganzen Zahlen ist ohne weiteres durch einfache Zuweisung möglich. Dabei treten keine Veränderungen des Zahlenwertes auf:

```
10 I%=32600
20 R=I%
30 PRINT I%,R
```

Die Zuweisung einer Gleitkommazahl an eine Integervariable hat die gleiche Wirkung wie die Anwendung der INT-Funktion auf eine Gleitkommazahl. Es werden die Nachkommastellen abgeschnitten. Die Gleitkommazahl muß allerdings dabei im zulässigen Bereich für ganze Zahlen liegen.

```
10 R1=3*10↑35+.0001
20 R2=3*10↑3+.9999
30 G2%=R2
40 PRINT G2%
50 G1%=R1
60 PRINT G1%
```

Die ABS-Funktion wird vor allem dazu eingesetzt, um Argumente und Parameter, die nur positive Werte annehmen dürfen, entsprechend umzuformen. Sie ist ferner sehr hilfreich bei der Vereinfachung von BASIC-Ausdrücken für die Ablaufkontrolle nach IF, ON, WHILE und UNTIL.

**Beispiel**

```
10 IF A<1 AND A>-1 THEN
```

kann mit ABS vereinfacht werden zu:

```
10 IF ABS(A)<1 THEN
```

## Trigonometrische Funktionen

Im Commodore-BASIC wird nur eine Grundausstattung an trigonometrischen Funktionen mitgeliefert: ATN, COS, SIN und TAN. Mit Hilfe einer Formelsammlung, die Sie ja vielleicht besitzen, lassen sich leicht die anderen benötigten Funktionen als benutzerdefinierte Funktionen aus den im ROM residenten

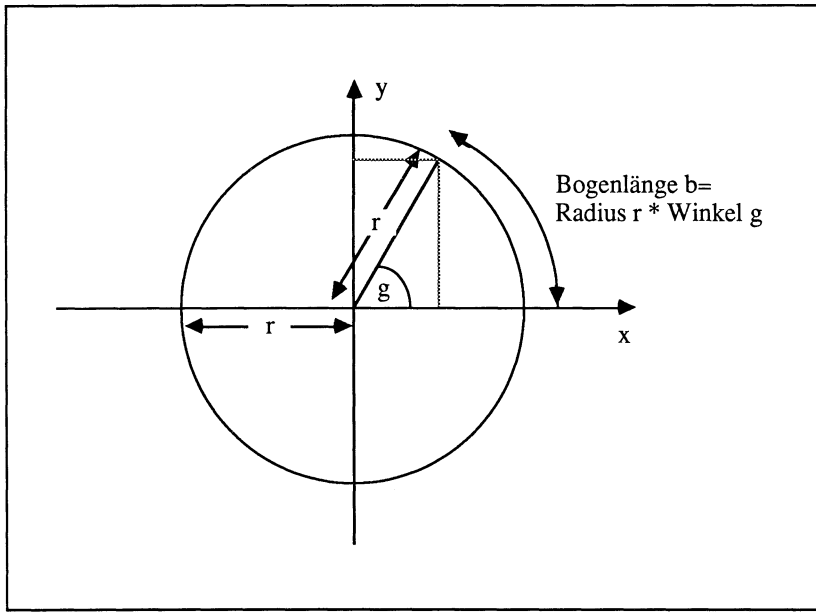



*Abb. 5.1: Bogenmaß eines Winkels*

zusammenmontieren. Unter dem Stichwort DEF FN in Kapitel 8 zeigen wir Ihnen viele Beispiele dafür.

Im Commodore-BASIC erfolgen alle Berechnungen von trigonometrischen Funktionen im Bogenmaß b.

Die Umrechnung in das Gradmaß g kann über die Formel

$$g=b*180/\pi$$

mit Hilfe des im Commodore-BASIC als ROM-Konstante verfügbaren Wertes, der dem π-Symbol direkt zugeordnet ist, bewerkstelligt werden. Bei Commodore-Rechnern, bei denen die Zahl π nicht speziell als Tastenbelegung vorgesehen ist, kann sie über ihren Tokenwert - CHR$(255) - auf den Bildschirm gedruckt oder auch einer Variablen zugeordnet werden. Geben Sie im Direktmodus ein:

```
PRINT"19000 PI=";CHR$(255)
```

Dann kann die nach dieser Eingabe auf dem Bildschirm erschienene Zeile mit dem π-Symbol über <RETURN> in das Programm übernommen werden.

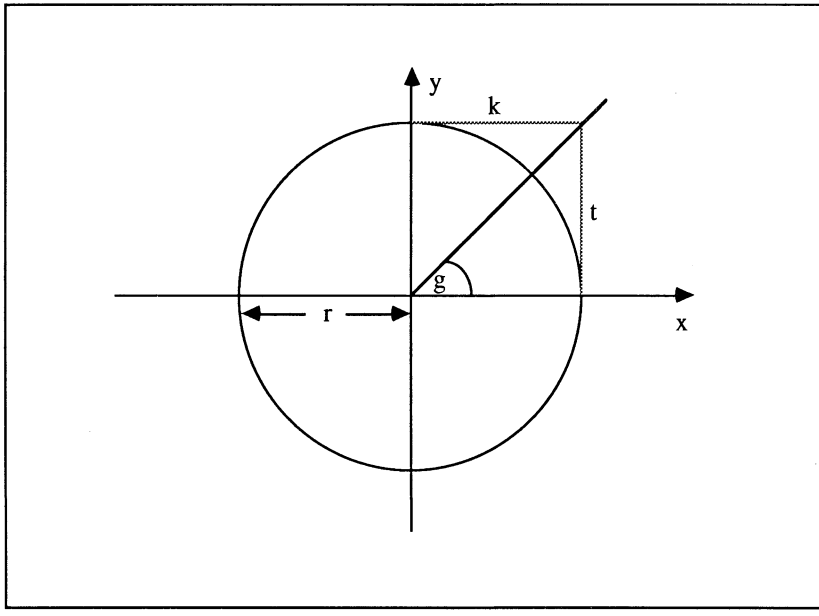



*Abb. 5.2: Tangens und Cotangens*

Betrachten Sie noch einmal die Abb. 5.1. Wir wissen, daß die folgenden mathematischen Beziehungen bestehen:

sin(g)=y/r und cos(g)=x/r

Für die Funktionen Tangens und Cotangens bestehen dann folgende Zusammenhänge:

tan(g)=t=x/y=sin(g)/cos(g)

und

cot(g)=k=y/x=cos(g)/sin(g)=1/tan(g)

Die Funktionen COS und SIN können zusammen mit dem CIRCLE-Befehl zu interessanten Grafiken führen und finden im übrigen im Zusammenhang mit naturwissenschaftlichen Programmen ebenso wie die anderen trigonometrischen Funktionen ihre Hauptanwendung.

Wir wollen eine kleine Tabelle mit Beispielen abgeleiteter trigonometrischer Funktionen aufstellen.

## Beispiele

```
61510 DEF FN SK(X)=1/COS(X)                    :REM SEKANS
61520 DEF FN SC(X)=1/SIN(X)                    :REM COSEKANS
61530 DEF FN CT(X)=1/TAN(X)                    :REM COTANGENS
61540 DEF FN AS(X)=1/ATN(X/SQR(1-X↑2))  :REM ARCUSSINUS X>0!
61550 DEF FN AC(X)=1.5708-ATN(X/SQR(1-X↑2))
                                               :REM ARCUSCOSINUS
61560 DEF FN AK(X)=ATN(X/SQR(X↑2+1))
                  +SGN(SGN(X)-1)*1.5708  :REM ARCUSSEKANS
61570 DEF FN CA(X)=ATN(X/SQR(X↑2-1))
                    +(SGN(X)-1)*1.5708  :REM ARCUSCOSEKANS
```

Zur Erhöhung der rechnerischen Genauigkeit ist es empfehlenswert, den Wertebereich für trigonometrische Berechnungen auf das Intervall von $-2*\pi$ bis $+2*\pi$ zu beschränken.

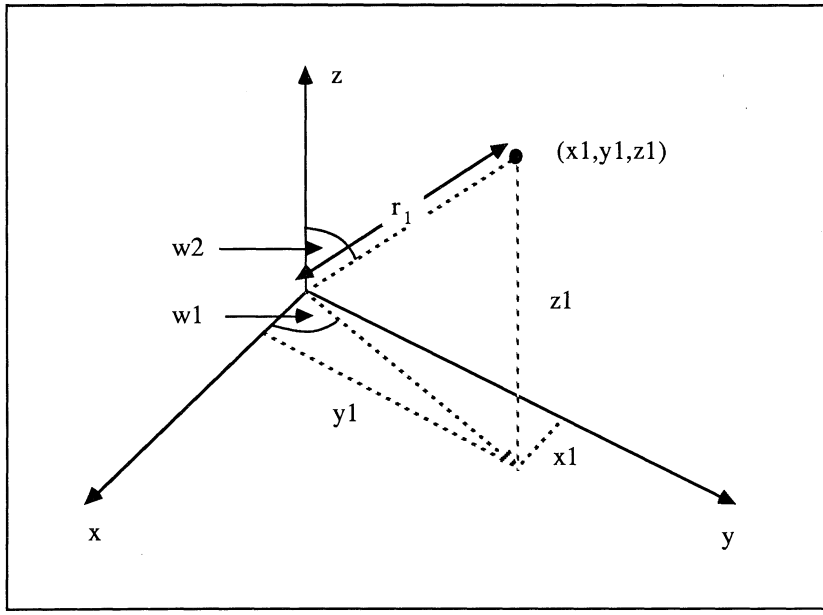



*Abb. 5.3: Darstellung von Polarkoordinaten*

## Beispiel

Ein weiteres interessantes Beispiel stellt die Umrechnung der Koordinatenangabe eines Punktes mit karthesischen Koordinaten (X1,Y1,Z1) in das Polarkoordinatensystem (W1,W2,R) dar. Falls Sie sich nicht mehr daran erinnern, was Polarkoordinaten sind, betrachten Sie die Abb. 5.3.

Das Beispielprogramm verlangt von Ihnen die Eingabe der Koordinaten X1, Y1 und Z1 und gibt die Polarkoordinaten W1, W2 und R aus:

```
51000 INPUT X1,Y1,Z1
51010 IF Z1<>0 THEN
      BEGIN
51020 : IF Z1<0          THEN GOSUB 54610
51030 : IF Z1>0          THEN GOSUB 54710
51040 BEND
51050 ELSE
52010 IF X1<>0 THEN
      BEGIN
52020 : IF X1<0 AND Y1=0 THEN GOSUB 54210
52030 :                  ELSE GOSUB 54310
52040 : IF X1>0          THEN GOSUB 54410
52050 BEND
52060 ELSE
53010 R=SQR(X1↑2+Y1↑2+Z1↑2)
53020 PRINT"W1=";W1;"W2=";W2;"R=";R
53030 END
54110 REM ************ X1=0 ************
54120 W1=SGN(Y1)*90
54130 RETURN
54210 REM ******** X1<0 AND Y1=0 *******
54220 W1=180
54230 RETURN
54310 REM ******* X1<0 AND Y1<>0 *******
54320 W1=SGN(Y1)*(180-ABS(ATN(Y1/X1))*180/π)
54330 RETURN
54410 REM ******* X1>0 AND Y1<>0 *******
54420 W1=SGN(Y1)*ABS(ATN(Y1/X1))*180/π
54430 RETURN
54510 REM *********** Z1=0 ************
54520 W2=90
54530 RETURN
54610 REM *********** Z1<0 ************
54620 W2=180+ATN(SQR(X1↑2+Y1↑2)/Z1)*180/π
```

```
54630 RETURN
54710 REM ********** Z1>0 ************
54720 W2=ATN(SQR(X1↑2+Y1↑2)/Z1)*180/π
54730 RETURN
```

### Andere arithmetische Funktionen

Die Exponentialfunktion EXP läßt Argumente von -1.7E+37 bis +88 zu. Für größere Argumente als +88 würde das Ergebnis der EXP-Funktion die obere darstellbare Zahlengrenze von +1.7E38 überschreiten, da die Funktion für positive Argumente stark anwächst. Sie nähert sich bei großen negativen Argumenten dem Wert 0, den sie auf unserem Rechner bereits bei EXP(-89) erreicht, in Wirklichkeit jedoch nie.

Die Zahl E ist mit dem Wert 2.7192818... gleichzusetzen, und der Ausgabewert der EXP-Funktion ist nichts anderes als:

```
EXP(X)=2.7192818↑X
```

Die Kurve der Exponentialfunktion sieht so aus, wie in Abb. 5.4 gezeigt.

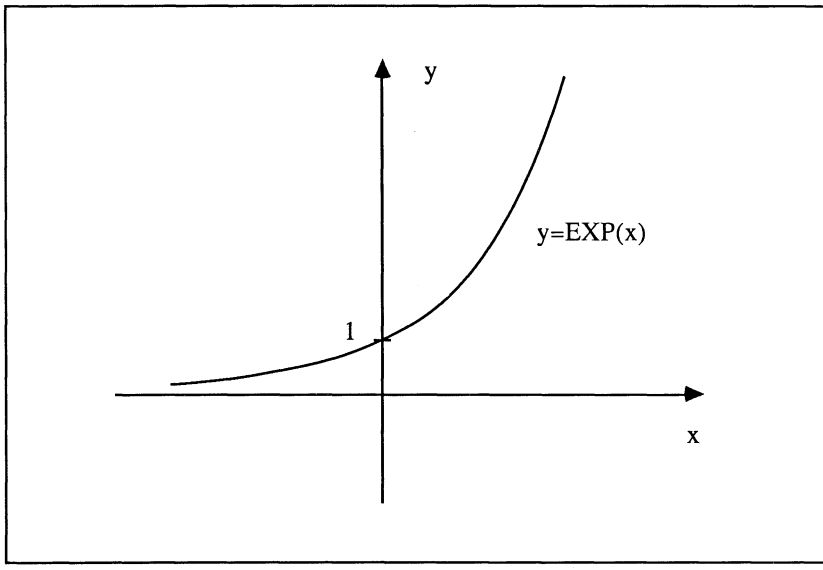



*Abb. 5.4: Exponentialfunktion*

Wie man sieht, schneidet sie die Y-Achse bei +1. Es gilt also:

```
EXP(0)=1
```

Die Umkehrfunktion zur Exponentialfunktion ist der natürliche Logarithmus, d. h. der Logarithmus zur Basis e. Er akzeptiert nur positive Argument aus dem Wertebereich von +2.9E-38 bis 1.7E38.

**Beispiele**

```
10 PRINT LOG(EXP(X))

10 PRINT EXP(LOG(X))

10 DEF FN LG(X)=LOG(X)/LOG(10) :REM LOGARITHMUS ZUR
BASIS 10
```

Die ersten beiden Beispiele sollten für gleiches X auch die gleichen Ergebnisse liefern, da die Ausdrücke äquivalent sind. Wenn Sie jedoch z. B. einmal für X den Wert .001 eingeben, sehen Sie folgendes:

`LOG(EXP(.001))` ergibt `1.00000016E-03`

und

`EXP(LOG(.001))` ergibt `1E-03`

Der Grund für diese verschiedenen Ergebnisse liegt darin, daß bei sehr kleinen und sehr großen Zahlen Rundungsfehler auftreten, die sich bei internen Zwischenergebnissen während der Berechnung des Funktionswertes ergeben. Diesen Effekt werden Sie bei allen Umkehrfunktionen feststellen können.

Wenn Sie also genau rechnen wollen, müssen Sie diese Abweichungen abfangen. Angenommen Sie wollen die Variable X auf 0 abfragen, wissen aber, daß durch Rundungsfehler Abweichungen im Bereich von 0.9E-7 dort auftreten, wo eigentlich 0 als Ergebnis erscheinen müßte, dann können Sie einfach die INT-Funktion zur Bereinigung verwenden:

```
10 IF INT(X*10E-7)=0 THEN PRINT"RICHTIG"
```

Bei diesem kleinen Trick muß man durch Ausprobieren natürlich erst herausfinden, wie groß die Fehlerspanne durch die Rundungsfehler ist, damit man sie sinnvoll korrigieren kann.

Aus den transzendenten Funktionen EXP und LOG lassen sich eine Reihe weiterer transzendenter Funktionen ableiten:

```
DEF FNSH(X)=(EXP(X)-EXP(-X))/2              :REM HYPERBELSINUS
DEF FNHC(X)=(EXP(X)+EXP(-X))/2              :REM HYPERBELCOSINUS
DEF FNTH(X)=EXP(-X)/(EXP(X)+EXP(-X)*2+1):REM HYPERBELTANGENS
DEF FNHS(X)=2/EXP(X)+EXP(-X)                :REM HYPERBELSEKANS
DEF FNHK(X)=2/EXP(X)-EXP(-X)                :REM HYPERBELCOSEKANS
DEF FNHT(X)=EXP(-X)/(EXP(X)-EXP(-X))*2+1:REM HYPERBELCOTANGENS
DEF FNHA(X)=LOG(X+SQR(X↑2+1))               :REM HYPERBELAREASINUS
DEF FNAH(X)=LOG(X+SQR(X↑2-1))           :REM HYPERBELAREACOSINUS
DEF FNYA(X)=(LOG(1+X)/(1-X))/2          :REM HYPERBELAREATANGENS
DEF FNYS(X)=LOG((SQR(-X↑2+1)+1)/X)      :REM HYPERBELAREASEKANS
DEF FNYK(X)=LOG((SGN(X)*SQR(X↑2+1)+1)/X):REM
HYPERBELAREACOSEKANS
DEF FNYT(X)=LOG((X+1)/(X-1))/2          :REMHYPERBELAREACOTANGENS
```

Für die Defininition des Hyperbelareacosekans haben wir die Funktion SGN verwendet. Diese analysiert das Vorzeichen einer Gleitkommazahl oder liefert das Ergebnis 0, wenn das Argument 0 ist. Für negative Zahlen ergibt die SGN-Funktion -1 und für positive Zahlen +1.

Man könnte die SGN-Funktion also als ein Element für eine dreiwertige Logik mit

0 für falsch
-1 für wahr
+1 für unentschieden oder nicht definiert

betrachten. Dabei wird eine solche Betrachtungsweise vom Commodore-BASIC nicht direkt unterstützt.

```
100 A=1
110 IF SGN(A)=1 THEN PRINT"A IST NOCH NICHT DEFINIERT"
120 A=SGN(A=A)
130 IF A=-1 THEN PRINT"A IST WAHR"
140 A=SGN(A<A)
150 IF A=0 THEN PRINT"A IST UNWAHR"
160 END
```

Die Wurzel-Funktion SQR als Umkehrfunktion zum Potenzieren mit dem Operator ↑ liefert uns auch ein paar schöne Beispiele für Rundungsfehler. Eigentlich ist die Wurzelfunktion ein Spezialfall der Potenzfunktion, nämlich:

```
SQR(X)=X↑.5
```

oder auch als

```
SQR(X)=X↑(1/2)
```

bekannt. Die Berechnungen mit SQR sind jedoch vorzuziehen, da sie in der Ausführung schneller sind.

**Beispiele**

```
10 IF SQR(9↑2)=(SQR(9)↑2) THEN
   PRINT"MUESSTE EIGENTLICH GLEICH SEIN"

10 A=SQR(.999999999↑2)
20 PRINT"A IST";A
30 IF A=1 THEN PRINT"A IST 1":GOTO 50
40 PRINT"A IST NICHT 1"
50 END
```

Beim zweiten Beispiel nutzt es auch nichts, wenn man die Funktion INT einsetzt:

```
30 IF INT(A)=1 THEN PRINT"A IST 1":GOTO 50
```

Woran liegt es nun, daß A=1 ist und doch nicht gleich 1 ist? Intern hat der Rechner für die Variable A noch mehr Stellen gespeichert als bei der Ausgabe mit PRINT berücksichtigt werden. Es muß also ein winziger Unterschied zwischen A und 1 bestehen. Wie klein diese Differenz ist, können Sie feststellen, wenn Sie die folgende Zeile in Ihrem Programm ändern:

```
30 IF ABS(1-A)<1E-38 THEN PRINT"A IST 1":GOTO 50
```

Die Funktion SQR erlaubt keine negativen Argumente, da das Rechnen mit komplexen Zahlen vom Commodore-BASIC leider nicht direkt unterstützt wird. Sie können natürlich komplexe Zahlen in Ihren Programmen verwenden, wenn Sie diese in ihren Realteil und Imaginärteil aufsplitten und alle Rechnungen entsprechend getrennt durchführen.

Eine besondere Bedeutung kommt der Quadratwurzel-Funktion SQR auf dem Gebiet der Statistik zu, wo sie für die Berechnung wichtiger elementarer Größen, wie z. B. der Streuung, benötigt wird.

**Beispiel**

```
100 SU=0:AB=0
110 DIM MW(5)                :REM MESSWERTE
```

```
120 MW(1)=5:MW(2)=4:MW(3)=5:MW(4)=5:MW(5)=6
130 FOR I=1 TO 5
140 : SU=SU+MW(I):REM SUMME DER MESSWERTE
150 NEXT I
160 SU=SU/5                  :REM MITTELWERT
170 PRINT"MITTELWERT=";SU
180 FOR I=1 TO 5
190 : IF SU<>MW(I) THEN AB=AB+(SU-MW(I))↑2
200 NEXT I  :REM QUADRATE DER ABWEICHUNGEN
210 PRINT"SUMME DER QUADRATE DER ABWEICHUNGEN="
    ;AB
220 SA=SQR(AB/5)  :REM STANDARDABWEICHUNG
230 PRINT"STANDARDABWEICHUNG=";SA
240 END
```

Das Beispiel wollen wir jetzt etwas verallgemeinern und für die statistischen Berechnungen eigene Benutzerfunktionen verwenden. Unser Programm kann dann 100 Meßwerte lesen, die in DATA-Zeilen angegeben sind und die Sie beliebig verändern können.

```
100 DF$=CHR$(147)
105 PRINT DF$
110 A=0
120 DIM W(101)
130 DEF FNW(N)=W(N)+W(N+1)+W(N+2)+W(N+3)
    +W(N+4)+W(N+5)+W(N+6)+W(N+7)+W(N+8)
    +W(N+9)
140 DEF FNS(N)=FNW(N)+FNW(N+10)+FNW(N+20)
    +FNW(N+30)+FNW(N+40)+FNW(N+50)+FNW(N+60)
    +FNW(N+70)+FNW(N+80)+FNW(N+90)
150 DEF FNM(W)=FNS(N)/W
160 DEF FNQ(I)=(W(I)-FNM(W))↑2
170 PRINT"STATISTIK BIS ZU 100 WERTEN"
180 INPUT"WIE VIELE WERTE WOLLEN SIE LESEN";W
190 IF W>100 THEN W=100
200 IF W<1 THEN GOTO 180
210 FOR I=1 TO W
220 : READ W(I)
230 NEXT I
240 FOR I=1 TO W
250 : A=A+FNQ(I)
260 NEXT I
270 PRINT"ANZAHL WERTE:";W
```

```
280 PRINT"MITTELWERT:";FNM(W)
290 PRINT"STANDARDABWEICHUNG:";SQR(A/W)
300 DATA 0,0,0,0,0,0,0,0,0,0
310 DATA 0,0,0,0,0,0,0,0,0,0
320 DATA 0,0,0,0,0,0,0,0,0,0
330 DATA 0,0,0,0,0,0,0,0,0,0
340 DATA 0,0,0,0,0,0,0,0,0,0
350 DATA 0,0,0,0,0,0,0,0,0,0
360 DATA 0,0,0,0,0,0,0,0,0,0
370 DATA 0,0,0,0,0,0,0,0,0,0
380 DATA 0,0,0,0,0,0,0,0,0,0
390 DATA 0,0,0,0,0,0,0,0,0,0
```

Bei Commodore-Rechnern, die nicht solche langen Programmzeilen wie etwa Zeile 140 erlauben, machen Sie einfach die Funktion FNS um einen Summanden kürzer und geben dann nur 90 Meßwerte ein.

Dieses Beispiel läßt außerdem Grenzen beim verschachtelten Aufbau von Benutzerfunktionen erkennen. Wenn es dem Rechner zuviel wird, meldet er:

```
?FORMULA TOO COMPLEX ERROR
```

Daran ist auch unser Plan gescheitert, der ursprünglich 400 Meßwerte für das Programm vorsah.

Erfassen Sie zunächst die DATA-Zeilen so, wie wir es hier abgebildet haben: Alle Werte sind 0. Speichern Sie das Programm in dieser Version vor der ersten Verwendung ab, damit Sie stets für die Eintragung von neuen Datenwerten einen sauberen Ausgangszustand herstellen können, indem Sie diese Version laden.

Die RND-Funktion gehört in den Bereich der Statistik und der Simulation. Diese Zufallsfunktion ermöglicht es, den an und für sich nach strengen Regeln im stets vorhersehbaren Rahmen ablaufenden Vorgängen in unserem Commodore-Rechner einen Hauch von Unbestimmtheit und Unberechenbarkeit zu geben. Diese Funktion ist deshalb insbesondere bei der Programmierung von Spielen zu finden.

Die RND-Funktion erlaubt als Argumente positive und negative numerische Ausdrücke und 0, aber die Wirkung dieser drei Gruppen von möglichen Argumentwerten ist völlig verschieden. Der RND-Funktion liegt ein Algorithmus zur Erzeugung einer scheinbar endlosen Reihe von Zahlen zugrunde, ausgehend von einem bestimmten Startwert, die nach statistischen Kriterien hinreichend zufälligen Charakter haben.

Der Aufruf von RND mit einem positiven Argument veranlaßt den Rechner, eine weitere Zahl der Zufallsreihe auszugeben.

Das Aufrufen von RND mit einem negativen Argument führt in reproduzierbarer Weise zur Bildung einer neuen Zufallsreihe, deren Startwert eindeutig durch den Betrag des Arguments festgelegt ist.

Das Aufrufen von RND mit dem Argument 0 führt unter Verwendung der internen Uhrzeit des Rechners zur Berechnung des Startwertes einer neuen Zufallszahlenfolge, was diese naturgemäß praktisch unreproduzierbar und kaum vorhersagbar werden läßt. Die Rechneruhr wird nämlich durch jedes RESET, z. B. auch beim Einschalten, auf 0 gesetzt und beginnt dann jedesmal von vorn. Sie kann außerdem gezielt aus einem BASIC-Programm heraus oder im Direktmodus manipuliert werden.

RND(0) ist aber nicht vollkommem wirkungsgleich mit RND(-TI), da der interne Algorithmus beide Argumente unterschiedlich behandelt. TI ist ja bekanntlich die Systemvariable, mit der Sie die Rechneruhr abfragen können. Der Unterschied läßt sich leicht durch systematische Versuche mit RND(0) und RND(-TI) als Start für verschiedene Zufallszahlenfolgen ermitteln. Dabei liefert RND(0) wirklich zufällige Zahlen für in größeren Abständen aufeinanderfolgende RND-Aufrufe zur Ausgabe von Zufallszahlen, wie sie etwa in Abhängigkeit von Benutzereingaben zu erwarten sind.

Für sehr schnell aufeinanderfolgende Abrufe neuer Zufallszahlen im Rahmen von Berechnungen in kompilierten BASIC-Programmen, wobei die Aufruffrequenz deutlich unter der Zeittaktfrequenz der internen Uhr liegt (etwa 1/120 Sekunde beim C128) ist von der Verwendung von RND(0) abzuraten, weil der Algorithmus, der dann auch vom Zeittakt gesteuert wird, deutlich schlechtere Zufallszahlen liefert. In diesem Fall wird die Verwendung von RND(-RND(-TI)) oder RND(-RND(0)) als Start für wirklich zufällig erscheinende Zahlenfolgen empfohlen.

Da die RND-Funktion nur Zufallszahlen zwischen 0 und 1 als Werte liefert, müssen diese gezielt mit einem Faktor multipliziert werden, um Zahlen aus einem bestimmten Wertebereich zu erhalten. So ist es z. B. möglich, den Rechner als Denkanstoß für das Ausfüllen des Lottozettels einzusetzen, wenn man den Wertebereich der RND-Funktion wie folgt auf den Bereich von 1 bis 49 einstellt:

```
LO=INT(RND(0)*49)+1
```

### Umwandlung von Zahlen in Zeichen

Die Funktionen HEX$, STR$ und CHR$ liegen im Grenzgebiet zwischen numerischen und Zeichenkettenfunktionen, da ihre Argumente Zahlen sind und

ihre Ergebnisse Zeichen bzw. Zeichenketten. Sie stellen also Umwandlungsfunktionen dar.

Den größten Argumentbereich hat die STR$-Funktion, die alle numerischen Ausdrücke zuläßt und als Zeichenkette ausgibt.

### Beispiele

```
10 PRINT STR$(1.7E38)+"1"
```

Richtig, die Ausgabe 1.7E+381 ist jetzt aber eine Zeichenkette.

```
10 PRINT STR$(STR$(1.7E38)+"1")
```

Falsch, da 1.7E+381 schon eine Zeichenkette ist und STR$ einen numerischen Ausdruck erwartet.

Das folgende Beispielprogramm rechnet eine Dezimalzahl in eine Binärzahl um:

```
100 PRINT CHR$(147)
110 DEF FNP(P)=INT(DZ(P+1)/2↑P)*2↑P
120 DEF FNZP(P)=-10↑P*(DZ(P+1)>2↑P-1)
130 REM*********************************
140 REM*    DEZIMALE TESTZAHL           *
150 REM*********************************
160 DZ=1*2↑8+1*2↑7+1*2↑6+1*2↑5+1*2↑4
          +1*2↑3+1*2↑2+0*2↑1+1*2↑0
170 REM* BLENDEN SIE TESTWEISE VER-     *
180 REM* SCHIEDENE ZWEIERPOTENZEN DURCH*
190 REM* MULTIPLIKATION MIT 0 STATT 1   *
200 REM* AUS                           *
210 DZ(9)=DZ:REM UEBERGABE DEZIMALZAHL
220 IF LEN(STR$(DZ))>9
       THEN PRINT"ZAHL ZU GROSS":STOP
230 REM* DIE STR$-FUNKTION IST BEI      *
240 REM* LAENGENABFRAGEN PRAKTISCH.     *
250 REM* DZ>10↑8 GINGE NATUERLICH AUCH *
260 REM*********************************
270 REM* UMWANDLUNG DEZIMAL IN BINAER   *
280 REM*********************************
290 FOR P=8 TO 0 STEP -1
300 : DZ(P)=DZ(P+1)-FNP(P)
310 : BZ=BZ+FNZP(P)
320 NEXT P
```

```
330 PRINT BZ;"DUALZAHL DEZIMAL CODIERT"
340 REM*********************************
350 REM* UMWANDLUNG IN ZEICHENKETTE    *
360 REM*********************************
370 PRINT STR$(BZ);"DUALZAHL ALS";
380 PRINT" ZEICHENKETTE"
390 PRINT STR$(BZ);"*10=";STR$(BZ)+"0"
400 PRINT"DIE STR$-FUNKTION IST ";
410 PRINT"NUETZLICH BEI DER SIMULATION"
420 PRINT"VON RECHENOPERATIONEN FUER ";
430 PRINT"DUALZAHLEN"
440 END
```

Die CHR$-Funktion hat eine ganz andere Aufgabe als die STR$-Funktion. Sie ordnet Zahlen aus dem Wertebereich 0 bis 255, also Byte-Werten, die Zeichen zu, die an der entsprechenden Position in der Tabelle der darstellbaren Zeichen stehen (siehe Anhang G).

PRINT CHR$(255) führt beispielsweise zur Ausgabe des letzten Zeichens in der Tabelle, dem Symbol π.

Einige Positionen der Zeichentabelle enthalten Codes, denen kein druckbares Zeichen zugeordnet ist. Dies führt auf dem Bildschirm und auf dem Drucker zur scheinbaren Ausgabe von Leerzeichen. Tatsächlich wird aber gar kein Zeichen ausgedruckt. Ein echtes Leerzeichen wird allein durch die Codes 32 und 160 erzeugt:

```
PRINT CHR$(32) oder PRINT CHR$(160)
```

Eine Reihe von Steuerzeichen haben eine direkte Wirkung auf die gerade laufende Bildschirmausgabe, wenn sie in einer zu druckenden Zeichenkette verwendet werden.

Um alle Codes zu sehen, geben Sie einmal die Zeile

```
FOR I=1 TO 255:PRINT CHR$(I);:NEXT I
```

ein.

In der alphabetischen Befehlsübersicht in Kapitel 8 finden Sie eine vollständige Liste aller mit CHR$ ansprechbaren Steuerzeichen und Halbgrafikzeichen. Damit Ihre Programme übersichtlicher und kompakter werden, sollten Sie es vermeiden, die Steuerzeichen durch Betätigen der entsprechenden Taste auf der Tastatur in die gerade erstellte Zeichenkette einzufügen, was auch möglich

wäre. Das führt im Druckbild zu unleserlichen Programmlistings mit unschönen mehrfach aneinandergereihten gleichartigen Steuerzeichen. Viel eleganter ist es, diese Steuerzeichen Variablen mit sprechenden Namen zuzuweisen. Wenn Sie die in der Tabelle unter dem Stichwort CHR$ vorgeschlagenen Namen verwenden, bleiben Ihre Programme stets miteinander verträglich.

Wir wollen nun in einem Beispielprogramm die Großbuchstaben des Alphabets mit Hilfe der CHR$-Funktion durch Blockgrafik darstellen und auf dem Bildschirm zeigen.

```
1000 COLOR 4,1
1010 DF$=CHR$(147)                  :REM LOESCHEN
1020 ZR$=CHR$(18)                 :REM INVERS EIN
1030 ZN$=CHR$(146)                :REM INVERS AUS
1040 DIM BH$(16) : REM HORIZONTALE BALKENELEM.
1050 BH$(1)=ZR$+CHR$(32)+ZN$     :REM VOLLBLOCK
1060 BH$(5)=CHR$(162)      :REM HALBBLOCK UNTEN
1070 BH$(13)=ZR$+BH$(5)+ZN$:REM HALBBLOCK OBEN
1080 DIM HB$(8)                     :REM DREIECKE
1090 HB$(5)=CHR$(169)   :REM DREIECK OBEN LINKS
1100 HB$(6)=CHR$(127)  :REM DREIECK OBEN RECHTS
1110 HB$(7)=ZR$+HB$(5)+ZN$   :REM DREIECK U. R.
1120 HB$(8)=ZR$+HB$(6)+ZN$   :REM DREIECK U. L.
1130 DIM VB$(16)            :REM VIERTELBLOECKE
1140 VB$(2)=CHR$(190)   :REM VIERTEL OBEN LINKS
1150 VB$(5)=CHR$(187)  :REM VIERTEL UNTEN LINKS
1160 SP=40              :REM SPALTEN BILDSCHIRM
1170 REM ************************************
1180 REM * ZEILENRASTER FUER GROSSBUCHSTABEN *
1190 REM * MIT BLOCKGRAFIKZEICHEN            *
1200 REM * 5 ODER 6 GROSSZEILEN A 4 NORMAL-  *
1210 REM * ZEILEN. EIN GROSSBUCHSTABE BE-    *
1220 REM * NOETIGT 4 SPALTEN OHNE LEERZEILE  *
1230 REM * ODER 5 MIT. AUF EINEM 40-SPALTEN- *
1240 REM * SCHIRM GIBT ES ALSO 50 ODER 60    *
1250 REM * POSITIONEN FUER GROSSBUCHSTABEN   *
1260 REM ************************************
1270 DIM AB$(32,4,4)   :REM MATRIX FUER GROSSB.
1280 DIM GB$(6,SP/4,4,4)         :REM POSITONEN
1290 DIM GS$(6*SP/4,4,4)       :REM EINGABEFELD
1300 REM ************************************
1310 REM *     ANFANGSBILDSCHIRM             *
1320 REM ************************************
1330 PRINT DF$:PRINT:PRINT
 :PRINT" BITTE WARTEN, ICH LERNE DAS ALPHABET"
```

```
1340 REM ************************************
1350 REM *       BUCHSTABENAUFBAU           *
1360 REM ************************************
1370 REM ******** BUCHSTABE A ***************
1380 AB$(1,1,1)=HB$(7)
1390 AB$(1,2,1)=BH$(1)
1400 AB$(1,3,1)=BH$(1)
1410 AB$(1,4,1)=BH$(1)
1420 AB$(1,1,2)=BH$(1)
1430 AB$(1,3,2)=BH$(1)
1440 AB$(1,1,3)=HB$(8)
1450 AB$(1,2,3)=BH$(1)
1460 AB$(1,3,3)=BH$(1)
1470 AB$(1,4,3)=BH$(1)
1480 REM ******** BUCHSTABE B ***************
1490 AB$(2,1,1)=BH$(1)
1500 AB$(2,2,1)=BH$(1)
1510 AB$(2,3,1)=BH$(1)
1520 AB$(2,4,1)=BH$(1)
1530 AB$(2,1,2)=BH$(1)
1540 AB$(2,2,2)=BH$(1)
1550 AB$(2,3,2)=BH$(13)
1560 AB$(2,4,2)=BH$(1)
1570 AB$(2,1,3)=HB$(8)
1580 AB$(2,2,3)=HB$(5)
1590 AB$(2,3,3)=HB$(8)
1600 AB$(2,4,3)=HB$(5)
1610 REM ******** BUCHSTABE C ***************
1620 AB$(3,1,1)=HB$(7)
1630 AB$(3,2,1)=BH$(1)
1640 AB$(3,3,1)=BH$(1)
1650 AB$(3,4,1)=HB$(6)
1660 AB$(3,1,2)=BH$(1)
1670 AB$(3,4,2)=BH$(1)
1680 AB$(3,1,3)=BH$(1)
1690 AB$(3,4,3)=BH$(1)
1700 REM ******** BUCHSTABE D ***************
1710 AB$(4,1,1)=BH$(1)
1720 AB$(4,2,1)=BH$(1)
1730 AB$(4,3,1)=BH$(1)
1740 AB$(4,4,1)=BH$(1)
1750 AB$(4,1,2)=BH$(1)
1760 AB$(4,4,2)=BH$(1)
1770 AB$(4,1,3)=HB$(8)
1780 AB$(4,2,3)=BH$(1)
```

```
1790 AB$(4,3,3)=BH$(1)
1800 AB$(4,4,3)=HB$(5)
1810 REM ******** BUCHSTABE E ****************
1820 AB$(5,1,1)=BH$(1)
1830 AB$(5,2,1)=BH$(1)
1840 AB$(5,3,1)=BH$(1)
1850 AB$(5,4,1)=BH$(1)
1860 AB$(5,1,2)=BH$(1)
1870 AB$(5,2,2)=BH$(5)
1880 AB$(5,3,2)=BH$(13)
1890 AB$(5,4,2)=BH$(1)
1900 AB$(5,1,3)=BH$(1)
1910 AB$(5,2,3)=VB$(5)
1920 AB$(5,3,3)=VB$(2)
1930 AB$(5,4,3)=BH$(1)
1940 REM ******** BUCHSTABE F ****************
1950 AB$(6,1,1)=BH$(1)
1960 AB$(6,2,1)=BH$(1)
1970 AB$(6,3,1)=BH$(1)
1980 AB$(6,4,1)=BH$(1)
1990 AB$(6,1,2)=BH$(1)
2000 AB$(6,2,2)=BH$(5)
2010 AB$(6,3,2)=BH$(13)
2020 AB$(6,1,3)=BH$(1)
2030 AB$(6,2,3)=VB$(5)
2040 AB$(6,3,3)=VB$(2)
2050 REM ******** BUCHSTABE G ****************
2060 AB$(7,1,1)=HB$(7)
2070 AB$(7,2,1)=BH$(1)
2080 AB$(7,3,1)=BH$(1)
2090 AB$(7,4,1)=HB$(6)
2100 AB$(7,1,2)=BH$(1)
2110 AB$(7,4,2)=BH$(1)
2120 AB$(7,1,3)=BH$(1)
2130 AB$(7,3,3)=BH$(1)
2140 AB$(7,4,3)=BH$(1)
2150 REM ******** BUCHSTABE H ****************
2160 AB$(8,1,1)=BH$(1)
2170 AB$(8,2,1)=BH$(1)
2180 AB$(8,3,1)=BH$(1)
2190 AB$(8,4,1)=BH$(1)
2200 AB$(8,2,2)=BH$(5)
2210 AB$(8,3,2)=BH$(13)
2220 AB$(8,1,3)=BH$(1)
2230 AB$(8,2,3)=BH$(1)
```

```
2240 AB$(8,3,3)=BH$(1)
2250 AB$(8,4,3)=BH$(1)
2260 REM ******** BUCHSTABE I ****************
2270 AB$(9,1,2)=BH$(1)
2280 AB$(9,2,2)=BH$(1)
2290 AB$(9,3,1)=BH$(1)
2300 AB$(9,4,2)=BH$(1)
2310 REM ******** BUCHSTABE J ****************
2320 AB$(10,3,1)=BH$(5)
2330 AB$(10,4,1)=HB$(6)
2340 AB$(10,4,2)=BH$(1)
2350 AB$(10,1,3)=BH$(1)
2360 AB$(10,2,3)=BH$(1)
2370 AB$(10,3,3)=BH$(1)
2380 AB$(10,4,3)=HB$(5)
2390 REM ******** BUCHSTABE K ****************
2400 AB$(11,1,1)=BH$(1)
2410 AB$(11,2,1)=BH$(1)
2420 AB$(11,3,1)=BH$(1)
2430 AB$(11,4,1)=BH$(1)
2440 AB$(11,2,2)=HB$(7)
2450 AB$(11,3,2)=HB$(6)
2460 AB$(11,1,3)=BH$(1)
2470 AB$(11,2,3)=HB$(5)
2480 AB$(11,3,3)=HB$(8)
2490 AB$(11,4,3)=BH$(1)
2500 REM ******** BUCHSTABE L ****************
2510 AB$(12,1,1)=BH$(1)
2520 AB$(12,2,1)=BH$(1)
2530 AB$(12,3,1)=BH$(1)
2540 AB$(12,4,1)=BH$(1)
2550 AB$(12,4,2)=BH$(1)
2560 AB$(12,4,3)=BH$(1)
2570 REM ******** BUCHSTABE M ****************
2580 AB$(13,1,1)=BH$(1)
2590 AB$(13,2,1)=BH$(1)
2600 AB$(13,3,1)=BH$(1)
2610 AB$(13,4,1)=BH$(1)
2620 AB$(13,1,2)=HB$(8)
2630 AB$(13,2,2)=BH$(1)
2640 AB$(13,3,2)=HB$(6)
2650 AB$(13,1,3)=HB$(7)
2660 AB$(13,2,3)=BH$(1)
2670 AB$(13,3,3)=HB$(5)
2680 AB$(13,1,4)=BH$(1)
```

```
2690 AB$(13,2,4)=BH$(1)
2700 AB$(13,3,4)=BH$(1)
2710 AB$(13,4,4)=BH$(1)
2720 REM ******** BUCHSTABE N ****************
2730 AB$(14,1,1)=BH$(1)
2740 AB$(14,2,1)=BH$(1)
2750 AB$(14,3,1)=BH$(1)
2760 AB$(14,4,1)=BH$(1)
2770 AB$(14,1,2)=HB$(8)
2780 AB$(14,2,2)=BH$(1)
2790 AB$(14,3,2)=HB$(6)
2800 AB$(14,2,3)=HB$(8)
2810 AB$(14,3,3)=BH$(1)
2820 AB$(14,4,3)=HB$(6)
2830 AB$(14,1,4)=BH$(1)
2840 AB$(14,2,4)=BH$(1)
2850 AB$(14,3,4)=BH$(1)
2860 AB$(14,4,4)=BH$(1)
2870 REM ******** BUCHSTABE O ****************
2880 AB$(15,1,1)=HB$(7)
2890 AB$(15,2,1)=BH$(1)
2900 AB$(15,3,1)=BH$(1)
2910 AB$(15,4,1)=HB$(6)
2920 AB$(15,1,2)=BH$(1)
2930 AB$(15,4,2)=BH$(1)
2940 AB$(15,1,3)=HB$(8)
2950 AB$(15,2,3)=BH$(1)
2960 AB$(15,3,3)=BH$(1)
2970 AB$(15,4,3)=HB$(5)
2980 REM ******** BUCHSTABE P ****************
2990 AB$(16,1,1)=BH$(1)
3000 AB$(16,2,1)=BH$(1)
3010 AB$(16,3,1)=BH$(1)
3020 AB$(16,4,1)=BH$(1)
3030 AB$(16,1,2)=BH$(1)
3040 AB$(16,3,2)=BH$(1)
3050 AB$(16,1,3)=HB$(8)
3060 AB$(16,2,3)=BH$(1)
3070 AB$(16,3,3)=HB$(5)
3080 REM ******** BUCHSTABE Q ****************
3090 AB$(17,1,1)=HB$(7)
3100 AB$(17,2,1)=BH$(1)
3110 AB$(17,3,1)=BH$(1)
3120 AB$(17,4,1)=HB$(6)
3130 AB$(17,1,2)=BH$(1)
```

```
3140 AB$(17,4,2)=BH$(1)
3150 AB$(17,1,3)=HB$(8)
3160 AB$(17,2,3)=BH$(1)
3170 AB$(17,3,3)=BH$(1)
3180 AB$(17,4,3)=BH$(1)
3190 AB$(17,4,4)=HB$(5)
3200 REM ******** BUCHSTABE R ****************
3210 AB$(18,1,1)=BH$(1)
3220 AB$(18,2,1)=BH$(1)
3230 AB$(18,3,1)=BH$(1)
3240 AB$(18,4,1)=BH$(1)
3250 AB$(18,1,2)=BH$(1)
3260 AB$(18,3,2)=BH$(1)
3270 AB$(18,4,2)=HB$(6)
3280 AB$(18,1,3)=HB$(8)
3290 AB$(18,2,3)=BH$(1)
3300 AB$(18,3,3)=HB$(5)
3310 AB$(18,4,3)=HB$(8)
3320 REM ******** BUCHSTABE S ****************
3330 AB$(19,1,1)=HB$(7)
3340 AB$(19,2,1)=BH$(1)
3350 AB$(19,3,1)=BH$(13)
3360 AB$(19,4,1)=BH$(1)
3370 AB$(19,1,2)=BH$(1)
3380 AB$(19,2,2)=BH$(5)
3390 AB$(19,3,2)=BH$(13)
3400 AB$(19,4,2)=BH$(1)
3410 AB$(19,1,3)=BH$(1)
3420 AB$(19,2,3)=BH$(5)
3430 AB$(19,3,3)=BH$(1)
3440 AB$(19,4,3)=HB$(5)
3450 REM ******** BUCHSTABE T ****************
3460 AB$(20,1,1)=BH$(1)
3470 AB$(20,1,2)=BH$(1)
3480 AB$(20,2,2)=BH$(1)
3490 AB$(20,3,2)=BH$(1)
3500 AB$(20,4,2)=BH$(1)
3510 AB$(20,1,3)=BH$(1)
3520 REM ******** BUCHSTABE U ****************
3530 AB$(21,1,1)=BH$(1)
3540 AB$(21,2,1)=BH$(1)
3550 AB$(21,3,1)=BH$(1)
3560 AB$(21,4,1)=HB$(6)
3570 AB$(21,4,2)=BH$(1)
3580 AB$(21,1,3)=BH$(1)
```

```
3590 AB$(21,2,3)=BH$(1)
3600 AB$(21,3,3)=BH$(1)
3610 AB$(21,4,3)=BH$(1)
3620 REM ******** BUCHSTABE V ****************
3630 AB$(22,1,1)=BH$(1)
3640 AB$(22,2,1)=BH$(1)
3650 AB$(22,3,1)=HB$(6)
3660 AB$(22,3,2)=HB$(8)
3670 AB$(22,4,2)=HB$(6)
3680 AB$(22,3,3)=HB$(7)
3690 AB$(22,4,3)=HB$(5)
3700 AB$(22,1,4)=BH$(1)
3710 AB$(22,2,4)=BH$(1)
3720 AB$(22,3,4)=HB$(5)
3730 REM ******** BUCHSTABE W ****************
3740 AB$(23,1,1)=BH$(1)
3750 AB$(23,2,1)=BH$(1)
3760 AB$(23,3,1)=BH$(1)
3770 AB$(23,4,1)=HB$(6)
3780 AB$(23,3,2)=HB$(7)
3790 AB$(23,4,2)=HB$(5)
3800 AB$(23,3,3)=HB$(8)
3810 AB$(23,4,3)=HB$(6)
3820 AB$(23,1,4)=BH$(1)
3830 AB$(23,2,4)=BH$(1)
3840 AB$(23,3,4)=BH$(1)
3850 AB$(23,4,4)=HB$(5)
3860 REM ******** BUCHSTABE X ****************
3870 AB$(24,1,1)=BH$(1)
3880 AB$(24,2,1)=HB$(6)
3890 AB$(24,3,1)=HB$(7)
3900 AB$(24,4,1)=BH$(1)
3910 AB$(24,2,2)=HB$(8)
3920 AB$(24,3,2)=HB$(6)
3930 AB$(24,1,3)=BH$(1)
3940 AB$(24,2,3)=HB$(5)
3950 AB$(24,3,3)=HB$(8)
3960 AB$(24,4,3)=BH$(1)
3970 REM ******** BUCHSTABE Y ****************
3980 AB$(25,1,1)=BH$(1)
3990 AB$(25,2,1)=HB$(6)
4000 AB$(25,2,2)=BH$(5)
4010 AB$(25,3,2)=BH$(1)
4020 AB$(25,4,2)=BH$(1)
4030 AB$(25,1,3)=BH$(1)
```

```
4040 AB$(25,2,3)=HB$(5)
4050 REM ******** BUCHSTABE Z ****************
4060 AB$(26,1,1)=BH$(1)
4070 AB$(26,3,1)=HB$(7)
4080 AB$(26,4,1)=BH$(1)
4090 AB$(26,1,2)=BH$(1)
4100 AB$(26,2,2)=HB$(7)
4110 AB$(26,3,2)=HB$(5)
4120 AB$(26,4,2)=BH$(1)
4130 AB$(26,1,3)=BH$(1)
4140 AB$(26,2,3)=HB$(5)
4150 AB$(26,4,3)=BH$(1)
4160 REM ******* ENDE BUCHSTABEN *************
4170 ZK$="                          SYBEX"
4180 REM ************************************
4190 REM * AUS ANSCHAULICHKEITSGRUENDEN WURDE*
4200 REM * AUF SCHLEIFENBILDUNG BEI DER DEFI-*
4210 REM * NITION DER BUCHSTABEN VERZICHTET. *
4220 REM ************************************
4230 IF LEN(ZK$)>50 THEN E=50:GOTO 4260
     :REM DRUCKAUSGABE MIT LEERZEILEN
4240 REM IF LEN(ZK$)>60 THEN E=60:GOTO 4260
     :REM DRUCKAUSGABE OHNE LEERZEILEN
4250 E=LEN(ZK$)
4260 FOR I=1 TO E
4270 : Z$=MID$(ZK$,I,1)      :REM I-TES ZEICHEN
4280 : AC=ASC(Z$)     :REM ASCII-CODE ERMITTELN
4290 : IF AC=32 THEN AC=96    :REM LEERZEICHEN
4300 : GC=AC-64:REM GROSSBUCHSTABENCODE ZUWEI.
4310 : FOR N=1 TO 4
4320 :   FOR K=1 TO 4
4330 :     GS$(I,N,K)=AB$(GC,N,K)
4340 :   NEXT K
4350 : NEXT N
4360 NEXT I
4370 REM ************************************
4380 REM * GROSSBUCHSTABEN AUF SCHIRM UEBERTR*
4390 FOR I=0 TO 5
4400 : FOR J=1 TO SP/4
4410 :   FOR N=1 TO 4
4420 :     FOR K=1 TO 4
4430 :       GB$(I,J,N,K)=GS$(I*10+J,N,K)
4440 :     NEXT K
4450 :   NEXT N
4460 : NEXT J
```

```
4470 NEXT I
4480 REM ************************************
4490 REM * FUER RECHNER OHNE CHAR-BEFEHL KANN*
4500 REM * EINE SIMULATION MIT PRINT VERWEN- *
4510 REM * DET WERDEN. BEISPIEL SIEHE UNTER  *
4520 REM * DEM STICHWORT CHAR                *
4530 REM ************************************
4540 REM *          DRUCKAUSGABE            *
4550 REM ************************************
4560 PRINT DF$
4570 FOR I=0 TO 4            :REM MIT LEERZEILE
4580 REM FOR I=0 TO 5       :REM OHNE LEERZEILE
4590 : FOR J=1 TO (SP/4)*2:M=M+1
4600 :   FOR N=1 TO 4
4610 :     FOR K=J-2 TO J+1:Z=Z+1
4620 :       CHAR 1,J+K,I*5+N-1,GB$(I,M,N,Z)
             :REM MIT LEERZEILE
4630 :   REM CHAR 1,J+K,I*4+N-1,GB$(I,M,N,Z)
             :REM OHNE LEERZEILE
4640 :     NEXT K:Z=0
4650 :   NEXT N:J=K-1
4660 : NEXT J:N=0
4670 NEXT I
4680 CHAR 1,3,23,"BITTE GEBEN SIE IHREN TEXT EIN"
     :INPUT ZK$
4690 PRINT DF$
4700 LZ=LEN(ZK$)
4710 LZ=50-LZ                :REM MIT LEERZEILE
4720 REM LZ=60-LZ           :REM OHNE LEERZEILE
4730 FOR L=1 TO LZ
4740 : BK$=BK$+" "
4750 NEXT L
4760 ZK$=ZK$+BK$
4770 BK$=""
4780 GOTO 4230
4790 END
```

Wenn Sie das Programm starten, müssen Sie ein wenig Geduld haben. Es braucht relativ lange, bis es die Buchstaben auf den Schirm bringt. Wer kann, sollte das Programm kompilieren. Eine zusätzliche Möglichkeit, die Ausführungsgeschwindigkeit zu erhöhen, wäre, das Drucken von leeren Zeichen abzufragen und zu überspringen.

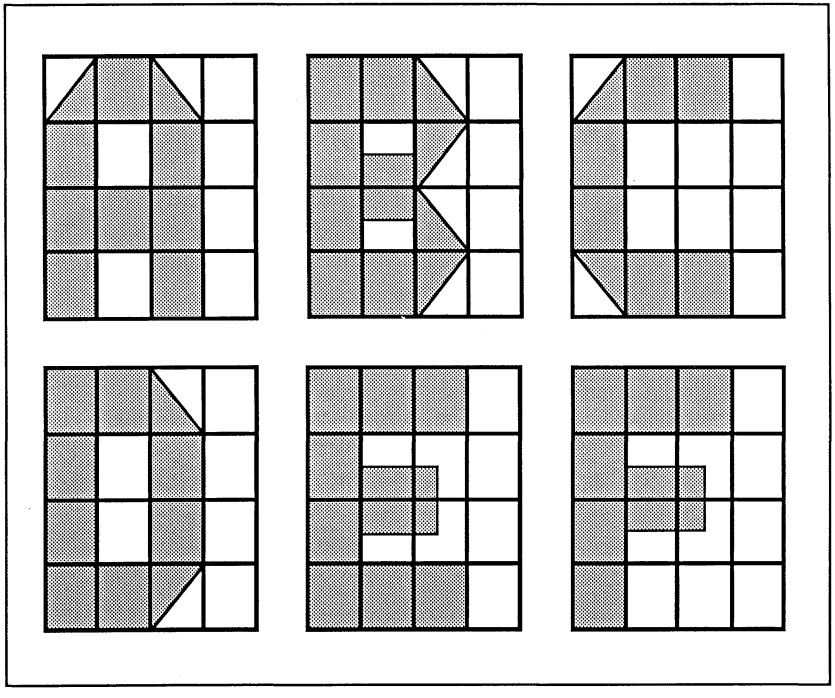
Sie haben bei dem Programm die Wahl, ob die Großbuchstabenzeilen ohne oder mit Leerzeile untereinander gedruckt werden sollen. Dabei müssen nur

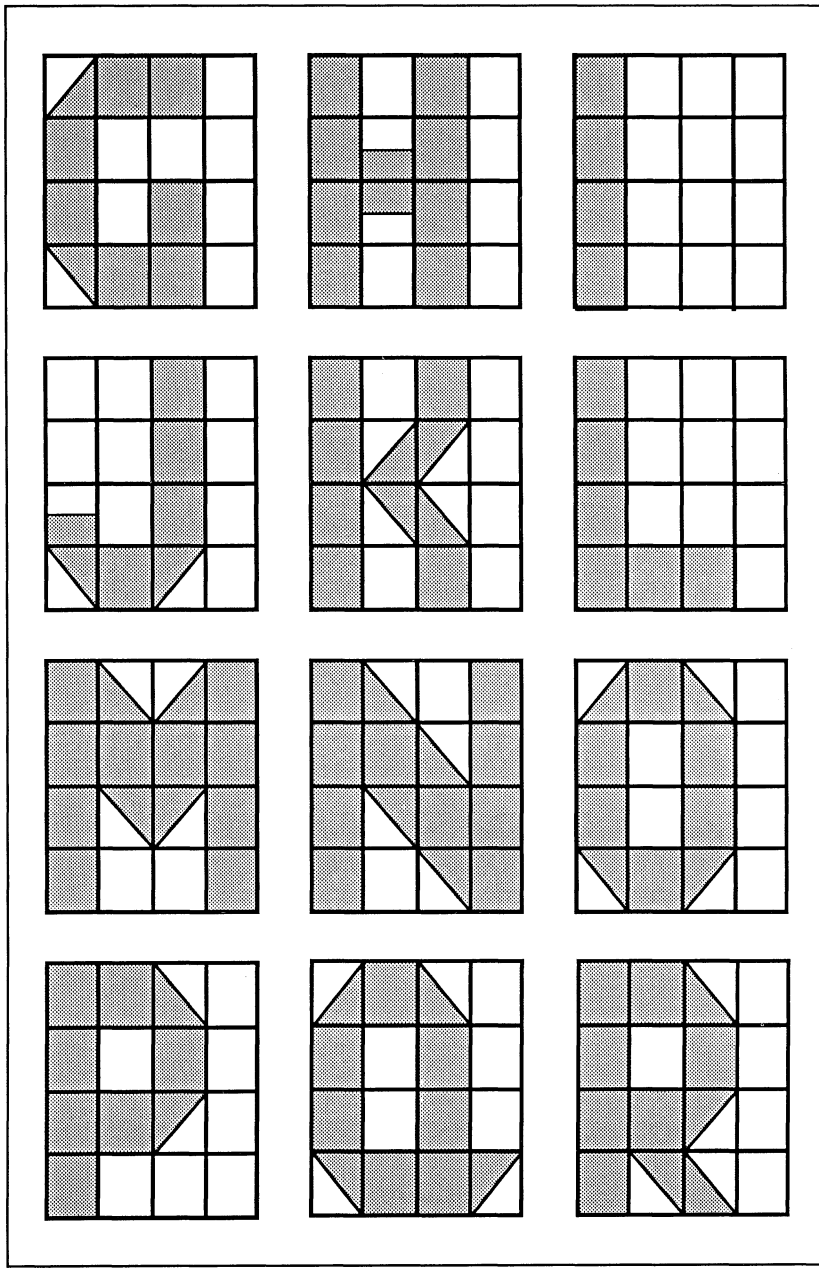
die auf REM gesetzten Zeilen für das Drucken ohne Leerzeilen aktiviert werden und die Zeilen für das Drucken mit Leerzeilen auf REM gesetzt werden. Bei manchen Grafiken ist es sicher wünschenswert, die Zeilen ohne Zwischenraum unmittelbar untereinander zu drucken, und mit dieser Methode erhalten Sie sich beide Möglichkeiten.

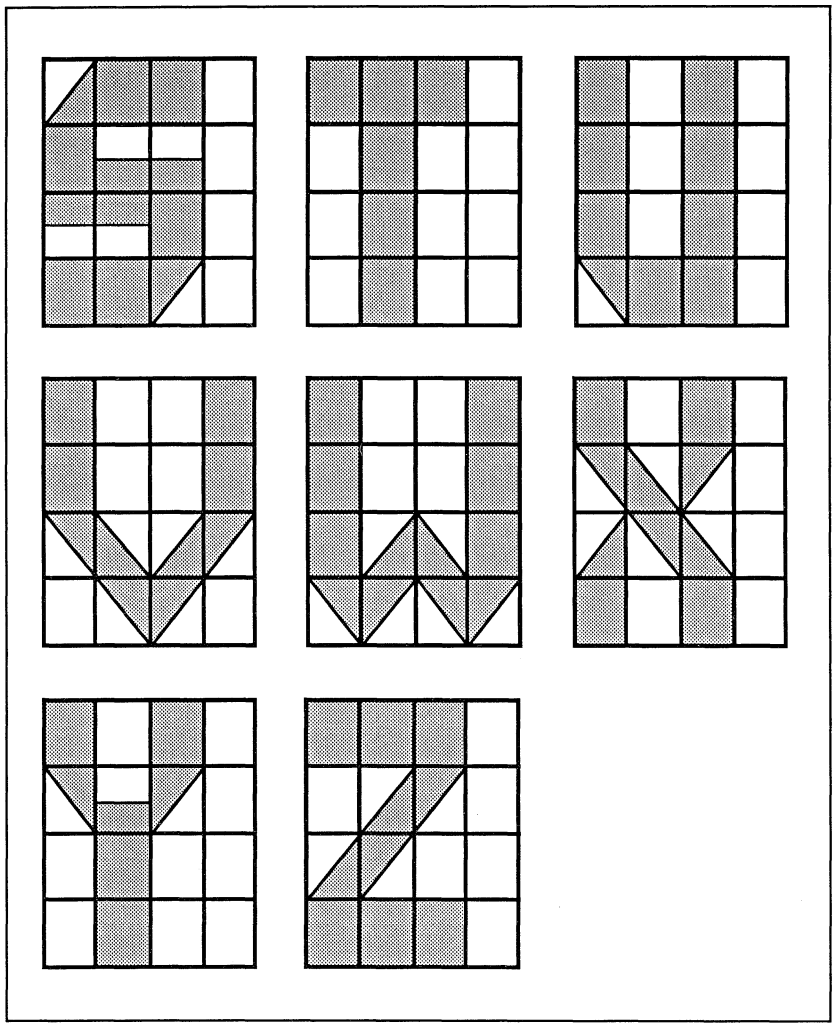
Selbstverständlich können Sie das Programm noch weiter verbessern. Wie wäre es mit farbigen Zeichen oder Laufschrift?

Am geeignetsten ist es, das Programm als Unterprogramm einzusetzen. Dann kann die zu druckende Zeichenkette übergeben werden und braucht selbstverständlich nicht vom Benutzer eingegeben zu werden. Auf diese Weise ist das Programm ideal für Demonstrationszwecke.

Das Alphabet wurde nach folgendem Vorbild gestaltet:







Sie können selbstverständlich statt des Alphabets einen Grafikzeichensatz für Großgrafikzeichen definieren.

Das Konzept des virtuellen Bildschirms, das wir in diesem Beispiel mit der Matrix für den Großzeichen-Bildschirm demonstriert haben, erlaubt es, Anwendungen sehr flexibel und änderungsfreundlich zu entwickeln.

In unserem Beispiel ist es denkbar, noch horizontales oder vertikales Hochrollen oder Drehungen um 90, 180 oder 270 Grad, Spiegelungen um Symmetrieachsen oder Laufschrift in dieses Programmbeispiel einzubringen. Viel Spaß beim Weiterentwickeln!

Die HEX$-Funktion wandelt ganze Zahlen aus dem Wertebereich von 0 bis 65535 in hexadezimale Adressen um. Sie wird hauptsächlich im Zusammenhang mit rechnerinternen Vorgängen und Adressen verwendet. Das rührt daher, daß der Rechner intern Byte-orientiert arbeitet und ein Byte, als Binärzahl interpretiert, eben Werte aus dem Bereich 0 bis 15, also 16 Werte, annehmen kann. Hexadezimal bedeutet schon vom Wort her: 16stellig. Da aber dafür unsere herkömmlichen Ziffern nicht ausreichen, werden in hexadezimaler Darstellung einfach die Zeichen A für 10, B für 11, C für 12, D für 13, E für 14 und F für 15 hinzugenommen. Beim Überschreiten der Zahl 15 zählt man genauso wie im Dezimalsystem mit 1 an der nächsten höherwertigen Stelle weiter.

15+1=16 dezimal

entspricht also:

F+1=10 hexadezimal

Der BASIC-Interpreter unterstützt allerdings das Rechnen mit hexadezimalen Zahlen nicht. Er behandelt sie wie Zeichenketten. Das heißt, falls wir mit Hex-Zahlen rechnen wollen, müssen wir sie zuvor mit DEC, der Umkehrfunktion von HEX$, in dezimale Zahlen zurückverwandeln. Dann führen wir im Dezimalsystem die gewünschte Rechnung durch und wandeln das Ergebnis anschließend mit HEX$ wieder in eine hexadezimale Zahl um.

Das Ergebnis von HEX$ wird immer vierstellig dargestellt, maximal ist es FFFF.

### Beispiele

```
10 X=F+1
```

Falsch, mit Hex-Zahlen kann nicht gerechnet werden.

```
10 X=HEX$(15)+1
```

Falsch, mit Hex-Zahlen kann nicht gerechnet werden.

```
10 IF "0010"=HEX$(15+1) THEN PRINT"RICHTIG"
```

Richtig.

# Zeichenkettenfunktionen

## Umwandlungsfunktionen von Zeichenketten in Zahlen

Die DEC-Funktion als Umkehrfunktion zur HEX$-Funktion akzeptiert als Argumente bis zu vierstellige positive Hex-Zahlen, also die Ziffern 0 bis 9 und die Buchstaben A bis F.

### Beispiele

```
10 PRINT DEC("-FF")
```
Falsch.

```
10 IF 255=DEC("FF") THEN PRINT "RICHTIG"
```
Richtig.

```
10 PRINT DEC("F+F")
```
Falsch.

```
10 IF 30=DEC("F")+DEC("F") THEN PRINT"RICHTIG"
```
Richtig.

```
10 IF "001E"=HEX$(DEC("F")+DEC("F"))THEN PRINT"RICHTIG"
```
Richtig.

```
10 PRINT DEC("F")+"F"
```
Falsch.

```
10 IF 16=DEC("F")+1 THEN PRINT "RICHTIG"
```
Richtig.

Die ASC-Funktion als Umkehrfunktion zur CHR$-Funktion liefert eine Byte-Zahl, die dem ASCII-Code (also der Position in der CBM-ASCII-Tabelle) für das erste Zeichen einer Zeichenkette entspricht.

### Beispiele

```
10 PRINT ASC(3)
```
Falsch, ASC ist nicht auf Zahlen anwendbar.

```
10 IF 32=ASC(" 3") THEN PRINT"RICHTIG"
```
Richtig.

```
10 IF "30"=ASC(CHR$(30)) THEN PRINT"RICHTIG"
```
Richtig.

VAL ist die Umkehrfunktion zu STR$. Sie erkennt alle zulässigen Zeichen, die bei der Umwandlung von Zahlen mit STR$ wie bei einer Ausgabe auf Drucker oder Bildschirm auftreten können. Als erstes Zeichen in der Argument-Zeichenkette muß jedoch eine Ziffer oder ein Vorzeichen erscheinen, sonst wird der Zeichenkettenausdruck als nicht numerischer String behandelt, und es erfolgt wie bei diesem die Ausgabe einer 0. Leerzeichen werden dabei ignoriert.

### Beispiele

```
10 IF 0=VAL(" +2") THEN PRINT"FUEHRENDE LEERSTELLE"

10 IF 2=VAL("+ 2") THEN PRINT"ERLAUBT"

10 IF 0=VAL("B10101") THEN PRINT"ERSTES ZEICHEN NICHT
NUMERISCH"

10 IF 20=VAL("+2.0E1") THEN PRINT"RICHTIG"

10 IF X=VAL(STR$(X)) THEN PRINT"RICHTIG"
```

Die LEN-Funktion gehört ebenfalls zu den Zeichenkettenfunktionen, die als Ergebnis Zahlen liefern. Da die LEN-Funktion die Länge einer Zeichenkette auswertet, die ja nicht länger als 255 Zeichen werden darf, ist das Ergebnis dieser Funktion eine Byte-Zahl, d. h. sie liegt im Wertebereich von 0 bis 255.

Die Hauptanwendung der Längenfunktion liegt bei der Schleifensteuerung zur Begrenzung von Laufvariablen und als Parameter von Zeichenkettenfunktionen, die Teilzeichenketten auswerten.

### Beispiele

```
10 IF LEN(X$)>0 THEN PRINT ASC(X$)
20 PRINT "LEERE ZEICHENKETTE"

10 FOR I=1 TO LEN("12")
20 : PRINT STR$(I)
30 NEXT I
```

### Funktionen für Teilzeichenketten

Die Funktionen LEFT$, MID$, RIGHT$ und INSTR helfen uns dabei, Teilzeichenketten aus einer Zeichenkette herauszuschneiden. Dabei ist MID$ der allgemeinere der drei ersten Befehle, da die beiden anderen eigentlich als Spezialfall von MID$ betrachtet werden können. MID$ und INSTR unterscheiden sich darin, daß MID$ nach Angabe einer Position eine Teilzeichenketten aus der vorgegebenen herauslöst und INSTR nach Angabe einer Teilzeichenkette deren Position in einer anderen ermittelt. INSTR liefert als Ergebnis eine Byte-Zahl zwischen 0 und 255, die die Position des ersten Auftretens der Suchzeichenkette angibt.

INSTR gehört deshalb eigentlich auch in die Gruppe aus dem Grenzbereich zwischen Zeichenketten- und numerischen Funktionen. Sie ergänzt aber in so bedeutender Weise die anderen drei Funktionen und insbesondere die MID$-Funktion, daß sie praktisch Hand in Hand mit diesen auftritt. INSTR erledigt sozusagen die detektivische Vorarbeit, um dann die Zeichenketten-Aufbereitungsspezialisten entsprechend mit Informationen zu versorgen und ihnen dann das Feld zu überlassen.

Die Funktionen für Teilzeichenketten eignen sich auch sehr gut zur Simulation von Rechenoperationen mit als Zeichenketten dargestellten Zahlen zu beliebiger Basis. Sie können über diese sogenannte Zeichenketten-Arithmetik Rechenoperationen mit wesentlich größeren Zahlen und größerer Genauigkeit durchführen, als dies mit den eingebauten Arithmetikfunktionen für dezimale Gleitkommazahlen möglich ist.

Wir haben beispielsweise eine als Zeichenkette dargestellte positive Binärzahl ohne Vorzeichen und wollen von ihr eine Zweierpotenz subtrahieren. Dann muß gezielt an der entsprechenden Position, falls diese eine 1 enthält, eine Null gesetzt werden. Die Position pflückt man mit MID$ heraus und setzt die Zeichenkette anschließend mit LEFT$, MID$ und RIGHT$ wieder zusammen.

Falls die Position aber eine 0 enthält, wird die Stelle auf 1 gesetzt, und die nächsthöhere Stelle wird auf 0 gesetzt, falls in ihr eine 1 steht. Wenn nicht, wiederholt sich das ganze für die nächsthöhere Position so lange, bis eine 1 gefunden wird.

Addition, Multiplikation und Division werden analog simuliert.

Bei mehrstelligem Divisor oder Multiplikator führt man die Operation einfach für jede auftretende Zweierpotenz, die nicht 0 ist, des Divisors bzw. des Multiplikators einzeln durch.

## Weitere Beispiele

```
10 P=INSTR("OTTO","TT",1)
20 IF "O"=MID$("OTTO",P+1,1) THEN PRINT"RICHTIG"


10 IF 4=INSTR("12345678","45",2) THEN PRINT"RICHTIG"


10 P=INSTR("OTTO","TO",1)
20 IF "OT"=LEFT$("OTTO",P-1) THEN PRINT"TREFFER"
```

Das folgende Beispiel eliminiert Leerstellen am Anfang und Ende von Zeichenketten:

```
100 ZK$="   DAS IST SIE     "
110 ZR$=CHR$(18):ZN$=CHR$(146):AF$=CHR$(34):
    DF$=CHR$(147)
120 PRINT DF$
130 RK$=ZK$ :REM ALTE AUFBEWAHREN
140 PRINT ZR$+RK$+ZN$;"  EINGABE":PRINT
150 FOR I=LEN(ZK$) TO 0 STEP -1
160 : IF RIGHT$(ZK$,1)=" "
      THEN ZK$=LEFT$(ZK$,LEN(ZK$)-1)
170 NEXT I
180 FOR I=LEN(ZK$) TO 0 STEP -1
190 : IF LEFT$(ZK$,1)=" "
      THEN ZK$=MID$(ZK$,2)
200 NEXT I
210 PRINT
220 PRINT"DIE ZEICHENKETTE ";AF$+RK$+AF$
230 PRINT"WURDE VON LEERSTELLEN BEFREIT"
240 PRINT
250 PRINT"SIE SIEHT JETZT SO AUS: ";
    ZR$+ZK$+ZN$
```

Das nächste Beispielprogramm wirft nun sämtliche Leerstellen aus einer Zeichenkette hinaus:

```
100 ZK$="   DAS IST SIE     "
110 ZR$=CHR$(18):ZN$=CHR$(146):AF$=CHR$(34):
    DF$=CHR$(147)
120 PRINT DF$
```

```
130 RK$=ZK$ :REM ALTE AUFBEWAHREN
140 PRINT ZR$+RK$+ZN$;"  EINGABE":PRINT
150 FOR I=LEN(ZK$) TO 1 STEP -1
160 : K=INSTR(ZK$," ",1)
170 : IF K>0 THEN ZK$=LEFT$(ZK$,K-1)+
      MID$(ZK$,K+1,LEN(ZK$))
180 NEXT I
190 PRINT
200 PRINT"DIE ZEICHENKETTE ";AF$+RK$+AF$
210 PRINT"WURDE VON LEERSTELLEN BEFREIT"
220 PRINT
230 PRINT"SIE SIEHT JETZT SO AUS: ";
    ZR$+ZK$+ZN$
```

Für bestimmte Aufgabenstellungen kann auch das umgekehrte Zusammenspiel der Teilzeichenketten interessant sein. Wenn eine Ähnlichkeitsanalyse von Zeichenketten durchgeführt werden soll, kann es sinnvoll sein, die Suchzeichenkette und/oder die untersuchte Zeichenkette Schritt für Schritt zu verkürzen bzw. zu verlängern.

Die drei Funktionen erwarten alle als ersten Parameter einen Zeichenkettenausdruck für die zu untersuchende Zeichenkette und eine Zahl, die angibt, wie viele Zeichen aus dieser Zeichenkette in die Ergebniszeichenkette übernommen werden sollen.

In der alphabetischen Befehlsübersicht finden Sie unter DEF FN noch weitere nützliche benutzerdefinierte Hilfsfunktionen für Zeichenketten.

### Die Funktionen SPC, TAB, POS und RWINDOW

Die ersten beiden Funktionen dienen der Ausgabeformatierung im Zusammenspiel mit dem PRINT- oder PRINT#-Befehl. Mit

```
PRINT SPC(numerischer Ausdruck)"Zeichenkette"
```

wird der angegebene Text vor der Ausgabe um soviel Cursorstellen nach rechts verschoben, wie der numerische Ausdruck angibt. Das Argument darf dabei die Werte von 0 bis 255 annehmen. Die Positionen bis zum Druck der Zeichenkette werden dabei auf dem Bildschirm nicht mit Leerzeichen gefüllt, sondern einfach übersprungen.

Der Befehl

```
PRINT TAB(numerischer Ausdruck)"Zeichenkette"
```

hat im Prinzip die gleiche Wirkung wie der Druckbefehl mit SPC. Der Unterschied ist, daß mit TAB feste Tabulationspunkte angesprungen werden, ausgehend vom linken Bildschirmrand, der mit PRINT TAB(0) angesprochen wird, während mit SPC die Cursorverschiebung relativ zur letzten Cursorposition stattfindet. Dabei werden dazwischenliegende Positionen übersprungen, aber nicht gelöscht.

Deshalb kann man bei TAB auch zwei aufeinanderfolgenden Angaben, von denen die zweite kleiner als die erste ist, nicht mit Erfolg ausführen:

```
PRINT TAB(10)"A"TAB(5)"B"
```

während das bei SPC problemlos ist:

```
PRINT SPC(10)"A"SPC(5)"B"
```

Bei TAB sind ebenso wie bei SPC Argumente von 0 bis 255 zugelassen.

TAB hat außerdem nur bei der Ausgabe auf den Bildschirm eine Funktion. Bei der Ausgabe auf den Drucker können Sie nur SPC zur Positionierung der Schreibstellen verwenden.

Die Funktion POS zeigt auch nur im Zusammenhang mit der Bildschirmausgabe eine Wirkung. Sie gibt nämlich die Position des Cursors innerhalb der aktuellen Schreibzeile an. Das ist immer die Zeile, in der die letzte Ausgabe erfolgte. Wenn Sie also relativ zu dieser letzten Ausgabe eine weitere Ausgabe innerhalb der gleichen Bildschirmzeile durchführen wollen, brauchen Sie nur zu dem Wert, der von POS geliefert wird, den gewünschten Abstand für die nächste Ausgabe hinzuzuaddieren.

Das ist besonders für die Ausgabe von Zeichenketten nützlich, deren Länge vorher nicht genau bekannt ist. Bei aufeinanderfolgenden Ausgaben mit fester Länge kann dagegen die absolute Positionierung der Schreibstelle mit TAB vorgenommen werden.

### Beispiele

```
10 INPUT"BITTE ZEICHENKETTE1,ZEICHENKETTE2
EINGEBEN";Z1$,Z2$
20 PRINT TAB(0)Z1$;TAB(POS(0)+4)Z2$
```

Will man relativ zur Schreibzeile zusätzlich noch die Druckposition einige Zeilen vorwärtsbewegen, braucht man nur TAB(POS(0)+40*N) bzw. TAB(POS(0)+80*N) für N Zeilen Vorschub einzugeben. Allerdings darf das Argument für TAB nicht größer als 255 werden. Falls Sie mehr Vorschub

wünschen, müssen Sie zusätzlich PRINT-Befehle einfügen. Jeder PRINT-Befehl, der nicht mit einem Semikolon abgeschlossen wird, erzeugt einen Zeilenvorschub.

Die Zeile

```
30 PRINT TAB(0)Z1$;TAB(POS(0)+40*3)Z2$
```

gibt die Zeichenkette Z2$ auf einem 40-Spalten-Bildschirm 3 Zeilen tiefer als die Zeichenkette Z1$ aus. Dazu kommt eine Verschiebung nach rechts in der Länge von Z1$.

Die Zeile

```
40 PRINT TAB(0)Z1$:PRINT:PRINT:PRINT:PRINT:PRINT
TAB(0)Z2$
```

druckt Z2$ genau 5 Zeilen unterhalb von Z1$ aus.

Mit einem kleinen Trick kann man auch TAB und POS nutzbringend für die Ausgabe auf dem Drucker einsetzen. Drucken Sie jede Zeile sowohl auf dem Bildschirm als auch auf dem Drucker, wobei Sie auf dem Bildschirm mit TAB und anschließend auf dem Drucker mit SPC(POS(2)-P1) die Druckposition verändern und diese Position stets in der Variablen P1 nachhalten. So können Sie sogar die relative Positionierung mit TAB(POS(0)+X) einsetzen. Es ist jedoch guter Programmierstil, stets in besonderen Zählern sowohl bei Bildschirm- als auch bei Druckerausgaben die aktuelle Zeilen- und Spaltenposition sowie die Seitenzahl nachzuführen.

Verwechseln Sie die TAB-Funktion nicht mit der TAB-Taste auf Ihrem Rechner. Beide haben zwar etwas mit Cursorsteuerung zu tun und helfen, Daten formatgerecht, z. B. in tabellarischer Form, auszudrucken. Die TAB-Taste legt jedoch Tabulatorstops für den ganzen Bildschirm fest, die mittels des Kommas in PRINT-Befehlen angesteuert werden können. Die TAB-Funktion dagegen wirkt immer nur innerhalb des aktuellen PRINT-Befehls und der aktuellen Ausgabezeile und nimmt keinen Bezug zu den mit der TAB-Taste gesetzten Tabulatorstops.

Die RWINDOW-Funktion, die die Parameter des aktuellen Bildschirmfensters ermittelt, bietet eine gewisse Unterstützung bei der Anpassung von Druckausgaben an die aktuelle Bildschirm- und Fenstergröße.

In fast allen Beispielprogrammen dieses Buchs, in denen die Formatierung auf dem Bildschirm eine wichtige Rolle spielt, wird die Variable SP verwendet, um Bildschirmausgaben auf die verfügbare Spaltenzahl abzustimmen.

Wenn ein Programm in BASIC 7.0 häufig auf einen 40- oder 80-Spalten-Bildschirm oder sogar in einem Programm abwechselnd auf einen 40- und einen 80-Zeichen-Schirm Ausgaben machen soll, kann die Variable SP an allen Stellen, wo sie auftritt durch RWINDOW(2) ersetzt werden.

Bei der Formatierung von Ausgaben auf Bildschirmfenster ist RWINDOW nur eingeschränkt nützlich, weil eine Abfrage nur auf den unteren rechten Eckpunkt möglich ist. Man kann sich also entweder darauf beschränken, nur Fenster zu verwenden, deren linker oberer Eckpunkt mit dem des gesamten Bildschirms identisch ist, oder man muß dessen Koordinaten in speziellen Variablen festhalten.

### Beispiel

```
10 X1=2:Y1=2
20 WINDOW X1,Y1,10,10,1
30 A$="WWWWWWWWWWWWWW"
40 PRINT LEFT$(A$,(RWINDOW(1)-X1))
```

## Grafik- und Sprite-Funktionen

Eine Vielzahl der neuen Funktionen im BASIC 7.0 gehört zu der Kategorie, die wir nun besprechen wollen. Es handelt sich um die Grafikfunktionen:

```
RCLR, RDOT, RGR, RLUM
```

und die Sprite-Funktionen:

```
BUMP, RSPCOLOR, RSPPOS, RSPRITE
```

Alle diese Funktionen haben eines gemeinsam. Sie dienen der Abfrage von voreingestellten Parametern oder sich ergebenden Positionen. Die Argumente sind bei den Funktionen durchaus verschieden und können im einzelnen dem speziellen Stichwort im alphabetischen Teil des Buches entnommen werden.

Die Funktion RCLR hilft uns bei der farbigen Ausgabe, die aktuelle Farbe auf die Situation auf dem Bildschirm abzustimmen. So kann vermieden werden, daß zufällig gefärbte Flächen nicht die Farbe ihrer Nachbarfläche oder des Hintergrunds annehmen. Die Funktion kann auch dazu verwendet werden, um nur sinnvolle Farbkombinationen für Schrift- und Hintergrundfarbe, wie z. B. Grün auf Schwarz, zuzulassen.

### Beispiele

```
10 IF RCLR(5)=6 THEN COLOR 0,1
```

```
10 COLOR 0,1:COLOR 5,4:COLOR 1,3:COLOR 4,16
20 FOR C=0 TO 6
30 : PRINT RCLR(C)
40 NEXT C
```

Die Funktion RDOT ist ein wichtiges Hilfsmittel im Zusammenhang mit den Befehlen LOCATE und SCALE, wenn Sie mit relativer Adressierung auf dem Grafikschirm arbeiten. So können Sie jederzeit die Position des unsichtbaren grafischen Cursors kontrollieren und vermeiden, daß versucht wird, außerhalb der Bildschirmkoordinaten zu schreiben.

### Beispiel

```
10 GRAPHIC 1,1:SCNCLR:SCALE 0
20 FOR I=0 TO 1024 STEP 10
30 : LOCATE I,100
40 : CHAR 1,15,20,STR$(RDOT(0))+" , "+STR$(RDOT(1))
50 NEXT I
60 GETKEY T$
70 GRAPHIC 0,1
```

Die Funktion RGR ermittelt den mit dem GRAPHIC-Befehl eingestellten Grafikmodus und erlaubt eine dem aktuellen Bildschirmmodus angepaßte Ausgabegestaltung.

### Beispiele

```
10 GRAPHIC 1,1:SCNCLR
20 CHAR 1,10,3,STR$(RGR(0))
30 FOR I=1 TO 500:NEXT I
40 GRAPHIC 2,1
50 PRINT TAB(81)SPC(10)RGR(0)
60 FOR I=1 TO 500:NEXT I
70 GRAPHIC 0,1
```

```
10 IF RGR(0)=0 OR RGR(0)=2 THEN PRINT"TEXTMODUS"
   ELSE CHAR 1,1,1,"GRAFIKMODUS"
```

Die Abfrage von RGR hilft, Fehlersituationen im Zusammenhang mit der Programmierung von Grafiken zu vermeiden. Grafikausgaben können entsprechend dem aktuellen Grafikmodus skaliert werden:

```
10 XMAX=320:YMAX=200
11 FOR I=0 TO 4
12 : GRAPHIC I,1
13 : FOR J=1 TO 1000:NEXT J
14 : ON RGR(0) GOSUB 17,21,25,29
15 NEXT I
16 STOP
17 GRAPHIC 0,1
18 PRINT"X=";40;"Y=";25
19 FOR J=1 TO 2000:NEXT J
20 RETURN
21 GRAPHIC 0,1
22 PRINT"X=";XMAX,"Y=";YMAX
23 FOR J=1 TO 2000:NEXT J
24 RETURN
25 GRAPHIC 0,1
26 PRINT "X=";XMAX;"Y=";160
27 FOR J=1 TO 2000:NEXT J
28 RETURN
29 GRAPHIC 0,1
30 PRINT"X=";160;"Y=";160
31 FOR J=1 TO 2000:NEXT J
32 RETURN
```

Die Funktion RLUM ermittelt die eingestellte Helligkeit. Diese Funktion gibt es nur in BASIC 3.5. Dort kann mit einem dritten Parameter beim COLOR-Befehl die gewünschte Helligkeit mit Werten von 0 (dunkel) bis 7 (hell) eingestellt werden. Diese Funktion ist als Ergänzung zu RCLR zu sehen, da es bei BASIC 3.5-Rechnern aufgrund der 8 Farbabstufungen 128 verschiedene Farbtönungen gibt. Das heißt, man muß nach der Farbabfrage noch eine Helligkeitsabfrage durchführen.

### Beispiel

```
100 ON RCLR(0)*RLUM(0) GOSUB 1001,...,1128
110 STOP
1001 COLOR 5,6,1:REM SCHRIFTFARBE MIT HINTERGRUND
ABSTIMMEN
.
.
.
1128 COLOR 5,4,7
```

Die Funktion BUMP ermittelt den Zusammenstoß eines Sprites mit einem anderen oder mit einem Hintergrundzeichen. Sie wird in der Regel zusammen mit dem COLLISION-Befehl verwendet, um die an einem Zusammenstoß beteiligten Sprites zu ermitteln und dann eine entsprechende Verzweigung durchzuführen. Der Wert von BUMP sollte zunächst einer Variablen zugewiesen werden, wenn er später noch einmal verwendet werden soll, da BUMP sofort nach dem Zusammenstoß wieder auf 0 gesetzt wird.

**Beispiel**

```
10 SK=BUMP(0):HK=BUMP(1)
20 IF SK AND 2 THEN GOSUB 500
```

Die Funktion RSPCOLOR ermittelt die Farbnummern für die Sprites im Mehrfarbenmodus, die mit dem Befehl SPRCOLOR gesetzt wurden. Sie unterstützt z. B. das Unsichtbarmachen von Sprites, indem die Spritefarbe gleich der Hintergrundfarbe gesetzt wird, oder ermöglicht spezielle Effekte durch Wechsel der Spritefarbe.

**Beispiel**

```
10 IF RSPCOLOR(0)=4 THEN SPRCOLOR 6,4
```

Die Funktion RSPPOS ermittelt die Position eines Sprites und seine Geschwindigkeit. Sie wird vor allem im Zusammenhang mit dem Befehl MOVSPR verwendet. Beachten Sie bitte, daß sich die gelieferten Koordinatenwerte stets auf den Bildschirm bei ausgeschalteter Skalierung beziehen, auch wenn Sie SCALE 1 programmiert haben.

**Beispiel**

```
10 IF RSPPOS(1,0)=1 THEN SPRITE 4,0,6
```

Die RSPRITE-Funktion ermöglicht, die aktuellen Eigenschaften eines Sprites, die mit dem SPRITE-Befehl gesetzt wurden, abzufragen und kann so eine gezielte Veränderung dieser Spezifikationen vorbereiten.

**Beispiel**

```
10 IF RSPRITE(4,5)=1 THEN SPRITE 4,0,6
```

## Systemfunktionen

Zu den Systemfunktionen gehört die schon eingangs besprochene Funktion FRE. Sie liefert uns den noch verfügbaren Speicherplatz in Bytes. Ihr Argument ist, wie wir uns erinnern, eine Dummy-Variable für alle BASIC-Versionen bis auf 7.0. In BASIC 7.0 ist der Wert von FRE(0) bzw. FRE(1) die Anzahl der freien Speicherstellen in der BANK 0 oder 1. Außerdem löst die Abfrage von FRE eine Garbage Collection aus.

Die drei Systemfunktionen zur Abfrage der peripheren Geräte Joystick, Lightpen und Paddles, die alle dieselben Anschlüsse verwenden, sind: JOY, PEN und POT. Sie lesen den Inhalt bestimmter Systemvariablen aus, die die Werte der Signale enthalten, die an den Steuerknüppelanschlüssen anliegen. Diese Werte können aufgrund von Signalschwankungen recht unstabil sein, so daß es stets empfehlenswert ist, über mehrere hintereinander ausgelesene Werte den Mittelwert zu bilden. In BASIC 2.0 und 4.0 können diese Funktionen mit DEF FN und PEEK nachgebildet werden.

Die JOY- und die POT-Funktion sind besonders für Steuerungszwecke im Zusammenhang mit animierter Grafik geeignet. Für die Kontrolle der Koordinaten bei der Erstellung von Grafiken sind die PEN- oder auch die POT-Funktion am besten geeignet, wobei allerdings ein Lichtstift nur bei bestimmten Kombinationen von Zeichenfarbe und Hintergrundfarbe, wie beispielsweise grün und schwarz, ein vernünftiges Arbeiten erlaubt.

Mit PEN oder POT kann außerdem auch ein Grafiktablett oder eine Rollmaus abgefragt werden. Dabei muß immer berücksichtigt werden, daß nur Werte von 0 bis 255 zurückgeliefert werden. Diese müssen, um den gesamten Bildschirm im hochauflösenden Grafikmodus, ansprechen zu können, mit einem Faktor multipliziert werden. Für die X-Koordinaten beträgt dieser Faktor etwa 1.25 und für die Y-Koordinaten 0.78.

Im Zusammenhang mit Auswahlmenüs ist die PEN-Funktion sicherlich am geeignetesten. Man könnte natürlich auch die JOY- oder POT-Funktion für diesen Zweck einsetzen. Das erfordert aber meistens einen höheren Programmieraufwand, weil man für den Benutzer einen Cursor simulieren muß.

Bei der Menüpunktauswahl mit Lichtstift benötigt man nur eine ausreichend große, helle Fläche, etwa 4 x 4 Zeichen groß, auf dunklem Hintergrund, die man als Ansetzpunkt für den Lichtstift vor die Menüpunkte setzt.

Weitere Systemfunktionen sind PEEK und POINTER.

Mit der Funktion PEEK, die ein Auslesen einzelner Bytes aus dem Arbeitsspeicher und aus den Registern verschiedener Hilfsprozessoren ermöglicht, kann beispielsweise die PEN-Funktion für den C64 nachgebildet werden.

**Beispiel**

```
10 DEF FNPN(X)=PEEK(53267)*0↑X+PEEK(53268)*0↑(1-X)
```

Vor dem Aufruf der Funktion PN muß zunächst der Rechner auf den Lichtstiftbetrieb eingerichtet werden.

```
20 GOSUB 9000:REM AKTIVIEREN LICHTSTIFT
30 PRINT FNPN(0):REM X-KOORDINATE
40 PRINT FNPN(1):REM Y-KOORDINATE
50 STOP
9000 REM AKTIVIEREN LICHTSTIFT
9010 POKE 53281,1
9020 RETURN
```

Die Funktion PN liefert die gleichen Ergebnisse wie die PEN-Funktion des BASIC 7.0.

Die Funktion POINTER ist ebenso wie PEN, POT, JOY und PEEK eine Funktion, die für das Auslesen bestimmter Systemvariablen verwendet wird.

## Kapitel 6

# Maschinennahe Befehle und Variablen

### Arbeiten mit Systembefehlen und Systemvariablen im Direktmodus

Im Commodore-BASIC, besonders im BASIC 7.0, werden uns viele Befehle zur Verfügung gestellt, mit denen wir im Direktmodus unmittelbar auf unseren Rechner einwirken können. Viele nützliche Funktionen können direkt, ohne daß wir ein Programm dafür brauchen, wie bei einem Taschenrechner auf dem Bildschirm angewendet werden. Für diesen superkomfortablen Quasi-Taschenrechner steht uns der ganze Schirm in seiner jeweils rechnerspezifischen Größe zur Verfügung.

**Beispiel**

```
Eingabe:    ?3+4↑2 <RETURN>
Anzeige:    19

            READY.
```

Ändern Sie jetzt die Eingabe ab:

```
Eingabe:    ?3+DEC("AAAA")
Anzeige:    43693

            READY.
```

Sie können praktisch alle Funktionen direkt aufrufen, wobei sich die Zeichenkettenfunktionen und die zur Umwandlung verschiedener Datentypen als besonders nützlich herausstellen. Die Umwandlungsfunktionen bestehen aus den Befehlen HEX$, DEC, CHR$, ASC, VAL und STR$.

Sie haben vielfältige Möglichkeiten zur Speicher- und Programmüberprüfung. Wenn Sie z. B. die hexadezimale Adresse einer Variablen wissen wollen, tippen Sie einfach:

```
? HEX$(POINTER(A))
```

Wollen Sie den ASCII-Code eines auf dem Bildschirm angezeigten Zeichens feststellen, fahren Sie einfach mit dem Cursor zu diesem Zeichen. Schließen Sie es in Anführungszeichen und runde Klammern ein, löschen Sie vor und hinter dem Zeichen mit Hilfe der speziellen Escape-Sequenzen ESC+"P" und ESC+"Q" alle anderen Zeichen aus dieser Zeile und schreiben zum Schluß noch ?ASC vor die Klammern. Wenn Sie nun RETURN drücken, erscheint die dem Zeichen entsprechende ASCII-Code-Nummer.

Mit der Systemfunktion PEEK können Sie die für Sie wichtigen Systemvariablen aus dem Systemspeicher auslesen, auf dem Schirm anzeigen und bei Bedarf sogar verändern und mit POKE in den Speicher zurückschreiben.

Mit ?FRE(1) kann man erfahren, wieviel Platz noch in der Datenspeicherbank übrig ist, oder mit ?FRE(0) feststellen, um wie viele Zeilen Sie etwa Ihr Programm in der Speicherbank 1 noch vergrößern können.

Wollen Sie gar ein Maschinenprogramm editieren und es anschließend in Ihrem BASIC-Programm in speziell dafür vorgesehenen REM-Zeilen abspeichern, damit es dort sicher aufgehoben ist und bequem zusammen mit dem BASIC-Programm abgespeichert und wieder geladen werden kann, brauchen Sie nur mit dem MONITOR-Befehl den eingebauten Maschinensprache-Monitor aufzurufen. Dieser stellt Ihnen alle für Ihr Vorhaben benötigten Befehle zur Verfügung. Anschließend können Sie Ihr Maschinenprogramm mit SYS oder USR aufrufen, um es zu testen.

Falls es für Testzwecke erforderlich ist, können Sie mit dem BSAVE-Befehl ein Datenfeld mitsamt den aktuellen Inhalten der Variablen auf der Diskette abspeichern. Sie müssen nur vorher mit der POINTER-Funktion die Start- und die Endadresse des betreffenden Speicherbereichs ermitteln.

Für den Programmtest und die Programmeditierung stehen Ihnen die Befehle AUTO, CMD, CONT, DELETE, GOTO, GOSUB, HELP, KEY, LIST, RENUMBER, RESUME, RUN, SLEEP, STOP, TRAP, TRON UND TROFF zur Seite.

Wenn Sie einmal völlig von vorne anfangen wollen, können Sie mit

```
NEW <RETURN>
```

das im Speicher befindliche Programm löschen.

Welche Möglichkeiten Ihnen die Befehle AUTO, DELETE, RENUMBER und KEY bieten, ist schon im zweiten Kapitel detailliert dargestellt worden. Für das Austesten Ihrer Programme können Sie mit RUN und Eingabe einer Zeilennummer oder GOTO bzw. GOSUB Zeilennummer die Programmaus-

führung gezielt ab einer bestimmten Zeile starten. Der Unterschied zwischen den Befehlen besteht darin, daß bei RUN alle Variableninhalte zurückgesetzt werden, bei GOTO und GOSUB hingegen bleiben sie erhalten, was manchmal wünschenswert sein kann. Mit GOSUB Zeilennummer können nur Unterprogramme angesprochen werden, die am Ende den RETURN-Befehl enthalten.

Jederzeit kann der Programmablauf mit der RUN/STOP-Taste unterbrochen werden. Bei kommerzieller Software ist diese Taste aus diesem Grund oft deaktiviert, und Sie werden dann kein Glück haben, wenn Sie darauf drücken. Auch gezielt in den Programmcode eingesetzte STOP-Befehle führen zur Programmunterbrechung. Anhand der vom System ausgegebenen Zeilennummer kann man den Programmfluß überprüfen, und auch Variablenwerte können abgefragt und verändert werden. Solange Sie das Programm selbst dabei nicht verändern, kann die Programmausführung jederzeit mit CONT an genau der Stelle fortgesetzt werden, wo sie unterbrochen wurde.

Die Programmunterbrechung kann genutzt werden, um mit

```
PRINT Variablenname <RETURN>
```

gezielt einzelne Variableninhalte abzufragen. Will man ein ganzes Datenfeld anschauen, braucht man nur

```
FOR I=1 TO N:PRINT A(I),:NEXT I <RETURN>
```

einzutippen.

Ist Ihnen das Programmverhalten einmal völlig schleierhaft, können Sie mit dem Befehl TRON, den Sie auch im Programmodus einsetzen können, erreichen, daß zusätzlich zu den normalen Programmausgaben fortlaufend die Zeilennummer der aktuell bearbeiteten Zeile in eckigen Klammern auf dem Schirm erscheint.

Haben Sie sich den nötigen Durchblick verschafft und wird Ihnen die Zeilennummernausgabe lästig, können Sie sie mit TROFF wieder abstellen.

Wenn Ihr Programm Ausgaben von mehreren Bildschirmseiten produziert, können Sie mit SLEEP 1 vor jedem Druckbefehl einen richtigen Zeitlupeneffekt erzielen. Sie können dann z. B. bei der Produktion von Grafiken genau verfolgen, was im einzelnen passiert und in welcher Reihenfolge Aktivitäten ablaufen.

Jedesmal wenn Ihr Programm beim Austesten mit einer Fehlermeldung abbricht, können Sie duch Eingabe von HELP oder über eine entsprechend belegte Funktionstaste, bzw. beim C128 die HELP-Taste den Rechner dazu bewegen, Ihnen die fehlerhafte Programmzeile anzuzeigen.

Für Testzwecke können Sie auf einfache Weise den Bildschirm in zwei logische Fenster teilen. Bringen Sie den Cursor etwa auf den rechten äußeren Rand der mittleren Bildschirmzeile. Nun betätigen Sie die ESC-Taste und drücken anschließend die Taste B. Mit dieser Aufteilung können Sie z. B. ständig die Programmzeilen eines Programmsegments im Auge behalten, während Sie gleichzeitig in dem anderen Fenster die Programmausgaben oder die mit TRON aktivierte Anzeige des Programmlaufs verfolgen.

Zu diesem Zweck listen Sie zunächst mit

```
LIST Zeilennummer1-Zeilennummer2
```

Ihr gewünschtes Programmstück in der unteren Bildschirmhälfte, indem Sie den LIST-Befehl knapp unterhalb der mittleren Bildschirmzeile eingeben. Dann definieren Sie, wie oben besprochen, das obere Bildschirmfenster, und starten Sie Ihr Programm mit RUN oder GOTO, nachdem Sie eventuell zuvor mit TRON die Anzeige der Zeilennummern aktiviert haben.

Falls das Programmsegment oder der erzeugte Bildschirmausdruck zu umfangreich sind, um gleichzeitig auf dem Bildschirm angezeigt zu werden, können Sie Ihr Listing auch mit

```
OPEN 4,4:CMD4:LIST
```

als Papierausdruck auf dem Drucker ausgeben.

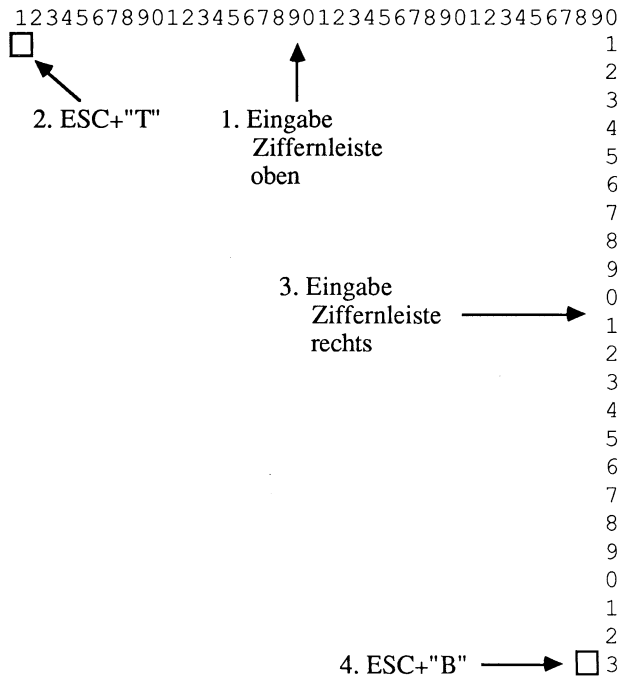
Sie können natürlich auch Papierausgaben mit entsprechender Veränderung des OPEN-Befehls (Kanalnummer von 4 auf 3 ändern) und der PRINT#-Befehle (entsprechend von 4 auf 3 ändern) zunächst auf dem Bildschirm betrachten. Hier können Sie durch Niederhalten der Commodore-Taste bzw. der Pfeil-links-Taste bei manchen CBM-Rechnern die Ausgabe langsamer ablaufen lassen, um den Druckaufbau genauer zu analysieren. Sie können sich auch die oberste Bildschirmzeile als Fenster definieren und dort eine Skala der Form:

```
1234567890123456789012345678901234567890
```

zur Überprüfung von Druckpositionen bei der Ausgabe anlegen.

Geben Sie die Ziffernfolge ein. Wenn nun Ihr Cursor in der ersten Spalte der zweiten Zeile steht, drücken Sie hintereinander die ESC- und die T-Taste. Ihr Bildschirm ist nun für die Überwachung der Druckausgaben vorbereitet.

Wenn Sie wollen, können Sie natürlich auch noch am rechten Bildschirmrand zusätzlich eine senkrechte Zählleiste anbringen, etwa so:




So ist Ihr Ausgabebildschirm perfekt.

Wenn Sie bestimmte von Ihren Programmen auf dem Bildschirm erzeugte Ausgabebilder konservieren wollen, drücken Sie vor oder nach der Ausgabe ESC, gefolgt von M. Das verhindert, daß der Bildschirm weiterrollt. Dann können Sie mit BSAVE ab der Adresse 1024 den dort befindlichen Bildschirmspeicher bis zum Ende (1024+24*Spaltenzahl) abspeichern.

Eine andere Möglichkeit ist, die in diesem Buch enthaltene Bildschirmausdruck-Routine, die in Kapitel 8 unter dem Stichwort PEEK steht, als Unterprogramm zu verwenden. Wie Sie fremde Routinen in Ihr eigenes Programm einbauen, lesen Sie bitte im Kapitel 2 nach.

Natürlich können wir im Direktmodus nicht nur ganze Bildschirminhalte auf die Diskette oder Kassette schreiben, sondern uns stehen alle Ein-/Ausgabebefehle sowie alle Dienstprogramm-Aufrufe für bestimmte Funktionen des Diskettenlaufwerks sozusagen auf Knopfdruck zur Verfügung. Mit dem KEY-Befehl wird uns augenblicklich die aktuelle Funktionstastenbelegung gezeigt.

Wenn Sie nun eine längere Arbeitssequenz planen, während der Sie überwiegend Diskettenfunktionen verwenden wollen, können Sie die 8 Funktionstasten mit

```
KEY Funktionstastennummer,"Befehl"+RETURN
```

neu belegen. RETURN wird durch CHR$(34) vertreten, und die folgenden Befehle könnten sinnvoll sein: APPEND, BACKUP, BLOAD, BOOT, BSAVE, CLOSE, COLLECT, CONCAT, COPY, DCLEAR, DCLOSE, DIRECTORY, DLOAD, DOPEN, DSAVE, PRINT DS$, DVERIFY, HEADER, LOAD, OPEN, PRINT#, RECORD, RENAME, SAVE, SCRATCH, PRINT ST. Daraus können Sie die 8 Befehle auswählen, die Sie voraussichtlich am häufigsten benötigen werden.

Für eine Aufräumaktion auf Ihren Disketten werden Sie zum Beispiel mit hoher Wahrscheinlichkeit die Befehle COPY, SCRATCH, RENAME, DIRECTORY, HEADER und COLLECT benötigen. Geben Sie bei der Funktionstastenbelegung gleich die Parameter für diese Befehle mit an, wie beispielsweise Gerätenummern und Laufwerknummern. Es können auch kurze Befehlssequenzen bei der Tastendefinition angegeben werden. Das ist besonders für längere und aufgrund ihrer Komplexität fehlerträchtige Befehlsfolgen nützlich. Dazu gehören z. B. die Befehle für den Direktzugriff auf Diskettendateien.

Bei der Belegung der Funktionstasten muß man allerdings berücksichtigen, daß alle Belegungen zusammengenommen nur 255 Zeichen umfassen dürfen. Bei umfangreichen Belegungen empfiehlt es sich auch, diese mit einem kleinen Hilfsprogramm vorzunehmen, das diese Belegungen fest enthält. Laden Sie dieses Programm, bevor Sie mit Ihrer eigentlichen Arbeit beginnen, lassen Sie es mit RUN laufen und löschen es dann mit NEW.

Bei Aufräumarbeiten mit Disketten ist es immer ratsam, eine ausreichende Anzahl neuer Disketten, bereits mit HEADER formatiert, bereit liegen zu haben. Sowohl bei der Namensvergabe als auch bei der Festlegung der Identifikationsnummer Ihrer Disketten sollten Sie einer bestimmten Systematik folgen, damit Sie den Überblick behalten.

Bevor man beginnt, auf einer Diskette Dateien zu löschen, sollte man sich deren Inhaltsverzeichnis mit dem Befehl DIRECTORY zuvor anschauen. Die Diskette, die wir bereinigen wollen, heißt "DISKUTILITIES 86" und zeigt die folgenden Dateien:

```
"UNITTOUNIT791128"     PRG
"TESTSORT1-860528"     PRG
"TESTSORT1-860428"     PRG
```

```
"TESTSORT2-860614"     PRG
"TESTSORT2-860702"     PRG
"WORK1-860702"         SEQ
"WORK2-860702"         SEQ
"DOK-SORT2-860525"     SEQ
"DOK-SORT1-850127"     SEQ
"DISKCOPY3-850731"     PRG
"FILEPRINT-810315"     PRG
```

Diese Diskette soll nun konsolidiert werden, d. h. überflüssige Daten sollen entfernt werden, und die Dateien sollen alphabetisch sortiert werden. Leider haben wir bisher dafür noch kein Hilfsprogramm erstellt.

Wir positionieren nun den Cursor auf die Zeile mit dem Namen TEST-SORT1-860528, da wir uns diese letzte Fassung der ersten Version unseres Sortierprogramms ansehen wollen. Die zweite Version haben wir bereits mit Erfolg gestestet. Da wir bereits auf zwei anderen Disketten eine Archivversion dieses Sorts aufgehoben haben, wollen wir die Kopie auf dieser Diskette löschen. Um aber sicherzugehen, daß es sich tatsächlich um diese erste Version handelt, schauen wir sie uns vor dem Löschen noch einmal an.

Vor den Namen tippen wir DLOAD, fahren mit dem Cursor unmittelbar hinter die zweiten Anführungszeichen und tippen einen Doppelpunkt, um den Rest der Zeile zu ignorieren. Wenn wir nun RETURN drücken, wird das Programm geladen. Mit LIST wird nun gesehen, ob es sich tatsächlich um das richtige Programm handelt.

Erneut lassen wir uns das Inhaltsverzeichnis zeigen und löschen nun mit SCRATCH beide Kopien der ersten Sortversion sowie die dazugehörige Arbeitsdatei WORK1 und die Dokumentationsdatei DOK-SORT1, die Beschreibungstexte enthält. Natürlich haben wir uns hierbei auch jeweils vergewissert, daß es sich tatsächlich um die richtigen Dateien handelt.

Sie finden die Vorsicht, die hier angewendet werden soll, vielleicht übertrieben. Aber bedenken Sie, daß der ganze Vorgang nur eine Sache von Minuten ist. Demgegenüber stehen möglicherweise Wochen oder Monate der Programmentwicklung, die sonst verloren sein könnten.

Da wir vermuten, daß sich die beiden Kopien der zweiten Sortversion nur im Namen unterscheiden, wollen wir das überprüfen. Die Version mit dem aktuellen Datum wird mit DLOAD in den Arbeitsspeicher geladen. Diese nun im Speicher befindliche Fassung wird mit der älteren Kopie unter Zuhilfenahme des DVERIFY-Befehls verglichen. Wenn sie übereinstimmen, wird die ältere von beiden mit SCRATCH gelöscht. Die verbleibende Version wird nun noch mit RENAME auf SORT2-aktuelles Tagesdatum in der amerikanischen Schreibweise geändert.

Jetzt kopieren wir nacheinander in alphabetischer Reihenfolge die Dateien auf der Diskette auf eine frisch formatierte, von der wir anschließend noch eine Sicherungskopie mit BACKUP erstellen. Jetzt verfügen wir über zwei Archivfassungen unserer aktuellen Diskettenutilities und eine Arbeitsdiskette für den täglichen Gebrauch.

## Die Systemvariablen

Beim Arbeiten mit Disketten oder auch im Laufwerk können hin und wieder Fehler auftreten, die beispielsweise durch eine fehlerhafte Diskette, ein schadhaftes Gerät, ein lockeres Verbindungskabel, ein fehlerhaftes Hilfsprogramm, eine Fehlbedienung von Ihnen etc. verursacht sein können. Zur Aufklärung von Fehlersituationen stehen Ihnen spezielle Systemvariablen DS, DS$ und ST zur Verfügung, deren Inhalt Ihnen Aufklärung über die Fehlerursache verschaffen kann.

Erhält man beispielsweise nach der Eingabe von

```
PRINT ST
```

das Ergebnis 128, so teilt uns das Ein-/Ausgabe-Statusbyte mit, daß das angesprochene Diskettenlaufwerk nicht da ist. Das kann bedeuten, daß es nicht eingeschaltet ist oder vielleicht das Verbindungskabel locker ist.

Wenn nach der Abfrage mit

```
PRINT DS$
```

die Meldung

```
?DEVICE NOT PRESENT ERROR
```

ausgegeben wird, haben Sie vermutlich vergessen, eine Diskette in das Laufwerk einzulegen oder die Laufwerkklappe herunterzudrücken bzw. den Laufwerkhebel herumzulegen.

Für Fehlersituationen, die durch fehlerhafte Programme, insbesondere aufgrund der Nichtbeachtung von BASIC-Sprachregeln entstehen, können Sie mit Hilfe der dafür bereitgestellten Systemvariablen EL, ER und ERR$ genauere Fehleranalysen vornehmen. Diese Informationen wie Fehlerzeile, Fehlernummer und Fehlertext werden zwar unmittelbar nach Auftreten des Fehlers auf dem Bildschirm angezeigt, können aber bei einer Fehlerbehandlung nach dem TRAP-Befehl auch separat abgefangen werden.

Für Leute, die bei der Arbeit am Computer völlig das Gefühl für Raum und Zeit verloren haben, bietet die eingebaute Uhr, zumindest, was die Zeit betrifft, Abhilfe. Sie kann jederzeit mit

```
PRINT TI$
```

abgefragt werden. Sie sollte allerdings unmittelbar nach dem Einschalten des Rechners mit

```
TI$="HHMMSS"
```

erst einmal vernünftig gestellt werden.

Falls Sie jetzt eine längere Denkpause einlegen wollen, tippen Sie einfach:

```
SCNCLR:DO:PRINT CHR$(19) TI$:LOOP <RETURN>
```

und Ihr Rechner wird zur Wohnzimmeruhr.

# Kapitel 7

# Dateiverwaltung

Wenn wir BASIC-Programme entwickeln, speichern wir unsere Informationen in der Regel in Dateien ab. Für den Computer bedeutet eine Datei dasselbe wie für uns ein Karteikasten-System. Das kann z. B. eine Adreßkartei mit Karteikarten sein, auf denen Informationen von Personen festgehalten sind, insbesondere der Name, die Straße, die Postleitzahl und die Stadt.

Jeder Karteikarte entspricht beim Computer ein Datensatz und jeder Detailinformation ein Feld. Eine Datei im Computersystem könnte dann etwa folgendermaßen aussehen:

Maria Schneider, Feldweg 33, 5300 Bonn
Alfred Schwerte, Parkstr.1a, 4600 Dortmund 2
Fred Langer, Am grünen Stein 12, 2000 Hamburg

Diese Datei umfaßt 3 Datensätze, und zwar den Namen und die zugehörige Adreßinformation für 3 Personen. Jeder Satz besteht wiederum aus 3 Satzfeldern:

NAME, STRASSE, PLZSTADT

Es ist in der Regel für ein BASIC-Programm, das mit den Objekten Satzfeld und Satz arbeitet, notwendig, daß alle Sätze dieselbe Anzahl von Feldern enthalten, da sonst nicht klar entschieden werden kann, im wievielten Satz man sich befindet.

Es gibt bei den Commodore-Rechnern im wesentlichen vier Dateitypen, von denen wir zwei näher betrachten wollen. Dabei gehen wir davon aus, daß Sie mit einem Diskettenlaufwerk arbeiten, da die meisten Dateitypen das ohnehin voraussetzen.

Die vier Dateitypen sind:

- Programmdatei (PRG)
- sequentielle Datei (SEQ)
- Direktzugriffs- oder relative Datei (REL)
- Benutzerdatei (USR)

Im Inhaltsverzeichnis der Diskette gibt die jeweilige Abkürzung hinter den Programmnamen Auskunft darüber, um welche Art von Datei es sich jeweils handelt.

Zwei von diesen Dateitypen wollen wir jedoch im Rahmen der Dateiverwaltung nicht näher erläutern.

## Programmdateien

Eine Programmdatei ist jedes normale BASIC-Programm, das wir erstellen und abspeichern. Dabei stellt die Programmdatei eine besondere Spezialisierung einer sequentiellen Datei dar. Deshalb ist es durchaus möglich, sie wie eine normale sequentielle Datei zu öffnen und zu lesen. Hierbei muß allerdings der etwas abweichende Dateianfang berücksichitgt werden. Außerdem enthält jede BASIC-Zeile Informationen über die vorhergehende und die nachfolgende BASIC-Programmzeile. Das heißt, wir müssen noch die besondere Datenstruktur des BASIC-Programms, die einer doppelt verketteten Liste entspricht, beachten.

Meist wird man das Programm jedoch mit LOAD laden und mit RUN starten. Korrekturen in den Zeilen sind einfach, wie in Kapitel 2 beschrieben, möglich, so daß es selten vorkommt, daß man sie wie eine Datei liest, um das Programm gezielt zu bearbeiten.

Solche Operationen wären beispielsweise bei der Erstellung von Verwendungslisten von Variablen, Befehlen und Unterprogrammen anzuwenden. Eine weitere Möglichkeit ergibt sich bei einer nachträglichen Strukturierung älterer Programme durch Einfügen von Leerzeilen und Einrückungen, wie bei blockorientierten Sprachen erlaubt ist.

## Benutzerdateien

Auf Benutzerdateien werden wir ebenfalls nicht weiter eingehen, da es sich hierbei nicht um ein von BASIC unterstütztes echtes Dateisystem handelt. Man kann zwar seinen eigenen Dateityp aufbauen, aber die gesamte Verwaltung des Lesens und Schreibens muß vollständig vom Benutzerprogramm vorgenommen werden und wird nicht vom System unterstützt. Dieser Dateityp wird deshalb in der Regel nur von sehr anspruchsvollen kommerziellen Programmen, etwa bei einem Datenbankprogramm verwendet.

## Sequentielle Dateien

Sequentielle Dateien heißen so, weil auf ihre Datensätze stets nur fortlaufend vom ersten bis zum letzten zugegriffen werden kann. Da sich bei dieser Daten-

speicherungsform kaum Leerräume zwischen den einzelnen Sätzen auf dem Datenträger befinden, gehen sequentielle Dateien sehr sparsam mit dem zur Verfügung stehenden Platz um.

Außerdem ist es beim Arbeiten mit dem Kassettenrecorder sowieso die einzige mögliche Form der Dateiverwaltung.

Das Dateiende (EOF - End Of File) kann bei einer sequentiellen Datei im Programm über die reservierte Variable ST (siehe dort) abgefragt werden. Wenn ST nämlich den Wert 64 hat, ist das Dateiende erreicht. In der Datei selbst ist das Dateiende mit CHR$(0) gekennzeichnet.

Ein großer Nachteil von sequentiellen Dateien ist allerdings ihre geringe Flexibilität. Gerade wegen der geringen Zwischenräume zwischen den einzelnen Informationen ist es nicht möglich, beispielsweise bei einer Adresse etwa die Angabe Parkstr. durch die längere Angabe Friedrich-Wilhelm-Str. zu ersetzen, weil dafür der Platz nicht ausreicht. Deshalb ist es sinnvoller, für Dateien, bei denen eine häufige Veränderung der Datensätze vorauszusehen ist, Direktzugriffs-Dateien zu verwenden.

## Direktzugriffs-Dateien

Der Name Direktzugriffs-Datei rührt daher, daß die Programme einen einzelnen Datensatz unabhängig von seiner relativen Anordnung innerhalb der Datei direkt auffinden können. Daher ist es nicht notwendig, vor dem Zugriff auf einen Datensatz erst andere Sätze zu lesen.

Direktzugriffs-Dateien haben allerdings den Nachteil, daß sie nicht sehr sparsam mit dem Speicherplatz umgehen. Für jedes Feld und auch für den ganzen Datensatz muß die maximale Größe vorher bestimmt und reserviert werden.

Nehmen wir einmal an, die Satzlänge in einer Direktzugriffs-Datei wäre 100 Bytes; dann wissen wir genau, wo der 9. Satz beginnt, nämlich 800 Bytes vom 1. Satz entfernt.

## Anlegen von sequentiellen und Direktzugriffs-Dateien

Ob eine Datei als sequentielle oder Direktzugriffs-Datei angelegt werden soll, muß schon beim OPEN-Befehl festgelegt werden. Im OPEN-Befehl (siehe dort) werden nämlich alle Charakteristika einer Datei bestimmt. So kann man den Parameter S für sequentielle Dateien und L für Direktzugriffs-Dateien angeben:

```
OPEN 1,8,2,"0:SEQDATEI,S,W"
```
in BASIC 2.0

bzw.

```
DOPEN#1,"SEQDATEI",W
```
in den anderen BASIC-Versionen

öffnet die sequentielle Datei mit dem Namen SEQDATEI auf Laufwerk 0 zum Schreiben (W). In eine sequentielle Datei kann allerdings nur geschrieben werden, wenn diese leer ist, sonst werden bereits bestehende Sätze überschrieben.

Wenn eine Direktzugriffs-Datei geöffnet wird, muß stets die Länge ihrer Datensätze dabei angegeben werden. Zusätzlich zu der im ungünstigsten Fall benötigten Anzahl von Zeichen muß dabei noch ein Zeichen für den Zeilenvorschub (carriage return) kalkuliert werden:

```
OPEN 1,8,2,"DIRDATEI,L,W"+CHR$(51)
```
in BASIC 2.0

bzw.

```
DOPEN#1,"DIRDATEI",L50
```
in den anderen BASIC-Versionen

öffnet eine Direktzugriffs-Datei mit Namen DIRDATEI, um sie neu anzulegen, deren Sätze eine maximale Länge von 50 Zeichen haben werden. Die Satzlänge darf bei einer relativen Datei die maximale Satzlänge von 254 nicht überschreiten.

Wenn eine relative Datei einmal angelegt ist, darf sie fortan nicht mehr mit einer anderen Satzlänge eröffnet werden, als bei der Neuanlage angegeben wurde, sonst gibt es eine Fehlermeldung. Daher kann es sinnvoll sein, diesen Parameter nur zur Eröffnung zu verwenden. Er bietet allerdings auch einen gewissen Schutz vor Irrtümern. Das heißt, Sie sollten eigentlich immer wissen, welche Datei Sie ansprechen wollen und welche Satzlänge diese Datei hat.

Wir wollen nun als erstes eine sequentielle Datei anlegen.

```
100 REM*********************************
110 REM* SEQUENTIELLE DATEI ANLEGEN    *
120 REM*********************************
130 DOPEN#1,"SEQDATEI",W
140 ANZAHL=20
150 PRINT#1,ANZAHL
160 FOR I=1 TO ANZAHL
170 : PRINT#1,I
180 NEXT I
```

```
190 CLOSE 1
200 END
```

Wir haben die Zahl 20 als Angabe dafür, wieviel Daten sich in der Datei SEQDATEI befinden, als erstes auf die Diskette geschrieben und danach die Zahlen 1 bis ANZAHL. Nun wollen wir sehen, ob es auch geklappt hat, und die Datei wieder auslesen:

```
100 REM*********************************
110 REM* SEQUENTIELLE DATEI LESEN      *
120 REM*********************************
130 DOPEN#1,"SEQDATEI",R
140 INPUT#1,ANZAHL
150 FOR I=1 TO ANZAHL
160 : INPUT#1,Z
170 : PRINT Z
180 : IF ST=64 THEN 200
190 NEXT I
200 CLOSE 1
210 END
```

Obwohl wir genau wissen, wieviel Daten in unserer Datei sind, fragen wir das Dateiende in Zeile 180 sicherheitshalber zusätzlich ab.

Nun wollen wir 20 weitere Daten an unsere Datei anhängen. Zu diesem Zweck lesen wir sie erneut und fügen bei Dateiende unsere weiteren Angaben hinzu. Wir werden dabei aber das Vater-Sohn-Prinzip verfolgen, d. h. aus Sicherheitsgründen wird nicht in derselben Datei wieder geschrieben, sondern die Datei wird noch einmal abgeschrieben und dann erweitert.

```
100 REM*********************************
110 REM* SEQUENTIELLE DATEI ERWEITERN  *
120 REM*********************************
130 MEHR=20
140 DOPEN#1,"SEQDATEI",R
150 DOPEN#2,"SEQDATEI2",W
160 INPUT#1,ANZAHL
170 AM=ANZAHL+MEHR
180 PRINT#2,AM
190 FOR I=1 TO ANZAHL
200 : INPUT#1,A
210 : PRINT#2,A
220 : IF ST=64 THEN 240
230 NEXT I
240 FOR I=ANZAHL+1 TO AM
```

```
250 : PRINT#2,I
260 NEXT I
270 CLOSE 1
280 CLOSE 2
290 END
```

Ob das Anhängen von weiteren Daten geklappt hat, erfahren Sie mit demselben Programm, mit dem Sie vorhin die Datei SEQDATEI gelesen haben. Ändern Sie nur den Dateinamen in Zeile 130 in SEQDATEI2 um:

```
130 DOPEN#1,"SEQDATEI2",R
```

Die Programme müssen für BASIC 2.0 so abgewandelt werden, daß die DOPEN-Befehle durch OPEN-Befehle mit entsprechenden Parametern ausgetauscht werden. Beachten Sie noch folgendes: Wenn Sie wiederholt versuchen, in eine Datei gleichen Namens zu schreiben, etwa weil es beim ersten Mal nicht geklappt hat, funktioniert das nur, wenn Sie den Klammeraffen @ vor den Dateinamen setzen. In BASIC 2.0 muß nach dem Klammeraffen noch ein Doppelpunkt stehen:

```
DOPEN#1,"@SEQDATEI",W
```

oder

```
OPEN 1,8,3,"@:SEQDATEI,S,W"
```

Das Anlegen, Lesen und Ändern von Direktzugriffs-Dateien funktioniert ähnlich. Der Hauptunterschied ist jedoch, daß Daten an beliebiger Stelle in der Datei geschrieben oder gelesen werden können. Dazu muß jeweils ein Zeiger positioniert werden. In BASIC 4.0 und 7.0 ist das leicht mit dem Systembefehl RECORD möglich. In den anderen Versionen ist das Auffinden bestimmter Sätze am leichtesten über eine Satznummer möglich, die der Anwender beim Anlegen der Datei zusammen mit dem Datensatz wegschreibt.

## Kapitel 8

# Alphabetische Übersicht über alle BASIC-Schlüsselwörter

Hinter jedem BASIC-Wort sind in Klammern die BASIC-Versionen angegeben, in denen das Schlüsselwort implementiert ist.

BASIC 2.0 - CBM 3xxx, VC20, C64, SX64, C4064
BASIC 4.0 - CBM 4xxx, CBM 8xxx
BASIC 4.0+- CBM 6xx, 7xx
BASIC 3.5 - C16, C116, Plus/4
BASIC 7.0 - C128

Wenn hinter der BASIC-Version ein * steht, bedeutet das, daß der betreffende Befehl nur über den Befehlskanal 15 gesendet werden kann. Beispielsweise kann in BASIC 2.0 der SCRATCH-Befehl nur wie folgt verwendet werden:

```
OPEN 1,8,15
PRINT#1,"SCRATCH:Dateiname"
CLOSE 1
```

Fast jedes Schlüsselwort kann bei der Eingabe abgekürzt werden. Die Abkürzungen stimmen in den einzelnen BASIC-Versionen teilweise nicht überein. Hier sind die Abkürzungen für BASIC 7.0 angegeben.

Der Befehlstyp, der als nächste Angabe folgt, kann entweder sein:

- Kommando: Das ist ein Befehl, der nur im Direktmodus an das System gegeben werden kann.
- Systembefehl: Das ist ein Befehl, der sowohl im Direkt- als auch im Programmodus eingesetzt werden kann und sich direkt an das Betriebssystem wendet.
- Anweisung: Ein Schlüsselwort, das mit Anweisung bezeichnet wird, kann nur im Programmodus verwendet werden.
- Funktion: Einer Funktion folgt immer ein Argument in Klammern. Sie liefert einen Wert als Ergebnis zurück.

Nach der Syntax, die genau die Sprachregeln angibt, nach denen das Kommando, der Befehl, die Anweisung oder die Funktion gebildet werden müssen,

folgt jeweils die Angabe der zulässigen Wertebereiche. Dabei werden unterschieden:
- Parameter: Das sind die Angaben, die nach Kommandos, Befehlen und Anweisungen folgen.
- Argumente: Hierbei handelt es sich um die Werte, die in Klammern hinter den Funktionen angegeben werden.
- Ergebniswerte: Bei jedem Stichwort wird ebenfalls die Bandbreite der Ergebniswerte aufgeführt.

Parameter, Argumente und Werte können sein:

| | |
|---|---|
| – Bitzahlen: | 0 oder 1 |
| – Laufwerknummern: | 0 oder 1 |
| – Farbnummern: | 1 - 16 |
| – Spritenummern: | 1 - 8 |
| – Geräteadressen: | 0 - 15 |
| – Dateinummern: | 0 - 15 |
| – Bytezahlen: | 0 - 255 |
| – Text-Bildschirmkoordinaten: | 0 - 39,0 - 24 |
| bzw. | 0 - 79,0 - 24 |
| – Grafik-Bildschirmkoordinaten: | 0 - 320,0 - 200 |
| bzw. | 0 - 160,0 - 200 |
| – Speicheradressen: | 0 - 65535 |
| – Zeilennummern: | 0 - 63999 |
| – positive, ganze Zahlen: | 0 - 32676 |
| – ganze Zahlen: | -32676 - 32676 |
| – Gleitkommazahlen: | -1.7E38 - 1.7E38 |
| – logische Werte: | -1 oder eine Gleitkommazahl zwischen -3267∂ und +32676 |
| – HEX-Zahlen: | Ziffern 1 - 9 und Buchstaben A - F |
| – Zeichen: | alle CBM-ASCII-Zeichen |
| – Zeichenketten: | 0 - 255 Zeichen |
| – Variablennamen: | Zeichenkette 1 - 255 mit Längeneinschränkungen |
| – Dateinamen: | Zeichenkette 1 - 16 mit Längeneinschränkungen |

Wenn für spezielle Funktionen oder Befehle weitere Einschränkungen bezüglich Parametern, Argumenten oder Werten bestehen, werden diese unter dem entsprechenden Stichwort erwähnt.

## (BASIC 2.0, 4.0, 3.5, 7.0) ABS

**Abkürzung:** A<SHIFT>B

**Typ:** numerische Funktion

**Syntax:**
```
ABS(numerische Konstante)
ABS(numerische Variable)
ABS(numerischer Ausdruck)
```

**Argumentbereich:** Gleitkommazahlen

**Ergebnisbereich:** positive Gleitkommazahlen

Die Funktion ABS liefert den Absolutwert ihres Arguments, d. h. positive Werte bleiben unverändert und negative werden in ihr positives Äquivalent umgewandelt.

### Beispiele

| | |
|---|---|
| `ABS(600)` | ergibt 600 |
| `ABS(-89.4)` | ergibt 89.4 |
| `ABS(7*2-90)` | ergibt 76 |

### Anmerkungen

Die ABS-Funktion kann auch verwendet werden, um das Vorzeichen einer Zahl zu ermitteln. Durch die Division

```
Zahl/ABS(Zahl)
```

ergibt sich für eine positive Zahl der Wert 1 und für eine negative der Wert -1. Selbstverständlich kann für die Ermittlung des Vorzeichens auch die SGN-Funktion genutzt werden (siehe dort).

# APPEND (BASIC 4.0, 7.0)

**Abkürzung:** A<SHIFT>P

**Typ:** Systembefehl für Diskettenlaufwerk

**Syntax:**
```
APPEND#Dateinr.,"Dateiname"
APPEND#Dateinr.,"Dateiname",
DLaufwerknr. ON UGerätenr.
```

**Parameterbereich:** Dateinummer
Dateiname
Laufwerknummer
Gerätenummer

Der Befehl APPEND öffnet die sequentielle Datei mit der logischen Dateinummer, sucht deren Ende und ermöglicht so das Anhängen weiterer Daten an diese Datei.

Die Laufwerk- und Gerätenummer dirigieren den Befehl auf das richtige Laufwerk, falls zwei Laufwerke angeschlossen sind.

### Anmerkung

Der APPEND-Befehl kann in den anderen BASIC-Versionen ziemlich leicht simuliert werden. Man braucht nur die Speicherstellen zu kennen, die die Anfangs- und Endadresse eines Programms enthalten. Beim C64 sind das beispielsweise die Speicherstellen 43 und 44. Die einzige Voraussetzung, damit hinterher ein sinnvolles Programm entsteht, ist, daß die Zeilennummern des zweiten Programms höher sind als die des ersten.

Nachdem das erste Programm geladen ist, schreiben Sie sich seine Anfangs- und Endadresse auf:

```
PRINT PEEK(43)          Anfangsadresse
PRINT PEEK(44)          Endadresse
```

Nun geben Sie ein:

```
POKE 43,(PEEK(45)+256*PEEK(46)-2)AND 255
POKE 44,(PEEK(45)+256*PEEK(46)-2)/256
```

Nun können Sie das zweite Programm in den Speicher laden, und mit den folgenden Befehlen wird das erste davorgespannt:

```
POKE 43,Anfangsadresse
POKE 44,Endadresse
```

Jetzt können Sie mit LIST feststellen, daß sich die beiden Programme hintereinander im Speicher befinden.

# ASC (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** A<SHIFT>S

**Typ:** numerische Funktion

**Syntax:**
```
ASC(Zeichenkette)
ASC(Zeichenkettenvariable)
```

**Argumentbereich:** Zeichenketten

**Ergebnisbereich:** Bytezahl

Die Funktion ASC ermittelt die ASCII-Codezahl des ersten Zeichens der angegebenen Zeichenkette oder des Textes, den die Zeichenkettenvariable vertritt. Sie ist das Gegenstück zur CHR$-Funktion (siehe dort). Die Umkehrfunktion zu ASC ist die CHR$-Funktion, d. h. sie hebt deren Wirkung wieder auf.

Außer in BASIC 7.0 liefert die Abfrage

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

die Fehlermeldung:

```
?ILLEGAL QUANTITY ERROR
```

Nur in BASIC 7.0 wird als Ergebnis der Wert 0 zurückgeliefert.

Diese Möglichkeit erschließen wir uns auch mit unserer Beispielfunktion für die anderen BASIC-Versionen:

```
DEF FNAC(A)=ASC(A$+" ")+32*(LEN(A$)=0)
```

Die Abfrage auf die leere Zeichenkette spielt im Zusammenhang mit dem Lesen von Dateien eine Rolle, daß das Endekennzeichen, CHR$(0), als Leerstring ausgegeben wird. Wir hängen mit unserer Beispielfunktion an jedes Zeichen noch ein Leerzeichen an. Der ASCII-Wert wird dadurch nicht beeinflußt, daß immer nur das erste Zeichen genommen wird. Falls wir aber auf die Zeichenkette der Länge 0 stoßen, wird ihr das Leerzeichen zugeteilt, und die logische Beziehung

```
32*(LEN(A$)=0)
```

liefert nur dann den Wert -32. Das Ergebnis ist 32-32, also 0.

### Beispiel

```
PRINT 255,CHR$(255),ASC(CHR$(255))
```

### Anmerkungen

- Eine ASCII-Codetabelle finden Sie in Anhang G. ASCII... bedeutet American Standard Code for Information Interchange.

- In BASIC 2.0 und 4.0 führt die Eingabe einer leeren Zeichenkette zu einem ILLEGAL QUANTITY ERROR. In BASIC 7.0 ergibt sich dafür der Wert 0. Probieren Sie es aus:

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

Die leere Zeichenkette, die kein Zeichen enthält, ist nicht zu verwechseln mit der Zeichenkette, die ein Leerzeichen enthält:

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

führt in jeder BASIC-Version zu der Ausgabe 32 für das Leerzeichen.

# ATN (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** A<SHIFT>T

**Typ:** trigonometrische Funktion

**Syntax:**
```
ATN(numerische Konstante)
ATN(numerischer Ausdruck)
```

**Argumentbereich:** Gleitkommazahlen

**Ergebnisbereich:** Gleitkommazahlen

Die Funktion ATN liefert den Arcustangens ihres Arguments im Bogenmaß. Der Arcustangens einer Zahl x ist definiert als der Winkel, dessen Tangens gleich x ist.

### Anmerkungen

Um Bogenmaß in Grad umzurechnen, beachten Sie, daß 180 Grad dem Wert $\pi$ im Bogenmaß entsprechen. Also müssen Sie ATN(X) mit 57.29578 multiplizieren, um den Arcustangens auch in Grad zu erhalten.

### Beispiel

```
PRINT "ATN(0.01745506479)";"=";
      ATN(0.01745506479)*57.29578;"GRAD"
```

## (BASIC 3.5, 7.0) AUTO

**Abkürzung:** A<SHIFT>U

**Typ:** Kommmando

**Syntax:**

```
AUTO
```
erzeugt Zeilennummern ab 10 in Zehnersprüngen
(BASIC 3.5)
schaltet Zeilennummernvorgabe ab
(BASIC 7.0)

```
AUTO N
```
erzeugt Zeilennummern ab N in Zehnersprüngen
(BASIC 3.5)
aktiviert Zeilennumerierung mit Schrittweite N
(BASIC 7.0)

```
AUTO N,S
```
erzeugt Zeilennummern ab N in S-Sprüngen
(BASIC 3.5)

**Parameterbereich:** Zeilennummern

**Ergebnisbereich:** Zeilennummern

Der AUTO-Befehl bewirkt bei der Eingabe eines BASIC-Programms eine automatische Vorgabe von Zeilennummern. Der Befehl hat für BASIC 3.5 zwei Parameter, die beide wahlfrei sind. Der erste Parameter gibt die Anfangszeilennummer an; fehlt er, wird bei BASIC 3.5 10 angenommen. Der zweite Parameter gibt die Schrittweite an, mit der die Zeilennummer automatisch erhöht wird. Wird er nicht angegeben, so wird ebenfalls 10 angenommen.

Beim C128 ist höchstens ein Parameter anzugeben, dessen Vorhandensein oder Weglassen die Zeilennummernvorgabe an- oder abschaltet. Die Eingabe der ersten Programmzeilennummer muß manuell vom Programmierer vorgegeben werden. Wird gleich nach der Zeilennummer RETURN gedrückt, so wird der Zeilenvorgabemodus zeitweilig verlassen. Er bleibt jedoch aktiv, bis der Befehl AUTO ohne Parameter eingegeben wird, und ist bei Eingabe einer neuen Zeilennummer sofort wieder verfügbar.

**Beispiel für C128:**

```
Eingabe:        AUTO 10 <RETURN>
Eingabe:        1000 REM <RETURN>
Vorgabe:        1010
Eingabe:        REM <RETURN>
   usw.
Vorgabe:        9010
Eingabe:        REM <RETURN>
Vorgabe:        9020
Eingabe:        <RETURN>
keine Vorgabe
Eingabe:        60100 REM <RETURN>
Vorgabe:        60110
```

**Anmerkung**

AUTO darf nur im Direktmodus angewendet werden.

## (BASIC 4.0, 3.5, 7.0) BACKUP

**Abkürzung:** BA<SHIFT>C

**Typ:** Systembefehl für Diskettenlaufwerk

**Syntax:**

```
BACKUP DLaufwerknummer TO
DLaufwerknummer,UGerätenummer

BACKUP DLaufwerknummer TO
DLaufwerknummer
```

**Parameterbereich:** Laufwerknummern
Gerätenummern

Der BACKUP-Befehl dient dazu, innerhalb eines Doppel-Diskettenlaufwerks der Typen 4040, 8050, 8250 und einer zukünftigen Doppellaufwerk-Ausführung der 1571 den gesamten Inhalt der Quelldiskette, die sich in dem zuerst angegebenen Laufwerk befindet, auf eine Zieldiskette, die in dem hinter TO gekennzeichneten Laufwerk eingelegt ist, vollständig zu duplizieren. Dabei werden die Disketten-Identifikation (ID), der Name, das Inhaltsverzeichnis und alle Daten eins zu eins auf die Zieldiskette übertragen, so daß eine exakte Kopie entsteht.

Das bedeutet natürlich auch, daß alle Dateien, die sich zuvor auf der Zieldiskette befanden, vollständig überschrieben werden.

Mit BACKUP können jedoch keine Duplikate von einer in einem anderen Diskettengerät befindlichen Diskette erstellt werden. Dazu müssen Sie den COPY-Befehl verwenden.

Auch zur Herstellung von Sicherungskopien kommerzieller Software für Ihren persönlichen Bedarf können Sie den BACKUP-Befehl in der Regel nicht verwenden, da das meistens durch Schutzverfahren der Hersteller verhindert wird. Die Erstellung von Sicherungskopien unter Programmkontrolle kann auch mit der DUPLICATE-Anweisung vorgenommen werden.

### Beispiel 1

```
Eingabe:  BACKUP D0 TO D1,U9
Ausgabe:  ARE YOU SURE?
```

Eingabe: Y (für Backup)
jede beliebige andere Eingabe (für Abbruch)

### Anmerkungen

U8 ist die Standardgerätenummer, die weggelassen werden kann. Alle anderen Gerätenummern müssen angegeben werden.

Achten Sie darauf, daß die Zieldiskette leer ist oder überschrieben werden darf.

Die Zieldiskette muß vor dem Backup nicht formatiert werden.

**Beispiel 2** (Verwendung von BACKUP unter Programmkontrolle)

```
100 PRINT"BACKUP, ALLE DATEN AUF DER ZIE
LDISKETTE GEHEN VERLOREN.":PRINT
110 PRINT"LEGEN SIE DIE QUELLDISKETTE IN
 LAUFWERK 1 EIN.":PRINT
120 PRINT"LEGEN SIE DIE ZIELDISKETTE IN
LAUFWERK 0 EIN.":PRINT
130 PRINT"HABEN SIE BEIDE DISKETTEN EING
ELEGT, UND WOLLEN SIE DIE ZIELDISKETTE I
N";
140 PRINT" LAUFWERK D0 WIRKLICH UEBERSCH
REIBEN?":PRINT
150 PRINT"ANTWORTEN SIE MIT J. JEDE ANDE
RE TASTE BRICHT DAS DUPLIZIEREN AB.":PRI
NT
160 GET A$:IF A$="" THEN 160
170 IF A$="J" THEN BACKUP D1 TO D0
180 IF NOT(A$="J") THEN PRINT"DUPLIZIER
EN WIRD NICHT DURCHGEFUEHRT.":PRINT
190 PRINT"WOLLEN SIE NOCH EINMAL DUPLIZI
EREN?"
200 GET A$:IF A$="" THEN 200
210 IF A$="J" THEN 110
220 END
```

## (BASIC 7.0, 4+) BANK

**Abkürzung:** B<SHIFT>A

**Typ:** Speicherverwaltungs-Funktion

**Syntax:** BANK(Nummer)

**Argumentbereich:** Banknummer 0 - 15

Das neue Betriebssystem des C128 kann verschiedene ROM- und RAM-Bänke verwalten. Mit der Anweisung BANK kann jederzeit eine andere 64K-Speicherbank angewählt werden. Das ist jedoch für die Speicherung von BASIC-Programmen und für den Variablenspeicher ohne Bedeutung, daß hierfür stets BANK 0 bzw. 1 automatisch vom Interpreter verwendet werden.

Alle Befehle jedoch, die sich auf eine bestimmte Speicheradresse beziehen, werden beim Umschalten der Speicherbank betroffen.

Für den C128 in der Grundausstattung ist nur BANK 0 und BANK 1 vorhanden. Insgesamt sind jedoch auch beim maximalen Speicherplatzausbau des C128 nur 8 RAM-Speicherbänke möglich, wobei z. Z. nur BANK 0 bis BANK 3 vom BASIC aus durch den BANK-Befehl unterstützt werden und so ein Ansprechen dieser Speicherbereiche ermöglichen, falls Sie Ihren C128 entsprechend hochgerüstet haben.

Dabei bleibt aber der maximale Bereich, der für das eigentliche BASIC-Programm zur Verfügung steht, auf eine 64 KByte große BANK (0) beschränkt, wobei für die Speicherung der Variablen eine weitere BANK (1) Verwendung findet. Die Zuteilung der Bänke für Programm und Daten durch den Interpreter können Sie durch den BANK-Befehl nicht beeinflussen.

Lediglich die Bänke 2 bis 7 können als sogenannte RAM-Disks verwendet werden. Der Name RAM-Disk rührt daher, daöß die Bänke zum schnellen temporären Zwischenspeichern von Daten verwendet werden, die nur während des Programmlaufs benötigt werden oder erst zu einem späteren Zeitpunkt auf die wesentlich langsamere Diskette zur endgültigen Speicherung geschrieben werden.

BASIC 7.0 stellt hierfür neben den Befehlen PEEK und POKE, die schon vom BASIC 2.0 her bekannt sind, zusätzlich die Befehle BSAVE, BLOAD, SWAP und STASH zur Verfügung.

Sie können selbstverständlich die Speicherbänke auch dazu verwenden, um Maschinenprogramme abzulegen, die von dort mit BSAVE auf Diskette gespeichert und mit BLOAD wieder geladen werden können.

**Beispiel**

```
BANK 3
BLOAD "TOOLKIT X,Y
SYS 37139
```

Der Wertebereich 4 bis 15 für den BANK-Parameter wird in der momentanen C128-Implementierung dazu verwendet, um den Zugriff auf verschiedene Kombinationen von RAM und ROM sowie im Adreßraum liegende Register verschiedener peripherer Bausteine zu ermöglichen. Das ist deshalb, notwendig, weil bestimmte Adressen durch mehrere übereinanderliegende ROM- und RAM-Bänke belegt sind. Die Mehrfachbelegung dieser Adressen wird durch das System für den BASIC-Programmierer völlig unbemerkt vorgenommen, solange er nicht explizit bestimmte Systemadressen und -routinen ansprechen will.

Deshalb ist die Information über die Funktion der einzelnen Speicherbänke nur als Hinweis zu verstehen:

BANK 15 dient zur Herstellung des Grundzustands des Systems.

BANK 8 ermöglicht Ihnen den Aufruf mit SYS auf Spracherweiterungen in eventuell angeschlossenen ROM-Modulen. Diese werden mit Sicherheit bald auf dem Markt erscheinen, um z. B. eine bessere Unterstützung für Grafikprogrammierung für den 80-Zeichen-Bildschirm zu gewähren.

**Beispiel**

```
100 BANK 0:POKE 8000,255
110 BANK 1:POKE 8000,100
120 BANK 0:PRINT PEEK(8000)
130 BANK 1:PRINT PEEK(8000)
```

**Anmerkungen**

Die Befehle PEEK, POKE, WAIT und SYS wirken natürlich immer auf die jeweils eingeschaltete Bank.

# (BASIC 7.0) BEGIN

**Abkürzung:** B<SHIFT>E

**Typ:** sekundäres Schlüsselwort (nur in Verbindung mit THEN)

**Syntax:**
```
IF...THEN BEGIN
        Anweisungen
        BEND
      ELSE...
```

Wenn eine ganze Gruppe von Anweisungen in Abhängigkeit von einer IF-Bedingung hinter dem THEN-Zweig ausgeführt werden soll, kann sie durch BEGIN und BEND in einem Block zusammengefaßt werden. Ein Block wird immer mit dem Schlüsselwort BEGIN eingeleitet, gefolgt von einer oder mehreren Programmzeilen und wird abgeschlossen durch das Schlüsselwort BEND (siehe auch BEND).

Ein BEGIN/BEND-Block kann nur nach THEN folgen. Ist die Bedingung hinter IF nicht erfüllt, werden die Anweisungen des THEN-Blocks nicht ausgeführt.

**Beispiel**

```
50000 TRUE = 0
50050 IF TRUE THEN BEGIN
50100 :              PRINT "WAHR"
50150 :              TRUE = 0
50200 :              GOTO 50500 ELSE END
50250 :            BEND
50300 REM      THEN END
50350 :        ELSE
50400 :              PRINT "UNWAHR"
50450 :              TRUE = 1
50500          ELSE END
50550 GOTO 50050
```

Die Zeilen 50200 und 50500 werden vom BASIC-Interpreter toleriert, sind aber in dieser Form nicht Bestandteil von BASIC 7.0. Diese Einfügungen erfolgten, um eine bessere Strukturierung zu bewirken, daß leider für den ELSE-Zweig der IF-Konstruktion die Blockbildung mit BEGIN und BEND nicht erlaubt ist.

### Anmerkung

In BASIC 2.0, 3.5 und 4.0 kann ein Unterprogramm die Funktion der Blockbildung wahrnehmen. Unter dem Stichwort BEND finden Sie ein Beispiel dafür.

## (BASIC 7.0) BEND

**Abkürzung:** BE<SHIFT>N

**Typ:** sekundäres Schlüsselwort (nur in Verbindung mit BEGIN)

**Syntax:**
```
IF...THEN BEGIN
      Anweisungen
      BEND
   ELSE...
```

BEND beendet eine Reihe von Anweisungen, deren Anfang durch den Befehl BEGIN (siehe auch BEGIN) festgelegt wurde. So können nach einer bedingten Verzweigung mehrere Anweisungen zusammengehörig ausgeführt werden.

### Anmerkung

In BASIC 2.0, 3.5 und 4.0 kann ein Unterprogramm die BEGIN/BEND-Struktur des BASIC 7.0 ersetzen. Im folgenden geben wir ein Beispiel dafür.

```
50000 TRUE = 0
50050 IF TRUE THEN GOSUB 60000 TRUE
50350 IF NOT TRUE THEN GOSUB 60300 NOT TRUE
50400 GOTO 50050
60000 REM ***** BEGIN TRUE *****
60050 PRINT "WAHR"
60100 TRUE = 0
60150 RETURN
60200 REM ***** BEND TRUE *****
60300 REM ***** BEGIN NOT TRUE *****
60350 PRINT "UNWAHR"
60400 TRUE = 1
60450 RETURN
60500 REM ***** BEND NOT TRUE *****
```

Beachten Sie bitte, daß in den Zeilen 50050 und 50350 die Wörter TRUE bzw. NOT TRUE nach den jeweiligen Unterprogrammaufrufen mit GOSUB wie Kommentare wirken. Sie werden vom BASIC-Interpreter ignoriert.

## BLOAD (BASIC 7.0)

**Abkürzung:** B<SHIFT>L

**Typ:** Systembefehl für Eingabe

**Syntax:**

```
BLOAD"Dateiname"

BLOAD"Dateiname",DLaufwerknummer,
UGerätenummer,ON BBanknummer,
PStartadresse
```

**Parameterbereich:**
Dateinamen
Laufwerknummern
Gerätenummern
Banknummern 0 - 15
Speicheradressen

Der Befehl BLOAD lädt eine Binärdatei an die Startadresse in der mit Banknummer bezeichneten Bank. Dabei muß der entsprechende Dateiname angegeben werden, während die anderen Parameter unter Berücksichtigung des jeweils voreingestellten Wertes auch weggelassen werden können:

| | |
|---|---|
| Laufwerknummer | Gibt die Nummer des Laufwerks an (0 oder 1), auf dem sich die Datei befindet. Standard ist 0. |
| Gerätenummer | Gibt die Geräteadresse für das Diskettenlaufwerk an (4 bis 15). Standard ist 8. |
| Banknummer | Gibt die Speicherbanknummer an (0 bis 15). Standard ist 0. |
| Startadresse | Bezeichnet die Adresse, an der das Programm in der Bank stehen soll (0 bis 65353). Standardmäßig wird die Adresse genommen, die sich aus den ersten beiden Bytes der zu ladenden Datei ergibt. |

```
BLOAD"DATEI1" ON B1,P8000
```

lädt die Datei mit dem Namen DATEI1 in die Bank mit der Nummer 1 ab Adresse 8000.

```
BLOAD "*
```

lädt das erste Programm auf dem Laufwerk 0 mit der Gerätenummer 8 in die Bank 0.

BLOAD kann verwendet werden, um Maschinenprogramme in die Speicherbänke 2 bis 7, falls vorhanden, zu laden. Auch Daten, die mit STASH, SWAP oder POKE dort abgelegt wurden und mit BSAVE auf eine Diskette gespeichert wurden, lassen sich so wieder laden. Ganze BASIC-Programme können so auf eine RAM-Disk für einen späteren schnellen Zugriff gebracht werden.

BLOAD kann auch als Anweisung in einem Programm enthalten sein. Dabei können für die Parameter Variablen eingesetzt werden, die dann allerdings in Klammern stehen müssen.

### Beispiel

```
20000 PGM$="DATEI1":L=0:N=0
20010 BLOAD (PGM$),D(L),ON B(N)
```

### Anmerkungen

Für Anwender von Rechnern mit BASIC 4.0 mit eingebautem Monitor oder Benutzer des im SYBEX-Assemblerkurs für den C64 enthaltenen Monitors oder eines anderen gleichwertigen Monitors ist das Laden von Binärdateien nur über den Monitor im Direktmodus möglich. Rufen Sie den Monitor auf, und laden Sie Ihre Binärdatei mit L (LOAD). Nach Verlassen des Monitors steht Ihnen die Datei dann zur weiteren Bearbeitung zur Verfügung.

Auch im C128-Maschinensprachemonitor gibt es den L-Befehl.

Wenn Sie über ein schnelles Laufwerk wie die SFD 1001 mit IEE-Bus oder die Harddisks 9060 und 9090 mit IEEE-Bus oder über die wesentlich schnellere RAM-Disk (für C128) verfügen, können Sie das Programmbeispiel unter BSAVE als Grundidee für Animationen mit sehr schnell nacheinander geladenen Bildschirmbildern nehmen. Ihrem ersten Trickfilm steht dann nichts mehr im Wege.

BLOAD bewirkt im Gegensatz zu DLOAD kein Löschen des Variablenbereichs und des im Speicher befindlichen Programms. Sie können also beim Nachladen von Programmen Variablen übergeben und einen entsprechenden Programmaufbau mit nachzuladenden Unterprogrammen vorsehen (Overlay-Technik).

# BOOT (BASIC 7.0)

**Abkürzung:** B<SHIFT>O

**Typ:** Systembefehl für Eingabe

**Syntax:**
```
BOOT

BOOT"Dateiname",DLaufwerknummer,
UGerätenummer
```

**Parameterbereich:** Dateinamen
Laufwerknummern
Gerätenummern

Der Befehl BOOT lädt und startet ein Programm von der Diskette. Dabei können alle Parameter weggelassen werden, dann wird die Information, welches Programm geladen werden soll, dem Sektor 0, Spur 1 entnommen. Diese Information kann nur mit Direktzugriffsbefehlen auf die Diskette gebracht werden.

Man kann aber auch den gewünschten Dateinamen, Laufwerknummer (Standard ist 0) und Gerätenummer (Standard ist 8) explizit angeben.

### Beispiel

Legen Sie die CP/M-Diskette in das Diskettenlaufwerk 1541 oder 1571, und geben Sie BOOT <RETURN> ein. Das CP/M-Betriebssystem wird in Ihren C128 geladen und automatisch gestartet.

Der BOOT-Befehl kann auch innerhalb von Programmen eingesetzt werden. Das kann z. B. sinnvoll sein, wenn Sie vom dem Laden und Starten eines kommerziellen Programms noch einige Voreinstellungen vornehmen wollen, die den Drucker oder einige wichtige Systemvariablen betreffen. Schreiben Sie sich dafür ein kleines BASIC-Programm als Startprozedur und schließen diese mit dem BOOT-Befehl für das kommerzielle Programm ab. Auf diese Art und Weise können Sie Parameter verändern, die aus dem gekauften Software-Paket heraus nicht zu erreichen sind.

## (BASIC 3.5, 7.0) BOX

**Abkürzung:** keine möglich

**Typ:** Grafikbefehl

**Syntax:**
```
BOX F,X1,Y1,X2,Y2,Winkel,Farbe
BOX ,X1,Y1,X2,Y2
```

**Parameterbereich:** F - ganze Zahlen von 0 - 3
Grafik-Bildschirmkoordinaten
Winkel - Gleitkommazahlen von 0 - 360
Farbe - Bitzahlen

Mit dem BOX-Befehl läßt sich ein Rechteck in beliebiger Größe und Drehung darstellen. Folgende Parameter können dabei angegeben werden:

| | |
|---|---|
| F | Farbquelle (kann entfallen)<br>0 - Hintergrund<br>1 - Vordergrund<br>2 - Mehrfarbenmodus 1<br>3 - Mehrfarbenmodus 2 |
| X1,Y1 | Zeile, Spalte oben links |
| X2,Y2 | Zeile, Spalte unten rechts |
| Winkel | Drehwinkel in Grad (kann entfallen, dann 0) |
| Farbe | Rechteck mit Farbe füllen (kann entfallen, dann 0)<br>0 - nein<br>1 - ja |

Dem BOX-Befehl wird das Koordinatensystem aus Abb. 8.1 zugrundegelegt, wobei sich der Nullpunkt aus mathematischer Sicht ungewohnt in der linken oberen Ecke befindet.

### Beispiel

```
100 GRAPHIC 2,1
110 BOX 1,120,120,128,30,45,1
120 BOX 1,120,120,128,30,135,1
```

Ein weiteres Beispiel für den C128 finden Sie unter dem Stichwort SPRITE.

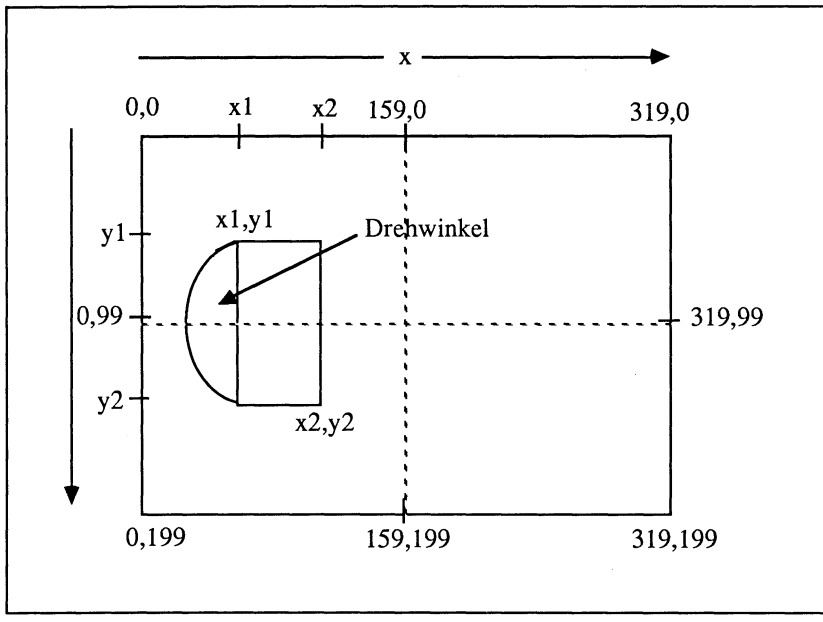



*Abb. 8.1: Das Koordinatensystem der Commodore-Rechner und der BOX-Befehl*

### Anmerkung

BOX wirkt in der derzeitigen Implementierung nur auf dem 40-Zeichen-Bildschirm.

# (BASIC 7.0) BSAVE

**Abkürzung:** B<SHIFT>S

**Typ:** Systembefehl für Ausgabe

**Syntax:**

```
BSAVE"Dateiname"

BSAVE"Dateiname",DLaufwerknummer,
UGerätenummer,ON BBanknummer,
PStartadresse TO PEndadresse
```

**Parameterbereich:** Dateinamen
Laufwerknummern
Gerätenummern
Banknummern 0 - 15
Speicheradressen

Der Befehl BSAVE speichert eine Binärdatei unter dem angegebenen Dateinamen ab. Dabei können auch die Banknummer (0 bis 15, Standard ist 0) und Start- bis Endadresse (0 bis 65535) angegeben werden.

Wenn Sie mit zwei Laufwerken arbeiten, können Sie auch die Laufwerknummer 0 oder 1 (Standard ist 0) eingeben. Ebenfalls kann wahlweise die Gerätenummer (4 bis 15) spezifiziert werden (Standard ist 8).

**Beispiele**

```
BSAVE"DATEI2",B1,P8500 TO P9000
```

speichert den Bereich von Adresse 8500 bis 9000 aus der Bank mit der Nummer 1 auf Diskette als Binärdatei unter dem Namen DATEI2 ab.

```
BSAVE"VIDEO",ON B0,P(DEC(0400)) TO P(DEC(07FF))
BSAVE"COLOR",ON B0,P(DEC(D800)) TO P(DEC(DBFF))
```

speichert ein Bild im 40-Zeichen-Modus für den C128 ab.

```
10000 SCNCLR:PRINT TAB(15)"TESTBILD"
10010 SLEEP2
10020 BSAVE"@TESTBILD",ON B0,P1024 TO P2024
```

```
10030 FOR I=1 TO 10
10040 : BLOAD"TESTBILD",ON B0,P1024+I*40
10050 NEXT I
10060 GETKEY T$
10070 END
```

### Anmerkungen

Der Befehl BSAVE kann auch als Programmbefehl eingesetzt werden.

Alle Parameter können als Variable angegeben werden, müssen aber dann in Klammern gesetzt werden.

Für Programmierer von BASIC 4.0 mit eingebautem Monitor oder C64-Benutzer, die mit dem Monitor des Assemblerkurses von SYBEX oder einem gleichwertigen arbeiten, ist es möglich, Binärdateien mit Hilfe des Befehls S (SAVE) aus dem Monitor heraus abzuspeichern. Dies kann allerdings nur im Direkt- und nicht im Programm-Modus geschehen.

Der im C128 vorhandene Maschinensprache-Monitor verfügt ebenfalls über den S-Befehl.

# (BASIC 7.0) BUMP

**Abkürzung:** B<SHIFT>U

**Typ:** Sprite-Funktion

**Syntax:**
```
BUMP(1)
BUMP(2)
```

**Argumentbereich:** die ganzen Zahlen 1 und 2
**Ergebnisbereich:** Bytezahl

Der Befehl BUMP fragt nach einer Sprite-Kollision ab, welche Sprites zusammengestoßen sind (siehe auch COLLISION).

Der Wert von BUMP(1) gibt Auskunft darüber, welche Sprites zusammengestoßen sind. Dabei wird der Wert nach den gesetzten Bits wie folgt berechnet:

```
SPRITE8 SPRITE7 SPRITE6 SPRITE5 SPRITE4 SPRITE3 SPRITE2 SPRITE1

   128     64      32      16       8       4       2       1
```

Liefert BUMP(1) den Wert 48, so bedeutet das, daß SPRITE6 und SPRITE5 miteinander kollidiert sind, da sich 48 aus dem Wert 32 - für SPRITE6 - plus 16 - für SPRITE5 - zusammensetzt.

Der Wert von BUMP(2) gibt nach dem gleichen Schema Auskunft darüber, ob eine Sprite-Kollision mit dem Hintergrund stattgefunden hat. Ergibt sich hier beispielsweise der Wert 74, so steht fest, daß SPRITE7, SPRITE4 und SPRITE2 mit einem Hintergrundzeichen zusammengestoßen sind.

## Beispiel

Die folgenden Zeilen testen, ob ein Sprite mit einem anderen zusammengestoßen ist, und geben dann eine entsprechende Meldung aus.

```
100 GRAPHIC 2,1,15
200 FOR SP=1 TO 8
210 IF (BUMP(1) AND 2↑(SP-1))=2↑(SP-1) THEN
    PRINT "SPRITE";SP;"IST KOLLIDIERT"
```

```
220 NEXT SP
230 END
```

In Zeile 100 wird der Bildschirm in den Text/Grafik-Modus umgeschaltet, wobei für den auszugebenden Text die unteren Zeilen ab Zeile 15 vorgesehen sind. In der Schleife ab Zeile 200 werden dann die 8 Sprites der Reihe nach durch Vergleichen des Bitmusters, das BUMP(1) liefert, auf Kollision überprüft, und entsprechend wird ein Text ausgegeben.

Um die Routine testen zu können, müssen Sie erst einmal ein paar Sprites definieren. Rufen Sie SPRDEF auf (siehe auch dort). Das Programm, das Sie vielleicht schon in den Arbeitsspeicher eingetippt haben, geht Ihnen dabei nicht verloren. Wählen Sie eine Sprite-Nummer, und geben Sie mit den Tasten 1 (Punkt löschen) und 2 (Punkt setzen) ein beliebiges Muster ein. Nach dem gleichzeitigen Drücken von SHIFT und RETURN können Sie ein weiteres Sprite erstellen. Wenn Sie noch einmal RETURN drücken, ist der Definitionsvorgang beendet.

Die Sprites, die Sie so definiert haben, wollen wir jetzt in unser Programm einbauen. Wir wählen z. B. die Sprites mit der Nummer 1 und 6 und aktivieren sie mit dem SPRITE-Befehl.

```
110 SPRITE 1,1
120 SPRITE 6,1
```

Da Sprite 1 und Sprite 6 auf dem Bildschirm an dieselbe Stelle, nämlich rechts oben, gesetzt werden, muß das Programm das als Kollision werten und meldet:

```
SPRITE 1 IST KOLLIDIERT
SPRITE 6 IST KOLLIDIERT
```

Falls man nach einer Kollision von Sprites in Abhängigkeit von den betroffenen Sprites eine besondere Programmroutine in Gang setzen will, empfiehlt sich eine Mehrfachverzweigung. Diesmal wollen wir als Beispiel eine Sprite/Hintergrund-Kollision mit BUMP(2) überprüfen.

```
100 SPRITE 5,1
110 A=BUMP(2):B=0
120 IF A<1 THEN 140
130 A=A/2:B=B+1:GOTO 120
140 ON B GOTO 1000,2000,3000,4000,5000,6000,7000,8000:
    GOTO 9000
1000 PRINT "SPRITE 1 IST KOLLIDIERT":GOTO 9010
2000 PRINT "SPRITE 2 IST KOLLIDIERT":GOTO 9010
3000 PRINT "SPRITE 3 IST KOLLIDIERT":GOTO 9010
4000 PRINT "SPRITE 4 IST KOLLIDIERT":GOTO 9010
5000 PRINT "SPRITE 5 IST KOLLIDIERT":GOTO 9010
```

```
6000 PRINT "SPRITE 6 IST KOLLIDIERT":GOTO 9010
7000 PRINT "SPRITE 7 IST KOLLIDIERT":GOTO 9010
8000 PRINT "SPRITE 8 IST KOLLIDIERT":GOTO 9010
9000 PRINT "ES HAT KEINE KOLLISION STATTGEFUNDEN"
9010 END
```

In Zeile 100 wird Sprite 5 zu Testzwecken aktiviert. Sie können hier selbstverständlich auch jedes andere Sprite einsetzen. Auf eine Umschaltung in den Text/Grafik-Modus wurde diesmal bewußt verzichtet, da so automatisch eine Kollision zwischen dem Programmlisting auf Ihrem Textbildschirm als Hintergrundzeichen und dem Sprite in der rechten oberen Ecke erzielt wird.

Geben Sie nun RUN ein. Jedesmal, wenn eine Programmzeile so lang ist, daß das SPRITE, das in der oberen rechten Ecke erscheint, davon berührt wird, erscheint die Meldung "SPRITE 5 IST KOLLIDIERT". Geben Sie ein paarmal hintereinander RUN ein. Wenn das Programmlisting nach etwa 5 Läufen oben aus dem Bildschirm herausgeschoben wurde, wird nur noch die Meldung "ES HAT KEINE KOLLISION STATTGEFUNDEN" gedruckt. Diese Meldung ist ihrerseits wieder so lang, daß sie eine Kollision mit dem Sprite bewirkt, wenn sie in seinen Bereich kommt.

# CATALOG (BASIC 4.0, 7.0)

**Abkürzung:** C<SHIFT>A

**Typ:** Systembefehl für Diskettenlaufwerk

**Syntax:**

```
CATALOG

CATALOG DLaufwerknummer

CATALOG DLaufwerknummer,
UGerätenummer,"Dateiname"

CATALOG "Dateiname"
```

**Parameterbereich:** Laufwerknummern
Gerätenummern
Dateinamen

Der Befehl CATALOG ohne Parameter gibt das Inhaltsverzeichnis einer Diskette auf dem Bildschirm aus. Wenn zwei Laufwerke angeschlossen sind, wird das Inhaltsverzeichnis von beiden hintereinander angezeigt. Möchte man nur eins sehen, muß man die Laufwerknummer angeben:

```
CATALOG D1
```

gibt das Inhaltsverzeichnis des zweiten Laufwerks aus (das erste wäre D0).

Der Dateiname kann Platzhalter (Joker) enthalten. Dabei dient das Fragezeichen als Ersatz für einzelne Buchstaben mitten im Namen, der Stern * ersetzt den letzten Teil eines Namens.

```
CATALOG D0,U8,"?ROG*"
```

würde z. B. die folgenden Einträge anzeigen:

```
PROG1
PROG2
PROGRAMM
FROGGER
```

# (BASIC 3.5, 7.0) CHAR

**Abkürzung:** CH<SHIFT>A

**Typ:** Grafikbefehl

**Syntax:**
```
CHAR F,X,Y,"Text",1
CHAR ,X,Y,"Text"
```

**Parameterbereich:** F - ganze Zahlen 0 - 3
Text-Bildschirmkoordinaten
Zeichenketten
Bitzahlen

**Ergebnisbereich:** Zeichenketten

Mit der Anweisung CHAR läßt sich ein Text innerhalb des aktuellen Bildschirmfensters an einer frei wählbaren Stelle im Textmodus oder in eine HIRES-Grafik einfügen. Dabei muß die Position dieses Textes genau angegeben werden:

| | |
|---|---|
| F | Farbquelle (kann entfallen, dann 1)<br>0 - Hintergrund<br>1 - Vordergrund<br>2 - Mehrfarbenmodus 1<br>3 - Mehrfarbenmodus 2 |
| X | Spaltennummer (max. 39 bzw. 79) |
| Y | Zeilennummer (max. 24) |
| Text | auszugebender Text |
| 1 | Text soll revers dargestellt werden (kann entfallen, dann normale Darstellung) |

Der Vorteil von CHAR gegenüber dem PRINT-Befehl liegt darin, daß auch nachträglich an Positionen gedruckt werden kann, die vor der aktuellen Cursorposition liegen oder weiter, als mit PRINT TAB(255) möglich ist, nach vorn gesprungen werden kann.

## Beispiel

Im Textmodus ist es wie auch bei PRINT-Anweisungen möglich, Steuerzeichen in den Ausgabetext einzufügen, die dann bei der Ausgabe auf dem Bildschirm wirksam werden.

```
100 KLINGEL$=CHR$(7)
110 FEHLER$=KLINGEL$+"EINGABEFEHLER"+KLINGEL$
120 CHAR 1,12,13,FEHLER$,1
```

Löschen Sie zunächst den Bildschirm mit SHIFT/CLR, und tippen Sie dann RUN.

Ein weiteres Beispiel für den Einsatz von CHAR finden Sie unter dem Stichwort SPRITE.

### Anmerkungen

Die CHAR-Anweisung entspricht dem bekannten PRINT AT anderer BASIC-Versionen und Spracherweiterungen für Commodore-Rechner.

Die CHAR-Funktion kann in BASIC 2.0 und 4.0 wie folgt simuliert werden:

```
100 CF$=CHR$(19)    :REM CURSOR HOME
110 CU$(1)=CHR$(17):REM CURSOR TIEF
120 FOR I=2 TO 24
130 : CU$(I)=CU$(I-1)+CU$(1)
140 NEXT I
145 STOP
150 A$="TEST":Z=10:S=19:GOSUB 62000
62000 PRINT CF$;CU$(Z);TAB(S)A$
62010 RETURN
```

Dabei geben in Zeile 150 die Variablen Z und S die gewünschte Zeilen- bzw. Spaltennummer an. Diese Werte können Sie für Ihre eigenen Zwecke verändern.

## (BASIC 2.0, 4.0, 3.5, 7.0) CHR$

**Abkürzung:** C<SHIFT>H

**Typ:** Zeichenkettenfunktion

**Syntax:**
```
CHR$(numerische Konstante)
CHR$(numerischer Ausdruck)
```

**Argumentbereich:** Bytezahl

**Ergebnisbereich:** Zeichen

Das Argument der Funktion CHR$ muß zwischen 0 und 255 liegen. Als Ergebnis erhalten wir das Zeichen mit dem entsprechenden ASCII-Code. Die CHR$-Funktion erschließt uns die volle Stärke des Commodore-Zeichensatzes, der leider aufgrund seiner etwas unübersichtlichen Codierung bisher nur wenig erschöpft wurde.

Die wichtigste Anwendung der CHR$-Funktion liegt im Bereich der Steuercodes für die PRINT- und CHAR-Anweisungen. Diese sollten der besseren Lesbarkeit willen, und um die Eingabe zu vereinfachen, stets mit Hilfe der CHR$-Funktion und gut lesbaren Namen zu Beginn eines jeden Programms zugewiesen werden. Dies ermöglicht außerdem eine vollständige Ausgabe auf Typenraddruckern und Nicht-Commodore-Druckern.

Der Autor schlägt die folgenden Festlegungen vor. Alle Befehle gelten immer innerhalb des aktuellen Bildschirmfensters. Falls kein Fenster definiert wurde, ist stets der ganze Bildschirm angesprochen.

### Steuerzeichen für den Cursor

| | |
|---|---|
| `CS$=CHR$(19)+CHR$(19)` | Cursor in linke obere Ecke des Bildschirms setzen, alle Bildschirmfenster werden aufgehoben |
| `CF$=CHR$(19)` | Cursor in linke obere Ecke des Bildschirmfensters setzen |

```
CA$=CHR$(27)+"J"
```

Cursor an den Beginn der aktuellen Zeile setzen

```
CN$=CHR$(141)
```

Cursor an den Beginn der folgenden Zeile setzen

```
DIM CL$(79)
CL$(1)=CHR$(157)
FOR I=2 TO 78
CL$(I)=CL$(I-1)+CL$(1)
NEXTI
```

Cursor um (X) Stellen nach links stellen

```
DIM CR$(78)
CR$(1)=CHR$(29)
FOR I=2 TO 78
CR$(I)=CR$(I-1)+CR$(1)
NEXTI
```

Cursor um (X) Stellen nach rechts stellen

```
DIM CH$(24)
CH$(1)=CHR$(145)
FOR I=2 TO 24
CH$(I)=CH$(I-1)+CH$(1)
NEXTI
```

Cursor um (X) Stellen nach oben stellen

```
DIM CU$(24)
CU$(1)=CHR$(17)
FOR I=2 TO 24
CU$(I)=CU$(I-1)+CU$(1)
NEXTI
```

Cursor um (X) Stellen nach unten stellen

```
CE$=CHR$(27)+"K"
```

Cursor neben das letzte gedruckte Zeichen der aktuellen Zeile stellen

```
CK$=CHR$(27)+"E"
```

Cursor-Blinken abstellen

```
C_$=CHR$(27)+"U"
```

Cursor auf Strichdarstellung umschalten (nur im 80-Zeichen-Modus)

```
CV$=CHR$(27)+"S"
```

Cursor auf Blockdarstellung zurückschalten

```
CB$=CHR$(27)+"F"
```

Cursor-Blinken anstellen

```
CT$=CHR$(9)
```
Cursor auf nächste Tabulatorposition stellen

### Tabulator-Steuerzeichen

```
TS$=CHR$(24)
```
Tabulatorstop an der aktuellen Cursorposition setzen oder löschen

```
TD$=CHR$(27)+"Z"
```
Alle Tabulatorstop löschen

```
TE$=CHR$(27)+"Y"
```
Tabulatorstops wieder setzen (alle 8 Spalten)

### Steuerzeichen zum Löschen

```
DIM DZ$(23)
DZ$(1)=CHR$(27)+"D"
FOR I=2 TO 23
DZ$(I)=DZ$(I-1)+DZ$(1)
NEXTI
```
Zeilen unterhalb der Cursorposition inclusive der aktuellen Zeile werden gelöscht

```
DR$=CHR$(27)+"@"
```
Alle Zeilen unterhalb und inclusive der aktuellen Zeile werden gelöscht

```
DF$=CHR$(147)
```
Das ganze Bildschirmfenster wird gelöscht

```
DS$=CS$+LF$
```
Der ganze Bildschirm wird gelöscht, und alle Fenster sind aufgehoben

```
DA$=CHR$(27)+"P"
```
Die aktuelle Zeile wird vom Beginn bis zur Cursorposition gelöscht

```
DE$=CHR$(27)+"Q"
```
Die aktuelle Zeile wird von der Cursorposition bis zum Ende gelöscht

### Bildschirm-Steuerzeichen

```
SB$=CHR$(15)
```
Zeichen auf dem 80-Zeichen-Schirm werden blinkend dargestellt

| | |
|---|---|
| `SV$=CHR$(11)` | Das Umschalten des Zeichenmodus wird unterbunden |
| `SZ$=CHR$(12)` | Das Umschalten des Zeichenmodus wird wieder zugelassen |
| `SI$=CHR$(27)+"I"` | Eine neue Zeile oberhalb der aktuellen Zeile wird eingefügt, wobei die restlichen Zeilen nach unten gerollt werden |
| `SM$=CHR$(27)+"M"` | Bildschirmrollen wird abgeschaltet |
| `SL$=CHR$(27)+"L"` | Bildschirmrollen wird eingeschaltet |
| `SA$=CHR$(27)+"H"` | Das Auslösen des Signaltons wird unterbunden |
| `SP$=CHR$(27)+"G"` | Das Auslösen des Signaltons wird zugelassen |
| `SW$=CHR$(7)` | Der Signalton wird ausgelöst |
| `SO$=CHR$(27)+"O"` | Grundzustand bei aktiviertem Einfüge-, Invers- und Anführungszeichen-Modus wird wieder hergestellt |
| `SX$=CHR$(27)+"X"` | 80-Zeichen-Modus wird auf 40 Zeichen umgeschaltet und umgekehrt |
| `SR$=CHR$(27)+"R"` | Bildschirm wird auf Inversdarstellung umgeschaltet (nur im 80-Zeichen-Modus) |
| `SN$=CHR$(27)+"N"` | Inversdarstellung wird auf Normal umgeschaltet (nur im 80-Zeichen-Modus) |
| `SH$=CHR$(27)+"V"` | Bildschirminhalt wird um eine Zeile oben gerollt |
| `SU$=CHR$(27)+"W"` | Bildschirminhalt wird um eine Zeile nach unten gerollt |

| | |
|---|---|
| `SE$=CHR$(27)+"A"` | Der automatische Einfügemodus wird eingeschaltet |
| `SS$=CHR$(27)+"C"` | Der automatische Einfügemodus wird ausgeschaltet |
| `SK$=CHR$(14)` | Bildschirm wird auf Klein-/Großschrift umgeschaltet |
| `SG$=CHR$(142)` | Bildschirm wird auf Großschrift/Grafik umgeschaltet |

**Steuerzeichen für Fenster**

| | |
|---|---|
| `FO$=CHR$(27)+"T"` | Linke obere Ecke des Bildschirmfensters wird an der Cursorposition festgelegt |
| `FU$=CHR$(27)+"B"` | Rechte untere Ecke des Bildschirmfensters wird an der Cursorposition festgelegt |

**Steuerzeichen für Zeichen**

| | |
|---|---|
| `ZR$=CHR$(18)` | Inversdarstellung für Zeichen wird eingeschaltet |

*Achtung:* Wenn Sie für die Definition eines Grafikzeichens die Inversdarstellung einschalten, müssen Sie sie danach unbedingt wieder ausschalten, da sie so lange gilt, bis sie wieder abgeschaltet wird.

| | |
|---|---|
| `ZN$=CHR$(146)` | Inversdarstellung für Zeichen wird abgeschaltet |
| `ZU$=CHR$(2)` | Unterstreichungsmodus für Zeichen wird eingeschaltet (nur im 80-Zeichen-Modus) |
| `ZB$=CHR$(15)` | Blinkdarstellung für Zeichen wird eingeschaltet (nur im 80-Zeichen-Modus) |

```
DIM ZF$(16)                    Zeichenfarbe (X) wird eingeschaltet
ZF$(1)=CHR$(144)               Schwarz
ZF$(2)=CHR$(5)                 Weiß
ZF$(3)=CHR$(28)                Rot
ZF$(4)=CHR$(159)               Türkis
ZF$(5)=CHR$(156)               Violett
ZF$(6)=CHR$(30)                Grün
ZF$(7)=CHR$(31)                Blau
ZF$(8)=CHR$(158)               Gelb
ZF$(9)=CHR$(129)               Orange
ZF$(10)=CHR$(149)              Braun
ZF$(11)=CHR$(150)              Hellrot
ZF$(12)=CHR$(151)              Dunkelgrau
ZF$(13)=CHR$(152)              Mittelgrau
ZF$(14)=CHR$(153)              Hellgrün
ZF$(15)=CHR$(154)              Hellblau
ZF$(16)=CHR$(155)              Hellgrau
```

## Semigrafik-Steuerzeichen

### Steuerzeichen für Symbole

```
DIM SM$(3)                     Symbole
SM$(1)=CHR$(113)               Ausgefüllter Kreis
SM$(2)=CHR$(119)               Leerer Kreis
SM$(3)=CHR$(122)               Raute
```

### Steuerzeichen für Rechtecke

```
DIM GR$(4)                     Blockgrafikzeichen für Rechtecke und
                               Gitter
GR$(1)=CHR$(176)               Linkes oberes Rechtecksegment
GR$(2)=CHR$(174)               Rechtes oberes Rechtecksegment
GR$(3)=CHR$(173)               Linkes unteres Rechtecksegment
GR$(4)=CHR$(189)               Rechtes unteres Rechtecksegment
```

### Steuerzeichen für Kreise

```
DIM GK$(4)                     Blockgrafikzeichen für Kreis und abge-
                               rundete Rechtecke
```

```
GK$(1)=CHR$(117)
GK$(2)=CHR$(105)
GK$(3)=CHR$(106)
GK$(4)=CHR$(107)
```

Linkes oberes Kreissegment
Rechtes oberes Kreissegment
Linkes unteres Kreissegment
Rechtes unteres Kreissegment

### Steuerzeichen für Verbindungen von Kreis- und Rechteckelementen

```
DIM GV$(7)
GV$(1)=CHR$(96)
GV$(2)=CHR$(98)
GV$(3)=CHR$(177)
GV$(4)=CHR$(171)
GV$(5)=CHR$(178)
GV$(6)=CHR$(179)
GV$(7)=CHR$(123)
```

Verbindungsstücke für Kreis- und Rechteckelemente
Einfache horizontale Verbindung
Einfache vertikale Verbindung
Horizontale Verbindung mit Mittelstrich nach oben
Vertikale Verbindung mit Mittelstrich nach rechts
Horizontale Verbindung mit Mittelstrich nach unten
Vertikale Verbindung mit Mittelstrich nach links
Kreuzungselement

### Steuerzeichen für Kantenlinien

```
DIM RK$(8)
RK$(1)=CHR$(111)
RK$(2)=CHR$(163)
RK$(3)=CHR$(112)
RK$(4)=CHR$(167)
RK$(5)=CHR$(186)
RK$(6)=CHR$(164)
RK$(7)=CHR$(108)
RK$(8)=CHR$(165)
```

Kantenorientierte Rechteckelemente
Linkes oberes Eckelement
Horizontale Verbindung längs der oberen Zeichenkante
Rechtes oberes Eckelement
Vertikale Verbindung längs der rechten Zeichenkante
Rechtes unteres Eckelement
Horizontale Verbindung längs der unteren Zeichenkante
Linkes unteres Eckelement
Vertikale Verbindung längs der linken Zeichenkante

### Steuerzeichen für horizontale Linien

```
DIM LH$(8)
LH$(1)=CHR$(163)
```

Horizontale Linien in Stufen
Waagerechte Linie ganz oben

```
LH$(2)=CHR$(101)      Eine Strichstärke darunter
LH$(3)=CHR$(100)      Zwei Strichstärken darunter
LH$(4)=CHR$(99)       Drei Strichstärken darunter
LH$(5)=CHR$(96)       Drei Strichstärken darunter
LH$(6)=CHR$(102)      Vier Strichstärken darunter
LH$(7)=CHR$(114)      Fünf Strichstärken darunter
LH$(8)=CHR$(164)      Waagerechte Linie ganz unten
```

### Steuerzeichen für schräge Linien

```
DIM LD$(3)            Schräge Linien
LD$(1)=CHR$(109)      Diagonale Linie von links oben nach
                      rechts unten
LD$(2)=CHR$(110)      Diagonale Linie von rechts oben nach
                      links unten
LD$(3)=CHR$(118)      Diagonales Kreuzungselement
```

### Steuerzeichen für vertikale Linien

```
DIM LV$(8)            Vertikale Linien in Stufen
LV$(1)=CHR$(165)      Senkrechte Linie ganz links
LV$(2)=CHR$(116)      Eine Strichstärke rechts daneben
LV$(3)=CHR$(103)      Zwei Strichstärken rechts daneben
LV$(4)=CHR$(98)       Drei Strichstärken rechts daneben
LV$(5)=CHR$(125)      Drei Strichstärken rechts daneben
LV$(6)=CHR$(104)      Vier Strichstärken rechts daneben
LV$(7)=CHR$(121)      Fünf Strichstärken rechts daneben
LV$(8)=CHR$(167)      Senkrechte Linie ganz rechts
```

### Steuerzeichen für horizontale Balken

```
DIM BH$(16)           Horizontale Balkenelemente
BH$(1)=ZR$+CHR$(32)+ZN$     Voller Block
BH$(2)=ZR$+CHR$(163)+ZN$:   Voller Block bis auf 1 Linie oben
BH$(3)=ZR$+CHR$(183)+ZN$:   Voller Block bis auf 2 Linien oben
BH$(4)=ZR$+CHR$(184)+ZN$:   Voller Block bis auf 3 Linien oben
BH$(5)=CHR$(162)      Voller Block bis auf 4 Linien oben
BH$(6)=CHR$(185)      Voller Block bis auf 5 Linien oben
BH$(7)=CHR$(175)      Voller Block bis auf 6 Linien oben
BH$(8)=CHR$(164)      REM Linie entlang der unteren Kante
```

```
BH$(16)=CHR$(163)
```
Linie entlang der oberen Kante (für negative Darstellung)

```
BH$(15)=CHR$(183)
```
2 Linien entlang der oberen Kante (für negative Darstellung)

```
BH$(14)=CHR$(184)
```
3 Linien entlang der oberen Kante (für negative Darstellung)

```
BH$(13)=ZR$+BH$(5)+ZN$
```
4 Linien entlang der oberen Kante (für negative Darstellung)

```
BH$(12)=ZR$+BH$(6)+ZN$
```
5 Linien entlang der oberen Kante (für negative Darstellung)

```
BH$(11)=ZR$+BH$(7)+ZN$
```
6 Linien entlang der oberen Kante (für negative Darstellung)

```
BH$(10)=ZR$+BH$(8)+ZN$
```
7 Linien entlang der oberen Kante (für negative Darstellung)

```
BH$(9)=BH$(1)
```
Voller Block

**Steuerzeichen für vertikale Balken**

```
DIM BV$(16)
```
Vertikale Balkenelemente

```
BV$(1)=ZR$+CHR$(32)+ZN$
```
Voller Block

```
BV$(2)=ZR$+CHR$(167)+ZN$
```
Voller Block bis auf 1 Linie rechts

```
BV$(3)=ZR$+CHR$(170)+ZN$
```
Voller Block bis auf 2 Linien rechts

```
BV$(4)=ZR$+CHR$(182)+ZN$
```
Voller Block bis auf 3 Linien rechts

```
BV$(5)=CHR$(161)
```
Voller Block bis auf 4 Linien rechts

```
BV$(6)=CHR$(181)
```
Voller Block bis auf 5 Linien rechts

```
BV$(7)=CHR$(180)
```
Voller Block bis auf 6 Linien rechts

```
BV$(8)=CHR$(165)
```
Linie entlang der linken Kante

```
BV$(16)=CHR$(167)
```
Linie entlang der rechten Kante (für negative Darstellung)

```
BV$(15)=CHR$(170)
```
2 Linien entlang der rechten Kante (für negative Darstellung)

```
BV$(14)=CHR$(182)
```
3 Linien entlang der rechten Kante (für negative Darstellung)

```
BV$(13)=ZR$+BV$(5)+ZN$
```
4 Linien entlang der rechten Kante (für negative Darstellung)

```
BV$(12)=ZR$+BV$(6)+ZN$
```
5 Linien entlang der rechten Kante (für negative Darstellung)

```
BV$(11)=ZR$+BV$(7)+ZN$
```
6 Linien entlang der rechten Kante (für negative Darstellung)

```
BV$(10)=ZR$+BV$(8)+ZN$
```
7 Linien entlang der rechten Kante (für negative Darstellung)

```
BV$(9)=BV$(1)
```
Voller Block

### Steuerzeichen für halbe Blöcke

| | |
|---|---|
| `DIM HB$(8)` | Grafik mit halben Blöcken |
| `HB$(1)=CHR$(162)` | Unterer halber Block |
| `HB$(2)=CHR$(161)` | Linker halber Block |
| `HB$(3)=ZR$+HB$(1)+ZN$` | Oberer halber Block |
| `HB$(4)=ZR$+HB$(2)+ZN$` | Rechter halber Block |
| `HB$(5)=CHR$(169)` | Dreieck in linker oberer Ecke |
| `HB$(6)=CHR$(127)` | Dreieck in rechter oberer Ecke |
| `HB$(7)=ZR$+HB$(5)+ZN$` | Dreieck in rechter unterer Ecke |
| `HB$(8)=ZR$+HB$(6)+ZN$` | Dreieck in linker unterer Ecke |

### Steuerzeichen für Viertelblöcke

| | |
|---|---|
| `DIM VB$(16)` | Viertelpunktgrafik |
| `VB$(1)=CHR$(32)` | Leeres Zeichenfeld |
| `VB$(2)=CHR$(190)` | Viertelblock oben links |
| `VB$(3)=CHR$(188)` | Viertelblock oben rechts |
| `VB$(4)=CHR$(172)` | Viertelblock unten rechts |
| `VB$(5)=CHR$(187)` | Viertelblock unten links |
| `VB$(6)=CHR$(191)` | Viertelblock oben links und unten rechts |
| `VB$(7)=ZR$+VB$(6)+ZN$` | Viertelblock oben rechts und unten links |
| `VB$(8)=CHR$(161)` | Halbblock links |
| `VB$(9)=ZR$+VB$(8)+ZN$` | Halbblock rechts |
| `VB$(10)=CHR$(162)` | Halbblock unten |
| `VB$(11)=ZR$+VB$(10)+ZN$` | Halbblock oben |
| `VB$(12)=ZR$+VB$(2)+ZN$` | Voller Block bis auf Viertelblock oben links |
| `VB$(13)=ZR$+VB$(3)+ZN$` | Voller Block bis auf Viertelblock oben rechts |
| `VB$(14)=ZR$+VB$(4)+ZN$` | Voller Block bis auf Viertelblock unten rechts |
| `VB$(15)=ZR$+VB$(5)+ZN$` | Voller Block bis auf Viertelblock unten links |
| `VB$(16)=ZR$+VB$(1)+ZN$` | Voller Block |

### Steuerzeichen für gepunktete Blöcke

| | |
|---|---|
| `DIM GP$(3)` | Gepunktete Halbblockgrafik |
| `GP$(1)=CHR$(166)` | Voller Block |

| | |
|---|---|
| `GP$(2)=CHR$(168)` | Halbblock unten |
| `GP$(3)=CHR$(124)` | Halbblock links |

### Sonstige Steuerzeichen

| | |
|---|---|
| `RT$=CHR$(13)` | Return |
| `AF$=CHR$(34)` | Anführungszeichen |

### Anmerkungen

Mit Hilfe der vorgenommenen Festlegungen für die CHR$-Codes lassen sich nun recht einfache Anwendungen mit Blockgrafik, wie z. B. ein Programm für horizontale oder vertikale Balkengrafik auch für Commodore-Rechner ohne hochauflösende Grafik erzeugen. Besonders vorteilhaft gegenüber Balkengrafiken im hochauflösenden Grafikmodus fallen die geringe Programmlaufzeit, die Ausgabemöglichkeit auf jeden Drucker mit Commodore-Grafikzeichensatz auch in Farbe un die wesentlich geringeren Druckzeiten auf.

Das Gegenstück zur CHR$-Funktion ist ASC (siehe dort).

Mit Hilfe der 16 Farbcodes können Sie den vorgestellten Satz von CHR$-Definitionen fast versechzehnfachen.

Genau wie Sie mit ZR$ bestimmte inverse Symbole von bereits vorhandenen abgeleitet haben, können Sie nun pro Farbe alle druckbaren Symbole und Halbgrafikzeichen speziell definieren.

Für die Definition eines farbigen Halbgrafikzeichen weisen Sie der betreffenden Zeichenketten-Variablen einen Farbcode ZF$() plus die Grafikzeichen-Variable BH$() oder BV$() und den Farbcode der von Ihnen generell gewählten Zeichenfarbe zu, damit ähnlich wie mit ZN$ bei der Inversdarstellung der Ausdruck weiterer Zeichen wieder im Normalmodus stattfinden kann.

*Hinweis:* Sie können natürlich die bisherigen Namen weitgehend auch für die Farbblockgrafik verwenden, indem Sie einfach anstelle von nicht indizierten Variablen indizierte nehmen und anstelle von indizierten dann doppelt indizierte. Dabei kennzeichnet der erste Index entsprechend der bisherigen Indizierung von ZF$() die Zeichenfarbe:

| | |
|---|---|
| `SM$(3,13)` | ausgefüllter roter Kreis |

oder

| | |
|---|---|
| `BH$(3,1)` | roter Vollblock |

# CIRCLE (BASIC 3.5, 7.0)

**Abkürzung:** C<SHIFT>I

**Typ:** Grafikbefehl

**Syntax:**
```
CIRCLE F,X,Y,RX,RY,AB,EB,D,W
CIRCLE ,,,RX
```

**Parameterbereich:**
F - ganze Zahlen 0 - 3
X,Y - Grafik-Bildschirmkoordinaten
RX,RY - positive Gleitkommazahlen
AB,EB,D - Gleitkommazahlen 0 - 360
W - Gleitkommazahl 1 bis 255

Mit dem CIRCLE-Befehl lassen sich beliebige Kreisformen darstellen, insbesondere Ellipsen, Vielecke, Kreisausschnitte und Kreisbögen. Die Parameter sind dabei die folgenden:

| | |
|---|---|
| F | Farbquelle (kann entfallen, dann 1)<br>0 - Hintergrund<br>1 - Vordergrund<br>2 - Mehrfarbenmodus 1<br>3 - Mehrfarbenmodus 2 |
| X | X-Koordinate des Kreismittelpunkts (kann entfallen, dann Cursorposition) |
| Y | Y-Koordinate des Kreismittelpunkts (kann entfallen, dann Cursorposition) |
| RX | Radius in X-Richtung |
| RY | Radius in Y-Richtung (kann entfallen, dann RX) |
| AB | Anfang des Bogens in Grad (kann entfallen, dann 0) |
| EB | Ende des Bogens in Grad (kann entfallen, dann Vollkreis) |
| D | Drehung in Grad im Uhrzeigersinn (kann entfallen, dann 0) |
| W | Winkel für Kreissegmente (kann entfallen, dann 2) |

**Beispiel:** Kuchengrafik

Wir zeichnen einen Kreis und teilen ihn je nach eingegebenen Prozentzahlen in Segmente auf (maximal 100). Die Eingabe der Prozentzahlen wird beendet, wenn die Zahl 0 eingegeben wird. Entspricht die Summe der Eingaben weni-

ger als 100%, so wird das letzte Segment vom Programm so groß gewählt, daß die Summe 100 ergibt. Beträgt die Summe der eingegebenen Prozentzahlen mehr als 100, so muß die Eingabe wiederholt werden. Ein beliebiger Tastendruck bringt Sie dann an den Anfang des Programms zurück.

Zum Schluß werden die Kreissegmente farbig ausgefüllt. Dabei wird nur jedes zweite berücksichtigt, da sonst die Konturen zwischen zwei Segmenten unsauber werden. Derselbe Effekt stellt sich ein, wenn die Kreissegmente sehr klein sind. Um den Kreismittelpunkt herum kann man diese Farbüberlappungen beobachten.

```
10000 DIM A(100),S(100)
10010 SUMME=0:ANZAHL=0:S(0)=0:N=0:J=0:K=-1
10020 SCNCLR
10030 PRINTTAB(14)"KUCHENGRAFIK"
10040 PRINT
10050 PRINT"BITTE EINGABEN IN PROZENT"
10060 PRINT"DIE SUMME ALLER EINGABEN ";
10070 PRINT"MUSS <100 SEIN"
10080 PRINT"EINGABEN BISHER:"
10090 ES$="WEITERE EINGABE ODER 0 FUER ENDE"
10100 FOR I=1 TO 100
10110 : CHAR 1,1,21,ES$
10120 : INPUT A(I)
10130 : SUMME=SUMME+A(I):ANZAHL=ANZAHL+1
10140 : A$=STR$(A(I))
10150 : SUMME$=STR$(SUMME)
10160 : GS$="GESAMTSUMME ="+SUMME$+"%"
10170 : CHAR 1,13,19,GS$+"  "
10180 : K=K+1
10190 : IF K=9 THEN J=J+1:K=0
10200 : CHAR 1,4*K,6+J,A$
10210 : IF A(I)=0 THEN
        BEGIN
10220 :   IF SUMME<100 THEN
          BEGIN
10230 :     S(I)=S(I-1)+(100-SUMME)*3.6
10240 :     GOTO 10370
10250 :   BEND
10260 :   IF SUMME=100 THEN
          BEGIN
10270 :     ANZAHL=ANZAHL-1
10280 :     GOTO 10370
10290 :   BEND
10300 :   CHAR 1,1,23,"BITTE NICHT MEHR ALS 100%
```

```
EINGEBEN"
10310 :   CHAR 1,1,24,"BITTE EINE TASTE DRUECKEN"
10320 :   J=0:K=0
10330 :   GETKEY T$:GOTO 10010
10340 : BEND
10350 : S(I)=S(I-1)+A(I)*3.6
10360 NEXT I
10370 GRAPHIC 1,1
10380 COLOR 1,1
10390 FOR I=0 TO ANZAHL-1
10400 : CIRCLE 1,160,100,100,,S(I),S(I+1)
10410 : DRAW 1 TO 160,100
10420 NEXT I
10430 I=0
10440 FOR F=0 TO 28 STEP 2
10450 : I=I+2
10460 : IF I> ANZAHL+2 THEN 10570
10470 : CIRCLE 0,160,100,59,,S(I-2)+1,S(I-2)+2
10480 : IF F=20 THEN COLOR 1,1:GOTO 10520
10490 : IF I=32 THEN N=N+2
10500 : IF I=56 THEN N=N-2
10510 : COLOR 1,INT((F+N)/2)+2
10520 : PAINT 1,,1
10530 : IF I<ANZAHL THEN CIRCLE 1,160,100,100,,
S(I-2),S(I-1)
10540 : IF I<ANZAHL THEN DRAW 1 TO 160,100
10550 NEXT F
10560 GOTO 10440
10570 GETKEY T$
10580 SCNCLR
10590 GRAPHIC 0,1
10600 END
```

In Zeile 10000 werden die Felder A und S, die die eingegebenen Prozentzahlen bzw. die entsprechend umgerechneten Gradzahlen für die verschiedenen Segmente aufnehmen sollen, mit 100 dimensioniert. Nach Startwertzuweisungen (Zeile 10010) und Löschen des Bildschirms (10020) sorgen die Zeilen 10030 bis 10200 dafür, daß der Eingabebildschirm aufgebaut und die Eingaben entgegengenommen (Zeile 10120) und quittiert werden (Zeile 10200). Dabei wird gleichzeitig die Summe der Prozentzahlen gebildet (Zeile 10130) und entsprechend angezeigt (Zeile 10170).

Die Eingabe wird vom Benutzer durch das Tippen einer 0 beendet. Vom Programm wird dann in den Zeilen 10210 bis 10350 geprüft, ob die Summe aller eingegebenen Prozentwerte 100% entspricht. Falls die Summe kleiner

ist, wird ein letztes Segment angelegt, das den Kreis bis 100 auffüllt (Zeilen 10220 bis 10250), falls sie größer ist, muß die Eingabe wiederholt werden (Zeilen 10300 bis 10330).

Beachten Sie dabei bitte die Zeile 10330. Hier wird das Programm mit Hilfe des Befehls GETKEY veranlaßt, so lange zu warten, bis der Benutzer eine beliebige Taste drückt. Erst dann wird zum Programmanfang bei Zeile 10010 verzweigt.

In der Zeile 10350 schließlich werden die jeweils eingegebenen Prozentzahlen in Gradzahlen umgewandelt, so daß sich der Kreis mit 360 Grad schließen kann.

In dieser großen Eingabeschleife (Zeilen 10100 bis 10360), die maximal 100 Eingaben zuläßt, wird in der Variablen ANZAHL die Zahl der tatsächlich eingegebenen Werte registriert (Zeile 10130).

Nun kann gezeichnet werden. Der Bildschirm wird mit GRAPHIC 1,1 in den hochauflösenden Modus umgeschaltet (Zeile 10370), und die Zeichenfarbe wird zunächst mit COLOR 1,1 auf Schwarz gesetzt (Zeile 10380).

Nun werden in der Schleife die Kreissegmente gezeichnet. Für den Kreisbogen wird dabei immer der CIRCLE-Befehl verwendet:

```
10400 CIRCLE 1,160,100,100,,S(I),S(I+1)
```

Die Angaben hinter CIRCLE haben folgende Bedeutung:

| | |
|---|---|
| 1 | gezeichnet wird in der Vordergrund- oder Zeichenfarbe |
| 160 | der Kreismittelpunkt liegt auf der Spalte 160 |
| 100 | und auf der Zeile 100 |
| 100 | der Radius in X-Richtung hat die Größe 100 |
| , | (keine Angabe) der Radius in Y-Richtung hat auch die Größe 100, also wird keine Ellipse, sondern ein Kreis gezeichnet |
| S(I) | Startpunkt des Kreisbogens ist bei S(I) (beim ersten Abschnitt 0) |
| S(I+1) | Endpunkt des Kreisbogens liegt bei S(I+1) (beim letzten Abschnitt 360) |

Jedesmal, wenn ein Kreisbogen gezeichnet wurde, wird mit DRAW 1 TO 160,100 vom Endpunkt aus eine Linie in der Zeichenfarbe (Parameter 1) zum Kreismittelpunkt gezogen (Zeile 10410).

Nun werden die Kreissegmente farbig ausgefüllt. Die Laufvariable F der Schleife in den Zeilen 10440 bis 10550 läuft über die 16 verfügbaren Farben

des C128. Wir gehen hier in Zweierschritten vor, um nur jedes zweite Segment, wie oben besprochen, auszufüllen.

Um mit dem Befehl PAINT die Flächen füllen zu können, muß sich der Cursor aktuell in der jeweiligen Fläche befinden. Dazu wird mit dem CIRCLE-Befehl unsichtbar, in der Hintergrundfarbe, mitten in jedem Segment ein kleiner Strich gezeichnet (Zeile 10470). Dieser Trick bringt uns ohne Koordinatenberechnungen in die jeweils gewünschte Fläche.

Bevor wir nun die Fläche mit Farbe füllen, müssen noch zwei Abfragen gemacht werden. Falls nämlich die Farbschleife als Vordergrundfarbe dieselbe Farbe liefert, die der Hintergrund bereits hat (Dunkelgrau), wollen wir die Fläche aus optischen Gründen schwarz färben (Zeile 10480), eine Farbe, die sonst nicht vorkommt. Und falls das letzte Kreissegment genau die gleiche Farbe erhielte wie das erste und so kein Unterschied mehr zwischen den Segmenten festzustellen wäre, lassen wir diese Farbe aus und nehmen die nächste (Zeile 10490).

Nun endlich wird die Farbe festgelegt:

```
10510 COLOR 1,INT((F+N)/2)+2
```

Damit durch unsere Zweierschrittschleife keine Farbe ausgelassen wird, dividieren wir dabei den Farbparameter durch 2, und da wir die Farbe Schwarz auslassen wollen, addieren wir noch 2 dazu. Die Variable N ist der Farbversatz, falls eine Farbe, wie oben erläutert, ausgelassen werden soll.

Da der schwarze Rand zwischen den Segmenten an manchen Stellen unschönerweise noch punktweise zu sehen ist, bessern wir die Ränder nun in der Füllfarbe noch einmal nach (Zeilen 10530 und 10540). Nur das letzte Segment wird aus optischen Gründen dabei ausgelassen.

Sie können sich nun diese Kuchengrafik so lange ansehen, bis Sie genug davon haben und eine beliebige Taste drücken (Zeile 10570). Erneut wird dann in Zeile 10580 der hochauflösende Schirm gelöscht, was normalerweise nicht nötig wäre, aber bei einem nochmaligen Programmstart das kurze Aufflackern der alten Grafik verhindert. Vielleicht wollen Sie ja das Programm mit anderen Werten ein zweites Mal starten.

## (BASIC 2.0, 3.5, 4.0, 7.0) CLOSE

**Abkürzung:** CL<SHIFT>O

**Typ:** Systembefehl für Ein-/Ausgabe

**Syntax:** CLOSE Dateinummer

**Parameterbereich:** Dateinummer

Die Anweisung CLOSE beendet die Ein- und Ausgabeoperationen von Disketten-Dateien. Man sagt, die Datei wird abgeschlossen. Die Anweisung bewirkt das Gegenteil wie die OPEN-Anweisung, die eine Datei zur Verarbeitung öffnet. Wenn eine CLOSE-Anweisung erfolgreich sein soll, dann muß vorher für dieselbe Dateinummer eine OPEN-Anweisung erfolgt sein.

CLOSE kann entweder als sofort ausführbarer Befehl oder als Programmbefehl eingesetzt werden.

Bei komplexen Unterprogrammfolgen kann es vorkommen, daß der ungeübte Programmierer versucht, eine Datei zu öffnen, die bereits geöffnet war. Dann erscheint die Fehlermeldung:

```
?FILE OPEN ERROR
```

Um das zu vermeiden, kann man vorsorglich jedem OPEN-Befehl einen zugehörigen CLOSE-Befehl vorausschicken, was nichts schadet, auch wenn die Datei bereits geschlossen ist. Es kommt lediglich zu einer etwas höheren Ausführungszeit des Programms.

Bei Dateien, auf die ein sequentiell schreibender Zugriff erfolgt und die nicht mit CLOSE geschlossen werden, wird keine Endemarke (EOF - End Of File) hinter den letzten Datensatz der Datei geschrieben. Deshalb geht in der Regel der letzte Block, der sich noch im Rechnerpuffer befindet, verloren. Die Datei wird in diesem Fall im Inhaltsverzeichnis mit einem * gekennzeichnet und kann dann nur noch bis auf den verlorenen Block mit Direktzugriffs-Befehlen gelesen werden.

Derselbe Fehler kann auch auftreten, wenn eine Datei nicht mit CLOSE geschlossen wurde und versucht wird, mit APPEND etwas anzuhängen, da APPEND indirekt einen OPEN-Befehl einschließt.

### Anmerkung

Wenn man sichergehen will, daß alle im Dateipuffer enthaltenen Zeichen vor dem Schließen der Datei ausgegeben wurden, kann man den Befehl PRINT# zu Hilfe nehmen, d. h.

```
OPEN 1,4:PRINT#1,END DATA:CLOSE 1
```

## (BASIC 2.0, 4.0, 3.5, 7.0) CLR

**Abkürzung:** C<SHIFT>L

**Typ:** Befehl zur Bildschirmsteuerung

**Syntax:** CLR

Diese Anweisung bewirkt keinesfalls dasselbe wie das Drücken der CLR-Taste, die den Bildschirm löscht. Die CLR-Anweisung löscht die Werte aller aktuellen Variablen und Dimensionen von Feldern. Ein Feld, das Sie nach Anwendung des CLR-Befehls benutzen wollen, muß in einer DIM-Anweisung neu definiert werden.

Alle numerischen Variablen erhalten den Wert 0 und alle Zeichenkettenvariablen einen Leerstring.

Außerdem wird, wenn DATA-Zeilen vorhanden sind, der Zeiger wieder auf den ersten DATA-Wert gesetzt (wie bei RESTORE). Zudem werden alle offenen Dateien geschlossen. Lediglich das Programm selbst bleibt von einer CLR-Anweisung unberührt.

Wenn Sie einen Teil der Variablen erhalten wollen, speichern Sie sie, nachdem Sie mit POINTER ihre Adresse festgestellt haben, mit STASH in einer anderen Speicherbank ab, bevor Sie den CLR-Befehl im BASIC-Programm verwenden. Anschließend können Sie die Variablen mit SWAP oder mit FETCH wieder zurückholen.

Falls Sie über keine freie Speicherbank verfügen, können Sie die Variableninhalte mit BSAVE auf eine Diskette schreiben und nach dem CLR-Befehl mit BLOAD wieder laden.

Wenn Ihr Rechner nur über BASIC 2.0 oder 4.0 verfügt, müssen Sie die Variableninhalte mit PRINT# satzweise auf die Diskette schreiben und nach CLR satzweise mit INPUT# wieder einlesen. Das kann jedoch etwas zeitaufwendig werden.

Die CLR-Funktion wird häufig im Zusammenhang mit Overlay-Programmierung und zur Wiederherstellung wohldefinierter Anfangszustände in bestimmten Situationen im Programmablauf eingesetzt.

### Anmerkung

Die Befehle RUN, LOAD und NEW bewirken automatisch immer die Ausführung einer CLR-Anweisung.

## (BASIC 2.0, 4.0, 3.5, 7.0) CMD

**Abkürzung:** C<SHIFT>M

**Typ:** Systembefehl für Ausgabe

**Syntax:** CMD Dateinummer

**Parameterbereich:** Dateinummer

Mit dem CMD-Befehl kann die Ausgabe, die normalerweise auf den Bildschirm geht, auf ein anderes Gerät umgelenkt werden, z. B. auf ein Diskettenlaufwerk, Kassettenrecorder, Modem oder Drucker.

Dem CMD-Befehl muß ein OPEN-Befehl vorangehen, der die Datei mit der entsprechenden Nummer öffnet. Dem CMD-Befehl folgende PRINT- und LIST-Anweisungen senden dann Ausgaben zu dem angewählten Ausgabegerät.

In dem nachfolgenden Beispiel wird ein Programmlisting über den Drucker, der die Gerätenummer 4 hat, ausgegeben:

```
OPEN 1,4
CMD 1,"PROGRAMMNAME"
PRINT:PRINT:LIST
PRINT#1
CLOSE 1
```

In dem obigen Beispiel sind alle Befehle als sofort auszuführende Befehle eingegeben worden. Sie können aber auch Bestandteil eines Programms sein.

Die folgenden Befehle erzeugen eine sequentielle Datei auf Diskette, und zwar mit dem Namen Listing:

```
OPEN 1,8,2,"0:LISTING,S,W"
CMD 1,"PROGRAMMNAME":PRINT:LIST
PRINT#1
CLOSE 1
```

### Anmerkungen

Es gibt drei Möglichkeiten, um die Wirkung des CMD-Befehls wieder aufzuheben:

1. Drücken Sie RUN/STOP und RESTORE, bzw. den Resetschalter.

2. Geben Sie einen weiteren CMD-Befehl mit der Gerätenummer 3 für den Bildschirm ein.

3. Man kann den Befehl PRINT#1 eingeben, so wie wir es auch in den Beispielen gesehen haben. Diese Methode ist den anderen vorzuziehen, da dabei automatisch auch der Ausgabepuffer geleert wird.

## (BASIC 4.0, 3.5, 7.0) COLLECT

**Abkürzung:** COLL<SHIFT>E

**Typ:** Systembefehl für Diskettenlaufwerk

**Syntax:**
```
COLLECT
COLLECT DLaufwerknummer ON UGerätenummer
COLLECT DLaufwerknummer,UGerätenummer
```

**Parameterbereich:** Laufwerknummern
Gerätenummern

Der Befehl COLLECT reorganisiert die Diskette. Dabei werden alle offenen Dateien gelöscht.

```
COLLECT D1 ON U8
```

führt diese Aufräumarbeiten z. B. auf Laufwerk 1 durch. Diese Schreibweise ist gleichbedeutend mit

```
COLLECT D1,U8
```

Alle Dateien, die nicht oder nicht mehr im Inhaltsverzeichnis eingetragen sind, weil sie z. B. mit SCRATCH überschrieben wurden oder nicht als Standarddatei mit Direktbefehlen auf die Diskette geschrieben wurden, gehen bei COLLECT verloren.

Aufgrund der Arbeitsweise von COLLECT werden Dateien, die nach dem SCRATCH-Befehl lediglich aus dem Inhaltsverzeichnis verschwunden sind, endgültig physisch freigegeben. Das geschieht dadurch, daß alle Blöcke, die noch als belegt gekennzeichnet sind, aber zu keinem Inhaltsverzeichnis-Eintrag gehören, definitiv in der Blockbelegungstabelle als frei und wieder verfügbarer angegeben werden. COLLECT bewirkt dabei auch das Löschen von mit * gekennzeichneten, teilweise gespeicherten und noch offenen Dateien. Auch im Zusammenhang mit Direktzugriffs-Dateien und User-Dateien kann die Verwendung von COLLECT zu Verlusten führen.

Der COLLECT-Befehl ist auch im Programm-Modus einsetzbar. Er kann beispielsweise dazu dienen, programmgesteuert irgendwelche Fehlersituationen zu bereinigen. Dabei ist jedoch zu bedenken, daß Wartezeiten in der Größenordnung von Minuten entstehen können.

### Beispiel

Wir rufen im Programm den COLLECT-Befehl auf. Vorher wird der Benutzer aufgefordert, die Gerätenummer und die Laufwerknummer anzugeben. Standardmäßig sind diese Werte 8 und 0.

```
30000 PRINT "GEBEN SIE DIE GERAETENUMMER EIN"
:INPUT G$:IF NOT(G$="8" OR G$="9" OR G$="10")THEN 30000
30010 PRINT"GEBEN SIE DIE LAUFWERKNUMMER EIN"
:INPUT L$:IF NOT(L$="1" OR L$="0")THEN 30010
30020 PRINT "HABEN SIE DIE DISKETTE EINGELEGT?"
30030 PRINT "ANTWORTEN SIE MIT /J/, JEDE
ANDERE TASTE BRICHT COLLECT AB"
30040 GET A$:IF A$="" THEN 30040
30050 IF A$="J" THEN
      BEGIN
30060 : IF G$="8" THEN
        BEGIN
30070 :   IF L$="1" THEN
          BEGIN
30080 :     COLLECT D1,U8
30090 :     GOTO 30300
30100 :   BEND
30110 :   COLLECT D0,U8
30120 :   GOTO 30300
30130 : BEND
30140 : IF G$="9" THEN
        BEGIN
30150 :   IF L$="1" THEN
          BEGIN
30160 :     COLLECT D1,U9
30170 :     GOTO 30300
30180 :   BEND
30190 :   COLLECT D0,U9
30200 :   GOTO 30300
30210 : BEND
30220 : IF L$="1" THEN
        BEGIN
30230 :   COLLECT D1,U10
30240 :   GOTO 30300
30250 : BEND
30260 : COLLECT D0,U10
30270 : GOTO 30300
30280 BEND
30290 PRINT "COLLECT WIRD NICHT DURCHGEFUEHRT"
      : GOTO 30310
30300 PRINT "COLLECT LAEUFT"
30310 END
```

*Hinweis:* Falls sich aus irgendeinem Grund der Schreibkopf Ihres Laufwerks in der Ruhestellung nicht mehr in der richtigen Position befindet, probieren Sie es einmal mit dem COLLECT-Befehl. In vielen Fällen arbeitet das Diskettenlaufwerk anschließend wieder einwandfrei.

# (BASIC 7.0) COLLISION

**Abkürzung:** CO<SHIFT>L

**Typ:** Sprite-Befehl

**Syntax:**
```
COLLISION X,Zeilennummer
COLLISION X
```

**Parameterbereich:** X - ganze Zahlen 0 - 2
Zeilennummern

Wie schon der Name besagt, testet die Anweisung COLLISION auf einen Zusammenstoß, und zwar von Sprites. Wenn im Programm dieser Befehl einmal abgearbeitet wurde, bewirkt fortan jede Sprite-Kollision einen Unterprogrammaufruf.

Der Parameter X bestimmt dabei, um welche Kollision es sich handelt:

| | |
|---|---|
| X=0 | Kollision zwischen zwei Sprites |
| X=1 | Kollision zwischen einem Sprite und dem Hintergrund |
| X=2 | Lichtgriffel (Lightpen) ist aktiv |

Nach einem erfolgten Zusammenstoß wird zu der angegebenen Zeilennummer verzweigt:

```
100 COLLISION 0,2000
```

Die Anweisung

```
500 COLLISION 0
```

ohne Angabe einer Zeilennummer, hebt die Anweisung aus Zeile 100 wieder auf.

### Anmerkung

Der Befehl BUMP fragt ab, welches Sprite zusammengestoßen ist (siehe dort).

# COLOR (BASIC 3.5, 7.0)

**Abkürzung:** COL<SHIFT>O

**Typ:** Grafikbefehl

**Syntax:** `COLOR Farbquelle,Farbnummer,Helligkeit`
(nur BASIC 3.5)

`COLOR Farbquelle,Farbnummer`

**Parameterbereich:** Farbquelle - ganze Zahlen 0 - 6
Farbnummern
Helligkeit - ganze Zahlen 0 - 7

COLOR steuert die Einstellung für Farbe und Helligkeit des Bildschirmrandes, Vorder- und Hintergrund.

Dabei können die Parameter folgende Werte annehmen:

| | |
|---|---|
| Farbquelle | 0 - Hintergrund |
| | 1 - Vordergrund |
| | 2 - Mehrfarbenmodus 1 |
| | 3 - Mehrfarbenmodus 2 |
| | 4 - Rand |
| | 5 - Textfarbe nur C128 |
| | 6 - Hintergrund im 80-Zeichen-Modus nur C128 |
| Farbnummer | 1 - Schwarz |
| | 2 - Weiß |
| | 3 - Rot |
| | 4 - Zyan |
| | 5 - Violett |
| | 6 - Grün |
| | 7 - Blau |
| | 8 - Gelb |
| | 9 - Orange |
| | 10 - Braun |
| | 11 - Rosarot |
| | 12 - Dunkelgrau |
| | 13 - Mittelgrau |
| | 14 - Hellgrün |

15 - Hellblau
16 - Hellgrau

nur BASIC 3.5:

Helligkeit Werte von 0 (dunkel) bis 7 (hell)
(kann entfallen, dann 6)

**Beispiel**

Ein vollständiges Beispiel für den C128 finden Sie unter dem Stichwort SPRITE.

# CONCAT (BASIC 4.0, 7.0)

**Abkürzung:** C<SHIFT>O

**Typ:** Systembefehl für Diskettenlaufwerk

**Syntax:**
```
CONCAT DLaufwerknummer,"Quelldatei" TO
DLaufwerknummer,"Zieldatei" ON
UGerätenummer

CONCAT "Quelldatei" TO "Zieldatei"
```

**Parameterbereich:** Laufwerknummern
Dateinamen
Gerätenummern

Der Befehl CONCAT verbindet zwei sequentielle Dateien miteinander. Wenn nur ein Laufwerk vorhanden ist oder wenn sich beide Dateien auf einer Diskette im Laufwerk D0 befinden, können die Laufwerknummern weggelassen werden.

### Beispiel

```
CONCAT "DATEI1" TO "DATEI2"
```

oder

```
CONCAT D0,"DATEI1" TO D0,"DATEI2" ON U8
```

hängt die Daten von DATEI1 an die sequentielle Datei DATEI2 hinten an.

### Anmerkungen

Mit CONCAT können Sie nicht zwei Programmdateien miteinander verknüpfen.

Achten Sie auch darauf, daß Ihre Dateinamen nicht zu lang sind, denn die maximale Länge der aus dem CONCAT-Befehl resultierenden Befehls-Zeichenkette, die zum Diskettenlaufwerk gesandt wird, darf insgesamt nur 40 Zeichen lang sein. Andernfalls wird der Name der Zieldatei unter Umständen verkürzt in das Inhaltsverzeichnis der Diskette aufgenommen.

## (BASIC 2.0, 4.0, 3.5, 7.0) CONT

**Abkürzung:** keine möglich

**Typ:** Systembefehl

**Syntax:** `CONT`

Nach einer Programmunterbrechung durch einen END- oder STOP-Befehl sowie nach einer Unterbrechung über die Tastatur - mittels der RUN/STOP-Taste - kann die Ausführung des Programms mit dem Befehl CONT (CONTINUE) fortgesetzt werden. In allen diesen Fällen setzt CONT die Ausführung des Programms mit der nächsten Anweisung fort (nicht notwendigerweise mit der nächsten Zeile).

Wurde das Programm durch einen Syntax- oder Eingabefehler unterbrochen oder wurden an ihm Änderungen vorgenommen, so ist eine Fortführung mit CONT nicht möglich.

Es können aber Variablenwerte abgefragt und verändert werden, und das Programm kann mit LIST betrachtet werden.

CONT als Direktbefehl entspricht dem Fortführungsbefehl RESUME in einer Fehlerbehandlungsroutine im Programm-Modus. Wollen Sie das Programm aber mit einem anderen Befehl als mit dem nächsten fortsetzen, müssen Sie nach der Programmunterbrechung GOTO mit Angabe der gewünschten Zeilennummer verwenden. Das würde im Programm-Modus dem Befehl RESUME plus Zeilennummer entsprechen.

### Anmerkungen

Der Befehl CONT kann außer in BASIC 7.0 nur im Direktmodus angewendet werden. In den anderen BASIC-Versionen führt er im Programm-Modus zu einer Endlosschleife. Insofern müßte man den Befehl CONT in BASIC 7.0 als Systembefehl und in den anderen BASIC-Versionen als Kommando bezeichnen.

# COPY (BASIC 2.0*, 4.0, 3.5, 7.0)

**Abkürzung:** CO<SHIFT>P

**Typ:** Systembefehl für Diskettenlaufwerk

**Syntax:**
```
COPY DLaufwerknummer,"Quelldatei" TO
DLaufwerknummer,"Neuedatei" ON
UGerätenummer

COPY "Quelldatei" TO "Neuedatei"
```

**Parameterbereich:** Laufwerknummern
Dateinamen
Gerätenummern

Mit dem COPY-Befehl kann eine Kopie einer Diskettendatei angefertigt werden. Dabei kann die Laufwerknummer entfallen, wenn nur ein Diskettenlaufwerk angeschlossen ist oder wenn sich die alte und die neue Datei beide auf Laufwerk D0 befinden.

**Beispiel**

```
COPY D0,"ALT" TO D1,"NEU" ON U8
```

bringt eine Kopie der Datei ALT unter dem Namen NEU auf die Diskette in Laufwerk D1.

Wenn der Name unverändert bleiben soll, kann er beim zweitenmal entfallen:

```
COPY D0,"ALT" TO D1 ON U8
```

Mit der Angabe nur der Laufwerknummer und eventuell der Gerätenummer können Sie das Kopieren einer ganzen Diskette, also aller Dateien, von der Quelldiskette auf die Zieldiskette veranlassen. Das geht zwar sehr langsam, aber viel sicherer als das physische Spur-für-Spur-Kopieren beim BACKUP-Befehl, das unweigerlich alle eventuell vorhandenen Dateien auf der Zieldiskette überschreibt.

Das COPY-Dienstprogramm des Diskettenlaufwerks verwendet nämlich das Inhaltsverzeichnis beider Disketten. Stellt es nun im Laufe des Kopierens fest,

daß die nächste zu kopierende Datei der Quelldiskette auf der Zieldiskette schon existiert, bricht es an dieser Stelle den Kopiervorgang ab. Alle Dateien, die hinter dieser betreffenden Datei im Inhaltsverzeichnis der Quelldiskette eingetragen sind, müssen Sie dann gezielt einzeln kopieren. Die davorliegenden Dateien sind dann jedoch schon erfolgreich kopiert worden.

Wenn Sie während des Kopiervorgangs nicht auf die Diskette zugreifen müssen, können Sie übrigens in der Zwischenzeit ohne weiteres an Ihrem Rechner weiterarbeiten.

### Anmerkung

Achten Sie darauf, daß die verwendeten Dateinamen nicht zu lang sind, andernfalls könnte der Name der Zieldatei abgeschnitten werden. Siehe auch Anmerkung zu CONCAT.

# COS (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** keine möglich

**Typ:** trigonometrische Funktion

**Syntax:** `COS(numerischer Ausdruck oder Variable oder Konstante)`

**Argumentbereich:** Gleitkommazahlen

**Ergebnisbereich:** Gleitkommazahlen

Die Funktion COS berechnet den Cosinus, wobei das Argument im Bogenmaß gegeben wird. Wird das Argument X in Grad angegeben, so kann man folgende Formel verwenden:

```
COS(X*0.0174533)
```

Die Berechnung erfolgt in einfacher Genauigkeit.

COS gehört ebenso wie SIN, TAN und ATN zu den eingebauten trigonometrischen Funktionen.

### Beispiel

Mit Hilfe der trigonometrischen Funktionen können interessante grafische Effekte erzielt werden. Zusammen mit dem CIRCLE-Befehl wollen wir in unserem Beispiel für den C128, C16, C116 und Plus/4 eine solche Grafik erstellen. Für den C64 kann diese Grafik mit einer Befehlserweiterung realisiert werden.

```
100 GRAPHIC 1,1
110 PI=3.1415
120 FOR I=0 TO 48 STEP .7
130 : A=I*PI/24
140 : X=160+90*COS(A)
150 : Y=100-45*SIN(A)
160 : W=ABS(30*COS(A))
170 : CIRCLE 1,X,Y,W,30,,,,10
```

```
180 NEXT I
190 GETKEY T$
200 GRAPHIC 0,1
210 END
```

Sie können die Grafik so lange betrachten, bis Sie eine beliebige Taste drücken.

# DATA (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** D<SHIFT>A

**Typ:** sekundäres Schlüsselwort (nur in Verbindung mit READ)

**Syntax:** DATA Konstante,Konstante,...,...

**Parameterbereich:** Zahlen und Zeichenketten

Die DATA-Anweisung dient dazu, numerische und Zeichenkettenkonstanten in einem Programm anzugeben. Sie kann an beliebigen Stellen im Programm stehen und ist nicht ausführbar, sondern wird, falls der Interpreter bei der Programmausführung auf sie stößt, einfach übersprungen. Die Anzahl der DATA-Anweisungen ist nicht begrenzt.

Man kann sich vorstellen, daß die DATA-Anweisungen ein der Reihe nach geordneter Pool von Konstanten ist. Dieser Pool kann der Reihe nach (sequentiell) durch die Anweisung READ in Programmvariablen übertragen werden. Durch die Anweisung RESTORE besteht die Möglichkeit, die Übertragung durch READ erneut mit dem ersten Element dieses Datenpools beginnen zu lassen.

Eine DATA-Anweisung kann numerische Daten enthalten:

```
100 DATA 99,88,77,66
```

oder Zeichenkettendaten:

```
110 DATA MONTAG,DIENSTAG,MITTWOCH
```

oder eine Kombination von beiden:

```
120 DATA JANUAR,,1,2,189.76
```

Zwei Kommas ohne Angabe dazwischen werden entweder als Ziffer 0 oder als Leerstring interpretiert, abhängig vom Variablentyp, in den gerade eingelesen wird.

Zeichenkettenkonstanten brauchen nicht in Anführungszeichen eingeschlossen zu werden, solange sie selbst keine Kommas, Grafikzeichen, Doppelpunkte oder Leerzeichen enthalten.

```
130 DATA "1,234,567.89 DM","200 DATA"
```

**Beispiel**

Wir wollen in unserem Beispielprogramm das Commodore-Zeichen drucken. Das Programm enthält in den DATA-Zeilen eine Digitalisierung des zu zeichnenden Bildes. Die Zahlen in den DATA-Anweisungen ab Zeile 280 bedeuten folgendes:

1. Die Angabe einer negativen Zahl bewirkt einen Zeilenvorschub (siehe Zeile 170).

2. Die Angabe einer Zahl, die größer ist als 200, bedeutet Ende des Bildes (siehe Zeile 180).

3. Ein Zahlenpaar A und B mit nicht negativen Zahlen A und B bedeutet, Position A bis B ist zu drucken, wobei das Druckmuster aus der Variablen A$ ab Position A+1 zu entnehmen ist. Die Variable B muß dabei drei Bedingungen erfüllen, die in Zeile 210 abgefangen werden:

- B muß größer als 0 sein.

- B muß kleiner als die Bildschirmbreite sein.

- B muß größer als A sein.

Wenn eine dieser Bedingungen nicht erfüllt ist, wird der Text "DATA-FEHLER" ausgegeben und die Bearbeitung abgebrochen.

Beachten Sie bei der Eingabe der DATA-Zeilen, daß die Zeilen 370 bis 450 den Zeilen 280 bis 360 in umgekehrter Reihenfolge entsprechen. Es ist also nicht nötig, die Zeilen doppelt einzugeben. Sie müssen nur die einmal eingegebenen Zeilen 280 bis 360 geeignet umnumerieren:

```
100 PRINT CHR$(147)
110 A$="COMMODORE"
120 H$=A$
130 A$=A$+H$
140 T=1
150 PRINT:PRINT:PRINT
160 READ A
```

```
170 IF A<0 THEN PRINT:GOTO 160
180 IF A>200 THEN 470
190 PRINTTAB(A+T);
200 READ B
210 IF B<1 OR B>40-T OR B<A THEN
    BEGIN
220 : PRINT "DATA-FEHLER"
230 : STOP
240 BEND
250 B$=MID$(A$,1,B-A+1)
260 PRINT B$;
270 GOTO 160
280 DATA 15,20,-1
290 DATA 13,20,-1
300 DATA 12,20,-1
310 DATA 11,20,-1
320 DATA 11,16,-1
330 DATA 10,15,20,28,-1
340 DATA 10,14,20,27,-1
350 DATA 10,14,20,26,-1
360 DATA 10,14,-1
370 DATA 10,14,-1
380 DATA 10,14,20,26,-1
390 DATA 10,14,20,27,-1
400 DATA 10,15,20,28,-1
410 DATA 11,16,-1
420 DATA 11,20,-1
430 DATA 12,20,-1
440 DATA 13,20,-1
450 DATA 15,20,-1
460 DATA 2000
470 END
```

Aus Gründen der Übersichtlichkeit wurde jede Zeile der Bildschirmgrafik in einer separaten DATA-Zeile abgelegt. Das ist selbstverständlich nicht nötig. Sie können durchaus soviel Werte hinter eine DATA-Anweisung schreiben, wie es die Programmzeilenlänge erlaubt.

*Hinweis:* Die BEGIN...BEND-Konstruktion in den Zeilen 210 läuft nur auf dem C128. Auf allen anderen Rechnern muß hier die unelegante Sprunganweisung GOTO eingesetzt werden:

```
210 IF B<1 OR B>40-T OR B<A THEN PRINT"DATA-FEHLER":
    GOTO 470
```

### Anmerkungen

Das Beispielprogramm zeichnet sich durch seine hohe Flexibilität aus. Es ist leicht möglich, durch Eingabe eines anderen Bildes in der oben beschriebenen digitalisierten Weise eine andere Grafik zu erzeugen.

Die DATA-Anweisung sollte, falls ein Diskettenlaufwerk zur Verfügung steht, nur sehr zurückhaltend eingesetzt werden, da das Einlesen von Daten aus einer Diskettendatei ein Programmsystem wesentlich flexibler und offener gegenüber Änderungen werden läßt. In der Testphase eines Programms kann es jedoch für kleine Testdatenmengen recht praktisch sein, einen späteren Dateizugriff erst einmal durch eine Lösung mit DATA-Zeilen zu simulieren.

Für Computer-Besitzer ohne Diskettenlaufwerk kann eine Lösung mit DATA-Zeilen gegenüber einer Kassettenrecorder-Datei durchaus in einem gewissen Rahmen als vertretbare Alternative angesehen werden.

Die Anweisung DATA ist nicht im Direktmodus möglich.

# DCLEAR (BASIC 7.0)

**Abkürzung:** DCL<SHIFT>E

**Typ:** Systembefehl für Ein-/Ausgabe

**Syntax:** DCLEAR DLaufwerknummer ON UGerätenummer

DCLEAR

**Parameterbereich:** Laufwerknummern
Gerätenummern

Die Anweisung DCLEAR schließt alle Kanäle zur Diskettenstation ab. Dabei bleibt aber eine Datei, die nicht vorher mit DCLOSE geschlossen wurde, zum Schreiben geöffnet. Wenn Sie diese Datei später lesen wollen, ist das nicht möglich.

Achten Sie also immer darauf, alle Dateien ordnungsgemäß zu schließen, bevor Sie mit DCLEAR den Anfangszustand wieder herstellen.

Man kann sagen, daß der DCLEAR-Befehl hauptsächlich dazu dient, Fehlermeldungen zu vermeiden, wie sie beim Eröffnungsversuch eines bereits geöffneten Kanals auftreten würden. Das kann beispielsweise beim häufigen Umlenken von Druckausgaben:

- vom Bildschirm auf die Diskette;
- vom Bildschirm auf den Drucker;
- vom Drucker auf die Diskette;
- vom Drucker auf den Bildschirm;
- von Diskette auf den Bildschirm;
- von Diskette auf den Drucker;

wie das unter Verwendung des CMD-Befehls unter Programm-Kontrolle möglich ist, geschehen.

Sie sollten sich aber auch durch die Verwendung des DCLEAR-Befehls, den Sie zur Vermeidung solcher Fehler einsetzen können, nicht verleiten lassen, den CMD-Befehl außer zu Testzwecken generell als Programmiertrick einzusetzen. Die Ausgabesteuerung sollten Sie besser direkt mit dem PRINT#- oder PRINT-Befehl über Parameter vornehmen. Das führt zu einem besseren Programmierstil.

# (BASIC 7.0) DCLOSE

**Abkürzung:** D<SHIFT>C

**Typ:** Systembefehl für Ein-/Ausgabe

**Syntax:**
```
DCLOSE
DCLOSE#Dateinummer
DCLOSE#Dateinummer ON UGerätenummer
```

**Parameterbereich:** Dateinummern
Gerätenummern

Die Anweisung DCLOSE ohne weitere Angaben schließt alle zuvor mit DOPEN geöffneten Dateien. Eine einzelne Datei kann unter Angabe ihrer Nummer beispielsweise mit dem Befehl

```
DCLOSE#1
```

geschlossen werden. Dabei muß die Dateinummer mit der bei DOPEN# vergebenen übereinstimmen.

Jeder endgültige Abschluß von Ausgaben während eines Programmlaufs sollte mit dem Befehl CLOSE enden. Das ist wichtig, damit keine Daten, die sich eventuell noch im Puffer befinden, verlorengehen.

Eine Datei auf einem Peripheriegerät kann ebenfalls nur mit CLOSE ordnungsgemäß abgeschlossen werden, d. h. mit einem besonderen Zeichen am Ende (EOF-Marke) markiert werden, was für die spätere Verarbeitung dieser Datei eine wichtige Voraussetzung ist.

Damit Sie sicher sein können, daß alle Dateien, mit denen Ihr Programm gearbeitet hat, auch wirklich abgeschlossen sind, können Sie unmittelbar vor dem logischen Programmende einen DCLOSE-Befehl einfügen.

DCLOSE kann auch zur Bereinigung oder Vermeidung der Situation eingesetzt werden, daß Sie mehr als die erlaubte Anzahl von Dateien gleichzeitig eröffnen wollen. Je nach Dateityp sind maximal 10 erlaubt.

# DEC (BASIC 3.5, 7.0)

**Abkürzung:** keine möglich

**Typ:** numerische Funktion

**Syntax:** `DEC("hexadezimale Zeichenkette")`

**Argumentbereich:** Zeichenketten aus maximal 4 Hex-Zahlen

**Ergebnisbereich:** Speicheradressen

Die Funktion DEC wandelt eine Hexadezimalzahl in ihr dezimales Äquivalent um. Der hexadezimale Ausdruck darf nur aus den Ziffern 0 bis 9 und aus den Buchstaben A bis F bestehen. Leerzeichen, die ebenfalls zugelassen sind, werden ignoriert. Maximal darf die Zeichenkette 4 Stellen enthalten. Die größte Zahl, die umgerechnet werden kann, ist also:

```
PRINT DEC("FFFF")
```

Ergebnis: 65535

Ihr Haupteinsatzgebiet findet die DEC-Funktion in der Umrechnung von Speicheradressen, die stets in hexadezimaler Schreibweise angegeben sind.

Da es außerdem in BASIC nicht möglich ist, mit hexadezimalen Zahlen zu rechnen, müssen diese stets mit DEC in ihre dezimale Form umgewandelt werden. Dann können die Berechnungen durchgeführt werden, und anschließend wird das Ergebnis mit HEX$, der Umkehrfunktion zu DEC, wieder zurückverwandelt.

### Beispiel

Mit unserer selbst definierten Funktion FNDC, zu der Sie unter dem Stichwort DEF FN weitere Erläuterungen finden, simulieren wir die DEC-Funktion, so daß sie auch in BASIC 2.0 und 4.0 verfügbar wird. Außerdem bietet sie die Möglichkeit, auch längere als vierstellige Zeichenketten umzurechnen. Leerstellen sind dabei im Gegensatz zur DEC-Funktion nicht zugelassen.

```
10 REM**********************************
11 REM*        HILFSFUNKTIONEN          *
12 REM**********************************
```

```
13 REM* DIE FUNKTION FNOF ERZEUGT EINE *
14 REM* FEHLERMELDUNG BEI EINGABE VON  *
15 REM* ZEICHEN GROESSER ALS F UND     *
16 REM* KLEINER ALS 0                  *
17 REM*********************************
18 DEF FNOF(I)=1.7*10↑37*
       (MID$(HX$,I,1)>"F" OR
        MID$(HX$,I,1)<"0" OR
        LEN (HX$)>4)*99
19 DEF FND1(I)=ASC(MID$(HX$,I,1)+" ")+
               48*(MID$(HX$,I,1)<":"
               AND MID$(HX$,I,1)>"/")
20 DEF FND2(I)=FNOF(I)+55*(MID$(HX$,I,1)<"G"
                     AND MID$(HX$,I,1)>"@")
21 DEF FND3(I)=FND1(I)+FND2(I)
22 REM*********************************
23 REM* SIMULATION DER DEC-FUNKTION    *
24 REM*********************************
25 DEF FNDC(HX)=FND3(1)*4096+FND3(2)*256
               +FND3(3)*16 +FND3(4)*16↑0
26 REM*********************************
27 REM* HX IST DUMMY-ARGUMENT          *
28 REM* HX$ IST INOFFIZIELLES ARGUMENT *
29 REM*********************************
30 REM********** TESTDATEN *************
31 HX$="0000":PRINT TAB(6) FNDC(HX)
32 HX$="F000":PRINT TAB(6) FNDC(HX)
33 HX$="000F":PRINT TAB(6) FNDC(HX)
34 HX$="AAAA":PRINT TAB(6) FNDC(HX)
35 HX$="FFFF":PRINT TAB(6) FNDC(HX)
36 HX$="1111":PRINT TAB(6) FNDC(HX)
37 HX$="   F":PRINT TAB(6) FNDC(HX)
38 END
```

Die unzulässige Zeichenkette in Zeile 37 erzeugt eine Fehlermeldung.

# DEF FN (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** D<SHIFT>E

**Typ:** Anweisung

**Syntax:** DEF FNName(Argument)=arithmetischer Ausdruck

**Parameterbereich:** Gleitkommazahlen

**Ergebnisbereich:** Gleitkommazahlen

Mit der Anweisung DEF FN können Sie eigene arithmetische Funktionen für eine Anwendung in einem BASIC-Programm definieren. Das Festlegen einer solchen Funktion erfordert, daß man die drei Unterscheidungselemente der Anweisung DEF FN angibt:

1. einen Namen für die Funktion, die definiert werden soll;

2. eine Variable, die in der Definition der Funktion benutzt wird;

3. einen arithmetischen Ausdruck, der die eigentlichen Berechnungen der Funktion festlegt.

Die Funktion muß vor ihrem ersten Aufruf im Programm definiert sein. Ein Aufruf der Funktion erfolgt in der Form:

```
FNName(W)
```

wobei W irgendein Wert, eine Variable oder ein arithmetischer Ausdruck ist. Beim Funktionsaufruf werden die folgenden Vorgänge ausgelöst:

1. Zunächst wird der Wert von W berechnet. Danach wird das Ergebnis anstelle der Variablen in der Funktionsdefinition eingesetzt.

2. Der arithmetische Ausdruck in der Funktionsdefinition wird mit dem Wert ausgewertet, der an die Variable übergeben wurde.

3. Das Ergebnis des arithmetischen Ausdrucks wird als Wert der Funktion zurückgegeben.

### Anmerkungen

DEF FN darf nicht im Direktmodus verwendet werden.

Bei der Definition eigener Funktionen können bereits selbst definierte benutzt werden.

Eine Funktion darf nur bei der Definition auf der linken Seite des Gleichheitszeichens stehen, sonst nur auf der rechten.

Auf der rechten Seite des Gleichheitszeichens dürfen bei der Funktionsdefinition numerische Ausdrücke, die auch logische Teilausdrücke enthalten können, auftreten.

## Grundstock für Ihre private Bibliothek von Funktionen

### Mathematische Funktionen

1. `DEF FNREZ(X)=1/X`

   liefert den reziproken Wert von X.

2. `DEF FNBOG(X)=X/(PI/180)` mit PI=3.14159

   wandelt Bogenmaß in Grad um.

3. `DEF FNGRA(X)=X*(PI/180)` mit PI=3.14159

   wandelt Grad in Bogenmaß um.

4. `DEF FNOBW(A)=6*A*A`

   berechnet die Oberfläche eines Würfels.

5. `DEF FNOBK(R)=4*PI*R↑2` mit PI=3.14159

   berechnet die Oberfläche einer Kugel.

6. `DEF FNINW(A)=A*A*A`

   berechnet den Inhalt eines Würfels.

7. `DEF FNINK(R)=4/3*PI*R↑3` mit PI=3.14159

   berechnet den Inhalt einer Kugel.

8. `DEF FNMOD(X)=X-INT(X/M)*M`

   berechnet X modulo M, also den Rest, der sich bei einer Division von X durch M ergibt.

9. `DEF FNIMT(X)=INT(X*(-1))*(-1)`

   rundet stets auf die nächstgrößere Zahl auf.

10. `DEF FNRH(X)=(-1)*(X>1)*(INT(LOG(X*1.2)/LOG(10))+1)`

    liefert die Anzahl der Vorkommastellen einer Zahl.

11. ```
    DEF FNT(K)=1.4189+(12*K)↑(-1)+(LOG(K)-1)*(.5+K)
    DEF FNPM(M)=INT(EXP(FNT(M)-FNT(M-E)-FNT(E))*(-1))*(-1)
    ```

    Kombinationen von E Elementen aus einer Menge M.

**Anwendungsbeispiel:**

```
100 DEF FNT(K)=1.4189+(12*K)↑(-1)+(LOG(K)-1)*(.5+K)
110 DEF FNPM(M)=INT(EXP(FNT(M)-FNT(M-E)-FNT(E))*(-1))*(-1)
120 INPUT"GEBEN SIE DIE GROESSE DER MENGE EIN";M
130 INPUT"GEBEN SIE DIE GROESSE DER TEILMENGE EIN";E
140 PRINT"IN EINER MENGE VON";M;"ELEMENTEN"
150 PRINT"GIBT ES";FNPM(M);"MOEGLICHKEITEN,"
160 PRINT"TEILMENGEN DER GROESSE";E;"ZU BILDEN"
170 END
```

**Anmerkung**

Für Leser, die sich in der Kombinatorik nicht so sehr auskennen, wollen wir das Programm anhand eines kleinen Beispiels per Hand nachvollziehen. Bei den Eingaben 4 für M und 3 für E erfahren wir z. B., daß es 4 Möglichkeiten gibt, Teilmengen mit je 3 Elementen aus einer Grundmenge mit 4 Elementen zu bilden.

**Erklärung:**

M:=(E1,E2,E3,E4)
```

1. Möglichkeit: (E1,E2,E3)
2. Möglichkeit: (E1,E2,E4)
3. Möglichkeit: (E1,E3,E4)
4. Möglichkeit: (E2,E3,E4)

### Trigonometrische Funktionen

20. `DEF FNCOT(X)=1/TAN(X)`
    Cotangens

21. `DEF FNARCCOT(X)=ATN(X)+1.5708`
    Arcus Cotangens

22. `DEF FNSEC(X)=1/COS(X)`
    Sekans

23. `DEF FNCSEC(X)=1/SIN(X)`
    Cosekans

24. `DEF FNARCSEC(X)=ATN(X/SQR(X*X+1))+SGN(SGN(X)-1)*1.5708`
    Arcus Sekans

25. `DEF FNARCCSC(X)=ATN(X/SQR(X*X-1))+(SGN(X)-1)*1.5708`
    Arcus Cosekans

26. `DEF FNARCSIN(X)=ATN(X/SQR(-X*X+1))`
    Arcus Sinus

27. `DEF FNSINH(X)=(EXP(X)-EXP(-X))/2`
    Sinus Hyperbolikus

28. `DEF FNARCCOS(X)=-ATN(X/SQR(-X*X+1))+1.5708`
    Arcus Cosinus

29. `DEF FNCOSH(X)=(EXP(X)+EXP(-X))/2`
    Cosinus Hyperbolikus

30. `DEF FNTANH(X)=EXP(-X)/(EXP(X)+EXP(-X))*2+1`
    Tangens Hyperbolikus

31. `DEF FNCOTH(X)=(EXP(-X)/EXP(X)-EXP(-X))*2+1`

### Textverarbeitungs-Funktionen

```
40. DEF FNKG(BU)=ASC(A$)+32*(95<ASC(A$))
```

BU ist eine Dummy-Variable, d. h. sie ist nur aus formal-syntaktischen Gründen spezifiziert, wird aber nicht für die Berechnung des Funktionswertes verwendet.

FNKG wandelt den Zeichencode von Kleinbuchstaben in den entsprechenden Großbuchstaben-Code um. Dabei muß der Variablen A$, die sozusagen der inoffizielle Parameter von FNKG ist, vor dem Funktionsaufruf ein Wert zugewiesen sein.

**Anwendungsbeispiel:**

```
100 DEF FNKG(BU)=ASC(A$)+32*(95<ASC(A$))
110 A$="a"
120 GA$=CHR$(FNKG(BU))
130 PRINT GA$
140 END
```

```
41. DEF FNZN(ZK)=INT((SP-LEN(A$))/2)
```

Die Funktion FNZN hilft bei der Zentrierung einer Zeichenkette, die kürzer als die Spaltenzahl des Bildschirms oder des Druckers ist. ZK ist eine Dummy-Variable, deren Wert unerheblich ist. Die Variablen SP und A$ sind die inoffiziellen Parameter der Funktion, die vor deren Aufruf gesetzt werden müssen.

**Anwendungsbeispiel:**

```
100 DEF FNZN(ZK)=INT((SP-LEN(A$))/2)
110 SP=40 :REM 40-ZEILEN-SCHIRM
120 A$="MITTE"
130 FOR I=1 TO FNZN(ZK)
140 : A$=" "+A$ :REM LINKS AUFFUELLEN
150 NEXT I
160 PRINT A$
170 END
```

```
42. DEF FNUM(BU)=ABS(1*(ASC(A$)=187)+
2*(ASC(A$)=188)+3*(ASC(A$)=189)+
4*(ASC(A$)=190)+5*(ASC(A$)=219)+
6*(ASC(A$)=220)+7*(ASC(A$)=221))
```

Die Funktion FNUM wandelt die Umlaute

ä, ö, ü, Ä, Ö, Ü und ß

in

ae, oe, ue, AE, OE, UE und ss

um. Eine solche Funktion werden Sie schätzen lernen, wenn Sie einen Commodore-Rechner mit DIN-Zeichensatz zusammen mit einem Drucker benutzen wollen, der keine Umlaute drucken kann. Auch wenn Sie Wörter mit Umlauten sortieren müssen, brauchen Sie die Funktion FNUM, damit die Sortierung richtig funktioniert.

Die Variable BU ist wieder eine Dummy-Variable, deren Wert für die Funktion unerheblich ist. Der inoffizielle Parameter für diese Funktion, A$, muß vor dem Funktionsaufruf zugewiesen werden.

**Anwendungsbeispiel:**

```
100 DEF FNUM(BU)=ABS(1*(ASC(A$)=187)+
     2*(ASC(A$)=188)+3*(ASC(A$)=189)+
     4*(ASC(A$)=190)+5*(ASC(A$)=219)+
     6*(ASC(A$)=220)+7*(ASC(A$)=221))
110 UM$(1)="ae":UM$(2)="oe":UM$(3)="ue":
    UM$(4)="ss":UM$(5)="AE":UM$(6)="OE":
    UM$(7)="UE"
120 AF$=CHR$(34)
130 B$="äöüßÄÖÜ"
140 PRINT"TABELLE FUER COMMODORE DIN-ASC"
150 PRINT"FUER ASCII-DRUCKER ODER FUER"
160 PRINT"TEXTANALYSE-ZWECKE SOWIE SORTIERUNG"
170 FOR I=1 TO LEN(B$)
180 : A$=MID$(B$,I,1)
190 : PRINT"CHR$(";ASC(A$);AF$;UM$(FNUM(BU))+AF$;
200 NEXT I
210 END
```

Sie können dieselbe Funktion für die Umwandlung von Steuerzeichen in lesbare und druckbare CHR$-Codes verwenden. Ersetzen Sie einfach die Codes für die Umlaute in der Funktionsdefinition durch Steuerzeichen-Codes, und die Umlauttabelle UM$ muß entsprechend zugeordnet werden.

Beispielsweise könnte es in der Funktionsdefinition

```
1*(ASC(A$)=18)
```

heißen und in der Umlauttabelle dafür:

```
UM$(1)="ZR$"   :REM REVERS AN
                    SIEHE CHR$-CODES
```

Eine andere Möglichkeit ist es, die Funktion zur Umsetzung von Werten für bestimmte Token des C128 und anderer Rechner oder auch bei Spracherweiterungen zu nehmen.

43. `DEF FNIN(I)=(-I*(SW$=MID$(TX$,I,LEN(SW$))))`

Mit der Funktion FNIN wird die INSTR-Funktion, die es nur in BASIC 3.5 und 7.0 gibt, simuliert. Dabei wird die Position einer Zeichenkette SW$ in der Zeichenkette TX$ ermittelt.

**Anwendungsbeispiel:**

```
100 REM*********************************
110 REM*       INSTR-SIMULATION          *
120 REM*********************************
130 SW$="TESTWORT":REM GESUCHTES WORT
140 TX$="HIER IST DAS TESTWORT ZU FINDEN"
    : REM DURCHSUCHTE ZEICHENKETTE
150 DEF FNIN(I)=(-I*(SW$=MID$(TX$,I,LEN(SW$))))
160 FOR I=1 TO (LEN(TX$)-LEN(SW$)+1)
170 : IF I=FNIN(I) THEN 190
180 NEXT I
190 PRINT I
```

## Systemfunktionen

50. Umwandlung der Bildschirm-Codes in ASCII-Code:

```
DEF FNBP(I)=PEEK(1023+I)

DEF FNB1(I)=FNBP(I)*ABS(FNBP(I)<128)
DEF FNB2(I)=  64   *ABS(FNBP(I)< 32)
DEF FNB3(I)=  32   *ABS(FNBP(I)> 63 AND FNBP(I)< 96)
DEF FNB4(I)=  64   *ABS(FNBP(I)> 96 AND FNBP(I)<129)
DEF FNB5(I)=(FNBP(I)-128)*(FNBP(I)>128)

DEF FNBS(I)=FNB1(I)+FNB2(I)+FNB3(I)+FNB4(I)+FNB5(I)
```

Die erste Funktion FNBP liest den Bildschirm-Code mit Hilfe der PEEK-Funktion, wenn Sie die Variable I von 1 bis 999 laufenlassen. Die nächsten fünf Funktionen, FNB1 bis FNB5, sind Hilfsfunktionen, die den Bildschirm-Code entsprechend der ASCII-Code-Tabelle umsetzen. Die letzte Funktion FNBS schließlich enthält den richtigen ASCII-Code für das zugehörige Zeichen auf dem Bildschirm. Was würde näher liegen, als mit dieser Funktion einen Bildschirm-Ausdruck auf den Drucker zu bringen? Die Anwendung der Funktionen in einer Hardcopy-Routine finden Sie als Beispielprogramm unter dem Stichwort PEEK.

```
51. DEF FNPX(X)=INT((PEN(0)-60)/KV)
    DEF FNPY(Y)=INT((PEN(1)-50)/KH)
```

Die Funktionen FNPX und FXPY fragen die Koordinaten X und Y des Lichtstifts ab und berechnen die zugehörigen Koordinaten für den Textbildschirm. Beim 40-Zeichen-Schirm betragen dabei die Korrekturkonstanten KV=6.499999999 und KH=6. Ein vollständiges Anwendungsbeispiel finden Sie unter dem Stichwort PEN.

```
52. DEF FNDEEK(N)=PEEK(N)+256*PEEK(N+1)
```

Mit der Funktion FNDEEK kann eine 16stellige Speicheradresse gelesen werden. Sie ist wertvoll, wenn man zwei aufeinanderfolgende Adressen lesen will.

```
53. DEF FNBB(45)=FNDEEK(45)
```

Die Funktion liefert die BASIC-Anfangsadresse.

```
54. DEF FNHA(AD)=INT(AD/256)
```

Mit dieser Funktion kann das höherwertige linke Adreßbyte isoliert werden, was häufig für Adreßrechnungen benötigt wird.

```
55. DEF FNLA(AD)=AD-FNHA(AD)*256
```

Diese Funktion isoliert das niederwertige rechte Adreßbyte, was ebenfalls für Adreßrechnungen benötigt wird.

```
56. DEF FNAC(A)=ASC(A$+" ")+32*(LEN(A$)=0)
```

Die Funktion FNAC liefert den ASCII-Code eines Zeichens, auch wenn es sich um die leere Zeichenkette handelt. Das war bisher außer in BASIC 7.0 mit der Funktion ASC nicht möglich. Eine solche Funktion ist nützlich im Zusammenhang mit dem Lesen von Dateien, da das Betriebssystem beim Einlesen des

Datei-Endekennzeichens, CHR$(0), dies in eine Zeichenkette der Länge 0, das heißt eine leere Zeichenkette, umwandelt.

```
57. DEF FNOF(I)=1.7*10↑37*(MID$(HX$,I,1)>"F"
                          OR MID$(HX$,I,1)<"0"
                          OR LEN(HX$)<>4)*99

    DEF FND1(I)=ASC(MID$(HX$,I,1)+" ")+48*(MID$(HX$,I,1)<":"
                AND MID$(HX$,I,1)>"/")
    DEF FND2(I)=FNOF(I)+55*MID$(HX$,I,1)<"G"
                          AND MID$(HX$,I,1)>"@")
    DEF FND3(I)=FND1(I)+FND2(I)

    DEF FNDC(HX)=FND3(1)*4096+FND3(2)*256+FND3(3)*16+FND3(4)*
    16↑0
```

Die Funktion FNDC rechnet eine vierstellige Hexadezimalzahl in eine Dezimalzahl um. Der inoffizielle Parameter HX$ muß dabei vor dem Aufruf der Funktion eine gültige vierstellige hexadezimale Zeichenkette, also mit den Ziffern 0 bis 9 und den Buchstaben A bis F, enthalten. Auch Leerstellen sind im Gegensatz zur DEC-Funktion, die in BASIC 3.5 und 7.0 für diese Zwecke zur Verfügung steht, nicht erlaubt.

Die Funktion FNOF, die als erste definiert wird, erzeugt die Fehlermeldung

```
?OVERFLOW ERROR IN Zeilennummer
```

falls in der Variablen HX$ unzulässige Werte stehen. Eine solche Fehlermeldung können Sie auch bei Ihren anderen selbst definierten Funktionen mit derselben Logik selbst erzeugen. Die Funktionen FND1, FND2 und FND3 sind Hilfsfunktionen.

Die Funktion FNDC bietet nicht nur die Möglichkeit, die DEC-Funktion in den BASIC-Versionen 2.0 und 4.0 zu verwenden,sondern läßt durch leichte Modifikation sogar zu, daß auch längere als vierstellige hexadezimale Zeichenketten in Dezimalzahlen umgewandelt werden. Zu diesem Zweck müssen nur weitere Summanden der Funktion FND3 bei der Addition in der Funktion FNDC hinzugefügt werden.

Dabei muß entsprechend der Wertigkeit der Hexstelle FND3 mit einer weiteren Potenz von 16 multipliziert werden. Für eine fünfstellige Hexadezimalzahl würde der Faktor 4096*16*FND3(1), also 65536*FND3(1), betragen. Dieser Summand wäre dann der erste in der Summe mit dem Parameter 1. Die Parameter der folgenden Summanden werden entsprechend umnumeriert: FND3(2), FND3(3) usw. Außerdem muß die Abfrage auf fehlerhafte Länge in der Fehlermeldungsfunktion FNOF von 4 auf 5 erhöht werden.

Ein Testbeispiel finden Sie unter dem Stichwort DEC.

58.
```
DEF FNOV(I)=1.7*10↑37*(MID$(BI$,I+1,1)<"0" OR
                       MID$(BI$,I+1,1)>"1" OR
                       LEN(BI$)<>16)*99
DEF FNB1(I)=0+(MID$(BI$,I+1,1)="1")*2↑(15-I)*(-1)
DEF FNB8(I)=FNB1(0)+FNB1(1)+FNB1(2)+FNB1(3)
           +FNB1(4)+FNB1(5)+FNB1(6)+FNB1(7)
DEF FNBF(BI)=FNB1(8)+FNB1(9)+FNB1(10)+FNB1(11)
            +FNB1(12)+FNB1(13)+FNB1(14)+FNB1(15)

DEF FNBI(BI)=INT(FNBF(BI)+FNB8(I))+FNOV(I)
```

Die Funktion FNBI berechnet aus einer 16stelligen Binärzahl die zugehörige Dezimalzahl. Dabei ist BI ein Dummy-Argument, während die Zeichenkette BI$ das inoffizielle Argument der Funktion ist. BI$ muß also vor dem ersten Aufruf der Funktion zugewiesen werden. So liefert z. B. im Programm

```
200 BI$="1000000000000011":PRINT FNBI(BI)
```

den Wert 32771.

FNOV ist unsere Fehlerfunktion, die sich rührt, wenn andere Zeichen als 0 oder 1 in der Zeichenkette BI$ stehen oder wenn BI$ nicht 16 Zeichen lang ist.

Die Funktionen FNB1, FNB8 und FNBF sind wieder Hilfsfunktionen.

Die Funktion kann leicht abgewandelt werden, so daß sie beispielsweise 8stellige Binärzahlen umrechnet. Ändern Sie dazu die Längenabfrage in der Fehlerfunktion FNOV von 16 auf 8. In der Funktion FNB1 wird der Potenzierungsexponent auf 7- I anstatt 15-I gesetzt, und die Funktion FNBF fällt völlig weg.

59.
```
DEF FNSK(DZ)=40*10↑-INT(LOG(ABS(DZ*2))/LOG(10)-1)
```

FNSK ist eine Skalierungsfunktion für Dezimalzahlen, die nicht 0 sein dürfen, für die Darstellung mit SCALE 1,500,500. DZ muß vor der Anwendung auf 0 abgefragt werden. Für die Skalierung muß die Funktion mit dem betragsmäßig größten Wert aufgerufen werden. Die Multiplikation aller Ausgaben mit dem Faktor SK, den die Funktion FNSK liefert, stellt sicher, daß alle Ausgaben auf den Bildschirm passen. Unter dem Stichwort SCALE finden Sie ein vollständiges Beispielprogramm.

60.
```
DEF FNHA(I)=INT((-58109*(I=0)-64256*(I=1)-FRE(I))/256)
DEF FNLA(I)=-58109*(I=0)-64256*(I=1)-FRE(I)-256*FNHA(I)
```

```
DEF FNHP(I)=FNHA(I)-28*(I=0)
     :REM ANDERE RECHNER FNHP(I)=PEEK(202)
DEF FNLP(I)=FNLA(I)-1*(I=0)
     :REM ANDERE RECHNER FNLP(I)=PEEK(201)
```

Die Funktionen FNHA und FNLA liefern die Länge eines Programms, das sich im Arbeitsspeicher befindet, und FNHP und FNLP liefern die POKE-Werte für seine Endadresse. Wenn man diese Adressen kennt, kann man beispielsweise Programme aneinanderhängen. Ein Beispiel für die Verwendung dieser Funktionen finden Sie in Kapitel 9.

61.
```
DEF FNDK(AD)=PEEK(AD)+256*PEEK(AD+1)
DEF FNLZ(AD)=PEEK(AD-1)
DEF FNAZ(AD)=PEEK(FNDK(AD+1)+I)
DEF FNGZ(AD)=FNLZ(AD+I)
```

Diese Funktionen bewerkstelligen ein Auslesen des Variablenspeichers, nachdem mit der POINTER-Funktion die Adresse der betreffenden Variable ermittelt worden ist. Unter dem Stichwort POINTER haben wir ein vollständiges Anwendungsbeispiel aufgeführt.

## (BASIC 2.0*, 4.0*, 3.5, 7.0) DELETE

**Abkürzung:** DE<SHIFT>L

**Typ:** Kommando

**Syntax:**
```
DELETE Z
DELETE Z1-Z2
DELETE Z1-
DELETE -Z2
```

**Parameterbereich:** Zeilennummern

Das Kommando DELETE löscht eine oder mehrere Programmzeilen. Dabei löscht

```
DELETE Z
```

nur die Zeile mit der Nummer Z.

```
DELETE Z1-Z2
```

löscht alle Programmzeilen von Z1 bis Z2 einschließlich. Den ganzen Rest des Programms ab Zeile Z1 läßt das Kommando

```
DELETE Z1-
```

verschwinden, und das Kommando

```
DELETE -Z2
```

löscht das Programm vom Anfang bis zur Zeile Z2.

### Anmerkungen

Eine einzelne Zeile läßt sich bequemer durch Eingabe ihrer Nummer und anschließendes Drücken von RETURN löschen:

```
100 <RETURN>
```

Das Kommando DELETE kann nur im Direktmodus angewendet werden.

# DIM (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** D<SHIFT>I

**Typ:** Befehlswort

**Syntax:** DIM Variable(ganze Zahl, ganze Zahl,...)

**Parameterbereich:** Variablennamen

**Argumentbereich:** positive ganze Zahlen

DIM (Abkürzung für Dimension) erlaubt dem Programmierer, eine Variable als Feld zu definieren. Ein Feld ist eine Liste von Variablen, die durchnumeriert sind. Die Nummer einer solchen Variablen heißt daher auch Index.

Neben dem Namen benötigt man daher zur Definition eines Feldes auch die Anzahl der einzelnen Feldelemente. Die Anzahl nennt man Dimension des Feldes. Wir sprechen manchmal von einem eindimensionalen Feld als einer "Liste" von Variablen und von einem zweidimensionalen Feld als einer "Tabelle".

```
DIM A(10)
```

definiert z. B. ein eindimensionales Feld mit dem Namen A, das 11 Elemente - A(0), A(1), ..., A(10) - aufnehmen kann.

```
DIM B$(4,4)
```

hingegen legt ein zweidimensionales Feld mit dem Namen B$ und 25 Elementen an.

Der Name selbst kennzeichnet das Feld. Ein Feld, dessen Name mit dem %-Zeichen endet, kann nur ganze Zahlen speichern. Endet der Name eines Feldes mit $, werden Zeichenketten erwartet. Die Namen von Feldern mit reellen Zahlen enden mit einem Buchstaben oder einer Ziffer.

Ein- oder zweidimensionale Felder mit höchstens 11 Elementen können ohne DIM-Anweisung verwendet werden. Sie werden dann automatisch angelegt, wenn sie das erste Mal aufgerufen werden.

DIM kann als sofort auszuführender Befehl oder als Programmanweisung eingesetzt werden. Es können auch mehrere Felder in einer DIM-Anweisung definiert werden, z. B.:

```
DIM A$(8), B(2,3), C%(15)
```

Durch die DIM-Anweisung wird für die Variable Speicherplatz reserviert. Diese Reservierung ist natürlich nur möglich, wenn noch genügend freier Speicherplatz vorhanden ist.

So ist z. B. beim C128 in der Grundausstattung das größte mögliche zweidimensionale Feld ein Integer-Feld mit 178 Zeilen und Spalten:

```
DIM A%(178,178)
```

was 31684 Feldelementen entspricht.

Für ein zweidimensionales Gleitkommazahlenfeld liegt die Grenze bei 112 Zeilen und Spalten:

```
DIM B(112,112)
```

was 12544 Feldelementen entspricht.

Zweidimensionale Zeichenkettenfelder dürfen höchstens 21025 Elemente enthalten, also 145 Zeilen und Spalten, besitzen:

```
DIM C$(145,145)
```

In einem eindimensionalen Zeichenkettenfeld lassen sich maximal 21415 Elemente abspeichern:

```
DIM C$(21415)
```

Der BASIC-Interpreter läßt zwar maximal 255 Dimensionen zu, aber beim C128 ist schon bei 14 Schluß. Auch wenn man nur 0 und 1 als Werte pro Dimension zuläßt, benötigt A% schon 32768 einzelne Feldelemente. Größere sinnvolle Felder sind von der Dimension her nicht möglich.

Bei

```
DIM A%(1,1,1,1,1,1,1,1,1,1,1,1,1,1)
```

ist Schluß, obwohl bei diesem Feld jede Dimension nur zwei Einheiten groß ist. Maximal kann man ein Zeichenkettenfeld mit 69 Dimensionen definieren,

die alle die Größe 0 haben, was aber wenig Sinn macht, da es nur ein Feldelement enthält.

Generell gilt, daß die maximal wählbare Größe eines Feldes vom freien Speicherplatz, dem Variablentyp und der Anzahl der Dimensionen abhängt.

Falls nur noch wenig freier Speicherplatz zur Verfügung steht, kann es sinnvoll sein, durch Konvertierung und Beschränkung auf eine Dimension eine bessere Speicherausnutzung auf Kosten eines höheren Programmieraufwandes und etwas längerer Rechenzeit zu erkaufen.

Dimensionieren Sie Ihre Felder stets zu Beginn des Programms, und zwar in der Reihenfolge Integer-, Gleitkomma- und dann Zeichenkettenfelder, da dies für Ihren Rechner weniger Speicherverwaltungsaufwand bedeutet.

Verhindern Sie evtl. durch eine entsprechende Abfrage, daß das Programmsegment, das die Dimensionierungen vornimmt, ein zweites Mal durchlaufen wird und so die Dimensionierung für ein Feld zweimal vorgenommen wird, da dies zu der Fehlermeldung

```
?REDIM'D ARRAY ERROR
```

führt. Diese Abfrage könnte so aussehen:

```
1000 IF S THEN GOTO 2000 SCHON DIMENSIONIERT
1010 S=NOT S
1020 DIM A(20)
  .
  .
  .
1990 GOTO 1000
2000 PRINT "DIE FELDER SIND SCHON DIMENSIONIERT"
2010 END
```

**Beispiel**

Das Beispielprogramm verlangt die Eingabe eines Datums, das anschließend auf Richtigkeit geprüft wird. Ferner berechnet es, um den wievielten Tag im Jahr es sich handelt. Dazu wird in Zeile 190 ein Feld mit der Dimension 12 definiert. In Zeile 200 erhalten die einzelnen Feldelemente Zahlenwerte, die den Tagen der einzelnen Monate entsprechen. Die Zeile 240 setzt das Element MO(2) auf 29, falls das Jahr ein Schaltjahr ist. Dies ist bekanntlich der Fall, wenn die Jahreszahl durch 4 teilbar ist, außer es handelt sich um ein nicht durch 400 teilbares Jahrhundert. Demnach war das Jahr 1900 kein Schaltjahr. Das Jahr 2000 ist wieder eines.

Die Feststellung, ob ein falsches Datum vorliegt, ist dann in Zeile 250 nur noch eine einfache Abfrage. In den Zeilen 270 bis 310 wird dann der Tag im Jahr berechnet.

```
100 PRINT CHR$(147)
110 PRINT"    D A T U M S P R U E F U N G"
120 PRINT"    --------------------------"
    : PRINT
130 INPUT"GEBEN SIE DEN TAG EIN";T
140 IF T<1 OR T>31 THEN PRINT"FALSCHER TAG"
    : GOTO 130
150 INPUT"GEBEN SIE DEN MONAT EIN";M
160 IF M<1 OR M>12 THEN PRINT"FALSCHER MONAT"
    : GOTO 150
170 INPUT"GEBEN SIE DAS JAHR EIN(4STELLIG)";J
180 IF J<1000 OR J>9999 GOTO 170
190 DIM MO(12)
200 DATA 31,28,31,30,31,30,31,31,30,31,30,31
210 FOR I=1 TO 12
220 : READ MO(I)
230 NEXT I
240 IF (J/4)=INT(J/4) AND (J/400)<>INT(J/400)
    THEN MO(2)=29
250 IF T<=MO(M) THEN PRINT"DAS DATUM IST RICHTIG"
260 IF T>MO(M) THEN PRINT"DAS DATUM IST FALSCH"
    :GOTO 330
270 A=T
280 IF M=1 GOTO 320
290 FOR I=1 TO M-1
300 : A=A+MO(I)
310 NEXT I
320 PRINT"TAG IM JAHR IST";A
330 END
```

Diese Prüfroutine kann bei geeigneter Numerierung als Unterprogramm in alle Programme eingebaut werden, in denen der Anwender aufgefordert wird, ein Datum einzugeben.

### Anmerkungen

Achten Sie bitte darauf, daß in BASIC nach guter nordamerikanischer Tradition auch bei den Feldern von null an gezählt wird.

| | |
|---|---|
| `DIM A(0)` | enthält also ein Element. |
| `DIM A(1)` | enthält zwei Elemente. |
| `DIM A(0,1)` | enthält zwei Elemente. |
| `DIM A(1,1)` | enthält vier Elemente, nämlich `A(0,0)`, `A(1,0)`, `A(0,1)` und `A(1,1)`. |

# (BASIC 4.0, 3.5, 7.0) DIRECTORY

**Abkürzung:** DI<SHIFT>R

**Typ:** Systembefehl für Diskettenlaufwerk

**Syntax:**

```
DIRECTORY DLaufwerknummer

DIRECTORY DLaufwerknummer,
UGerätenummer,"Dateiname"
```

**Parameterbereich:** Laufwerknummern
Gerätenummern
Dateinamen

Mit diesem Befehl kann man sich das Inhaltsverzeichnis einer Diskette ansehen. Dabei wird das Inhaltsverzeichnis an eine freie Speicherstelle des Betriebssystems geschrieben wird, und ein im RAM vorhandenes BASIC-Programm bleibt erhalten.

Wenn zwei Laufwerke angeschlossen sind, wird das Inhaltsverzeichnis von beiden hintereinander auf dem Bildschirm ausgegeben. Wünscht man nur eins zu sehen, kann man die Laufwerknummer angeben.

Der Dateiname kann Platzhalter enthalten, z. B. das Fragezeichen für ein einzelnes Zeichen und den Stern * für den ganzen Rest eines Dateinamens.

```
DIRECTORY D1,U8,"?ROG*"
```

gibt alle Dateien von Laufwerk D1 aus, deren zweiter bis vierter Buchstabe ROG ist, also z. B.:

```
PROG1
PROG2
FROGGER
```

Bei der Anzeige auf dem Bildschirm erscheint nach dem DIRECTORY-Befehl die oberste Zeile in inverser Schrift. Sie enthält u. a. den Diskettennamen. Darunter werden nun die Dateien aufgeführt, die auf der Diskette gespeichert sind.

Die Zahl links neben dem Dateinamen gibt dabei die Anzahl der Blöcke an, die die Datei belegt. Hinter dem Dateinamen steht eine Kennung, die den Dateityp dokumentiert. Dabei steht

| | |
|---|---|
| PRG | für Programmdatei |
| SEQ | für sequentielle Datei |
| REL | für relative Datei |
| USR | für User-(Anwender-)Datei |

### Anmerkungen

Beim VC20 und C64 gibt es keinen speziellen Befehl zum Ansehen des Inhaltsverzeichnisses. Man muß statt dessen

```
LOAD"$",8
```

und anschließend

```
LIST
```

eingeben. Dabei geht allerdings das im Arbeitsspeicher stehende Programm verloren. Einen Ausweg für dieses Problem bietet das Beispielprogramm unter dem Stichwort GET#.

## (BASIC 4.0, 3.5, 7.0) DLOAD

**Abkürzung:** D<SHIFT>L

**Typ:** Systembefehl für Eingabe

**Syntax:**
```
DLOAD "Dateiname"

DLOAD "Dateiname",DLaufwerknummer,
UGerätenummer
```

**Parameterbereich:** Dateinamen
Laufwerknummern
Gerätenummern

Der Befehl DLOAD überträgt ein Programm von der Diskette in den Arbeitsspeicher. Dabei muß der Dateiname spezifiziert werden, z. B.:

```
DLOAD "TEST"
```

gleichbedeutend mit

```
DLOAD "TEST",D0,U8
```

Wenn nur ein Laufwerk angeschlossen ist oder wenn sich die gesuchte Datei im Laufwerk mit der Nummer D0 befindet, kann die Angabe der Laufwerknummer weggelassen werden.

Für die Angabe des Dateinamens können teilqualifizierende Namen und Variablen verwendet werden. Dabei kann ? die Stelle eines beliebigen einzelnen Zeichens an einer Stelle im Dateinamen annehmen. Mit * kann für eine ganze Teilzeichenkette, die den Abschluß des Namens bildet, ein Ersatz geschaffen werden. ? und * nennt man in diesem Zusammenhang auch Joker.

### Beispiel

```
DLOAD"??B*"
```

findet und lädt z. B. das Programm SYBEX von der Diskette im Laufwerk D0.

```
S$="SYBEX"
DLOAD(S$),D1
```

findet und lädt das Programm SYBEX von der Diskette im Laufwerk D1.

### Anmerkungen

In BASIC 2.0 heißt der entsprechende Befehl LOAD (siehe dort).

Die zugehörigen Kommandos zum Speichern von Dateien heißen DSAVE bzw. SAVE (siehe dort).

Bei DLOAD werden, falls Sie keine besonderen Vorkehrungen treffen, alle Variableninhalte und das vorher im Speicher befindliche Programm gelöscht.

# (BASIC 3.5, 7.0) DO LOOP EXIT

**Abkürzung:** DO LO<SHIFT>O EX<SHIFT>I

**Typ:** elementare Anweisung

**Syntax:**
```
DO: Anweisung
  Anweisung:EXIT
LOOP
```

Der Befehl DO leitet einen Programmabschnitt ein, der mit dem Befehl LOOP endet. Da sich der Interpreter beim Programmlauf die Adresse des DO-Befehls gemerkt hat, geht er nach Erreichen von LOOP wieder dorthin zurück.

Diese Endlosschleife kann nur mit dem Befehl EXIT verlassen werden. Dann wird der nächste Befehl, der dem LOOP folgt, ausgeführt.

Die DO/LOOP/EXIT-Konstruktion kann in den BASIC-Versionen 2.0 und 4.0 folgendermaßen nachgebildet werden:

```
100 FOR DO=0 TO 1:DO=0:REM DO
110 PRINT"SCHLEIFE"
120 IF X=5 THEN 140    :REM EXIT
130 NEXT DO            :REM LOOP
140 END:REM ODER JEDE BELIEBIGE ANWEISUNG
```

Die Schleife wird so lange wiederholt ausgeführt, bis die in Zeile 120 stehende Bedingung erfüllt ist.

# DO LOOP UNTIL (BASIC 3.5, 7.0)

**Abkürzung:** DO LO<SHIFT>O U<SHIFT>N

**Typ:** elementare Anweisung

**Syntax:**
```
DO:Anweisung
  Anweisungen
LOOP UNTIL logischer Ausdruck
```

Dieser Befehl leitet einen Programmteil ein, der in einer Schleife so lange abgearbeitet wird, bis eine bestimmte Bedingung, die hinter UNTIL spezifiziert ist, eingetreten ist. Dann wird die Schleife verlassen.

Auch diese Schleifenkonstruktion kann in BASIC 2.0 und 4.0 nachgebildet werden.

```
100 FOR DO=0 TO 1:DO=0     :REM DO
110 : IF UNTIL>9 THEN 140 :REM UNTIL
120 : PRINT"SCHLEIFE"
130 NEXT DO                :REM LOOP
140 END:REM ODER JEDE BELIEBIGE ANWEISUNG
```

# (BASIC 3.5, 7.0) DO LOOP WHILE

**Abkürzung:** DO LO<SHIFT>O W<SHIFT>H

**Typ:** elementare Anweisung

**Syntax:**
```
DO:Anweisung
  Anweisungen
LOOP WHILE
```

Dieser Befehl ist ein geeignetes Hilfsmittel, um mit BASIC strukturiert zu programmieren. Er leitet eine Schleife ein, die so lange wiederholt wird, wie die Bedingung, die hinter WHILE spezifiziert ist, noch erfüllt ist. Ist sie nicht mehr erfüllt, wird die Schleife kein weiteres Mal durchlaufen.

Die Nachbildung der DO/LOOP/WHILE-Logik sieht in BASIC 2.0 und 4.0 folgendermaßen aus:

```
100 FOR DO=0 TO 1:DO=0          :REM DO
110 : IF WHILE+NOT(A<9) THEN 140:REM WHILE
120 : PRINT"SCHLEIFE"
130 NEXT DO                     :REM LOOP
140 END:REM ODER JEDE BELIEBIGE ANWEISUNG
```

## Beispiel

In unserem Beispielprogramm wollen wir die Primzahlen bis zur Zahl 400 berechnen. Sie können diese Grenze beliebig verändern. Dazu müssen nur die Konstanten in den Zeilen 110 und 130 geändert werden.

```
100 PRINT CHR$(147)
110 K=200
120 PRINT"BERECHNUNG DER PRIMZAHLEN BIS";2*K
130 DIM M(200)
140 ZAEHL=0
150 FOR I=0 TO K
160 : M(I)=1
170 NEXT I
180 FOR I=0 TO K
```

```
190 : IF M(I)<>0 THEN
      BEGIN
200 :   PRIM=I+I+3
210 :   PRINT PRIM,;
220 :   Z=I+PRIM
230 :   DO WHILE Z<=K
240 :      M(Z)=0
250 :      Z=Z+PRIM
260 :   LOOP
270 :   ZAEHL=ZAEHL+1
280 : BEND
290 NEXT I
300 PRINT
310 PRINT ZAEHL;"PRIMZAHLEN GEFUNDEN"
320 END
```

# (BASIC 4.0, 7.0) DOPEN

**Abkürzung:** D<SHIFT>O

**Typ:** Systembefehl für Ein-/Ausgabe

**Syntax:**

```
DOPEN#Dateinummer,"Dateiname"

DOPEN#Dateinummer,"Dateiname",
DLaufwerknummer,UGerätenummer

DOPEN#Dateinummer,"Dateiname",
LSatzlänge oder W
```

**Parameterbereich:** Dateinummern
Dateinamen
Laufwerknummern
Gerätenummern

Der Befehl DOPEN öffnet die über Dateinummer und Dateiname spezifizierte Datei zum Lesen oder Schreiben.

Falls nur ein Laufwerk angeschlossen ist oder wenn sich die Datei auf Laufwerk D0 befindet, kann die Laufwerknummer entfallen.

Die Parameter L mit Angabe einer Satzlänge und W sind nur notwendig, wenn die Datei zum Schreiben geöffnet wird. Sind sie nicht angegeben, so ist die Datei zum Lesen geöffnet worden. Der Parameter L wird dabei nur bei relativen Dateien benötigt, um die Satzlänge, beispielsweise L38, anzugeben.

### Beispiele

```
DOPEN#1,"DATEI1",D1,U8
```

öffnet die Datei mit dem Namen DATEI1 in Laufwerk D1 zum Lesen.

```
DOPEN#1,"DATEI2",L25
```

öffnet die relative Datei mit dem Namen DATEI2 und der Satzlänge 25 zum Schreiben.

### Anmerkungen

Mit DOPEN werden den Ein-/ und Ausgabevorgängen auch Pufferbereiche zugeordnet, so daß sich in der Regel mindestens ein Satz, also ein Ergebnis eines Schreib- oder Lesevorgangs, dort befindet. Diese Information kann verlorengehen, wenn der Ein-/Ausgabevorgang nicht ordnungsgemäß mit CLOSE oder DCLOSE beendet wird.

Außerdem bleibt eine mit DOPEN geöffnete Datei bei nicht ordnungsgemäßem Abschluß im Inhaltsverzeichnis der Diskette weiterhin als geöffnet gekennzeichnet, und die Datei erhält kein Endekennzeichen hinter dem letzten Satz. Dies kann zu großen Problemen bei der weiteren Verarbeitung führen.

## (BASIC 3.5, 7.0) DRAW

**Abkürzung:** D<SHIFT>R

**Typ:** Grafikbefehl

**Syntax:** DRAW F,X1,Y1 TO X2,Y2 TO X3,Y3 TO ..

**Parameterbereich:** F - ganze Zahlen 0 - 3
Grafik-Bildschirmkoordinaten

Die Anweisung DRAW dient zum Verbinden zweier Grafikpunkte mit einer geraden Linie. Im Spezialfall, wenn die Koordinaten der beiden Punkte gleich sind, entsteht nur ein einzelner Punkt. Die Parameter enthalten dabei folgende Angaben:

| | |
|---|---|
| **F** | Farbquelle (kann entfallen, dann 1)<br>0 - Hintergrund<br>1 - Vordergrund<br>2 - Mehrfarbenmodus 1<br>3 - Mehrfarbenmodus 2 |
| **X1** | X-Koordinate des Startpunktes (kann entfallen, dann Cursorposition) |
| **Y1** | Y-Koordinate des Startpunktes (kann entfallen, dann Cursorposition) |
| **X2** | X-Koordinate des Endpunktes (kann entfallen, dann wie X1) |
| **Y2** | Y-Koordinate des Endpunktes (kann entfallen, dann wie Y1) |

### Beispiel

Ein vollständiges Beispiel für den C128 finden Sie unter dem Stichwort SPRITE.

# DS (BASIC 3.5, 7.0)

**Abkürzung:** keine möglich

**Typ:** reservierte Variable

**Funktion:** Fehlernummer bei Diskettenzugriff bereitstellen

In der Variablen DS befindet sich nach dem Auftreten eines Fehlers bei Diskettenzugriff die entsprechende Fehlernummer. Mit

```
PRINT DS
```

kann diese Fehlernummer abgefragt werden. Geben Sie z. B. den Befehl

```
OPEN 1,8,3,"XYLOPHON"
```

ein. Angenommen Sie haben die Datei mit dem Namen XYLOPHON nicht auf Ihrer Diskette, dann erhalten Sie nach Abfrage mit DS die Fehlernummer 62.

## Anmerkungen

Den zugehörigen Fehlertext kann man in der Zeichenkettenvariablen DS$ (siehe dort) abfragen.

## (BASIC 3.5, 7.0) DS$

**Abkürzung:** keine möglich

**Typ:** reservierte Variable

In der Variablen DS$ kann der Fehlertext nach Auftreten eines Fehlers bei Diskettenzugriff abgefragt werden. Geben Sie einmal absichtlich den folgenden nicht ausführbaren Befehl ein:

```
OPEN 1,8,3,"TROMPETE"
```

falls Sie keine Datei dieses Namens auf der Diskette haben. Mit

```
PRINT DS$
```

erfahren Sie dann, um welchen Fehler es sich handelt:

```
62, FILE NOT FOUND,00,00
```

Die Fehlernummer 62 ist dieselbe, die Sie auch in der Variablen DS (siehe dort) abfragen könnten. Der Text, der sich anschließt, bedeutet, daß die Datei auf der Diskette nicht gefunden wurde. Die beiden sich anschließenden Zahlen geben die Spurnummer der Diskette und die Sektor/Blocknummer an, bei der der Fehler aufgetreten ist.

# DSAVE (BASIC 4.0, 3.5, 7.0)

**Abkürzung:** D<SHIFT>S

**Typ:** Systembefehl für Ausgabe

**Syntax:**
```
DSAVE "Dateiname"

DSAVE "Dateiname",DLaufwerknummer,
UGerätenummer
```

**Parameterbereich:** Dateinamen
Laufwerknummern
Gerätenummern

Mit dem Systembefehl DSAVE können Sie Programme, die sich im Rechnerspeicher befinden, auf Diskette abspeichern. Dabei ist ein Dateiname anzugeben, z. B.:

```
DSAVE "TEST",D1,U8
```

Wenn nur ein Diskettenlaufwerk angeschlossen ist oder das Laufwerk D0 angesprochen werden soll, kann die Laufwerknummer weggelassen werden.

Das Programm ist nun dauerhaft auf einer Diskette im Laufwerk 0 abgelegt. Der Dateiname darf aus maximal 16 Zeichen bestehen, und die zweiten Anführungszeichen können auch weggelassen werden.

```
DSAVE "TEST567890123456
```

funktioniert also auch.

*Achtung:* Wenn man einen Dateinamen wählt, der auf der Diskette bereits existiert, meldet sich der Commodore auch - nach allerdings ziemlich kurzer Zeit - mit READY. Er hat jedoch nichts getan, unser Programm ist nicht gespeichert worden.

Manchmal kann es jedoch Absicht sein, daß wir einen Programmnamen wählen, den es auf der Diskette schon gibt, nämlich dann, wenn wir dieses Pro-

gramm überarbeitet haben und unter demselben Namen wieder speichern wollen. Dann müssen wir das dem System folgendermaßen anzeigen:

```
DSAVE "@TEST"
```

und das Programm wird unter dem Namen TEST abgelegt.

### Anmerkungen

In BASIC 2.0 heißt der Schreibbefehl für die Diskette SAVE (siehe dort).

Die Befehle für das Laden von Programmen heißen DLOAD bzw. LOAD (siehe dort).

# DVERIFY (BASIC 7.0)

**Abkürzung:** D<SHIFT>V

**Typ:** Systembefehl für Ausgabe

**Syntax:**
```
DVERIFY"Dateiname",DLaufwerknummer,
UGerätenummer

DVERIFY"Dateiname"
```

**Parameterbereich:** Dateinamen
Laufwerknummern
Gerätenummern

Der DVERIFY-Befehl vergleicht das Programm, das sich aktuell im Rechnerspeicher befindet, mit der Kopie auf der Diskette. Wenn keine Abweichung festgestellt wird, erscheint die Meldung

```
OK
```

andernfalls sieht man auf dem Bildschirm

```
?VERIFY ERROR
```

Wenn die Laufwerknummer, die D0 oder D1 sein kann, weggelassen wird, sucht der Befehl die Datei standardmäßig auf der Diskette im Laufwerk D0. Die Gerätenummer, die Werte von U4 bis U15 annehmen kann, ist ohne Angabe mit U8 initialisiert.

### Beispiel

```
100 REM TEST <RETURN>

DSAVE"TEST" <RETURN>

200 REM ZUSATZZEILE <RETURN>
```

```
DVERIFY"TEST" <RETURN>

?VERIFY ERROR
```

So können Sie natürlich leicht einen VERIFY-Fehler erzeugen.

# END (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** keine möglich

**Typ:** Befehl zur Programm-Ablaufsteuerung

**Syntax:** `END`

Die END-Anweisung beendet die Ausführung eines Programms, schließt alle Dateien ab und übergibt die Kontrolle an die Befehlsebene des Systems.

Der Rechner beendet einen Programmlauf auch dann, wenn er die letzte Programmzeile ausgeführt hat und keinen Befehl mehr findet. Wird ein Programm jedoch modular organisiert, so ist es notwendig, beispielsweise das Hauptprogramm von den Unterprogrammen durch END oder STOP zu trennen.

Für die Fehlersuche ist es wichtig zu wissen, daß durch die END-Anweisung weder Variableninhalte, noch Feldzeiger oder das Programm selbst gelöscht werden.

### Anmerkungen

Der STOP-Befehl unterbricht ebenfalls einen Programmlauf und führt dann zu einer BREAK-Meldung auf dem Bildschirm, die Ihnen genau mitteilt, an welcher Stelle das Programm endete (siehe auch STOP).

Wurde das Programm durch einen END- oder STOP-Befehl abgebrochen, kann es mit Hilfe des CONT-Befehls hinter der Abbruchstelle wieder fortgesetzt werden (siehe auch CONT).

# (BASIC 7.0) ENVELOPE

**Abkürzung:** E<SHIFT>N

**Typ:** Sound-Befehl

**Syntax:**
```
ENVELOPE Hüllkurvennummer,
Anschwellzeit,Abschwellzeit,
Dauer,Ausklingzeit,
Wellenform,Impulsbreite
```

**Parameterbereich:** ganze Zahlen in den unten angegebenen Bereichen

Mit ENVELOPE werden bei der Tonerzeugung die Hüllkurven definiert. 10 verschiedene Instrumente können beim C128 durch Angabe einer Hüllkurvennummer (0-9) ausgewählt werden. Die anderen Parameter variieren dann den Ton in der folgenden Bandbreite:

| | |
|---|---|
| Hüllkurvennummer | Werte von 0 bis 9 |
| Anschwellzeit (attack) | Werte von 0 bis 15 |
| Abschwellzeit (decay) | Werte von 0 bis 15 |
| Dauer (sustain) | Werte von 0 bis 15 |
| Ausklingzeit (release) | Werte von 0 bis 15 |
| Wellenform (wave) | 0 - Dreieckschwingung<br>1 - Sägezahnschwingung<br>2 - Rechteckschwingung<br>3 - Rauschen<br>4 - Ringmodulation |
| Impulsbreite | Werte von 0 bis 4095<br>(nur für Wellenform 2 - Rechteck) |

# ER EL ERR$ (BASIC 3.5, 7.0)

**Abkürzung:** nur für ERR$ möglich: E<SHIFT>R

**Typ:** reservierte Variablen

**Syntax:** ER EL ERR$(X)

Wenn man mit dem Befehl TRAP eine Fehlerauffangroutine angesteuert hat, kann man dort in der Variablen ER die Nummer der fehlerhaften BASIC-Zeile abfragen (siehe auch TRAP). Die Fehlermeldung selbst ist in ERR$ als Text gespeichert.

### Beispiel

Das folgende kleine Programm gibt alle Fehlermeldungen aus, die zur Verfügung stehen:

```
100 FOR I=1 TO 41
110 PRINT ERR$(I)
120 NEXT I
```

Sie können nach dem Abfangen eines Fehlers mit TRAP und dem Abfragen der Fehler-Codes in Ihren Programme deutsche Fehlermeldungen ausgeben, die Sie in einem Datenfeld abgelegt haben:

```
10000 DIM FE$(41)
10010 FE$(1)="LAUFWERK NICHT BEREIT, BITTE EINSCHALTEN
UND TASTE DRUECKEN"
10020 TRAP 40000
10030 DIRECTORY:STOP
40000 IF ER=5 THEN PRINT FE$(1)
40010 GETKEY T$
40020 RESUME 10030
40030 END
```

Am zweckmäßigsten verwendet man für eine Fehler-Fallunterscheidung den Befehl ON...GOSUB.

## (BASIC 2.0, 4.0, 3.5, 7.0) EXP

**Abkürzung:** E<SHIFT>X

**Typ:** numerische Funktion

**Syntax:** EXP(numerische Konstante oder Variable oder Ausdruck)

**Argumentbereich:** Gleitkommazahlen -1.7E38 – 88.0296

Die EXP-Funktion liefert den natürlichen Exponenten einer Zahl, d. h. die Potenz von e, wobei e den Wert 2.71828183 hat. Der Ausdruck

EXP(V)

bedeutet e hoch V.

Das Argument muß kleiner oder gleich 88.0296 sein. Andernfalls wird "Overflow" angezeigt.

### Beispiel

```
100 REM****** EXPONENTIALFUNKTION ******
110 FOR I=-9 TO 9
120 : PRINT"EXP(";I;")=";EXP(I)
130 NEXT I
140 END
```

# FAST (BASIC 7.0)

**Abkürzung:** F<SHIFT>A

**Typ:** Programmierhilfe

**Syntax:** FAST

Dieser Befehl hat nur im 80-Zeichen-Modus eine Wirkung. Dort bewirkt er eine doppelt so schnelle Text- oder Grafikausgabe. Im 40-Zeichen-Modus wird der Bildschirm abgeschaltet, und die Kontrolle kann nur mit SLOW oder Drücken der RUN/STOP- und RESTORE-Taste zurückgewonnen werden.

### Beispiel

```
100 FAST
110 FOR I=1 TO 200
120 NEXT I
130 SLOW
140 END
```

# (BASIC 7.0) FETCH

**Abkürzung:** F<SHIFT>E

**Typ:** Speicherverwaltungsbefehl

**Syntax:** FETCH Anzahl,Adresse1,Adresse2,Bank

**Parameterbereich:** Speicheradressen
Banknummern

Mit FETCH kann ein ganzer Speicherbereich aus der Speicherbank mit der angegebenen Nummer in den Arbeitsspeicher geladen werden. Der Bereich wird an die durch Adresse1 gekennzeichnete Stelle abgelegt. Adresse2 gibt an, ab welcher Speicheradresse die angegebene Anzahl Bytes geholt werden sollen.

Die Adresse1 bezieht sich auf die Bank, die mit dem BANK-Befehl aktiviert wurde. Standardmäßig ist das die Bank 15.

Die Umkehrung von FETCH ist der STASH-Befehl.

**Beispiel**

```
FETCH 999,1023,1000,1
```

Es sollen 999 Bytes aus der Bank 1 ab Adresse 1000 geholt werden und im Arbeitsspeicher ab Adresse 1023, also dem Bildschirmspeicher, abgelegt werden.

Der FETCH-Befehl zeigt nur Wirkung, wenn an den Rechner eine RAM-Disk angeschlossen ist.

# FILTER (BASIC 7.0)

**Abkürzung:** F<SHIFT>I

**Typ:** Sound-Befehl

**Syntax:**
```
FILTER Frequenzgrenze,Tiefpaßfilter
ein/aus,Bandpaßfilter ein/aus,
Hochpaßfilter ein/aus,Resonanz
```

**Parameterbereich:** Frequenzgrenze ganze Zahlen 0 - 2047
Resonanz ganze Zahlen 0 - 15
die anderen Parameter sind Bitzahlen

Mit der Anweisung FILTER können Sie die Klangfilter für die Töne setzen. Die Parameter bedeuten dabei im einzelnen:

| | |
|---|---|
| Frequenzgrenze | ab hier ist der Filter wirksam |
| Tiefpaßfilter | 0 - ausgeschaltet |
| | 1 - eingeschaltet |
| Bandpaßfilter | 0 - ausgeschaltet |
| | 1 - eingeschaltet |
| Hochpaßfilter | 0 - ausgeschaltet |
| | 1 - eingeschaltet |
| Resonanz | Werte von 0 bis 15 |

### Beispiel

In unserem Beispiel wollen wir eine Melodie immer abwechselnd mit einem der Filter und dann wieder ohne spielen.

```
100 ENVELOPE 7,10,9,9,5,1,3000:PLAY"T7"
110 TEMPO 10
120 FILTER 1024,1,0,0,8
130 PLAY"U9 V2 O5 X1
    HGFIGGGGHDQGGAAI$B$B$B$BHAR"
140 PLAY"U9 V1 O5 X0
    HGFIGGGGHDQGGAAI$B$B$B$BHAR"
150 FILTER 1024,0,1,0,8
```

```
160 PLAY"U9 V2 O5 X1
    HGFIGGGGHDQGGAAI$B$B$B$BHAR"
170 PLAY"U9 V1 O5 X0
    HGFIGGGGHDQGGAAI$B$B$B$BHAR"
180 FILTER 1024,0,0,1,8
190 PLAY"U9 V2 O5 X1
    HGFIGGGGHDQGGAAI$B$B$B$BHAR"
200 PLAY"U9 V1 O5 X0
    HGFIGGGGHDQGGAAI$B$B$B$BHAR"
```

# FOR...TO...STEP.../NEXT

(BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** F<SHIFT>O T<SHIFT>O ST<SHIFT>E N<SHIFT>E

**Typ:** Befehlswort zur Programmstrukturierung

**Syntax:**

```
FOR Variable=Anfang TO Ende
STEP Schrittweite/NEXT Variable

FOR Variable=Anfang TO Ende/
NEXT Variable

FOR Variable=Anfang TO Ende/NEXT
```

Die FOR-Anweisung dient dazu, Schleifen zu programmieren. Dies bedeutet, daß man die Ausführung einer bestimmten Folge von Programmzeilen wiederholen kann. Die Anzahl der Wiederholungen wird über eine Laufvariable kontrolliert, von der man angibt, welche Werte sie zu durchlaufen hat, z. B.:

```
10 FOR I=1 TO 10
```

Wenn STEP weggelassen wird, ist die Schrittweite automatisch 1. Im obigen Beispiel soll die Variable I also die ganzen Zahlen von 1 bis 10 durchlaufen.

Das Schlüsselwort TO zeigt an, bis wohin die Laufvariable Werte durchlaufen soll, und muß unbedingt angegeben werden.

Ebenso notwendig ist das Wort NEXT, das das Ende der FOR-Schleife angibt. Jede beliebige BASIC-Anweisung kann als Zeile innerhalb einer FOR-Schleife erscheinen. Oft machen die Zeilen innerhalb der Schleifen von der Laufvariablen Gebrauch, z. B.:

```
10 FOR I=1 TO 10
20 : PRINT I
30 NEXT I
```

Die Werte für Anfang, Ende und Schrittweite können negative oder positive Konstanten, Variablen oder Ausdrücke sein, z. B.:

```
10 J=5
20 FOR I=J+5 TO 1 STEP -1
30 : PRINT I
40 NEXT
```

Die Schleife wird ein letztes Mal durchlaufen, wenn der Endwert erreicht ist, und erst wenn der Wert der Variablen größer (bei positiver Schrittweite) oder kleiner (bei negativer Schrittweite) als der Endwert ist, wird die FOR-Schleife beendet, d. h. die Anweisung nach der NEXT-Anweisung wird ausgeführt.

Die FOR-Schleife wird immer mindestens einmal durchlaufen, auch wenn die Laufvariable von vornherein schon größer als der Endwert ist:

```
100 FOR I=1 TO -1
110 : PRINT I
120 NEXT I
```

Es ist auch zulässig, daß FOR-Schleifen innerhalb von anderen FOR-Schleifen erscheinen. Diese werden dann verschachtelte Schleifen genannt. Dabei muß jede Schleife ihre eigene Laufvariable haben und die zuletzt begonnene muß zuerst durch ein NEXT abgeschlossen werden.

Außerdem ist eine Verschachtelungstiefe von maximal 10 Schleifen zulässig.

```
10 DIM A(10,20)
20 FOR I=1 TO 10
30 : FOR J=1 TO 20
40 :   LET A(I,J)=1
50 : NEXT J
60 NEXT I
```

In diesem Beispiel wird die 10x20-Matrix elementeweise auf 1 gesetzt.

Wenn verschachtelte Schleifen an derselben Stelle im Programm beendet werden sollen, kann das in einer einzigen NEXT-Anweisung geschehen. Es muß allerdings auf die Reihenfolge geachtet werden. Im vorigen Beispiel müßte es dann heißen:

```
50 NEXT J,I
```

Wird eine FOR-Schleife als direkt auszuführender Befehl verwendet, so ist eine Mehrfach-Anweisungszeile notwendig, z. B.:

```
FOR J=1 TO 5:PRINT CHR$(J):NEXT
```

### Anmerkung

Weitere Möglichkeiten, solche Programmblöcke zu bilden, bieten die BASIC-Versionen 3.5 und 7.0 mit den Befehlen DO...LOOP...EXIT, DO...LOOP... UNTIL und DO...LOOP...WHILE (siehe dort).

### Beispiel

Wir wollen in unserem Beispielprogramm einen einfachen Benchmark durchführen. Darunter versteht man ganz allgemein ein Verfahren, die Leistungsfähigkeit von Computern zu messen, um diese miteinander vergleichen zu können. Um einen einfachen und direkten Vergleich zwischen den Mikrocomputern unterschiedlicher Hersteller machen zu können, werden vielfach simple BASIC-Programme vorgeschlagen, die einfache Operationen in einer großen Zahl von Wiederholungen durchführen und deren Laufzeit mittels einer Stoppuhr gemessen wird. Die Anweisung zur Durchführung der Messung entnehmen Sie bitte den REM-Zeilen des Programms selbst.

```
100 REM DURCHFUEHRUNG DES BENCHMARKS
110 REM ---------------------------
120 REM 1. RUN EINTIPPEN
130 REM 2. GLEICHZEITIG RETURN UND DIE
140 REM    STOPPUHR DRUECKEN
150 REM 3. BEIM ERTOENEN DES SIGNALS
160 REM    DIE STOPPUHR ANHALTEN
170 REM
180 PRINT CHR$(147)
190 DATA 1,2,3,4,5,6,7,8,9,0
200 DIM X(10)
210 FOR I=1 TO 100
220 : FOR J=1 TO 10
230 :   READ X(J)
240 : NEXT J
250 : RESTORE :REM VON VORNE LESEN
260 NEXT I
270 PRINT CHR$(7);"ENDE DES BENCHMARKS"
280 END
```

## (BASIC 2.0, 4.0, 3.5, 7.0) FRE

**Abkürzung:** F<SHIFT>R

**Typ:** Systemfunktion

**Syntax:** FRE(Variable)

**Argumentbereich:** Gleitkommazahlen 0 - 1

**Ergebnisbereich:** Speicheradressen

Die FRE-Funktion zeigt die Anzahl von Speicherbytes an, die noch für ein BASIC-Programm zur Verfügung steht. Der Wert des Arguments von FRE ist unwesentlich. Er hat keinen Einfluß auf das Ergebnis. Man kann ihn deshalb als Dummy-Argument bezeichnen.

Nur in BASIC 7.0 kommt der Variablen insofern eine Bedeutung zu, als daß FRE(0) den freien Speicherplatz in der Speicherbank 0 angibt. 1, 2 und 3 gelten entsprechend für die anderen Bänke, falls Sie Ihren Rechner erweitert haben. FRE(1) gibt also an, wieviel Speicherplatz noch im Variablenspeicherbereich der Bank 1 frei ist.

FRE löst außerdem jedesmal eine Garbage Collection, d. h. eine Bereinigung des Variablenspeichers zum Wiederverfügbarmachen von freigewordenem Speicherplatz aus.

### Anmerkungen

Der C64 antwortet nach dem Einschalten auf die Eingabe PRINT FRE(0) mit einer negativen Zahl, -26627. Um die richtige Zahl herauszubekommen, müssen Sie

```
PRINT FRE(0) + 2↑16
```

eintippen. Diesmal ergibt sich die Zahl 38909. Diese Zahl ist aber zu groß, um in 16 Bits dargestellt zu werden. Die größte so darstellbare Zahl ist nämlich 2↑15, also 32768. Übersteigt aber der noch verfügbare Speicherplatz diese Zahl, so wird noch das Vorzeichenbit mit herangezogen, und es ergeben sich negative Zahlen.

# GET (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** G<SHIFT>E

**Typ:** Eingabe-Anweisung

**Syntax:** GET Variable

**Parameterbereich:** Variablennamen

Die GET-Anweisung untersucht übernimmt aus dem Tastaturpuffer das erste in ihm enthaltene Zeichen und weist es der angegebenen Variablen zu. Sie kann beispielsweise in der folgenden Form benutzt werden:

```
10 GET Z$
```

Haben Sie in der Zeit, in der die Tastatur abgefragt wird, keine Taste gedrückt, weist die GET-Anweisung der Variablen Z$ in unserem Fall eine leere Zeichenkette zu. Das Drücken der RETURN-Taste nach Eintippen eines Zeichens ist dabei nicht notwendig.

Um aber sicherzustellen, daß der Computer so lange wartet, bis Sie eine Taste gedrückt haben, wird die GET-Anweisung gewöhnlich in eine Schleife eingebettet:

```
10 GET Z$:IF Z$="" THEN GOTO 10
```

Diese Schleife wiederholt sich, bis eine Taste gedrückt wird und ihr Zeichenwert der Variablen Z$ zugewiesen ist.

Der Typ der gedrückten Taste muß immer mit dem Variablentyp übereinstimmen, der hinter GET angegeben wird. Wenn eine Zahl erwartet wird, muß auch eine Zifferntaste gedrückt werden, sonst gibt es einen TYPE MISMATCH ERROR:

```
10 GET A:IF A=0 THEN GOTO 10
20 PRINT A
```

GET gibt nicht automatisch den Wert, den es erhalten hat, auf dem Bildschirm aus. Es bleibt dem Programmierer überlassen, ob und wo ein Eingabezeichen

ausgegeben werden soll. Das ist einer der wichtigsten Unterschiede zwischen GET und INPUT (siehe dort). INPUT gibt jedes auf der Tastatur eingetippte Zeichen unmittelbar auf dem Bildschirm aus.

### Anmerkungen

In BASIC 7.0 gibt es die wesentlich komfortablere Anweisung GETKEY.

GET darf nicht im Direktmodus verwendet werden.

# GET# (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** G<SHIFT>E#

**Typ:** Eingabe-Anweisung

**Syntax:** GET#Dateinummer,Variablenliste

**Parameterbereich:** Dateinummern
Variablennamen

Die GET#-Anweisung arbeitet genauso wie die GET-Anweisung, nur liest sie einzelne Zeichen von einem zuvor mit OPEN geöffneten Eingabegerät, wie z. B. Kassettenrecorder oder Diskettenlaufwerk.

GET# kann nur im Programmodus und nicht im Direktmodus angewendet werden, da sonst ein

```
?ILLEGAL DIRECT ERROR
```

auftritt.

## Beispiel

In unserem Beispielprogramm wollen wir Daten von der Diskette lesen, und zwar das Inhaltsverzeichnis. In BASIC 2.0 wird das aktuell im Speicher befindliche Programm beim Laden des Inhaltsverzeichnisses überschrieben. Das passiert nicht, wenn Sie das folgende Programm mit der GET#-Routine anwenden. Hängen Sie es als Unterprogramm an jedes Ihrer Programm, und schon können Sie in BASIC 2.0 genauso komfortabel arbeiten wie in den anderen BASIC-Versionen mit dem DIRECTORY-Befehl.

```
60000 PRINT CHR$(147)
60010 OPEN 1,8,0,"$"
60020 GET#1,D$,D$
60030 FOR I=1 TO 4
60040 : GET#1,D$
60050 NEXT I
60060 IF ST THEN CLOSE 1:END
```

```
60070 GET#1,Z$
60080 IF Z$="" THEN PRINT:GOTO 60030
60090 IF Z$=CHR$(34) THEN S=NOT S
60100 IF S THEN PRINT Z$;
60110 GOTO 60070
```

Wenn Sie also während des Testens eines Programms das Inhaltsverzeichnis der Diskette einsehen wollen, brauchen Sie nur RUN 60000 oder GOTO 60000 einzutippen. Ihr Programm im Arbeitsspeicher geht dabei nicht verloren.

### Anmerkung

GET# darf nicht im Direktmodus verwendet werden.

# GETKEY (BASIC 3.5, 7.0)

**Abkürzung:** GETK<SHIFT>E

**Typ:** Eingabe-Anweisung

**Syntax:** GETKEY Variable

**Parameterbereich:** Variablennamen

Die Anweisung GETKEY ordnet das Zeichen einer beliebigen auf der Tastatur gedrückten Taste einer Variablen zu. Dabei muß der Typ der gedrückten Taste mit dem Variablentyp hinter GETKEY übereinstimmen.

In einem Programm kann so z. B. die weitere Ausführung von dem Drücken einer bestimmten Taste abhängig gemacht werden:

```
100 PRINT"DRUECKEN SIE A, UM VON VORN ZU BEGINNEN"
100 GETKEY Z$:IF Z$="A" GOTO 10
```

Die Anweisung GETKEY arbeitet ähnlich wie GET (siehe dort). Der einzige Unterschied ist, daß bei GETKEY so lange auf den Tastendruck gewartet wird, bis er erfolgt.

### Beispiel

Sämtliche Beispiele in diesem Buch, die mit dem hochauflösenden Grafikbildschirm arbeiten, enthalten die GETKEY-Anweisung. Normalerweise würde nämlich jede Grafik sofort nach ihrer Erstellung wieder verschwinden. Wenn sich jedoch die GETKEY-Anweisung anschließt, bleibt sie so lange zu sehen, bis der Zuschauer eine beliebige Taste drückt. Dann schalten wir in den Textmodus zurück und beenden das Programm. So haben wir es auch bei dem folgenden Beispiel gemacht:

```
100 GRAPHIC 1,1
110 FOR V=4 TO 24 STEP 4
120 : CIRCLE 1,160-V,100,V,V-3
130 NEXT V
140 FOR V=24 TO 48 STEP 4
```

```
150 : CIRCLE 1,112+V,100,V,V-3
160 NEXT V
170 FOR V=48 TO 72 STEP 4
180 : CIRCLE 1,208-V,100,V,V-3
190 NEXT V
200 FOR V=72 TO 100 STEP 4
210 : CIRCLE 1,64+V,100,V,V-3
220 NEXT V
230 GETKEY T$
240 GRAPHIC 0,1
250 END
```

Die Anweisung GETKEY in Zeile 230 ermöglicht uns das beliebig lange Anschauen der erstellten Grafik.

### Anmerkungen

GETKEY kann nicht im Direktmodus verwendet werden.

Die Anweisung GETKEY ist zwar komfortabel, aber nicht unersetzbar. Ein bißchen umständlicher läßt sich Zeile 230 in BASIC 2.0 und 4.0 so ausdrücken:

```
230 GET A$:IF A$="" THEN 230
```

oder ganz einfach:

```
230 GOTO 230
```

Aus dieser Endlosschleife läßt sich das Programm allerdings nur noch durch Drücken von RUN/STOP und RESTORE herausholen.

# GO 64 (BASIC 7.0)

**Abkürzung:** keine möglich

**Typ:** Systembefehl

**Syntax:** `GO 64`

Dieser Befehl teilt dem Betriebssystem des C128 mit, daß man in den 64er-Modus überwechseln möchte. Nach der Bestätigung erscheint der Anfangsbildschirm des C64 mit seinen 40 Zeichen pro Zeile, und nun sind alle Programme des Commodore lauffähig.

Für den umgekehrten Weg, das Zurückschalten in den 128er-Modus gibt es selbstverständlich kein Kommando, da ein solches ja im C64 auch nicht vorhanden ist und 100%ige Kompatibilität gewahrt werden sollte. In diesem Fall muß man das Gerät entweder aus- und wieder einschalten oder einen Reset auslösen.

### Anmerkung

Auch für das Umschalten vom 128- in den 64-Modus kann man anstelle des Befehls GO 64 den RESET-Knopf einsetzen. Dabei muß jedoch gleichzeitig die Commodore-Taste gedrückt werden.

# (BASIC 2.0, 4.0, 3.5, 7.0) GOSUB RETURN

**Abkürzung:** GO<SHIFT>S RE<SHIFT>T

**Typ:** Befehlswort zur Programm-Ablaufsteuerung

**Syntax:** GOSUB Zeilennummer/RETURN

**Parameterbereich:** Zeilennummern

Der GOSUB-Befehl erzeugt einen Sprung in ein Unterprogramm, das sich ab der angegebenen Zeilennummer befindet. Hier werden dann die Anweisungen der Unterroutine ausgeführt, bis der Rechner auf ein RETURN trifft. Das ist auch der bedeutendste Unterschied zum GOTO-Befehl (siehe dort). Dann wird die Kontrolle des Programmablaufs an die Anweisung zurückgegeben, die unmittelbar auf den GOSUB-Befehl folgt.

GOSUB kann entweder als direkt auszuführender Befehl oder als Programmanweisung benutzt werden:

```
GOSUB 1000
```

wobei 1000 die erste Zeile eines Unterprogramms ist.

Unterprogramme können auf diese Weise auch geschachtelt werden. Bei jedem GOSUB werden Vermerke in einem Stapelspeicher (in der Fachsprache Stack) gemacht, die hintereinander abgelegt werden. Jedes RETURN vermindert die Stapeltiefe wieder um 1.

Es ist möglich, daß mehrere RETURN-Anweisungen vorhanden sind, um nach einem GOSUB-Befehl zurückzuverzweigen. Wenn das Programm auf ein RETURN trifft, ohne daß zuvor GOSUB ausgeführt wurde, dann wird die Fehlermeldung RETURN WITHOUT GOSUB erscheinen.

Gelegentlich kann es vorkommen, daß Sie zwischen verschiedenen Unterroutinen auswählen möchten, und zwar in Abhängigkeit von dem Wert einer Variablen. In solchen Situationen bietet sich die ON...GOSUB-Anweisung an:

```
ON S GOSUB 1000,2000,3000,4000
```

Dabei wird die Kontrolle des Programms an eine von vier Unterroutinen geschickt, wenn S die Werte 1 bis 4 annimmt (siehe auch ON...GOSUB).

Ein Unterprogramm kann sich selbst wieder aufrufen, dann spricht man von einem rekursiven Aufruf. Jedoch kann das nicht beliebig oft geschehen. Die Gründe dafür wollen wir kurz erläutern. Jedesmal, wenn das Programm mit GOSUB eine Verzweigung zu der hinter GOSUB angegebenen Zeilennummer durchführt, wird gleichzeitig der Programmzeiger, der auf die Anweisung nach dem GOSUB zeigt, auf den Speicherstapel (Stack) gelegt, damit das Programm, nachdem es auf RETURN gestoßen ist, an dieser Stelle weitermachen kann. Wenn nun innerhalb des Unterprogramms wieder ein Unterprogrammaufruf steht, wird auch dafür eine Adresse auf den Stapel gelegt usw.

Das letzte Element, das auf den Stapel gelegt wurde, muß als erstes wieder heruntergenommen werden. Diese Technik nennt man LIFO - Last In, First Out. Im Commodore-BASIC sind maximal 25 Stapelebenen vorgesehen. Wenn Sie also mehr als 25 Unterprogrammaufrufe ineinander verschachteln, dann läuft dieser Stapel über.

### Beispiel

In unserem Beispiel wird in der Zeile 10080 das Unterprogramm aufgerufen, das ab der Zeile 50000 beginnt. Bei diesem Unterprogramm handelt es sich um die Beispielroutine, die Sie auch unter dem Stichwort INT finden.

```
10000 PRINT CHR$(147)
10010 PRINT"  FREITAGE, DIE AUF EINEM 13. LIEGEN"
10020 PRINT"  ZEITRAUM:  1.1.1984 BIS 31.12.2000"
10030 PRINT"  ---------------------------------"
10040 Z=0:TA=2:T=13
10050 FOR I=1984 TO 2000
10060 : FOR K=1 TO 12
10070 :   M=K:J=I
10080 :   GOSUB 50000
10090 :   IF W<>6 GOTO 10150
10100 :   Z=Z+1
10110 :   PRINTTAB(TA)"13.";:PRINT USING"##";K;:PRINT".";
10120 :   PRINT USING"####";I;
10130 :   IF TA>16 THEN TA=2:PRINT:GOTO 10150
10140 :   TA=TA+12
10150 : NEXT K
10160 NEXT I
10170 PRINT
10180 PRINTTAB(11)"ANZAHL = ";Z
10190 END
50000 REM ******************************
50010 REM T=TAG
50020 REM M=MONAT
50030 REM J=JAHR
```

```
50040 IF M=1 OR M=2 THEN J=J-1:M=M+12
50050 H=T+2*M+INT((3*M+3)/5)+J+INT(J/4)-INT(J/100)
50060 H=H+INT(J/400)+1
50070 W=INT(((H/7-INT(H/7))*700+.5)/100)+1
50080 RETURN
```

## Anmerkungen

Durch den Systembefehl RENUMBER in den BASIC-Versionen 3.5 und 7.0 werden alle Zeilennummern nach einem GOSUB-Befehl folgerichtig neu vergeben.

# GOTO (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** G<SHIFT>O

**Typ:** Befehlswort zur Programm-Ablaufsteuerung

**Syntax:** GOTO Zeilennummer

**Parameterbereich:** Zeilennummern

Der Befehl GOTO führt einen unbedingten Sprung auf die angegebene Zeilennummer aus, z. B.:

```
1000 GOTO 2000
```

Enthält die angesprungene Zeile keine ausführbare Anweisung, z. B. REM oder DATA, so wird die nächste ausführbare Anweisung nach dieser Zeile ausgeführt.

GOTO kann entweder als sofort auszuführender Befehl oder als Programmanweisung verwendet werden. Im ersten Fall bewirkt es einen Programmstart ab der angegebenen Zeile (wie auch RUN):

```
GOTO 200
```

Die Programmausführung beginnt in Zeile 200.

### Anmerkungen

Durch den Systembefehl RENUMBER in den BASIC-Versionen 3.5 und 7.0 werden alle Zeilennummern nach einem GOTO folgerichtig vergeben.

Fehlt die Zeilennummer, die in einem GOTO angegeben ist, in dem Programm, so wird beim Durchlaufen eines derartigen Befehls das Programm mit einer Fehlermeldung abgebrochen.

Für mehrfaches Verzweigen siehe auch unter dem Stichwort ON...GOTO.

# (BASIC 3.5, 7.0) GRAPHIC

**Abkürzung:** G<SHIFT>R

**Typ:** Grafikbefehl

**Syntax:**
```
GRAPHIC Modus,Löschen,Zeile
GRAPHIC Modus
```

**Parameterbereich:** Modus - ganze Zahlen 0 - 5
Löschen - Bitzahl
Zeile - Zeilennummern

Mit der GRAPHIC-Anweisung kann der Grafikmodus eingestellt werden. Außerdem kann gleichzeitig mit dem zweiten Parameter festgelegt werden, ob der Bildschirm gelöscht werden soll. Ist er 1, dann wird gelöscht. Wenn er 0 ist oder weggelassen wird, wird kein SCNCLR durchgeführt.

Über den dritten Parameter kann man eine Zeilennummer festlegen, in welcher der Textbildschirm beginnen soll (nur bei Modus 2 oder 4). Das ist aber nur in BASIC 7.0 möglich. In BASIC 3.5 liegt der Textteil bei den gemischten Text/Grafikmodi immer in den unteren 5 Zeilen.

Die Grafikmodi sind im einzelnen:

0 40-Zeichen-Textmodus
1 hochauflösende Grafik
2 Grafik/Text gemischt
3 Mehrfarbenmodus
4 Grafik/Text gemischt im Mehrfarbenmodus
5 80-Zeichen-Textmodus (nur BASIC 7.0)

**Beispiel**

```
100 J=320
110 GRAPHIC 1,1
120 FOR I=10 TO 100 STEP 10
130 : J=J-10
140 : BOX 1,I,I,J,J-120
```

```
150 NEXT I
160 GETKEY T$
170 GRAPHIC 0,1
180 END
```

In der Zeile 110 wird der hochauflösende Grafikmodus eingeschaltet. Das ist absolut notwendig, wenn man einen der Grafikbefehle BOX, CIRCLE, DRAW usw. erfolgreich einsetzen will. Genauso wichtig ist es, vor Beenden des Programms den Textmodus wieder einzuschalten (Zeile 170).

## (BASIC 4.0, 3.5, 7.0) HEADER

**Abkürzung:** HE<SHIFT>A

**Typ:** Systembefehl für Diskettenlaufwerk

**Syntax:**
```
HEADER "Diskettenname"

HEADER "Diskettenname",
IIdentifikationsnummer,
DLaufwerknummer,UGerätenummer
```

**Parameterbereich:** Diskettennamen
Identifikationsnummer - ganze Zahlen 0 - 99
Laufwerknummern
Gerätenummern

Jede fabrikneue Diskette muß vor Gebrauch zunächst formatiert werden. Dazu dient der HEADER-Befehl:

```
HEADER "DISKETTE 1",I01,D1,U8
```

Die Diskette erhält einen Namen, der maximal aus 16 Zeichen bestehen darf, und eine Identifikationsnummer. Diese Nummer kann entfallen, wenn die Diskette schon einmal formatiert wurde und nur für einen neuen Gebrauch leer gemacht werden soll.

Ebenso kann die Laufwerknummer entfallen, wenn nur ein Diskettenlaufwerk angeschlossen ist oder das Laufwerk mit der Nummer D0 angesprochen werden soll.

Da nach dem Formatieren alles, was sich zuvor auf der Diskette befunden hat, rettungslos verloren ist, werden Sie zur Sicherheit vom System gefragt:

```
ARE YOU SURE?
SIND SIE SICHER?
```

Erst wenn Sie diese Frage mit Y beantworten, beginnt das Formatieren.

### Anmerkung

In BASIC 2.0 wird eine neue Diskette mit einem speziellen OPEN-Befehl formatiert (siehe dort).

## (BASIC 3.5, 7.0) HELP

**Abkürzung:** HE<SHIFT>L

**Typ:** Programmierhilfe

**Syntax:** HELP

Bricht ein Programm mit einer Fehlermeldung ab, so kann man sich durch Druck auf die HELP-Taste die fehlerhafte Zeile direkt ansehen.

Auch im Programm- oder im Direktmodus kann man den HELP-Befehl eingeben. Dann erscheint die Programmzeile, die den Fehler enthält, in reverser Darstellung auf dem Bildschirm. Wenn sich mehrere Anweisungen, durch Doppelpunkt getrennt, in der Programmzeile befinden, wird nur die Anweisung revers dargestellt, die den Fehler enthält. Der Rest der Zeile erscheint normal.

Zur Fehlerbehandlung ist dieser Befehl äußerst nützlich. Der Fehler kann, falls ersichtlich, gleich in der angezeigten Programmzeile korrigiert werden. RETURN drücken und RUN tippen, dann wird sich zeigen, ob es der einzige Fehler im Programm war und alles jetzt zur Zufriedenheit funktioniert.

### Anmerkungen

Beim C16 und C116 liegt der HELP-Befehl auf einer Funktionstaste und wird auf Tastendruck hin ausgeführt. Der Teil der BASIC-Zeile, ab der der Fehler lokalisiert wurde, beginnt zu blinken.

Beim C128 gibt es auf der Tastatur eine spezielle Taste, die mit HELP beschriftet ist.

Im 80-Zeichen-Modus beim C128 wird der fehlerhafte Teil unterstrichen.

# HEX$ (BASIC 3.5, 7.0)

**Abkürzung:** H<SHIFT>E

**Typ:** numerische Funktion

**Syntax:** HEX$(numerischer Ausdruck)

**Argumentbereich:** Speicheradressen

**Ergebnisbereich:** hexadezimale Zahlen 0000 - FFFF

Die Funktion HEX$ wandelt die in Klammern angegebene Dezimalzahl in eine Hexadezimalzahl um. Die Dezimalzahl darf dabei im Bereich von 0 bis 65535 liegen.

### Beispiel

```
PRINT HEX$(15)
```

ergibt die hexadezimale Zahl A.

Wir erstellen nun ein komfortables Programm, das dem Benutzer die Eingabe von dezimalen Zahlen erlaubt und die entsprechende Hexadezimalzahl dazu ausrechnet:

```
100 PRINT CHR$(147)
110 PRINTTAB(7)"HEXADEZIMALE WERTE"
120 PRINTTAB(7)"------------------":PRINT
130 PRINT"BITTE WERTE EINGEBEN, 0 FUER ENDE":PRINT
140 PRINTTAB(16)"DEZ";TAB(24)"HEX"
150 PRINTTAB(16)"---";TAB(24)"---"
160 WINDOW 0,9,39,24,1
170 INPUT D
180 IF D<0 OR D>65535 GOTO 170
190 IF D=0 THEN 220
200 PRINTTAB(15)D;TAB(24)HEX$(D)
210 GOTO 170
220 PRINT CHR$(19)+CHR$(19)+CHR$(147)
230 END
```

Die HEX$-Funktion wird in der Zeile 200 eingesetzt.

Das Fenster, das in der Zeile 160 definiert wird, verhindert, daß die Überschriften, die sich nun außerhalb des aktuellen Eingabefensters befinden, oben aus dem Bildschirm herausgeschoben werden, wenn wir unten auf dem Schirm angekommen sind. So verschiebt sich nur das Eingabefenster.

Bei Programmende muß das Fenster wieder gelöscht werden. Das geschieht in Zeile 220.

# IF...THEN (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** nur für THEN möglich: T<SHIFT>H

**Typ:** Befehlswort zur Programmstrukturierung

**Syntax:** IF Bedingung THEN Anweisung(en)

Die IF-Anweisung gibt uns die Möglichkeit, innerhalb eines Programms Entscheidungen zu treffen. Wenn die Bedingung nach dem IF wahr ist, führt der Rechner den auf THEN folgenden Befehl aus.

Dabei kann die Bedingung ein logischer Ausdruck, ein numerischer Ausdruck oder eine Variable sein. Logische Ausdrücke ergeben entweder den Wert -1 (wahr) oder 0 (falsch). Numerische Ausdrücke oder Variablen ergeben falsch, wenn ihr Wert 0 ist, und wahr bei jedem anderen Wert.

Wenn die Anweisung nach dem THEN ein Sprung zu einer Zeilennummer darstellt, kann das Wort THEN oder das Wort GOTO entfallen:

```
IF A$="JA" THEN 1000
```

oder:

```
IF A$="JA" GOTO 1000
```

In allen Fällen, in denen die Bedingung falsch ergibt, werden die Anweisungen nach dem THEN ignoriert, und die Verarbeitung fährt in der nächsten Zeile fort.

Logische Ausdrücke sind Gleichungen oder Ungleichungen, die den Wert wahr oder falsch annehmen und in denen z. B. die folgenden Symbole vorkommen:

| | |
|---|---|
| = | ist gleich |
| <> | ist nicht gleich (ungleich) |
| < | ist kleiner als |
| > | ist größer als |
| <= | ist kleiner gleich |
| >= | ist größer gleich |

Außerdem können die BASIC-Wörter AND, OR und NOT ebenfalls innerhalb eines logischen Ausdrucks verwendet werden (siehe dort), z. B.:

```
10 IF NOT(A=1 OR B=77) THEN 2000
```

### Beispiel

In unserem Beispielprogramm wollen wir den Euklidischen Algorithmus zur Bestimmung des größten gemeinsamen Teilers zweier natürlicher Zahlen programmieren. Der Algorithmus läßt sich folgendermaßen formulieren:

1. Dividiere die größere Zahl durch die kleinere.

2. Ist der Rest gleich Null, dann ist der Divisor der größte gemeinsame Teiler.

3. Ist der Rest nicht gleich Null, dann dividiere den alten Divisor durch den erhaltenen Rest.

4. Gehe zu Schritt 2 zurück.

Das Programm dazu sieht so aus:

```
100 PRINT CHR$(147)
110 PRINT"GEBEN SIE 2 POSITIVE GANZE ZAHLEN EIN"
120 PRINT"(0,0 FUER ENDE)"
130 PRINT:INPUT A,B
140 IF A=0 AND B=0 THEN END
150 IF A<1 OR B<1 THEN 110
160 A=INT(A):B=INT(B)
170 IF A>B THEN 200
180 IF A=B THEN 230
190 H=A:A=B:B=H
200 R=A-INT(A/B)*B
210 IF R=0 THEN 230
220 A=B:B=R:GOTO 200
230 PRINT"GROESSTER GEMEINSAMER TEILER =";B
240 GOTO 130
```

### Anmerkungen

In den BASIC-Versionen 3.5 und 7.0 ist die Kombination IF...THEN...ELSE vorgesehen, die durch das ELSE noch in der gleichen Zeile das Ausführen einer Anweisung für den Fall, daß die Bedingung falsch ist, möglich macht (siehe dort).

# IF...THEN...:ELSE (BASIC 3.5, 7.0)

**Abkürzung:** IF...T<SHIFT>H...E<SHIFT>L

**Typ:** Befehlswort zur Programmstrukturierung

**Syntax:**
```
IF Bedingung THEN Anweisung
:ELSE Anweisung
```

Diese Anweisung beinhaltet die Möglichkeiten der IF...THEN-Anweisung (siehe dort). Zusätzlich kann hier durch die ELSE-Anweisung noch in der gleichen Zeile angegeben werden, was geschehen soll, falls die Bedingung, die hinter IF steht, nicht erfüllt ist.

### Anmerkung

In BASIC 2.0 und 4.0 kann der ELSE-Zweig ganz leicht in einer ON-Anweisung simuliert werden. So läßt sich folgende Zeile aus dem BASIC 3.5 oder 7.0:

```
IF A=0 THEN GOTO 100 :ELSE GOTO 200
```

ohne ELSE folgendermaßen in einer Zeile schreiben:

```
ON -(A=0) THEN GOTO 100:GOTO 200
```

## (BASIC 2.0, 4.0, 3.5, 7.0) INPUT

**Abkürzung:** keine möglich

**Typ:** Eingabe-Anweisung

**Syntax:**

```
INPUT Variablenliste

INPUT "Zeichenkettenkonstante";
Variablenliste
```

**Parameterbereich:** Variablennamen

Die INPUT-Anweisung weist den Rechner an, auf Daten zu warten, die über die Tastatur eingegeben werden. Die Daten werden an die Variablen übergeben, die in der INPUT-Anweisung spezifiziert werden, und müssen mit deren Typ übereinstimmen.

```
10 INPUT A$
```

beispielsweise erwartet eine Zeichenkette als Eingabe.

Wenn eine INPUT-Anweisung ausgeführt wird, gibt der Rechner ein Fragezeichen auf dem Bildschirm aus, um anzuzeigen, daß er auf Eingabedaten wartet. Im Unterschied zur GET-Anweisung (siehe dort), erzeugt das eingetippte Zeichen unmittelbar ein "Echo" auf dem Bildschirm.

Die zweite Schreibweise für INPUT erlaubt, zunächst einen erläuternden Text auf dem Bildschirm auszugeben, bevor eine Eingabe gemacht werden soll, z. B.:

```
10 INPUT"GEBEN SIE DREI ZAHLEN EIN";Z1,Z2,Z3
```

Diese Anweisung bewirkt auf dem Bildschirm die Anzeige:

```
GEBEN SIE DREI ZAHLEN EIN?
```

und wartet dann auf die Eingabe dreier Zahlen über die Tastatur. Diese können entweder, durch Kommas voneinander getrennt, alle in einer Zeile getippt werden:

```
GEBEN SIE DREI ZAHLEN EIN?1,2,3
```

oder jede Zahl wird, gefolgt von einem RETURN, in einer gesonderten Zeile eingegeben. In diesem Fall wird der Rechner nach der Eingabe vor jeder folgenden Eingabe zwei Fragezeichen ausgeben:

```
GEBEN SIE DREI ZAHLEN EIN?1
??2
??3
```

### Anmerkungen

Falls bei der Eingabe in einer Zeile mehr Werte eingetippt werden, als das Programm erwartet, erscheint die Meldung:

```
?EXTRA IGNORED
```

Die zusätzlichen Werte werden also ignoriert, was keinen Fehler verursacht, und die Verarbeitung kann weitergehen.

Wenn eine Eingabe gemacht wird, bei der der Datentyp nicht mit der Variablen übereinstimmt, heißt es:

```
?REDO FROM START
```

und ein erneut erscheinendes Fragezeichen fordert den Benutzer zur korrekten Eingabe auf.

Wollen Sie einen Zeichenkettenwert eingeben, der ein Komma oder einen Doppelpunkt enthält, muß die gesamte Zeichenkette in Anführungszeichen eingeschlossen werden. Als Antwort auf die INPUT-Anweisung:

```
INPUT"WIE LAUTET IHR NAME";N$
```

können Sie z. B. eingeben:

```
?"SCHUSTER,FRANK"
```

In diesem Fall wird der Zeichenkettenvariablen N$ eine Zeichenkette mit einer Länge von 14 Zeichen zugewiesen:

```
SCHUSTER,FRANK
```

Sollten Sie vergessen, solch eine Zeichenkette in Anführungszeichen einzuschließen, wird der Rechner sie als zwei verschiedene Zeichenketten lesen, die

durch Komma getrennt sind. Da aber in diesem Fall die INPUT-Anweisung nur eine Variable enthält, der ein Wert zugewiesen werden soll, erscheint die Fehlermeldung "?EXTRA IGNORED".

Drücken Sie als Antwort auf eine INPUT-Anweisung die RETURN-Taste, ohne irgendwelche Daten einzugeben, wird die in der INPUT-Anweisung enthaltene Variable den Wert behalten, den sie besaß, bevor die Anweisung ausgeführt wurde. Das folgende kurze Programm zeigt diese Eigenart:

```
10 LET I=1000
20 LET Z$="COMPUTER"
30 INPUT I
40 INPUT Z$
50 PRINT I,Z$
```

Als Antwort auf die beiden INPUT-Eingaben drücken Sie nun die RETURN-Taste. Nachdem beide INPUT-Anweisungen ausgeführt worden sind, wird die folgende Zeile auf dem Bildschirm ausgedruckt:

```
1000          COMPUTER
```

Sie sehen, daß beide Variablen, I und Z$, noch dieselben Werte enthalten wie vor den INPUT-Anweisungen.

### Anmerkungen

Die INPUT-Anweisung akzeptiert nur Eingaben von maximal 80 Zeichen Länge, da der für die INPUT-Zeichenkette zur Verfügung stehende Eingabepuffer auf diese Länge beschränkt ist. Nur beim C128 dürfen es 160 Zeichen sein.

INPUT kann nicht im Direktmodus verwendet werden.

# INPUT# (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** I<SHIFT>N

**Typ:** Eingabe-Anweisung

**Syntax:** INPUT#Gerätenummer,Variablenliste

**Parameterbereich:** Gerätenummern
Variablen

Die INPUT#-Anweisung liest Datenangaben von externen Datendateien, die auf Kassette oder Diskette gespeichert sind und mit OPEN eröffnet worden sind, und weist diese Angaben den entsprechenden Variablen zu. Anzahl der Daten und Datentyp müssen mit der Variablenliste übereinstimmen, sonst wird der Zustand

```
?OUT OF DATA
```

angezeigt, das Programm fährt allerdings in seiner Ausführung fort.

Anders als die GET#-Anweisung, die externe Dateien Zeichen für Zeichen abgreift, liest INPUT# die gesamten Dateneingaben auf einmal. Dabei erkennt INPUT# das RETURN-Zeichen (CHR$(13)), Kommas, Doppelpunkte und Semikolons als Trennungszeichen zwischen den einzelnen Daten einer Datei. Wenn Sie eine Datei erzeugen (siehe PRINT#), müssen Sie darauf achten, daß eins dieser Zeichen zwischen den einzelnen Daten der Datei steht.

Da der Eingabepuffer nur beim C128 160 Zeichen, aber sonst 80 Zeichen faßt, darf eine Zeichenkette, einschließlich der Trennzeichen, nicht länger sein.

Das folgende Programm liest Daten vom Kassettenrecorder:

```
10 OPEN1,1
20 INPUT#1,A$,B$
```

## Beispiel

Wir wollen die sequentielle Datei, die mit dem Beispielprogramm unter dem Stichwort PRINT# erzeugt worden ist, wieder lesen und die Angaben in einer Tabelle ausgeben.

Die Datei ist auf Diskette gespeichert und enthält Informationen über mehrere Angestellte einer fiktiven Firma. Zu jedem Angestellten gibt es vier Datenangaben:

- in der Variablen K$ ein Kennzeichen, das verrät, ob der Arbeitnehmer Stundenlohn (S) erhält oder Gehalt (G);

- in der Variablen N$ den Nachnamen des Arbeitnehmers;

- in der Variablen V$ den Vornamen des Arbeitnehmers;

- in der Variablen G das Entgelt des Arbeitnehmers, also Stundenlohn beim Kennzeichen S oder Monatsgehalt beim Kennzeichen G.

Die allererste Datenangabe der Datei ist eine Zahl, die angibt, wie viele Personalakten in der Datei gespeichert sind.

```
100 OPEN 1,8,2,"ARBEITNEHMER,S,R"
110 INPUT#1,Z
120 DIM K$(Z),N$(Z),V$(Z),G(Z)
130 FOR I=1 TO Z
140 : INPUT#1,K$(I),N$(I),V$(I),G(I)
150 NEXT I
160 CLOSE 1
170 PRINT CHR$(147)
180 PRINT"    MITARBEITERLISTE"
190 PRINT:PRINT
200 PRINT"GEHALT:"
210 PRINT
220 FOR I=1 TO Z
230 : IF K$(I)="G" THEN PRINT N$(I)","LEFT$(V$(I),1)
      "."TAB(14)G(I)
240 NEXT I
250 PRINT:PRINT
260 PRINT"STUNDENLOHN:"
270 PRINT
280 FOR I=1 TO Z
290 : IF K$(I)="S" THEN PRINT N$(I)","LEFT$(V$(I),1)
      "."TAB(14)G(I)
300 NEXT I
310 END
```

Die sequentielle Diskettendatei mit dem Namen ARBEITNEHMER wird in Zeile 100 zum Lesen geöffnet. Beachten Sie, daß die Datei mit der Gerätenum-

mer 1 eröffnet wurde. Jede INPUT#-Anweisung im Programm muß sich deshalb auf die Nummer beziehen.

In der Zeile 110 wird die Anzahl Z der Personalakten in der Datei gelesen. Nun kann das Programm die vier Felder festlegen, die endgültig die Daten speichern werden. Nachdem alle Angaben eingelesen worden sind, können sie ihrem Kennzeichen entsprechend unter der Überschrift GEHALT: oder STUNDENLOHN: aufgelistet werden.

### Anmerkung

INPUT# kann nur als Programmanweisung benutzt werden.

## (BASIC 3.5, 7.0) INSTR

**Abkürzung:** IN<SHIFT>S

**Typ:** Zeichenkettenfunktion

**Syntax:** INSTR(Zeichenkette1,Zeichenkette2)

INSTR(Zeichenkette1,Zeichenkette2, Startnummer)

**Argumentbereich:** Zeichenkettenvariablen
ganze Zahlen 0 - 255

**Ergebnisbereich:** ganze Zahlen 0 - 255

Mit der INSTR-Funktion kann die Position, ab der sich die Zeichenkette2 in der Zeichenkette1 befindet, bestimmt werden. Die Startnummer kann dabei angeben, ab dem wievielten Zeichen die erste Zeichenkette durchsucht werden soll.

Falls sich die Zeichenkette2 nicht in der Zeichenkette1 befindet, wird der Wert 0 ermittelt.

### Beispiel

```
100 A$="EIN SATZ IST AUCH EINE ZEICHENKETTE"
110 B$="KETTE"
120 PRINT INSTR(A$,B$)
```

Dieses kleine Programm zeigt nach dem Start das Ergebnis

```
31
```

an.

### Anmerkung

In den BASIC-Versionen 2.0 und 4.0 kann man den INSTR-Befehl mit der folgenden kleinen Routine simulieren:

```
1000 P=0: REM INSTRING-UNTERPROGRAMM
1010 FOR I=1 TO LEN(A$)-LEN(B$)+1
1020 : IF MID$(A$,I,LEN(B$))=B$ THEN P=I:RETURN
1030 NEXT I
1040 RETURN
```

Dieses Unterprogramm sucht die Position P, ab der sich die Zeichenkette B$ in der Zeichenkette A$ befindet, und ist also gleichbedeutend mit der Anweisungszeile:

```
1000 P=INSTR(A$,B$):RETURN
```

## (BASIC 2.0, 4.0, 3.5, 7.0) INT

**Abkürzung:** keine möglich

**Typ:** numerische Funktion

**Syntax:**
```
INT(numerische Variable,
Konstante oder Ausdruck)
```

**Argumentbereich:** Gleitkommazahlen

**Ergebnisbereich:** ganze Zahlen

Die Funktion INT liefert die größte ganze Zahl, die kleiner oder gleich ihrem Argument ist. Bei positiven Zahlen beseitigt INT nur die Nachkommastellen der Zahl:

`INT(5.5)` ergibt 5

Bei negativen Zahlen liefert INT die nächstniedrigere negative Zahl, also:

`INT(-5.5)` ergibt -6

### Beispiel

Unser Beispielprogramm ist das Unterprogramm, das unter dem Stichwort GOSUB aufgerufen wird. Es berechnet den Wochentag aus einem Datum, das in der Form

J=Jahr, M=Monat und T=Tag

gegeben ist. Der 25.10.1985 ist somit durch

J=1985, M=10 und T=25

gegeben.

Das Ergebnis wird in der Variablen W zurückgegeben, wobei W die Werte von 1 bis 7 haben kann, für 1=Sonntag, 2=Montag usw.

Das Programm sieht folgendermaßen aus:

```
50000 REM ****************************
50010 REM T=TAG
50020 REM M=MONAT
50030 REM J=JAHR
50040 DEF FNMOD(X)=X-INT(X/D)*D
50050 IF M=1 OR M=2 THEN J=J-1:M=M+12
50060 H=T+2*M+INT((3*M+3)/5)+J+INT(J/4)-INT(J/100)
50070 H=H+INT(J/400)+1
50080 D=7:W=FNMOD(H)
50090 RETURN
```

*Hinweis:* Dieses Programm ist als Unterprogramm konzipiert und daher selbständig, ohne aufrufendes Hauptprogramm, nicht lauffähig. Ein solches Hauptprogramm, das die Werte für unser Unterprogramm zur Verfügung stellt, finden Sie unter dem Stichwort GOSUB.

Die in der Zeile 50060 codierte Formel

```
50060 H=T+2*M+INT((3*M+3)/5)+J+INT(J/4)-INT(J/100)
```

macht intensiven Gebrauch von der Funktion INT und geht auf den deutschen Mathematiker Chr. Zeller zurück.

Die zweite interessante Zeile ist die folgende:

```
50080 D=7:W=FNMOD(H)
```

In dieser Zeile wird die mathematische Aufgabe

W = H modulo 7

gelöst, die besagt, daß der Variablen W der Divisionsrest aus der Division von H durch 7 zugewiesen werden soll. Die Funktion FNMOD, die wir für diese Aufgabe einsetzen, wurde unter dem Stichwort DEF FN definiert.

*Hinweis:* In einigen BASIC-Erweiterungen steht dafür speziell eine MOD-Funktion zur Verfügung, die das Ganze natürlich wesentlich vereinfacht.

# (BASIC 3.5, 7.0) JOY

**Abkürzung:** J<SHIFT>O

**Typ:** Joystick-Funktion

**Syntax:**
```
JOY(1)
JOY(2)
```

**Argumentbereich:** ganze Zahlen 1 - 2

**Ergebnisbereich:** ganze Zahlen 0 - 8 und 128 - 136

Mit der JOY-Funktion werden die Joystick-Eingänge abgefragt (Eingang 1 und Eingang 2). Dabei sind folgende Werte von Bedeutung:

| | |
|---|---|
| 0 | keine Funktion |
| 1 | oben |
| 2 | oben rechts |
| 3 | rechts |
| 4 | unten rechts |
| 5 | unten |
| 6 | unten links |
| 7 | links |
| 8 | oben links |
| 128 | Feuerknopf |

Wenn der Feuerknopf gleichzeitig mit einer der anderen Funktionen betätigt wird, treten Werte über 128 auf.

**Beispiel**

```
100 IF JOY(1)>128 THEN PRINT"SCHUSS"
```

# KEY (BASIC 3.5, 7.0)

**Abkürzung:** K<SHIFT>E

**Typ:** Programmierhilfe

**Syntax:**
```
KEY
KEY Nummer,"Tastenbelegung"
```

**Parameterbereich:** ganze Zahlen 1 - 8

Mit dem KEY-Befehl lassen sich die vordefinierten Funktionstasten anders belegen. Wenn Sie nur das Wort KEY allein eingeben, werden alle Tastenbelegungen auf dem Bildschirm aufgelistet.

Mit

```
KEY 4,"LIST"
```

wird beispielsweise der LIST-Befehl auf die Funktionstaste F4 gelegt. Jedesmal, wenn Sie nun auf die Taste F4 drücken, erscheint das Wort LIST auf dem Bildschirm mit dem Cursor dahinter.

Wenn die Anweisung nach Drücken der F4-Taste auch sofort ausgeführt werden soll, können Sie auch noch das RETURN gleich mit auf die Funktionstaste legen. Der Befehl dazu sieht so aus:

```
KEY 4,"LIST"+CHR$(13)
```

Bei manchen Befehlen ist gleich das erste Anführungszeichen vor der folgenden Zeichenkette mit aufgeführt. Das wird mit der Angabe von CHR$(34) erreicht.

Wenn Sie das System einschalten, sind die Funktionstasten wie folgt vorbelegt:

| Taste | Funktion | Bedeutung |
|---|---|---|
| F1 | GRAPHIC | schaltet in den Grafikmodus um |
| F2 | DLOAD" | lädt Programm von Diskette |

| Taste | Funktion | Bedeutung |
|---|---|---|
| F3 | DIRECTORY | zeigt Inhaltsverzeichnis der Diskette |
| F4 | SCNCLR | löscht Bildschirm im Grafikmodus |
| F5 | DSAVE" | speichert Programm auf Diskette |
| F6 | RUN | startet Programm |
| F7 | LIST | listet Programmzeilen auf dem Bildschirm |
| F8 | MONITOR | schaltet auf Maschinensprache-Monitor um |

Lediglich die Taste F8 ist beim C16 und C116 anders belegt, und zwar mit HELP. Und beim Plus/4 ist zusätzlich die Taste F1 anders belegt, und zwar mit SYS1525: 3-PLUS-1.

### Anmerkung

Wenn Sie beim C16 oder C116 die Funktionstasten umbelegt haben und sie wieder in den Grundzustand, der nach dem Einschalten herrscht, zurückversetzen wollen, können Sie das am schnellsten mit

```
SYS 62359
```

erreichen.

# LEFT$ (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** LE<SHIFT>F

**Typ:** Zeichenkettenfunktion

**Syntax:**
```
LEFT$(Zeichenkettenvariable,
-konstante oder -ausdruck,
ganze Zahl)
```

**Argumentbereich:** Zeichenketten
Bytezahl

**Ergebnisbereich:** Zeichenketten

Die Funktion LEFT$ löst den linken Teil aus einer gegebenen Zeichenkette heraus. Demgemäß hat sie zwei Parameter. Der erste Parameter ist die Zeichenkette, aus der von links her so viele Zeichen herauszulösen sind, wie der zweite Parameter angibt. Dieser kann Werte zwischen 0 und 255 annehmen. Ist er größer, so wird die Fehlermeldung ILLEGAL QUANTITY ERROR ausgegeben.

Das folgende kleine Beispielprogramm

```
10 A$="COMMODORE"
20 B$=LEFT$(A$,3)
30 PRINT B$
```

ergibt in B$ die Ausgabe COM.

Wenn der zweite Parameter 0 ist, wird eine leere Zeichenkette zurückgeliefert, ist er größer als die Zeichenkette, dann ist die ganze Zeichenkette das Ergebnis.

### Anmerkung

LEFT$ kann bequem dazu verwendet werden, um den Cursor auf dem Bildschirm an bestimmte Stellen zu positionieren. Dazu muß im Programm eine Zeichenkette aus Cursor-Kontrollzeichen erzeugt werden, z. B. A$="<Pfeil nach unten><Pfeil nach unten>...usw.".Dann kann beispielsweise mit PRINT LEFT$(A$,3) eine Cursor-Bewegung von 3 Zeilen nach unten ausgelöst werden.

## (BASIC 2.0, 4.0, 3.5, 7.0) LEN

**Abkürzung:** keine möglich

**Typ:** numerische Funktion

**Syntax:** `LEN(Zeichenkettenvariable, -konstante oder -ausdruck)`

**Argumentbereich:** Zeichenketten

**Ergebnisbereich:** Bytezahl

Die Funktion LEN ermittelt die Länge einer Zeichenkette. Dabei werden auch Leerzeichen und nichtdruckbare Zeichen mitgezählt.

### Beispiele

```
10 A$="MORGENSTUND' HAT "
20 B$="GOLD IM MUND"
30 C$=""
40 PRINT LEN(A$+B$),LEN(C$)
50 END

RUN
29        0
READY.
```

Die Funktion LEN ist sehr praktisch, wenn Sie eine Zeichenkette zentriert auf dem Bildschirm ausgeben wollen. Für einen Bildschirm mit 40 Zeichen pro Zeile lautet die Formel als Teil einer PRINT TAB-Funktion:

```
100 INPUT"GEBEN SIE EINE ZEICHENKETTE EIN";A$
110 PRINT TAB((40-LEN(A$))/2)A$
120 END
```

wobei A$ die zu zentrierende Zeichenkette ist. Es wird die Differenz zwischen der Länge der Zeichenkette und der Breite des Bildschirms ermittelt und durch zwei geteilt.

# LET (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** L<SHIFT>E

**Typ:** elementares Befehlswort

**Syntax:** LET Variable=Wert

**Parameterbereich:** Variablennamen
Zahlen und Zeichenketten

Die LET-Anweisung weist einer Variablen einen Wert zu. Wenn die Variable noch nicht im Programm existiert, gibt LET ihr einen Anfangswert. Hat die Variable bereits einen Wert, so erhält sie durch LET einen neuen.

Die LET-Anweisung kann als direkt auszuführender Befehl oder als Programmanweisung verwendet werden. Das Wort LET kann ersatzlos weggelassen werden und wird deshalb selten benutzt.

```
10 LET A=1000
```

ist gleichbedeutend mit:

```
10 A=1000
```

## (BASIC 2.0, 4.0, 3.5, 7.0) LIST

**Abkürzung:** L<SHIFT>I

**Typ:** Programmierhilfe

**Syntax:**

```
LIST
```
listet alle Programmzeilen

```
LIST L
```
listet die Programmzeile mit der Nummer L

```
LIST L1-L2
```
listet die Programmzeilen L1 bis L2

```
LIST -L2
```
listet alle Programmzeilen bis einschließlich L2

```
LIST L1-
```
listet alle Programmzeilen von L1 bis zum Ende

**Parameterbereich:** Zeilennummern

Der LIST-Befehl bewirkt die Ausgabe von Programmzeilen auf dem Rechner. Das Programm muß sich zu dieser Zeit im Arbeitsspeicher befinden. LIST wird meist als direkt auszuführender Befehl verwendet, kann aber auch Teil eines Programms sein.

Dabei muß man zwischen den BASIC-Versionen unterscheiden. Wenn in BASIC 2.0 oder 4.0 LIST als Programmanweisung eingesetzt wird, endet die Programmausführung nach dem Listing. Auch wenn CONT eingegeben wird, ist keine Fortsetzung möglich, es wird lediglich das Programm erneut aufgelistet.

In BASIC 3.5 und 7.0 wird das Programm nach dem Listen weiter bearbeitet:

```
100 PRINT"VOR DEM LIST-BEFEHL"
110 LIST
120 PRINT"NACH DEM LIST-BEFEHL"
130 END
```

### Anmerkungen

Falls eine Programmliste nicht auf den Bildschirm paßt, rollt der Rechner die Bildschirmausgabe bis zum Ende des Programms hoch. Um dieses "Scrollen" zu verlangsamen, kann man die CTRL-Taste drücken, während das Programm aufgelistet wird.

Beim C128 und beim C16, C116 und Plus/4 liegt der LIST-Befehl auf einer Funktionstaste und wird auf Tastendruck hin ausgeführt.

Beim C128 kann das Listen von Programmen mit der No Scroll-Taste so lange unterbrochen werden, bis man eine beliebige Taste zur Fortsetzung drückt.

## (BASIC 2.0, 4.0, 3.5, 7.0) LOAD

**Abkürzung:** L<SHIFT>O

**Typ:** Systembefehl für Eingabe

**Syntax:**
```
LOAD "DATEINAME"
```
gleichbedeutend mit:
```
LOAD "DATEINAME",1
```
für das Laden von Kassette

```
LOAD "DATEINAME",8
```
nur in BASIC 2.0 für das Laden von Diskette

```
LOAD "$",8
```
nur in BASIC 2.0 für das Laden des Inhaltsverzeichnisses der Diskette

**Parameterbereich:** Dateinamen
Gerätenummern

### 1. Laden von Kassette

Der LOAD-Befehl holt ein BASIC-Programm von einer Kassettendatei und bringt es in den Rechnerspeicher. Dabei kann die Gerätenummer 1 weggelassen werden. Auch der Dateiname muß nicht unbedingt angegeben werden. Dann wird allerdings das nächste Programm, das der Rechner findet, geladen.

Wenn Sie den LOAD-Befehl eingegeben haben, der Kassettenrecorder jedoch nicht eingeschaltet war, gibt der Rechner die Meldung aus:

```
PRESS PLAY ON TAPE
```
(d. h. betätigen Sie die PLAY- oder Starttaste)

Sobald Sie dieser Aufforderung gefolgt sind, wird der Rechner den Recorder so lange laufen lassen, bis das nächste oder, falls ein Dateinamen angegeben wurde, das gewünschte Programm gefunden wurde. Dieses kann dann mit RUN gestartet werden.

**Anmerkung**

Einen sogenannten Autostart, d. h. automatischen Programmstart nach dem Laden, erzielen Sie, wenn Sie die Tasten SHIFT und RUN/STOP gleichzeitig drücken. Damit wird das nächste Programm geladen und automatisch gestartet.

**2. Laden von Diskette** (nur BASIC 2.0)

Um BASIC-Programme von Diskette zu laden, müssen Sie in den Versionen 4.0, 3.5 und 7.0 den Befehl DLOAD verwenden (siehe dort). Nur in BASIC 2.0 wird der LOAD-Befehl mit gleichzeitiger Angabe der Gerätenummer (8 für Diskettenlaufwerk) benutzt:

```
LOAD "DATEINAME",8
```

Sollte das von Ihnen gewünschte Programm auf der Diskette nicht vorhanden sein, wird der Rechner die folgende Fehlermeldung ausgeben:

```
? FILE NOT FOUND ERROR
```
(d. h. Datei nicht gefunden)

Einfacher und sicherer ist es deshalb, sich zunächst das Inhaltsverzeichnis der Diskette anzeigen zu lassen:

```
LOAD"$",8
```

Sobald es geladen ist, können Sie es sich mit LIST auf dem Bildschirm anzeigen lassen. Bei den BASIC-Versionen 4.0, 3.5 und 7.0 dient dazu der DIRECTORY-Befehl (siehe dort). Um ein Programm zu laden, brauchen Sie nun nur noch mit dem Cursor zu der betreffenden Zeile hochzufahren, LOAD vor dem Dateinamen einzufügen (oder lO), hinter die zweiten Anführungszeichen ,8: zu setzen und RETURN zu drücken. Der Doppelpunkt hinter der 8 bewirkt dabei, daß der Rest der Zeile ignoriert wird, sonst müßte er explizit gelöscht werden, um einen Fehler zu vermeiden.

Der Befehl

```
LOAD"*",8
```

lädt das erste Programm von der Diskette, und

```
LOAD"PR*",8
```

lädt das erste Programm, dessen Name mit PR beginnt.

### Anmerkung

Wenn Sie ein Programm oder Inhaltsverzeichnis in den Arbeitsspeicher laden, verlieren Sie jedes Programm, das sich vorher im Speicher befunden hat. Sie sollten sich daher vergewissern, daß Sie das gerade im Speicher enthaltene Programm gesichert haben, bevor Sie den LOAD-Befehl benutzen.

Einen Ausweg für dieses Problem zeigt das Beispielprogramm unter dem Stichwort GET#.

# LOCATE (BASIC 3.5, 7.0)

**Abkürzung:** LO<SHIFT>C

**Typ:** Grafikbefehl

**Syntax:** LOCATE X,Y

**Parameterbereich:** Bildschirmkoordinaten in den unten aufgeführten Bereichen

Der Befehl LOCATE setzt den unsichtbaren Grafikcursor an die angegebenen Koordinaten. Dabei können die X- und die Y-Koordinate je nach gewähltem Modus folgende Maximalwerte annehmen:

Modus 0 X max. 39, Y max. 24
Modus 1 X max. 319, Y max. 199
Modus 2 X max. 319, Y max. 159
Modus 3 X max. 159, Y max. 199
Modus 4 X max. 159, Y max. 159
Modus 5 X max. 79, Y max. 24

Falls die Skalierung mit SCALE geändert wurde, können maximal 1024 Bildschirmpunkte angesprungen werden.

Wenn man den Cursor relativ zur aktuellen Position verschieben will, kann man die Koordinaten mit entsprechenden Vorzeichen angeben:

```
LOCATE +10,-12
```

bringt den Cursor um 10 Pixel nach rechts und 12 Pixel nach oben.

### Beispiel

Wir wollen in unserem Beispielprogramm ein Karomuster erzeugen, indem wir mit LOCATE den Grafikcursor relativ verschieben und dann die Fläche, in der wir uns aktuell befinden, mit PAINT färben.

```
100 COLOR 0,2
110 GRAPHIC 1,1
```

```
120 FOR I=0 TO 319 STEP 16
130 : DRAW 1,I,0 TO I,199
140 NEXT I
150 FOR I=0 TO 199 STEP 14
160 : DRAW 1,0,I*1.1 TO 319,I*1.1
170 NEXT I
180 F=3
190 FOR J=1 TO 199 STEP 32
200 : LOCATE 1,J
210 : FOR I=1 TO 10
220 :   COLOR 1,F
230 :   PAINT 1,,1
240 :   LOCATE +32,+0
250 :   F=F+1:IF F=17 THEN F=3
260 : NEXT I
270 NEXT J
280 GETKEY T$
290 GRAPHIC 0,1
300 COLOR 0,12
310 END
```

# LOG (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** keine möglich

**Typ:** trigonometrische Funktion

**Syntax:** `LOG(numerische Variable, Konstante oder Ausdruck)`

**Argumentbereich:** Gleitkommazahlen größer als 0

**Ergebnisbereich:** Gleitkommazahlen

Die Funktion LOG liefert den natürlichen Logarithmus (zur Basis e) einer Zahl. Dabei muß das Argument größer als null sein. Die LOG-Funktion ist die Umkehrfunktion der EXP-Funktion.

### Beispiel

```
100 FOR I=0.0001 TO 100 STEP 5
110 : PRINT "LOG(";I;")=";LOG(I)
120 NEXT I
130 END
```

### Anmerkung

Null oder negative Argumente verursachen die folgende Fehlermeldung:

```
?ILLEGAL QUANTITY ERROR
```

## (BASIC 2.0, 4.0, 3.5, 7.0) MID$

**Abkürzung:** M<SHIFT>I

**Typ:** Zeichenkettenfunktion

**Syntax:** MID$(Z$,A,L)
Z$ ist Zeichenkettenvariable, -konstante oder -ausdruck
A ist die Nummer des Zeichens, ab dem der Teilstring beginnt
L ist die Länge des Teilstrings

**Argumentbereich:** Zeichenketten
Bytezahlen

**Ergebnisbereich:** Zeichenketten

Mit der MID$-Funktion kann ein Teilstring aus einer Zeichenkette herausgelöst werden. Demgemäß hat diese Funktion drei Parameter.

Der erste Parameter Z$ ist die vorgelegte Zeichenkette.

Der zweite Parameter A legt fest, beim wievielten Zeichen die neue Zeichenkette beginnen soll. Er kann Werte zwischen 0 und 255 annehmen. Ist A aber größer als die Länge der Zeichenkette, so ist der Teilstring leer.

Der dritte Parameter L gibt die Anzahl der zu übertragenden Zeichen an. Ist er größer als die mögliche Anzahl der Zeichen, wird der ganze Rest der Zeichenkette genommen; ebenso, wenn er völlig fehlt.

### Beispiel

```
10 A$="GUTEN"
20 B$="MORGEN TAG ABEND"
30 PRINT A$;MID$(B$,8,3)
40 END

RUN
GUTEN TAG
READY.
```

### Anmerkungen

In BASIC 7.0 kann über die MID$-Funktion eine Zeichenkette teilverändert werden:

```
100 A$="HEUTE IST SEHR SCHOENES WETTER"
110 MID$(A$,11,4)="KEIN"
120 PRINT A$
```

In der Zeichenkette A$ werden in diesem Beispiel die Zeichen ab der Position 11 in der Länge 4 durch das Wort KEIN ersetzt. Das Ergebnis dieses kleinen Programms wäre dann:

```
HEUTE IST KEIN SCHOENES WETTER
```

Weitere Zeichenkettenfunktionen sind LEFT$, RIGHT$ und INSTR (siehe dort).

# (BASIC 3.5, 7.0) MONITOR

**Abkürzung:** MO<SHIFT>N

**Typ:** Systembefehl

**Syntax:** MONITOR

Mit dem MONITOR-Befehl können Sie in den Maschinensprache-Modus umschalten. Zurück in den BASIC-Modus kommen Sie durch Eingabe von X.

Die Maschinensprache-Befehle wollen wir nur alphabetisch aufzeigen, aber nicht ausführlich erläutern:

| | | |
|---|---|---|
| A | Assemble | Assemblieren |
| C | Compare | Vergleichen zweier Speicherbereiche |
| D | Disassemble | Disassemblieren |
| F | Fill | Speicherbereich mit Byte füllen |
| G | Go | Sprung zur angegebenen Adresse |
| H | Hunt | Suchen nach einer Zeichenfolge |
| L | Load | Datei laden |
| M | Memory | Speicher anzeigen |
| R | Register | Register anzeigen |
| S | Save | Datei schreiben |
| T | Transfer | Speicherbereich kopieren |
| V | Verify | Datei vergleichen |
| X | Exit | Rückkehr ins BASIC |
| > | | Speicher ändern |
| ; | | Register ändern |
| @ | | Fehlermeldung anzeigen |

# MOVSPR (BASIC 7.0)

**Abkürzung:** M<SHIFT>O

**Typ:** Sprite-Befehl

**Syntax:**
```
MOVSPR Spritenummer,X,Y

MOVSPR Spritenummer,
Winkel#Geschwindigkeit
```

**Parameterbereich:** Spritenummern
Bildschirmkoordinaten
Winkel - Gleitkommazahlen 0 - 360
Geschwindigkeit - ganze Zahlen 0 - 15

MOVSPR bringt das Sprite mit der angegebenen Nummer an die Stelle auf dem Bildschirm, die durch die Koordinaten X und Y festgelegt werden. Dabei ist auch eine relative Verschiebung zur aktuellen Position des Sprites möglich, indem die Koordinaten mit den entsprechenden Vorzeichen angegeben werden:

```
MOVSPR 3,-10,+15
```

bewegt das Sprite mit der Nummer 3 um 10 Pixel nach links und 15 Pixel nach unten.

Die zweite Form der MOVSPR-Anweisung setzt das bezeichnete Sprite unter dem angegebenen Winkel mit der gewählten Geschwindigkeit in ständige Bewegung:

```
MOVSPR 2,90#10
```

Diese Anweisung dreht das Sprite mit der Nummer 2 um 90 Grad, und zwar ziemlich schnell. Der Geschwindigkeitsparameter kann die Werte 0 (langsam) bis 15 (schnell) annehmen.

Wenn das Sprite dabei den Bildschirmrand erreicht, läuft es hinaus und auf der entgegengesetzten Seite wieder hinein.

### Beispiel

Um ein Sprite in Bewegung zu setzen, müssen wir erst einmal eines definieren. Dazu rufen wir mit SPRDEF (siehe auch dort) die Definitionsmatrix auf und erstellen mit den Zifferntasten 1 und 2 eine Figur. Als erstes müssen wir die Nummer des Sprites festlegen. Wir wählen SPRITE NUMBER 1. Wenn Sie die Nummer 1 bereits mit einem Sprite belegt haben, das Sie nicht verlieren möchten, sollten Sie eine andere Nummer wählen. Allerdings müssen Sie dann auch diese andere Nummer in den Zeilen 180, 190, 220, 240 und 280, und zwar immer anstelle des ersten Parameters im nachfolgenden Programm einsetzen.

Wir haben für unser Sprite 24 Spalten und 21 Reihen zur Verfügung. Mit dem Cursor können wir frei in dem Feld umherfahren und nach dem folgenden Zählschema unsere Figur erstellen:

```
----------xxxxx---------
---------xxxxxxx--------
---------xxxx-xx--------
-------xxxxxxxxxx-------
-----xx-x-xxxxxx--------
----x--x--xxxxx---------
------x-----xx----------
-----x----xxxxxx--------
---------xxxxxxxxxxxxx--
--------xxxxxxxxx-------
-------xx-xxxxxxx-------
-----xxx----xxx---------
----xx-----xxxxx--------
----------xxxxxxx-------
----------xxxxxxxx------
---------xxx---xxxx-----
---------xxx----xxx-----
--------xxxx---xxx------
------xxxx-----xxx------
------xx------xxx-------
------xx-------xxxx-----
```

Drücken Sie nun SHIFT/RETURN und noch einmal RETURN. Jetzt sind wir wieder im Textmodus. Wir wollen als Hintergund für unsere Sprite-Animation das Beispiel, das als letztes unter dem Stichwort SSHAPE angegeben wurde, verwenden. Das vollständige Programm sieht nun so aus:

```
100 PRINT CHR$(147)
110 GRAPHIC 1,1
115 COLOR 0,1
120 DRAW 1,10,0 TO 20,8 TO 20,20 TO 0,20
    TO 0,8 TO 10,0
125 BOX 1,8,8,12,12,1
126 BOX 1,8,14,12,20,1
130 SSHAPE A$,0,0,20,20
140 FOR I=0 TO 199 STEP 20
150 : FOR J=0 TO 320 STEP 40
160 :   GSHAPE A$,J,I
170 NEXT J,I
180 SPRITE 1,1,11
190 MOVSPR 1,0,70
200 FOR L=1 TO 5
210 : FOR K=0 TO 499
220 :   MOVSPR 1,+1,+0
230 : NEXT K
240 : MOVSPR 1,+0,+39
250 NEXT L
260 GETKEY T$
270 COLOR 0,12
280 SPRITE 1,0
290 GRAPHIC 0,1
300 END
```

Der Programmteil bis zur Zeile 170 ist derjenige, der den Hintergrund, unsere Reihenhäuser, produziert (siehe auch SSHAPE). In Zeile 180 wird Sprite 1 mit der Farbe Rosa aktiviert und in Zeile 190 links zwischen die oberen beiden Häuserreihen auf den Bildschirm gesetzt:

```
190 MOVSPR 1,0,70
```

Fünfmal wird dann das Sprite von links nach rechts über den Schirm geschoben:

```
220 MOVSPR 1,+1,+0
```

Dazwischen wird es jedesmal um 39 Positionen nach unten zwischen die nächsten zwei Häuserreihen verschoben:

```
240 MOVSPR 1,+0,+39
```

In dieser Siedlung ist es auch gewiß kein Wunder, wenn man sein Haus nicht mehr wiederfindet.

## (BASIC 2.0, 4.0, 3.5, 7.0) NEW

**Abkürzung:** keine möglich

**Typ:** Systembefehl

**Syntax:** NEW

Dieser Befehl löscht das aktuelle Programm im Rechnerspeicher. Nach der Eingabe von NEW können Sie das Programm nicht wieder zurückholen, es sei denn, Sie haben es vorher auf Diskette oder Kassette gespeichert. Außerdem werden alle Variablen zurückgesetzt:

```
V=5
PRINT V
5
NEW
PRINT V
0
```

NEW wird meist als direkt auszuführender Befehl verwendet, um den Arbeitsspeicher aufzuräumen. Wenn NEW als Programmanweisung eingesetzt wird, löscht es das Programm, in dem es sich befindet.

### Anmerkungen

Daß ein mit NEW gelöschtes Programm doch nicht völlig verschwunden ist, wissen Sie vielleicht, da es ja fast in jeder BASIC-Erweiterung einen OLD-Befehl gibt, der das Programm wieder hervorbringt.

Für den C128 genügt eine einzige Zeile, und das Programm läßt sich wieder listen:

```
POKE PEEK(45)+256*PEEK(46)+1,28:SYS 20303
```

Die Routine für den C64 und VC20 ist etwas länger. Lassen Sie sich bei der Eingabe der folgenden Zeilen nicht durch die Meldung OUT OF MEMORY ERROR beirren.

```
POKE 46,PEEK(56)-1:POKE 45,PEEK(55)+247:CLR <RETURN>
POKE PEEK(44)*256+PEEK(43)+1,PEEK(44) <RETURN>
63999 <RETURN>
FOR I=PEEK(44)*256+PEEK(43) TO PEEK(46)*256+PEEK(45):IF
PEEK(I) OR PEEK(I+1) OR PEEK(I+2) THEN NEXT <RETURN>
POKE 45,(I+3) AND 255:POKE 46,(I+3)/256:CLR <RETURN>
```

## (BASIC 2.0, 4.0, 3.5, 7.0) ON GOSUB

**Abkürzung:** ON GO<SHIFT>S

**Typ:** Befehlswort zur Programm-Ablaufsteuerung

**Syntax:**
```
ON Variable oder Ausdruck
GOSUB Zeilennummernliste
```

**Parameterbereich:** positive Gleitkommazahlen
Zeilennummern

Mit dem Befehlswort ON wird eine Auswahl eingeleitet. Gewählt wird zwischen mehreren Zeilen einer Liste in Abhängigkeit von der Variablen oder des Ausdrucks nach dem Schlüsselwort ON:

```
100 ON N GOSUB 1000,2000,3000
```

Je nach dem Wert von N wird auf die entsprechende Zeilennummer in der Liste verzweigt. Hat also die Variable N im obigen Beispiel den Wert 2, so wird zur Zeile 2000 gesprungen. Der Ausdruck nach ON wird als gerundeter Integer-Wert berechnet und dann entsprechend verzweigt.

Ist der Wert der Variablen hinter ON kleiner als null, wird das Programm mit der folgenden Fehlermeldung abbrechen:

```
?ILLEGAL QUANTITY ERROR IN 100
```

Die Variable in der ON-Anweisung in Zeile 100 hat einen Wert außerhalb des erlaubten Bereichs.

### Beispiel

Die ON...GOSUB-Anweisung eignet sich ideal für Mehrfachverzweigungen nach einer Menü-Auswahl. In unserem Menü haben Sie die Wahl, in welcher der vier Grundrechenarten Sie sich üben wollen:

```
100 REM**********************************
110 REM*         RECHENQUIZ              *
120 REM**********************************
```

```
130 DF$=CHR$(147)
140 PRINT:PRINT DF$
150 PRINT TAB(14)"RECHENQUIZ"
160 PRINT:PRINT
170 PRINT TAB(10)"1. ADDITION"
180 PRINT TAB(10)"2. SUBTRAKTION"
190 PRINT TAB(10)"3. MULTIPLIKATION"
200 PRINT TAB(10)"4. DIVISION"
210 PRINT TAB(10)"5. ENDE"
220 PRINT:PRINT:PRINT:PRINT
230 INPUT"  WAEHLEN SIE EINE OPTION (1-5)"
    ;WAHL
240 IF WAHL<1 OR WAHL>5 THEN PRINT CHR$(7)
    :GOTO 230
250 ON WAHL GOSUB 280,420,570,710,270
260 GOTO 130
270 END
280 REM*********************************
290 REM*        ADDITION                 *
300 REM*********************************
310 PRINT DF$
320 PRINT TAB(14)"ADDITION"
330 PRINT
340 FOR I=1 TO 10
350 : A=INT(RND(RND(TI))*100)
360 : B=INT(RND(RND(TI))*100)
370 : PRINT A;"+";B;"=";:INPUT SUMME
380 : IF SUMME<>(A+B) THEN PRINT
      "LEIDER FALSCH, DAS ERGEBNIS WAR"
      ;A+B:ELSE PRINT"RICHTIG"
390 NEXT I
400 GETKEY T$
410 RETURN
420 REM*********************************
430 REM*        SUBTRAKTION              *
440 REM*********************************
450 PRINT DF$
460 PRINT TAB(14)"SUBTRAKTION"
470 PRINT
480 FOR I=1 TO 10
490 : A=INT(RND(RND(TI))*100)
500 : B=INT(RND(RND(TI))*100)
510 : IF B>A THEN H=B:B=A:A=H
520 : PRINT A;"-";B;"=";:INPUT SUMME
530 : IF SUMME<>(A-B) THEN PRINT
```

```
      "LEIDER FALSCH, DAS ERGEBNIS WAR"
      ;A-B:ELSE PRINT"RICHTIG"
540 NEXT I
550 GETKEY T$
560 RETURN
570 REM*********************************
580 REM*         MULTIPLIKATION          *
590 REM*********************************
600 PRINT DF$
610 PRINT TAB(14)"MULTIPLIKATION"
620 PRINT
630 FOR I=1 TO 10
640 : A=INT(RND(RND(TI))*10)
650 : B=INT(RND(RND(TI))*100)
660 : PRINT A;"*";B;"=";:INPUT SUMME
670 : IF SUMME<>(A*B) THEN PRINT
      "LEIDER FALSCH, DAS ERGEBNIS WAR"
      ;A*B:ELSE PRINT"RICHTIG"
680 NEXT I
690 GETKEY T$
700 RETURN
710 REM*********************************
720 REM*         DIVISION                *
730 REM*********************************
740 PRINT DF$
750 PRINT TAB(14)"DIVISION"
760 PRINT
770 FOR I=1 TO 10
780 : B=INT(RND(RND(TI))*20)+1
790 : A=(INT(RND(RND(TI))*10)+1)*B
800 : PRINT A;":";B;"=";:INPUT SUMME
810 : IF SUMME<>(A/B) THEN PRINT
      "LEIDER FALSCH, DAS ERGEBNIS WAR"
      ;A/B:ELSE PRINT"RICHTIG"
820 NEXT I
830 GETKEY T$
840 RETURN
```

Das Programm kann von Ihnen beliebig verändert werden. In der jetzigen Version bietet es in jeder Rechenart 10 Aufgaben an. Nach einem Tastendruck kommt man wieder zur Hauptauswahl zurück. Sie könnten dem Benutzer freistellen, wie viele Aufgaben er jeweils lösen will. Man kann dabei die richtigen und die falschen Antworten zählen. Außerdem kann man den Anwender die Größenordnung der zu lösenden Aufgaben wählen lassen.

# ON GOTO (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** ON G<SHIFT>O

**Typ:** Befehlswort zur Programm-Ablaufsteuerung

**Syntax:**
```
ON Variable oder Ausdruck
GOTO Zeilennummernliste
```

**Parameterbereich:** Gleitkommazahlen
Zeilennummern

Der ON...GOTO-Befehl arbeitet im Prinzip genauso wie der ON...GOSUB-Befehl (siehe dort), nur daß nach der Verzweigung zu der entsprechenden Zeilennummer kein Rücksprung durch RETURN ausgelöst wird.

### Beispiel

Das Beispiel, das unter dem Stichwort GOSUB aufgeführt ist, läßt sich fast genauso mit ON...GOTO realisieren. Sie brauchen lediglich die folgenden Zeilen zu ändern:

```
250 ON WAHL GOTO 280,420,570,710,270
```

Zeile 260 entfällt

```
410 GOTO 130
560 GOTO 130
700 GOTO 130
840 GOTO 130
```

## (BASIC 2.0, 4.0, 3.5, 7.0) OPEN

**Abkürzung:** O<SHIFT>P

**Typ:** Systembefehl für Ein-/Ausgabe

**Syntax:**

```
OPEN Dateinummer

OPEN Dateinummer,Gerätenummer

OPEN Dateinummer,Gerätenummer,
Sekundäradresse

OPEN Dateinummer,Gerätenummer,
Sekundäradresse,"Dateiname,
Typ,Modus"
```

**Parameterbereich:**
Dateinummern
Gerätenummern
Dateinamen
Sekundäradresse - Bytezahlen
Typ - Zeichen S oder R
Modus - Zeichen R, W oder A

Der OPEN-Befehl öffnet eine Datei zum Lesen oder Schreiben. Bevor die Befehle INPUT#, PRINT#, GET# oder CMD benutzt werden können, muß einmal ein OPEN für die gewünschte Datei gegeben werden. Automatisch wird auch durch den OPEN-Befehl ein Dateipuffer angelegt, der die unterschiedlichen Arbeitsgeschwindigkeiten der beteiligten Geräte ausgleichen kann. Gelesen und geschrieben wird dann immer pufferweise.

Die Parameter für den OPEN-Befehl sind im einzelnen:

- Dateinummer: ist der logische Name des Übertragungskanals und kann irgendeine Zahl zwischen 1 und 255 sein. Diese Zahl ist auch jeweils die Referenz in den INPUT#-, GET#- und PRINT#-Befehlen.

- Gerätenummer: gibt das Gerät an, das angesprochen wird. Dabei sind folgende Zuordnungen bereits vorgesehen.

| Gerätenummer | Gerät |
|---|---|
| 0 | Tastatur |
| 1 | Kassettenrecorder |
| 2 | RS-232-Gerät |
| 3 | Bildschirm |
| 4 | Drucker |
| 5 | Drucker |
| 8 | Diskettenlaufwerk |
| 4-127 | Geräte über den seriellen Bus |
| 128-255 | Geräte über den seriellen Bus, mit Zeilenvorschub nach Zeilenende |

Wenn keine Angabe gemacht wird, ist automatisch das Gerät mit der Nummer 1, der Kassettenrecorder, angesprochen.

- Sekundäradresse: kann eine Zahl von 0 bis 255 sein. Dieselbe Sekundäradresse kann je nach Gerät unterschiedliche Wirkung haben:

| Gerät | Sekundäradresse | Funktion |
|---|---|---|
| Tastatur | 1-255 | keine Wirkung |
| Bildschirm | 1-255 | keine Wirkung |
| Kassettenrecorder | 0 | Banddatei lesen |
| | 1 | Banddatei schreiben |
| | 2 | Banddatei schreiben, mit Endemarke (EOT) |
| Diskettenlaufwerk | 1-14 | Datenkanal öffnen |
| | 15 | Befehlskanal öffnen |
| Drucker | 0 | Großschrift-/Grafikmodus |
| | 7 | Groß-/Kleinschriftmodus |

- Hinter dem Dateinamen können der Typ (S - sequentielle Datei, R
- Random-Datei) und der Modus (R - Read, Lesen, W - Write, Schreiben, A - Append, Anhängen) angegeben werden.

Wir wollen uns einige Beispiele anschauen. Nehmen wir an, Sie möchten auf Kassette eine Datei mit dem Namen DATEI1 erzeugen. Andere Dateien sind nicht geöffnet, deshalb können Sie der Datei einfach die Nummer 1 geben. Der OPEN-Befehl sieht dann folgendermaßen aus:

```
OPEN 1,1,1,"DATEI1"
```

In diesem Fall ist also die erste Eins die von Ihnen gewählte Dateinummer. Die zweite Eins ist die Gerätenummer, hier der Kassettenrecorder. Die dritte Eins zeigt an, daß Sie eine Datei zum Schreiben öffnen.

Mit PRINT#1 können Sie nun in die Datei schreiben und sie dann mit CLOSE 1 schließen.

Später möchten Sie die Datei wieder öffnen und die gespeicherten Daten lesen. Dazu können Sie fast den gleichen OPEN-Befehl benutzen, mit dem Sie die Datei zum Schreiben geöffnet haben. Nur der dritte Parameter verändert sich. Um eine Kassettendatei zum Lesen zu öffnen, müssen Sie als Sekundäradresse 0 verwenden:

```
OPEN 1,1,0,"DATEI1"
```

Nun können Sie die in der Datei gespeicherten Daten z. B. mit INPUT#1 lesen.

Beim Öffnen einer Diskettendatei werden dann noch Typ und Modus spezifiziert. Außerdem können die Laufwerke 0 oder 1 angesprochen werden. Mit dem folgenden Befehl wird eine sequentielle Datei mit dem Namen DATEI2 im Laufwerk 0 zum Schreiben geöffnet:

```
OPEN 1,8,2,"0:DATEI2,S,W"
```

Wenn Sie nur ein Laufwerk angeschlossen haben, können Sie die Null weglassen. Um dieselbe Datei zum Lesen zu öffnen, tippen Sie:

```
OPEN 1,8,2,"DATEI2,S,R"
```

### Anmerkungen

Der OPEN-Befehl kann im BASIC 2.0 auch dazu dienen, eine Anzahl sehr nützlicher Diskettenbefehle zu geben. Beispielsweise können Sie eine Datei kopieren, umbenennen, löschen oder eine neue Diskette formatieren. In allen diesen Fällen muß die Sekundäradresse 15 verwendet werden. Im folgenden finden Sie eine allgemeine Form, um diese Befehle zu formulieren:

```
OPEN 1,8,15,"BEFEHL"
```

Hier ist 1 die Dateinummer. Sie können natürlich auch eine andere wählen. Wenn einmal eine Dateinummer mit der Sekundäradresse 15 geöffnet ist, können Sie auch die PRINT#-Anweisung benutzen, um Diskettenbefehle zu geben.

In diesem Fall verwenden Sie die folgende Befehlsform:

```
PRINT#1,"BEFEHL"
```

Es folgt nun eine Zusammenfassung der wichtigsten Befehle:

Der COPY-Befehl (siehe auch COPY). Dieser Befehl kopiert eine Datei. Die Kopie wird auf dem gleichen Laufwerk hergestellt wie das Original. Der folgende Befehl zum Beispiel stellt eine Kopie der Datei mit Namen ALT her; die Kopie soll NEU heißen:

```
OPEN 1,8,15,"C:NEU=ALT"
```

Der RENAME-Befehl (siehe auch RENAME). Mit diesem Befehl erhält eine beliebige Diskettendatei einen neuen Namen:

```
OPEN 1,8,15,"R:NEUER NAME=ALTER NAME"
```

Der SCRATCH-Befehl (siehe auch SCRATCH). Dieser Befehl löscht eine Datei auf der Diskette. Wenn Sie diesen Befehl benutzt haben, gibt es keine Möglichkeit, die Datei wiederherzustellen:

```
OPEN 1,8,15,"S:DATEINAME"
```

Der NEW-Befehl (siehe auch HEADER). Dieser Befehl initialisiert eine neue Diskette. Sie müssen diesen Befehl anwenden, bevor der Rechner irgendeine Information auf einer fabrikneuen Diskette speichern soll:

```
OPEN 1,8,15,"N:DISKETTENNAME,ID"
```

ID können zwei beliebige Zeichen sein, die nur dazu dienen, die Diskette zu kennzeichnen.

In allen Beispielen ist jeder Befehl in abgekürzter Form gebraucht worden. Beachten Sie, daß Sie auch mit dem vollen Namen jedes Befehls arbeiten können, z. B.:

```
OPEN 1,8,15,"SCRATCH:DATEINAME"
```

# Operatoren

## Arithmetische Operatoren

Die folgenden arithmetischen Operatoren sind immer Bestandteil eines jeden BASIC:

| Symbol | Beispiel | Funktion |
|---|---|---|
| = | `10 A=B` | Weist einer Variablen einen Wert zu. |
| | `20 LET A=B` | LET kann auch entfallen. |
| + | `10 A=B+4` | Addition. |
| - | `20 A=B-2` | Subtraktion. |
| ↑ | `30 PRINT 6↑2` | Exponentiation; im Beispiel 6 . |
| / | `40 C=A/8` | Division. |
| * | `50 C=B*5` | Multiplikation. |
| = | `10 IF A=B THEN GOTO 1000` | A gleich B. |
| < | `10 IF A<B THEN GOTO 2000` | A ist kleiner als B. |
| > | `10 IF A>B THEN GOTO 3000` | A ist größer als B. |
| <= | `10 IF A<=B THEN GOTO 50` | A ist kleiner oder gleich B. |
| <> | `10 IF A<>B THEN GOTO 900` | A ist ungleich B. |

## Logische Operatoren

Zu den logischen Operatoren gehören AND, OR, NOT und XOR, das allerdings als Verknüpfungsfunktion implementiert ist. Ausführliche Wertetabellen zu den Operatoren finden Sie in Kapitel 3.

### Zeichenkettenoperatoren

Es gibt für Zeichenketten nur das Pluszeichen als Operator.

| Symbol | Beispiel | Funktion |
|---|---|---|
| + | `A$="HEUTE"+"MORGEN"` | Aneinanderhängen von Zeichenketten |

## (BASIC 3.5, 7.0) PAINT

**Abkürzung:** P<SHIFT>A

**Typ:** Grafikbefehl

**Syntax:** PAINT Farbquelle,X,Y,Modus

**Parameterbereich:** Farbnummern
Grafik-Bildschirmkoordinaten
Modus - Bitzahl

Mit PAINT lassen sich umschlossene Flächen in einer bestimmten Farbe ausfüllen. Wenn die Angabe der Farbquelle (0, 1, 2 oder 3) weggelassen wird, wird standardmäßig die Farbe des Vordergrunds (Nummer 1) zum Färben genommen.

Es wird immer die Fläche gefärbt, in der sich die angegebenen Koordinaten X,Y befinden. Die Koordinatenangaben können auch weggelassen werden, dann wird die Fläche, in der sich zur Zeit der Cursor befindet, genommen.

Modus kann 0 oder 1 gesetzt werden und bewirkt dann die Färbung wie die gewählte Farbquelle (0) oder eine Nicht-Hintergrundfarbe (1). Standardmäßig ist hier der Wert 0 voreingestellt.

### Beispiel

Probieren Sie einmal das folgende Programm aus:

```
100 GRAPHIC 1,1
110 COLOR 0,7
120 COLOR 1,8
130 FOR I=100 TO 10 STEP -10
140 : CIRCLE 1,160,100,I
150 NEXT I
160 FOR I=10 TO 100 STEP 20
170 : PAINT 1,160,99+I
180 NEXT I
190 GETKEY T$
```

```
200 GRAPHIC 0,1
210 COLOR 0,12
220 END
```

In Zeile 170 werden die Flächen ausgefüllt. Dabei muß stets gewährleistet sein, daß sich der Cursor in der betreffenden Fläche befindet. Ist seine Position genau auf dem Rand der Fläche, wird nicht die Fläche ausgefüllt, sondern nur die Farbe des Randes geändert.

Die Grafik kann dank der GETKEY-Anweisung in Zeile 190 wieder so lange betrachtet werden, bis eine beliebige Taste gedrückt wird. Danach werden der Grafikmodus und die Hintergrundfarbe zurückgesetzt.

## (BASIC 2.0, 4.0, 3.5, 7.0) PEEK

**Abkürzung:** PE<SHIFT>E

**Typ:** Systemfunktion

**Syntax:** PEEK (Adresse)

**Argumentbereich:** Speicheradressen

**Ergebnisbereich:** Bytezahl

Die PEEK-Funktion liefert den Inhalt einer bestimmten Speicherstelle, die als Dezimalzahl gegeben ist. PEEK kann sowohl im Programm- als auch im Direktmodus verwendet werden. Mit Hilfe dieses Befehls kann der Inhalt von Systemvariablen, Zeigern, Registern usw. untersucht werden. Will man den Inhalt der Speicherstelle verändern, muß man den Befehl POKE (siehe dort) anwenden.

Da Speichergröße und -organisation bei den einzelnen Commodore-Rechnern unterschiedlich ist, gibt es bei der PEEK-Funktion die meisten Probleme bezüglich der Übertragbarkeit von Programmen. Deshalb müssen wir auch hier zwischen den einzelnen Rechnern unterscheiden.

### Beispiele

Bei den benutzerdefinierten Systemfunktion unter dem Stichwort DEF FN ist u. a. auch die Funktion BS aufgeführt, die Bildschirm-Codes liest und in ASCII-Codes umwandelt. Diese Eigenschaft können wir uns zunutze machen, um einen Bildschirmausdruck (Hardcopy) auf einem ASCII-Drucker (z. B. Commodore-Drucker) auszugeben:

```
10000 REM******************************
10010 REM*    BILDSCHIRMAUSDRUCK       *
10020 REM******************************
10030 FOR I=1 TO 255
10040 : POKE 1024+I,I
10050 NEXT I
```

```
10060 SP=40              :REM SPALTENZAHL
10070 ZR$=CHR$(18)       :REM INVERS AN
10080 ZN$=CHR$(146)      :REM INVERS AUS
10090 RT$=CHR$(13)       :REM RETURN-CODE
10100 OPEN 4,4,7         :REM GROSS/KLEIN
10110 REM OPEN 4,4,0     :REM GRAFIK
10120 REM*******************************
10130 REM*   FUNKTIONSDEFINITIONEN      *
10140 REM*******************************
10150 DEF FNBP(I)=PEEK(1023+I)
10160 :                   REM LESE SCHIRM
10170 REM**UMWANDLUNG VON BSC IN ASCII**
10180 REM******  HILFSFUNKTIONEN  ******
10190 DEF FNB1(I)=FNBP(I)
                        * ABS(FNBP(I)<128)
10200 DEF FNB2(I)=64  * ABS(FNBP(I)< 32)
10210 DEF FNB3(I)=32  *
         ABS(FNBP(I)>63 AND FNBP(I)<96)
10220 DEF FNB4(I)=64  *
        ABS(FNBP(I)>96 AND FNBP(I)<129)
10230 DEF FNB5(I)=(FNBP(I)-128)
                        * (FNBP(I)>128)
10240 REM******** BSC ====> ASC ********
10250 DEF FNBS(I)=FNB1(I)+FNB2(I)+FNB3(I)
                          +FNB4(I)+FNB5(I)
10260 REM*******************************
10270 REM*  AUSDRUCK DES BILDSCHIRMS    *
10280 REM*******************************
10290 FOR Z=0 TO 23
10300 : FOR S=1 TO SP
10310 :    IF FNBS(SP*Z+S)>0 THEN PRINT#4,
           CHR$(ABS(FNBS(SP*Z+S)));
10320 :    REM** AUSDRUCK REVERSZEICHEN *
10330 :    IF FNBS(SP*Z+S)<0 THEN A$=
           CHR$(FNBS(ABS(FNBS(SP*Z+S))))
10340 :    IF FNBS(SP*Z+S)<0 THEN
              PRINT#4,ZR$+A$+ZN$;
10350 : NEXT S
10360 : PRINT#4,RT$
10370 NEXT Z
10380 CLOSE 4
10390 END
```

Wenn Sie den Kanal zu Ihrem Drucker öffnen, müssen Sie auf die Sekundäradresse achten. Die Zeilen 10100 und 10110 bieten Ihnen beide Möglichkeiten:

Ausgabe im Großschrift-/Kleinschriftmodus oder Ausgabe im Großschrift-/Grafikmodus. Achten Sie nur darauf, daß immer die nicht benötigte Zeile mit REM deaktiviert wird, sonst erhalten Sie einen ?FILE OPEN ERROR.

In den Zeilen 10030 bis 10050 gibt unser Programm zu Testzwecken den ganzen verfügbaren Zeichensatz auf dem Bildschirm aus. Diese Ausgabe können Sie selbstverständlich weglassen, wenn Sie das Programm beispielsweise als Unterprogramm zum Ausdruck selbsterstellter Bildschirmvorlagen verwenden wollen.

Das Programm funktioniert nur im Textmodus und nicht im hochauflösenden Grafikmodus.

### Beispiel für C64 und C128

PEEK muß eine Adresse zwischen 0 und 65535 adressieren und gibt einen Dezimalwert zwischen 0 und 255 aus.

```
PRINT PEEK(53280)
```

ergibt den Wert des Farbbytes für den Bildschirmrahmen.

Beim C64 ist das die Zahl 254 für Dunkelblau, beim C128 ist es 253 für Hellgrün. Wenn Sie die Farbe mit POKE ändern wollen, betrachten Sie einmal das Beispiel unter diesem Stichwort.

# PEN (BASIC 7.0)

**Abkürzung:** P<SHIFT>E

**Typ:** Lichtstift-Funktion

**Argumentbereich:** ganze Zahlen 0 bis 4

**Ergebnisbereich:** Text-Bildschirmkoordinaten (80-Zeichen-Modus)
Grafik-Bildschirmkoordinaten

**Syntax:** PEN(X)

Die PEN-Funktion ermittelt die aktuelle Lichtstift-Position. Dabei kann im Grafikmodus die X-Koordinate über PEN(0) und die Y-Koordinate über PEN(1) abgefragt werden. PEN(2) und PEN(3) liefern die X- und die Y-Koordinate im 80-Zeichen-Modus, und PEN(4) enthält den Wert 1, wenn sich die Koordinaten seit der letzten Abfrage geändert haben, sonst 0.

Die Werte, die von der PEN-Funktion geliefert werden, sind sehr instabil. Es empfiehlt sich daher stets, über mehrere Werte den Mittelwert zu bilden. Es kann von Vorteil sein, wenn bei der Abfrage der Stiftposition der Bildschirmhintergrund weiß ist.

Wenn kein gültiger Wert erfaßt wird, enthält PEN den Wert 0.

## Beispiel

In unserem Beispielprogramm fragen wir dreimal die Position des Lichtstifts ab, berechnen daraus den Mittelwert und geben an der errechneten Stelle auf dem Grafikbildschirm mit dem CHAR-Befehl einen * aus. Die Anweisung GETKEY gibt uns dabei die Möglichkeit, den Lichtstift in Ruhe zu positionieren. Erst wenn eine beliebige Taste gedrückt wird, werden die Lichtstift-Werte erfaßt.

```
100 REM********************************
110 REM*    LICHTSTIFTABFRAGE          *
120 REM*    AUF GRAFIKSCHIRM           *
130 REM********************************
```

```
140 KH=6:KV=6.499999999
150 DEF FNPX(X)=INT((PEN(0)-60)/KV)
160 DEF FNPY(Y)=INT((PEN(1)-50)/KH)
170 COLOR 0,2
180 GRAPHIC 1,1
190 GETKEY T$
200 DO
210 : DO WHILE PEN(4):REM PEN GEAENDERT?
220 :   FOR I=1 TO 3 :REM MEHRFACHLESEN
230 :     X(I)=FNPX(X):REM LICHTSTIFT
240 :     IF PEN(0)=0 THEN X(I)=0
250 :     Y(I)=FNPY(Y)
260 :     IF PEN(1)=0 THEN Y(I)=0
270 :   NEXT I
280 :   X=(X(1)+X(2)+X(3))/3
290 :   Y=(Y(1)+Y(2)+Y(3))/3
300 :   CHAR 1,X,Y,"*":REM POSITIONSANZEIGE
310 :   GETKEY T$
320 :   SCNCLR
330 : LOOP  :REM LOESCHEN + PEN-ABFRAGE
340 LOOP
350 END
```

# PLAY (BASIC 7.0)

**Abkürzung:** P<SHIFT>L

**Typ:** Sound-Befehl

**Syntax:** PLAY "Stimme Oktave Instrument Lautstärke Filter ein/aus Noten"

**Parameterbereich:** Zeichenketten aus den unten aufgeführten Zeichen

Der Befehl PLAY gibt dem Sound-Chip SID Informationen über Tonhöhe, Ton-Hüllkurve, Lautstärke, Filter und Namen der zu spielenden Noten. Die Werte werden in einer Zeichenkette übergeben. Bei allen Tonbezeichnungen, auch bei den Halbtönen, wurden die in der Musik üblichen Namen vergeben. Die einzige Ausnahme hierbei bildet das H, hier wurde die Bezeichnung B gewählt:

| | |
|---|---|
| **Stimme** | Werte von V1 bis V3 |
| **Oktave** | Werte von O0 bis O6 |
| **Instrument** | T0 - Klavier |
| | T1 - Akkordeon |
| | T2 - Zirkusorgel |
| | T3 - Trommel |
| | T4 - Flöte |
| | T5 - Gitarre |
| | T6 - Cembalo |
| | T7 - Orgel |
| | T8 - Trompete |
| | T9 - Xylophon |
| **Lautstärke** | Werte von U0 (leise) bis U9 (laut) |
| **Filter** | X0 - ausgeschaltet |
| | X1 - eingeschaltet |
| **Noten** | C, D, E, F, G, A, B |

Bei den Noten kann man außerdem noch weiter differenzieren. Es können z. B. Zwischentöne, halbe Noten usw. erzeugt werden, indem folgende Symbole vorangestellt werden. Die eingestellte Tondauer gilt immer so lange, bis eine andere angegeben wird.

| | |
|---|---|
| # | Halbton nach oben |
| $ | Halbton nach unten |
| W | ganze Note |
| H | halbe Note |
| Q | Viertelnote |
| I | Achtelnote |
| S | Sechzehntelnote |
| . | punktierte Note (eineinhalbfache Dauer) |

Für die Länge einer Pause R kann ebenfalls eins von den oben aufgeführten Symbolen vorangestellt werden.

**Beispiel**

Das folgende Lied stellt den Kanon "Froh zu sein, bedarf es wenig" vor, der in der Lautstärke 12 und mit dem Tempo 10 gespielt werden soll:

```
10 VOL 12
20 TEMPO 10
30 PLAY "O5 .C O4 I $B Q A F E G F A G $B A O5 C
        O4 C E F F"
40 END
```

# POINTER (BASIC 7.0)

**Abkürzung:** PO<SHIFT>I

**Typ:** Systemfunktion

**Syntax:** POINTER(Variablenname)

**Argumentbereich:** Zeichenkettenvariablen

**Ergebnisbereich:** Speicheradressen im Bereich der Bank 1

Das Byte an der zurückgegebenen Adresse und das nachfolgende Byte enthalten die Adresse der Speicherstelle in der Bank 1, an der sich der Variablenbereich befindet, der der abgefragten Variablen zugeordnet ist.

### Beispiel

```
100 REM*********************************
110 REM ZUGRIFF AUF VARIABLENSPEICHER
120 REM*********************************
130 A=-1:A%=-255:A$="ABC" :REM TESTDATEN
140 REM***** ALTERNATIVE TESTDATEN *****
150 REM A=1:A%=255:A$="ABCABC":REM DATEN
160 REM A=255:A%=-32767:A$="A":REM DATEN
170 REM A=1.7*10↑35:A%=0:A$="":REM DATEN
180 REM A=-9*10↑-35:A%=9:A$="":REM DATEN
190 REM******** HILFSVARIABLEN *********
200 A1$="":A2$="":A3$=""
210 BANK1:REM FUER RICHTIGE SPEICHERBANK
220 REM*********************************
230 REM       FUNKTIONSDEFINITIONEN
240 REM*********************************
250 DEF FNDK(AD)=PEEK(AD)+256*PEEK(AD+1)
    :REM DK-FUNKTION LIEST ADRESSZEIGER
260 DEF FNLZ(AD)=PEEK(AD-1)
    :REM LZ-FUNKTION LIEST EIN BYTE
270 DEF FNAZ(AD)=PEEK(FNDK(AD+1)+I)
    :REM AZ-FUNKTION HOLT ZEICHENKETTEN-
```

```
        ADRESSE
280 DEF FNGZ(AD)=FNLZ(AD+I)
    :REM GZ-FUNKTION POSITIONIERT AUF
        FOLGEBYTE
290 REM*********************************
300 REM  LESEN DES GLEITKOMMASPEICHERS
310 REM*********************************
320 BANK1:REM FUER RICHTIGE SPEICHERBANK
330 AD=POINTER(A):REM ZEIGER AUF ADRESSE
340 FOR I=1 TO 5
350 : A1$=A1$+STR$(FNGZ(AD))
360 NEXT I: REM I IST ARGUMENT VON FNGZ!
370 REM*********************************
380 REM   LESEN DES INTEGERSPEICHERS
390 REM*********************************
400 BANK1:REM FUER RICHTIGE SPEICHERBANK
410 AD=POINTER(A%):REMZEIGER AUF ADRESSE
420 FOR I=1 TO 2
430 : A2$=A2$+STR$(FNGZ(AD))
440 NEXT I: REM I IST ARGUMENT VON FNGZ!
450 REM*********************************
460 REM LESEN DES ZEICHENKETTENSPEICHERS
470 REM*********************************
480 BANK1:REM FUER RICHTIGE SPEICHERBANK
490 AD=POINTER(A$):REMZEIGER AUF ADRESSE
500 FOR I=0 TO LEN(A$)-1
510 : A3$=A3$+STR$(FNAZ(AD))
520 NEXT I: REM I IST ARGUMENT VON FNAZ!
530 REM*********************************
540 REM AUSGABE DER SPEICHERDARSTELLUNG
550 REM*********************************
560 PRINT"SPEICHERINTERNE DARSTELLUNG ";
570 PRINT"DER INHALTE NICHT ";
580 PRINT"INDIZIERTER VARIABLEN:"
590 PRINTTAB(80)"GLEITKOMMAZAHL INTERN "
    ;A1$
600 PRINT" (DARSTELLUNG INKLUSIVE "
610 PRINT" VORZEICHEN UND         "
620 PRINT" EXPONENTIALDARSTELLUNG)"
630 PRINTTAB(80)"GANZE ZAHL INTERN      "
    ;A2$
640 PRINT" (DARSTELLUNG ALS        "
650 PRINT" BINAER CODIERTE         "
660 PRINT" DEZIMALZAHLEN MIT       "
670 PRINT" DARSTELLUNG NEGATIVER   "
```

```
680 PRINT" ZAHLEN IM ZWEIER-      "
690 PRINT" KOMPLEMENT)            "
700 PRINTTAB(80)"ZEICHENKETTE INTERN   "
    ;A3$
710 PRINT" (DARSTELLUNG ALS FOLGE "
720 PRINT" VON ASCII-CODES)       "
```

Damit die voreingestellten Funktionen problemlos funktionieren und damit die Variablenadressen eindeutig und konstant festgelegt sind, sollten alle Variablen vor dem ersten Funktionsaufruf mit Wertzuweisungen oder mit dem DIM-Befehl initialisiert werden. Außerdem sollte die Reihenfolge: einfache ganze Zahlen, Gleitkommazahlen, Zeichenkettenvariablen, gefolgt von indizierten ganzen Zahlen, Gleitkommazahlen, Zeichenkettenvariablen eingehalten werden.

## (BASIC 2.0, 4.0, 3.5, 7.0) POKE

**Abkürzung:** PO<SHIFT>K

**Typ:** Systembefehl

**Syntax:** `POKE Adresse,Wert`

**Parameterbereich:** Speicheradressen
Bytezahl

Der POKE-Befehl macht das Gegenteil von PEEK (siehe dort), er schreibt nämlich einen Wert in eine bestimmte Speicherstelle. Auch hier muß zwischen den einzelnen Commodore-Rechnern unterschieden werden. Die Adresse, in die ein Wert zwischen 0 und 255 geschrieben werden kann, muß im Bereich von 0 bis 65535 liegen.

### Beispiele

Beim C64 und auch beim C128 liegt der Wert, der die Farbe des Bildschirmrahmens enthält, in der Speicherstelle 53280. Voreingestellt ist beim C64 der Wert 254 und beim C128 der Wert 253, wie Sie mit PEEK leicht erfahren konnten (siehe dort).

Wir wollen nun die Bildschirmrahmen-Farbe ändern und dabei alle Farbmöglichkeiten ausprobieren:

```
FOR I=0 TO 255:POKE53280,I:FOR J=1 TO 300:NEXT J,I
```

schreibt in die Speicherstelle, die die Farbnummer für den Bildschirmrahmen enthält, alle zulässigen Werte der Reihe nach hinein. Nach Beendigung der Schleife zeigt der Bildschirmrahmen die Farbe für die Nummer 255. Um dieselbe Rahmenfarbe wie nach dem Einschalten zu erhalten, können Sie den Befehl

```
POKE 53280,254
```
beim C64

bzw.

```
POKE 53280,253
```
beim C128

eingeben.

Ein ganz anderes Beispiel wollen wir nun für den POKE-Befehl zeigen. Wir haben in den einleitenden Kapiteln schon über Token gesprochen. Zur Erinnerung: Das sind die internen 1 Byte langen Abkürzungen für die BASIC-Schlüsselwörter. Sie sind als Werte ab Hexadezimal 80 oder Dezimal 128 verschlüsselt.

Bei dem umfangreichen Befehlsvorrat des C128 reichte allerdings ein Byte nicht aus, und so wurde bei manchen Schlüsselwörtern ein zweites Byte hinzugenommen.

Unser Beispielprogramm gibt alle Token-Werte zusammen mit dem zugehörigen BASIC-Schlüsselwort aus. Damit das Programm einwandfrei funktioniert, ist es absolut notwendig, daß sich der Rechner im Grundzustand befindet. Lösen Sie also vorsichtshalber ein RESET aus.

```
10 GOTO 1000
99 REM                                  :
100 :REM BITTE ZEILEN 10 UND 99 UNVER-
200 :REM AENDERT ABTIPPEN
210 :REM BITTE DOPPELPUNKT HIER         ↑
270 :REM UNBEDINGT EINGEBEN, DA DAS
280 :REM PROGRAMM SELBSTMODIFIZIEREND
290 :REM IST
1000 REM ******************************
1001 REM * TOKENLISTE   FUER BASIC 7.0 *
1002 REM *                       UND 3.5 *
1009 REM ******************************
1010 :
1011 REM ******************************
1012 REM * FALLS SIE FUER DEN BASIC-    *
1013 REM * SPEICHER EINEN ANDEREN       *
1014 REM * BEREICHSANFANG ALS HEX 1024 *
1015 REM * VERWENDEN, ADDIEREN SIE      *
1016 REM * BITTE DIE DIFFERENZ ZU DER   *
1017 REM * KONSTANTEN SC, DIE DIE       *
1018 REM * POSITION DES ERSTEN ZEICHENS*
1019 REM * IN ZEILE 99 ENTHAELT (REM). *
1020 REM ******************************
1030 SC=7184:REM SPEICHERSTELLE FUER REM
1035 SC=SC+DF     :REM DF<>0 IF SC<>7184
1040 FOR TK=128 TO 254
1050 : REM MEHRFACHBELEGUNG BEI 206
```

```
1051 : IF TK=206 THEN BEGIN
1052 :                   NR=17:GOSUB 6000
1053 :                         GOTO  1100
1054 :                 BEND
1060 : REM MEHRFACHBELEGUNG BEI 254
1061 : IF TK=254 THEN BEGIN
1062 :                   NR=38:GOSUB 6000
1063 :                         GOTO  1100
1064 :                 BEND
1070 : PRINT
1075 : PRINT"TOKEN=";TK
1080 :                 :REM AENDERE
1085 : POKE SC,TK      :REM 1-BYTE-TOKEN
1090 : LIST 99         :REM IN ZEILE 99
1095 :
1100 NEXT TK
1200 NR=0:TK=255:GOSUB6000:REM DRUCKE PI
1300 END
6000 REM ******** ZWEI TOKENBYTES ******
6010 REM TOKEN 206+9 BIS 206+17 FREI
6015 REM 206+18 BIS 206+38 FARBSTEUERUNG
6020 REM AB 206 BZW. 254+38 CBMASC-CODES
6025 REM QUIT, OFF SIND UNIMPLEMENTIERT
6030 FOR B2=1 TO NR
6040 : PRINT:PRINT"TOKEN=";TK;
6045 : IF TK<255 THEN PRINT"+";B2
6050 : POKE SC,TK    :REM AENDERE 1. BYTE
6060 : POKE SC+1,B2 :REM      UND 2. BYTE
6070 : LIST 99       :REM      IN ZEILE 99
6080 NEXT B2
6090 RETURN ***** ZWEI TOKENBYTES ******
```

Der entscheidende POKE-Befehl befindet sich in der Programmzeile 1085. Hier wird das Token-Byte in Zeile 99 geändert. Damit diese Position genau getroffen wird, ist es unbedingt notwendig, daß Sie die ersten zwei Zeilen genauso eingeben, wie wir es vorschlagen, da jede Leerstelle den Wert verändert und das Programm dann nicht mehr funktioniert.

Die Ausgabe des Programms zeigt uns nun die Token für die BASIC-Schlüsselwörter in der Version 7.0. Wir lassen hier die Zeilennummer 99 aus optischen Gründen weg:

```
TOKEN=128 : END
TOKEN=129 : FOR
TOKEN=130 : NEXT
```

```
TOKEN=131 : DATA
TOKEN=132 : INPUT#
TOKEN=133 : INPUT
TOKEN=134 : DIM
TOKEN=135 : READ
TOKEN=136 : LET
TOKEN=137 : GOTO
TOKEN=138 : RUN
TOKEN=139 : IF
TOKEN=140 : RESTORE
TOKEN=141 : GOSUB
TOKEN=142 : RETURN
TOKEN=143 : REM
TOKEN=144 : STOP
TOKEN=145 : ON
TOKEN=146 : WAIT
TOKEN=147 : LOAD
TOKEN=148 : SAVE
TOKEN=149 : VERIFY
TOKEN=150 : DEF
TOKEN=151 : POKE
TOKEN=152 : PRINT#
TOKEN=153 : PRINT
TOKEN=154 : CONT
TOKEN=155 : LIST
TOKEN=156 : CLR
TOKEN=157 : CMD
TOKEN=158 : SYS
TOKEN=159 : OPEN
TOKEN=160 : CLOSE
TOKEN=161 : GET
TOKEN=162 : NEW
TOKEN=163 : TAB(
TOKEN=164 : TO
TOKEN=165 : FN
TOKEN=166 : SPC(
TOKEN=167 : THEN
TOKEN=168 : NOT
TOKEN=169 : STEP
TOKEN=170 : +
TOKEN=171 : -
TOKEN=172 : *
TOKEN=173 : /
TOKEN=174 : ↑
TOKEN=175 : AND
```

```
TOKEN=176 : OR
TOKEN=177 : >
TOKEN=178 : =
TOKEN=179 : <
TOKEN=180 : SGN
TOKEN=181 : INT
TOKEN=182 : ABS
TOKEN=183 : USR
TOKEN=184 : FRE
TOKEN=185 : POS
TOKEN=186 : SQR
TOKEN=187 : RND
TOKEN=188 : LOG
TOKEN=189 : EXP
TOKEN=190 : COS
TOKEN=191 : SIN
TOKEN=192 : TAN
TOKEN=193 : ATN
TOKEN=194 : PEEK
TOKEN=195 : LEN
TOKEN=196 : STR$
TOKEN=197 : VAL
TOKEN=198 : ASC
TOKEN=199 : CHR$
TOKEN=200 : LEFT$
TOKEN=201 : RIGHT$
TOKEN=202 : MID$
TOKEN=203 : GO
TOKEN=204 : RGR
TOKEN=205 : RCLR
TOKEN=206 + 1 : nicht belegt
TOKEN=206 + 2 : POT
TOKEN=206 + 3 : BUMP
TOKEN=206 + 4 : PEN
TOKEN=206 + 5 : RSPPOS
TOKEN=206 + 6 : RSPRITE
TOKEN=206 + 7 : RSPCOLOR
TOKEN=206 + 8 : XOR
TOKEN=206 + 9 : RWINDOW
TOKEN=206 +10 : POINTER
TOKEN=207 : JOY
TOKEN=208 : RDOT
TOKEN=209 : DEC
TOKEN=210 : HEX$
TOKEN=211 : ERR$
```

```
TOKEN=212 : INSTR
TOKEN=213 : ELSE
TOKEN=214 : RESUME
TOKEN=215 : TRAP
TOKEN=216 : TRON
TOKEN=217 : TROFF
TOKEN=218 : SOUND
TOKEN=219 : VOL
TOKEN=220 : AUTO
TOKEN=221 : PUDEF
TOKEN=222 : GRAPHIC
TOKEN=223 : PAINT
TOKEN=224 : CHAR
TOKEN=225 : BOX
TOKEN=226 : CIRCLE
TOKEN=227 : GSHAPE
TOKEN=228 : SSHAPE
TOKEN=229 : DRAW
TOKEN=230 : LOCATE
TOKEN=231 : COLOR
TOKEN=232 : SCNCLR
TOKEN=233 : SCALE
TOKEN=234 : HELP
TOKEN=235 : DO
TOKEN=236 : LOOP
TOKEN=237 : EXIT
TOKEN=238 : DIRECTORY
TOKEN=239 : DSAVE
TOKEN=240 : DLOAD
TOKEN=241 : HEADER
TOKEN=242 : SCRATCH
TOKEN=243 : COLLECT
TOKEN=244 : COPY
TOKEN=245 : RENAME
TOKEN=246 : BACKUP
TOKEN=247 : DELETE
TOKEN=248 : RENUMBER
TOKEN=249 : KEY
TOKEN=250 : MONITOR
TOKEN=251 : USING
TOKEN=252 : UNTIL
TOKEN=253 : WHILE
TOKEN=254 + 1 : nicht belegt
TOKEN=254 + 2 : BANK
TOKEN=254 + 3 : FILTER
```

```
TOKEN=254 + 4 : PLAY
TOKEN=254 + 5 : TEMPO
TOKEN=254 + 6 : MOVSPR
TOKEN=254 + 7 : SPRITE
TOKEN=254 + 8 : SPRCOLOR
TOKEN=254 + 9 : RREG
TOKEN=254 +10 : ENVELOPE
TOKEN=254 +11 : SLEEP
TOKEN=254 +12 : CATALOG
TOKEN=254 +13 : DOPEN
TOKEN=254 +14 : APPEND
TOKEN=254 +15 : DCLOSE
TOKEN=254 +16 : BSAVE
TOKEN=254 +17 : BLOAD
TOKEN=254 +18 : RECORD
TOKEN=254 +19 : CONCAT
TOKEN=254 +20 : DVERIFY
TOKEN=254 +21 : DCLEAR
TOKEN=254 +22 : SPRSAV
TOKEN=254 +23 : COLLISION
TOKEN=254 +24 : BEGIN
TOKEN=254 +25 : BEND
TOKEN=254 +26 : WINDOW
TOKEN=254 +27 : BOOT
TOKEN=254 +28 : WIDTH
TOKEN=254 +29 : SPRDEF
TOKEN=254 +30 : QUIT
TOKEN=254 +31 : STASH
TOKEN=254 +32 : nicht belegt
TOKEN=254 +33 : FETCH
TOKEN=254 +34 : nicht belegt
TOKEN=254 +35 : SWAP
TOKEN=254 +36 : OFF
TOKEN=254 +37 : FAST
TOKEN=254 +38 : SLOW
TOKEN=255 : π
```

Es ist außerordentlich nützlich, die Token seines Rechners zu kennen. Wenn Sie z. B. ein Programm von einem Commodore-Rechner auf ein anderes Modell übertragen wollen, müssen einige Token geändert werden, da selbst gleichlautenden Befehle von Commodore in den einzelnen BASIC-Versionen teilweise verschiedene Token zugeordnet wurden.

Weiterhin ist es möglich, ASCII-Dateien, die von Textverarbeitungsprogrammen oder von anderen Rechnern stammen, mit Hilfe der Tokenliste in lauf-

fähige Commodore-BASIC-Programme umzusetzen.

Das Problem der Programmübernahme wird in Kapitel 11 ausführlich besprochen.

Die Token-Verschlüsselung beim C64 ist bis auf die Befehle, die der C128 zusätzlich hat, vollkommen identisch mit der des C128. Unterschiede in der Token-Zuordnung ergeben sich nur zwischen BASIC 4.0 und BASIC 7.0. Im BASIC 4.0 wurden die Diskettenbefehle anders verschlüsselt als beim C128. Der Vollständigkeit halber geben wir diese Token hier auch an:

**Unterschiedliche Token von BASIC 4.0 zu BASIC 7.0:**

| Token (dez) | BASIC-Schlüsselwort |
|---|---|
| 204 | CONCAT |
| 205 | DOPEN |
| 206 | DCLOSE |
| 207 | RECORD |
| 208 | HEADER |
| 209 | COLLECT |
| 210 | BACKUP |
| 211 | COPY |
| 212 | APPEND |
| 213 | DSAVE |
| 214 | DLOAD |
| 215 | CATALOG |
| 216 | RENAME |
| 217 | SCRATCH |
| 218 | DIRECTORY |

Diese Token müssen bei einer Programmübernahme von BASIC 4.0 nach 7.0 oder umgekehrt umgesetzt werden.

## (BASIC 2.0, 4.0, 3.5, 7.0) POS

**Abkürzung:** keine möglich

**Typ:** Funktion zur Ausgabeformatierung

**Syntax:** `POS(Dummy-Argument)`

**Argumentbereich:** beliebig

**Ergebnisbereich:** ganze Zahlen 0 - 39
bzw. 0 - 79

Die Funktion POS liefert als Ergebnis die aktuelle Cursor-Position auf dem Bildschirm. Das Argument hat dabei keinen Einfluß auf die Funktion. Wenn während eines Programmlaufs kein Cursor angezeigt wird, z. B. bei einer Zeichenkettenmanipulation, dann wird die Position des gerade verarbeiteten Zeichens angezeigt:

```
PRINT "LETZTES ZEICHEN";POS(0)
```

bewirkt die Ausgabe:

```
LETZTES ZEICHEN 15
```

Wenn sich bei POS(0) der Wert 0 ergibt, ist also die Position ganz links am Bildschirmrand gemeint.

### Beispiel

Mit dem folgenden Programm werden die Spalten auf Ihrem Bildschirm durchnumeriert und angezeigt:

```
100 DF$=CHR$(147)        :REM LOESCHEN
110 SM$=CHR$(27)+"M"     :REM ROLLEN AUS
120 SL$=CHR$(27)+"L"     :REM ROLLEN AN
130 PRINT DF$+SM$
140 PRINT "1"
150 PRINT " 2"
```

```
160 FOR I=1 TO 36
170 : PRINT TAB (I) POS(0)+2
180 NEXT I
190 PRINT SL$
200 GETKEY T$
210 END
```

Die Anweisung GETKEY in Zeile 200 ist sehr bequem, da sie die Programmausgabe so lange nicht durch ein störendes READY. verunstaltet, bis eine beliebige Taste gedrückt wird. In den BASIC-Versionen 2.0 und 4.0 ist sie allerdings nicht vorhanden und muß daher durch die etwas längere Anweisung:

```
200 GET T$:IF T$="" THEN 200
```

ersetzt werden.

# (BASIC 7.0) POT

**Abkürzung:** P<SHIFT>O

**Typ:** Drehregler-Funktion

**Syntax:** POT(X)

**Argumentbereich:** ganze Zahlen 1 - 4

**Ergebnisbereich:** Bytezahl

Die Funktion POT ermittelt die aktuellen Werte für die Paddles 1 bis 4. Dabei sind an einem Port jeweils zwei Drehregler gleichzeitig angeschlossen.

Die Werte, die von der POT-Funktion ermittelt werden, bewegen sich von 0, wenn das Rad ganz nach rechts gedreht wird, bis 255, wenn es ganz links anschlägt. Dieser Wert ist um 256 vermehrt, wenn gleichzeitig der Feuerknopf gedrückt wird.

## Beispiel

In unserem Beispielprogramm positionieren wir mit Hilfe der Regler 1 und 2 einen Kreis auf dem Grafikbildschirm. Drehregler 1 steuert dabei die X-Koordinate und Regler 2 die Y-Koordinate des Kreismittelpunkts. Die Werte, die von den Reglern geliefert werden, müssen noch gewichtet werden, damit wir mit unserem Kreismittelpunkt alle Positionen auf dem Schirm erreichen können.

```
100 REM PADDLE-PROGRAMM
110 TRUE=-1:W=0:P(1)=0:P(2)=0
120 GRAPHIC 1,1
130 DO WHILE TRUE
140 : IF POT(1)<>P(1) THEN P(1)=POT(1):W=1
150 : IF POT(2)<>P(2) THEN P(2)=POT(2):W=1
160 : IF W=1 THEN GOSUB 200
170 LOOP
180 GRAPHIC 0,1
190 END
```

```
200 REM KREIS ZEICHNEN
210 SCNCLR
220 CIRCLE 1,P(1)*1.2,P(2)/1.2,30
230 W=0
240 RETURN
```

### Anmerkung

Die Funktion POT kann recht einfach in BASIC 2.0 nachgebildet werden:

```
DEF FNP1(X)=-PEEK(36872)*(X=1)-PEEK(36873)*(X=2)
```

ermöglicht ein Auslesen der Potentiometerwerte für Drehregler 1 und Drehregler 2 am Anschluß 1.

## (BASIC 2.0, 4.0, 3.5, 7.0) PRINT

**Abkürzung:** ?

**Typ:** Ausgabebefehl

**Syntax:**
```
PRINT Argument
PRINT Argument,Argument,...
PRINT Argument;Argument;...
```

**Parameterbereich:** alle Zahlen und Zeichen

Die Anweisung PRINT gibt Daten auf dem Bildschirm aus. Dabei sind Argumente anzugeben, die entweder Konstanten (Zahlen und Zeichenketten), Variablen (numerische und Zeichenkettenvariablen), Ausdrücke (arithmetische, logische und Zeichenkettenausdrücke) grafische Zeichen und Kontrollzeichen sein können. Zeichenkonstante sind in Anführungszeichen zu setzen:

```
PRINT "NANU"
```

Nach jeder PRINT-Anweisung folgt automatisch ein Zeilenvorschub, und eine PRINT-Anweisung ohne Angabe von Argumenten bewirkt die Ausgabe einer Leerzeile.

Zur Steuerung der Ausgabepositionen gibt es unterschiedliche Möglichkeiten:

- Komma (,) oder Semikolon (;)
- die Funktionen TAB und POS
- PRINT USING mit Formatierungskennzeichen (nur in BASIC 3.5 und 7.0)

1. Der Gebrauch des Kommas. Ein Komma zwischen zwei Argumenten hinter einer PRINT-Anweisung bewirkt, daß das zweite Argument ab der nächsten voreingestellten Tabulatorstelle, abhängig von der augenblicklichen Position des Cursors, ausgegeben wird.

   Beim VC20 mit seinem 22-Spalten-Bildschirm gibt es eine Stoppstelle in Spalte 11.

   Bei den Commodore-Rechnern mit 40-Spalten-Bildschirm liegen die voreingestellten Tabulatoren in den Spalten 11, 21 und 31.

Und schließlich bei der 80-Spalten-Darstellung gibt es Stoppstellen in den Spalten 11, 21, 31, 41, 51, 61 und 71.

Beendet ein Komma die Liste der Argumente, so erfolgt die Ausgabe der nächsten PRINT-Anweisung in derselben Zeile, falls die letzte Stoppstelle noch nicht überschritten wurde.

2. Der Gebrauch des Semikolons. Wenn ein Semikolon zwei Zeichenkettenelemente eines PRINT-Befehls trennt, werden diese Elemente auf dem Bildschirm nebeneinander, ohne Zwischenraum, ausgegeben. Endet eine PRINT-Anweisung mit einem Semikolon, so wird jede folgende PRINT-Anweisung ihre Ausgabe dort fortsetzen, wo die vorige Ausgabe aufhörte. Nehmen wir z. B. folgende Zeilen:

```
10 LET N$="HANS"
20 PRINT "HALLO ";
30 PRINT N$
```

Das Semikolon am Ende der Zeile 20 verhindert, daß der Rechner in eine neue Zeile für den folgenden PRINT-Befehl vorrückt. Das Ergebnis dieser Zeilenfolge ist daher:

```
HALLO HANS
```

3. Der Gebrauch der TAB-Funktion. Das TAB-Argument zeigt die Spaltennummer an, hinter der das nächste Ausgabeelement erscheinen soll.

```
PRINT TAB(5)"HIER"
```

bedeutet, daß das H von HIER in Spalte 6 erscheinen wird und die restlichen Zeichen anschließend.

4. Der Gebrauch der SPC-Funktion. Die SPC-Funktion setzt eine bestimmte Anzahl von Leerzeichen in eine zu druckende Zeile. Dabei kann eine Anzahl von 0 bis 255 Leerzeichen angegeben werden. Die folgende PRINT-Anweisung enthält ein Beispiel für SPC:

```
PRINT "X";SPC(15);"Y"
```

In der Ausgabezeile werden X und Y 15 Stellen auseinanderliegen.

5. Das Drucken von Anführungszeichen. Der einzige Weg, um ein Anführungszeichen mit PRINT auszugeben, geht über den ASCII-Zeichencode. Da der ASCII-Code für das Anführungszeichen die Zahl 34 ist, ordnet die folgende Anweisung dieses Zeichen der Variablen A$ zu:

```
A$=CHR$(34)
```

Dann kann A$ in der folgenden Art und Weise benutzt werden, um Anführungszeichen auf dem Bildschirm auszugeben:

```
PRINT A$;"DA";A$
```

Diese Anweisung erzeugt die folgende Ausgabe:

```
"DA"
```

# PRINT# (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** P<SHIFT>R

**Typ:** Ausgabebefehl

**Syntax:** PRINT#Gerätenummer,V1;V2;V3;...

**Parameterbereich:** Gerätenummern
Zahlen und Zeichen

Der PRINT#-Befehl sendet Daten zu einer auf Kassette oder Diskette gespeicherten Datei. Dem Befehl PRINT# muß ein OPEN-Befehl vorausgehen, der die Verbindung zu dem Gerät mit der logischen Gerätenummer herstellt und angibt, ob gelesen oder geschrieben werden soll.

Wenn Sie Daten in eine Datei schreiben, müssen Sie sich immer überlegen, wie Sie eine Angabe von der nächsten Angabe trennen. Ein solches Trennzeichen kann z. B. ein Komma, ein Semikolon oder das RETURN-Zeichen (CHR$(13)) sein. Wenn Sie nämlich später mit Hilfe des INPUT#-Befehls die Datei wieder lesen, müssen die Elemente der Datei durch ein Zeichen getrennt sein, das von INPUT# erkannt wird.

Man erhält automatisch das RETURN-Zeichen als Trennzeichen, wenn man jedes Datenelement mittels einer eigenen PRINT#-Anweisung abschickt:

```
100 PRINT#1,V1
110 PRINT#1,V2
120 PRINT#1,V3
130 PRINT#1,V4
```

Eine andere Möglichkeit ist die, hinter jedes Datenelement ein Komma zu setzen:

```
100 K$=","
110 PRINT#1,V1;K$;V2;K$;V3;K$;V4;K$
```

### Beispiel

Wir wollen zeigen, wie unter Verwendung von PRINT# eine sequentielle Datei gebildet wird. Das Programm erzeugt eine Datei mit dem Namen

ARBEITNEHMER, die mehrere Personalakten einer fiktiven Firma enthält. Über jeden Arbeitnehmer speichert das Programm vier Angaben:

- in der Variablen K$ das Kennzeichen (G oder L), das angibt, ob der Arbeitnehmer Gehalts- oder Lohnempfänger ist;
- in der Variablen N$ den Nachnamen des Arbeitnehmers;
- in der Variablen V$ den Vornamen des Arbeitnehmers;
- in der Variablen G das Entgelt des Arbeitnehmers, als Monatsgehalt, wenn das Kennzeichen G lautet, oder als Stundenlohn, wenn das Kennzeichen L ist.

Diese Personaldaten entnimmt das Programm den DATA-Anweisungen, die am Ende der Programmliste stehen. Ein Datenelement nach dem anderen wird aus den DATA-Zeilen gelesen und in der Datei gespeichert.

```
100 OPEN 1,8,2,"@:ARBEITNEHMER,S,W"
110 READ Z
120 PRINT#1,Z
130 FOR I=1 TO Z
140 : READ K$,N$,V$,G
150 : PRINT#1,K$
160 : PRINT#1,N$
170 : PRINT#1,V$
180 : PRINT#1,G
190 NEXT I
200 CLOSE 1
210 END
220 DATA 8
230 DATA G,ADAM,LUDWIG,1900
240 DATA L,DUDEN,ROSA,8.95
250 DATA L,GRAU,DETLEF,18.75
260 DATA G,IGEL,MARTA,2600
270 DATA L,RABE,BERT,7.75
280 DATA L,SAUER,FRED,21.29
290 DATA G,SOMMER,CLARISSA,3000
300 DATA L,TAMM,DIETER,7.25
```

In BASIC 7.0 wird in Zeile 100 der Doppelpunkt nach dem Klammeraffen weggelassen. Zeile 100 öffnet eine sequentielle Diskettendatei zum Schreiben. Dann wird der erste DATA-Wert aus Zeile 220 gelesen, der angibt, wie viele Personalakten in der Datei gespeichert sind. Diese Angabe erleichtert später das Lesen der Datei (siehe auch INPUT#). Wenn eine solche Angabe fehlt und

man nicht genau weiß, wie viele Daten in der Datei sind, muß man das Ende der Datei (EOF - End Of File) durch ein vereinbartes Zeichen kenntlich machen, das beim Lesen dann nach jeder eingelesenen Angabe abgefragt wird.

Nachdem wir nun wissen, daß sich 8 Personalakten in der Datei befinden, wird in einer FOR-Schleife in den Zeilen 130 bis 190 genauso oft der Lese- und Schreibvorgang ausgeführt. Da jedes Datenelement einzeln durch seine eigene PRINT#-Anweisung gesendet wird, ist sichergestellt, daß jeder Angabe ein RETURN-Zeichen folgt. Wenn wir diese Datei lesen, kann die INPUT#- Anweisung zwischen den einzelnen Elementen unterscheiden.

Starten Sie das Programm mit RUN, und speichern Sie es ab. Dann können Sie im Inhaltsverzeichnis feststellen, daß sich dort außer den Programmdateien, die auf Ihrer Diskette sind, nun auch eine sequentielle Datei befindet:

```
1     "ARBEITNEHMER"     SEQ
```

Unser Beispielprogramm unter dem Stichwort INPUT# zeigt Ihnen, wie man die Daten aus einer solchen sequentiellen Datei wieder liest.

Bitte beachten Sie: Bei der Eingabe von PRINT# kann die Abkürzung ? für PRINT nicht verwendet werden, da sonst ein Syntax Error auftritt. Das Nummernzeichen # muß außerdem ohne Zwischenraum nach dem PRINT eingegeben werden. Auch ein nachträgliches Löschen des Leerraums führt zu einem Fehler. In diesem Fall sollte besser der vollständige Befehl neu eingegeben werden.

## (BASIC 3.5, 7.0) PRINT USING

**Abkürzung:** `? US<SHIFT>I`

**Typ:** Ausgabebefehl

**Syntax:**
```
PRINT USING "Format";Variable

PRINT USING Zeichenkettenvariable;
Variable
```

**Parameterbereich:** Zeichenkettenvariablen

Die Anweisung PRINT USING dient zur Steuerung des Ausgabeformats von numerischen Werten und von Zeichenketten.

Zur Formatierung numerischer Ausgaben können eine Reihe von Sonderzeichen verwendet werden. Diese werden in Anführungszeichen oder als Variable nach dem Schlüsselwort USING angegeben. Nach dieser Zeichenkette folgt ein Semikolon und danach der Wert, der im angegebenen Format ausgegeben werden soll.

Beim Aufbau des Formatstrings können folgende Zeichen verwendet werden:

\# Das Nummernzeichen zeigt die Ausgabe eines numerischen Zeichens an. Für jedes Zeichen muß jeweils ein Nummernzeichen stehen.

Der Dezimalpunkt wird verwendet, um die Position des Dezimalpunktes bei der Ausgabe anzuzeigen. Soll z. B. ein numerischer Wert mit 3 Stellen vor und 2 Stellen nach dem Dezimalpunkt ausgegeben werden, so ist der Formatierungsstring wie in Zeile 120 des nachfolgenden Beispielprogramms zu codieren. Die Ausgabe wird automatisch den Positionen nach dem Dezimalpunkt entsprechend gerundet. Bitte beachten Sie auch Zeile 150.

Leerzeichen nach dem letzten Nummernzeichen im Formatierungsstring werden als Leerzeichen ausgegeben. Auf diese Weise kann man Trennungen bei mehreren Ausdrücken in der Ausgabeliste erhalten (vgl. Zeile 160).

+ Ein Pluszeichen zu Beginn oder Ende des Ausgabeformates bewirkt, daß das Vorzeichen des Wertes als Plus- bzw. Minuszeichen entsprechend ausgegeben wird (vgl. Zeilen 180, 220, 280).

– Ein Minuszeichen am Ende eines Formates wird bei negativen Werten als Minus und bei positiven Werten als Leerstelle ausgegeben (vgl. Zeile 320).

** Ein Doppelstern zu Beginn des Formatierungsstrings füllt alle nicht belegten führenden Stellen mit einem Stern auf. Da diese Funktion dazu dient, bei numerischen Ausgaben Manipulationen zu verhindern, wird dieses Format als "Scheckschutzformat" und das Formatierungssymbol als "Scheckschutzzeichen" bezeichnet (vgl. Zeile 340).

$$ Zwei Dollarzeichen zu Beginn des Formats bewirken ein bewegliches (gleitendes) Dollarzeichen. Das Dollarzeichen wird dann unmittelbar vor der Zahl ausgegeben (vgl. Zeile 420).

Desweiteren ist eine Kombination aus Scheckschutzsymbol und gleitendem Dollarzeichen möglich, was durch die Zeichenfolge **$ zu Beginn eines Formatierungsstrings angegeben wird (vgl. Zeile 440).

, Ein Komma links vor dem Dezimalpunkt bewirkt, daß nach jeder dritten Ziffer links vom Dezimalpunkt zur übersichtlicheren Gliederung einer großen Zahl ein Komma eingefügt wird (vgl. Zeile 480). Wird das Komma am Ende des Formats gesetzt, so gilt es als Konstante und wird ausgegeben (vgl. Zeile 500). Wird das Komma in Verbindung mit dem Exponentialformat (s. u.) angegeben, bleibt es ohne Bedeutung.

↑↑↑↑ Vier Exponentialzeichen besagen, daß die auszugebenden Werte in der Exponentialschreibweise darzustellen sind (vgl. Zeilen 520, 540, 560).

= Ein Gleichheitszeichen veranlaßt die linksbündige Ausgabe der Daten.

> Das Größerzeichen druckt die Daten rechtsbündig.

### Beispiel

```
10 REM****** PRINT USING-BEISPIEL ******
30 A=367:B=367.55:C=367.555:D=16387.378
70 A$="VIOLA":B$="VEILCHEN"
90 PRINT"BEISPIELAUSGABE";TAB(28)"USING"
   :PRINT
100 U$="###"
```

```
110 PRINT USING U$;A;:GOSUB 1000
120 U$="###.##"
130 PRINT USING U$;B;:GOSUB 1000
150 PRINT USING U$;C;:GOSUB 1000
160 U$="###.##  "
170 PRINT USING U$;B,C;:GOSUB 1000
180 U$="+###.##"
190 PRINT USING U$;B,-B;:GOSUB 1000
210 PRINT USING U$;-B,B;:GOSUB 1000
220 U$="+###.##DM"
230 PRINT USING U$;B,-B;:GOSUB 1000
240 U$="-###.##DM"
250 PRINT USING U$;B,-C;:GOSUB 1000
260 U$="-###.##"
270 PRINT USING U$;B;:GOSUB 1000
280 U$="###.##+  "
290 PRINT USING U$;B,-B;:GOSUB 1000
300 U$="###.##DM+"
310 PRINT USING U$;B,-B;:GOSUB 1000
320 U$="###.##-  "
330 PRINT USING U$;B,-B;:GOSUB 1000
340 U$="**######.##  "
350 PRINT USING U$;B,-C;:GOSUB 1000
360 U$="**+######.##  "
370 PRINT USING U$;B,-C;:GOSUB 1000
380 U$="+*******.##  "
390 PRINT USING U$;B,-C;:GOSUB 1000
400 U$="-******.##  "
410 PRINT USING U$;B,-C;:GOSUB 1000
420 U$="$$######.##"
430 PRINT USING U$;B,-C;:GOSUB 1000
440 U$="**$######.## "
450 PRINT USING U$;B,-C;:GOSUB 1000
460 U$="+######,.## "
470 PRINT USING U$;B,-C;:GOSUB 1000
480 U$="+#######,.## "
490 PRINT USING U$;D,-D;:GOSUB 1000
500 U$="####.##, "
510 PRINT USING U$;B;:GOSUB 1000
520 U$="###.##↑↑↑↑"
530 PRINT USING U$;B;:GOSUB 1000
540 U$=".####↑↑↑↑"
550 PRINT USING U$;888888;:GOSUB 1000
560 U$="+.##↑↑↑↑"
570 PRINT USING U$;123;:GOSUB 1000
```

```
600 U$="##.##"
610 PRINT USING U$;B;:GOSUB 1000
620 U$=".##"
630 PRINT USING U$;.999;:GOSUB 1000
650 PRINT USING "#";A$,B$;:GOSUB 1000
680 PRINT
690 PRINT".........0.........0.........0"
990 END
1000 REM*** ERLAUTERUNGEN DRUCKEN ******
1010 PRINT TAB(28);U$
1020 RETURN
```

### Anmerkungen

Das Format kann auch vorab in einer Zeichenkettenvariablen abgelegt werden. Das sieht dann beispielsweise so aus:

```
100 F$="+###.##"
110 PRINT USING F$;B$
```

Übersteigt bei numerischer Ausgabe die Anzahl der spezifizierten Zeichen im Ausgabeformat die Zahl 24, so wird die Fehlermeldung "?ILLEGAL FUNCTION CALL" angezeigt.

In BASIC 2.0 und 4.0 kann der PRINT USING-Befehl mit Hilfe der Funktion STR$ und anderen Zeichenkettenfunktionen simuliert werden. Ein entsprechendes Beispiel finden Sie unter dem Stichwort STR$.

Zur Steuerung der Ausgabe gibt es weiterhin die Funktionen TAB, SPC und POS (siehe dort).

# (BASIC 3.5, 7.0) PUDEF

**Abkürzung:** P<SHIFT>U

**Typ:** Ausgabebefehl

**Syntax:** PUDEF "*., "

Mit dem Befehlswort PUDEF kann die Formatangabe für eine PRINT USING-Anweisung nachträglich, wie im weiteren beschrieben, verändert werden. Dabei können maximal 4 Zeichen angegeben werden, es ist aber auch möglich, jeweils nur eines zu verwenden. Der Stern * bewirkt, daß führende Leerstellen durch Sterne ersetzt werden (Scheckschutz). Der Dezimalpunkt ersetzt bei der deutschen Schreibweise die Tausendergliederung, ebenso wie das Dezimalkomma im Deutschen anstelle des Punktes stehen kann. Das vierte Zeichen definiert das Dollarzeichen um.

## Beispiele

```
100 PUDEF "*., "
110 PRINT USING "##,###.##";1234.567
```

Die Ausgabe lautet:

```
*1.234,56
```

```
100 PRINT USING"#$##";1
110 PUDEF"   £"
120 PRINT USING"#$##";1
```

Die Ausgabe lautet:

```
$1
£1
```

```
100 LH$(8)=CHR$(164):REM UNTERSTRICH
110 PUDEF"   "+LH$(8)
120 PRINT USING"##$";1.111
```

Die Ausgabe lautet:

```
_1
```

```
100 ZF$(11)=CHR$(150):REM HELLROT
110 PUDEF"    "+ZF$(11)
120 PRINT USING"#$#";1
```

Das Programm führt zur Ausgabe einer roten 1. Diese Technik kann z. B. zur Hervorhebung negativer Zahlen verwendet werden. Die Farbe muß allerdings hinterher wieder auf normal zurückgeschaltet werden, da auch alle folgenden Ausgaben in Rot erfolgen.

## (BASIC 3.5, 7.0) RCLR

**Abkürzung:** R<SHIFT>C

**Typ:** Grafikfunktion

**Syntax:** RCLR (X)

**Argumentbereich:** ganze Zahlen 0 - 6

**Ergebnisbereich:** Farbnummern

Die Funktion RCLR enthält die aktuellen Farbwerte. Die Zahlen 1 bis 16 entsprechen dabei den bekannten Farben Schwarz bis Hellgrau.

Die Parameter 0 bis 6 der Funktion stehen für folgende Farbquellen:

0 Hintergrundfarbe (40-Zeichen-Modus)
1 Grafik-Zeichenfarbe
2 Grafik-Multicolor-Zeichenfarbe 1
3 Grafik-Multicolor-Zeichenfarbe 2
4 Rahmenfarbe (40-Zeichen-Modus)
5 Schriftfarbe (40- und 80-Zeichen-Modus)
6 Hintergrundfarbe (80-Zeichen-Modus)

Die Funktion RCLR kann dazu verwendet werden, die Färbung von Flächen mit benachbarten Flächen oder der Hintergrundfarbe abzustimmen.

### Beispiel

Im folgenden Beispielprogramm werden Flächen mit zufällig gebildeten Farbparametern gefärbt. Dabei wird RCLR eingesetzt, um zu prüfen, daß die beiden ineinanderliegenden Rechtecke nicht die gleiche Farbe erhalten.

```
100 GRAPHIC 1,1
110 COLOR 0,2
120 BOX 1,100,60,220,140
130 COLOR 1,INT(RND(RND(TI))*13)+3
140 PAINT 1,160,100,1
```

```
150 BOX 1,80,40,240,160
160 F=INT(RND(RND(TI))*13)+3
170 IF RCLR(1)=F THEN 160
180 COLOR 1,F
190 PAINT 1,81,41,1
200 SCNCLR
210 GOTO 120
```

Das Programm läuft endlos weiter, wenn Sie es nicht mit RUN STOP/ RESTORE stoppen.

## (BASIC 3.5, 7.0) RDOT

**Abkürzung:** R<SHIFT>D

**Typ:** Grafikfunktion

**Syntax:** RDOT(X)

**Argumentbereich:** ganze Zahlen 0 - 2

**Ergebnisbereich:** Grafik-Bildschirmkoordinaten
Bitzahlen

Die Funktion RDOT liefert den Farbwert und die Koordinaten des Grafikcursors.

0 X-Koordinate des Grafikcursors
1 Y-Koordinate des Grafikcursors
2 RDOT(2)=0 bedeutet Hintergrundfarbe
RDOT(2)=1 bedeutet Vordergrundfarbe

RDOT läßt sich gut dazu verwenden, um Positionen auf dem Bildschirm aufzusuchen, die man absolut nicht berechnen kann oder möchte und die sich durch eine Verschiebung aus der vorhergehenden Cursorposition ergeben.

### Beispiel

In unserem Beispielprogramm zeichnen wir auf diese Weise ganz einfach Quadrate, die vorgegebene Kreise einschließen:

```
100 COLOR 0,15
110 GRAPHIC 1,1
120 FOR I=20 TO 199 STEP 40
130 : FOR J=20 TO 319 STEP 40
140 :   CIRCLE 1,J,I,20
150 :   DRAW TO J-20,RDOT(1)
160 :   DRAW TO RDOT(0),I+20
170 :   DRAW TO J+20,RDOT(1)
```

```
180 :   DRAW TO RDOT(0),I-20
190 :   DRAW TO J,RDOT(1)
200 : NEXT J
210 NEXT I
220 GETKEY T$
230 GRAPHIC 0,1
240 COLOR 0,13
250 END
```

## (BASIC 2.0, 4.0, 3.5, 7.0) READ

**Abkürzung:** RE<SHIFT>A

**Typ:** Eingabebefehl

**Syntax:** READ Variablenliste

**Parameterbereich:** Zahlen und Zeichenketten

Der READ-Befehl ist eng mit der DATA-Anweisung verbunden. READ liest die in den DATA-Zeilen abgelegten Angaben nacheinander und ordnet jeden eingelesenen Wert einer Variablen zu. Dabei müssen die Datentypen unbedingt übereinstimmen, d. h. Textkonstanten können nur in Zeichenkettenvariablen übertragen werden usw.

```
READ M$,N,I%
```

liest nacheinander eine Zeichenkette, eine Dezimalzahl und eine ganze Zahl.

Tatsächlich behandelt der Rechner die in den DATA-Anweisungen gespeicherten Werte als eine sequentielle Datei. Automatisch wird ein Zeiger auf die aktuelle Angabe in der Datei gesetzt. Anfangs wird der Zeiger auf den ersten Datenwert in der ersten DATA-Anweisung eingestellt. Wenn eine READ-Anweisung dann den laufenden Wert gelesen hat, wird der Zeiger zum nächsten Datenwert vorwärtsbewegt.

Nachdem der READ-Befehl ausgeführt worden ist, führt der Interpreter die nächste auf den READ-Befehl folgende Anweisung aus.

Wenn Sie versuchen, mehr Werte zu lesen als angegeben, wird Ihr Programm mit folgender Fehlermeldung enden:

```
?OUT OF DATA ERROR IN 10
```

wobei 10 in diesem Fall die Zeilennummer der READ-Anweisung angibt.

Sind mehrere READ-Befehle im Programm vorhanden, so muß unbedingt durch die Ablaufsteuerung des Programms sichergestellt werden, daß sie in

der richtigen Reihenfolge zum Zuge kommen bzw. daß die zugehörigen DATA-Werte in der entsprechenden Reihenfolge angeordnet sind. Da alle DATA-Zeilen, unabhängig, wo sie sich im Programm befinden, vom Programm stets im Zusammenhang gesehen werden, werden sie insgesamt der Reihe nach abgearbeitet.

Die Nichtbeachtung dieser Tatsache ist eine häufige Fehlerquelle. Mit dem RESTORE-Befehl, der es in BASIC 7.0 sogar erlaubt, die nächste READ-Aktion gezielt auf eine bestimmte DATA-Zeile zu lenken, kann man dieses Problem wesentlich besser in den Griff bekommen. DATA-Zeilen können für eine READ-Anweisung gruppiert werden, und vor dem ersten Lesevorgang wird mit RESTORE der Zugriff richtig positioniert. Das ist vor allem bei eventuell alternierender Ausführung verschiedener READ-Anweisungen von Bedeutung.

Ein vollständiges Beispiel finden Sie unter dem Stichwort RESTORE.

### Anmerkung

Der Befehl RESTORE (siehe dort) setzt den Datenzeiger wieder auf den allerersten DATA-Wert zurück und ermöglicht somit das erneute Einlesen aller DATA-Werte.

# (BASIC 4.0, 7.0) RECORD#

**Abkürzung:** R<SHIFT>E#

**Typ:** Systembefehl für Ein- und Ausgabe

**Syntax:**
```
RECORD#Dateinummer,Datensatznummer

RECORD#Dateinummer,Datensatznummer,
Bytenummer
```

**Parameterbereich:** Bytezahlen 1 - 255
Speicheradressen
Bytezahlen 1 - 254

Da relative Dateien in beliebiger Reihenfolge gelesen und beschrieben werden können, benötigen wir ein Hilfsmittel, einen Zeiger auf die gewünschte Stelle in der Datei positionieren zu können. Das bewerkstelligt der Befehl RECORD#. Mit ihm können Sie den Zeiger zum Lesen oder Schreiben in einer relativen Datei direkt auf den angegebenen Datensatz oder sogar auf das gewünschte Byte setzen. Die entsprechende Datei muß vorher geöffnet werden:

```
100 DOPEN#1,"DATEI1"
110 RECORD#1,8,12
120 INPUT#1,Z$
```

In der relativen Datei DATEI1 wird nun als erstes das 12. Byte im 8. Satz gelesen.

Die Angabe des dritten Parameters, der Bytenummer, muß nicht unbedingt erfolgen.

Erst wenn bei einer relativen Datei der Dateizeiger mit RECORD# positioniert wurde, kann ein Lesen der Daten mit GET# oder INPUT# durchgeführt werden.

Vor jeder Eingabe in eine relative Datei mit PRINT# sollte nach der Zeigerpositionierung zuerst über die Systemvariable ST die Statusmeldung abgefragt

werden. Es kann nämlich passieren, daß für den zu schreibenden Satz ein neuer Block der Diskette benötigt wird. Dann erfährt man in ST:

```
?50,RECORD NOT PRESENT
```

In diesem Fall kann die Eingabe nicht erfolgen.

## (BASIC 2.0, 4.0, 3.5, 7.0) REM

**Abkürzung:** keine möglich

**Typ:** elementares Befehlswort

**Syntax:** `REM Text`

**Parameterbereich:** Zahlen und Zeichen

Das Befehlswort REM (remark = Bemerkung) erlaubt es Ihnen, Bemerkungen und Erläuterungen in Ihr Programm aufzunehmen. Nach dem Schlüsselwort REM können Sie jede Art von Text einfügen, um das Programm übersichtlich zu gestalten und die Beschreibung wesentlicher Programmkonstruktionen aufzunehmen, z. B.:

```
1000 REM***UNTERPROGRAMM ZUM DRUCKEN***
```

REM-Anweisungen erzeugen keine Tätigkeit; sie werden während des Programmlaufs ignoriert, dennoch nehmen sie Speicherplatz ein.

Qualitativ gute Software enthält ausreichende und gut formulierte Texte als Kommentare. Man spricht daher auch im Bereich der Software-Technologie vom "selbstdokumentierenden Charakter" der Programme.

### Anmerkungen

Grafikzeichen in REM-Anweisungen müssen in Anführungszeichen eingeschlossen sein, da sie sonst als BASIC-Schlüsselwörter interpretiert werden.

REM-Anweisungen können auch am Schluß einer Mehrfachanweisungszeile stehen. Wenn danach noch ein BASIC-Befehl folgt, wird er ignoriert.

# RENAME (BASIC 2.0*, 4.0, 3.5, 7.0)

**Abkürzung:** RE<SHIFT>N

**Typ:** Systembefehl für Diskettenlaufwerk

**Syntax:**
```
RENAME "Alter Name" TO "Neuer Name"

RENAME "Alter Name" TO "Neuer Name",
DLaufwerknummer,UGerätenummer
```

**Parameterbereich:** Dateinamen
Laufwerknummern
Gerätenummern

Mit dem Befehl RENAME kann eine Datei umbenannt werden. Dabei kann die Laufwerknummer weggelassen werden, wenn nur ein Diskettenlaufwerk angeschlossen ist oder wenn sich die Datei auf dem Laufwerk D0 befindet.

Der Vorgang bezieht sich nur auf das Inhaltsverzeichnis, nicht auf den Inhalt der Dateien. Die Dateinamen dürfen keine Jokerzeichen enthalten, und das Umbenennen kann nur auf derselben Diskette stattfinden.

Wenn der neu vergebene Name bereits auf der Diskette existiert, wird die Meldung:

```
?FILE EXISTS
```

ausgegeben.

### Beispiel

```
RENAME "DATEI" TO "PROGRAMM"
```

# (BASIC 3.5, 7.0) RENUMBER

**Abkürzung:** REN<SHIFT>U

**Typ:** Kommando

**Parameterbereich:** Zeilennummern

**Syntax:**

`RENUMBER`
numeriert das gesamte Programm in der Schrittweite 10 durch

`RENUMBER N`
numeriert das Programm ab Zeile N in der Schrittweite 10 durch

`RENUMBER ,S`
numeriert das gesamte Programm in der Schrittweite S durch

`RENUMBER N,S`
numeriert das Programm ab Zeile N in der Schrittweite S durch

Dieser Systembefehl numeriert ein im Rechnerspeicher befindliches Programm neu durch. Dabei werden nicht nur die Zeilennummern zu Beginn einer Zeile, sondern alle Zeilennummern in den BASIC-Anweisungen GOTO, GOSUB, THEN, ELSE, ON...GOTO, ON...GOSUB etc. angeglichen.

Wenn keine Schrittweite angegeben wird, wird automatisch 10 angenommen.

## Beispiel

```
RENUMBER 1000,5
```

numeriert das Programm mit Zeilennummern ab 1000 in Fünferschritten neu durch.

# RESTORE (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** RE<SHIFT>S

**Typ:** sekundäres Schlüsselwort (nur in Verbindung mit READ und DATA)

**Syntax:** RESTORE

RESTORE Zeilennummer (nur C128)

**Parameterbereich:** Zeilennummern (nur C128)

Der RESTORE-Befehl wird im Zusammenhang mit den READ- und DATA-Anweisungen benutzt. Jedesmal, nachdem ein READ-Befehl ausgeführt wurde, zeigt der Zeiger hinterher auf das nächste Datenelement in den DATA-Zeilen.

Manchmal ist es nützlich, diesen Zeiger auf den Anfang des "Datenpools" zurücksetzen zu können. Diese Aufgabe erfüllt der RESTORE-Befehl.

Nur in BASIC 7.0 ist es zudem möglich, mit RESTORE und Angabe einer Zeilennummer den Zeiger zum Lesen der Werte gezielt auf eine bestimmte DATA-Zeile zu setzen. Diese Eigenschaft kann von unschätzbarem Vorteil sein.

### Beispiel

Dieses Beispiel zeigt mit dem RESTORE-Befehl des C128, wie der Zeiger zum Lesen mit READ je nach Benutzereingabe auf die verschiedenen DATA-Zeilen gesetzt wird.

```
105 DF$=CHR$(147)
100 PRINT DF$
110 PRINT"TEST DES BEFEHLS RESTORE"
120 PRINT"BEI EINGABE VON E ENDET DER TEST"
130 READ X              :REM ANZAHL LESEN
140 DO UNTIL A$="E"
150 : RESTORE 520       :REM DATENANFANG
```

```
160 : PRINT:PRINT"DAS SIND ALLE DATEN:"
170 : FOR I=1 TO X
180 :   READ S$
190 :   PRINT S$" ";
200 : NEXT I
210 : PRINT
220 : FOR I=1 TO X
230 :   READ N
240 :   PRINT N"  ";
250 : NEXT I
260 : PRINT:PRINT:PRINT"WAEHLEN SIE BITTE AUS"
270 : INPUT"Z FUER ZEICHENKETTEN, N FUER ZAHLEN"
      ;A$
280 : IF A$="E" THEN END
290 : IF NOT(A$="Z" OR A$="N") THEN 270
300 : IF A$="Z" THEN
      BEGIN
310 :   PRINT DF$
320 :   PRINT"DAS SIND NUR DIE ZEICHENKETTEN:"
330 :   RESTORE 520   :REM ZEICHENKETTEN
340 :   FOR I=1 TO X
350 :     READ S$
360 :     PRINT S$" ";
370 :   NEXT I
380 :   PRINT
390 :   GOTO 490
400 : BEND
410 : PRINT DF$
420 : PRINT"DAS SIND NUR DIE ZAHLEN:"
430 : RESTORE 540            :REM ZAHLEN
440 : FOR I=1 TO X
450 :   READ N
460 :   PRINT N"  ";
470 : NEXT I
480 : PRINT
490 LOOP
500 END
510 DATA 7
520 DATA ROSA,BLAU,GRAU,GELB
530 DATA ROT,WEISS,BRAUN
540 DATA 1,2,3,4,5,6,7
```

Die DATA-Werte in den Zeilen 510 bis 540 bestehen sowohl aus Zeichenketten als auch aus Zahlen. Wenn der Anwender auf Anfrage des Programms

ein Z (für Zeichenkette) eingibt, wird der Zeiger zum Lesen der Daten auf Zeile 520 gesetzt:

```
330 RESTORE 520
```

Anschließend werden wieder alle Daten ausgegeben, daher:

```
150 RESTORE 520
```

Wenn bei der Eingabe aber ein N (für numerische Werte) eingegeben wurde, muß der DATA-Zeiger folgendermaßen gesetzt werden:

```
430 RESTORE 540
```

Der Wert aus der DATA-Zeile 510 wird im ganzen Programm nur einmal gelesen, in Zeile 130:

```
130 READ X                :REM ANZAHL LESEN
```

# (BASIC 3.5, 7.0) RESUME

**Abkürzung:** RES<SHIFT>U

**Typ:** Befehlswort zur Fehlerbehandlung

**Syntax:**
```
RESUME
RESUME Zeilennummer
RESUME NEXT
```

**Parameterbereich:** Zeilennummern

Wenn mit einem TRAP-Befehl die Ausführung eines BASIC-Programms unterbrochen und eine Fehlerauffangroutine eingeleitet worden ist, kann mit RESUME das Fortsetzen der Programmausführung erreicht werden (siehe auch TRAP, ER und ERR$).

Dabei wird, wenn RESUME ohne weitere Angaben verwendet wird, versucht, in derselben Zeile mit der Programmausführung fortzufahren, in der der Fehler aufgetreten ist.

Man kann aber auch hinter RESUME eine Zeilennummer angeben, die festlegt, in welcher Programmzeile die Verarbeitung nach der Fehlerunterbrechung fortgesetzt werden soll.

Und schließlich kann mit RESUME NEXT bei dem Befehl, der hinter dem fehlerhaften folgt, weitergemacht werden.

# RGR (BASIC 3.5, 7.0)

**Abkürzung:** R<SHIFT>G

**Typ:** Grafikfunktion

**Syntax:** RGR(A)

**Argumentbereich:** Zahlen und Zeichen

**Ergebnisbereich:** ganze Zahlen 0 - 5

Die Funktion RGR liefert Werte von 0 bis 5, die den jeweiligen Grafikmodus kennzeichnen, der zuvor mit GRAPHIC eingestellt wurde. Die Variable von RGR ist eine sogenannte Dummy-Variable, d. h. ihr Wert hat keinen Einfluß auf das Ergebnis der Funktion. Dieses Ergebnis gibt folgendermaßen Aufschluß über den eingestellten Grafikmodus:

| | |
|---|---|
| RGR(A)=0 | 40-Zeichen-Textmodus |
| RGR(A)=1 | hochauflösender Grafikmodus |
| RGR(A)=2 | gemischter Grafik-/Textmodus |
| RGR(A)=3 | Mehrfarben-Grafikmodus |
| RGR(A)=4 | gemischter Mehrfarben-Grafik-/Textmodus |
| RGR(A)=5 | 80-Zeichen-Textmodus |

## (BASIC 2.0, 4.0, 3.5, 7.0) RIGHT$

**Abkürzung:** R<SHIFT>I

**Typ:** Zeichenkettenfunktion

**Syntax:**
```
RIGHT$(Zeichenkettenvariable,
-konstante oder -ausdruck,Zahl)
```

**Argumentbereich:** Zeichenketten
Bytezahlen

**Ergebnisbereich:** Zeichenketten

Die Funktion RIGHT$ löst den rechten Teil aus einer Zeichenkette heraus. Demgemäß hat sie zwei Parameter. Der erste Parameter ist eine Zeichenkette, aus der von rechts her so viele Zeichen herauszulösen sind, wie der zweite Parameter angibt.

Die Anweisung

```
PRINT RIGHT$("HANDBUCH",4)
```

wird das Wort BUCH auf dem Bildschirm ausgeben.

Der zweite Parameter kann Werte zwischen 0 und 255 annehmen. Ist er 0, so wird ein Leerstring übertragen, ist er größer als die Länge der Zeichenkette, so wird die Zeichenkette als ganzes übertragen.

### Anmerkungen

Zwei andere Zeichenkettenfunktionen, die das Abgreifen eines Teils einer Zeichenkette erlauben, sind LEFT$ und MID$ (siehe dort).

# RLUM (BASIC 3.5)

**Abkürzung:** nicht in BASIC 7.0

**Typ:** Grafikfunktion

**Syntax:** RLUM(X)

**Argumentbereich:** ganze Zahlen 0 - 4

**Ergebnisbereich:** ganze Zahlen 0 - 7

Die Funktion RLUM gibt die Helligkeit der Farbquelle X an. Diese Helligkeit kann nur in BASIC 3.5 als dritter Parameter im COLOR-Befehl angegeben werden. Dabei können Werte von 0 (dunkel) bis 7 (hell) angenommen werden.

## (BASIC 2.0, 4.0, 3.5, 7.0) RND

**Abkürzung:** R<SHIFT>N

**Typ:** numerische Funktion

**Syntax:** RND(Zahl)

**Argumentbereich:** Gleitkommazahlen

**Ergebnisbereich:** Gleitkommazahlen 0 - 1

RND ergibt eine Zufallszahl zwischen 0 und 1. Tatsächlich sind die von RND erzeugten Zahlen nicht wirklich zufällig, sondern das Ergebnis einer komplexen Berechnung, die der Rechner ausführt. Für die meisten Programme sind sie jedoch hinreichend zufällig.

Das Argument von RND kontrolliert den Startwert der Funktion. Es legt fest, mit welchem Wert die Reihe von Zufallszahlen beginnt. Wenn das Argument 0 ist, benutzt der Rechner für die Berechnung der Zufallszahlen den laufenden Wert von TI, der eingebauten Systemvariablen für die Uhrzeit.

Für einen negativen Parameterwert gibt RND immer eine ganz bestimmte "Zufalls"zahl aus. Sie können zwei identische Reihen von Zufallszahlen erzeugen, wenn Sie jede Reihe mit dem Aufruf von RND und dem negativen Argument beginnen. Z. B. wird das folgende Programm immer wieder die gleiche Reihe von Zufallszahlen erzeugen, ganz gleich, wie oft Sie es starten:

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

Der Aufruf von RND in Zeile 10 mit einem negativen Argument legt einen vorhersagbaren Anfangspunkt für die Serie der Zahlen fest. Das Ergebnis ist, daß der Aufruf von RND innerhalb der FOR-Schleife (in Zeile 30 mit positivem Argument) immer die gleichen fünf Zufallszahlen erzeugt. Wenn Sie nun jedoch Zeile 10 löschen, wird der Anfangspunkt der Reihe unvorhersehbar, und die FOR-Schleife produziert immer wieder, wenn sie ausgeführt wird, eine neue und andere Serie von Zufallszahlen.

In einigen Fällen kann es durchaus sinnvoll sein, immer wieder mit dem gleichen Satz Zufallszahlen zu arbeiten. Bisher haben wir aber nur Zufallszahlen zwischen 0 und 1 erzeugt, manchmal benötigen wir aber auch größere. Das läßt sich einfach mit folgender Formel bewerkstelligen:

```
Z=INT((O-U+1)*RND(1))+U
```

Die ganze Zufallszahl Z nimmt die Werte von U (untere Grenze) bis O (obere Grenze) an.

Für zufällige ganze Zahlen Z im Bereich 1 bis O (einschließlich) verkürzt sich die Formel auf

```
Z=INT(O*RND(1))+1
```

### Beispiel

Wir simulieren 6 Würfel, die wie beim Würfelspiel zufällige Augenzahlen annehmen.

```
100 FOR J=1 TO 3
110 : PRINT TAB(6)"WURFNUMMER";J
120 : FOR I=1 TO 6
130 :   W(I)=INT(RND(RND(TI))*6)+1
140 :   PRINT"WUERFELNUMMER";I;"HAT";W(I);"AUGEN"
150 : NEXT I
160 NEXT J
170 END
```

# (BASIC 7.0) RREG

**Abkürzung:** R<SHIFT>R

**Typ:** Systembefehl

**Syntax:** RREG Variable1,Variable2,Variable3, Variable4

**Parameterbereich:** Variablennamen

Mit der Funktion RREG können den vier numerischen Variablen die Inhalte des

- Akkumulators
- X-Registers
- Y-Registers
- Statusregisters

zugewiesen werden. Diese Werte, die Sie sich beispielsweise mit PRINT ansehen können, sind dieselben in dezimaler Schreibweise wie die Werte, die Sie sehen, wenn Sie den Befehl MONITOR eingeben. Dabei bedeutet:

| | |
|---|---|
| PC | Program Counter (Programmzähler) |
| SR | Status Register |
| AC | Accumulator |
| XR | X-Register |
| YR | Y-Register |
| SP | Stack Pointer (Stapelzeiger) |

# RSPCOLOR (BASIC 7.0)

**Abkürzung:** RSP<SHIFT>C

**Typ:** Sprite-Funktion

**Syntax:** RSPCOLOR(X)

**Argumentbereich:** ganze Zahlen 1 und 2

**Ergebnisbereich:** Farbnummern

Die Funktion RSPCOLOR liefert den aktuellen Farbwert für die beiden Multicolor-Farben, die mit SPRCOLOR für die Sprites eingestellt worden sind. Dabei gibt

```
RSPCOLOR(1)
```

die erste Farbnummer und

```
RSPCOLOR(2)
```

die zweite an.

Die Farbnummern sind auch hier die üblichen Werte von 1 für Schwarz bis 16 für Hellgrau.

# (BASIC 7.0) RSPPOS

**Abkürzung:** R<SHIFT>S

**Typ:** Sprite-Funktion

**Syntax:** RSPPOS(Spritenummer,X)

**Argumentbereich:** Spritenummern
ganze Zahlen 0 - 2

**Ergebnisbereich:** Bildschirmkoordinaten
ganze Zahlen 0 - 15

Die Funktion RSPPOS gibt die aktuelle Position des Sprites mit der angegebenen Nummer an. Dabei kann man über den zweiten Parameter X steuern, ob man die X- oder die Y-Koordinate ansehen möchte:

X=0 - X-Koordinate
X=1 - Y-Koordinate
X=2 - Geschwindigkeit

Wenn man mit X=2 die Geschwindigkeit zu sehen wünscht, wird ein Wert von 0 bis 15 angezeigt.

### Beispiel

```
PRINT RSPPOS(5,1)
```

liefert die Y-Koordinate des Sprites mit der Nummer 5.

# RSPRITE (BASIC 7.0)

**Abkürzung:** RSP<SHIFT>R

**Typ:** Sprite-Funktion

**Syntax:** RSPRITE(Spritenummer,X)

**Argumentbereich:** Spritenummern
ganze Zahlen 0 - 5

**Ergebnisbereich:** Bitzahlen
Farbnummern

Die Funktion RSPRITE gibt Auskunft über verschiedene Eigenschaften des Sprites mit der angegebenen Spritenummer. Die gewünschte Auskunft wird dabei über das Argument X der Funktion angewählt:

| | |
|---|---|
| **X=0** | 1, falls das Sprite aktiv ist<br>0 sonst |
| **X=1** | liefert die Spritefarbe (1-16) |
| **X=2** | 1, falls das Sprite Priorität vor dem Hintergrund hat<br>0 sonst |
| **X=3** | 1, falls das Sprite in X-Richtung vergrößert ist<br>0 sonst |
| **X=4** | 1, falls das Sprite in Y-Richtung vergrößert ist<br>0 sonst |
| **X=5** | 1, falls für das Sprite der Mehrfarbenmodus definiert ist<br>0 sonst |

Diese Eigenschaften wurden zuvor mit dem Befehl SPRITE aktiviert. Falls das abgefragte Sprite bisher nicht definiert wurde, liefern alle Werte bis auf die Farbe eine Null. Die Farbnummer dagegen entspricht der Spritenummer:

```
?RSPRITE(8,1)
```

ergibt 8.

**Beispiel**

```
100 PRINT RSPRITE(5,1)
```

liefert den Farbwert für das Sprite mit der Nummer 5.

# RUN (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** R<SHIFT>U

**Typ:** Befehl zur Programm-Ablaufsteuerung

**Syntax:** RUN
beginnt die Ausführung mit der ersten Programmzeile

RUN N
beginnt die Ausführung mit der Zeile N

**Parameterbereich:** Zeilennummern

Der Befehl RUN veranlaßt den Rechner, mit der Ausführung des gerade im Arbeitsspeicher befindlichen Programms zu beginnen. Die Werte aller Variablen, die in vorhergegangenen Programmläufen ermittelt wurden, werden dabei vorweg gelöscht.

Der Befehl RUN kann auch in ein Programm eingebaut werden. Dann wird das Programm immer wieder von vorne ausgeführt und ist nur noch durch Drücken der Taste RUN STOP zu unterbrechen.

Zum Testen, besonders von längeren Programmen, wird häufig der Befehl RUN mit Angabe einer Zeilennummer verwendet. Damit kann ein für den Moment uninteressanter, vielleicht zeitaufwendiger Programmteil übersprungen werden. Auch hierbei werden alle Variablenwerte wieder auf 0 gesetzt. Das macht auch den Unterschied zu einem Programmstart mit

```
GOTO Zeilennummer
```

oder

```
GOSUB Zeilennummer
```

aus. Denn bei diesen beiden Formen behalten alle Variablen ihre alten Werte.

### Anmerkung

Beim C128 und beim C16 und C116 liegt der RUN-Befehl auf einer Funktionstaste und kann auf Tastendruck hin sofort ausgeführt werden.

# (BASIC 7.0) RWINDOW

**Abkürzung:** R<SHIFT>W

**Typ:** Bildschirm-Funktion

**Syntax:** RWINDOW(X)

**Argumentbereich:** ganze Zahlen 0 - 2

**Ergebnisbereich:** ganze Zahlen 0 - 24
ganze Zahlen 0 - 39 oder 0 - 79
die Zahl 40 oder 80

Mit der Funktion RWINDOW kann man die Parameter des aktuellen Fensters abfragen. Dabei gibt die Funktion mit dem Argument 0 die Anzahl der Zeilen und mit dem Argument 1 die Anzahl der Spalten des Fensters an.

RWINDOW(2) liefert entweder den Wert 40 oder den Wert 80, je nachdem, ob wir uns im 40- oder im 80-Zeichen-Modus befinden.

## Beispiel

```
100 A$="**********************************
**************************************
**************************************
**************************************"
110 FOR I=1 TO INT(LEN(A$)/RWINDOW(2))+1
120 : PRINT MID$(A$,I,RWINDOW(2))
130 NEXT I
140 END
```

# SAVE (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** S<SHIFT>A

**Typ:** Systembefehl zur Ausgabe

**Syntax:**
```
SAVE "DATEINAME"
```
gleichbedeutend mit:
```
SAVE "DATEINAME",1
```
für das Speichern auf Kassette

```
SAVE "DATEINAME",8
```
nur in BASIC 2.0 für das Speichern auf Diskette

**Parameter:** Dateinamen
Gerätenummern

## 1. Speichern auf Kassette

Der SAVE-Befehl speichert das laufende Programm auf Kassette und erzeugt dabei eine neue oder überschreibt eine schon vorhandene BASIC-Programmdatei. Wenn die Gerätenummer 1 weggelassen wird, wird automatisch der Kassettenrecorder angesprochen. Der Dateiname darf bis zu 16 Zeichen lang sein. Wenn Sie den Befehl

```
SAVE "TEST"
```

eingegeben haben und der Kassettenrecorder noch nicht aufnahmebereit ist, dann erscheint die Meldung

```
PRESS RECORD & PLAY ON TAPE
```
(d. h. drücken Sie die RECORD- und PLAY-Taste gleichzeitig)

Der Rechner wird den Recorder automatisch stoppen, wenn der Sicherungsvorgang beendet ist.

## 2. Speichern auf Diskette

In der BASIC-Version 2.0 wird der SAVE-Befehl auch zum Speichern von Programmen auf Diskette verwendet. Bei den anderen Versionen ist dafür der

Diskettenbefehl DSAVE zuständig (siehe dort). Die Gerätenummer 8 für das Diskettenlaufwerk muß dabei angegeben werden:

```
SAVE "TEST",8
```

Solange noch kein Programm unter dem zugewiesenen Namen auf Diskette existiert, wird dieser Befehl eine neue Programmdatei erzeugen und auf Diskette sichern. Existiert schon ein Programm unter diesem Namen, so meldet sich der Rechner nach kurzer Zeit zwar auch wieder mit READY., er hat jedoch Ihr Programm nicht erneut gespeichert.

Zuweilen kann es jedoch Absicht sein, daß Sie den gleichen Namen wählen, nämlich dann, wenn Sie ein Programm überarbeitet haben und unter gleichem Namen wieder abspeichern wollen. Das können Sie dem System durch Voranstellen des Klammeraffen anzeigen:

```
SAVE "@:TEST",8
```

Mit diesem Befehl muß man sehr sorgsam umgehen, da natürlich die Originalversion von TEST dabei verlorengeht.

# SCALE (BASIC 3.5, 7.0)

**Abkürzung:** SC<SHIFT>A

**Typ:** Grafikbefehl

**Syntax:**
```
SCALE 1,XMAX,YMAX
SCALE 0
```

**Parameterbereich:** Bitzahlen
XMAX - Gleitkommazahlen 160 - 1023
YMAX - Gleitkommazahlen 200 - 1023

Mit der Anweisung SCALE können Sie in den Grafikmodi 1, 2, 3 und 4 die Skalierung verändern. SCALE 1 bewirkt, daß die Koordinaten X und Y die voreingestellten Werte 0 bis 1023 annehmen, anstatt 0 bis 159, 0 bis 199 bzw. 0 bis 319.

SCALE 0 schaltet wieder in den normalen Zustand zurück.

Die Angaben für XMAX und YMAX sind fakultativ.

Mit dem SCALE-Befehl läßt sich natürlich die Auflösung des Bildschirms nicht erhöhen. Aber alle Ausgaben auf dem Grafikschirm können so vorgenommen werden, als stünde ein Raster mit 1024 x 1024 Punkten zur Verfügung, von denen 864 x 820 mit SCALE 1 auf dem Bildschirm dargestellt werden können.

Bei der Skalierung von Ausgaben auf dem Schirm wendet der Rechner bei maximalen Auflösungen einen Faktor von 3.2 für die X-Achse und 5.12 für die Y-Achse bei der Abbildung des virtuellen Bildschirmrasters auf den physikalischen Schirm an. Diese Faktoren bzw. die entsprechenden bei einer anderen Skalierung sollten bei allen Zugriffen auf den physikalischen Bildschirm, beispielsweise beim teilweisen Abspeichern eines Bildes mit BSAVE berücksichtigt werden.

Eine Veränderung des Maßstabs ist sehr praktisch, da sie relativ einfach eine Anpassung von Bildschirmausgaben erlaubt. Bei der Darstellung von Statistiken mit Balkendiagrammen kann man die Skalierung dem größten vorhandenen Wert anpassen und kann so ohne große Umrechnungen eine größere Bandbreite von Daten abdecken.

**Beispiel**

```
100 REM*********************************
110 REM*          BALKENGRAFIK           *
120 REM*********************************
130 DIM A(30),S(20):CS$=CHR$(19)+CHR$(19)
140 DEF FNI(I)=
      A(I)+A(I+1)+A(I+2)+A(I+3)+A(I+4)
150 DEF FNMW(N)=
     (FNI(1)+FNI(6)+FNI(11)+FNI(16))/N
160 DEF FNSK(DZ)=40*10↑
      -INT(LOG(ABS(DZ*2))/LOG(10)-1)
170 REM*********************************
180 REM*    AUFBAU EINGABESCHIRM         *
190 REM*********************************
200 ANZAHL=0:N=0:J=0:K=-1:MI=1.7*10↑37:
MB=0:MW=0:MA=-1.7*10↑37:GI=0:KI=0
210 SCNCLR
220 PRINT TAB(14)"BALKENGRAFIK"
230 PRINT
240 PRINT"MAXIMAL 20 EINGABEN"
250 PRINT"DER MITTELWERT ALLER ";
260 PRINT"EINGABEN BILDET DIE X-ACHSE"
270 PRINT"EINGABEN BISHER:"
280 ES$=
    "WEITERE EINGABE ODER E FUER ENDE"
290 FOR I=1 TO 18
300 : REM*********************************
310 : REM*          DATENEINGABE         *
320 : REM*********************************
330 : CHAR 1,1,14,ES$
340 : A$="":WINDOW 34,14,39,15:INPUT A$
350 : IF A$=" " OR A$="" THEN 330
360 : IF A$="E" OR "E"=LEFT$(A$,1)
        THEN F=I-2:I=20:GOTO 530 EXIT
370 : ANZAHL=ANZAHL+1
380 : A(I)=VAL(A$)
390 : IF ABS(A(I))>ABS(MB) THEN MB=A(I)
    : REM BESTIMMUNG MAXIMALER BETRAG
391 : IF A(I)>=MA OR MA=-1.7*10↑37
      THEN MA=A(I):GI=2*I+2*(I>0)
    : REM BESTIMMUNG GROESSTER WERT
392 : IF A(I)<=MI OR MI=1.7*10↑37
      THEN MI=A(I):KI=2*I+2*(I>0)
    : REM BESTIMMUNG KLEINSTER WERT
```

```
400 : MW=FNMW(I):MITTELWERT$=STR$(MW)+"     "
410 : GS$="MITTELWERT="+MITTELWERT$+"  "
420 : NA$="ANZAHL="+STR$(ANZAHL)
430 : MB$="GROESSTER BETRAG="+STR$(MB)
440 : REM*******************************
450 : REM*   ANZEIGE DER EINGABEDATEN  *
460 : REM*******************************
470 : CHAR 1,01,15,GS$+"         "
480 : CHAR 1,05,18,NA$+"         "
490 : CHAR 1,05,21,MA$+"         "
500 : K=K+1
510 : IF K=3 THEN J=J+1:K=0
520 : CHAR 1,11*K+5,7+J,A$
530 NEXT I
540 COLOR 0,1:GRAPHIC 1,1:SCNCLR:
    COLOR 1,4:COLOR 5,3
550 IF MB<>0 THEN FK=FNSK(MB)
560 SCALE 1,1000,1000
570 DRAW 1,0,500 TO 1000,500
580 FOR I=0 TO 24 STEP 2
590 : CHAR 1,39,I,"+"
600 NEXT I
610 DRAW 1,0,500 TO 1000,500
620 REM*******************************
630 REM*    ZEICHNE BALKENSAEULEN     *
640 REM*******************************
650 REM* DIE BALKEN ZEIGEN NEGATIVE   *
660 REM* UND POSITIVE ABWEICHUNGEN VON *
670 REM* DER X-ACHSE AUF              *
680 DRAW 1,985,985 TO 985,0
690 FOR I=0 TO 17
695 : W=A(I+1):IF W=0 AND I>F THEN W=MW
700 : BOX 1,10+I*50,ABS(500-A(I+1)*FK),
                              50+I*50,500
710 : REM* DER MITTELWERT BILDET DIE  *
720 : REM* X-ACHSE                   *
730 NEXT I
740 REM*******************************
750 REM*   BESCHRIFTUNG DER GRAFIK    *
760 REM*******************************
770 CHAR 1,00,24,GS$:CHAR 1,KI,23,STR$(MI):
    CHAR 1,22,24,"(MW)     UNTERHALB"
780 X$="A-B-C-D-E-F-G-H-I-J-K-L-M-N-O"
    +"-P-R-S"+"-"+"MW"
785 Y0=ABS(500+MW*FK):X=INT(Y0/39):
```

```
    IF ABS(INT(MW/39)-X)<26 OR MA=0 OR
      MI=0 THEN DRAW 1,0,Y0 TO 1000,Y0
      : CHAR 1,30,X,"X=0-ACHSE"
790 CHAR 1,01,12,X$
800 CHAR 1,00,00,MB$:CHAR 1,GI,1,STR$(MA)
    : CHAR 1,30,00,"OBERHALB"
810 GETKEY T$
820 COLOR 0,12:COLOR 5,14:COLOR 4,14:
    GRAPHIC 0,1
830 CHAR 1,19,24,"NEUE GRAFIK J/N? ":
    WINDOW 37,24,38,24:
    INPUT A$:PRINT CS$
840 IF A$="J" OR A$=CHR$(202)
    THEN RUN:SCNCLR
    : REM ERNEUTE WERTEEINGABE
850 SCNCLR:END:REM PROGRAMMENDE
```

# SCNCLR (BASIC 3.5, 7.0)

**Abkürzung:** S<SHIFT>C

**Typ:** Bildschirmbefehl

**Syntax:** SCNCLR Modus

**Parameterbereich:** ganze Zahlen 0 - 5

SCNCLR löscht den Bildschirm im Grafik- oder im Textmodus. Der entsprechende Modus muß durch den Parameter hinter der Anweisung spezifiziert werden. SCNCLR wird sehr häufig verwendet. Fast in jedem Grafikprogramm wird vorher oder zwischendurch der Bildschim gelöscht. Besonders in Demonstrationsprogrammen, die endlos weiterlaufen, wird meist zwischen jedem neuen Bild der Schirm gelöscht.

Bitte beachten Sie, daß mit SCNCLR zwar der Bildschirminhalt gelöscht wird, es wird jedoch kein Grundzustand hergestellt. D. h. beispielsweise, daß veränderte Farben für Vorder- und Hintergrund nicht auf ihre Ursprungsfarbe zurückgeschaltet werden. Wenn Sie also nach Ablauf eines Programms Ihren gewohnten graugrünen Schirm wiedersehen wollen, müssen Sie die Farbparameter dafür explizit angeben.

### Beispiel

In unserem Demonstrationsprogramm, das von 100 zufällig erzeugten Anfangspunkten Linien zur Bildschirmmitte zieht, wird der Schirm zwischen den einzelnen Bildern immer gelöscht.

```
100 COLOR 0,8
110 GRAPHIC 1,1
120 COLOR 1,1
130 FOR I=1 TO 100
140 : LOCATE RND(TI)*319,RND(TI)*199
150 : DRAW TO 160,100
160 NEXT I
170 SCNCLR
180 GOTO 120
```

# (BASIC 2.0*, 4.0, 3.5, 7.0) SCRATCH

**Abkürzung:** SC<SHIFT>R

**Typ:** Systembefehl für Ein-/Ausgabe

**Syntax:**

```
SCRATCH "Dateiname"

SCRATCH "Dateiname",DLaufwerknummer,
UGerätenummer
```

**Parameterbereich:** Dateinamen
Laufwerknummern
Gerätenummern

Der Systembefehl SCRATCH löscht die angegebene Datei von der Diskette. Wenn nur ein Laufwerk angeschlossen ist oder wenn die zu löschende Datei auf dem Laufwerk D0 ist, kann die Laufwerknummer weggelassen werden.

Da dieser Befehl mit Vorsicht zu gebrauchen ist, verlangt der Rechner vor der Ausführung im Direktmodus noch einmal nach einer Bestätigung:

```
?ARE YOU SURE
```

Erst wenn Sie mit Y für Yes antworten, wird die Datei gelöscht.

Gelöscht ist die Datei allerdings momentan nur im Inhaltsverzeichnis. Mit Direktzugriffs-Befehlen kann sie dann immer noch gelesen werden. Der Platz im Inhaltsverzeichnis wird von der nächsten neu hinzukommenden Datei belegt. Erst wenn COLLECT eingegeben wird, wird die Datei endgültig gelöscht.

### Anmerkung

In BASIC 2.0 kann der SCRATCH-Befehl nur mit PRINT# über den Kanal 15 geschickt werden (siehe auch OPEN).

# SGN (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** S<SHIFT>G

**Typ:** numerische Funktion

**Syntax:** SGN(numerischer Ausdruck)

**Argumentbereich:** Gleitkommazahlen

**Ergebnisbereich:** -1, 0, 1

Die Funktion SGN ist die Signum-Funktion, d. h. sie gibt das Vorzeichen eines numerischen Argumentes wieder. Sie kann nur die Werte 0, 1 oder -1 annehmen, gemäß folgender Definition:

| | |
|---|---|
| SGN(X)=1 | falls X größer als 0 |
| SGN(X)=0 | falls X=0 |
| SGN(X)=-1 | falls X kleiner als 0 |

**Beispiel**

```
10 INPUT X
20 ON SGN(X)+2 GOTO 30,40,50
30 PRINT "X IST NEGATIV":GOTO 60
40 PRINT "X IST NULL":GOTO 60
50 PRINT "X IST POSITIV":GOTO 60
60 END
```

## (BASIC 2.0, 4.0, 3.5, 7.0) SIN

**Abkürzung:** S<SHIFT>I

**Typ:** trigonometrische Funktion

**Syntax:** SIN(Zahl)

Die Funktion SIN berechnet den Sinus, wobei das Argument im Bogenmaß angegeben wird. Wird das Argument in Grad angegeben, so kann man folgende Formel verwenden:

```
SIN(X*0.00174533)
```

### Beispiel

Das Beispielprogramm zeichnet eine Sinuskurve als Sternchengraph.

```
100 REM********** SINUSKURVE ***********
110 PRINT CHR$(147)
120 FOR I=0 TO 20 STEP .25
130 : X=INT(15+13*SIN(I))
140 : PRINT TAB(X);"***"
150 NEXT I
```

# SLEEP (BASIC 7.0)

**Abkürzung:** S<SHIFT>L

**Typ:** elementares Befehlswort zur Programm-Ablaufsteuerung

**Syntax:** SLEEP X

**Parameterbereich:** Speicheradressen

Mit dem Befehl SLEEP wird die Programmausführung für eine bestimmte Zeitspanne unterbrochen. Der Parameter X, der Werte von 1 bis 65536 annehmen kann, gibt dabei diese Pause in Sekunden an.

### Beispiel

```
SLEEP 25
```

unterbricht die Programmausführung für die Dauer von 25 Sekunden.

Der Befehl kann z. B. sinnvoll eingesetzt werden, wenn man beispielsweise langsam mitverfolgen will, wie eine Grafik erstellt wird.

### Beispiel

```
100 PI=3.1415
110 GRAPHIC 1,1
120 I=0
130 R=100*COS(20*I)
140 S=160+R*COS(I)
150 Z=100-R*SIN(I)
160 FOR A=1 TO 360
170 : I=A*PI/180
180 : R=100*COS(20*I)
190 : SP=160+R*COS(I)
200 : ZE=100-R*SIN(I)
210 : SLEEP 1
```

```
220 : DRAW 1,S,Z TO SP,ZE
230 : S=SP:Z=ZE
240 NEXT A
250 GETKEY T$
260 GRAPHIC 0,1
270 END
```

Bei dieser Grafik können Sie in Ruhe verfolgen, wie sie aufgebaut wird.

# SLOW (BASIC 7.0)

**Abkürzung:** SL<SHIFT>O

**Typ:** Programmierhilfe

**Syntax:** SLOW

SLOW halbiert die Ausführungsgeschwindigkeit der Programme. Das ist besonders bei der Übernahme älterer Programme vom C64 interessant, die zeitabhängige Steuerungen verwenden. Das ist zwar oft auf schlechten Programmierstil zurückzuführen, aber SLOW kann hier weiterhelfen.

Allerdings ist der C128 aufgrund seines umfangreichen BASIC und aufwendigerer Speicherverwaltung im SLOW-Modus etwa ein Drittel langsamer als der C64. Umgekehrt ist der C64 ein Drittel langsamer als der C128 im FAST-Modus. Die Verdopplung der sogenannten MIPS durch die Verdopplung der Prozessor-Taktfrequenz beim C128 gegenüber dem C64 verleitet leicht zur Überschätzung der Geschwindigkeitsdifferenz. MIPS wird nicht umsonst scherzhaft mit Meaningless Indicator of Performance Speed übersetzt.

SLOW bewirkt, daß wieder Ausgaben auf dem 40-Zeichen-Schirm möglich werden, der nach FAST automatisch inaktiviert wird.

Mit folgendem kleinen Meßprogramm kann der Geschwindigkeitsunterschied zwischen FAST und SLOW überprüft werden:

```
100 X=1
110 ON X GOSUB 170,170,190
120 FOR I=1 TO 10000
130 : I=I
140 NEXT I
150 T(X)=TI-T1
160 GOTO 110
170 IF X=1 THEN SLOW:T1=TI:X=2:
          ELSE FAST:T1=TI:X=3
180 RETURN
190 SLOW
200 PRINT"ZEIT FUER SLOW-MODUS: ";T(2)
210 PRINT"ZEIT FUER FAST-MODUS: ";T(3)
220 END
```

## (BASIC 3.5, 7.0) SOUND

**Abkürzung:** S<SHIFT>O

**Typ:** Sound-Befehl

**Syntax:**
```
SOUND Stimme,Frequenz,Dauer,
Klang,Max.Frequenz,Schrittweite,
Wellenform,Impulsbreite
```

**Parameterbereich:** ganze positive Zahlen in den unten angegebenen Bereichen

Mit dem SOUND-Befehl lassen sich 1024 verschiedene Frequenzen mit einer Dauer zwischen einer fünfzigstel Sekunde und knapp 22 Minuten ausgeben. Dabei wird schon der nächste Programmschritt vollzogen, während noch der Ton zu Ende gespielt wird.

Mit dem SOUND-Befehl lassen sich leichter als mit ENVELOPE, FILTER und PLAY besondere Klangeffekte erzielen, und zwar mit den folgenden Parametern:

| | |
|---|---|
| **Stimme** | 1 - Tongenerator 1 erzeugt Töne |
| | 2 - Tongenerator 2 erzeugt Töne |
| | 3 - Tongenerator 1 erzeugt Rauschen |
| **Frequenz** | Werte von 0 bis 1023 (BASIC 3.5) |
| | Werte von 0 bis 65535 (BASIC 7.0) |
| **Dauer** | Werte von 0 bis 65535 (BASIC 3.5) |
| | Werte von 0 bis 32767 (BASIC 7.0) |

Die restlichen Parameter gelten nur für BASIC 7.0:

| | |
|---|---|
| **Klang** | 0 - anschwellend |
| | 1 - abschwellend |
| | 2 - oszillierend |
| **Max.Frequenz** | Werte von 0 bis 65535 |
| **Schrittweite** | Werte von 0 bis 32767 |
| **Wellenform** | 0 - Dreieckschwingung |
| | 1 - Sägezahnschwingung |
| | 2 - Rechteckschwingung |

3 - Rauschen

**Impulsbreite** Werte von 0 bis 4095

Da beim SOUND-Befehl die Noten nicht direkt mit ihrem Namen, sondern über den zweiten Parameter, Frequenz, eingegeben werden müssen, folgt nun eine Tabelle, die die entsprechenden Werte enthält:

| Note | Frequenzwert |
|---|---|
| C | 4297 |
| C# | 4543 |
| D | 4822 |
| D# | 5100 |
| E | 5412 |
| F | 5724 |
| F# | 6069 |
| G | 6429 |
| G# | 6807 |
| A | 7217 |
| A# | 7643 |
| B | 8102 (entspricht H) |
| C | 8578 |

Diese Frequenzwerte müssen für die nächstniedrigere Oktave halbiert und für die nächsthöhere verdoppelt werden.

## (BASIC 2.0, 4.0, 3.5, 7.0) SPC

**Abkürzung:** SP<SHIFT>C

**Typ:** Funktion zur Ausgabeformatierung

**Syntax:** PRINT SPC(X)

**Argumentbereich:** Bytezahlen

Mit SPC kann bei der Ausgabe auf Bildschirm oder auf Drucker die angegebene Anzahl von Druckpositionen übersprungen werden. Dabei darf der Parameter X maximal die Zahl 255 annehmen. Der Ausgabesprung erfolgt relativ zur aktuellen Cursorposition. Die übersprungenen Stellen werden dabei nicht mit Leerzeichen gefüllt, sondern ihr ursprünglicher Inhalt bleibt erhalten.

### Beispiel

```
100 PRINT CHR$(147)
110 FOR I=1 TO 10000
120 : PRINT SPC(2)"*";
130 NEXT I
140 END
```

# SPRCOLOR (BASIC 7.0)

**Abkürzung:** SPR<SHIFT>C

**Typ:** Sprite-Befehl

**Syntax:** SPRCOLOR Mehrfarbenwert1, Mehrfarbenwert2

**Parameterbereich:** Farbnummern

Der Befehl SPRCOLOR setzt die Farben für die Sprites im Mehrfarbenmodus. Die Werte können mit RSPCOLOR (siehe dort) abgefragt werden. Auch hier werden als Parameter wieder die Werte 1 bis 16 für die verfügbaren Farben akzeptiert.

Voraussetzung dafür, daß die Farben zur Geltung kommen, ist das Einschalten des Mehrfarbenmodus für den reinen Grafikschirm mit

```
GRAPHIC 3,1
```

oder

```
GRAPHIC 4,1
```

für den gemischten Grafik-/Textbildschirm.

In diesem Modus können dann die Objekte außer in der Hintergrund- oder Vordergrundfarbe auch noch in den Zusatzfarben angezeigt werden.

# (BASIC 7.0) SPRDEF

**Abkürzung:** SPR<SHIFT>D

**Typ:** Sprite-Befehl

**Syntax:** SPRDEF

Der komfortable Sprite-Editor des C128 stellt Ihnen diesen mächtigen Befehl zur Verfügung. Nach dem Aufruf sehen Sie auf der linken Seite des Bildschirms ein Feld von 24 mal 21 Zeichen, der Größe eines Sprites. Hierdrin können Sie mit dem Cursor die Grafik entwerfen, die Sie fortan als Sprite benutzen wollen.

Als erstes müssen Sie nach Aufforderung die Spritenummer festlegen. Dabei sind Werte von 1 bis 8 zugelassen. Wenn Sie eine Nummer wählen, unter der Sie zuvor schon ein Sprite definiert haben, erscheint dieses Sprite in vergrößerter Form in dem Definitionsfeld und in normaler Größe in der oberen rechten Ecke des Bildschirms. Zum Editieren wird immer die achtfache Vergrößerung genommen. Sie können das Sprite nun verändern oder mit SHIFT/CLR löschen.

Der Cursor im Definitionsfeld erscheint als Kreuz (Pluszeichen), und Sie können ihn mit den normalen Cursortasten steuern. Um nun einen Punkt zu setzen, verwenden Sie die Zifferntaste 2, und um einen Punkt zu löschen, die Taste mit der Ziffer 1.

Wenn Sie die Grafik zu Ihrer Zufriedenheit erstellt haben, verlassen Sie den Editiervorgang durch Drücken der Tasten SHIFT/RETURN und können nun ein weiteres Sprite erstellen.

Durch nochmaliges Drücken von RETURN wird der Sprite-Editor wieder verlassen, und das Sprite kann nun, falls gewünscht, mit SPRSAV (siehe dort) abgespeichert werden.

Die Grafik wird automatisch in ein binäres Zeichenmuster umgesetzt und kann jederzeit im Programm verwendet werden.

**Beispiel**

Unter dem Stichwort SPRITE haben wir mit einfachen Zeichenbefehlen die Sprites 1 und 2 als Rennwagen definiert. Den Wagen mit der Nummer 2 wollen wir jetzt verbessern.

Vergeben Sie also die Spritenummer 2, und füllen Sie das Sprite-Definitionsfeld gemäß dem folgenden Zählschema. Dabei stehen Striche für nicht gesetzte Punkte und Kreuze für gesetzte Punkte:

```
------xxxxxxxxxxxx------
------xxxxxxxxxxxx------
------xxxxxxxxxxxx------
---------xxxxxx---------
----xxxx-xxxxxx-xxxx----
----xxxx--xxxx--xxxx----
----xxxx--xxxx--xxxx----
----xxxxxxxxxxxxxxxx----
----xxxx--xxxx--xxxx----
----xxxx--xxxx--xxxx----
--------xx-xx-xx--------
--------xx----xx--------
--------xx----xx--------
----xxx--xx--xx--xxx----
----xxx--xxxxxx--xxx----
----xxxxxxxxxxxxxxxx----
----xxx---xxxx---xxx----
--------xxxxxxxx--------
---xxxxxxxxxxxxxxxxxx---
---xxxxxxxxxxxxxxxxxx---
-----xxxxxxxxxxxxxx-----
```

Drücken Sie nun SHIFT/RETURN und noch einmal RETURN. Dann löschen wir in dem Programm, das unter SPRITE aufgeführt ist, die Zeile 110, die zuvor den zweiten Rennwagen genauso wie den ersten erzeugt hat:

```
110 <RETURN>
```

Lassen Sie das Programm einmal laufen. Nun, welcher Wagen gefällt Ihnen besser?

# (BASIC 7.0) SPRITE

**Abkürzung:** S<SHIFT>P

**Typ:** Sprite-Befehl

**Syntax:**
```
SPRITE Spritenummer,Sprite ein/aus,
Farbe,Priorität,X-Vergrößerung,
Y-Vergrößerung,Multicolor ein/aus
```

**Parameterbereich:** Spritenummern
Bitzahlen
Farbnummern

Die Anweisung SPRITE ruft ein vorher definiertes Sprite auf. Es erscheint unter Angabe seiner Nummer mit der folgenden Anweisung oben rechts auf dem Bildschirm:

```
SPRITE 5,1
```

Die Spritenummer kann dabei eine Zahl von 1 bis 8 sein. Die anderen Parameter können wahlweise angegeben werden und haben folgende Bedeutung:

| | |
|---|---|
| **Spritenummer** | Nummer des Sprites (1-8) |
| **Sprite ein/aus** | 0 - Sprite wird gelöscht |
| | 1 - Sprite erscheint auf dem Schirm |
| **Farbe** | Farbwert des Sprites (1-16) |
| **Priorität** | 0 - Sprite liegt im Vordergrund |
| | 1 - Sprite liegt im Hintergrund |
| **X-Vergrößerung** | 0 - normal |
| | 1 - Sprite wird in X-Richtung verdoppelt |
| **Y-Vergrößerung** | 0 - normal |
| | 1 - Sprite wird in Y-Richtung verdoppelt |
| **Multicolor ein/aus** | 0 - Mehrfarbenmodus ist ausgeschaltet |
| | 1 - Mehrfarbenmodus ist eingeschaltet |

**Beispiele**

```
SPRITE 4,1,1,0,1,1
```

Das Sprite mit der Nummer 4 erscheint in Schwarz auf dem Bildschirm, und zwar in doppelter Größe im Vordergrund, d. h. es liegt vor allen anderen Bildschirmdaten und wird, wenn man es darüber hinwegbewegt, diese anderen Dinge verdecken.

Nun wollen wir mit Hilfe der Befehle BOX, DRAW, CHAR, COLOR, GRAPHIC, MOVSPR, SSHAPE und SPRITE eine bewegte Grafik auf dem Bildschirm erstellen.

Das Objekt, das wir zunächst in der linken oberen Ecke des Bildschirms darstellen wollen, soll ein Rennwagen sein, der aus einer Karosserie, 4 Reifen und einer Windschutzscheibe besteht. Doch zunächst schalten wir in den Grafikmodus um und verändern auch die Farben:

```
10 COLOR 0,1:GRAPHIC 1,1
20 DRAW 1,18,20 TO 19,20 TO 20,21 TO 24,21 TO 26,20:
   REM WINDSCHUTZSCHEIBE
30 DRAW 1,26,12 TO 25,27 TO 19,27 TO 18,12
40 DRAW 1,18,12 TO 19,11 TO 20,10 TO 21,9 TO 23,09
   TO 24,10 TO 25,11 TO 26,12
50 DRAW 1,19,11 TO 19,08,25,11 TO 25,08:
   REM AUSPUFF
60 BOX 1,28,12,30,15:REM REIFEN RECHTS
61 BOX 1,14,12,16,15:REM REIFEN LINKS
62 BOX 1,27,23,29,25:REM REIFEN RECHTS
63 BOX 1,15,23,17,25:REM REIFEN LINKS
```

Nun können wir das Programm schon laufen lassen, und unser Rennwagen erscheint links oben auf dem Bildschirm. Damit aus ihm ein Sprite wird, verwenden wir die Befehle SSHAPE und SPRSAV:

```
90 SSHAPE A$,14,8,30,32
100 SPRSAV A$,1
110 SPRSAV A$,2
```

Sprite 1 und Sprite 2 enthalten nun das Bitmuster unserer Grafik.

Damit haben wir jetzt die Möglichkeit geschaffen, unsere Rennwagen auf dem Bildschirm an beliebiger Stelle zu positionieren und sogar zu bewegen:

```
120 SPRITE 1,1,8,0,1,1,0:REM SPRITE 1 VERGROESSERT
UND GELB
130 SPRITE 2,1,11,0,1,1,0:REM SPRITE 2 VERGROESSERT
UND ORANGE
140 MOVSPR 1,240,0
150 MOVSPR 2,120,0
```

```
160 MOVSPR 1,180 #5
170 MOVSPR 2,180 #8
```

Damit die Rennwagen auf einer Straße fahren, fügen wir noch eine Mittellinie hinzu:

```
180 BOX 1,150,35,195,40,90,1
190 BOX 1,150,135,195,140,90,1
200 BOX 1,150,215,195,220,90,1
```

Zum Schluß wollen wir noch eine Ziellinie einzeichnen und mit Hilfe des Befehls CHAR das Wort Ziel in den Grafikbildschirm setzen:

```
210 DRAW 1,50,180 TO 300,180
220 DRAW 1,50,180 TO 50,190
230 DRAW 1,300,180 TO 300,190
240 DRAW 1,50,190 TO 300,190
250 CHAR 1,20,23,"ZIEL"
```

Obwohl das Programm nicht aufwendig ist, gibt es noch eine schnellere Art, Sprites zu definieren, und zwar mit SPRDEF (siehe dort).

# SPRSAV (BASIC 7.0)

**Abkürzung:** SPR<SHIFT>S

**Typ:** Sprite-Befehl

**Syntax:**
```
SPRSAV Spritenummer,
Zeichenkettenvariable

SPRSAV Zeichenkettenvariable,
Spritenummer
```

**Parameterbereich:** Spritenummern
Zeichenkettenvariablen

Wenn wir mit Hilfe des Befehls SSHAPE (siehe dort) Grafikinformationen in einer Zeichenkettenvariablen abgelegt haben, können wir diese als Sprite unter Angabe einer Nummer (max. 8) speichern:

```
SPRSAV 3,A$
```

definiert das Sprite Nummer 3.

Eine komfortablere Möglichkeit, Sprites zu erzeugen, bietet jedoch die Anweisung SPRDEF (siehe dort). Dabei wird zunächst die Spritenummer angegeben, und wir können nun den SPRSAV-Befehl mit umgekehrten Parametern verwenden, um Grafikinformationen in einer Zeichenkettenvariablen zu speichern:

```
SPRSAV Z$,4
```

weist das binäre Pixelmuster des Sprites mit der Nummer 4 der Variablen Z$ zu.

## (BASIC 2.0, 4.0, 3.5, 7.0) SQR

**Abkürzung:** S<SHIFT>Q

**Typ:** numerische Funktion

**Syntax:** SQR(Zahl)

**Argumentbereich:** positive Gleitkommazahlen

**Ergebnisbereich:** positive Gleitkommazahlen

Die Funktion SQR errechnet die Quadratwurzel eines nichtnegativen Argumentes. Ist das Argument negativ, so erscheint die Fehlermeldung "ILLEGAL FUNCTION CALL".

Die Wurzelfunktion ist die Umkehrung zum Potenzieren, was in BASIC durch den Operator ↑ implementiert ist. Im Grunde ist die Wurzelfunktion auch nichts anderes. Die folgenden Ausdrücke sind äquivalent:

```
A=SQR(X)

A=X↑.5

A=X↑1/2
```

# SSHAPE/GSHAPE (BASIC 3.5, 7.0)

**Abkürzung:** S<SHIFT>S G<SHIFT>S

**Typ:** Grafikbefehl

**Syntax:**

```
SSHAPE Zeichenkettenvariable,
X1,Y1,X2,Y2

GSHAPE Zeichenkettenvariable,X,Y
```

**Parameterbereich:** Zeichenkettenvariablen
Bildschirmkoordinaten

Eine Grafik, die innerhalb des Rechtecks mit den diagonal gegenüberliegenden Eckpunkten X1,Y1 und X2,Y2 liegt, kann mit der Anweisung SSHAPE einer Zeichenkettenvariablen zugeordnet werden. Diese Grafik kann beispielsweise mit den Grafikbefehlen BOX, CIRCLE, DRAW (siehe dort) usw. erstellt worden sein. Der angegebene Grafikbereich darf allerdings die richtige Größe für ein Sprite (21 x 24 Pixels) nicht überschreiten, sonst wird die Fehlermeldung

```
?STRING TOO LONG ERROR
```

ausgegeben. Eine Zeichenkettenvariable kann überdies auch nicht mehr als 255 Zeichen aufnehmen.

Diese Grafik kann mit Hilfe der Anweisung GSHAPE an beliebiger Stelle auf dem Bildschirm wieder angezeigt werden:

```
GSHAPE A$,20,30
```

wobei die Koordinaten X und Y, in unserem Beispiel 20 und 30, immer den linken oberen Eckpunkt der Grafik markieren.

### Beispiel

Wir definieren in der oberen linken Ecke des Bildschirms eine einfache Grafik, die wir dann mit SSHAPE in einer Variablen speichern und mit

GSHAPE in einer Schleife so oft wieder auf den Bildschirm bringen, bis der ganze Schirm voll ist.

```
100 PRINT CHR$(147)
110 GRAPHIC 1,1
120 DRAW 1,10,0 TO 20,10 TO 10,20 TO 0,10 TO 10,0
130 SSHAPE A$,0,0,20,20
140 FOR I=0 TO 320 STEP 20
150 : FOR J=0 TO 199 STEP 20
160 :   GSHAPE A$,I,J
170 NEXT J,I
180 GETKEY T$
190 GRAPHIC 0,1
200 END
```

Das Muster kann so lange betrachtet werden, bis eine beliebige Taste gedrückt wird.

Ersetzen Sie einmal die Zeile 120 durch die folgende:

```
120 CIRCLE 1,10,10,10
```

und schon erhalten Sie ein anderes Muster. Ihrer Fantasie sind dabei keine Grenzen gesetzt.

Ein schönes Beispiel für moderne Wohnkultur erhalten Sie mit der folgenden Veränderung des Grundprogramms:

```
120 DRAW 1,10,0 TO 20,8 TO 20,20 TO 0,20 TO 0,8 TO 10,0
125 BOX 1,8,8,12,12,1
126 BOX 1,8,14,12,20,1
150 FOR J=0 TO 199 STEP 40
```

Alle anderen Zeilen bleiben so, wie im ersten Beispiel angegeben.

# ST oder STATUS (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** keine möglich

**Typ:** reservierte Variable

**Syntax:** `ST` oder `STATUS`

Die ST-Funktion liefert eine Zahl, die den Zustand des Systems nach einer Eingabe- oder Ausgabeoperation anzeigt. Der Wert von ST gibt Ihnen spezielle Informationen über eine geöffnete Datei nach einer Eingabe- oder Ausgabeoperation. Dabei lassen sich verschiedene Fehler identifizieren, je nachdem, welche Bits gesetzt sind:

| BIT | WERT | DISKETTENDATEI (serieller BUS) | BANDDATEI |
|---|---|---|---|
| 0 | 1 | Schreibvorgang zu langsam | keine Bedeutung |
| 1 | 2 | Lesevorgang zu langsam | keine Bedeutung |
| 2 | 4 | keine Bedeutung | zu kurzer Block |
| 3 | 8 | keine Bedeutung | zu langer Block |
| 4 | 16 | keine Bedeutung | Lesefehler |
| 5 | 32 | keine Bedeutung | Prüfsummenfehler |
| 6 | 64 | Dateiende | Dateiende |
| 7 | 128 | Device not present (Gerät nicht angeschlossen) | Bandende |

Das Dateiende, das mit ST=64 abgefragt werden kann, ist normalerweise sowieso durch eine Dateiende-Markierung gegeben. Unmittelbar nach ST=64 wird zusätzlich das Bit 1 gesetzt, so daß ST dann den Wert 66 enthält.

Wenn ST=0 ist, dann ist kein Fehler aufgetreten.

Wenn bei einer Banddatei ST=4 oder ST=8 geworden ist, so bedeutet das, daß ein Programm irrtümlich als Datei gelesen wurde und nun für den Puffer entweder zu kurz (ST=4) oder zu lang (ST=8) ist.

### Beispiel

Der folgende Test kann nach einer GET#-Operation durchgeführt werden:

```
30 IF ST=64 GOTO 100
```

Nachdem beim Lesen das Dateiende erreicht wird, kann dann in Zeile 100 die weitere Verarbeitung der Daten stattfinden.

### Anmerkungen

Das Statusbyte für Band- und Diskettenoperationen ist in der Speicherstelle 144 zu finden und das für die RS-232-Schnittstelle in Speicherstelle 663. Mit POKE können die Werte dort selbstverständlich verändert werden.

Die Variable ST ist kein reserviertes BASIC-Schlüsselwort, d. h. Variablennamen wie etwa AST sind zulässig. ST wird wie auch TI und TI$ nicht im normalen BASIC-Variablenbereich abgelegt.

# STASH (BASIC 7.0)

**Abkürzung:** S<SHIFT>T

**Typ:** Speicherverwaltungsbefehl

**Syntax:** STASH Anzahl,Adresse1,Adresse2,Bank

**Parameterbereich:** Speicheradressen
Banknummern

Der Befehl STASH ermöglicht den Transport einer gewissen Anzahl von Bytes aus dem Arbeitsspeicher (Bank 1) in einen Speicherbereich der unter Bank angegebenen Nummer. Dabei wird aus dem Arbeitsspeicher der Bereich ab Adresse1 in den Bankbereich ab Adresse2 geschrieben.

Die Adressen können Werte von 0 bis 65535 annehmen, und die Banknummer kann die Zahlen 1 bis 15 annehmen. Standardmäßig ist die Bank 15 implementiert. Mit dem BANK-Befehl kann die aktuelle Bank verändert werden (siehe dort).

Die weggeschriebenen Bereiche können mit dem FETCH-Befehl wieder geladen werden.

Voraussetzung für das Funktionieren dieser beiden Befehle ist allerdings eine Speichererweiterung des C128.

### Beispiel

```
STASH 999,1023,10000,7
```

Der Bildschirmspeicherbereich, der ab Adresse 1023 steht, wird in die Bank 7 ab Adresse 10000 abgespeichert.

## (BASIC 2.0, 4.0, 3.5, 7.0) STOP

**Abkürzung:** ST<SHIFT>O

**Typ:** Programmierhilfe

**Syntax:** STOP

Die Anweisung STOP veranlaßt den Rechner, einen Programmlauf zu stoppen. Wenn die Abarbeitung eines Programms auf diese Anweisung stößt, wird das Programm mit der Meldung

```
BREAK IN xxx
```

abgebrochen, wobei xxx die aktuelle Zeilennummer ist, die die STOP-Anweisung enthält.

Der Unterschied zur END-Anweisung liegt also in der Ausgabe dieser Meldung, so daß man, wenn mehrere STOP-Anweisungen in einem Programm eingebaut sind, sofort weiß, in welcher Programmzeile das Programm abgebrochen wurde.

Wie nach einem Programmende mit dem END-Befehl können auch hier Variablenwerte abgefragt und verändert werden, oder das Programm kann mit CONT fortgesetzt werden.

# STR$ (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** ST<SHIFT>R

**Typ:** Zeichenkettenfunktion

**Syntax:** STR$(numerische Konstante, Variable oder Ausdruck)

**Argumentbereich:** Gleitkommazahlen

**Ergebnisbereich:** Zeichenketten

Die Funktion STR$ liefert eine Zeichenkette, die den im Argument enthaltenen numerischen Wert darstellt. Bei einem negativen Argument ist die erste Stelle des Resultates das Vorzeichen. Das Gegenstück zur Funktion STR$ ist die Funktion VAL.

## Beispiele

Das Konvertieren einer Zahl in eine Zeichenkette erlaubt es Ihnen, die Zeichenkette für die Ausgabe so aufzubereiten, wie Sie sie brauchen. Wir wollen ein Programm erstellen, das beliebig eingegebene Werte für DM und Pfennig immer in die gleiche Form bringt. Dabei soll die Ausgabe, von hinten betrachtet, wie folgt aufgebaut sein:

- Hinter den Wert werden die Zeichen DM gedruckt.
- Die Zahl hat immer zwei Nachkommastellen.
- Davor steht ein Dezimalpunkt.
- Vor dem Punkt steht immer noch mindestens eine Ziffer, und sei es 0.
- Die Ziffern vor dem Dezimalpunkt werden so gegliedert, daß alle 3 Stellen ein Komma die großen Zahlen unterteilt.

So soll beispielsweise die Eingabe der Zahl:

```
1234567.1234
```

zu der Ausgabe:

```
1,234,567.12 DM
```

führen.

*Hinweis:* Dieselbe Wirkung läßt sich in BASIC 3.5 und 7.0 mit dem Befehl PRINT USING erzielen (siehe dort).

```
100 PRINT CHR$(147)
110 PRINT"FORMATIERUNG FUER MARK UND PFENNIG"
120 PRINT:PRINT"EINGABE DER WERTE, 0 FUER ENDE:"
130 PRINT:INPUT DM
140 IF DM=0 THEN END
150 DM=INT(DM*100+.5)/100
160 DM$=STR$(DM)
170 DM$=MID$(DM$,2,LEN(DM$)-1)
180 P=0:C$="":D$=""
190 FOR I=1 TO LEN(DM$)
200 : IF MID$(DM$,I,1)="." THEN P=I
210 NEXT I
220 IF P=0 THEN P=LEN(DM$)+1:DM$=DM$+".0"
230 DM$=DM$+"0"
240 C$=MID$(DM$,P,3)
250 IF P=1 THEN D$="0":GOTO 390
260 DM$=MID$(DM$,1,P-1)
270 IF LEN(DM$)<=3 THEN D$=DM$:GOTO 390
280 T=INT(LEN(DM$)/3)-1
290 L=LEN(DM$)-(T+1)*3
300 IF L<>0 THEN D$=LEFT$(DM$,L)+","
310 L=L+1
320 IF T<>0 THEN
    BEGIN
330 : FOR I=1 TO T
340 :   D$=D$+MID$(DM$,L,3)+","
350 :   L=L+3
360 : NEXT I
370 BEND
380 D$=D$+MID$(DM$,L,3)
390 P$=D$+C$+" DM"
400 PRINT"===>";P$
410 GOTO 130
```

Das Programm muß alle möglichen Eingabeformen berücksichtigen und ist deshalb relativ umfangreich. Der Einsatz der STR$-Funktion erfolgt in Zeile 160. Von da ab wird der zur Zeichenkette umgewandelte Wert für die Ausgabe aufbereitet.

*Achtung:* Da alle Zahlen, die länger als neun Stellen sind, rechnerintern in der wissenschaftlichen Schreibweise gespeichert werden, arbeitet das Programm nur für Zahlen bis zu dieser Länge richtig.

Wir wollen nun als nächstes eine Tabelle der Speicheradressen von 0 bis 65535 erstellen, und zwar in dezimaler, binärer und hexadezimaler Darstellung. Achten Sie darauf, wie wir in diesem Programm die Funktion STR$ einsetzen:

```
100 PRINT CHR$(147)
110 PRINT" DEZ"TAB(13)"BINAER"TAB(30)"HEX"
120 WINDOW 0,3,39,24,1
130 DIM BZ(18):I=17
140 FOR N=0 TO 65535
150 : GOSUB 180
160 NEXT N
170 END
180 I=I-1
190 IF I>1 AND BZ(I)=1 THEN
    BZ(I)=0:GOSUB 180
    : RETURN
200 BZ(I)=1
210 FOR Z=15 TO 0 STEP -1
220 : BI$=RIGHT$(STR$(BZ(Z)),1)+BI$
230 NEXT Z
240 PRINT N;TAB(8) BI$ TAB(30) HEX$(N)
250 BI$="":I=16
260 :       RETURN
270 PRINT CHR$(19)+CHR$(19)
280 END
```

Dieses Programm stellt im übrigen ein schönes Beispiel für einen rekursiven Programmaufruf dar. Die Unterroutine ab Zeile 180 ruft sich selbst in Zeile 190 wieder auf. Dabei werden maximal 16 Stack-Ebenen angelegt, was in der Variablen I kontrolliert wird. Über die Bedeutung des Stapelspeichers in Verbindung mit dem Befehl GOSUB informieren Sie sich bitte unter dem Stichwort GOSUB.

Beachten Sie auch, daß das GOSUB in Zeile 150 logisch mit dem RETURN in Zeile 190 korrespondiert und das GOSUB in Zeile 190 mit dem RETURN in Zeile 260.

Wie Sie sehen, werden die Binärzahlen nicht mit Hilfe eines Divisionsalgorithmus aus den Dezimalzahlen errechnet, was zwar sonst oft üblich, aber wesentlich umständlicher ist. In unserem Programm wird einfach von rechts nach links eine Zeichenkette aus Nullen und Einsen aufgebaut.

Die Ausgabe der Dezimalzahl als Hexadezimalzahl kann in BASIC 3.5 und 7.0 direkt mit der HEX$-Funktion bewerkstelligt werden, in den anderen BASIC-Versionen müßte das jedoch genauso gelöst werden, wie der Aufbau der binären Zeichenkette. Auch das Fenster, das bewirkt, daß die Überschrift erhalten bleibt, wenn die Ausgabe länger als 24 Zeilen wird (Zeile 120), ist ein Komfort, den nur BASIC 7.0 bietet.

Da die Ausgabe des Programms ziemlich lang ist, wollen wir nur die wichtigsten, die ersten 255 Werte auflisten:

| DEZ | BINAER | HEX |
|---|---|---|
| 1 | 0000000000000001 | 0001 |
| 2 | 0000000000000010 | 0002 |
| 3 | 0000000000000011 | 0003 |
| 4 | 0000000000000100 | 0004 |
| 5 | 0000000000000101 | 0005 |
| 6 | 0000000000000110 | 0006 |
| 7 | 0000000000000111 | 0007 |
| 8 | 0000000000001000 | 0008 |
| 9 | 0000000000001001 | 0009 |
| 10 | 0000000000001010 | 000A |
| 11 | 0000000000001011 | 000B |
| 12 | 0000000000001100 | 000C |
| 13 | 0000000000001101 | 000D |
| 14 | 0000000000001110 | 000E |
| 15 | 0000000000001111 | 000F |
| 16 | 0000000000010000 | 0010 |
| 17 | 0000000000010001 | 0011 |
| 18 | 0000000000010010 | 0012 |
| 19 | 0000000000010011 | 0013 |
| 20 | 0000000000010100 | 0014 |
| 21 | 0000000000010101 | 0015 |
| 22 | 0000000000010110 | 0016 |
| 23 | 0000000000010111 | 0017 |
| 24 | 0000000000011000 | 0018 |
| 25 | 0000000000011001 | 0019 |
| 26 | 0000000000011010 | 001A |
| 27 | 0000000000011011 | 001B |
| 28 | 0000000000011100 | 001C |
| 29 | 0000000000011101 | 001D |
| 30 | 0000000000011110 | 001E |
| 31 | 0000000000011111 | 001F |

| DEZ | BINAER | HEX |
|---|---|---|
| 32 | 0000000000100000 | 0020 |
| 33 | 0000000000100001 | 0021 |
| 34 | 0000000000100010 | 0022 |
| 35 | 0000000000100011 | 0023 |
| 36 | 0000000000100100 | 0024 |
| 37 | 0000000000100101 | 0025 |
| 38 | 0000000000100110 | 0026 |
| 39 | 0000000000100111 | 0027 |
| 40 | 0000000000101000 | 0028 |
| 41 | 0000000000101001 | 0029 |
| 42 | 0000000000101010 | 002A |
| 43 | 0000000000101011 | 002B |
| 44 | 0000000000101100 | 002C |
| 45 | 0000000000101101 | 002D |
| 46 | 0000000000101110 | 002E |
| 47 | 0000000000101111 | 002F |
| 48 | 0000000000110000 | 0030 |
| 49 | 0000000000110001 | 0031 |
| 50 | 0000000000110010 | 0032 |
| 51 | 0000000000110011 | 0033 |
| 52 | 0000000000110100 | 0034 |
| 53 | 0000000000110101 | 0035 |
| 54 | 0000000000110110 | 0036 |
| 55 | 0000000000110111 | 0037 |
| 56 | 0000000000111000 | 0038 |
| 57 | 0000000000111001 | 0039 |
| 58 | 0000000000111010 | 003A |
| 59 | 0000000000111011 | 003B |
| 60 | 0000000000111100 | 003C |
| 61 | 0000000000111101 | 003D |
| 62 | 0000000000111110 | 003E |
| 63 | 0000000000111111 | 003F |
| 64 | 0000000001000000 | 0040 |
| 65 | 0000000001000001 | 0041 |
| 66 | 0000000001000010 | 0042 |
| 67 | 0000000001000011 | 0043 |
| 68 | 0000000001000100 | 0044 |
| 69 | 0000000001000101 | 0045 |
| 70 | 0000000001000110 | 0046 |
| 71 | 0000000001000111 | 0047 |
| 72 | 0000000001001000 | 0048 |
| 73 | 0000000001001001 | 0049 |

| DEZ | BINAER | HEX |
|---|---|---|
| 74 | 0000000001001010 | 004A |
| 75 | 0000000001001011 | 004B |
| 76 | 0000000001001100 | 004C |
| 77 | 0000000001001101 | 004D |
| 78 | 0000000001001110 | 004E |
| 79 | 0000000001001111 | 004F |
| 80 | 0000000001010000 | 0050 |
| 81 | 0000000001010001 | 0051 |
| 82 | 0000000001010010 | 0052 |
| 83 | 0000000001010011 | 0053 |
| 84 | 0000000001010100 | 0054 |
| 85 | 0000000001010101 | 0055 |
| 86 | 0000000001010110 | 0056 |
| 87 | 0000000001010111 | 0057 |
| 88 | 0000000001011000 | 0058 |
| 89 | 0000000001011001 | 0059 |
| 90 | 0000000001011010 | 005A |
| 91 | 0000000001011011 | 005B |
| 92 | 0000000001011100 | 005C |
| 93 | 0000000001011101 | 005D |
| 94 | 0000000001011110 | 005E |
| 95 | 0000000001011111 | 005F |
| 96 | 0000000001100000 | 0060 |
| 97 | 0000000001100001 | 0061 |
| 98 | 0000000001100010 | 0062 |
| 99 | 0000000001100011 | 0063 |
| 100 | 0000000001100100 | 0064 |
| 101 | 0000000001100101 | 0065 |
| 102 | 0000000001100110 | 0066 |
| 103 | 0000000001100111 | 0067 |
| 104 | 0000000001101000 | 0068 |
| 105 | 0000000001101001 | 0069 |
| 106 | 0000000001101010 | 006A |
| 107 | 0000000001101011 | 006B |
| 108 | 0000000001101100 | 006C |
| 109 | 0000000001101101 | 006D |
| 110 | 0000000001101110 | 006E |
| 111 | 0000000001101111 | 006F |
| 112 | 0000000001110000 | 0070 |
| 113 | 0000000001110001 | 0071 |
| 114 | 0000000001110010 | 0072 |
| 115 | 0000000001110011 | 0073 |

| DEZ | BINAER | HEX |
|---|---|---|
| 116 | 0000000001110100 | 0074 |
| 117 | 0000000001110101 | 0075 |
| 118 | 0000000001110110 | 0076 |
| 119 | 0000000001110111 | 0077 |
| 120 | 0000000001111000 | 0078 |
| 121 | 0000000001111001 | 0079 |
| 122 | 0000000001111010 | 007A |
| 123 | 0000000001111011 | 007B |
| 124 | 0000000001111100 | 007C |
| 125 | 0000000001111101 | 007D |
| 126 | 0000000001111110 | 007E |
| 127 | 0000000001111111 | 007F |
| 128 | 0000000010000000 | 0080 |
| 129 | 0000000010000001 | 0081 |
| 130 | 0000000010000010 | 0082 |
| 131 | 0000000010000011 | 0083 |
| 132 | 0000000010000100 | 0084 |
| 133 | 0000000010000101 | 0085 |
| 134 | 0000000010000110 | 0086 |
| 135 | 0000000010000111 | 0087 |
| 136 | 0000000010001000 | 0088 |
| 137 | 0000000010001001 | 0089 |
| 138 | 0000000010001010 | 008A |
| 139 | 0000000010001011 | 008B |
| 140 | 0000000010001100 | 008C |
| 141 | 0000000010001101 | 008D |
| 142 | 0000000010001110 | 008E |
| 143 | 0000000010001111 | 008F |
| 144 | 0000000010010000 | 0090 |
| 145 | 0000000010010001 | 0091 |
| 146 | 0000000010010010 | 0092 |
| 147 | 0000000010010011 | 0093 |
| 148 | 0000000010010100 | 0094 |
| 149 | 0000000010010101 | 0095 |
| 150 | 0000000010010110 | 0096 |
| 151 | 0000000010010111 | 0097 |
| 152 | 0000000010011000 | 0098 |
| 153 | 0000000010011001 | 0099 |
| 154 | 0000000010011010 | 009A |
| 155 | 0000000010011011 | 009B |
| 156 | 0000000010011100 | 009C |
| 157 | 0000000010011101 | 009D |

| DEZ | BINAER | HEX |
|---|---|---|
| 158 | 0000000010011110 | 009E |
| 159 | 0000000010011111 | 009F |
| 160 | 0000000010100000 | 00A0 |
| 161 | 0000000010100001 | 00A1 |
| 162 | 0000000010100010 | 00A2 |
| 163 | 0000000010100011 | 00A3 |
| 164 | 0000000010100100 | 00A4 |
| 165 | 0000000010100101 | 00A5 |
| 166 | 0000000010100110 | 00A6 |
| 167 | 0000000010100111 | 00A7 |
| 168 | 0000000010101000 | 00A8 |
| 169 | 0000000010101001 | 00A9 |
| 170 | 0000000010101010 | 00AA |
| 171 | 0000000010101011 | 00AB |
| 172 | 0000000010101100 | 00AC |
| 173 | 0000000010101101 | 00AD |
| 174 | 0000000010101110 | 00AE |
| 175 | 0000000010101111 | 00AF |
| 176 | 0000000010110000 | 00B0 |
| 177 | 0000000010110001 | 00B1 |
| 178 | 0000000010110010 | 00B2 |
| 179 | 0000000010110011 | 00B3 |
| 180 | 0000000010110100 | 00B4 |
| 181 | 0000000010110101 | 00B5 |
| 182 | 0000000010110110 | 00B6 |
| 183 | 0000000010110111 | 00B7 |
| 184 | 0000000010111000 | 00B8 |
| 185 | 0000000010111001 | 00B9 |
| 186 | 0000000010111010 | 00BA |
| 187 | 0000000010111011 | 00BB |
| 188 | 0000000010111100 | 00BC |
| 189 | 0000000010111101 | 00BD |
| 190 | 0000000010111110 | 00BE |
| 191 | 0000000010111111 | 00BF |
| 192 | 0000000011000000 | 00C0 |
| 193 | 0000000011000001 | 00C1 |
| 194 | 0000000011000010 | 00C2 |
| 195 | 0000000011000011 | 00C3 |
| 196 | 0000000011000100 | 00C4 |
| 197 | 0000000011000101 | 00C5 |
| 198 | 0000000011000110 | 00C6 |
| 199 | 0000000011000111 | 00C7 |

| DEZ | BINAER | HEX |
|---|---|---|
| 200 | 0000000011001000 | 00C8 |
| 201 | 0000000011001001 | 00C9 |
| 202 | 0000000011001010 | 00CA |
| 203 | 0000000011001011 | 00CB |
| 204 | 0000000011001100 | 00CC |
| 205 | 0000000011001101 | 00CD |
| 206 | 0000000011001110 | 00CE |
| 207 | 0000000011001111 | 00CF |
| 208 | 0000000011010000 | 00D0 |
| 209 | 0000000011010001 | 00D1 |
| 210 | 0000000011010010 | 00D2 |
| 211 | 0000000011010011 | 00D3 |
| 212 | 0000000011010100 | 00D4 |
| 213 | 0000000011010101 | 00D5 |
| 214 | 0000000011010110 | 00D6 |
| 215 | 0000000011010111 | 00D7 |
| 216 | 0000000011011000 | 00D8 |
| 217 | 0000000011011001 | 00D9 |
| 218 | 0000000011011010 | 00DA |
| 219 | 0000000011011011 | 00DB |
| 220 | 0000000011011100 | 00DC |
| 221 | 0000000011011101 | 00DD |
| 222 | 0000000011011110 | 00DE |
| 223 | 0000000011011111 | 00DF |
| 224 | 0000000011100000 | 00E0 |
| 225 | 0000000011100001 | 00E1 |
| 226 | 0000000011100010 | 00E2 |
| 227 | 0000000011100011 | 00E3 |
| 228 | 0000000011100100 | 00E4 |
| 229 | 0000000011100101 | 00E5 |
| 230 | 0000000011100110 | 00E6 |
| 231 | 0000000011100111 | 00E7 |
| 232 | 0000000011101000 | 00E8 |
| 233 | 0000000011101001 | 00E9 |
| 234 | 0000000011101010 | 00EA |
| 235 | 0000000011101011 | 00EB |
| 236 | 0000000011101100 | 00EC |
| 237 | 0000000011101101 | 00ED |
| 238 | 0000000011101110 | 00EE |
| 239 | 0000000011101111 | 00EF |
| 240 | 0000000011110000 | 00F0 |
| 241 | 0000000011110001 | 00F1 |

| DEZ | BINAER | HEX |
|---|---|---|
| 242 | 0000000011110010 | 00F2 |
| 243 | 0000000011110011 | 00F3 |
| 244 | 0000000011110100 | 00F4 |
| 245 | 0000000011110101 | 00F5 |
| 246 | 0000000011110110 | 00F6 |
| 247 | 0000000011110111 | 00F7 |
| 248 | 0000000011111000 | 00F8 |
| 249 | 0000000011111001 | 00F9 |
| 250 | 0000000011111010 | 00FA |
| 251 | 0000000011111011 | 00FB |
| 252 | 0000000011111100 | 00FC |
| 253 | 0000000011111101 | 00FD |
| 254 | 0000000011111110 | 00FE |
| 255 | 0000000011111111 | 00FF |

# SWAP (BASIC 7.0)

**Abkürzung:** S<SHIFT>W

**Typ:** Speicherverwaltungsbefehl

**Syntax:** SWAP Anzahl,Startadresse,Bank,
Bankadresse

**Parameterbereich:** Speicheradressen
Banknummern

Der SWAP-Befehl tauscht ohne Zwischenspeichern einen Bereich des Arbeitsspeichers mit einem Bereich der angegebenen Speicherbank aus. Dabei gibt Anzahl die Anzahl der auszutauschenden Bytes an (Werte von 0 bis 65535).

Die Startadresse (Werte von 0 bis maximal 65535) bezieht sich dabei auf den Bereich im Arbeitsspeicher (Bank 1), der ausgetauscht werden soll, und die Bankadresse (ebenfalls Werte von 0 bis 65535) auf den Bereich in der Bank, mit der ausgetauscht werden soll. Es können die Bänke 1 bis 15 angesprochen werden, allerdings nur, wenn für den C128 eine entsprechende Speichererweiterung eingebaut wurde.

# Symbole

Spezielle Symbole und Sonderzeichen können in einem BASIC-Programm eine große Rolle spielen.

| Symbol | Beispiel | Funktion |
|---|---|---|
| : | `10 A=1:B=2:C=3` | Erlaubt mehrere Befehle pro Zeile. |
| ; | `10 PRINT A;B` | Ausgabe in derselben Zeile. Zwischen den Elementen werden 2 Leerstellen ausgegeben. |
| + | `20 PRINT A$+B$` | Ausgabe in derselben Zeile. Die Zeichenkettenelemente werden verkettet. |
| , | `10 PRINT A,B` | Ausgabe in derselben Zeile. Elemente werden auf Tabulatorpositionen gesetzt (11, 21, 31 usw.). |
| | `LOAD "Name",D` | Trennt Parameter in Befehlen wie LOAD, SAVE, VERIFY usw. |
| ? | `10 ?A` | Abkürzung für PRINT. |
| $ | `10 A$="ABCDEF"` | Kennzeichen für Zeichenkette (STRING). |
| % | `10 A%=INT(B)` | Kennzeichen für ganze Zahl (INTEGER). |
| " | `10 A$="ABCDEF"` | Schließt Zeichenketten ein. |

# SYS (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** S<SHIFT>Y

**Typ:** Befehlswort

**Syntax:** SYS Speicheradresse

SYS Speicheradresse,A,X,Y,S
(nur BASIC 7.0)

**Parameterbereich:** Speicheradressen

Der SYS-Befehl gibt die Kontrolle an eine bestimmte Routine in Maschinensprache, die an der angegebenen Speicheradresse beginnt. Am Ende dieser Maschinenroutine muß ein RTS-Befehl (ReTurn from Subroutine) stehen, damit die Programmkontrolle an das BASIC-Programm zurückkehren kann.

In BASIC 7.0 können beim Aufruf einer Maschinenroutine durch den SYS-Befehl außerdem noch Parameter für den Akkumulator (A), das X-Register (X), das Y-Register (Y) und das Statusregister (S) übergeben werden.

### Anmerkungen

Der USR-Befehl übergibt ebenfalls die Kontrolle an eine Maschinensprache-Routine und gestattet zusätzlich die Übergabe eines Parameterwertes (siehe USR).

Wenn Sie bei den Rechnern C16 und C116 die Funktionstastenbelegung geändert haben, können Sie sie mit dem Befehl

```
SYS 62359
```

wieder in die Standardbelegung nach dem Einschalten zurückversetzen.

## (BASIC 2.0, 4.0, 3.5, 7.0) TAB

**Abkürzung:** T<SHIFT>A

**Typ:** Funktion zur Ausgabeformatierung

**Syntax:** PRINT TAB(X)

**Argumentbereich:** Bytezahl

Mit PRINT TAB kann die Ausgabe von Zeichen auf dem Bildschirm oder Drucker an eine bestimmte Position gebracht werden. Ausgehend vom linken Rand (TAB(0)) können dabei maximal 255 Positionen übersprungen werden (TAB(255)). Die übersprungenen Zeichen werden dabei nicht gelöscht.

Der Unterschied zur Funktion SPC ist, daß diese immer relativ zur Cursorposition vorgeht, während die Funktion TAB die Positionen, vom linken Rand ausgehend, konstant ansteuert.

Wenn Sie unmittelbar hinter einer TAB-Funktion eine zweite mit einem kleineren Parameter angeben, kann das deshalb nicht richtig funktionieren:

```
PRINT TAB(10)"ZWEITER";TAB(5)"ERSTER"
```

führt zur Ausgabe:

```
ZWEITERERSTER
```

während es mit SPC so aussieht:

```
ZWEITER     ERSTER
```

# TAN (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** keine möglich

**Typ:** trigonometrische Funktion

**Syntax:** `TAN(numerische Konstante, Variable oder Ausdruck)`

**Argumentbereich:** Gleitkommazahlen

**Ergebnisbereich:** Gleitkommazahlen

Die Funktion TAN berechnet den Tangens eines Argumentes, das im Bogenmaß gegeben ist. Ist das Argument in Grad angegeben, so kann man

```
TAN(X*.0174533)
```

zur Umrechnung verwenden.

## Beispiel

Wir wollen in einer Wertetabelle die Tangenswerte für Argumente im Bereich von -1.6π bis +1.6π ausgeben:

```
100 DEF FN R(X)=INT(100*X+.5)/100
110 PRINT CHR$(147)
120 PRINT TAB(4)"DIE TAN-FUNKTION"
130 PRINT
140 PRINT"   ARGUMENT        TAN"
150 PRINT"   --------        ---"
160 PRINT
170 FOR I=-1.6 TO 1.7 STEP .2
180 : I=FN R(I)
190 : PRINT TAB(3)I;TAB(8)"*PI";TAB(17);FN R(TAN(I*3.1416))
200 NEXT I
210 GETKEY T$
220 END
```

Die Laufvariable I in der FOR-Schleife ab Zeile 170 bestimmt die Argumente für die TAN-Funktion in Zeile 190. Wenn Sie das Programm mit RUN starten, sehen Sie die folgende Ausgabe:

```
 DIE TAN-FUNKTION

ARGUMENT        TAN
--------        ---

-1.6 *PI         3.08
-1.4 *PI        -3.08
-1.2 *PI        -.73
-1   *PI         0
-.8  *PI         .73
-.6  *PI         3.08
-.4  *PI        -3.08
-.2  *PI        -.73
 0   *PI         0
 .2  *PI         .73
 .4  *PI         3.08
 .6  *PI        -3.08
 .8  *PI        -.73
 1   *PI         0
 1.2 *PI         .73
 1.4 *PI         3.08
 1.6 *PI        -3.08
```

## Anmerkung

Bezüglich weiterer Informationen über die trigonometrischen Funktionen siehe auch die Einträge unter SIN und COS.

# TEMPO (BASIC 7.0)

**Abkürzung:** T<SHIFT>E

**Typ:** Sound-Befehl

**Syntax:** TEMPO X

**Parameterbereich:** Bytezahl

Mit der Anweisung TEMPO wird die Geschwindigkeit festgelegt, mit der nach PLAY die Noten gespielt werden. Normalerweise werden sie so gespielt, als sei TEMPO mit dem Parameter 8 angegeben worden.

X kann Werte von 0 bis 255 annehmen. Die entsprechende Zeit in Sekunden beträgt etwa:

19,22 / X Sekunden

### Beispiel

Zur Demonstration der TEMPO-Anweisung spielen wir den Anfang des Liedes "Alle Vögel sind schon da" in drei verschiedenen Tempi: zunächst langsam, dann mittelschnell und zum Schluß ganz schnell.

```
10 FOR G=8 TO 48 STEP 20
20 TEMPO G
30 PLAY ".CIEGO5CO4AO5CO4AHGIR.FIGECHDHC"
40 FOR I=1 TO 700:NEXT I
50 NEXT G
60 END
```

Die Darstellungsform der Noten nach dem PLAY-Befehl ist unter diesem Stichwort detailliert erläutert.

Wenn Sie RUN eingeben, wird die Melodie zunächst mit TEMPO 8 gespielt, im zweiten Durchgang dann mit TEMPO 28 und zum Schluß mit TEMPO 48.

## (BASIC 2.0, 4.0, 3.5, 7.0) TI oder TIME

**Abkürzung:** keine möglich

**Typ:** reservierte Variable

**Syntax:** `TI` oder `TIME`

Diese Variable liest die eingebaute Uhr des Rechners ab und gibt den laufenden Wert mit Einschränkungen quarzgenau aus. Eine solche Zeitkontrolle ist unverzichtbar bei Programmen für Messungen, Steuerungen und Simulationen. Sie können also Anweisungen wie die folgenden benutzen, um herauszufinden, wie lange Ihr Rechner gelaufen ist:

Anzahl von 1/60 Sekunden seit Beginn:

```
PRINT TI
```

Anzahl der Sekunden seit Beginn:

```
PRINT TI/60
```

Anzahl der Minuten seit Beginn:

```
PRINT TI/3600
```

Anzahl der Stunden seit Beginn:

```
PRINT TI/216000
```

Die Werte für TI bewegen sich von 0 bis maximal 5 184 000, der Anzahl von 1/60 Sekunden pro Tag, dann beginnt die Zählung wieder bei 0.

### Anmerkungen

Sie können sich allerdings auf die Genauigkeit der Uhr nicht mehr unbedingt verlassen, wenn Sie z. B. für Ein- und Ausgabeoperationen den Kassettenrecorder zwischendurch benutzen.

Die Variable TI ist kein reserviertes BASIC-Wort, d. h. Variablen wie z. B. STIER sind in Programmen zulässig. Wie auch ST und TI$ wird TI in einem gesonderten Bereich, nicht im normalen BASIC-Variablenbereich abgelegt.

## (BASIC 2.0, 4.0, 3.5, 7.0) TI$ oder TIME$

**Abkürzung:** keine möglich

**Typ:** reservierte Zeichenkettenvariable

**Syntax:** `TI$` oder `TIME$`

Die Variable TI$ liefert einen Zeichenkettenwert aus sechs Ziffern, die die Uhrzeit enthält. Die sechs Ziffern der Zeichenkette zeigen Stunden (HH), Minuten (MM) und Sekunden (SS) wie folgt an:

```
HHMMSS
```

Also bedeutet beispielsweise der TI$-Wert 112415 11:24:15 Uhr.

Um die TI$-Funktion als Tageszeit benutzen zu können, müssen Sie erst die laufende Uhrzeit einstellen. Dies können Sie in einer Zuweisungs-Anweisung durchführen, genauso als wäre TI$ eine einfache Zeichenkettenvariable. Nehmen wir einmal an, die augenblickliche Zeit sei 12.30 Uhr. Sie können die TI$-Funktion wie folgt einstellen:

```
TI$="123000"
```

Anschließend können Sie den Wert der Variablen abfragen, wann immer Sie die Uhrzeit wissen möchten:

```
PRINT TI$
```

### Anmerkungen

Genauso wie TI liefert TI$ fehlerhafte Werte, wenn ein Kassettenrecorder in Funktion war, wie etwa beim Sichern oder Laden von Programmen.

TI$ ist wie auch TI und ST kein reserviertes BASIC-Schlüsselwort, d. h. Variablennamen wie etwa STIMME$ sind zulässig. Die Variable TI$ wird ebenfalls nicht im normalen BASIC-Variablenspeicherbereich abgelegt.

# TRAP (BASIC 3.5, 7.0)

**Abkürzung:** T<SHIFT>R

**Typ:** Befehl zur Fehlerbehandlung

**Syntax:**
```
TRAP Zeilennummer
TRAP
```

**Parameterbereich:** Zeilennummern

Der Befehl TRAP verhindert, daß das Programm mit einer Fehlermeldung abgebrochen wird. Statt dessen wird zu einer Routine verzweigt, deren Beginn durch die Zeilennummer festgelegt wird. Hier kann der Benutzer Fehlerabfrage- und -auffangmaßnahmen vorsehen. Die Nummer der fehlerhaften Zeile kann in der Variablen ER abgefragt werden (siehe auch ER), die Fehlermeldung selbst in ERR$ (siehe auch ERR$). Der Befehl RESUME veranlaßt die Wiederaufnahme des Programmablaufs (siehe auch RESUME).

Wird die Anweisung TRAP ohne Angabe einer Zeilennummer verwendet, so wird beim nächsten auftretenden Fehler keine Fehlerbehandlung eingeleitet.

### Beispiel

Wir wollen bei einem auftretenden Fehler zur Zeile 500 verzweigen, Fehlercode, betroffene Zeilennummer und Fehlermeldung ausgeben und anschließend den Programmablauf fortsetzen.

```
100 TRAP 500
110 PRINT "FEHLERBEHANDLUNG MIT TRAP"
    :PRINT
120 A$=3
130 PRINT "PROGRAMMENDE"
140 STOP
500 PRINT "ES IST EIN FEHLER MIT DEM CODE"
    ;ER
510 PRINT "IN PROGRAMMZEILE";EL
    ;"AUFGETRETEN":PRINT
520 PRINT "DIE FEHLERMELDUNG LAUTET: "
```

```
    ;ERR$(ER):PRINT
530 PRINT "DRUECKEN SIE EINE BELIEBIGE TASTE"
540 GETKEY T$
550 RESUME 130
```

# TRON TROFF (BASIC 3.5, 7.0)

**Abkürzung:** TR<SHIFT>O TRO<SHIFT>F

**Typ:** Programmierhilfen

**Syntax:**
```
TRON
TROFF
```

Mit dem TRON-Befehl wird die Programmablaufverfolgung eingeschaltet (TRace ON) und mit TROFF wieder ausgeschaltet (TRace OFF). Programmablaufverfolgung bedeutet, daß die Nummern der gerade vom Programm bearbeiteten Zeilen auf dem Bildschirm angezeigt werden.

Das Programm, das bei eingeschalteter TRACE-Funktion läuft, protokolliert in eckigen Klammern jede Zeilennummer, die während der Programmbearbeitung ausgeführt wird, auf dem Bildschirm.

Der TRACE-Befehl stellt somit eine wirksame Unterstützung beim Austesten eines Programms dar, da auf diese Weise Fehler sehr schnell lokalisiert werden können.

### Anmerkungen

TRON und TROFF können sowohl im Direktmodus als auch in einer Programmzeile angewendet werden.

## (BASIC 2.0, 4.0, 3.5, 7.0) USR

**Abkürzung:** U<SHIFT>S

**Typ:** Funktion zur Programm-Ablaufsteuerung

**Syntax:** USR(Argument)

**Argumentbereich:** Gleitkommazahlen

Die USR-Funktion ermöglicht es Ihnen, aus einem BASIC-Programm heraus eine Maschinensprache-Unterroutine aufzurufen. Dabei kann, anders als beim SYS-Befehl, ein numerischer Ausdruck als Argument mitgegeben werden.

USR speichert den Wert des Arguments immer in einer festen Speicherstelle, so daß dieser Wert für die Maschinensprache-Routine, die von USR aufgerufen wird, verfügbar ist. Da USR eine Funktion ist, kann es natürlich in einem BASIC-Programm nicht allein stehen, sondern muß Teil einer Anweisung sein. USR könnte z. B. als Teil einer PRINT-Anweisung erscheinen:

```
PRINT USR(V)
```

oder als Teil einer Wertzuweisung:

```
LET T=USR(V)
```

Die Wirkung von USR ist die folgende:

1. Es speichert den Wert des Argumentes, hier von V, in den Speicherstellen 97 bis 102, dem Gleitkommaregister.

2. Dann führt es den Maschinensprache-Befehl JSR (Jump to SubRoutine - Sprung in eine Unterroutine) zu einer bestimmten Speicherstelle aus. Dort befinden sich weitere Angaben für die Verarbeitung eines Maschinenprogramms.

Die Speicheradresse dieses USR-Vektors hängt selbstverständlich davon ab, welchen Commodore-Rechner Sie benutzen. Beim VC20 ist es die Speicheradresse 0, beim C64 und beim C128 ist es Adresse 784. Diese Adressen enthalten den Maschinensprache-Befehl JMP (JuMP - Sprung). Bevor Sie die USR-

Funktion aufrufen, müssen Sie die Anfangsadresse Ihrer Maschinensprache-Routine in die beiden Speicherstellen holen, die dem JMP-Befehl folgen: Speicherplätze 1 und 2 beim VC20; Speicherplätze 785 und 786 beim C64 und C128. Beim C16, C116 und Plus/4 ist die Startadresse in den Speicherstellen 1281 und 1282 abgelegt.

Wir wollen einmal annehmen, Sie haben eine Maschinensprache-Routine für den VC20 geschrieben und im Speicher ab der Dezimaladresse 7424 abgelegt. Das hexadezimale Gegenstück dieser Adresse ist 1D00. Die letzten zwei Ziffern der hexadezimalen Adresse müssen Sie an die Speicherstelle 1 bringen und die höheren Ziffern an die Speicherstelle 2; dies erreichen Sie mit den folgenden beiden Befehlen:

```
POKE 1,0
POKE 2,29
```

wobei 29 das dezimale Gegenstück zu 1D ist.

Beim C16, C116 und Plus/4 heißt es dann:

```
POKE 1281,0
POKE 1282,29
```

Auf dem C64 oder C128 können Sie die gleiche Anfangsadresse an die Speicherstellen 785 und 786 bringen:

```
POKE 785,0
POKE 786,29
```

Die Werte dieser Vektoren testen Sie mit einem einfachen RTS-Befehl (der Befehlscode hierfür ist dezimal 96), den Sie an die Speicherstelle 7424 bringen:

```
POKE 7424,96
```

Nach diesen Vorbereitungen können Sie den USR-Befehl eingeben. Der durch USR ermittelte Wert ist einfach nur das Argument der Funktion selbst. Der Befehl

```
PRINT USR(25)
```

wird zum Beispiel den Wert 25 auf dem Bildschirm ausgeben. Das liegt daran, daß der durch USR ermittelte Wert aus dem Gleitkommaregister genommen wird, derselben Stelle, an der das Argument von USR gespeichert ist, wenn die Funktion angesprochen wird.

Da Ihre Maschinensprache-Routine (in diesem Fall einfach RTS) diesen Wert nicht beeinflußt, wird er unverändert von der Funktion zurückgegeben.

## Anmerkungen

Der SYS-Befehl übergibt die Programmkontrolle direkt an eine Maschinensprache-Routine an einer bestimmten Adresse (siehe auch unter SYS).

Der NEW-Befehl ändert weder eine im Speicher abgelegte Maschinensprache-Routine noch die Vektoradresse, die auf diese Routine hinweist.

# VAL (BASIC 2.0, 4.0, 3.5, 7.0)

**Abkürzung:** V<SHIFT>A

**Typ:** numerische Funktion

**Syntax:** VAL(Zeichenkettenkonstante, -variable oder -ausdruck)

**Argumentbereich:** Zeichenketten

**Ergebnisbereich:** Gleitkommazahlen

Die Funktion VAL liefert den numerischen Wert einer Zeichenkette. Dabei muß der erste Wert der Zeichenkette +, -, . oder eine Ziffer von 0 bis 9 sein, sonst ist das Ergebnis null. Führende Leerzeichen werden ignoriert. Enthält die Zeichenkette unzulässige Zeichen, so wird bis dahin umgewandelt und der Rest ohne Fehlermeldung weggelassen. Zugelassen sind aber Ausdrücke in wissenschaftlicher Schreibweise mit dem Buchstaben E.

## Beispiele

```
PRINT VAL("123")              ergibt 123
PRINT VAL("  -0.15")          ergibt -.15
PRINT VAL("12ABC34")          ergibt 12
PRINT VAL("+99.4321E2")       ergibt 9943.21
PRINT VAL("DM50")             ergibt 0
```

# (BASIC 2.0, 4.0. 3.5, 7.0) VERIFY

**Abkürzung:** V<SHIFT>E

**Typ:** Systembefehl für Ein-/Ausgabe

**Syntax:**
```
VERIFY
VERIFY"Dateiname"
VERIFY"Dateiname",Gerätenummer
```

**Parameterbereich:** Dateinamen
Gerätenummern

Der VERIFY-Befehl vergleicht das aktuelle Programm im Rechnerspeicher mit einer bestimmten Programmdatei auf Diskette oder Diskette. Wenn die beiden Programme identisch sind, gibt der Rechner die Meldung aus:

```
OK
```

Weichen die Programme voneinander ab, werden Sie dagegen die Meldung

```
?VERIFY ERROR
```

sehen. Es ist empfehlenswert, den VERIFY-Befehl jedesmal dann zu benutzen, wenn Sie ein Programm auf Kassette oder Diskette speichern, um sicherzustellen, daß die Speicheroperation erfolgreich und fehlerfrei durchgeführt wurde.

Die Gerätenummer ist 1 für Kassette und 8 für Diskette. Wird sie weggelassen, dann ist sie standardmäßig 1.

## Beispiel

```
SAVE"PROG1",8
VERIFY"PROG1",8
```

Wenn kein Dateiname angegeben wird, wird das nächste Programm auf der Kassette überprüft.

# VOL (BASIC 3.5, 7.0)

**Abkürzung:** V<SHIFT>O

**Typ:** Sound-Befehl

**Syntax:** VOL X

**Parameterbereich:** ganze Zahlen 0 - 8 BASIC 3.5
ganze Zahlen 0 - 15 BASIC 7.0

Die Lautstärke eines Tons kann mit dem VOL-Befehl in mehreren Stufen eingestellt werden. In BASIC 3.5 sind für X Werte von 0 bis 8, in BASIC 7.0 Werte von 0 bis 15 zugelassen. Dabei wird mit 0 jeweils der leiseste und mit 8 bzw. 15 der lauteste Ton erzeugt.

Alle Töne, die durch die Befehle SOUND oder PLAY gespielt werden, erklingen in der durch VOL eingestellten Lautstärke.

## Beispiel

Wir spielen mit SOUND eine Tonleiter. Die Frequenzwerte für die Töne C, D, E, F, G, A, H, C werden vorher zugewiesen. Die Lautstärke variiert von zuerst ganz leise (0) bis laut (15).

```
10 W(1)=4297:W(2)=4822:W(3)=5412:W(4)=5724:
   W(5)=6429:W(6)=7217:W(7)=8102:W(8)=8578
20 FOR LAUT=0 TO 15
30 : VOL LAUT
40 : FOR N=1 TO 8
50 :   SOUND 1,W(N),25
60 : NEXT N
70 NEXT LAUT
80 END
```

## (BASIC 2.0, 4.0, 3.5, 7.0) WAIT

**Abkürzung:** W<SHIFT>A

**Typ:** Systembefehl zur Eingabe

**Syntax:**
```
WAIT Adresse,N1
WAIT Adresse,N1,N2
```

wobei N1 und N2 numerische Ausdrücke sind.

**Parameterbereich:** Speicheradressen
Bytezahlen

Der WAIT-Befehl unterbricht die Programmausführung so lange, bis der Wert der adressierten Speicherstelle so verändert wird, daß durch Verknüpfung mit dem ersten und zweiten numerischen Ausdruck ein von null verschiedener Wert erzeugt wird. Die Verknüpfung geht dabei folgendermaßen vor sich:

N1 und N2 enthalten ganzzahlige Werte zwischen 0 und 255. N2 kann auch weggelassen werden. An seiner Stelle wird dann der Wert 0 angenommen. Zunächst wird der Inhalt der angegebenen Speicherstelle mit N2 über XOR (exklusives ODER) verknüpft. Das Ergebnis wird dann mit N1 über ein AND (logisches UND) verknüpft. Der Programmablauf bleibt so lange unterbrochen, bis dieses Ergebnis einen von null verschiedenen Wert liefert.

Die XOR-Logik sieht wie folgt aus:

```
1 XOR 1 = 0
1 XOR 0 = 1
0 XOR 1 = 1
0 XOR 0 = 0
```

Da beim WAIT-Befehl mit Speicheradressen gearbeitet wird, lassen sich natürlich keine Beispiele angeben, die für alle Rechner passen. Beim C64 und C128 bewirkt z. B.:

```
POKE 198,0:WAIT 198,1
```

daß gewartet wird, bis ein Zeichen im Tastaturpuffer ist.

### Anmerkungen

Wenn Sie jedoch nur eine einfache Warteschleife in Ihr Programm einbauen wollen, ist der WAIT-Befehl viel zu kompliziert. Eine genaue zeitlich begrenzte Pause läßt sich mit Hilfe der TI-Funktion (siehe dort) einbauen. Eine einfachere und natürlich weniger exakte Möglichkeit ist die, eine leere FOR-Schleife einzubauen:

```
100 FOR I=1 TO 1000:NEXT I
```

wird eine Pause von etwas mehr als einer Sekunde erzeugen. Experimentieren Sie selbst mit Pausen verschiedener Länge, etwa nachdem Sie auf dem Bildschirm gedruckt haben und bevor Sie ihn wieder löschen.

# (BASIC 7.0) WIDTH

**Abkürzung:** WI<SHIFT>D

**Typ:** Grafikbefehl

**Syntax:**
```
WIDTH 1
WIDTH 2
```

**Parameterbereich:** ganze Zahlen 1 und 2

Mit dem Befehl WIDTH kann die Strichstärke für Grafik bestimmt werden. Wenn also die Striche besonders dick aussehen sollen, wählen Sie WIDTH 2.

Alle Linien, die aufgrund der Zeichenbefehle DRAW, CIRCLE, BOX usw. gezeichnet werden, haben die doppelte Strichstärke. Um wieder auf normal umzuschalten, geben Sie WIDTH 1 ein. In einem Programm könnte das etwa folgendermaßen aussehen:

## Beispiel

Wir zeichnen konzentrische Kreise mit doppelter Strichstärke und dann zwischen die Kreise noch einmal Kreise mit einfacher Strichstärke.

```
100 GRAPHIC 1,1
110 WIDTH 2
120 FOR I=10 TO 100 STEP 10
130 : CIRCLE 1,160,100,I
140 NEXT I
150 WIDTH 1
160 FOR I=5 TO 100 STEP 10
170 : CIRCLE 1,160,100,I
180 NEXT I
190 GETKEY T$
200 GRAPHIC 0,1
210 END
```

Die entstandene Grafik können Sie so lange betrachten, bis Sie irgendeine beliebige Taste drücken. Dann wird der hochauflösende Bildschirm-Modus verlassen, und Sie kehren in den normalen Textbildschirm zurück.

# WINDOW (BASIC 7.0)

**Abkürzung:** W<SHIFT>I

**Typ:** Befehl zur Bildschirmsteuerung

**Syntax:**
```
WINDOW SOL,ZOL,SUR,ZUR
WINDOW SOL,ZOL,SUR,ZUR,1
```

**Parameterbereich:** Bildschirmkoordinaten
Bitzahlen

Mit der WINDOW-Anweisung kann man den Bildschirm in mehrere Bereiche gliedern, die unabhängig voneinander beschrieben werden können. Leider ist nur ein Fenster vorgesehen, so daß man nicht mehrere Bereiche gleichzeitig beschreiben kann. Will man das trotzdem machen, so muß das Fenster jedesmal neu definiert werden.

Abhängig von der Bildschirmgröße (40 oder 80 Zeilen) müssen die 4 Eckpunkte des Fensters festgelegt werden:

| | |
|---|---|
| SOL | Spalte oben links (min. 0) |
| ZOL | Zeile oben links (min. 0) |
| SUR | Spalte unten rechts (max. 39 bzw. 79) |
| ZUR | Zeile unten rechts (max. 24) |
| 1 | Fenster automatisch löschen |

### Beispiel

In unserem Beispiel wollen wir dem Benutzer die Möglichkeit geben, durch Eingabe einer Zeilen- und einer Spaltenangabe eine Aufteilung des Bildschirms in maximal 4 Fenster zu erhalten. Diese wird durch ein gezeichnetes Kreuz auf dem Bildschirm sichtbar gemacht. Die Funktionstasten F1, F3, F5 und F7 sollen sodann ein Hin- und Herspringen zwischen den einzelnen Fenstern ermöglichen. Durch Drücken von F2 kann man an den Anfang des Programms und somit zu einer neuen Fensteraufteilung gelangen.

Die Anzahl der Zeilen muß dabei mindestens 5 und die der Spalten mindestens 6 betragen, da ein sinnvolles Arbeiten in kleineren Fenstern nicht mehr möglich ist. Die Höchstwerte liegen für die Zeilen bei 24 und für Spalten bei 40.

Wenn Sie eine Aufteilung in 2 Fenster wünschen, geben Sie einen der beiden Werte mit 0 ein. Die Eingabe

```
0,7
```

würde beispielsweise bedeuten, daß keine Zeilenanzahl eingegeben wird, also der Bildschirm vertikal bei Spalte 7 in 2 Fenster nebeneinander aufgeteilt wird.

Umgekehrt bedeutet die Eingabe

```
12,0
```

daß keine Aufteilung nach Spalten gewünscht wird. Demnach wird der Bildschirm horizontal bei Zeile 12 in 2 Fenster untereinander aufgeteilt.

Wenn Sie überhaupt keine Aufteilung wünschen, können Sie

```
0,0
```

eingeben, und der ganze Bildschirm steht Ihnen zur Verfügung.

Das Programm sieht nun folgendermaßen aus:

```
61000 REM ******************************
61010 REM *   FENSTERVERWALTUNG          *
61020 REM ******************************
61030 REM DAS PROGRAMM IST AUCH AUF 7XX-
      RECHNERN MIT ESC-CODES FUER FEN-
      STERSTEUERUNG REALISIERBAR SOWIE
      MIT POWER (FKT-TASTEN) AUF 8XXX
61040 :
61050 DIM GV$(7)        :REM GITTERGRAFIK
61060 GV$(1)=CHR$(96)   :REM HORIZONTALES
                         VERBINDUNGSSTUECK
61070 GV$(2)=CHR$(98)   :REM VERTIKALES
                         VERBINDUNGSSTUECK
61080 GV$(7)=CHR$(123):REM KREUZUNGS-
                             STUECK
61090 OS=0              :REM ECKEOBENLINKS
61100 OZ=0              :REMECKEOBENRECHTS
61110 UZ=0              :REM ECKE UNTENRTS
61120 US=0              :REMECKEUNTENLINKS
61130 DIM FR$(4)
61140 DIM W(16)
```

```
61150 CF$=CHR$(19)     :REM HOME
61160 CS$=CF$+CF$      :REM HOME+HOME
61170 SM$=CHR$(27)+"M":REM SCROLLEN AUS
61180 SL$=CHR$(27)+"L":REM SCROLLEN AN
61190 SP=40            :REM SPALTEN
61200 Z=0              :REM ZEILE
61210 S=0              :REM SPALTE
61220 DF$=CHR$(147)    :REM LOESCHEN
61230 RT$=CHR$(13)     :REM RETURN
61240 E$="EINGABE FUER"
61250 H$=" HORIZONTALE":V$=" VERTIKALE "
61260 F$="FENSTERAUFTEILUNG       :"
61270 F4$=" 4 FENSTER  :"
61280 Z1$="  ZEILE,0         Z.B. 10,0"
61290 Z2$="  0,SPALTE        Z.B. 0,19"
61300 Z3$="  ZEILE,SPALTE    Z.B. 10,19"
61310 PRINT CS$+DF$+E$;H$;TAB(40)F$;Z1$
61320 PRINT TAB(40)E$;V$;TAB(40)F$;Z2$
61330 PRINT TAB(40)E$;F4$;Z3$
61340 CHAR 1,9,15,
          "F1,F3,F5,F7 FENSTER 1 BIS 4"
61350 CHAR 1,9,19,"EINGABE ZEILE,SPALTE"
61360 :
61370 REM*******************************
61380 REM*        EINGABE              *
61390 REM*******************************
61400 DO
61410 : EINGABE=1:FALSCH=0
61420 : INPUT Z,S
61430 : IF NOT (Z<25 AND Z>-1
          AND  S<SP AND S>-1)
        THEN
        BEGIN
61440 :   PRINT"BITTE 0<= ZEILE < 25
                      0<= SPALTE<";SP
61450 :   Z=1:S=80:FALSCH=1
61460 : BEND
61470 :
61480 LOOP WHILE EINGABE=FALSCH
61490 :
61500 REM******** ENDE EINGABE *********
61510 :
61520 :
61530 REM*******************************
61540 REM*       FENSTERRAHMEN         *
```

```
61550 REM*******************************
61560 PRINT DF$+CS$:FR$(1)="":FR$(2)="":
                    FR$(3)="":FR$(4)=""
61570 IF S>0 AND S<SP
      THEN WINDOW S,0,S,24
61580 FOR I=1 TO 25
61590 : FR$(4)=FR$(4)+GV$(2)
61600 NEXT I
61610 IF S>0 AND S<SP-1
      THEN PRINT SM$+FR$(4)+CS$
61620 IF S<SP-1 THEN
      BEGIN
61630 : DO
61640 :      FR$(2)=FR$(2)+GV$(1)
61650 : LOOP WHILE LEN(FR$(2))<ABS(S)
61660 BEND
61670 IF S>0 AND S<SP
      THEN FR$(2)=FR$(2)+GV$(7)
61680 IF S<SP-1 THEN
      BEGIN
61690 : DO
61700 :      FR$(3)=FR$(3)+GV$(1)
61710 : LOOP UNTIL
        LEN(FR$(3))=ABS(SP-S-1)
61720 BEND
61730 FR$(1)=FR$(2)+FR$(3)
61740 WINDOW 0,0,39,Z
61750 IF Z>0 AND Z<24 AND S<SP-1
      THEN CHAR 1,0,Z,SM$+FR$(1)
61760 :
61770 REM***** ENDE FENSTERRAHMEN ******
61780 :
61790 REM*******************************
61800 REM*      TASTENBELEGUNG          *
61810 REM*******************************
61820 WS$=""
61830 U1$="GOSUB 62070"+RT$
61840 U2$="GOSUB 62130"+RT$
61850 U3$="GOSUB 62190"+RT$
61860 U4$="GOSUB 62240"+RT$
61870 U5$="61310"+RT$
61880 KEY 1,WS$+U1$    :REM FENSTER 1
61890 KEY 3,WS$+U2$    :REM FENSTER 2
61900 KEY 5,WS$+U3$    :REM FENSTER 3
61910 KEY 7,WS$+U4$    :REM FENSTER 4
```

```
61920 KEY 2,"GOTO"+U5$ :REM FENSTER NEU
61930 PRINT CS$:WINDOW 0,23,2,24
61940 PRINT SL$:GOSUB 62020
61950 END
61960 :
61970 REM***** ENDE TASTENBELEGUNG *****
61980 :
61990 REM******************************
62000 REM*    FENSTERDEFINITIONEN      *
62010 REM******************************
62020 IF S=0 THEN S=SP-1
62030 IF Z=0 THEN Z=24
62040 OS=0:OZ=0:US=S-1:UZ=Z-1
62050 PRINT CS$:WINDOW OS,OZ,US,UZ
62060 RETURN
62070 IF Z<1 OR Z>23 THEN OZ=0:UZ=23
62080 WINDOW OS,OZ,US,OZ+4
62090 OS=0:OZ=0:US=S-1:UZ=Z-1
62100 :
62110 PRINT DF$+CS$:WINDOW OS,OZ,US,UZ
62120 RETURN
62130 IF Z<1 OR Z>23 THEN OZ=0:UZ=23
62140 WINDOW OS,OZ,US,OZ+4
62150 IF S<39 THEN OS=S+1
62160 OZ=0:US=SP-1:UZ=Z-1
62170 PRINT DF$+CS$:WINDOW OS,OZ,US,UZ
62180 RETURN
62190 IF Z<1 OR Z>23 OR OZ>23 THEN OZ=0
62200 WINDOW OS,OZ,US,OZ+4
62210 OS=0:OZ=Z+1-1*(0↑(Z-24)):
      US=S-1:UZ=24
62220 PRINT DF$+CS$:WINDOW OS,OZ,US,UZ
62230 RETURN
62240 IF Z<1 OR Z>22 OR OZ>23 THEN OZ=0
62250 WINDOW OS,OZ,US,OZ+4
62260 IF S<39 THEN OS=S+1
62270 OZ=Z+1-1*(0↑(Z-24)):US=SP-1:UZ=24
62280 PRINT DF$+CS$:WINDOW OS,OZ,US,UZ
62290 RETURN
```

Im ersten Teil des Programms bis zur Zeile 61300 erfolgen die Dimensionierungen und Startwertzuweisungen. Hier wurde das Prinzip verwirklicht, das unter dem Stichwort CHR$ vorgeschlagen wurde. Alle Steuerzeichen, die im Programm Verwendung finden, werden zu Anfang über die CHR$-Funktion

Variablen zugewiesen. Eine Liste von möglichen Zuweisungen finden Sie unter dem Stichwort CHR$.

Der Programmteil, der die Zeilen 61310 bis 61520 umfaßt, baut den Eingabebildschirm auf, nimmt die Eingaben entgegen und dokumentiert sie. Beachten Sie bitte, wie mit Hilfe der CHAR-Anweisung Meldungen gezielt auf dem Bildschirm plaziert werden können (Zeilen 61340 und 61350). Die Zulässigkeit der Eingabewerte - maximal 24 für die Zeilenanzahl und maximal SP-1 für die Spaltenanzahl, also 39 beim 40-Zeichen-Bildschirm, - wird geprüft (Zeile 61430), und gegebenenfalls muß die Eingabe vom Benutzer wiederholt werden.

Entsprechend der Eingabe wird nun in den Zeilen 61530 bis 61780 die Grafik für die Fensterrahmen ausgegeben. Dabei müssen die Spezialfälle berücksichtigt werden, wenn nur 2 Fenster erscheinen sollen, einer der Eingabewerte, Zeile oder Spalte, also 0 war. In diesem Fall wird nur eine Unterteilungslinie gezeichnet.

Ab Zeile 61790 bis Zeile 61980 werden nun die Funktionstasten belegt. F1 enthält dann den Sprungbefehl in das Unterprogramm ab Zeile 62070, der sogleich ausgeführt wird, da die Variable RT$ den Steuercode für RETURN enthält. F3 verzweigt zum Unterprogramm ab Zeile 62130, F5 zum Unterprogramm ab Zeile 62240 und F7 zum Unterprogramm ab Zeile 61860. Die Funktionstaste F2 wird mit dem Befehl GOTO 61310 <RETURN> belegt, der nach dem Drücken der Taste F2 einen Sprung an den Programmanfang zur erneuten Eingabe der Fenstergrößen bewirkt.

*Hinweis:* Die Zeilennummern hinter den GOSUB- bzw. hinter dem GOTO-Befehl sind fest vorgegeben. Falls Sie in Ihrem Programm eine andere Numerierung wählen, müssen Sie diese Zeilennummern entsprechend ändern. Auch wenn Sie das Programm mit RENUMBER umnumerieren, ist es erforderlich, daß Sie die Zeilennummern in den Zeilen 61830 bis 61870 von Hand anpassen, da der RENUMBER-Befehl keine Zeilennummern in Zeichenketten erkennen und umnumerieren kann.

Im letzten Programmteil, der in Zeile 61990 beginnt, werden entsprechend der gedrückten Funktionstaste die Fenster definiert. Dabei werden die Eingabewerte in den Variablen Z (Zeilenanzahl) und S (Spaltenanzahl) für die Berechnung der WINDOW-Parameter herangezogen. Ein kleiner Trick läßt dabei die Anzeige des GOSUB-Befehls verschwinden, die nach jedem Drücken einer Funktionstaste auf dem Bildschirm erscheint. Ein Hilfsfenster von 4 Zeilen im jeweiligen Fenster nimmt diesen Text auf (Zeilen 62080, 62140, 62200 und 62250) und wird dann wieder gelöscht.

Nachdem Sie das Programm gestartet und mit Eingabewerten versehen haben, können Sie durch Drücken der Funktionstasten F1, F3, F5 und F7 in die einzelnen Fenster springen. Es sieht also so aus, als ob Sie 4 Fenster gleichzeitig zur Verfügung haben. Das ist natürlich nicht der Fall. Das jeweils zuvor aktuelle Fenster wird nach jedem Funktionstastendruck aufgehoben, und das nächste wird neu definiert.

*Hinweis:* Vielleicht können Sie sich vorstellen, dieses Fensterprogramm in ein anderes Programm einzubauen. Dann ist es eleganter, die Sprünge von einem in ein anderes Fenster über andere Tasten als die Funktionstasten zu steuern, z. B. über die Nummerntasten 1 bis 4. Unser Programm ist natürlich flexibler, denn es arbeitet quasi im Direktmodus. Sie können mit DIRECTORY Inhaltsverzeichnisse in die Fenster laden usw. Für spezielle Zwecke, in denen das nicht erforderlich ist, gibt es also Lösungen, die komfortabler aber nicht so flexibel sind.

# (BASIC 7.0) XOR

**Abkürzung:** X<SHIFT>O

**Typ:** Verknüpfungs-Funktion

**Syntax:** XOR(A,B)

**Argumentbereich:** Speicheradressen

**Ergebnisbereich:** Speicheradressen

Die Funktion XOR ist ein logischer Operator, von dem Entscheidungen für den Programmablauf abgeleitet werden können. Er tritt beispielsweise bei den Anweisungen IF und WHILE auf. Man bezeichnet XOR auch als "exklusives ODER" oder "Antivalenz". Die Wahrheitstabelle für XOR sieht folgendermaßen aus:

```
A   B   XOR
1   1    0
1   0    1
0   1    1
0   0    0
```

Wenn A und B zwei Ausdrücke sind, dann ist also A XOR B genau dann wahr, wenn entweder A oder B wahr ist.

Die Parameter A und B dürfen Werte von 0 bis 65535 annehmen.

## Beispiel

```
PRINT XOR(30,40)
```
ergibt 54

30 entspricht binär 011110
40 entspricht binär 101000

verknüpft mit XOR 110110=54 dezimal

## Anmerkung

Neben XOR gibt es noch die logischen Operatoren OR, AND und NOT.

Kapitel 9

# Programmstrukturierung und -steuerung in BASIC

## Darstellungshilfsmittel

Der Computer wird dazu verwendet, um Probleme zu lösen. Dazu müssen wir uns zunächst darüber im klaren sein, wie das Problem aussieht, und versuchen, einen möglichst genauen Anforderungskatalog zu erstellen.

Danach wird ein geeigneter Algorithmus gesucht. Ein Algorithmus ist eine endliche, exakte und korrekte Vorschrift, die bei geeigneten Eingabedaten zwangsläufig zu korrekten Ausgabedaten führt.

Diese Verarbeitungsvorschrift muß man nicht sofort in BASIC ausdrücken, sondern man kann sie auch zunächst in einem sogenannten Pseudocode formulieren. Wir wollen einmal ein Beispiel betrachten. Es soll in maximal 7 Versuchen eine Zufallszahl von 1 bis 100 geraten werden. Im Pseudocode kann man das Problem folgendermaßen ausdrücken:

```
Erzeuge Zufallszahl
Wiederhole siebenmal:
   Einlesen einer Zahl;
   Prüfen, ob Zahl gleich Zufallszahl ist;
        wenn ja, dann positive Meldung und Ende;
        andernfalls prüfen, ob Zahl größer als
        Zufallszahl ist;
            wenn ja, Meldung "Zahl zu groß";
            andernfalls Meldung "Zahl zu klein";
Wiederholungsende
Negative Meldung
Programmende
```

Dieser Pseudocode wird natürlich von unserem Rechner so nicht verstanden. Er ist aber trotzdem sehr nützlich, weil der Programmierer gezwungen wird, das Problem, das gelöst werden soll, auszuformulieren. Außerdem ist der

Pseudocode rechner- und sprachenunabhängig. Er kann im Nachhinein in jede beliebige Programmiersprache umgesetzt werden.

Schematischer und in der Logik besser durchschaubar ist die Darstellung des Problems in einem Fluß- oder Ablaufdiagramm. Unsere kleine Aufgabe sähe damit aus, wie in Abb. 9.1 gezeigt.

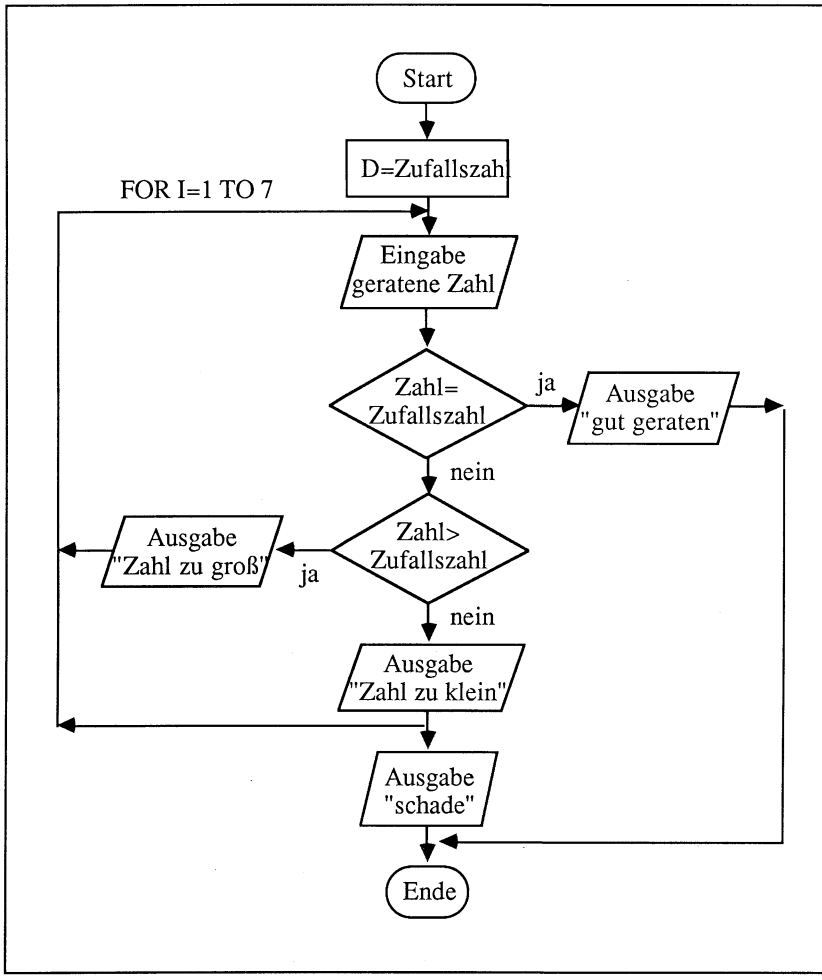
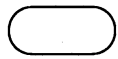
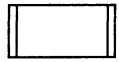
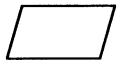


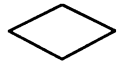

*Abb. 9.1: Programmablaufplan (PAP) oder Flußdiagramm*

Die im Programmablaufplan, auch kurz PAP genannt, verwendeten Sinnbilder sind nach DIN 66001 genormt und haben folgende Bedeutung:


Anfang oder Ende


Unterprogramm


Ein- oder Ausgabe


Verzweigung


Flußlinie


Anschlußpunkt

Als Hilfsmittel zum Zeichnen dieser Symbole können fertige Schablonen verwendet werden.

Eine andere Art zur Darstellung einer Programmlogik stellt das Blockdiagramm dar. Man nennt es auch Strukturdiagramm, Struktogramm oder nach seinem Erfinder Nassi-Shneiderman-Diagramm. Unsere kleine Aufgabe sieht, mit dem Struktogramm dargestellt, aus wie in Abb. 9.2.

Eine Gegenüberstellung der wichtigsten Programmstrukturen einmal im Flußdiagramm und einmal im Struktogramm zeigt die Abb. 9.3.

Der Programmentwurf ist das Wichtigste beim Programmieren und nicht etwa das reine Codieren, also das Formulieren des Algorithmus in irgendeiner Programmiersprache, wie man vielleicht meinen könnte. Fehler, die bei der Planung gemacht wurden, sind die schlimmsten, denn sie können dazu führen, daß womöglich wochenlange Arbeit umsonst war und von anderen Ausgangspunkten aus wiederholt werden muß.

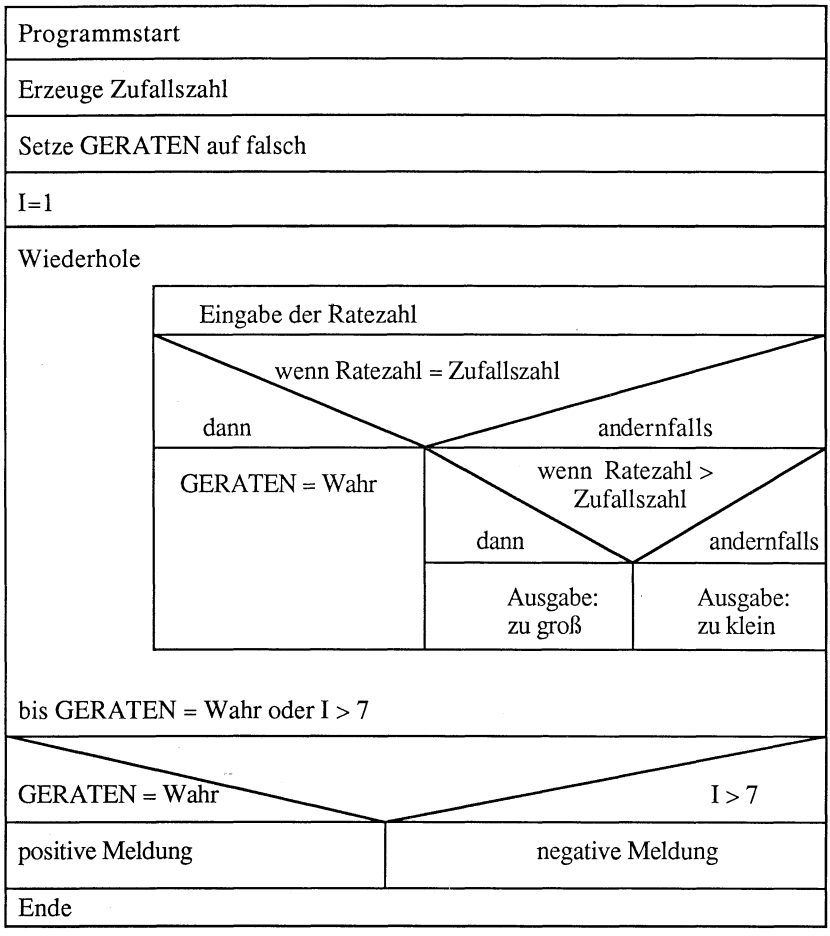



*Abb. 9.2: Blockdiagramm oder Struktogramm*

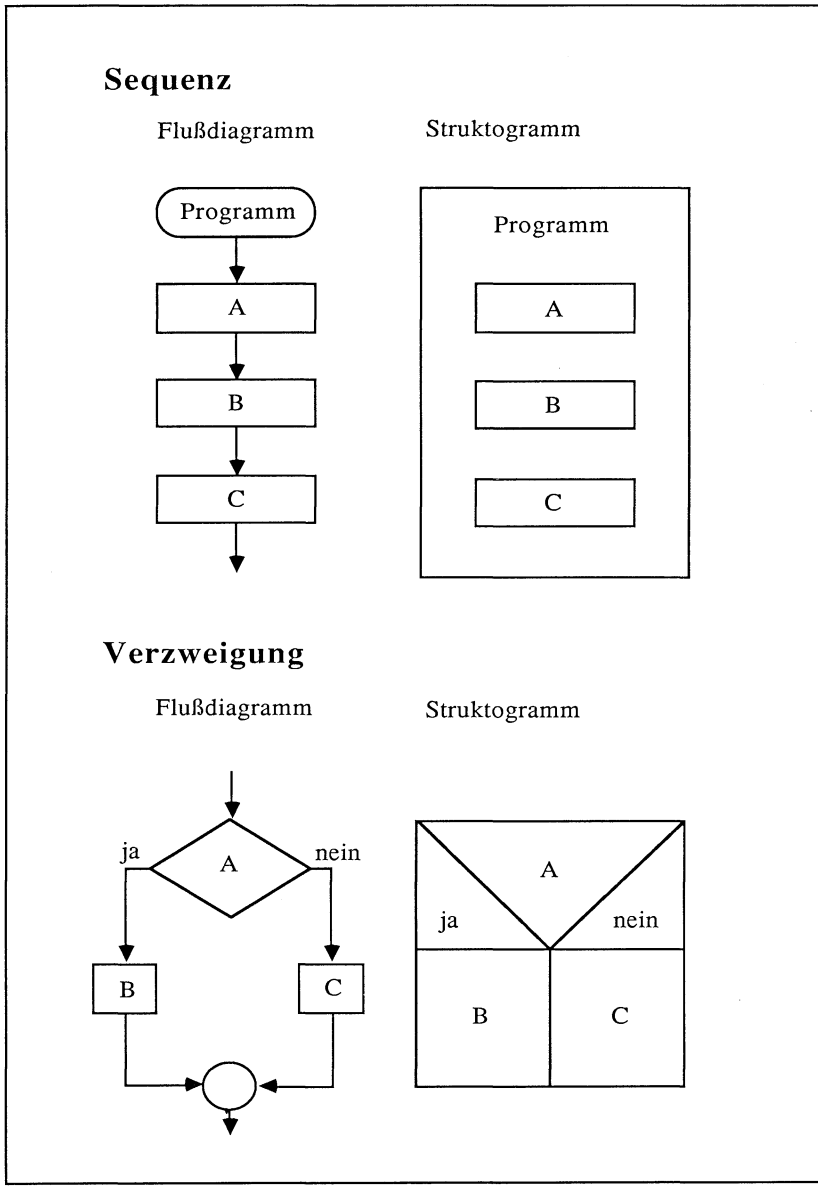
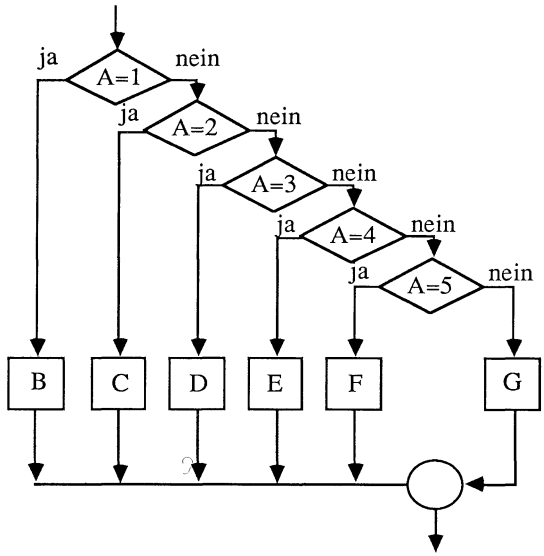



*Abb. 9.3 (Teil 1): Die wichtigsten Strukturen im Flußdiagramm und im Blockdiagramm (Sequenz und Verzweigung)*

Flußdiagramm




Struktogramm

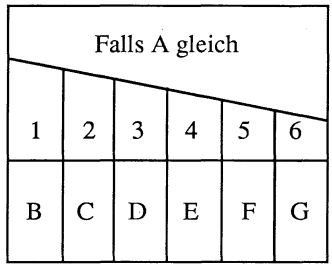
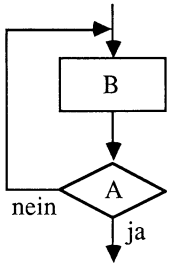


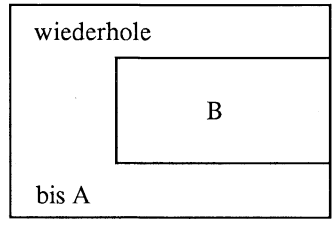

*Abb. 9.3 (Teil 2): Die wichtigsten Strukturen im Flußdiagramm und im Blockdiagramm (Mehrfachentscheidung)*

**wiederhole bis ...**

Flußdiagramm

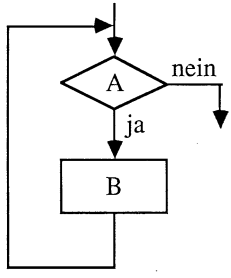



Struktogramm




**solange ... tue**

Flußdiagramm




Struktogramm

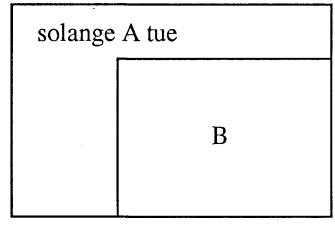



*Abb. 9.3 (Teil 3): Die wichtigsten Strukturen im Flußdiagramm und im Blockdiagramm (Schleifen)*

## Datenorientierte Strukturierung

Neben der ablauforientierten Programmstruktur ist noch die Struktur der Daten für die Programmgestaltung von erheblicher Wichtigkeit. Sie spiegelt sich im Aufbau der Dateien und ihrer Beziehung untereinander wider, also auch in der Ausgestaltung der Variablen.

Davon ist ebenfalls die Gestaltung der Programmroutinen betroffen, die Datenzugriffe und -manipulationen durchführen, weil dabei stets bestimmte Beziehungen der Daten untereinander programmtechnisch entsprechend berücksichtigt werden müssen.

Eine Postdresse hat z. B. eine sozusagen natürlich vorgegebene Struktur, die aus einzelnen Teilfeldern wie Name, Vorname, Straße, Hausnummer, Postleitzahl, Stadt, Kreis und Land bestehen kann.

Ein Personalkennsatz läßt sich ebenfalls auf naheliegende Weise in einzelne Felder wie Name, Vorname, Personalnummer, Dienstanschrift, Diensttelefon, Alter, Geschlecht, Beruf, Position, Betriebszugehörigkeit und Gehalt zergliedern. Dabei kommt einzelnen Feldern eine unterschiedliche Bedeutung zu. Beschreibende Felder wie Alter und Geschlecht haben Attributcharakter und werden weniger häufig zur Herstellung von Beziehungen mit verschiedenen Datenwerten eingesetzt, wie beispielsweise der Name, über den ja eine Beziehung zum Datenwert Adresse besteht. Der Datenwert Adresse ist als Spezialfall Dienstadresse sogar eine größere Teilstruktur des Personalsatzes.

Wichtige Felder wie Namen oder die Personalnummer werden in der Regel häufig verwendet, um bestimmte Datensätze in einem Datenfeld oder einer Datei aufzufinden. Man nennt solche Felder, die unbedingt in jedem Datensatz gleichermaßen vorhanden sein müssen, Schlüsselfelder (englisch: Key).

BASIC bietet nur in beschränktem Maße Unterstützung für Datenstrukturen an. Es kennt zwar Satzschlüssel bei relativen Dateien, diese müssen aber zwischen 1 und 65535 liegen. Schlüsselbegriffe wie Personalnummer lassen sich meist durch relative einfache mathematische Umformungen auf den Bereich 1 bis 65535 abbilden.

Etwas schwieriger ist es mit Schlüsseln wie dem Namen. Hierbei kann zwar prinzipiell mit Hilfe eines Algorithmus aufgrund der alphabetischen Sortierung und der Länge eine Zuordnung zu den Zahlen 1 bis 65535 vorgenommen werden, doch ist das schon recht komplex umd muß vielen Ausnahmefällen, wie z. B. Namensgleichheit (zweimal Hans Mueller) Rechnung tragen. Hier ist es sinnvoller, mit einer sequentiellen Datei, in DATA-Feldern oder in einem Datenfeld eine einfache Suchtabelle zu bilden, die jeweils Paare der Art Name,

Zahl enthält. Dabei gibt die Zahl genau eine Satznummer an und ermöglicht so eine Zuordnung von Namen und Satznummern.

Für das strukturierte Lesen und Schreiben von Daten aus/in Dateien bietet BASIC mit der Variablenliste nach INPUT# und PRINT# und mit der Möglichkeit der gezielten Angabe einer Byte-Position im Satz beim RECORD#-Befehl hinreichende Möglichkeiten.

Dabei sollte die Zusammengehörigkeit der einzelnen Satzfelder in Form eines Satzes am besten durch eine Zeichenkettenvariable entsprechenden Namens ausgedrückt werden. Einem Satzfeld entspricht dann immer ein Feld der Feldvariablen.

Bei Sätzen, die nur Zahlenwerte enthalten, ist natürlich ein Gleitkommazahlenfeld bzw. sogar ein Integerzahlenfeld effizienter, da dann entsprechende Umformungen vorgenommen werden können.

**Beispiel**

```
100 DIM AD$(7)                          :REM ADRESSATZ
110 AD$(1)="0000"                       :REM POSTLEITZAHL
120 AD$(2)="**********"                 :REM STADT
130 AD$(3)="**********"                 :REM STRASSE
140 AD$(4)="NNNN"                       :REM HAUSNUMMER
150 AD$(5)="**********"                 :REM NACHNAME
160 AD$(6)="**********"                 :REM VORNAME 1
170 AD$(7)="**********"                 :REM VORNAME 2
```

Dabei ist es sicher sinnvoll, eine weitere Zeichenkettenvariable als komplette Adresse zu definieren, da BASIC nicht das Lesen aller Zeichenkettenfelder als quasi eine Zeichenkette erlaubt:

```
180 AD$=AD$(1)+AD$(2)+AD$(3)+AD$(4)+AD$(5)
       +AD$(6)+AD$(7)                   :REM ADRESSE
```

Umgekehrt ist das Herausgreifen einzelner Felder aus der langen Zeichenkette AD$ mit der INSTR-Funktion einfach möglich.

Um die Zeichenkettenfelder Postleitzahl und Hausnummer wie Zahlen zu verarbeiten, können sie mit der Funktion VAL Ziffer für Ziffer umgewandelt werden.

In analoger Weise wie eine Variablenstruktur für die Adresse gebildet wurde, kann auch ein Personalsatz einer Feldvariablen zugewiesen werden. Dabei ist es günstig, wenn der gleiche Index für den Namen in beiden Felder gewählt

wird. Daß beide Felder für den Namen die gleiche Länge, wird in der Regel ohnehin möglich sein.

Die Bildung komplexer Variablenstrukturen läßt sich in ähnlicher Weise wie bei den Satzvariablen durchführen.

## Ablauforientierte Strukturierung

Für die rein ablauforientierte Strukturierung ist BASIC wesentlich besser ausgestattet.

### Einfache Strukturierung

Die Zeilennumerierung als einfachstes Gestaltungsmittel erlaubt durch Bildung von Nummernblöcken Gruppenbildungen von Programmzeilen, um Zusammengehörigkeiten auszudrücken. Normalerweise arbeitet BASIC die Programmzeilen entsprechend der aufsteigenden Zeilennummernfolge ab.

Ein weiteres einfaches, aber sehr effektives Strukturierungsmittel ist der REM-Befehl. Er gestattet es, das Programm mit Sternchenzeilen optisch zu zergliedern, wobei Sie selbstverständlich auch andere Sonderzeichen oder Grafikzeichen verwenden können. Besonders wirkungsvoll ist es, besondere Programmabschnitte mit eingerahmten Kommentarkästen einzuleiten. Gezielte Erläuterungen zu einzelnen Anweisungen werden, wenn es der Platz gestattet, in einer REM-Anweisung in der rechten Hälfte der betreffenden Programmzeile eingefügt.

### Schleifen

Für die Schleifenbildung zur wiederholten Ausführung bestimmter Anweisungsgruppen gibt es in BASIC die Befehlskombination FOR-NEXT, DO-LOOP, DO-WHILE-LOOP, DO-LOOP-WHILE, DO-UNTIL-LOOP, DO-LOOP-UNTIL und GOTO.

Der GOTO-Befehl ist allerdings strikten Verfechtern der strukturierten Programmierung ein Dorn im Auge und wird für die Schleifenbildung wirklich nicht benötigt, da die anderen BASIC-Möglichkeiten hierfür voll ausreichen.

Für reine Zählschleifen, die von komplexen logischen Bedingungen abhängen, ist eine der fünf DO-LOOP-Varianten als Hilfsmittel zu wählen. Dabei sollten die Anweisungen der Wiederholungsgruppe aus optischen Gründen jeweils zwei Spalten eingerückt werden. Da BASIC eine solche Einrückung norma-

lerweise ignoriert, kann man einen Doppelpunkt jeweils in jede Programmzeile schreiben, die Anweisungen der Wiederholungsgruppe enthält.

Wenn mehrere FOR-NEXT- oder DO-LOOP-Schleifen ineinander verschachtelt werden, schlagen wir vor, pro einschließender Schleifengruppe jeweils zwei weitere Spalten einzurücken.

Falls in einer FOR- oder DO-Schleife in Abhängigkeit von einer Bedingung die Schleife mit GOTO bzw. EXIT verlassen werden soll, darf im Sinne einer strukturierten Programmierung nur unmittelbar hinter das NEXT- bzw. das LOOP-Befehlswort in die folgende Programmzeile verzweigt werden.

### Verzweigungen

Die wichtigsten Elemente zur Steuerung des Programmablaufs und logischen Gliederung eines Programms stellen die Befehle zur Verzweigung der Programmkontrolle dar.

Es gibt die unbedingten Verzweigungen, die mit GOSUB und GOTO gesteuert werden können. GOSUB unterstützt auch deshalb ein strukturiertes Programmieren, weil es die Bildung von Anweisungsgruppen zur Realisierung einer mehrfach benötigten Funktion erlaubt, dabei aber den sequentiellen Verarbeitungsfluß nicht unterbricht, da die Kontrolle nach Ausführung des entsprechenden Unterprogramms an die nächste Anweisung nach GOSUB zurückgegeben wird. Man kann sich das auch so vorstellen, daß die Programmzeilen der mit GOSUB angesprungenen Routine anstelle von GOSUB in das Programm eingebettet werden können.

GOTO sollte nur sehr restriktiv ausschließlich für Sprünge über eine überschaubare Zeilennummerndistanz und in Richtung aufsteigender Zeilennummern eingesetzt werden.

Aufgrund einer gewissen Laxheit des BASIC-Interpreters ist es möglich, unmittelbar hinter der als Sprungziel angegebenen Zeilennummer noch ein Textlabel anzugeben. Dieses Textlabel kann man in einer REM-Zeile wiederholen, die man als Sprungziel direkt vor der eigentlichen Zielroutine angibt.

Alle Unterprogramme sollten durch einen Kommentarkopf eingeleitet werden und in einer geschlossenen Gruppe am Programmende stehen.

Eine besondere Form der unbedingten Verzweigung stellen die Befehle RETURN und RESUME dar, die für eine ordnungsgemäße Beendigung der GOSUB- und TRAP-Routinen erforderlich sind.

Die Befehlswortkombinationen zur bedingten Verzweigung ON-GOSUB, ON-GOTO, IF-THEN, IF-THEN-ELSE stellen die mächtigste Möglichkeit zur Ablaufsteuerung von BASIC-Programmen dar. Zugleich erlauben sie eine klare Gliederung des Programms. Für die Fallunterscheidung für mehr als vier verschiedene Fälle ist generell ON-GOSUB die erste Wahl, auch wenn das erst eine entsprechende Aufbereitung der Bedingung als Folge natürlicher Zahlen erfordert.

Handelt es sich bei den Routinen, die je nach Sachlage zur Weiterverarbeitung angesprungen werden sollen, um keine mehrfach verwendbaren Programmteile, sollte ON-GOTO eingesetzt werden, wobei die angesprochenen Anweisungsgruppen unmittelbar hinter der Zeile mit dem GOTO codiert werden sollen. Auch diese Anweisungsgruppen können zwei Spalten eingerückt werden.

IF-THEN und IF-THEN-ELSE eignen sich vor allem für die Behandlung komplexer Bedingungen oder für Verzweigungen mit nicht allzu stark aufgefächerter Fallunterscheidung. Dabei sollte sowohl die THEN-Befehlsgruppe als auch die ELSE-Befehlsgruppe jeweils zwei Spalten eingerückt werden und ELSE und THEN sollten bündig untereinander ausgerichtet stehen.

Spezielle Formen der bedingten Verzweigung für die Behandlung von Sprite-Kollisionen bzw. von Fehlersituationen stehen mit COLLISION und TRAP zur Verfügung. Diese beiden Befehle erlauben eine ausführliche Behandlung der entsprechenden Sonderfälle an geeigneter Stelle, ohne die Programmlogik für die Behandlung der Normalverarbeitung zu überfrachten und intransparent erscheinen zu lassen.

Eine elegante, wenn auch durch die Beschränkung auf numerische Werte etwas abgeschwächte Möglichkeit zur Strukturierung von BASIC-Programmen stellen die benutzerdefinierten Funktionen dar. Sie erlauben es, in noch stärkerem Maße als Unterprogramme mit GOSUB gezielt für einzelne Funktionen Befehlsgruppen zusammenzufassen. Da sie innerhalb von Ausdrücken, also auch in Bedingungen, als Parameter oder Argument sowie in PRINT- und CHAR-Anweisungen auftreten können und überdies mit einem halbwegs sprechenden Namen aufgerufen werden können, tragen sie so erheblich zur Lesbarkeit von BASIC-Programmen bei. Sie sollten so weit wie möglich verwendet werden, wobei allerdings ihre Handhabung bei voller Ausreizung ihrer Möglichkeiten im Grenzfall doch etwas kompliziert werden kann.

Die Definition von Benutzerfunktionen kann am Programmanfang stehen oder auch am Programmende in der Nähe der Unterprogramme. Allerdings müssen die Definitionen dann als Unterprogramm realisiert werden, so daß ein am Programmanfang befindliches GOSUB in geeigneter Weise für die Initialisierung sorgt.

### Unterbrechungen und Verzögerungen

Die Befehle END und STOP erlauben einen Programmabbruch an beliebiger Stelle im Programm.

Die Befehle WAIT, GETKEY, INPUT und INPUT# gestatten das Warten auf ein bestimmtes Ereignis. Das läßt sich jedoch auch mit einer einfachen DO-LOOP-Kombination mit eingeschlossener fortlaufender Abfrage auf den Wert einer Systemfunktion oder einer Speicherstelle mit anschließendem EXIT-Befehl für eine Vielzahl von Problemstellungen realisieren.

Mit SLEEP ist eine Verzögerung um einen bestimmten Zeitfaktor möglich.

## Kapitel 10

# Programmentwicklung und Austesten in BASIC

Die Zeit, die Sie für das Erlernen der grundlegenden Programmiertechniken benötigen, stellt eine gute Investition dar. Neben dem offensichtlichen Vorteil, den Computer endlich zur Abwicklung einer großen Zahl von Aufgaben nach eigenen Vorstellungen einsetzen zu können, gibt es eine Reihe weiterer Vorteile, die nicht so unmittelbar auf der Hand liegen.

Wenn Sie erst einmal eine Computersprache erlernt haben, fällt Ihnen das Erlernen einer weiteren Programmiersprache sehr leicht. Obwohl die verschiedenen Sprachen unterschiedliche Befehlswörter und Regeln besitzen, gibt es praktisch keine Unterschiede bei den wesentlichen Programmiertechniken.

Die meisten Computer-Anwendungen erfordern eine dauerhafte Speicherung von Daten auf externen Speichermedien. Das wird in Kapitel 5 behandelt. Jedes Software-System, das mit Dateien arbeitet, stellt gewisse Grundanforderungen an den Computer:

- zusätzliche Daten in bestehende Dateien einfügen;
- Dateien ausdrucken;
- Daten ändern;
- Daten sortieren;
- Daten suchen.

Jeder dieser Anforderungen läßt sich eine bestimmte Programmiertechnik zuordnen.

Für die Handhabung und Pflege von Dateien benötigt man im Grunde nur einen bestimmten Satz von Programmierverfahren, den man immer wieder anwendet. Das gleiche gilt auch für die Ein- und Ausgabe auf den Bildschirm sowie für Ausgaben auf Drucker oder Plotter.

Wenn Sie es also richtig anfangen, brauchen Sie nur ein einziges Mal Routinen für bestimmte elementare Aufgabenstellungen zu erstellen. Falls Sie irgendwann später für ein neues Projekt diese Programmteile benötigen, übernehmen Sie sie einfach in Ihr neues Programmsystem, statt sie speziell für die neue Anwendung zu erstellen.

In dem Maße, wie Ihre persönliche Bibliothek von Programmteilen anwächst, brauchen Sie bei der Programmerstellung nur noch Ihre Routinen einzusetzen und können auf das langwierige und fehlerträchtige Niederschreiben einer großen Zahl einzelner Programmzeilen verzichten. Das führt letztlich zu einer enormen Verkürzung des Zeitbedarfs für die sogenannte Programmcodierung und für die Testläufe zur Fehlerbeseitigung. Sie erwerben so die Fähigkeit, modular zu programmieren, und Ihre Programme werden übersichtlicher und leichter verständlich. Das ist auch eine der wesentlichen Anforderungen im Bereich der professionellen Programmierung.

Natürlich müssen Sie, bevor wir uns im folgenden intensiv mit Programmiertechniken beschäftigen, die ersten Kapitel des Buches durchgearbeitet haben. Falls Sie einen Befehl nicht verstehen sollten, sehen Sie noch einmal in der alphabetischen Befehlsübersicht nach. Selbstverständlich ist auch Ihr Hersteller-BASIC-Handbuch eine weitere mögliche Informationsquelle.

Bevor man sich in ein größeres Programmiervorhaben stürzt, ist es sehr sinnvoll, zuerst einmal die Anwendung am Schreibtisch zu skizzieren. Das braucht keine detaillierte Ausarbeitung zu sein, sondern sollte die gesamte Anwendung in einzelne Teilaufgaben untergliedern. Der Vorgang der Programmentwicklung kann daher vereinfacht in folgende Phasen unterteilt werden:

1. Problemanalyse und -eingrenzung, Festlegen der Anforderungen und Gliederung in Teilaufgaben

2. Systemdesign
   - Festlegen der Datenstrukturen und Dateien
   - Festlegen der Bildschirmein- und -ausgaben
   - Festlegen der Druckausgaben
   - Festlegen der Verarbeitungslogik und der Ablaufsteuerung

3. Basisroutinen
   - Entwickeln von Zugriffsroutinen für Dateiein- und -ausgaben
   - Ein- und Ausgaberoutinen
   - Entwicklung zentraler Algorithmen und Hilfsfunktionen

4. Schreiben und Test von Programmteilen für jede Teilaufgabe

5. Gesamt- und Praxistest des kompletten Programmsystems

6. Programmpflege und laufende Verbesserung

Die vorgeschlagene Vorgehenssystematik für die Erstellung größerer Programmkomplexe verwendet sowohl Elemente der sogenannten "TOP DOWN-Programmierung" als auch der "BOTTOM UP-Programmierung".

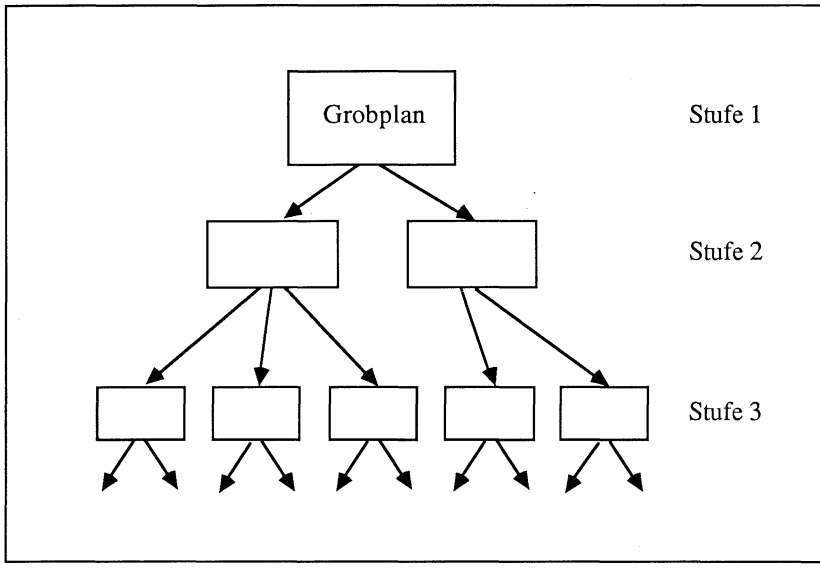



*Abb. 10.1: Schematische Darstellung der TOP DOWN-Programmierung*

Ein schrittweises Aufgliedern der gesamten Programmieraufgabenstellung in Teilaufgaben, die dann vor der eigentlichen Programmierung erst detailliert spezifiziert werden, entspricht der Entwicklungsphilosophie vom Groben zum Feinen (TOP DOWN).

Die Identifizierung allgemein verwendbarer Dienstprogramme für Ein-/Ausgabe auf Bildschirm, periphere Speicher und Drucker, die dann gezielt bei der Realisierung einzelner Anwendungsfunktionen eingesetzt werden, kann als Element der BOTTOM UP-Methode betrachtet werden: schrittweise aus elementaren Programmteilen komplexere Systembestandteile bis zur kompletten Anwendung hin aufzubauen.

Die Mehrzahl der notwendigen Basisroutinen sollte nur einmal erstellt werden und für verschiedene Anwendungen mit allenfalls geringfügigen Änderungen einsetzbar sein. Dies sollte schon bei der ersten Realisierung im Auge behalten werden.

Für eilige Programmierer, die bereit sind, dafür Geld auszugeben, sei auf die Möglichkeit hingewiesen, fertige Sätze von Bildschirmmasken-Routinen, Dateiverwaltungen und Druckerroutinen zu erwerben. Diese werden in Form von Programmgeneratoren, als Datenbanksysteme mit Programmschnittstelle

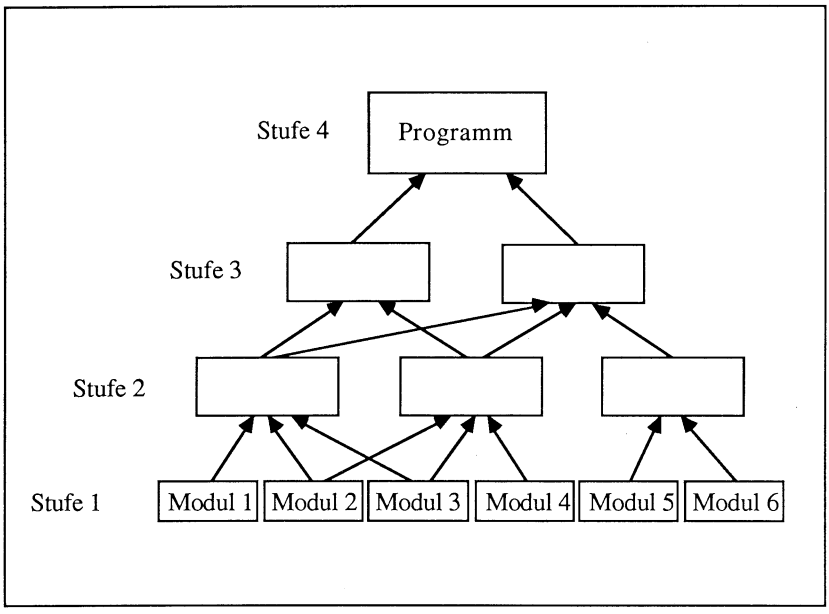



*Abb. 10.2: Schematische Darstellung der BOTTOM UP-Programmierung*

oder als Bestandteil von Spracherweiterungen wie beispielsweise MACRO BASIC oder auch als EXBASIC LEVEL II-Modulsätze zur Verfügung gestellt.

## Modulare Programmierung

Für den Programmieranfänger ist in der Regel jedoch unsicher, wie er am besten Anwendungssysteme während der Entwicklung so in Teilsysteme aufgliedert, daß sich bei der Realisierung einigermaßen vernünftig abgegrenzte Unterprogramme einer optimalen Größe ergeben. Dabei besteht ein Unterprogramm in der Regel aus mehreren bausteinartig verwendeten Programmstücken (Module), die eine bestimmte Funktion realisieren. Zusätzlich enthält dieses Unterprogramm noch einige verbindende sowie unterprogrammspezifische Programmzeilen.

Als Faustregel sollte der Anfänger folgende Kriterien heranziehen:

- ein Modul sollte nur etwa 1 - 2 Bildschirmseiten groß sein;
- mit einem Programm-Modul soll möglichst nur eine Funktion realisiert werden (z. B. Daten sortieren);

- ein Unterprogramm sollte nicht größer als 2 bis 10 Bildschirmseiten sein;
- ein Unterprogramm sollte möglichst wenig Daten von einem anderen Programmteil übernehmen oder an dieses übergeben müssen;
- ein Unterprogramm sollte möglichst universell verwendbar sein.

Man kann aus Unterprogrammen eine ganze Unterprogrammhierarchie aufbauen. Beispielsweise prüft ein generelles Sortierunterprogramm aufgrund verfügbarer Informationen wie Dateigröße, Satzlänge, Anzahl Sortierfelder und Sortierung der Datei, welches Sortierverfahren am besten geeignet ist und ruft dann das Unterprogramm mit dem am meisten geeigneten Sortierverfahren auf.

Die Unterprogramme und das gesamte Programmsystem sollten strukturiert im Sinne des Kapitels 9 programmiert werden. Zusätzlich kann man sich einige Standards überlegen, die die formale Gestaltung der Unterprogramme und Module betreffen: gerahmte Überschrift in Kommentarzeilen, Variablenübersicht usw. Wichtig ist, daß Sie auch nach einiger Zeit und für andere Anwendungen oder auch beim Programmtausch mit anderen Programmierern die schon erstellten Unterprogramme ohne großes Rätselraten direkt nutzbringend einsetzen können.

```
50000 REM*******************************
50010 REM* ROUTINE ZUM EINLESEN VOM    *
50020 REM*       BILDSCHIRM            *
50030 REM*******************************
50040 REM*******************************
50050 REM* DIE VARIABLEN HABEN FOL-     *
50060 REM* GENDE BEDEUTUNG:             *
50070 REM*                             *
50080 REM* NN$ - NACHNAME               *
50090 REM* VN$ - VORNAME                *
50100 REM* ST$ - STRASSE                *
50110 REM* HN  - HAUSNUMMER             *
50120 REM* PL  - POSTLEITZAHL           *
50130 REM* CI$ - STADT                  *
50140 REM*******************************
```

Dies bedingt auch, daß man genau festlegt, wie Daten zwischen Unterprogrammen ausgetauscht werden, welche Namen für Variablen, und zwar für Unterprogramm-interne (lokale) und für generell verwendete Variablen (globale) vergeben werden. Ferner sollte dokumentiert sein, welche Unterprogramme und welche Module dieses Unterprogramm aufrufen und welche anderen Unterprogramme und Module dieses Unterprogramm selbst aufruft.

Diese ganzen Informationen kann man in Kommentarzeilen zu Anfang des entsprechenden Programmteils unterbringen. Hierher gehören auch alle Variablendefinitionen und Zuweisungen von Anfangswerten. Das ist zwar in BASIC nicht erforderlich, da die Verwendung von Variablen auch ohne Anfangswertzuweisung möglich ist, wobei sie dann automatisch den Wert 0 bzw. leere Zeichenkette annehmen. Gleichzeitig kann man vermerken, in welchen Zeilen die Variablen vorkommen.

Weiter ist es sinnvoll, alle innerhalb eines Unterprogramms verwendeten Module, Funktionen und Ansprungstellen im Unterprogrammkopf zu dokumentieren und dabei festzuhalten, in welchen Zeilen sie verwendet werden.

In eine professionelle Dokumentation gehört außerdem:

- Beschreibung der Konfiguration
- BASIC-Version
- Erweiterungen
- Datenbanksystem
- Hauptspeicherkapazität
- Peripheriegeräte
- Bildschirmmodus
  - Text
  - Blockgrafik
  - Mehrfarbengrafik
  - hochauflösende Grafik
  - 80 oder 40 Zeichen pro Zeile

Außerdem könnte noch eine kurze Beschreibung des Ablaufs und der Handhabung des Programms enthalten sein.

Unschön ist allerdings, daß man diese Dokumentation ständig auf den neuesten Stand bringen muß, wenn sich aufgrund von Einfügungen Zeilennummern ändern oder neue Variablen, Unterprogramme oder Funktionen hinzukommen. Daher ist es ratsam, die Zeilennumierung anfänglich in Zehnerschritten vorzunehmen, um Platz für spätere Einfügungen zu lassen.

Neue Variablennamen, Funktionen oder Unterprogramme sollten eigentlich die Ausnahme sein, wenn der Programmentwurf einigermaßen Hand und Fuß hatte.

Als Beispiel können Sie die folgende Verwendungsliste benutzen, mit der der Autor schon seit Jahren arbeitet.

```
0 REM**********************************
1 REM* VERSIONSNUMMER                  *
```

```
2 REM* PROGRAMMNAME                          *
3 REM* AUTOR            DATUM               *
4 REM* BENUTZTE LITERATUR                    *
5 REM* ADRESSE                               *
6 REM* BENOETIGTE HARDWARE                   *
7 REM* BENOETIGTE SOFTWARE                   *
8 REM* BENUTZUNGSHINWEISE                    *
9 REM************************************
9100 REM AUTOMATISCHER VERWENDUNGS-
        NACHWEIS (CROSSREFERENCE)
9105 REM ZNNN BESTIMMTE ZEILENNUMMER
        RZN REFERENZZEILE (AUFTRETEN)
9106 REM KN:=BEZEICHNUNG VON KONSTANTEN
        NZ:=NAECHSTE ZEILE
9107 REM VN:=BEZEICHNUNG VON VARIABLEN
        IW:=ANFANGSWERT, D:=DIMENSION
9108 REM UPGM:=UNTERPROGRAMM
        OPGM:=OVERLAY-PROGRAMMTEIL
        MPGM:=MASCHINENPROGRAMM
9109 REM FNAME:=FUNKTIONSNAME
9110 REM FOLGENDES SCHEMA WIRD VERWENDET
9120 REM ZUWEISUNG DES ANFANGSWERTES:
        VERWENDUNGSNACHWEIS
9150 REM NUMERISCHE KONSTANTE:KNAME:=IW:
        ON 0 GOTO RZ1,RZ2,...,RZN
9160 REM BEISPIEL:
        100 C1=5:ON 0 GOTO 200,300,400
9170 REM 200 X=X1*Y+34
9180 REM 300 Y=Y-X/C1
9190 REM 400 END
9200 REM NUMERISCHE VARIABLE:VNAME:=IW:
        ON 0 GOTO RZ1,RZ2,...,RZN
9250 REM DIM VN1(D):GOSUB 61000 INIT VN1
       :ON 0 GOTO RZ1,RZ2,...,RZN
9300 REM DIM VN2(D1,...,DN):GOSUB 61050
      INIT VN2:ON 0 GOTO RZ1,RZ2,...,RZN
9350 REM ZEICHENKONSTANTE:KNAME$:="IW":
        ON 0 GOTO RZ1,RZ2,...,RZN
9400 REM ZEICHENVARIABLE:VNAME$:="IW":
        ON 0 GOTO RZ1,RZ2,...,RZN
9450 REM DIM VN3$(D1,...,DN):GOSUB 61000
     INIT VN3$:ON 0 GOTO RZ1,RZ2,...,RZN
9550 REM DEF FN FNAME(ARGUMENT):
        ON 0 GOTO RZ1,RZ2,...,RZN
9600 GOTO 9999:REM UNTERPROGRAMME,
```

```
     SPRUNGLISTEN, EXTERNE PROGRAMME
9605 REM MASCHINENPROGRAMME
9701 GOSUB 61000 "PGM1":ON 0 GOSUB
     RZ11,RZ12,...,R1ZN:REM KENNZEICHNUNG
9702 GOSUB 61050 "PGM2":ON 0 GOSUB
     RZ21,RZ22,...,RZ2N:REM KENNZEICHNUNG
9703 .
9704 .
9705 .
9706 GOSUB 61950 "PGMN":ON 0 GOSUB
     RZN1,RZN2,...,RZNN:REM KENNZEICHNUNG
9751 GOTO ZNN1 "LABEL1":ON 0 GOTO
     RZ11,RZ12,...,RZ1N:REM KENNZEICHNUNG
9752 GOTO ZNN2 "LABEL2":ON 0 GOTO
     RZ21,RZ22,...,RZ2N:REM KENNZEICHNUNG
9753 .
9754 .
9755 .
9756 GOTO ZNNN "LABELN":ON 0 GOTO
     RZN1,RZN2,...,RZNN:REM KENNZEICHNUNG
9800 SYS ZNN1 "LABEL1": ON 0 GOTO
     RZ11,RZ12,...,RZ1N:REM KENNZEICHNUNG
9801 SYS ZNN2 "LABEL2": ON 0 GOTO
     RZ21,RZ22,...,RZ2N:REM KENNZEICHNUNG
9803 .
9804 .
9805 .
9806 SYS ZNNN "LABELN": ON 0 GOTO
     RZN1,RZN2,...,RZNN:REM KENNZEICHNUNG
9850 USR ZNN1 "LABEL1": ON 0 GOTO
     RZ11,RZ12,...,RZ1N:REM KENNZEICHNUNG
9851 USR ZNN2 "LABEL2": ON 0 GOTO
     RZ21,RZ22,...,RZ2N:REM KENNZEICHNUNG
9853 .
9854 .
9855 .
9856 USR ZNNN "LABELN": ON 0 GOTO
     RZN1,RZN2,...,RZNN:REM KENNZEICHNUNG
9900 DLOAD "OPGM1": ON 0 GOTO
     RZ11,RZ12,...,RZ1N:REM KENNZEICHNUNG
9901 DLOAD "OPGM2": ON 0 GOTO
     RZ21,RZ22,...,RZ2N:REM KENNZEICHNUNG
9903 .
9904 .
9905 .
```

```
9906 DLOAD "OPGMN": ON 0 GOTO
     RZN1,RZN2,...,RZNN:REM KENNZEICHNUNG
```

Die hier vorgestellte Fassung bezieht sich auf das Hauptprogramm. Daher tauchen hier auch alle global verwendeten Konstanten, symbolische Namen für Bildschirmsteuercodes oder wichtige Systemadressen auf sowie alle global verwendeten Variablen, Unterprogramme und Funktionen.

Für die Programmodule und Unterprogramme wird man in der Regel mit einem wesentlich kleineren Crossreferenz-Programmabschnitt auskommen. In vielen Fällen, wo nur sehr wenige oder nur sehr kurze Programmteile das Hauptprogramm ergänzen, ist es denkbar, diese Verwendungshinweise ebenfalls im Hauptprogramm zu führen.

Da dieser Crossreferenzteil bei vielen Programmen ähnlich sein wird, ist es sinnvoll, ihn auf einer Diskette oder Kassette abzuspeichern und zu Beginn einer neuen Programmerstellung jeweils wieder in den Rechner zu laden.

Alle Unterprogramme und Module sollten Sie ebenfalls jeweils einzeln auf eine speziell dafür reservierte Diskette abspeichern und sich so allmählich eine Programmbibliothek aufbauen. Bei Bedarf können diese Programmteile dann mit dem MERGE-Befehl in ein zu erstellendes Programm eingefügt werden. Wer eine BASIC-Version verwendet, die diesen Befehl nicht enthält, findet im alphabetischen Teil des Buches unter dem Stichwort APPEND Hinweise darauf, wie man den Befehl auch simulieren kann.

Spätestens wenn man mit dem MERGE-Befehl arbeiten will, fällt einem auf, daß man die Numerierung der einzelnen Unterprogramme unter Umständen nicht geeignet gewählt hat. Deshalb ist es sinnvoll, sich schon von vornherein Gedanken darüber zu machen, in welchem Zeilennummernbereich man die einzelnen Programmteile unterbringen will. Sicher ist es nicht falsch, wenn Sie Module beispielsweise erst ab 50000 beginnen lassen, damit es beim MERGE-Befehl keine Probleme mit sich überschreibenden Zeilennummern gibt.

*Hinweis:* Der CONCAT- und auch der APPEND-Befehl, die ein Anfügen von Dateien bzw. von Daten an eine bestehende Datei auf der Diskette ermöglichen, erlauben nicht ohne weiteres, BASIC-Programme aneinanderzuhängen, obwohl ansonsten PRG-Dateien als sequentielle Dateien geöffnet und behandelt werden können. Das liegt unter anderem daran, daß dabei bestimmte Informationen wie Programmlänge, Ladeadresse usw., die sich zu Beginn der ersten PRG-Datei befinden, nicht geändert bzw. am Beginn der zweiten Datei nicht entfernt werden.

## Zusammenfassung Programmentwicklung

Bevor wir die Besprechung der Programmentwicklung abschließen, geben wir eine kurze Zusammenfassung des bisher Gesagten:

1. Entwurf am Schreibtisch
2. Verwendung einer Modul- oder Unterprogrammbibliothek
3. Nach Möglichkeit Verwendung eines Toolkits
4. Crossreferenz und Dokumentation im Programmkopf
5. Strukturierung des Programms und der Unterprogramme entsprechend Kapitel 9.

Manchem Leser mag noch unklar sein, worin der Unterschied zwischen einem Modul und einem Unterprogramm besteht. Ein Modul ist in der Regel wesentlich kürzer als ein Unterprogramm, vielleicht so groß wie eine Bildschirmseite, und ist noch knapper dokumentiert.

Ein Unterprogramm kann bis zu 10 Bildschirmseiten lang sein und kann eine ganze Reihe Module verwenden. Falls Ihr Programmsystem nun zu groß wird, um in den Hauptspeicher zu passen, muß die sogenannte Overlay-Technik (Überlagerung) angewendet werden. Das heißt, daß das gesamte Programm jeweils nur teilweise geladen wird und bei Bedarf die Programmstücke untereinander ausgetauscht werden müssen.

Diese teilgeladenen Programmstücke müssen natürlich alleine ablauffähig sein, d. h., es müssen alle aufgerufenen Unterprogramme und verwendeten Variablen sowie benötigte Routinen zur Ein- und Ausgabe auf Bildschirm, Speichergeräte und Drucker vorhanden sein. Ein Modul kann in mehreren Unterprogrammen verwendet werden.

Eine spezielle Klasse von Modulen stellen die Programmzeilen dar, in denen Benutzerfunktionen definiert werden. Aus einer zentralen Unterprogrammbibliothek werden diejenigen in das jeweilige Unterprogramm kopiert, die dort benötigt werden.

Ein für sich lauffähiges Programmstück besteht aus einem oder mehreren Unterprogrammen. Also kann man sagen, daß ein Unterprogramm im Gegensatz zu einem Modul prinzipiell die kleinste eigenständige lauffähige sinnvolle Programmeinheit darstellt.

## Overlay-Technik

Falls ein Unterprogramm in Overlay-Technik eingesetzt werden soll, müssen bei der Erstellung alle benötigten Module physisch hineinkopiert werden, die

sonst nur insgesamt einmal im ganzen Programm, und zwar üblicherweise alle in einem großen Block am Programmende vorkommen.

Wie funkioniert nun ein Overlay? Es gibt prinzipiell zwei Arten von Overlays, allerdings mit allen möglichen Zwischenformen:

1. Das kalte Nachladen mit Variablenverlust (Laden im Direktmodus mit LOAD bzw. DLOAD oder im simulierten Direktmodus): Die nachgeladenen Programme überschreiben sowohl das im Speicher befindliche Programm als auch die Variablenbereiche. Das nachgeladene Programm wird nicht automatisch gestartet.

2. Das warme Nachladen mit Variablenerhalt (Laden im Programmodus mit LOAD bzw. DLOAD): Die nachgeladenen Programme überschreiben nur das im Speicher befindliche Programm ganz oder teilweise. Die Variablenbereiche bleiben dabei erhalten. Das nachgeladene Programm wird automatisch gestartet.

Die erste Form ist von geringer Bedeutung. Dadurch, daß beim Nachladen alle bisherigen Variablenwerte verlorengehen, besteht eine starke Einschränkung. Hier kann man nur so verfahren, daß man die einzelnen Overlay-Ebenen so wählt, daß sie keine gemeinsamen Werte benutzen, oder die Daten müssen beim Verlassen der einen Ebene in einer temporären sequentiellen Datei zwischengespeichert werden und nach Laden des nächsten Unterprogramms wieder eingelesen werden. Das ist, wie man sich denken kann, sehr aufwendig.

Außer BASIC 2.0 bieten alle Commodore-BASIC-Versionen den Befehl DLOAD, mit dem nun das warme Nachladen realisiert werden kann. Das folgende kleine Programm würde also das Nachladen bewirken:

```
100 A=10
110 DLOAD "TEST"
```

In BASIC 2.0 heißt die Zeile 110:

```
110 LOAD "TEST",8
```

So unproblematisch, wie es vielleicht aussieht, ist die Overlay-Technik eigentlich nicht. Man muß bedenken, daß im Speicher hinter jedem Programm der Teil folgt, der die Daten aufnimmt. Wenn nun das aufrufende Programm kleiner ist als das aufgerufene, so ragt das zweite in den Variablenspeicher des ersten hinein, und die Variablenwerte sind verloren.

Beim warmen Nachladen muß also beim Start des ersten Unterprogramms mindestens soviel Speicherplatz reserviert werden, wie das größte Unterprogramm der ganzen Overlay-Struktur benötigt.

Wie können Sie nun sicherstellen, daß das erste Programm auch das größte ist. Mit PRINT FRE(0) können Sie nach dem Laden eines Programms immer feststellen, wieviel Speicherplatz noch übrig ist. Laden Sie bitte alle Programme, die Sie in Ihrer Overlay-Struktur verwenden wollen, hintereinander, und überprüfen Sie jedesmal den Speicherplatz mit PRINT FRE(0). Das Programm, das den wenigsten Platz übrig läßt, muß das größte sein.

Nun kommt es nur noch darauf an, das erste Unterprogramm so aufzublasen, daß es diese Größe erreicht. Natürlich könnten Sie das möglicherweise mit dem Einfügen von Dummy-Befehlen erreichen. Aber es gibt eine Methode, die wesentlich professioneller und genauer arbeitet. Dazu müssen Sie die Zeiger abfragen und ändern, die auf Anfang und Ende des größten gefundenen Programms zeigen.

Diese Speicherstellen können von Rechner zu Rechner verschieden sein. Beim C16, C116, Plus/4 und C64 stehen die Endadressen in den Speicherstellen 43 und 44, beim 8032 handelt es sich um 42 und 43. Beim C128 finden Sie die Endadresse für den BASIC-Text mit:

```
PRINT PEEK(47)
```
Ergebnis: niederwertiges Adreßbyte

```
PRINT PEEK(48)
```
Ergebnis: höherwertiges Adreßbyte

Damit Sie nun die Längen aller Ihrer Overlay-Programmteile auf die Länge des längsten Unterprogramms bringen können, kopieren Sie bitte das folgende Hilfsprogramm an das Ende Ihres Overlay-Programmteils als Unterprogramm. Dann starten Sie es mit GOSUB 10000. Es gibt jeweils die Endadresse des BASIC-Quelltextes an. Die größte so ermittelte Endadresse mit größtem Highbyte-Wert (bei Gleichheit des Highbytes zusätzlich den größten Lowbyte-Wert als Auswahlkriterium verwenden) muß fortan als künstlicher Programmende-Wert gesetzt werden.

```
10000 REM*********************************
10010 REM ANZEIGE DER FREISPEICHERBEREICHE
10020 REM UND POKE-WERTE FUER OVERLAY C128
10030 REM*********************************
10040 LA=47 :REM LOWBYTE BASIC-TEXTENDE
10050 HA=48 :REM HIGHBYTE BASIC-TEXTENDE
10060 REM*********************************
10070 REM DEFINITION DER FUNKTIONEN
10080 REM*********************************
10090 REM HIGHBYTE LAENGE DES PROGRAMMS
10100 DEF FNHA(I)=INT((-58109*(I=0)
```

```
                -64256*(I=1)-FRE(I))/256)
10110 REM LOWBYTE LAENGE DES PROGRAMMS
10120 DEF FNLA(I)=(-58109*(I=0)
      -64256*(I=1)-FRE(I))-256*FNHA(I)
10130 REM HIGHBYTE ENDADRESSENERMITTLUNG
      FUER NACHFOLGENDES OVERLAY-PROGRAMM
10140 DEF FNHP(I)=FNHA(I)-28*(I=0)
     :REM ANDERE RECHNER FNHP(I)=PEEK(202)
10150 REM LOWBYTE ENDADRESSENERMITTLUNG
      FUER NACHFOLGENDES OVERLAY-PROGRAMM
10160 DEF FNLP(I)=FNLA(I)-1*(I=0)
     :REM ANDERE RECHNER FNLP(I)=PEEK(201)
10170 REM*********************************
10180 REM       AUSGABE DER WERTE
10190 REM*********************************
10200 PRINTTAB(80)"FREIBEREICH BANK 0: ";
10210 PRINT"WERT FUER HIGHBYTE"
10220 PRINT 58109-FNHA(0)*256-FNLA(0);
10230 PRINTTAB(20)"POKE ";HA;",";FNLP(0)
10240 PRINTTAB(20)"WERT FUER LOWBYTE"
10250 PRINTTAB(20)"POKE ";LA;",";FNLP(0)
10260 PRINTTAB(80)"FREIBEREICH BANK 1:"
10270 PRINT 64256-FNHA(1)*256-FNLA(1)
10280 REM MAXIMALES HP: POKE HA,FNHP(0)
10290 REM MIT MAXIMALEM LP:POKE LA,FNLP(0)
```

Dabei wird natürlich dieses Unterprogramm bei der Längenermittlung mit ausgewertet. Betrachten Sie dies als Sicherheitsmarge, die noch, falls es einmal knapp werden sollte, abgezogen werden kann.

Nun laden Sie wieder Ihr erstes Unterprogramm und versehen dieses mit den Adressen des größten Programms:

```
POKE 47,Lowbyte für neues künstliches Programmende

POKE 48,Highbyte für neues künstliches Programmende
```

Nun glaubt der Rechner, das erste Programm benötige soviel Platz, wie die beiden Adressen angeben. Speichern Sie das Programm nun, ohne es vorher zu starten, ab, sonst können Sie es nur noch im aufgeblasenen Zustand abspeichern. Außerdem können Änderungen im Programmtext die Werte des Endezeigers verändern.

Das gleiche gilt auch für die anderen Unterprogramme. Wenn Sie ein Unterprogramm in Overlay-Technik geladen haben, dürfen Sie es nie abspeichern,

da sonst der ganze Speicherbereich bis zum Programmende-Zeiger des ersten und nun größten Programms abgespeichert würde. So kann aus einer Mücke ein Elefant werden. Beim Testen und Ändern müssen Sie die Unterprogramme also immer direkt laden und abspeichern.

Achten Sie in der Testphase auch darauf, daß unter Umständen ein bisher kleines Programm das größte werden kann, was dann wieder bei der Platzreservierung berücksichtigt werden muß. Vielleicht lassen Sie zumindest in der Erstellungsphase beim ersten Unterprogramm noch etwas Spielraum nach oben, damit Sie die Speicherplatzzuweisung nicht jedesmal zu ändern brauchen.

Wenn ein Overlay-Programm nachgeladen wird, wird es automatisch gestartet. Alle Befehle im aufrufenden Programm, die nach dem Lade-Befehl folgen, werden also nie ausgeführt:

```
100 A=10
110 DLOAD "TEST"
120 PRINT "A"
```

Zeile 120 wird nie erreicht, da nach dem Laden des Programms TEST dieses gleich gestartet wird.

Diese einfachste Form des Programm-Overlays, die wir hier vorgestellt haben, bedingt natürlich, daß alle Overlay-Programmteile in ihrem Kopfteil ein gemeinsames übergeordnetes Steuerprogramm enthalten, das die Gesamtablaufsteuerung übernimmt. Außerdem muß peinlich genau darauf geachtet werden, daß alle gemeinsam benutzten Variablen nur im Hauptprogramm initialisiert und dimensioniert werden.

Durch entsprechende Veränderung des Anfangszeigers für den BASIC-Text, der sich beim C128 an den Adressen 45 und 46 befindet, kann auf das zeitaufwendige Mitladen und Mitabspeichern des Hauptprogramms natürlich verzichtet werden. Durch derartige Manipulation ist es sogar möglich, den Overlay-Teil in mehrere Teile zu splitten, die getrennt voneinander geladen und abgespeichert werden können. Dies führt zu wesentlich kürzeren Wartezeiten und erlaubt noch komplexere Überlagerungsstrukturen.

Die Verwendung der Overlay-Technik ist in Abb. 10.3 schematisch dargestellt.

Diese interessante und schon sehr fortgeschrittene Programmiertechnik wollen wir an dieser Stelle nicht mehr weiter verfolgen.

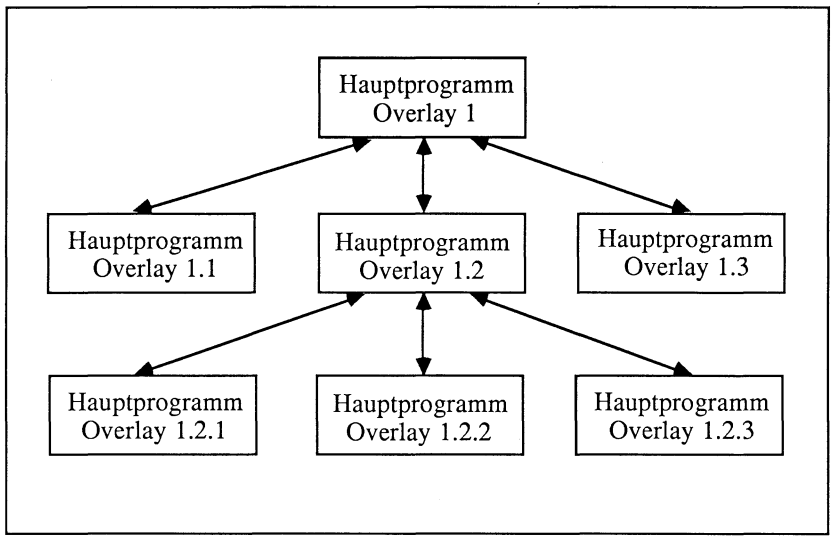



*Abb. 10.3: Schematische Darstellung der Overlay-Technik*

## Prototyping und Bildschirm-Layout

Die Commodore-Rechner bieten uns eine ebenso einfache wie effektive Methode, einfache Bildschirmmasken zu entwerfen.

Schreiben Sie einfach auf den Bildschirm, ohne die RETURN-Taste zu betätigen. Benutzen Sie statt dessen nur die Cursor-Steuerungstasten oder die RETURN-Taste mit niedergedrückter SHIFT-Taste. Ein Drücken der RETURN-Taste würde ja den Inhalt der aktuellen Eingabezeile an den BASIC-Interpreter übergeben, der dies im besten Falle mit READY oder im Normalfall mit einer Fehlermeldung beantwortet. Das kann das bereits aufgebaute Bild empfindlich durcheinanderbringen.

Wir schalten zusätzlich mit ESC + "M" das Bildschirmrollen ab, damit nicht durch versehentliches Drücken von RETURN oder Überschreiten der 25. Zeile die oberste Zeile aus dem Schirm hinausgeschoben wird. Wir lassen beim Bildschirmentwurf außerdem die 5 ersten Spalten und die letzte Zeile frei.

Wenn der Entwurf fertig ist, fahren wir, von unten nach oben fortschreitend, die Zeilen ab und setzen eine Zeilennummer, gefolgt von einem Fragezeichen

plus Anführungszeichen, an den Anfang und drücken RETURN. Auch wenn anschließend die aktuelle Zeile von READY. überschrieben wird, macht das gar nichts, denn die vorige Zeile wurde bereits richtig in das Programm übernommen.

Dabei müssen natürlich abfallende Zeilennummern vergeben werden, sonst erhalten wir eine an der untersten Zeile gespiegelte Bildschirmmaske.

**Beispiel**

```
          1. DATEIVERWALTUNG

          2. DATENEINGABE/-AENDERUNG/-LOESCHEN

          3. ABFRAGE/REPORTERSTELLUNG

          4. DATENAUSTAUSCH

          5. ENDE

10000? BS-LOESCHEN
10010?
10020?
10030?"1. DATEIVERWALTUNG
10040?
10050?"2. DATENEINGABE/-AENDERUNG/-LOESCHEN
10060?
10070?"3. ABFRAGE/REPORTERSTELLUNG
10080?
10090?"4. DATENAUSTAUSCH
10100?
10110?"5. ENDE
10120?
10130?
```

Sie können natürlich auch Blockgrafikzeichen verwenden, die direkt über die Tastatur erreichbar sind. In unserem Beispiel stellt das Nummernzeichen den gepunkteten Block, der auf der +-Taste liegt, dar:

```
10010?"######                    ######
10020?"#                              #
10030?"#   1.                         #
```

Es ist jedoch zu empfehlen, die explizite Angabe der Blockgrafikzeichen nur für einen ersten Entwurf zu verwenden und statt dessen später die im Buch unter dem Stichwort CHR$ vorgeschlagenen Zeichenvariablen zu verwenden.

Die Programme werden so übersichtlicher, und in der Regel wird Speicherplatz gespart, da bestimmte längere Zeichenketten mehrfach verwendet werden. Soll das Programm wahlweise mit einem 40- oder 80-Spalten-Bildschirm arbeiten können, so ist dann auch leichter an einer zentralen Stelle ein Parameter geändert, etwa so:

```
100 GP$(1)=CHR$(166)  :REM GEPUNKTETER VOLLBLOCK
110 SP=RWINDOW(2)     :REM ABFRAGE BILDSCHIRMGROESSE
120 FOR I=1 TO SP
130 : BQ(1)=BQ(1)+GP$(1)  :REM BALKEN QUER
140 NEXT I
```

Die Zeilen 10020 und 10030 könnten dann so aussehen:

```
10020 PRINTTAB(0)GP$(1);TAB(SP)GP$(1)
```

Ein Vorgehen in dieser Weise, bei dem man erst einige wichtige Komponenten des Programms skizziert und erst, wenn dieser auf dem Rechner bereits ablauffähige und präsentierbare Entwurf zur Zufriedenheit aller ausfällt, darangeht, eine professionellere Version des Programms zu erstellen, nennt man Prototyping.

Wenn Sie für die Menüpunktnummern mit Hilfe der Rvs On- und Rvs Off-Taste eine invertierte Darstellung gewählt haben oder Textteile unter Verwendung der Farbcode-Tasten gefärbt haben, werden Sie feststellen, daß dies bei der Umwandlung in Programmzeilen und anschließendem Ausdruck verlorengegangen ist. Das liegt daran, daß PRINT einfach Färbungen und Invertierungen in Zeichenketten ignoriert. Sie müssen also bei der Übernahme der Bildschirm-Layouts in Programmzeilen entsprechende Steuercodes mit Hilfe der in Kapitel 8 unter CHR$ aufgeführten Variablen in die Zeichenkette einfügen.

Falls beispielsweise die Auswahlziffern vor den einzelnen Menüpunkten invers dargestellt werden sollen, muß z. B. die Zeile 10030 wie folgt umgeformt werden:

```
10030 PRINT GP$(1);TAB(10)ZR$;"1";ZN$;
            "Auswahltext";TAB(SP)GP$(1)
```

Im Ein-/Ausgabebereich wird für komplexere Ein-/Ausgabeabläufe häufig eine Simulation mit READ und DATA-Zeilen als einfaches und schnelles Ver-

fahren für eine Prototyp-Entwicklung verwendet. Dabei lassen sich sowohl rein sequentielle Dateizugriffe als auch relative Dateien mit RESTORE Zeilennummer nachbilden. Hierbei wird allerdings das Schreiben in DATA-Zeilen von BASIC nicht direkt unterstützt. Eine Alternative, die Lesen und Schreiben ohne Klimmzüge ermöglicht, ist die Verwendung von Feldvariablen (indizierten Variablen).

## Benutzerführung und Menütechnik

Programme werden zuallererst, ob gerechtfertigt oder nicht, nach ihrer optischen Präsentation auf dem Bildschirm beurteilt. Der beste Algorithmus und die raffinierteste Dateiverwaltung finden keine Würdigung, wenn der Programmablauf schwer verständlich bleibt und die Benutzereingaben kompliziert und umständlich vonstatten gehen.

Die am häufigsten angewendete Technik zur Gestaltung von Benutzerdialogen ist die Menütechnik, die dem Anwender mehrere Auswahlpunkte auf einem Ausgabebild anbietet. Eine entsprechende Eingabe führt zur Ausführung der unter dem Menüpunkt aufgeführten Funktion.

Ein solches Menübild sollte nie zu viele Punkte enthalten, und es muß unbedingt auf einen einfachen Bildaufbau geachtet werden, der eine rasche Orientierung erlaubt.

Ein Beispiel für ein solches Auswahlmenü finden Sie in Kapitel 8 unter dem Stichwort ON...GOSUB. Dort wird vorgeschlagen, das Programm so zu erweitern, daß zusätzlich Benutzereingaben zur Ablaufsteuerung möglich werden. Weitere Auswahlpunkte würden aber schnell zur Überfrachtung dieses Menübildes führen. Was liegt also näher, als diesem Hauptmenü ein weiteres Menübild mit weiteren Auswahlpunkten folgen zu lassen. Das läßt sich relativ einfach bewerkstelligen. Fügen Sie folgende Zeilen in das Programm ein:

```
305 PRINT DF$
306 PRINT TAB(14)"ADDITION"
307 GOSUB 1000 UNTERMENUE
445 PRINT DF$
446 PRINT TAB(14)"SUBTRAKTION"
447 GOSUB 1000 UNTERMENUE
595 PRINT DF$
596 PRINT TAB(14)"MULITPLIKATION"
597 GOSUB 1000 UNTERMENUE
735 PRINT DF$
736 PRINT TAB(14)"DIVISION"
737 GOSUB 1000 UNTERMENUE
```

```
1000 REM*********************************
1010 REM*       UNTERMENUE                *
1020 REM*********************************
1060 PRINT:PRINT
1070 PRINT TAB(10)
     "1. ENDE NACH 10 RICHTIGEN ANTWORTEN"
1080 PRINT TAB(10)
     "2. ENDE NACH 10 ANTWORTEN"
1090 PRINT TAB(10)
     "3. ENDE NACH EINGABE VON E"
1100 PRINT TAB(10)
     "4. ZURUECK ZUM HAUPTMENUE"
1110 PRINT:PRINT:PRINT:PRINT
1120 INPUT"WAEHLEN SIE EINE OPTION (1 - 4)"
     ;WAHL
1130 IF WAHL<1 OR WAHL>4 THEN PRINT CHR$(7)
     :GOTO 1120
1140 ON WAHL GOSUB 1280,1380,1480,1580
1150 RETURN
```

Sie können jetzt durch Einfügen von

```
1145 GOSUB 2000 UNTERMENUE-EBENE 2
```

geschickt eine weitere Untermenü-Ebene einführen. Es können zwar beliebig viele Menüs aneinandergehängt werden, aber mehr als 5 Ebenen werden unübersichtlich. Selbstverständlich muß das Programm entsprechend den neuen Unterprogrammaufrufen in Zeile 1140 und bei weiteren Menüsprung-Verteilern mit Unterprogrammen versehen werden. Außerdem muß auch die Verarbeitung im Hauptprogramm die neuen Optionen berücksichtigen.

Neben Auswahlmenüs, die nur die Auswahl eines Menüpunktes erlauben, ist es natürlich auch sinnvoll, Menüs einzubeziehen, die eine Mehrfachauswahl gestatten.

Für unsere Anwendung könnte das z. B. im folgenden weiteren Untermenü der Fall sein:

```
AUSWAHL DES RECHENMODUS

1. RECHENOPERATIONEN MIT 3STELLIGEN ZAHLEN
   (VOREINSTELLUNG 2STELLIGE ZAHLEN)

2. ABSPEICHERN DER TREFFERRATE
   (VOREINSTELLUNG KEIN ABSPEICHERN)
```

```
3. ZWEI-PERSONEN-WETTBEWERB MIT
   ABWECHSELNDER EINGABE
   (VOREINSTELLUNG EINE PERSON)

STELLEN SIE BITTE DEN VERARBEITUNGSMODUS
DURCH ANGABE VON 1, 2 ODER 3 EIN, UND
BEENDEN SIE IHRE EINGABE DURCH RETURN.
```

Eine noch größere Flexibilität erreichen Sie natürlich, wenn Sie mehrere Eingabefelder, und zwar pro Menüpunkt immer eins, erlauben, die die direkte Eingabe von Werten zulassen.

**Beispiel**

```
RECHENMODUS

1. WIEVIELSTELLIGE ZAHLEN (1 - 4):____

2. WIE VIELE AUFGABEN     (1 -99):____

3. WIE VIELE PERSONEN     (1 - 4):____

4. ANZEIGEN DER LOESUNG     (J/N):____

BEENDEN SIE IHRE EINGABE DURCH DRUECKEN
DER LEERTASTE.
```

Für die Gestaltung eines solchen Menüs verwendet man am besten Bildschirmfenster für die Eingabebereiche, um eine kontrollierte und korrekte Eingabe zu gewährleisten. Dazu setzt man den Cursor durch einen entsprechenden INPUT-Befehl und PRINT mit Cursor-Steuertasten (siehe CHR$-Variablen in Kapitel 8) hinter den ersten Menüpunkt und definiert um den Cursor herum ein Fenster, etwa in der Größe der maximal zulässigen Eingabe.

Wenn diese Eingabe mit RETURN oder Cursor runter oder hoch beendet wird, wird der Cursor hinter den nächsten Auswahlpunkt gesetzt, dort eine INPUT-Anweisung aktiviert und ebenfalls ein Fenster definiert.

Durch Drücken der Leertaste wird dann die Eingabe beendet. Da eventuell vergessen werden kann, einzelne Eingabe durch Drücken von RETURN zu übernehmen, kann zur Verbesserung die in diesem Buch in Kapitel 4 vorgestellte Routine zum direkten Lesen des Bildschirms verwendet werden. Dann kann auf den INPUT-Befehl und auf das Betätigen der RETURN-Taste

verzichtet werden. Nach Drücken der Leertaste werden einfach alle vier Eingabefelder ausgelesen.

Eine weitere Verbesserung kann darin bestehen, die Eingabefelder schon mit Werten vorzubesetzen, die dann wahlweise übernommen oder verändert werden können.

Im Zusammenhang mit der Verwendung von Eingabefeldern ist es jedoch unerläßlich, die Eingaben auf Zulässigkeit und Konsistenz zu überprüfen.

Deshalb ist es sinnvoll, am unteren Rand des Eingabebildes mehrere Zeilen für Fehlermeldungen vorzusehen. Bei Fehleingaben wird einfach das Eingabebild erneut angezeigt, und ein passender Hinweistext erscheint im unteren Bildschirmbereich. Zusätzlich kann das fehlerhafte Eingabefeld und die falsche Eingabe invertiert oder in Rot angezeigt werden.

Dieser untere Zeilenbereich kann auch benutzt werden, um Hilfstexte einzublenden, die der Anwender über eine Funktionstaste oder durch die Eingabe eines Fragezeichens gezielt anfordern kann. Für umfangreichere Erklärungen empfiehlt es sich aber, eine neue Bildschirmseite mit dem Hilfstext auszugeben. Dabei sollten jedoch alle bisher gemachten Eingaben erst gelesen werden, damit beim anschließenden Wiederherstellen des Eingabebildes diese Eingaben anstelle der voreingestellten Werte angezeigt werden können.

Wenn Sie der einzige Benutzer des Programmsystems sind oder wenn jemand aufgrund häufiger Benutzung mit den Menüfolgen schon recht vertraut ist, erscheint eine starr vorgeschriebene Abfolge über mehrere Menüs recht lästig und zeitaufwendig. Es ist daher zweckmäßig, eine Durchwahl über mehrere Menü-Ebenen hinweg vorzusehen, um den Auswahlprozeß abzukürzen.

Noch schneller geht es natürlich, wenn man in einer besonderen Zeile eine ganze Kette von Eingaben, die durch ein besonders festgelegtes Trennzeichen untereinander begrenzt werden, vorsieht. So kann man mit einer Eingabe dem Programm alle benötigten Parameter übergeben. Dies erfordert jedoch eine manchmal aufwendige Routine, um den Inhalt der Zeichenkette zu interpretieren.

Ein schönes, aber anspruchsvolles Beispiel für eine solche Eingabezeile könnte sein, einem Funktionen-Plotprogramm über eine Eingabezeile die Funktionsformel und die Parameter für die Darstellung zu übergeben.

## Austesten von Programmen

Die Fehler, die Ihnen bei der Programmerstellung als erstes begegnen werden, sind die SYNTAX-Fehler. Dabei gibt die Meldung

```
?SYNTAX ERROR IN Zeilennummer
```

genau an, in welcher Zeile sich dieser Tippfehler befindet. Sie können sich die entsprechende Zeile mit

```
LIST Zeilennummer
```

oder beim C128 durch Drücken der HELP-Taste und beim C16, C116 und Plus/4 durch Drücken der Funktionstaste F4 anzeigen lassen. Bei den Rechnern mit der eingebauten HELP-Funktion wird nun sogar das falsche Wort invers oder blinkend dargestellt.

Diese Art von Fehlern läßt sich also relativ leicht finden und beheben. Was ist aber, wenn Ihr Programm trotzdem noch nicht die erwarteten Ergebnisse hervorbringt.

Logische Fehler können der Grund dafür sein, und leider sind sie wesentlich schwerer zu finden als Tippfehler, und nicht nur deshalb, weil der BASIC-Interpreter sie nicht entdecken und anzeigen kann. Dennoch gibt es eine ganze Reihe von Hilfsmitteln, besonders im BASIC 7.0 sind viele neue Fehlersuchbefehle hinzugekommen.

Ein Programm, das sich noch in der Testphase befindet, sollte immer anders aussehen als das endgültige. An geeigneten Stellen kann man Zwischenwerte für Variablen ausdrucken lassen, um auf diese Art und Weise Berechnungen nachzuvollziehen. Vergessen Sie auch nie, nach einem unverhofften Programmabbruch, mit

```
PRINT Variablenname
```

die Werte aller Variablen abzufragen, um anhand von unsinnigen Größen womöglich Rückschlüsse zu ziehen.

Sie können auch während des Programmlaufs die RUN/STOP-Taste drücken und sich die Werte von Variablen mit PRINT ansehen und sie sogar verändern. Wenn Sie danach

```
CONT
```

eingeben (continue - fortfahren), wird das Programm ordnungsgemäß zu Ende geführt.

Diese Methode ist allerdings sehr unexakt, da man nie genau abschätzen kann, in welcher Zeile das Programm mit seiner Bearbeitung nun angelangt ist. Besser ist es dann schon, systematisch in allen Programmzweigen den Befehl

```
STOP
```

einzusetzen. Das Programm reagiert jedesmal, wenn es auf einen STOP-Befehl trifft, mit der Meldung:

```
BREAK IN Zeilennummer
```

und man kann nachvollziehen, welcher Programmzweig durchlaufen wurde. Auch nach der Programmunterbrechung durch den STOP-Befehl gibt es die Möglichkeit, Variablenwerte anzusehen und zu verändern. Mit CONT kann das Programm nach dem STOP wieder fortgesetzt werden.

In BASIC 3.5 und 7.0 gibt es noch eine genauere Methode der Programmablaufverfolgung, den Befehl TRON. Wenn TRON (TRace ON) aktiv ist, erscheinen auf dem Bildschirm alle Zeilennummern des Programms in eckigen Klammern in der Reihenfolge ihrer Abarbeitung.

**Beispiel:**

```
10 A=0
20 FOR I=1 TO 5
30 A=A+1
40 PRINT A
50 NEXT I

TRON

READY.
RUN
[10][20][30][40] 1
[50][30][40] 2
[50][30][40] 3
[50][30][40] 4
[50][30][40] 5
[50]
READY.
```

Der Trace-Modus läßt sich mit dem Befehl TROFF (TRace OFF) wieder abschalten.

Wenn Sie ein Programm nicht mehr ganz von Anfang testen wollen, können Sie auch eine Zeilennummer angeben, ab der die Programmausführung beginnen soll. Dabei gibt es die beiden möglichen Befehle im Direktmodus:

```
RUN Zeilennummer
```

und

```
GOTO Zeilennummer
```

Beide beginnen die Programmausführung an der angegebenen Zeilennummer. Der Unterschied besteht darin, daß bei der Eingabe des Befehls RUN alle Variablenwerte gelöscht werden, während sie bei GOTO erhalten bleiben.

Auch der Befehl GOSUB kann im Direktmodus verwendet werden, um gezielt Unterprogramme auszutesten.

Wenn man vermeiden will, daß die Programmausführung bei Auftreten eines Fehlers abgebrochen wird, kann man in BASIC 3.5 und 7.0 mit dem Befehl TRAP eine Fehlerfalle bauen. Durch die Angabe einer Zeilennummer hinter TRAP:

```
100 TRAP Zeilennummer
```

wird bei Auftreten eines Programmfehlers zu der angegebenen Zeilennummer verzweigt. Dort kann man dann eine Fehlerroutine einbauen, die möglicherweise den Fehler dokumentiert und behandelt. Hilfreich sind dabei die Variablen ER - Fehlernummer, EL - Fehlerzeile und ERR$ - Fehlermeldung.

Den Abschluß einer solchen Fehlerroutine bildet stets der Befehl

```
RESUME Zeilennummer
```

der angibt, bei welcher Zeilennummer der Programmbetrieb weitergehen soll.

## Zusammenfassung "Programme austesten"

Wir haben nun eine ganze Reihe Techniken kennengelernt, wie man ein Programm zum Laufen bringen kann. Nur eigentlich genügt es nicht, wenn ein Programm einmal läuft. Es sollte immer laufen. Immer bedeutet, daß alle Programmzweige in der Testphase mindestens einmal durchlaufen wurden. Immer bedeutet auch, daß unsinnige Eingaben nicht zum Absturz des Pro-

gramms führen. Also geben Sie beim Testen auch mal Werte ein, die "eigentlich" gar nicht vorkommen dürfen. Wie Sie Ihr Programm absturzsicher machen und gegen unqualifizierte Eingaben schützen, erfahren Sie unter anderem auch in Kapitel 4.

# Kapitel 11

# Übertragbarkeit von BASIC-Programmen

Von besonderem Interesse für Umsteiger ist immer wieder die Lösung des Problems: Wie bringe ich Programme und Daten von einem Gerät auf das andere. Prinzipiell gibt es dabei nicht viele Probleme, wenn man einige Dinge beachtet.

## Kompatibilität

Viele BASIC-Anweisungen der einzelnen Versionen 2.0, 4.0, 3.5 und 7.0 sind identisch. Natürlich ist das BASIC 7.0 das umfangreichste und komfortabelste. Unterschiede, die beim Übertragen vom einem zu einem anderen System Probleme bereiten, werden hauptsächlich durch die Farben, Speicherbelegung, hochauflösende Grafik und Sprites verursacht.

### Programmübertragung vom C64 oder VC20 auf Rechner der 2000-, 3000-, 4000- und 8000-Serie

Wenn keine C64-spezifischen Befehle benutzt werden, sind Programme des C64 ohne weiteres auf den größeren Commodore-Rechnern lauffähig. Sie können aber nach dem Laden zunächst weder gelistet noch gestartet werden. Der Grund dafür ist einfach der, daß der Speicherbereich für BASIC-Programme bei den größeren CBM-Rechnern bei 1024 beginnt, während er beim C64 bei 2048 liegt. Diese absolute Adresse wird auch beim Speichern des Programms auf einer Kassette festgehalten.

Nun kann man natürlich den Zeiger auf den Programmbeginn entsprechend verbiegen. Der Nachteil bei dieser Methode ist jedoch, daß man das jedesmal nach dem Laden des Programms erneut machen muß.

Eine bessere und endgültige Lösung ist, das ganze Programm zu verschieben. Dazu muß man wissen, daß in den ersten zwei Bytes jeder BASIC-Programmzeile der Vorwärtszeiger enthalten ist, der die absolute Adresse der nächsten Zeile gespeichert hat. Wenn jetzt beispielsweise eine Programmzeile eingefügt wird, werden die restlichen Adressen vom BASIC-Interpreter neu berechnet. Diese Eigenschaft des Interpreters können wir uns zunutze machen.

Wir verschieben zunächst im Direktmodus das ganze Programm um 1024 Speicherstellen beim C64 (beim VC20 ohne Speichererweiterung müssen es 3072 sein):

```
FOR I=1025 TO 256*(PEEK(43)+1):POKE I,PEEK(I+1024):NEXT
```

Damit sind wir aber noch nicht am Ziel, denn die absoluten Adreßhinweise in den Programmzeilen haben sich durch diese Verschiebung noch nicht geändert. Wenn wir nun aber eine neue Programmzeile ganz am Anfang einfügen, muß der BASIC-Interpreter dieses Update für uns erledigen:

```
0 REM DUMMY-ANWEISUNG
```

Zum Schluß muß nur noch das Programmende neu berechnet werden:

```
POKE 43,PEEK(43)-4
```

oder für den Programmtransfer vom VC20 ohne Speichererweiterung:

```
POKE 43,PEEK(43)-12
```

Nun können Sie das Programm ganz nach Belieben laufen lassen, speichern oder laden, ganz wie bei einem normalen Programm für diesen Rechner.

Wenn es allerdings Probleme wegen rechnerspezifischer Befehle wie z. B. POKE gibt, sollten Sie das nächste Kapitel besonders aufmerksam lesen.

## Programmübertragung und -anpassung von Rechnern der 2000-, 3000-, 4000- und 8000-Serie auf den C64 und VC20

Abgesehen von den Problemen, die sich durch die unterschiedliche Anzahl von Zeichen pro Spalte - 80, 40 oder nur 20 - beim Lauf eines Programms ergeben könnten, ist die physikalische Übertragung von Programmen an sich keines. Der VC20 und der C64 besitzen ein Verschiebeprogramm, das Programme von anderen CBM-Rechnern, die also in höheren Speicherbereichen liegen, automatisch an die richtigen Stellen bringt.

### Programmanpassung

Wie wir schon gesehen haben, ist das physikalische Laden eines solchen Programms in den C64 kein Problem. Wenn es aber trotzdem nicht läuft, kann das rechnerspezifische Ursachen haben. Die folgenden Unterschiede müssen auf jeden Fall berücksichtigt und ausgemerzt werden:

## Tastaturbelegung

Jeder Taste auf der Tastatur ist ein Code zugeordnet, dessen Wert, wenn die entsprechende Taste gedrückt wird, in einer Speicherstelle der Zeropage abgefragt werden kann. Beim C64 ist das die Speicherstelle 203, bei den größeren CBM-Rechnern ist es 151. Mit den beiden Programmzeilen

```
10 PRINT PEEK (203)
20 GOTO 10
```

kann der Wert einer gedrückten Taste beim C64 angezeigt werden und mit

```
10 PRINT PEEK (151)
20 GOTO 10
```

für die anderen CBM-Rechner.

Wenn in dem Programm, das Sie übertragen haben, also ein Tastendruck abgefragt wird, wie das oft bei Spielen der Fall ist, müssen Sie den Code der nachfolgenden Tabelle entsprechend ändern:

| TASTE | C64/C128 | CBM 3xxx | CBM 4xxx/8xxx |
|---|---|---|---|
| A | 10 | 48 | 65 |
| B | 28 | 30 | 66 |
| C | 20 | 31 | 67 |
| D | 18 | 47 | 68 |
| E | 14 | 63 | 69 |
| F | 21 | 39 | 70 |
| G | 26 | 46 | 71 |
| H | 29 | 38 | 72 |
| I | 33 | 53 | 73 |
| J | 34 | 45 | 74 |
| K | 37 | 37 | 75 |
| L | 42 | 44 | 76 |
| M | 36 | 29 | 77 |
| N | 39 | 22 | 78 |
| O | 38 | 60 | 79 |
| P | 41 | 52 | 80 |
| Q | 62 | 64 | 81 |
| R | 17 | 55 | 82 |
| S | 13 | 40 | 83 |
| T | 22 | 62 | 84 |
| U | 30 | 28 | 85 |
| V | 31 | 23 | 86 |

| TASTE | C64/C128 | CBM 3xxx | CBM 4xxx/8xxx |
|---|---|---|---|
| W | 9 | 56 | 87 |
| X | 23 | 24 | 88 |
| Y | 25 | 54 | 89 |
| Z | 12 | 32 | 90 |
| 1 | 56 | 26 | 49 |
| 2 | 59 | 18 | 50 |
| 3 | 8 | 25 | 51 |
| 4 | 11 | 42 | 52 |
| 5 | 16 | 34 | 53 |
| 6 | 19 | 41 | 54 |
| 7 | 24 | 58 | 55 |
| 8 | 27 | 50 | 56 |
| 9 | 32 | 57 | 57 |
| ← | 57 | 75 | 95 |
| + | 40 | 17 | 43 |
| - | 43 | 9 | 45 |
| £ | - | - | 48 |
| CLR/HOME | 51 | 7 | 19 |
| INST/DEL | 0 | - | 20 |
| @ | 46 | 15 | 64 |
| * | 49 | 33 | 58 |
| ↑ | 54 | 59 | 94 |
| = | 53 | 1 | 61 |
| RETURN | 1 | 27 | 13 |
| , | 47 | 70 | 44 |
| . | 44 | 2 | 46 |
| / | 55 | 49 | 47 |
| CRSR rechts | 2 | - | 29 |
| CRSR unten | 7 | - | 17 |
| Leertaste | 60 | 6 | 32 |
| TAB | - | - | 9 |
| ESC | - | - | 27 |
| OFF/RVS | - | - | 18 |
| REPEAT | - | - | 255 |

Bei den 3x-, 4x- und 8x-Rechnern kann man zudem über die Speicherstelle 152 erfahren, ob die SHIFT-Taste gedrückt wird:

`PRINT PEEK(152)` ergibt 1, wenn die SHIFT-Taste gedrückt ist, sonst 0

Beim C64 ist die Speicherstelle 653 in dem Fall interessant.

```
PRINT PEEK(653)
```

ergibt 1, wenn die SHIFT-Taste gedrückt ist
ergibt 2, wenn die Commodore-Taste gedrückt ist
ergibt 4, wenn die CTRL-Taste gedrückt ist
sonst 0

## Bildschirmspeicher

Da der Bildschirmspeicher bei den größeren CBM-Rechnern zwischen den Adressen 32768 und 33767 liegt, beim C64 hingegen zwischen 1024 und 2023, muß die Differenz von 31744 zu den jeweiligen Adressen addiert bzw. davon subtrahiert werden. Hinzu kommt noch für den C64 immer eine entsprechende Farbgebung für das Zeichen, die man hinter jeden POKE-Befehl mit einem zweiten POKE-Befehl

```
POKE 1024 + 54272,F
```

dahinterhängen muß. Die Farbe F entspricht dabei den üblichen C64-Farbnummern von 0 bis 15.

Bei der umgekehrten Programmanpassung vom C64 auf größere CBM-Rechner müssen diese Farbgebung-POKEs entfernt werden.

## Groß-/Kleinschreibung

Ein weiterer POKE-Befehl der CBM-Systeme kann sehr leicht für den C64 modifiziert werden. Wenn man den Befehl

```
POKE 59468,12
```

eingibt, schaltet der Rechner in Großschrift/Grafik um. Zurück in den Groß-/Kleinschriftmodus (Einschaltzustand) kommt man mit:

```
POKE 59468,14
```

Beim C64 wird mit

```
PRINT CHR$(14)
```

in Groß-/Kleinschreibung umgewandelt und mit

```
PRINT CHR$(142)
```

zurück in Großschrift/Grafik (Einschaltzustand).

## Anpassungen zwischen Rechnern der 2000-, 3000-, 4000- und 8000-Serie

Zwischen den Rechnern dieser vier Serien gibt es prinzipiell keine großen Unterschiede. Störend kann sich lediglich auswirken, daß die 8000-Rechner 80 Zeichen pro Zeile darstellen können, was auf den anderen CBM-Rechnern, die alle nur 40 Zeichen pro Zeile haben, zu unschönen Ausgaben führen kann.

Wie wir im vorigen Abschnitt bei der Übertragung zwischen C64 und VC20 und den größeren CBM-Rechnern schon beschrieben haben, muß auch hier teilweise der Anfang der BASIC-Programme verschoben werden. Die Adressen, die auf den Anfang der BASIC-Programme zeigen, sind bei den einzelnen Rechnern die folgenden:

| | |
|---|---|
| CBM 2000 | Adressen 122 und 123 |
| CBM 3000 | Adressen 40 und 41 |
| CBM 4000 | Adressen 40 und 41 |
| CBM 8000 | Adressen 40 und 41 |
| VC20 | Adressen 43 und 44 |
| C64 | Adressen 43 und 44 |
| C128 | Adressen 45 und 46 |

Wie man die Programme so verschiebt, daß sie an den jeweils erforderlichen Adressen stehen, ist im vorigen Abschnitt am Beispiel des C64 beschrieben.

Die großen Ähnlichkeiten der 2000- und 3000-Serie werden nur durch unterschiedliche Handhabung von Eingabepuffern und Pointern geschmälert. Bei Rechnern der 4000- und 8000-Serie kommen die neuen komfortablen Diskettenbefehle dazu, die auf den anderen CBM-Rechnern nicht zur Verfügung stehen. Außerdem ist die Tastaturbelegung dieser beiden anders als bei den 3000-Rechnern, wie man der Tabelle im vorigen Abschnitt entnehmen kann.

## Übertragung von Programmen der 8000-Rechner auf C128

Die zahlreichen komfortablen Diskettenbefehle des BASIC 4.0 sind auch im BASIC 7.0 des C128 wieder vertreten. Dennoch sind die Programme der 8000-Rechner nicht ohne weiteres auf dem C128 lauffähig. Wie immer bei Übertragungen zwischen verschiedenen Rechnern müssen zunächst die Adressen nach den Systembefehlen PEEK, POKE, USR und SYS umgesetzt werden.

Aber auch bei Befehlen, die es auf beiden Rechnern gibt, läuft nicht alles reibungslos. Die Token für Befehle wie CONCAT, COPY, COLLECT SCRATCH, DIRECTORY u. a. sind nämlich unterschiedlich. Auf dieses Problem geht das Handbuch des C128 in Anhang M ein. Dort ist ein Programm

angegeben, das die Token eines 8000-Programms in die entsprechenden Token des C128 umsetzen soll. Damit das Programm läuft, sollten Sie die folgenden Zeilen korrigieren:

Zeile 191: GOTO 195 muß durch GOTO 200 ersetzt werden.
Zeile 192: C$(I)#... muß durch C$(I)=... ersetzt werden.
Zeile 9020: DSAVE;DLOAD muß DSAVE,DLOAD heißen.

### Programme auf den Rechnern der 700-Serie

Die Rechner dieser Serie laufen mit einer erweiterten Version des BASIC 4.0, das im großen und ganzen dem BASIC 3.5 entspricht. Das Hauptproblem ergibt sich hier durch das unterschiedliche Diskettenformat. Als Lösung bietet sich eine Programmübertragung über Kassettenrecorder an.

### Programmübertragungen auf dem C128 zwischen dem 64-Modus und dem 128-Modus

Wenn Sie nun Besitzer eines C128 sind, können Sie beliebig Programme, die im 64-Modus geschrieben und abgespeichert wurden, im 128-Modus laden und umgekehrt. Das bringt viele Vorteile mit sich, auch wenn die Programme höchstwahrscheinlich in dem jeweils anderen Modus zunächst nicht laufen. Doch darauf kommen wir später zurück.

Worin besteht nun der Vorteil, ein Programm, das z. B. im 128-Modus geschrieben wurde, im 64-Modus zu laden? Wenn Sie mit BASIC schon Erfahrungen gesammelt haben, fällt Ihnen vielleicht auf, daß beim C128 neben einigen Toolkit-Funktionen auch die komfortablen Befehle MERGE, RENUMBER oder FIND nicht oder nur unvollständig implementiert wurden. Das ist trotz der anderen vielfältigen Möglichkeiten im BASIC 7.0 ein echter Mangel.

Da es aber für den C64 auf dem Markt bereits eine große Anzahl von BASIC-Erweiterungen gibt, können Sie ohne weiteres darauf zurückgreifen. Aus diesem Grund haben wir auch in den Anhängen den vollständigen alphabetischen Befehlsvorrat von SIMON´S BASIC, EXBASIC LEVEL II, Super Expander und HONEY.AID aufgeführt.

Sie können nun Ihr Programm im 64-Modus laden und mit Hilfe von MERGE oder FIND oder anderen Toolkit-Befehlen, die den Zeileninhalt nicht verändern, bequem editieren. Die Einschränkung, daß der Zeileninhalt nicht verändert werden darf, hat folgenden Hintergrund. Wie Sie ja wissen, werden alle BASIC-Schlüsselwörter intern in Token umgesetzt. Diese Werte stimmen beim C128 im 64- und im 128-Modus nicht überein.

## Anhang A

# BASIC-Befehlserweiterung Simon's BASIC für den C64 und C128

Simon's BASIC ist eine Befehlserweiterung für den C64, die Ihnen mehr als 110 zusätzliche Befehle zur Verfügung stellt. Wenn Sie das Programm geladen haben, erscheint auf Ihrem Bildschirm die Meldung:

```
** EXPANDED CBM V2 BASIC ***
30719 BASIC BYTES FREE
```

Problemlos läßt es sich selbstverständlich im 64er-Modus in den C128 laden, und auch das im Handel erhältlich Steckmodul kann verwendet werden.

Im folgenden geben wir eine alphabetische Übersicht der Befehle mit Erläuterungen:

### ANGL (Grafikbefehl)

**Funktion:** Radius einzeichnen

**Syntax:** `ANGL X1,Y1,Winkel,X2,Y2,Farbe`

Mit dem Befehl ANGL kann man einen Radius in eine Ellipse einzeichnen, auch ohne daß die Ellipse auf dem Bildschirm vorhanden ist. Dabei ist X1,Y1 der Mittelpunkt und somit Startpunkt, und X2,Y2 sind die Koordinaten der Halbachsen.

### ARC (Grafikbefehl)

**Funktion:** Bogen zeichnen

**Syntax:**
```
ARC X1,Y2,Startwinkel,Endwinkel,
Abstand,X2,Y2,Farbe
```

Mit dem Befehl ARC können Sie einen Bogen zeichnen. Dabei ist X1,Y1 der Mittelpunkt, X2 der X-Radius und Y2 der Y-Radius der Ellipse.

## AT (Zeichenketten-Operation)

**Funktion:** Ausgabe einer Zeichenkette positionieren

**Syntax:**
```
PRINT AT(Spalte, Zeile)"Zeichenkette"

PRINT "Zeichenkette1" AT(Spalte,Zeile)
"Zeichenkette2"
```

Mit PRINT AT kann eine Zeichenkette an der Bildschirmposition, die durch Spalte (0 bis 39) und Zeile (0-24) festgelegt wird, ausgegeben werden.

## AUTO (Programmierhilfe)

**Funktion:** Automatische Zeilennumerierung

**Syntax:**
```
AUTO Startnummer,Schrittweite
```

Der Befehl AUTO kann mit einer Schrittweite angegeben werden und erzeugt dann bei der Programmeingabe automatisch die Nummer der nächsten BASIC-Zeile, und zwar mit der angegebenen Schrittweite.

## BCKGNDS (Bildschirmbefehl)

**Funktion:** Hintergrundfarbe eines Zeichens angeben

**Syntax:**
```
BCKGNDS Farbe1,Farbe2,Farbe3,Farbe4
```

Mit dem BCKGNDS-Befehl werden die Hintergrundfarben festgelegt, und zwar:

| | |
|---|---|
| Farbe1 | normale Hintergrundfarbe |
| Farbe2 | Hintergrundfarbe für geshiftete Zeichen |
| Farbe3 | Hintergrundfarbe für reverse Zeichen |
| Farbe4 | Hintergrundfarbe für reverse geshiftete Zeichen |

Mit dem Befehl NRM wird wieder der Einschaltzustand erreicht.

## **BFLASH** (Bildschirmbefehl)

**Funktion:** Farbe des Bildschirmrahmens ändern

**Syntax:** `BFLASH Geschwindigkeit,Farbe1,Farbe2`

Der Befehl BFLASH schaltet die Farbe des Bildschirmrahmens zwischen Farbe1 und Farbe2 hin und her, und zwar mit der angegebenen Geschwindigkeit (0 - schnell bis 255 - langsam).

Der Befehl BFLASH 0 beendet das Umschalten.

## **BLOCK** (Grafikbefehl)

**Funktion:** Ausgefülltes Rechteck zeichnen

**Syntax:** `BLOCK X1,Y1,X2,Y2,Farbe`

Ein Rechteck wird gezeichnet und mit der angegebenen Farbe ausgefüllt. Dabei markiert X1,Y1 die linke obere Ecke und X2,Y2 die rechte untere. X darf maximal 65535 und Y maximal 255 betragen. Vergessen Sie nicht, vorher mit HIRES die hochauflösende Grafik einzuschalten.

## **CALL** (Anweisung)

**Funktion:** Sprung zu einer mit PROC definierten Routine

**Syntax:** `CALL Marke`

So wie Sie mit GOTO eine festgelegte Zeilennummer anspringen können, erreichen Sie über CALL Marke einen Programmteil, der mit

```
PROC Marke
```

beginnt und mit

```
END PROC
```

aufhört.

## CENTRE (Zeichenketten-Operation)

**Funktion:** Zentrieren einer Zeichenkette

**Syntax:**
```
CENTRE "Zeichenkette"
CENTRE Zeichenkettenvariable
```

Die angegebene Zeichenkette wird in der Mitte der Bildschirmzeile ausgegeben.

## CGOTO (Anweisung)

**Funktion:** Sprung zu variabler Zeilennummer

**Syntax:**
```
CGOTO Ausdruck
```

Die Zeilennummer, zu der verzweigt werden soll, wird erst bei der Ausführung des Befehls berechnet.

## CHAR (Textbefehl)

**Funktion:** Zeichen auf Grafikbildschirm schreiben

**Syntax:**
```
X1,Y1,Zeichencode,Farbe,Größe
```

X1,Y1 bezeichnet die linke obere Ecke, an der das Zeichen beginnen soll, dessen ASCII-Code, Farbe und Ausdehnung nach unten (maximal 255) ebenfalls angegeben sein muß.

## CHECK (Spritebefehl)

**Funktion:** Abfrage auf Sprite-Kollision

**Syntax:**
```
IF CHECK(Spritenummer1,Spritenummer2)=0
THEN...

IF CHECK(0)=0 THEN...
```

Wenn die Sprites mit den angegebenen Nummern zusammengestoßen sind, wird die Anweisung hinter THEN ausgeführt. CHECK(0) prüft die Kollision eines Sprites mit einem Bildschirmzeichen.

## CIRCLE (Grafikbefehl)

**Funktion:** Ellipse zeichnen

**Syntax:** `CIRCLE X1,Y1,X2,Y2,Farbe`

Mit CIRCLE wird eine Ellipse (Sonderform Kreis) gezeichnet mit dem Mittelpunkt X1,Y1 und dem Radius X2 in X-Richtung und Y2 in Y-Richtung.

## CMOB (Spritebefehl)

**Funktion:** Farbgebung für Mehrfarben-Sprites

**Syntax:** `CMOB Farbe1,Farbe2`

Mit CMOB werden die beiden zusätzlichen Farben für alle Mehrfarben-Sprites festgelegt.

## COLD (Programmierhilfe)

**Funktion:** System-Neustart

**Syntax:** `COLD`

Nach dem COLD-Befehl befindet sich der Rechner wieder im Einschaltzustand von Simon´s BASIC.

## COLOUR (Grafikbefehl)

**Funktion:** Hintergrund- und Rahmenfarbe setzen

**Syntax:** `COLOUR Rahmenfarbe,Hintergrundfarbe`

Dieser Befehl ändert unter Angabe der bekannten Farbparameter die Hintergrund- und Rahmenfarbe.

```
COLOUR 0,1
```

zum Beispiel ändert den Bildschirmhintergrund in Schwarz (0) und den Rand in Weiß (1).

## COPY (Bildschirmbefehl)

**Funktion:** HIRES- oder MULTI-Bildschirm ausdrucken

**Syntax:** `COPY`

Der COPY-Befehl gibt eine Hardcopy des HIRES- oder MULTI-Bildschirms auf dem Drucker aus.

## CSET (Grafikbefehl)

**Funktion:** Umschalten des Zeichensatzes

**Syntax:**
```
CSET 0
CSET 1
CSET 2
```

Mit CSET kann vom Programm aus der Zeichensatz umgestellt werden. 0 bringt uns in den Großschrift-/Grafikmodus und 1 in den Groß-/Kleinschriftmodus. Mit CSET 2 kann die letzte Grafik noch einmal dargestellt werden.

## DELAY (Programmierhilfe)

**Funktion:** Geschwindigkeit zum Auflisten eines Programms festlegen

**Syntax:** `DELAY Geschwindigkeit`

Mit DELAY wird die Geschwindigkeit geregelt, mit der ein Programm gelistet wird. Dabei ist 1 die höchste und 255 die niedrigste Geschwindigkeitsstufe. Beim Listen muß die SHIFT-Taste gedrückt werden, damit das DELAY-Tempo in Kraft tritt.

## DESIGN (Spritebefehl)

**Funktion:** Zeichen in Zeichensatz verändern

**Syntax:**
```
DESIGN 0,Startadresse
DESIGN 1,Startadresse + 49152
```

Der DESIGN-Befehl reserviert Speicherplatz für ein Sprite im hochauflösenden Modus (DESIGN 0) oder im Mehrfarbenmodus (DESIGN 1). Die Startadresse errechnet sich dabei aus der Blocknummer mal 64, da jedes Sprite 64 Bytes Speicherplatz benötigt.

## DETECT (Spritebefehl)

**Funktion:** Vorbereitung für CHECK-Befehl

**Syntax:**
```
DETECT 0
DETECT 1
```

Bevor der CHECK-Befehl zur Abfrage auf Sprite-Kollision ausgeführt werden kann, muß zweimal der Befehl DETECT gegeben werden, damit das Register gelöscht wird und die nächste Kollision registrieren kann. Dabei wird der Parameter 0 angegeben, wenn eine Sprite/Sprite-Kollision abgefragt werden soll, und 1 für die Kollision zwischen einem Sprite und einem Bildschirmzeichen.

## DIR (Diskettenbefehl)

**Funktion:** Inhaltsverzeichnis einer Diskette zeigen

**Syntax:**
```
DIR"$
DIR"$Programmname
```

Mit DIR"$ kann man sich das ganze Inhaltsverzeichnis einer Diskette ansehen, und mit DIR"$Programmname kann man einzelne Programme heraussuchen. Dabei können bei der Angabe des Programmnamens die Zeichen ? und * als Joker verwendet werden.

## DISABLE (Eingabebefehl)

**Funktion:** Aufheben des ON KEY-Befehls

**Syntax:**
```
DISABLE
```

Mit DISABLE muß der ON KEY-Befehl aufgehoben werden, sonst verbleibt das restliche Programm in einer Endlosschleife.

## DISAPA (Programmierhilfe)

**Funktion:** Anzeige der zu schützenden Programmzeile

**Syntax:** `Zeilennummer DISAPA Programmzeile`

Wenn zu Beginn einer Programmzeile DISAPA steht, wird diese nach Eingabe des Befehls SECURE 0 bis auf die Zeilennummer beim Listen unsichtbar bleiben. Ohne SECURE können Sie feststellen, daß DISAPA 4 Doppelpunkte vor die zu schützende Zeile setzt:

```
100 DISAPA:::: REM GEHEIM
```

## DISK (Diskettenbefehl)

**Funktion:** Diskettendatei öffnen, Anweisung ausführen, schließen

**Syntax:** `DISK "Anweisung`

Der DISK-Befehl ersetzt das Öffnen und Schließen des Diskettenkanals und führt zwischendurch noch die angegebene Anweisung aus.

## DISPLAY (Programmierhilfe)

**Funktion:** Belegung der Funktionstasten anzeigen

**Syntax:** `DISPLAY`

Mit der DISPLAY-Anweisung können Sie sich die Funktionstastenbelegung ansehen.

## DIV (arithmetische Funktion)

**Funktion:** Ergebnis einer Ganzzahl-Division ohne Rest

**Syntax:** `DIV(Zahl1,Zahl2)`

Der Befehl DIV wandelt die beiden Zahlen zunächst durch Abschneiden der Nachkommastellen in ganze Zahlen um und liefert dann das Ergebnis der Division Zahl1/Zahl2 ohne den Rest.

## DOWNB, DOWNW (Bildschirmbefehl)

**Funktion:** Bildschirm nach unten rollen

**Syntax:**
```
DOWNB erste Zeile,erste Spalte,
letzte Zeile,letzte Spalte

DOWNW erste Zeile,erste Spalte,
letzte Zeile,letzte Spalte
```

Der durch die Zeilen und Spalten bezeichnete Bildschirmausschnitt wird um eine Zeile nach unten gerollt. Dabei wird der Inhalt aus dem Bereich hinausgeschoben und ist nicht mehr sichtbar (DOWNB), oder er wird hinausgeschoben und kommt an der anderen Seite wieder zum Vorschein (DOWNW).

## DRAW (Grafikbefehl)

**Funktion:** Figur zeichnen

**Syntax:**
```
DRAW "Ziffernfolge",X,Y,Zeichenfarbe

DRAW Zeichenkettenvariable,X,Y,Zeichenfarbe
```

DRAW zeichnet eine beliebige Figur in der angegebenen Zeichenfarbe nach folgendem Schema: Startpunkt ist X,Y, und die Zeichenrichtung wird mit Hilfe von Ziffern, die auch zuvor in einer Zeichenkettenvariablen abgelegt werden können, wie folgt bestimmt:

| Ziffer | Bewegung um einen Punkt nach: | sichtbar/unsichtbar |
|---|---|---|
| 0 | rechts | unsichtbar |
| 1 | oben | unsichtbar |
| 2 | unten | unsichtbar |
| 3 | links | unsichtbar |
| 4 | unten | unsichtbar |
| 5 | rechts | sichtbar |
| 6 | oben | sichtbar |
| 7 | unten | sichtbar |
| 8 | links | sichtbar |
| 9 | Ende | |

## DUMP (Programmierhilfe)

**Funktion:** Variablenwerte anzeigen

**Syntax:** `DUMP`

Mit DUMP kann man sich die Variablenwerte in der Reihenfolge ihres Auftretens im Programm ansehen.

## DUP (Zeichenketten-Operation)

**Funktion:** Zeichenkette vervielfältigen

**Syntax:**
```
DUP ("Zeichenkette",Faktor)
DUP (Zeichenkettenvariable,Faktor)
```

Der Befehl DUP vervielfältigt die angegebene Zeichenkette oder Variable so oft, wie der Faktor (maximal 65535) angibt. Wird dabei die maximale Zeichenkettenlänge von 255 überschritten, ergeben sich unvorhersehbare Vervielfältigungen.

## ENVELOPE (Soundbefehl)

**Funktion:** Hüllkurve festlegen

**Syntax:** `ENVELOPE v,a,d,s,r`

Die Parameter bedeuten im einzelnen:

| | |
|---|---|
| **v** | voice (Stimme) mit Werten von 1 bis 3 |
| **a** | attack (Einschwingen) mit Werten von 0 bis 15 |
| **d** | decay (Absinken) mit Werten von 0 bis 15 |
| **s** | sustain (Nachklingen) mit Werten von 0 bis 15 |
| **r** | release (Ausklingen) mit Werten von 0 bis 15 |

## ERRLN (Fehlervariable)

**Funktion:** Speichern der Fehlermeldung

**Syntax:** z. B. `PRINT ERRLN`

In ERRLN wird bei Auftreten eines Fehlers die zugehörige Fehlermeldung gespeichert. Folgende Fehler sind möglich:

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

## ERRN (Fehlervariable)

**Funktion:** Speichern der Fehlernummer

**Syntax:** z. B. `PRINT ERRN`

In ERRN wird bei Auftreten eines Fehlers die zugehörige Fehlernummer gespeichert. Die möglichen Fehler finden Sie unter dem Stichwort ERRLN.

## EXEC

**Funktion:** Unterprogrammaufruf

**Syntax:** `EXEC Marke`

So wie Sie mit GOSUB ein Unterprogramm an einer festgelegten Zeilennummer anspringen können, erreichen Sie mit EXEC Marke ein Unterprogramm, das mit

```
PROC Marke
```

beginnt und mit

```
END PROC
```

endet. Danach wird der auf EXEC folgende Befehl ausgeführt.

## EXOR (logische Operation)

**Funktion:** Exklusiv-Oder-Operation

**Syntax:** `EXOR (Zahl1,Zahl2)`

Die Zahlen Zahl1 und Zahl2 werden in Binärzahlen umgewandelt und dann über Exklusiv-Oder miteinander verknüpft. Dabei können die Zahlen aus verschiedenen Zahlensystemen stammen und Werte von 0 bis 65535 annehmen.

## FCHR (Bildschirmbefehl)

**Funktion:** Bildschirmbereich mit Zeichen füllen

**Syntax:** `FCHR X,Y,Spaltenanzahl,Zeilenanzahl,`
`Zeichencode`

Der Bildschirmbereich, dessen linke obere Ecke an den Koordinaten X,Y liegt und dessen Größe durch die angegebene Spaltenanzahl (0 bis 39) und Zeilenanzahl (0 bis 24) definiert ist, wird mit dem Zeichen, dessen Code angegeben ist, ausgefüllt.

## FCOL (Bildschirmbefehl)

**Funktion:** Zeichenfarbe festlegen

**Syntax:** `FCOL X,Y,Spaltenanzahl,Zeilenanzahl,Farbe`

FCOL legt die Zeichenfarbe in dem Bildschirmbereich fest, dessen linke obere Ecke an den Koordinaten X,Y liegt und dessen Größe durch die angegebene Spaltenanzahl (0 bis 39) und Zeilenanzahl (0 bis 24) definiert ist.

## **FETCH** (Eingabebefehl)

**Funktion:** Eingabezeichen kontrollieren

**Syntax:** `FETCH "Taste",Tastenanzahl,Variable`

Mit FETCH kann die Art und Anzahl von gedrückten Tasten kontrolliert werden. Dabei werden nur die Zeichen zugelassen, die in Anführungszeichen stehen. Ihre Anzahl wird als zweiter Parameter angegeben, und abgelegt wird die Zeichenkette in der bezeichneten Variablen.

Zusätzlich zu normalen Zeichenketten sind die folgenden drei Möglichkeiten vorgesehen:

| Taste | Funktion |
|---|---|
| HOME | alle Großbuchstaben sind zugelassen (CHR$(65) bis CHR$(90)) |
| CRSR unten | alle Zahlen und Satzzeichen sind zugelassen (CHR$(32) bis CHR$(64)) |
| CRSR rechts | alle Großbuchstaben auch mit SHIFT sind zugelassen |

## **FILL** (Bildschirmbefehl)

**Funktion:** Bildschirmbereich mit Farbe füllen

**Syntax:**
```
FILL X,Y,Spaltenanzahl,Zeilenanzahl,
Zeichencode,Farbe
```

Der Bildschirmbereich, dessen linke obere Ecke an den angegebenen Koordinaten X,Y liegt und dessen Größe durch Spaltenanzahl (0 bis 24) und Zeilenanzahl (0 bis 39) definiert ist, wird mit dem Zeichen, dessen Code angegeben ist, in der entsprechenden Farbe ausgefüllt.

## **FIND** (Programmierhilfe)

**Funktion:** Suchen von Zeichenketten

**Syntax:**
```
FINDSchlüsselwort
FINDZeichenkette
```

Der FIND-Befehl zeigt alle Zeilennummern an, in denen das gesuchte BASIC-Wort oder die Zeichenkette vorkommt. Dabei wird jedes Leerzeichen nach FIND schon als zu suchendes Zeichen angesehen.

## FLASH (Bildschirmbefehl)

**Funktion:** Farbe blinken lassen

**Syntax:** `FLASH Farbe,Geschwindigkeit`

Das Blinken der angegebenen Farbe mit einer Geschwindigkeit von 0 bis 255 wird eingeschaltet. Mit OFF wird das Blinken wieder abgestellt.

## FRAC (arithmetische Funktion)

**Funktion:** Nachkommaziffern einer Gleitkommazahl

**Syntax:** `FRAC(Zahl)`

Die Funktion FRAC gibt nur die Nachkommastellen einer Zahl auf 9 Stellen aus.

## GLOBAL (Wertzuweisung)

**Funktion:** Aufheben des LOCAL-Befehls

**Syntax:** `GLOBAL`

Mit GLOBAL werden alle lokalen Wertzuweisungen für Variablen aufgehoben. Die ursprünglichen Werte gelten wieder.

## GRAPHICS (Grafikbefehl)

**Funktion:** Liefert die Zahl 53248

**Syntax:** `Variable=GRAPHICS`

In der Variablen wird die Konstante 53248, die Adresse des VIC, abgelegt.

## HI COL (Grafikbefehl)

**Funktion:** Farben wie bei MULTI einstellen

**Syntax:** `HI COL`

Nachdem in einem Programm die MULTI-Farben mit LOW COL geändert wurden, können sie mit HI COL wieder zurückgeholt werden.

## HIRES (Grafikbefehl)

**Funktion:** Grafikmodus einschalten

**Syntax:** `HIRES Zeichenfarbe,Hintergrundfarbe`

Der Befehl ist in jedem Programm notwendig, das Grafikbefehle verwendet, da hierbei eine hochauflösende Bildschirmdarstellung (320 x 200 Bildschirmpunkte) gebraucht wird.

## HRDCPY (Bildschirmbefehl)

**Funktion:** Bildschirm ausdrucken

**Syntax:** `HRDCPY`

Mit diesem Befehl wird ein normal auflösender Bildschirm auf dem Drucker ausgegeben.

## IF...THEN...ELSE (Anweisung)

**Funktion:** Bedingung prüfen und Anweisungen ausführen

**Syntax:**
```
IF Bedingung THEN Anweisung1:
ELSE: Anweisung2
```

Falls die Bedingung hinter IF wahr ist, wird Anweisung1 ausgeführt, andernfalls Anweisung2.

## INKEY (Eingabebefehl)

**Funktion:** Funktionstastendruck ermitteln

**Syntax:** `INKEY`

INKEY liefert den Wert der gedrückten Funktionstaste (1 bis 16). Dabei erreicht man die Tasten wie folgt:

| | |
|---|---|
| F1, F3, F5, F7 | normal drücken |
| F2, F4, F6, F8 | mit SHIFT drücken |
| F9, F11, F13, F15 | mit C= drücken |
| F10, F12, F14, F16 | mit C= und SHIFT drücken |

Wenn keine Funktionstaste gedrückt wurde, wird der Wert 0 zurückgeliefert.

*Hinweis:* Es ist sehr schwierig, genau in dem Augenblick eine Funktionstaste zu drücken, wenn der Befehl INKEY die Tasten abfragt. Deshalb ist es sinnvoll, den Befehl in eine Schleife einzubauen. Auch dann muß man die entsprechende Taste die ganze Zeit gedrückt halten, damit man den richtigen Zeitpunkt nicht verpaßt:

```
10 FOR I=1 TO 1000:A=INKEY:NEXT
20 PRINT A
```

## INSERT (Zeichenketten-Operation)

**Funktion:** Einsetzen einer Zeichenkette in eine andere

**Syntax:** `INSERT ("Zeichenkette1","Zeichenkette2", Position)`

Die Zeichenkette1 wird ab der angegebenen Position in die Zeichenkette2 eingefügt. Der Positions-Parameter muß dabei um mindestens 1 kleiner sein als die Länge der Zeichenkette2.

## INST (Zeichenketten-Operation)

**Funktion:** Überschreiben eines Teils einer Zeichenkette

**Syntax:** `INST ("Zeichenkette1","Zeichenkette2", Position)`

Die Zeichenkette2 wird nach der angegebenen Position mit der Zeichenkette1 überschrieben. Die Positionsangabe kann dabei maximal 255 sein, wenn sie aber die Länge der Zeichenkette2 überschreitet, kommt Schrott dabei heraus.

## INV (Bildschirmbefehl)

**Funktion:** Bildschirmbereich invertieren

**Syntax:** `INV X,Y,Spaltenanzahl,Zeilenanzahl`

Der Bildschirmbereich, dessen linke obere Ecke an den Koordinaten X,Y liegt und dessen Größe durch Spaltenanzahl (0 bis 39) und Zeilenanzahl (0 bis 24) bestimmt ist, wird invers dargestellt.

## JOY (Joystickbefehl)

**Funktion:** Stellung des Joysticks ermitteln

**Syntax:** `Variable=JOY`

Der Wert in der Variablen gibt Auskunft über die Stellung des Joysticks:

| | |
|---|---|
| 0 | Mittelstellung |
| 1 | Nord |
| 2 | Nordost |
| 3 | Ost |
| 4 | Südost |
| 5 | Süd |
| 6 | Südwest |
| 7 | West |
| 8 | Nordwest |
| 128 | Feuerknopf gedrückt |

## KEY (Programmierhilfe)

**Funktion:** Belegen der Funktionstasten

**Syntax:** `KEY Tastennummer,"Befehl"`

Mit Hilfe des KEY-Befehls können die Funktionstasten mit Befehlen von maximal 15 Zeichen belegt werden. Wenn der Befehl nach Drücken der Taste

auch sofort ausgeführt werden soll, muß RETURN in Form von + CHR$(13) angehängt werden:

```
KEY 1,"LIST"+CHR$(13)
```

Es stehen Ihnen 16 Funktionstasten zur Verfügung:

| | |
|---|---|
| F1,F3,F5,F7 | normal drücken |
| F2,F4,F6,F8 | gleichzeitig mit SHIFT drücken |
| F9,F11,F13,F15 | gleichzeitig mit C= drücken |
| F10,F12,F14,F16 | gleichzeitig mit SHIFT und C= drücken |

## LEFTB, LEFTW (Bildschirmbefehl)

**Funktion:** Bildschirmbereich rollen

**Syntax:**
```
LEFTB erste Zeile,erste Spalte,letzte Zeile,
letzte Spalte

LEFTW erste Zeile,erste Spalte,letzte Zeile,
letzte Spalte
```

Der durch die Zeilen und Spalten festgelegte Bildschirmausschnitt wird um eine Spalte nach links gerollt. Dabei wird der Inhalt aus dem Bereich hinausgeschoben und ist nicht mehr sichtbar (LEFTB), oder er wird aus dem Bereich hinausgeschoben und kommt auf der anderen Seite wieder zum Vorschein (LEFTW).

## LIN (Programmierhilfe)

**Funktion:** Zeilennummer der Cursorposition feststellen

**Syntax:** `LIN`

PRINT LIN gibt die Nummer der Bildschirmzeile aus, in der sich der Cursor zu der Zeit befindet.

## LINE (Grafikbefehl)

**Funktion:** Linien zeichnen

**Syntax:** `LINE X1,Y1,X2,Y2,Zeichenfarbe`

Der Befehl LINE zeichnet in der angegebenen Farbe eine Linie vom Anfangspunkt X1,Y1 bis zum Endpunkt X2,Y2.

## LOCAL (Wertzuweisung)

**Funktion:** Lokale Wertzuweisung einer Variablen

**Syntax:** `LOCAL Variablenliste`

Nachdem eine Variable in der Liste hinter LOCAL aufgeführt wurde, kann ihr ein Wert zugewiesen werden, der nur in einem Programmteil Gültigkeit hat. Dieser Zustand hält an, bis der Befehl GLOBAL gegeben wird.

## LOOP...EXIT IF...END LOOP (Anweisung)

**Funktion:** Schleife

**Syntax:**
```
LOOP: Anweisungen: EXIT IF Bedingung:
END LOOP
```

Die Anweisungen hinter LOOP werden so lange ausgeführt, bis die Bedingung nach EXIT IF erfüllt ist. Dann wird der nächste Befehl hinter END LOOP abgearbeitet.

## LOW COL (Grafikbefehl)

**Funktion:** Farben wechseln

**Syntax:** `LOW COL Farbe1,Farbe2,Farbe3`

Mit LOW COL können 3 weitere Farben für das Zeichnen bereitgestellt werden.

## MEM

**Funktion:** Zeichensatz von ROM in RAM bringen

**Syntax:** `MEM`

Um den Zeichensatz neu zu definieren (z. B. mit DESIGN) muß er zunächst vom ROM- in den RAM-Bereich (Arbeitsspeicher) verlegt werden.

## MERGE (Programmierhilfe)

**Funktion:** Dazuladen eines Programms zu einem im Arbeitsspeicher befindlichen

**Syntax:** `MERGE "Programmname",Gerätenummer`

Wenn Sie mit MERGE ein Programm zu einem im Arbeitsspeicher befindlichen dazuladen, müssen Sie auf eine geeignete Numerierung der Programme achten, da sonst Zeilennummern überschrieben werden könnten.

## MMOB (Spritebefehl)

**Funktion:** Sprite auf dem Bildschirm zeigen

**Syntax:**
```
MMOB Spritenummer,X1,Y1,X2,Y2,Größe,
Geschwindigkeit
```

Das Sprite mit der angegebenen Nummer erscheint auf dem Bildschirm und kann bewegt werden. Der Start wird dabei durch die Koordinaten X1,Y1 festgelegt, das Ziel durch X2,Y2. Die Größe des Sprites kann auch verdoppelt werden:

0 normale Größe
1 doppelte Größe in X-Richtung
2 doppelte Größe in Y-Richtung
3 doppelte Größe in X- und Y-Richtung

Die Geschwindigkeit variiert von 1 (langsam) bis 255 (schnell).

## MOB OFF (Spritebefehl)

**Funktion:** Sprite vom Bildschirm löschen

**Syntax:** `MOB OFF Spritenummer`

Das Sprite mit der angegebenen Nummer verschwindet vom Bildschirm.

## MOB SET (Spritebefehl)

**Funktion:** Sprite-Eigenschaften festlegen

**Syntax:**
```
MOB SET Spritenummer,Block,Farbe,
Priorität,Modus
```

Die Eigenschaften des Sprites mit der angegebenen Nummer werden unter Angabe seiner Blocknummer und Farbe (0 bis 15) festgelegt. Wenn die Priorität 0 ist, bewegt sich das Sprite vor den Bildschirmzeichen, bei 1 dahinter. Für Modus kann man 0 (hochauflösende Grafik) oder 1 (Mehrfarbenmodus) wählen.

## MOD (arithmetische Funktion)

**Funktion:** Rest nach einer Ganzzahl-Division

**Syntax:**
```
MOD (Zahl1,Zahl2)
```

Der Befehl MOD wandelt die beiden angegebenen Zahlen zunächst durch Abschneiden der Nachkommastellen in ganze Zahlen um und ermittelt dann den Rest der Division Zahl1/Zahl2.

## MOVE (Bildschirmbefehl)

**Funktion:** Bildschirmbereich duplizieren

**Syntax:**
```
MOVE X1,Y1,Spaltenanzahl,Zeilenanzahl,X2,Y2
```

Der Bildschirmbereich, dessen linke obere Ecke an den Koordinaten X1,Y1 liegt und dessen Größe durch Spaltenanzahl (0 bis 39) und Zeilenanzahl (0 bis 24) bestimmt ist, wird so dupliziert, daß seine linke obere Ecke an die Koordinaten X2,Y2 zu liegen kommt.

## MULTI (Grafikbefehl)

**Funktion:** Mehrfarbenmodus einschalten

**Syntax:**
```
HIRES Zeichenfarbe,Hintergrundfarbe:
MULTI Farbe1,Farbe2,Farbe3
```

Wird nach dem HIRES-Befehl der MULTI-Befehl eingeschaltet, so bewirkt er, daß die Strichstärke bei den Grafiken verdoppelt wird. Zudem lassen sich weitere Farben angeben. Auf dem Bildschirm befinden sich dann nicht mehr wie bei HIRES 320 x 200 Punkte, sondern nur noch 160 x 200 Punkte.

## MUSIC (Soundbefehl)

**Funktion:** Noten festlegen

**Syntax:** `MUSIC Tondauer,"Noten"`

Mit MUSIC kann eine Melodie aufgebaut werden. Die Spieldauer der Töne kann dabei mit Werten von 1 (kurz) bis 255 (lang) festgelegt werden. Die Zeichenkette "Noten" kann zuvor einer Zeichenkettenvariablen zugewiesen werden und wird wie folgt zusammengestellt:

| | |
|---|---|
| SHIFT/CLR | 1 - Stimme 1 |
| | 2 - Stimme 2 |
| | 3 - Stimme 3 |
| F1 | die folgende Note ist 1/16 Note |
| F3 | die folgende Note ist 1/8 Note |
| F5 | die folgende Note ist 1/4 Note |
| F7 | die folgende Note ist 1/2 Note |
| F2 | die folgende Note ist 1/1 Note |
| F4 | die folgende Note ist 2/1 Note |
| F6 | die folgende Note ist 4/1 Note |
| F8 | die folgende Note ist 8/1 Note |

Die Noten müssen mit einem Namen (C-H) und einer Oktave (0-7) eingegeben werden. Halbtöne erreicht man durch gleichzeitiges Drücken des Namens mit der SHIFT-Taste. Als letztes in der Zeichenkette muß die Kombination SHIFT/CLR-G eingegeben werden.

## NO ERROR (Fehlerbehandlung)

**Funktion:** Unterdrücken einer Fehlermeldung

**Syntax:** `NO ERROR`

Mit diesem Befehl kann einmalig die Ausgabe einer Fehlermeldung unterdrückt werden.

## **NRM** (Grafikbefehl)

**Funktion:** Umkehrfunktion zu HIRES oder MULTI

**Syntax:** `NRM`

Der NRM-Befehl schaltet nach einem HIRES-Befehl wieder zurück in den Textmodus mit normal auflösendem Bildschirm.

## **OFF** (Bildschirmbefehl)

**Funktion:** Umkehrfunktion zu FLASH

**Syntax:** `OFF`

OFF stellt das Bildschirm-Blinken, das mit dem FLASH-Befehl ausgelöst wurde, wieder ab.

## **OLD** (Programmierhilfe)

**Funktion:** Umkehrfunktion zu NEW

**Syntax:** `OLD`

Da nach Eingabe des Befehls NEW das Programm nicht völlig verschwunden ist, kann es mit Hilfe des Befehls OLD wieder zurückgeholt werden. Nach Eingabe einer Programmzeile hilft er allerdings nicht mehr.

## **ON ERROR** (Fehlerbehandlung)

**Funktion:** Sprung nach Programmfehler

**Syntax:** `ON ERROR: GOTO Zeilennummer`

Wenn ein Programmfehler auftritt, veranlaßt der Befehl ON ERROR, daß zu der angegebenen Zeilennummer verzweigt wird, in der eine Fehlerbehandlungsroutine beginnen könnte.

## ON KEY (Eingabebefehl)

**Funktion:** Sprungbefehl nach Tastendruck

**Syntax:**
```
ON KEY "Zeichenkette",:GOTO Zeilennummer

ON KEY Zeichenkettenvariable,:
GOTO Zeilennummer
```

Wenn eine einzige Taste aus der angegebenen Zeichenkette gedrückt wird, fährt das Programm in der Zeile fort, die durch die Nummer hinter GOTO bezeichnet wird.

```
10 ON KEY "DA",:GOTO 30
20 GOTO 20
30 PRINT "ENDE"
```

Das Programm schreibt das Wort ENDE erst, wenn die Taste D oder die Taste A gedrückt wird. Auf jeden anderen Tastendruck erfolgt keine Reaktion.

Wenn der Rest des Programmablaufs nicht vom ON KEY-Befehl abhängig sein soll, kann dieser mit DISABLE aufgehoben werden.

## OPTION (Programmierhilfe)

**Funktion:** Befehle aus Simon's BASIC optisch hervorheben

**Syntax:**
```
OPTION 10
```

Wenn OPTION 10 angegeben wird, werden alle Befehle aus Simon's BASIC beim Listen revers dargestellt. OPTION mit einem beliebigen Parameter schaltet diesen Modus wieder aus.

## OUT (Fehlerbehandlung)

**Funktion:** Standard-Fehlermeldungen wieder einschalten

**Syntax:**
```
OUT
```

Es erscheinen bei Auftreten von Programmfehlern wieder die üblichen Fehlermeldungen des C64 und nicht mehr die speziellen von Simon's BASIC.

## PAGE (Programmierhilfe)

**Funktion:** Einteilen eines Programms in Teilstücke

**Syntax:** `PAGE Zeilenanzahl`

Mit PAGE kann ein Programm zum Listen in Teilstücke zerlegt werden. Es werden immer nur so viele Zeilen gezeigt, wie durch den Parameter Zeilenanzahl (maximal 255) festgelegt wurde. Das nächste Programmstück erreicht man dann durch Drücken von <RETURN>. PAGE0 schaltet in das normale Listen zurück.

## PAINT (Grafikbefehl)

**Funktion:** Fläche farbig ausfüllen

**Syntax:** `PAINT X,Y,Zeichenfarbe`

Die geschlossene Fläche, in der der Punkt X,Y liegt, wird in der Zeichenfarbe ausgefüllt.

## PAUSE (Programmierhilfe)

**Funktion:** Programmausführung verzögern

**Syntax:**
```
PAUSE Sekunden
PAUSE "Text",Sekunden
```

Mit PAUSE kann die Programmausführung für die Länge der angegebenen Sekunden unterbrochen werden. Dabei kann ein erläuternder Text zuvor ausgegeben werden. Falls man die Pause früher abbrechen möchte, kann man das durch Drücken von <RETURN> erreichen.

## PENX (Lichtgriffelbefehl)

**Funktion:** X-Position des Lichtgriffels ermitteln

**Syntax:** `Variable=PENX`

Der Variablen wird die X-Koordinate (0 bis 320) der Lichtgriffel-Position zugewiesen.

## PENY (Lichtgriffelbefehl)

**Funktion:** Y-Position des Lichtgriffels ermitteln

**Syntax:**
```
Variable=PENY
```

Der Variablen wird die Y-Koordinate (0 bis 200) der Lichtgriffel-Position zugewiesen.

## PLACE (Zeichenketten-Operation)

**Funktion:** Position einer Zeichenkette in einer anderen suchen

**Syntax:**
```
PLACE ("Zeichenkette1","Zeichenkette2")
```

PLACE sucht die Position, ab der sich die Zeichenkette1 in der Zeichenkette2 befindet. Wird sie nicht gefunden, liefert PLACE den Wert 0.

## PLAY (Soundbefehl)

**Funktion:** Wiedergabe einer Melodie

**Syntax:**
```
PLAY 0
PLAY 1
PLAY 2
```

Die Melodie, die mit MUSIC komponiert wurde, wird gespielt, und die Programmausführung wird fortgesetzt (PLAY 2) oder hält an (PLAY 1). Mit PLAY 0 wird die Musik abgeschaltet.

## PLOT (Grafikbefehl)

**Funktion:** Bildpunkte setzen

**Syntax:**
```
PLOT X,Y,Zeichenfarbe
```

Der PLOT-Befehl setzt einen Punkt an der angegebenen Koordinate. Dabei kann mit dem dritten Parameter die gewünschte Farbe bestimmt werden. Der Wert für X muß im Bereich von 0 bis 319 und für Y von 0 bis 199 liegen.

## **POT** (Paddlebefehl)

**Funktion:** Widerstand der Paddles ermitteln

**Syntax:**
```
Variable=POT(0)
Variable=POT(1)
```

Die Stellung (0 bis 255) der Paddles 0 und 1 wird in der Variablen festgelegt.

## **PROC** (Anweisung)

**Funktion:** Unterprogramm mit Namen versehen

**Syntax:** `PROC Marke`

Mit PROC wird einem Programmabschnitt ein Name zugewiesen, über den dieser Teil dann als Unterprogramm über CALL Marke oder EXEC Marke angesprungen werden kann. Das Ende des Unterprogramms wird durch END PROC gekennzeichnet.

## **RCOMP** (Anweisung)

**Funktion:** Bedingung der letzten IF...THEN..ELSE-Abfrage prüfen

**Syntax:** `RCOMP: Anweisung1: ELSE: Anweisung2`

Wenn die Bedingung der letzten IF...THEN...ELSE-Abfrage wahr ist, dann wird Anweisung1 ausgeführt, andernfalls Anweisung2.

## **REC** (Grafikbefehl)

**Funktion:** Rechteck zeichnen

**Syntax:** `REC X,Y,A,B,Zeichenfarbe`

Das Rechteck mit den Seitenlängen A und B wird, beginnend an der linken oberen Ecke (Koordinaten X und Y), in der angegebenen Zeichenfarbe gezeichnet.

## RENUMBER (Programmierhilfe)

**Funktion:** Zeilennummern umnumerieren

**Syntax:** `RENUMBER Startnummer,Schrittweite`

Mit RENUMBER werden die Zeilen eines Programms in der angegebenen Schrittweite und beginnend mit der Startnummer neu durchnumeriert.

*Achtung:* Zeilennummern nach GOTO und GOSUB usw. bleiben dabei unberücksichtigt.

## REPEAT...UNTIL (Anweisung)

**Funktion:** Schleife

**Syntax:** `REPEAT: Anweisungen: UNTIL Bedingung erfüllt`

Die Anweisungen hinter REPEAT werden so lange wiederholt, bis die Bedingung, die nach UNTIL angegeben ist, erfüllt ist.

## RESET (Programmierhilfe)

**Funktion:** DATA-Zeiger auf bestimmte DATA-Zeile setzen

**Syntax:** `RESET DATA-Zeilennummer`

Der READ-Befehl liest alle DATA-Angaben sequentiell. Mit RESET und Angabe einer Zeilennummer wird veranlaßt, daß der nächste READ-Befehl in dieser DATA-Zeile weiterliest.

## RESUME (Eingabebefehl)

**Funktion:** Aktivieren des vorausgehenden ON KEY-Befehls

**Syntax:** `RESUME`

Wenn im Programm der ON KEY-Befehl verwendet wird, kann er zwischenzeitlich mit DISABLE abgeschaltet und mit RESUME dann wieder aktiviert werden.

## RETRACE (Programmierhilfe)

**Funktion:** Anzeigen des letzten TRACE-Bildschirms

**Syntax:** `RETRACE`

Wenn bei einem Programmabbruch der Bildschirm gelöscht wird, kann durch RETRACE die Anzeige des letzten TRACE-Fensters wiederholt werden.

## RIGHTB, RIGHTW (Bildschirmbefehl)

**Funktion:** Bildschirmbereich rollen

**Syntax:**
```
RIGHTB erste Zeile,erste Spalte,
letzte Zeile,letzte Spalte

RIGHTW erste Zeile,erste Spalte,
letzte Zeile,letzte Spalte
```

Der durch die Zeilen und Spalten festgelegte Bildschirmausschnitt wird um eine Spalte nach rechts gerollt. Dabei wird der Inhalt aus dem Bereich hinausgeschoben und ist nicht mehr sichtbar (RIGHTB), oder er wird hinausgeschoben und kommt an der anderen Seite wieder zum Vorschein (RIGHTW).

## RLOCMOB (Spritebefehl)

**Funktion:** Sprite auf dem Bildschirm bewegen

**Syntax:**
```
RLOCMOB Spritenummer,X,Y,Größe,
Geschwindigkeit
```

Ein auf dem Bildschirm dargestelltes Sprite mit der angegebenen Nummer kann zu der mit X,Y bezeichneten Stelle auf dem Bildschirm bewegt werden. Die Größe des Sprites kann dabei verdoppelt werden:

0 normale Größe
1 doppelte Größe in X-Richtung
2 doppelte Größe in Y-Richtung
3 doppelte Größe in X- und Y-Richtung

Die Geschwindigkeit variiert von 1 (langsam) bis 255 (schnell).

## ROT (Grafikbefehl)

**Funktion:** Figur drehen

**Syntax:** `ROT Drehwinkel,Figurgröße`

Eine mit DRAW erstellte Figur kann mit dem ROT-Befehl gedreht und vergrößert dargestellt werden. Der Parameter für die Größe kann 1 bis 255 betragen, und die Werte für den Drehwinkel können Sie der folgenden Tabelle entnehmen:

| Drehwinkel | Realer Winkel in Grad |
|---|---|
| 0 | 0 |
| 1 | 45 |
| 2 | 90 |
| 3 | 135 |
| 4 | 180 |
| 5 | 225 |
| 6 | 270 |
| 7 | 315 |

## SCRLD (Bildschirmbefehl)

**Funktion:** Bildschirmdaten laden

**Syntax:**
```
SCRLD 2,8,2,"Dateiname"
SCRLD 1,1,1,"Dateiname"
```

Die mit SCRSV gespeicherten Bildschirmdaten können mit SCRLD wieder von der Diskette (1. Fall) oder von der Kassette (2. Fall) geladen werden.

## SCRSV (Bildschirmbefehl)

**Funktion:** Bildschirmdaten speichern

**Syntax:**
```
SCRSV 2,8,2,"Dateiname,S,W"
SCRSV 1,1,1,"Dateiname"
```

Die Bildschirmdaten eines normal auflösenden Bildschirms können unter dem angegebenen Dateinamen als sequentielle Datei auf Diskette (1. Fall) oder auf

Kassette (2. Fall) abgespeichert werden. Mit SCRLD kann die Datei wieder geladen werden.

## SECURE (Programmierhilfe)

**Funktion:** Schützen von Programmzeile

**Syntax:** `SECURE Dummy-Parameter`

Der Parameter kann einen beliebigen Wert von 0 bis 255 annehmen. Wenn eine Programmzeile mit DISAPA beginnt, wird sie nach der Eingabe von SECURE beim Listen bis auf ihre Zeilennummer unsichtbar bleiben.

## SOUND (Soundbefehl)

**Funktion:** Liefert die Zahl 53972

**Syntax:** `Variable=SOUND`

In der Variablen wird die Konstante 53972, die Adresse des SID, abgelegt.

## TEST (Grafikbefehl)

**Funktion:** Punktposition bestimmen

**Syntax:** `Variable=TEST(X,Y)`

Die angegebene Variable erhält den Wert 1, wenn an den getesteten Koordinaten (X,Y) ein Punkt sitzt; sonst ist sie 0.

## TEXT (Textbefehl)

**Funktion:** Zeichenkette auf Grafikbildschirm schreiben

**Syntax:** `TEXT X,Y,"Text",Zeichenfarbe,Größe,Abstand`

Mit dem Befehl TEXT kann ein angegebener Text mit der linken oberen Ecke an den Koordinaten X,Y auf dem Grafikbildschirm ausgegeben werden. Wird dabei in der Zeichenkette unmittelbar hinter dem ersten Anführungszeichen CTRL-A gedrückt, erscheint der Text in Großschrift, mit CTRL-B erscheint

er in Kleinschrift. Mit Größe kann der Text nach unten gedehnt werden, und mit Abstand wird der Zwischenraum zwischen den einzelnen Zeichen bestimmt.

## TRACE (Programmierhilfe)

**Funktion:** Anzeigen der gerade bearbeiteten Programmzeilennummer

**Syntax:** `TRACE 10`

Wenn vor einem Programmstart der Befehl TRACE 10 eingegeben wird, erscheint nach der Eingabe von RUN in der rechten oberen Bildschirmecke ein Fenster, in dem die letzten Zeilennummern des Programms angezeigt sind. Als unterste Zahl erscheint zusätzlich die Zeilennummer der aktuell bearbeiteten Zeile.

## UPB, UPW (Bildschirmbefehl)

**Funktion:** Bildschirmbereich rollen

**Syntax:**

```
UPB erste Zeile,erste Spalte,letzte Zeile,
letzte Spalte

UPW erste Zeile,erste Spalte,letzte Zeile,
letzte Spalte
```

Der durch die Zeilen und Spalten festgelegte Bildschirmausschnitt wird um eine Zeile nach oben gerollt. Dabei wird der Inhalt aus dem Bereich hinausgeschoben und ist nicht mehr sichtbar (UPB), oder er wird hinausgeschoben und kommt auf der anderen Seite wieder zum Vorschein (UPW).

## USE (Zeichenketten-Operation)

**Funktion:** Numerische Werte für die Ausgabe formatieren

**Syntax:** `USE "Format","Zeichenkette":PRINT`

Die Zeichenkette wird in dem vorgegebenen Format ausgegeben. Das Format wird dabei so aufgebaut, daß für jede Ziffer ein Nummernzeichen # steht. Zusätzlich gibt es den Dezimalpunkt. Das Format kann auch vorher einer Variablen zugewiesen worden sein.

```
10 A$=STR$(10.5↑5)
20 USE "###.##",A$:PRINT
```

Das Ergebnis ist 628.15.

## VOL (Soundbefehl)

**Funktion:** Einstellen der Lautstärke

**Syntax:** `VOL Lautstärke`

Mit diesem Befehl wird für alle Stimmen die Lautstärke mit Werten von 0 (aus) bis 15 (laut) festgelegt.

## WAVE (Soundbefehl)

**Funktion:** Festlegen der Stimme und der Wellenform

**Syntax:** `WAVE Stimme,Wellenform`

Mit WAVE werden die Stimme und die Wellenform festgelegt. Die Stimme kann Werte von 1 bis 3 annehmen und der Parameter für die Wellenform Werte von 0 bis 7.

## % (Umwandlungsfunktion)

**Funktion:** Binärzahl in Dezimalzahl umwandeln

**Syntax:** `PRINT % 8-stellige Binärzahl`

Die %-Funktion liefert den Wert der angegebenen 8-stelligen Binärzahl (nur 0 oder 1 sind als Ziffern zugelassen).

## $ (Umwandlungsfunktion)

**Funktion:** Hexadezimalzahl in Dezimalzahl umwandeln

**Syntax:** `PRINT $ 4-stellige Hexadezimalzahl`

Die $-Funktion liefert den Wert der angegebenen 4-stelligen Hexadezimalzahl (nur Ziffern 0 bis 1 und Buchstaben A bis F sind zugelassen).

## Anhang B

# Befehlserweiterung EXBASIC LEVEL II

Mit dieser Befehlserweiterung steht uns auf den Commodore-Rechnern 2000, 3000, 4000, 8000, C64 und C128 im 64er-Modus ein komfortabler Befehlssatz zur Verfügung. 57 zusätzliche Befehle ermöglichen die professionelle Programmierung in BASIC 2.0 und 4.0.

Im folgenden geben wir eine alphabetische Befehlsübersicht:

### ADSR (Soundbefehl) nur C64

**Funktion:** Tongenerator einstellen

**Syntax:**
```
ADSR Stimme,Wellenform,A,D,S,R
ADSR Stimme,Wellenform,A,D,S,R,Pulsweite
```

Die Parameter dieses Befehls haben folgende Bedeutung:

| | |
|---|---|
| **Stimme** | Werte von 1 bis 3 |
| **Wellenform** | 17 Dreicksform<br>33 Sägezahn<br>65 Rechteckschwingung<br>129 Rauschen |
| **A** (Attack) | Werte von 0 bis 15 |
| **D** (Decay) | Werte von 0 bis 15 |
| **S** (Sustain) | Werte von 0 bis 15 |
| **R** (Release) | Werte von 0 bis 15 |
| **Pulsweite** | Werte von 0 bis 4095 (nur bei Rechteckschwingung) |

### AUTO (Programmierhilfe)

**Funktion:** Automatische Zeilennummernvorgabe

**Syntax:**
```
AUTO
AUTO Startnummer
AUTO Startnummer,Schrittweite
```

Mit AUTO werden für die Programmeingabe automatisch Zeilennummern vorgegeben. Existiert bereits eine Zeile mit dieser Nummer, so wird das durch einen * hinter der Zeilennummer kenntlich gemacht.

## **BASIC** (Programmierhilfe)

**Funktion:** Umschalten ins Commodore-BASIC

**Syntax:** `BASIC`

Mit dem Befehl BASIC kommt man ins normale Commodore-BASIC zurück. Um wieder EXBASIC verwenden zu können, brauchen Sie dann nur PRINT USR(0) einzugeben.

## **BEEP** (Soundbefehl) nicht C64

**Funktion:** Signalton ausgeben

**Syntax:**
```
BEEP
BEEP Tonhöhe
BEEP Tonhöhe,Tondauer
```

Mit BEEP kann ein Signalton mit der angegebenen Tonhöhe (0 - hoch bis 255 - tief) und Tondauer (0 -kurz bis 255 - lang) gesendet werden.

Durch Drücken der STOP-Taste bekommt man die Kontrolle wieder, und kann dann den Ton mit BEEP OFF abschalten.

## **BEGINLINE** (Bildschirmbefehl) nur 8001

**Funktion:** Bildschirmzeichen löschen

**Syntax:** `BEGINLINE`

Alle Zeichen in der Bildschirmzeile, die links vom Cursor (ausschließlich) stehen, werden gelöscht.

## **BORDER** (Grafikbefehl) nur C64

**Funktion:** Bildschirm-Rahmenfarbe festlegen

**Syntax:** `BORDER Farbnummer`

Mit dem BORDER-Befehl kann die Farbe des Bildschirmrahmens über eine Farbnummer (1 bis 16) gewählt werden.

## CALL (Befehlswort)

**Funktion:** Maschinenprogrammaufruf mit Parameterübergabe

**Syntax:** `CALL (Parameterliste)`

Mit CALL wird ein Maschinenprogramm aufgerufen. Dabei können im Gegensatz zu SYS Parameter übergeben werden. Die Einsprungadresse muß zuvor mit DEF CALL festgelegt worden sein.

## CEEK (Grafikbefehl) nur C64

**Funktion:** Zeichen vom Bildschirm lesen

**Syntax:**
```
CEEK Position,S
CEEK Position,C
```

Mit CEEK wird der Zeichencode des Zeichens an der angegebenen Position ermittelt (Modus S) oder seine Farbnummer (Modus C).

## COKE (Grafikbefehl) nur C64

**Funktion:** Zeichen auf den Bildschirm setzen

**Syntax:** `COKE Position,Zeichencode,Farbnummer`

Mit COKE wird an der angegebenen Position (0 bis 999) das durch den Code bestimmte Zeichen in der gewählten Farbe (1 bis 16) auf den Bildschirm gesetzt.

## CURSOR (Grafikbefehl) nur C64

**Funktion:** Cursorfarbe bestimmen

**Syntax:** `CURSOR Farbnummer`

Mit dem CURSOR-Befehl kann eine Cursorfarbe von 1 bis 16 gewählt werden.

## DEC (Mathematische Funktion)

**Funktion:** Umwandlung in Dezimalzahl

**Syntax:** `DEC ("Hexadezimalzahl")`

Mit DEC wird die hexadezimale Zeichenkette in eine Dezimalzahl umgewandelt. Die Hexadezimalzahl muß entweder zwei- oder vierstellig sein und sollte, wenn das Ergebnis sinnvoll sein soll, nur die Ziffern 0 bis 9 und die Buchstaben A bis F enthalten.

## DEEK (Befehlswort)

**Funktion:** Zwei aufeinanderfolgende Speicherstellen auslesen

**Syntax:** `DEEK (Adresse)`

DEEK arbeitet so ähnlich wie PEEK, nur daß gleich der Inhalt zweier aufeinanderfolgender Speicherstellen ausgelesen wird, und zwar die an der angegebenen Adresse und die an der Adresse + 1.

Den ausgelesenen Wert müssen Sie folgendermaßen interpretieren: Der ganzzahlige Anteil des Wertes nach der Division durch 256 (höherwertiges Byte) ist der Inhalt der oberen Speicherstelle, also Adresse + 1. Der Rest der Division war der Inhalt der unteren Speicherstelle (Adresse).

### Beispiel:

Wenn Sie mit

```
PRINT DEEK (75)
```

den Wert 767 erhalten, dann stand in Speicherstelle 75 der Wert 255 und in Speicherstelle 76 der Wert 2, denn

```
767 = 2 * 256 + 255
```

Diese Ergebnisse können Sie mit

```
PRINT PEEK (75)
PRINT PEEK (76)
```

ganz leicht nachprüfen.

## DEF CALL (Anweisung)

**Funktion:** Adresse für Maschinenprogrammaufruf festlegen

**Syntax:** `DEF CALL = Adresse`

Mit CALL(Parameterliste) kann ein Maschinenprogramm mit Übergabe von mehreren Parametern aufgerufen werden. Zuvor muß aber die Einsprungadresse für CALL mit DEF CALL festgelegt werden, da ja beim Aufruf CALL(Parameterliste) keine Adresse mitgegeben werden kann.

Welche Adresse zur Zeit für CALL festgelegt ist, können Sie mit

```
PRINT DEEK(1021)
```

erfahren.

## DEF USR (Anweisung)

**Funktion:** Adresse für Maschinenprogrammaufruf festlegen

**Syntax:** `DEF USR = Adresse`

Mit USR(Parameter) kann ein Maschinenprogramm mit Parameterübergabe aufgerufen werden. Zuvor muß aber die Einsprungadresse für USR im USR-Vektor mit DEF USR festgelegt werden, da ja beim Aufruf USR(Parameter) keine Adresse mitgegeben werden kann.

Welche Adresse zur Zeit für USR festgelegt ist, können Sie mit

```
PRINT DEEK(1)
```

erfahren.

## DEL (Programmierhilfe)

**Funktion:** Programmteile löschen

**Syntax:** `DEL Zeilennummern`

Mit Angabe von Zeilennummern wird ein ganzer Teilbereich eines Programms auf einmal gelöscht.

## DELLINE (Bildschirmbefehl) nur 8001

**Funktion:** Zeile löschen

**Syntax:** `DELLINE`

Die Bildschirmzeile, in der sich der Cursor befindet, wird gelöscht.

## DISPOSE (Anweisung)

**Funktion:** Stapel leeren

**Syntax:**
```
DISPOSE NEXT
DISPOSE NEXT Variable
DISPOSE RETURN
DISPOSE CLR
```

Wenn im Programm Schleifen oder Unterprogramme bearbeitet werden, wird intern auf dem Stapel (Stack) eine Rücksprungadresse abgelegt. Ein Stapel hat immer eine LIFO-Struktur (Last In - First Out), d. h. die zuletzt abgelegte Information muß als erste wieder herausgenommen werden. Diese Schachtelungsebenen muß man auch beim Programmieren berücksichtigen.

Mit DISPOSE NEXT oder DISPOSE NEXT Variable wird die letzte FOR... NEXT-Schleife vom Stapel genommen, ohne daß im Programm das zugehörige NEXT oder NEXT Variable durchlaufen wird.

DISPOSE RETURN holt das nächste Unterprogramm vom Stapel und veranlaßt einen Sprung auf die nächsthöhere Ebene. Das kann auch das Hauptprogramm sein.

Mit DISPOSE CLR schließlich wird der ganze Stapel auf einmal geleert, also alle Schleifen geschlossen, und auf die Hauptprogrammebene verzweigt.

## **DOKE** (Befehlswort)

**Funktion:** In zwei aufeinanderfolgende Speicherstellen schreiben

**Syntax:** `DOKE Adresse,Wert`

DOKE funktioniert so ähnlich wie POKE, nur daß mit DOKE gleichzeitig zwei aufeinanderfolgende Speicherstellen angesprochen werden, und zwar Adresse und Adresse + 1.

Damit Sie sich vorstellen können, welche Werte in diese beiden Speicherstellen geschrieben werden, dividieren Sie den hinter DOKE angegebenen Wert durch 256 (höherwertiges Byte). Der ganzzahlige Anteil dieser Division wird in die Speicherstelle mit der Adresse + 1 geschrieben, der Rest in die Speicherstelle mit der Adresse.

**Beispiel:**

```
DOKE 128,800
```

schreibt die Zahl 3 an die Adresse 129 und die Zahl 32 an die Adresse 128, denn

```
800 = 3 * 256 + 32
```

Prüfen Sie die Inhalte der beiden Speicherstellen doch mit

```
PRINT PEEK (128)
PRINT PEEK (129)
```

nach.

## **DUMP** (Programmierhilfe)

**Funktion:** Variablenwerte zeigen

**Syntax:** `DUMP`

Die Werte von allen nicht indizierten Variablen werden in der Reihenfolge ihres Auftretens im Programm aufgelistet.

## ENDLINE (Bildschirmbefehl) nur 8001

**Funktion:** Bildschirmzeichen löschen

**Syntax:** `ENDLINE`

Alle Zeichen in der Bildschirmzeile rechts vom Cursor (einschließlich) werden gelöscht.

## EVAL (Zeichenkettenfunktion)

**Funktion:** Zeichenkettenausdruck in Zahl umwandeln

**Syntax:** `EVAL ("Zeichenkettenausdruck")`

EVAL ist die Verbesserung des Befehls VAL aus dem normalen Commodore-BASIC. Während VAL nur den Wert der ersten Zahl in einem Zeichenkettenausdruck berechnet, kann EVAL den ganzen Ausdruck berechnen und dessen Wert ausgeben:

```
PRINT VAL ("25+25")              ergibt 25
10 PRINT EVAL ("25+25")          ergibt 50
```

EVAL kann nicht im Direktmodus verwendet werden.

## EXEC (Befehlswort)

**Funktion:** BASIC-Befehl ausführen

**Syntax:** `EXEC ("BASIC-Befehl")`

Mit EXEC kann im Programmodus jeder beliebige BASIC-Befehl ausgeführt werden. Hinter EXEC darf immer nur ein Befehl stehen, und EXEC darf sich nicht selbst aufrufen. Bei einem PRINT-Befehl kann es Probleme geben, wenn Sie einen Text in Anführungszeichen ausgeben wollen:

```
100 EXEC ("PRINT"WIESO"")
```

In diesem Fall müssen Sie die Zeichenkette WIESO vorher einer Variablen zuweisen:

```
100 A$="WIESO"
101 EXEC ("PRINT A$")
```

## **FAST** (Programmierhilfe) nicht C64

**Funktion:** Bildschirm-Ausgabegeschwindigkeit erhöhen

**Syntax:**
```
FAST
FAST OFF
```

Das Listen und auch PRINT-Anweisungen werden auf dem Bildschirm mit einer höheren Geschwindigkeit als normal ausgeführt. Mit FAST OFF wird in die ursprüngliche Geschwindigkeit zurückgeschaltet.

## **FIND** (Programmierhilfe)

**Funktion:** BASIC-Wort, Text, Variable in Programmbereich suchen

**Syntax:**
```
FIND Suchwort,Zeilennummern
FIND Suchwort
```

Mit FIND können Sie BASIC-Wörter, Text (in Anführungszeichen) und Variablen suchen. Dabei kann die Suche auf einen bestimmten Bereich des Programms eingeschränkt werden. Ausgegeben werden alle Zeilen, in denen das Suchwort vorkommt.

## **FRAC** (Mathematische Funktion)

**Funktion:** Nachkommastellen einer Dezimalzahl

**Syntax:**
```
FRAC(Z)
```

FRAC liefert nur die Stellen, die hinter dem Dezimalpunkt stehen, und zwar maximal 9-stellig.

## **GO** (Programmierhilfe) nicht C64

**Funktion:** Umschalten in den Maschinensprache-Monitor

**Syntax:**
```
GO
```

Aus dem BASIC springt dieser Befehl in den Maschinensprache-Monitor, auf dessen Kommandos wir hier nicht weiter eingehen wollen. Zurückkehren können Sie mit X.

## GROUND (Grafikbefehl) nur C64

**Funktion:** Bildschirm-Hintergrundfarbe festlegen

**Syntax:** `GROUND Farbnummer`

Mit dem GROUND-Befehl kann die Hintergrundfarbe von 1 bis 16 gewählt werden.

## HARDCOPY (Bildschirmbefehl)

**Funktion:** Bildschirminhalt ausdrucken

**Syntax:** `HARDCOPY`

Mit diesem Befehl kann der Bildschirminhalt auf dem Drucker ausgegeben werden.

## HELP (Programmierhilfe)

**Funktion:** Kommandos anzeigen

**Syntax:**
```
HELP
HELP *
```

Mit HELP werden Kommandos von EXBASIC auf dem Bildschirm aufgelistet. Bei HELP * kommen noch die normalen BASIC-Schlüsselwörter Ihres Rechners mit dazu.

## HEX$ (Mathematische Funktion)

**Funktion:** Umwandlung in Hexadezimalzahl

**Syntax:** `HEX$ (Zahl)`

Mit HEX$ können Zahlen in ihr hexadezimales Äquivalent umgewandelt werden. Für Zahlen bis 255 ist das Ergebnis zweistellig, und für Zahlen von 256 bis 65535 (maximal zulässige Zahl) ist das Ergebnis vierstellig.

## HIMEM (Programmierhilfe)

**Funktion:** BASIC-Bereich nach oben begrenzen

**Syntax:** `HIMEM Adresse`

Dem BASIC-Bereich steht nach dem Befehl nur noch der Bereich ab Adresse 1024 bis zur angegebenen Endadresse zur Verfügung. Die Adresse nach HIMEM muß mindestens 1260 lauten (2058 für C64) und höchstens 32768 (65535 für C64).

## HPLOT (Grafikbefehl)

**Funktion:** Horizontale Balken zeichnen

**Syntax:**
```
HPLOT numerischer Ausdruck
HPLOT numerischer Ausdruck,Farbe    nur C64
```

Mit HPLOT lassen sich waagerechte Balken auf einem Bildschirm mit einer Auflösung von 320 x 25 Zeichen (40-Zeichen-Bildschirm) bzw. vom 640 x 25 Zeichen (80-Zeichen-Bildschirm) zeichnen. Startpunkt ist die momentane Cursorposition, und die Länge des Balkens richtet sich nach dem Ergebnis des numerischen Ausdrucks. Dieser kann jeder mathematische Ausdruck sein, der als Ergebnis eine Zahl liefert, die in unserem Beispiel höchstens 319 bzw. 639 betragen darf.

Beim C64 haben Sie zusätzlich die Möglichkeit, die Farbe (1 bis 16) des Balkens zu bestimmen.

## IF...THEN...ELSE (Anweisung)

**Funktion:** Bedingung prüfen und Anweisungen ausführen

**Syntax:** `IF Bedingung THEN Anweisung1 ELSE Anweisung2`

Die IF...THEN-Anweisung aus dem Commodore-BASIC ist auch in EXBASIC um die sinnvolle Ergänzung ELSE bereichert worden. Die Anweisung nach ELSE wird ausgeführt, wenn die Bedingung, die hinter IF steht, nicht erfüllt ist.

## INPUTFORM (Eingabebefehl)

**Funktion:** Eingabe einer Zeichenkette

**Syntax:**
```
INPUTFORM Zeichenkettenvariable
INPUTFORM "Text";Zeichenkettenvariable
INPUTFORM "Text";Zeichenkettenvariable,Länge
```

INPUTFORM arbeitet ähnlich wie INPUT. Allerdings sind hier als Eingabezeichen auch Anführungszeichen, Kommas usw. erlaubt. Der Cursor erscheint als feststehender Unterstrich, und es wird kein Fragezeichen als Eingabeaufforderung ausgegeben. Zusätzlich kann eine Länge festgelegt werden, die die Anzahl der Eingabezeichen beschränkt.

## INPUTLINE (Eingabebefehl)

**Funktion:** Eingabe einer Zeichenkette

**Syntax:**
```
INPUTLINE Zeichenkettenvariable
INPUTLINE "Text";Zeichenkettenvariable
```

INPUTLINE arbeitet ähnlich wie INPUT. Allerdings sind hier als Eingabezeichen auch Anführungszeichen, Doppelpunkte, Kommas usw. erlaubt. INPUTLINE gibt kein Fragezeichen als Eingabeaufforderung aus und liest nur eine einzige Zeichenkette von maximal 79 ein.

## INSTR (Zeichenkettenfunktion)

**Funktion:** Position einer Zeichenkette in einer anderen bestimmen

**Syntax:**
```
INSTR (Zeichenkette1,Zeichenkette2)

INSTR (Zeichenkette1,Zeichenkette2,
Startposition)
```

Die Zeichenkette1 wird in der Zeichenkette2 gesucht, und das Ergebnis ist die Position, ab der sie in ihr vorkommt. Es gibt dabei die Möglichkeit, eine Startposition in der zu durchsuchenden Zeichenkette2 anzugeben.

## INSTLINE (Bildschirmbefehl) nur 8001

**Funktion:** Bildschirmzeile einfügen

**Syntax:** `INSTLINE`

Oberhalb der Zeile, in der der Cursor steht, wird eine leere Bildschirmzeile eingeschoben.

## KEY (Programmierhilfe) nur C64

**Funktion:** Funktionstastenbelegungsanzeige

**Syntax:** `KEY`

Mit KEY können Sie sich die Belegung der 8 Funktionstasten des C64 anzeigen lassen:

```
KEY1,"LIST"
KEY2,"AUTO"
KEY3,"DUMP ←"
KEY4,"MATRIX ←"
KEY5,"RUN ←"
KEY6,"GOTO"
KEY7,"FIND"
KEY8,"MEM ←"
```

← steht für die RETURN-Taste und bedeutet, daß der Befehl nach Drücken der Funktionstaste auch sofort ausgeführt wird.

Mit KEYNummer,"BASIC-Befehl" können Sie die Belegung der einzelnen Tasten auch verändern.

Mit KEY ON läßt sich die ursprüngliche Belegung wieder herstellen.

## LETTER (Programmierhilfe)

**Funktion:** Schriftmodus umschalten

**Syntax:**
```
LETTER
LETTER OFF
```

Mit LETTER wird der Rechner auf Groß-/Kleinschrift umgeschaltet. Mit LETTER OFF wird er wieder in Großschrift/Grafik zurückgeschaltet. Je nach Einschaltmodus bei den verschiedenen Geräten funktionieren die Befehle genau andersherum.

## MATRIX (Programmierhilfe)

**Funktion:** Variablenwerte zeigen

**Syntax:** `MATRIX`

Die Variablenwerte aller indizierten Variablen werden in der Reihenfolge ihres Auftretens im Programm aufgelistet.

## MAX (Mathematische Funktion)

**Funktion:** Maximum suchen

**Syntax:** `MAX (Variablenliste)`

MAX liefert den größten Wert aus der angegebenen Variablenliste.

## MEM (Programmierhilfe)

**Funktion:** Speicherplatz anzeigen

**Syntax:** `MEM`

Mit MEM bekommen Sie genaue Auskunft über verbrauchten und noch vorhandenen Speicherplatz:

```
MEMORY:      30719 BYTES   Speicherplatz insgesamt
PROGRAM:     2     BYTES   Speicherplatz für Programm
VARIABLES:   0     BYTES   Speicherplatz für Variablen
ARRAYS:      0     BYTES   Speicherplatz für Felder
STRINGS:     0     BYTES   Speicherplatz für Zeichenketten
REK:         0     BYTES   REK-Speicherplatz (entfällt beim C64)
FREE:        30717 BYTES   Speicherplatz noch verfügbar
```

## MERGE (Programmierhilfe)

**Funktion:** Programm dazuladen

**Syntax:**
```
MERGE
MERGE "Dateiname"
MERGE* "Dateiname"
```

Mit MERGE können Sie ein weiteres Programm zu dem im Arbeitsspeicher befindlichen hinzuladen. Achten Sie dabei aber auf geeignete Zeilennumerierung. Für das Laden von Kassette geben Sie MERGE (das nächste Programm wird geladen) bzw. MERGE mit einem bestimmten Namen ein.

Befindet sich das Programm auf Diskette, so müssen Sie MERGE* und den entsprechenden Dateinamen angeben.

## MIN (Mathematische Funktion)

**Funktion:** Minimum suchen

**Syntax:** `MIN (Variablenliste)`

MIN liefert den größten Wert aus der angegebenen Variablenliste.

## ODD (Mathematische Funktion)

**Funktion:** Prüfen, ob Zahl gerade oder ungerade

**Syntax:** `ODD (Zahl)`

Die Boolsche Funktion ODD liefert den Wert 0, wenn die angegebene Zahl gerade ist. Ist sie ungerade, ergibt sich für ODD der Wert -1.

## ON/OFF (Programmierhilfe)

**Funktion:** Tastenwiederholfunktion und Fehlersuche ein-/ausschalten

**Syntax:**
```
ON
OFF
```

Bei Systemstart mit SYS37100 ist der ON-Modus automatisch eingeschaltet. In fehlerhaften Zeilen blinkt der Cursor an der Stelle, wo sich der Fehler befindet.

Beim C64 ist dieser Zustand immer eingeschaltet und kann nicht durch OFF wie bei den anderen Rechnern verlassen werden.

## ON ERROR GOTO (Fehlerbehandlung)

**Funktion:** Sprungbefehl nach Programmfehler

**Syntax:** `ON ERROR GOTO Zeilennummer`

Wenn ein Programmfehler auftritt, wird zu der angegebenen Zeilennummer verzweigt, wo eine Fehlerbehandlungsroutine stehen könnte.

## ON...RESTORE (Anweisung)

**Funktion:** RESTORE-Befehl bedingt ausführen

**Syntax:**
```
ON numerischer Ausdruck RESTORE
Zeilennummernliste
```

Von dem numerischen Ausdruck hinter ON hängt ab, mit welcher Zeilennummer der RESTORE-Befehl ausgeführt wird. Dabei wird nur der ganzzahlige Anteil des errechneten Ausdrucks betrachtet. Ist er kleiner als 0, so erfolgt eine Fehlermeldung. Wenn er gleich 0 ist, wird kein RESTORE, sondern der nächste Befehl im Programmablauf ausgeführt. Für die ganzzahligen Ergebnisse 1, 2, 3, 4 usw. wird jeweils RESTORE mit der 1. Zeilennummer in der Liste, mit der 2., mit der 3, mit der 4. usw. durchgeführt, je nach Länge der Liste.

## PAUSE (Soundbefehl) nur C64

**Funktion:** Melodie anhalten

**Syntax:** `PAUSE numerischer Ausdruck`

Der ganzzahlige Anteil des numerischen Ausdrucks bestimmt die Länge der Pause. Ein Wert von 1 entspricht dabei einer Pause von 1/16 Sekunde, 2 entspricht 1/8 Sekunde usw.

## PLAY (Soundbefehl) nur C64

**Funktion:** Melodie spielen

**Syntax:** `PLAY Stimme,Tonhöhe`

Mit PLAY kann eine fertige Melodie abgespielt werden. Dabei kann die Stimme (1 bis 3) und die Tonhöhe (0 bis 65535) angegeben werden.

## POINT (Grafikbefehl)

**Funktion:** Prüfen, ob Grafikpunkt gesetzt ist

**Syntax:** `POINT Spalte,Zeile`

Ob mit SET an einer bestimmten Stelle ein Punkt gesetzt ist oder nicht, können Sie mit POINT erfahren. Diese Boolsche Funktion ist entweder 0 (Punkt ist nicht gesetzt) oder 1 (Punkt ist gesetzt).

## PRINT USING (Anweisung)

**Funktion:** Formatierte Ausgabe

**Syntax:** `PRINT USING "Format",Variablenliste`

Die Variablen der angegebenen Variablenliste werden auf dem Bildschirm formatiert ausgegeben. Dabei muß für jede Variable ein Format in der Zeichenkette vorgesehen werden. Das Format wird folgendermaßen festgelegt:

| | |
|---|---|
| # | für jede Ziffer, mehr als eine Null vor dem Komma entfällt |
| * | für jede Ziffer, mehr als eine Null vor dem Komma wird zu * |
| . | Dezimalpunkt |
| + | Vorzeichen wird ausgegeben |
| - | nur negatives Vorzeichen wird ausgegeben |
| DM | Währung wird mit ausgegeben |
| $ | Währung wird mit ausgegeben |
| % | Prozentzeichen wird mit ausgegeben |

**Beispiel:**

```
PRINT USING "DM###.##",.12
```

ergibt

```
DM 0.12
```

Falls die zu formatierende Zahl wertmäßig größer ist, als das Format vorsieht, erscheint als Ausgabe nicht die Zahl, sondern das Format selbst.

*Hinweis:* Die Formatzeichenkette kann zuvor einer Zeichenkettenvariablen zugewiesen werden. Auch die gezielte Positionierung mit PRINT @ ist hier möglich.

## PRINT@ (Grafikbefehl)

**Funktion:** Bildschirmausgabe positionieren

**Syntax:** `PRINT@ Position,"Text"`

Der Text wird an die mit Position bezeichnete Stelle auf dem Bildschirm geschrieben. Bei Rechnern mit 40 Zeichen pro Zeile darf der Positionsparameter dabei Werte von 0 bis 999 annehmen. Wenn Sie einen Rechner mit 80 Zeichen pro Zeile haben, liegen die Positionen zwischen 0 und 1999.

## REK (Anweisung) nicht C64

**Funktion:** Maximalzahl Unterprogrammebenen erhöhen

**Syntax:** `REK numerischer Ausdruck`

Wie viele Unterprogramme Sie auf Ihrem Rechner schachteln können, finden Sie ganz schnell mit der folgenden Programmzeile heraus:

```
100 Z=Z+1:PRINT Z:GOSUB 100
```

Meistens sind es 24 oder 26, und dann erfolgt die Fehlermeldung ?OUT OF MEMORY ERROR IN 100.

Mit REK wird diese Maximalzahl erhöht. Der ganzzahlige Anteil des numerischen Ausdrucks wird intern mit 64 multipliziert, und das ergibt die neue Anzahl der Unterprogrammebenen. Bedenken Sie dabei, daß eine Ebene 4 Bytes Speicherplatz belegt, also z. B.:

`REK 64` ergibt 4096 Ebenen und belegt 16 KBytes.

## **RENUM** (Programmierhilfe)

**Funktion:** Programm umnumerieren

**Syntax:**
```
RENUM
RENUM Startnummer
RENUM Startnummer,Schrittweite
```

Mit RENUM können Sie ein Programm neu durchnumerieren. Wenn Sie keine weiteren Angaben machen, wird mit der Numerierung bei 10 angefangen und in Zehnerschritten durchnumeriert. Sie können aber auch die erste Zeilennummer und die Schrittweite selbst bestimmen.

Alle Zeilennummern nach GOTO, GOSUB, THEN usw. werden entsprechend mitverändert.

## **RESET** (Grafikbefehl)

**Funktion:** Grafikpunkt löschen

**Syntax:**
```
RESET Spalte,Zeile
```

In der angegebenen Spalte (maximal 79 bzw. 159) und Zeile (maximal 49) wird ein Zeichen gelöscht.

## **RESTORE** (Anweisung)

**Funktion:** DATA-Zeiger positionieren

**Syntax:**
```
RESTORE Zeilennummer
```

Die DATA-Liste in einem Programm wird vom READ-Befehl sequentiell abgearbeitet. Die Funktion des DATA-Zeigers ist dabei, nach jedem Lesen auf das nächste Element weiterzurücken. Mit RESTORE konnte er allerdings dazu bewegt werden, wieder vorne beim allerersten DATA-Element zu beginnen.

In EXBASIC ist es mit der Angabe einer Zeilennummer hinter RESTORE nun auch möglich, den DATA-Zeiger auf eine bestimmte Zeile zu positionieren, in der mit dem Lesen fortgefahren werden soll.

## RESUME (Fehlerbehandlung)

**Funktion:** Ende der Fehlerroutine

**Syntax:**
```
RESUME
RESUME NEXT
RESUME Zeilennummer
```

Wenn mit ON ERROR GOTO ein Programmfehler abgefangen und zu einer Fehlerbehandlungsroutine verzweigt wurde, veranlaßt der RESUME-Befehl den Rücksprung zu der Zeile, in der der Fehler aufgetreten ist. Mit RESUME NEXT wird zu der Folgezeile nach der Zeile, die den Fehler verursacht hat, verzweigt, und RESUME Zeilennummer springt nach der Fehlerroutine zu der angegebenen Zeilennummer.

## RND (Mathematische Funktion)

**Funktion:** Zufallszahl erzeugen

**Syntax:**
```
RND (Maximalzahl)
RND (Maximalzahl)+Versatz
```

Anders als im Commodore-BASIC erzeugt RND ganze Zufallszahlen zwischen 1 und der Maximalzahl. Wird ein positiver oder negativer Versatz angegeben, so verschiebt sich der Zufallszahlenbereich um diesen Wert nach oben oder nach unten.

## ROUND (Mathematische Funktion)

**Funktion:** Runden

**Syntax:**
```
ROUND (Zahl,Stellenanzahl)
```

Mit ROUND wird die angegebene Zahl auf soviel Stellen gerundet, wie der zweite Parameter, Stellenanzahl, angibt.

## SEC (Befehlswort)

**Funktion:** Programmpause

**Syntax:**
```
SEC Anzahl Sekunden
```

Mit SEC kann genau angegeben werden, wieviel Sekunden das Programm warten soll. Angaben von 0 bis 255 sind möglich.

## SET (Grafikbefehl)

**Funktion:** Grafikpunkt setzen

**Syntax:** `SET Spalte,Zeile`

In der angegebenen Spalte (maximal 79 bzw. 159) und Zeile (maximal 49) wird ein kleines Quadrat gesetzt. Eingegebene Werte bis einschließlich 255 werden auf die Maximalwerte reduziert, alle anderen führen zu einer Fehlermeldung.

## SPACE (Bildschirmbefehl)

**Funktion:** Fenster anlegen und ausfüllen

**Syntax:**
```
SPACE erste Spalte,erste Zeile,
letzte Spalte,letzte Zeile,Zeichencode,Farbe
```

Auf dem Bildschirm wird durch die angegebenen Spalten (0 bis 39 bzw. 79) und Zeilen (0 bis 24) ein Fenster definiert und mit dem dem Code entsprechenden Zeichen ausgefüllt. Dieser Parameter kann auch weggelassen werden, dann wird das Fenster mit Leerzeichen gefüllt.

Beim C64 kann außerdem die Farbe in dem definierten Fenster mit einer Farbnummer von 1 bis 16 gewählt werden.

## SPACE (Programmierhilfe)

**Funktion:** Leerzeichen einfügen

**Syntax:**
```
SPACE
SPACE OFF
```

Mit SPACE können zur besseren Lesbarkeit Leerzeichen in die Programmlistings eingefügt werden. Mit SPACE OFF erhalten Sie wieder das normale Listing.

## STOP ON/STOP OFF (Programmierhilfe)

**Funktion:** STOP-Taste für Monitor-Modus aktivieren/deaktivieren

**Syntax:**
```
STOP ON
STOP OFF
```

Um ein Maschinensprache-Programm im Monitor-Modus unterbrechen zu können, müssen Sie mit STOP ON zuvor die STOP-Taste aktivieren. STOP OFF macht diesen Vorgang wieder rückgängig.

## STRING$ (Zeichenkettenbefehl)

**Funktion:** Zeichenkette erzeugen

**Syntax:**
```
STRING$ (Anzahl,"Text")
STRING$ (Anzahl,numerischer Ausdruck)
```

Mit STRING$ wird eine Zeichenkette der angegebenen Länge aufgebaut. Dabei wird entweder das erste Zeichen des in Anführungszeichen gesetzten Textes wiederholt oder das Zeichen, dessen ASCII-Code dem ganzzahligen Anteil des numerischen Ausdrucks entspricht. Die Gesamtlänge der neuen Zeichenkette darf dabei die Zahl 255 nicht überschreiten.

## SWAP (Anweisung)

**Funktion:** Variableninhalte vertauschen

**Syntax:**
```
SWAP Variable1,Variable2
```

Die Inhalte der beiden Variablen werden ohne eine Hilfsvariable direkt vertauscht.

## TRACE (Programmierhilfe)

**Funktion:** Programmablaufverfolgung

**Syntax:**
```
TRACE
TRACE OFF
```

Wenn Sie TRACE eingeschaltet haben, erscheint in der obersten Bildschirmzeile immer die Zeile, die gerade bearbeitet wird. Der aktuelle Befehl beginnt dabei mit einem inversen Zeichen.

*C64:* Ein Dauerdruck auf die CTRL-Taste bewirkt dabei, daß das Programm so langsam abgearbeitet, daß nur 2 Befehle pro Sekunde ausgeführt werden. Ein Druck auf die Commodore-Taste bringt den Programmablauf wieder auf normale Geschwindigkeit.

*Andere Rechner:* Ein Druck auf die rechte SHIFT-Taste bewirkt den langsamen Programmablauf, ein Druck auf die linke SHIFT-Taste legt die schnellere Gangart ein.

## VARPTR (Befehlswort)

**Funktion:** Variablenadresse anzeigen

**Syntax:** `VARPTR (Variablenname)`

Mit PRINT VARPTR und der Angabe einer Variablen können Sie erfahren, an welcher Adresse sich diese im Speicher befindet. Falls die Variable zu dem Zeitpunkt noch nicht existiert, wird sie durch VARPTR angelegt.

## VOLUME (Soundbefehl) nur C64

**Funktion:** Lautstärke regeln

**Syntax:** `VOLUME Lautstärke`

Mit VOLUME können Sie die Lautstärke von 0 - leise bis 15 - laut einstellen.

## VPLOT (Grafikbefehl)

**Funktion:** Senkrechte Balken zeichnen

**Syntax:**
```
VPLOT numerischer Ausdruck
VPLOT numerischer Ausdruck,Farbe   nur C64
```

Mit VPLOT lassen sich senkrechte Balken auf einem Bildschirm mit einer Auflösung von 40 x 200 Zeichen (40-Zeichen-Bildschirm) bzw. 80 x 200 Zeichen (80-Zeichen-Bildschirm) zeichnen. Startpunkt ist die momentane Cur-

sorposition, und die Länge des Balkens richtet sich nach dem Ergebnis des numerischen Ausdrucks. Dieser kann jeder mathematische Ausdruck sein, der als Ergebnis eine Zahl liefert, die hier höchstens Werte bis 199 annehmen darf. Auch Werte bis einschließlich 255 werden noch akzeptiert, aber auf 199 reduziert.

Beim C64 haben Sie zusätzlich die Möglichkeit, die Farbe des Balkens (1 bis 16) zu bestimmen.

. (Symbol)

**Funktion:** Letzte Zeile anzeigen

Syntax:

| | |
|---|---|
| `LIST .` | listet die zuletzt eingegebene Zeile |
| `LIST-.` | listet vom Anfang bis zur zuletzt eingegebenen Zeile |
| `LIST.-` | listet von der zuletzt eingegebenen Zeile bis zum Ende |
| `GOTO.` | führt das Programm ab der zuletzt eingegebenen Zeile aus |
| `RUN.` | führt das Programm ab der zuletzt eingegebenen Zeile aus |

Zusätzlich kann der Punkt in Programmen die Zeilennummernangaben hinter THEN, ELSE, GOTO und GOSUB ersetzen.

´ (Symbol)

Funktion: REM ersetzen

Anstelle des Schlüsselwortes REM kann bei EXBASIC eine Kommentarzeile auch mit einem Apostroph eingeleitet werden.

## Anhang C

# BASIC-Befehlserweiterung Honey.Aid für den C64

Honey.Aid ist eine Befehlserweiterung für den C64, die Sie zusammen mit dem Lernkurs aus dem SYBEX-Verlag "Commodore 64 BASIC-Kurs", Ref.-Nr. 3400 (mit Diskette) oder 3401 (mit Kassette) erhalten. Diese 27 zusätzlichen Befehle erleichtern vor allem die Programmeingabe und das Programmieren mit Grafik und Sound.

Dabei sind alle Programmierhilfen direkte Befehle, die nicht in Programme eingebaut werden können, alle anderen Honey.Aid-Befehle dagegen können wahlweise als direkte Befehle oder in einem Programm verwendet werden.

Wir geben nun einen Überblick über die Befehle in alphabetischer Reihenfolge:

### APPEND (Programmierhilfe)

**Abkürzung:** A<SHIFT>P

**Funktion:** Koppeln von Programmen

Dieser Befehl kann nur im Zusammenhang mit dem HMEM-Befehl (siehe dort) verwendet werden. Das im oberen Speicherbereich befindliche Programm wird mit APPEND an das im unteren Bereich befindliche angehängt. Achten Sie dabei auf geeignete Numerierung vor dem Koppeln (siehe auch NUMBER).

### AUTO (Programmierhilfe)

**Abkürzung:** <SHIFT>A oder A<SHIFT>U

**Funktion:** Automatische Zeilennumerierung

Der Befehl AUTO kann mit einer Schrittweite angegeben werden und erzeugt dann bei der Programmeingabe automatisch die Nummer der nächsten

BASIC-Zeile, und zwar mit der angegebenen Schrittweite. Wenn der Parameter weggelassen wird, wird automatisch die Schrittweite 10 vorgegeben:

| | |
|---|---|
| `AUTO 100` | liefert eine Schrittweite von 100 |
| `AUTO` | liefert eine Schrittweite von 10 |

## BRK (Programmierhilfe)

**Abkürzung:** `<SHIFT>B` oder `B<SHIFT>R`

**Funktion:** Programmunterbrechung

Wenn ein Maschinensprache-Monitor geladen wurde, veranlaßt der Befehl BRK den Einsprung aus dem BASIC da hinein.

Im BASIC wird ein Reset ausgelöst.

## CBM (Programmierhilfe)

**Abkürzung:** `<SHIFT>C` oder `C<SHIFT>B`

**Funktion:** Commodore-Modus

Wenn vor der Erstellung eines BASIC-Programms der Befehl CBM eingegeben wird, können zwar die Programmierhilfen von Honey.Aid genutzt werden, Grafik- und Soundbefehle erzeugen aber die Meldung ?syntax error in Zeilennummer. So wird verhindert, daß Programme geschrieben werden, die ohne Honey.Aid nicht laufen würden.

CBM wird durch EXTEND (siehe dort) wieder rückgängig gemacht.

## CHANGE (Programmierhilfe)

**Abkürzung:** `C<SHIFT>H`

**Funktion:** Austausch von Zeichenketten

Der Befehl CHANGE sucht im Programm alle Stellen, an denen eine bestimmte Zeichenkette auftritt, und ersetzt sie durch die als zweites angegebene Zeichenkette. Die beiden Zeichenketten können in beliebige Begrenzungszeichen eingeschlossen werden, die natürlich selbst in der Zeichenkette nicht vorkommen dürfen, z. B.:

| | |
|---|---|
| `CHANGE.A$.B$.` | Begrenzung durch Punkte |
| `CHANGEQ100Q110Q` | Begrenzung durch Q |

## COLOR oder COLOUR (Grafikbefehl)

**Abkürzung:** `C<SHIFT>O`

**Funktion:** Farbe einstellen

Dieser Befehl ändert unter Angabe der bekannten Farbparameter die Hintergrund- und Rahmenfarbe.

```
COLOR 0,1
```

zum Beispiel ändert den Bildschirmhintergrund in Schwarz (0) und den Rand in Weiß (1).

```
COLOR 15
```

ändert den Hintergrund in Grau (15) und läßt den Rand unverändert.

## DELETE (Programmierhilfe)

**Abkürzung:** `<SHIFT>D` oder `D<SHIFT>E`

**Funktion:** Zeilen löschen

Der Befehl DELETE ermöglicht es uns, eine Reihe aufeinanderfolgender Zeilen eines BASIC-Programms zu löschen:

| | |
|---|---|
| `DELETE -` | löscht das ganze Programm. |
| `DELETE 100` | löscht die Zeile 100. |
| `DELETE 100-` | löscht alle Zeilen ab 100 (einschließlich). |
| `DELETE 100-250` | löscht alle Zeilen von 100 bis 250 (einschließlich). |
| `DELETE -250` | löscht alle Zeilen bis 250 (einschließlich). |

## ENVELOPE (Soundbefehl)

**Abkürzung:** `EN<SHIFT>V`

**Funktion:** Hüllkurve

Die allgemeine Form des Befehls lautet:

```
ENVELOPE v,a,d,s,r
```

Die Parameter bedeuten im einzelnen:

| | |
|---|---|
| v | voice (Stimme) mit Werten von 1 bis 3 |
| a | attack (Einschwingen) mit Werten von 0 bis 15 |
| d | decay (Absinken) mit Werten von 0 bis 15 |
| s | sustain (Nachklingen) mit Werten von 0 bis 15 |
| r | release (Ausklingen) mit Werten von 0 bis 15 |

Wenn irgendeiner der Werte a, d, s oder r nicht verändert werden soll, dann braucht dieser Wert nicht in der Liste angegeben zu werden.

## EXTEND (Programmierhilfe)

**Abkürzung:** <SHIFT>E oder E<SHIFT>X

**Funktion:** Umschalten in Honey.Aid-Modus

Mit diesem Befehl wird das Honey.Aid reaktiviert, nachdem es mit CBM (siehe dort) abgeschaltet wurde.

## FIND (Programmierhilfe)

**Abkürzung:** <SHIFT>F oder F<SHIFT>I

**Funktion:** Suchen von Zeichenketten

Der Befehl FIND sucht die in den Begrenzungszeichen angegebene Zeichenkette im Programm. Die Suche kann auf einen bestimmten Bereich durch Angabe von Zeilennummern beschränkt werden:

```
FIND.A.          sucht und listet alle Zeilen, die A enthalten.
FIND.A.,-200     sucht nur in den Zeilen bis 200.
```

## HIRES (Grafikbefehl)

**Abkürzung:** H<SHIFT>I

**Funktion:** Grafikmodus einschalten

Der Befehl ist in jedem Programm notwendig, das Grafikbefehle verwendet, da hierbei eine hochauflösende Bildschirmdarstellung gebraucht wird. Es können 3 Parameter angegeben werden:

```
HIRES Stiftfarbe,Papierfarbe,0/1
```

Wenn Stift- und Papierfarbe weggelassen werden, bleiben die bisher gewählten Farben bestehen. Der dritte Parameter gibt an, ob der Bildschirm vorher gelöscht werden (1) oder ob das bisherige Bild bestehen bleiben soll (0).

Bei der Erläuterung zum Befehl LINE ist ein Beispiel für HIRES angegeben.

## HMEM (Programmierhilfe)

**Abkürzung:** H<SHIFT>M

**Funktion:** Speicherschutz

Der Befehl HMEM reserviert und schützt den aktuell durch ein Programm belegten Speicherplatz und erlaubt, daß der auf das Programm folgende Speicher durch andere Programme belegt wird. Bevor man Programme mit APPEND (siehe dort) aneinanderhängt, muß HMEM aufgerufen werden.

## KILL (Programmierhilfe)

**Abkürzung:** <SHIFT>K oder K<SHIFT>I

**Funktion:** Honey.Aid ausschalten

Mit KILL kann Honey.Aid verlassen werden. Es ist jedoch durch Eingabe von SYS750 möglich, es zu reaktivieren.

## LINE (Grafikbefehl)

**Abkürzung:** LI<SHIFT>N

Mit LINE kann eine Linie zwischen zwei angegebenen Koordinaten gezogen werden:

```
LINE X1,Y1,X2,Y2,N
```

Der Parameter N legt dabei die Farbe fest:

0 Linie wird in der Papierfarbe gezeichnet (gelöscht).
1 Linie wird in der Tintenfarbe gezeichnet (bei HIRES angegeben).
2 Linie wird invers gezeichnet, wenn ein bereits gezeichneter Punkt gekreuzt wird, sonst in der Tintenfarbe.

**Beispiel**

```
10 HIRES 12,15,1
20 LINE 10,50,100,150,1
```

Die Werte für X müssen im Bereich von 0 bis 319 und für Y von 0 bis 199 liegen.

## LMEM (Programmierhilfe)

**Abkürzung:** <SHIFT>L oder L<SHIFT>M

**Funktion:** Speicherbereich freigeben

Mit diesem Befehl wird die Sperre, die durch den HMEM-Befehl gesetzt wurde, wieder aufgehoben und der gesamte Speicherbereich für BASIC-Programme freigegeben. Nach APPEND (siehe dort), ist LMEM überflüssig, da auf diesen Befehl automatisch die Rückkehr zum normalen Speicher erfolgt.

## NRM (Grafikbefehl)

**Abkürzung:** N<SHIFT>R

**Funktion:** Textmodus einschalten

Der NRM-Befehl schaltet nach einem HIRES-Befehl wieder zurück in den Textmodus. In der Praxis wird dieser Befehl selten gebraucht, da in Honey. Aid die Grafik so lange gezeigt wird, bis irgendein Tastendruck erfolgt.

NRM wird nur vor einem STOP-Befehl im Programm benötigt. Falls aus Versehen ein STOP-Befehl ohne ein NRM benutzt wurde, kann der Befehl nachträglich als direkter Befehl eingegeben werden:

```
10 HIRES 12,15,1
20 LINE 10,50,100,150,1
30 STOP
```

## NUMBER (Programmierhilfe)

**Abkürzung:** <SHIFT>N oder N<SHIFT>U

**Funktion:** Umnumerieren

Der Befehl NUMBER entspricht in anderen BASIC-Versionen dem Befehl RENUMBER. Das Umnumerieren eines Programms unter Angabe der gewünschten Zeilennummern ist damit möglich:

```
NUMBER Anfangszeilennummer,Schrittweite
```

Wenn die Anfangszeilennummer weggelassen wird, beginnt die Numerierung mit 100, und wenn die Schrittweite weggelassen wird, beträgt sie automatisch 10.

Alle Zeilennummern nach den Befehlen GOTO, GOSUB, THEN, RUN oder LIST werden ebenfalls geändert. Falls bei der Programmerstellung fehlerhafte Aufrufe von Zeilennummern, die im Programm gar nicht vorkommen, erzeugt wurden, können sie nun mit FIND.65535. gesucht werden, da Honey. Aid diese Zeilennummern durch 65535 ersetzt.

## OLD (Programmierhilfe)

**Abkürzung:** <SHIFT>O oder O<SHIFT>L

**Funktion:** Umkehrfunktion zu NEW

Da nach Eingabe des Befehls NEW (siehe Kapitel 9) das Programm nicht völlig verschwnden ist, kann es mit Hilfe des Befehls OLD wieder zurückgeholt werden. Nach DELETE hilft er allerdings nicht mehr.

## PLAY (Soundbefehl)

**Abkürzung:** PL<SHIFT>A

**Funktion:** Wiedergabe einer Melodie

Dies ist ein sehr nützlicher Befehl. Er dient dazu, eine Melodie zu spielen. Im Extremfall könnte mit dem PLAY-Befehl ein Musikstück von 6630 Tönen gespielt werden.

In seiner einfachsten Form nimmt der PLAY-Befehl eine einzige Zeichenkette als Parameter, z. B. PLAY A$ oder PLAY "CDEFG". Die Zeichenkette definiert die aktuellen Noten, die gespielt werden sollen.

Zur Darstellung eines einzelnen Tons werden maximal 4 Symbole benötigt:

- Ein Symbol legt die Stimme fest, die die Note wiedergeben soll: <BLK> oder <CTRL>1 legen Stimme 1 fest, <WHT> oder <CTRL>2 die Stimme 2 und <RED> oder <CTRL>3 die Stimme 3.

- Ein zweites Symbol dient dazu, die Tonhöhe festzulegen: C D E F G A H geben die natürlichen Töne an, SHIFT mit C D F G A die Halbtöne. Ein R oder ein Bindestrich geben die Pause an.

- Ein drittes Symbol dient zur Festlegung der Oktave. Die hierfür gebräuchliche Bezeichnung sind die Ziffern 0 bis 7.

- Als vierte und letzte Angabe wird schließlich noch die Dauer des Tons benötigt. Zu diesem Zweck werden die Funktionstasten eingespannt: <f1> legt eine ganze Note fest, <f3> eine halbe Note, <f5> eine Viertelnote und <f7> eine Achtelnote. Die Funktionstasten mit der SHIFT-Taste dienen für die kürzeren Noten: <f2> für Sechzehntel, <f4> für Zweiunddreißigstel, <f6> für Vierundsechzigstel und <f8> für Einhundertachtundzwanzigstel.

**Beispiel**

```
PLAY "<BLK>C4<F5><WHT><RED>3"
```

## PLOT (Grafikbefehl)

**Abkürzung:** P<SHIFT>L

**Funktion:** Bildpunkte setzen

Der PLOT-Befehl setzt einen Punkt an der angegebenen Koordinate. Dabei kann mit dem dritten Parameter die gewünschte Farbe bestimmt werden:

```
PLOT X-Koordinate,Y-Koordinate,Farbe
```

Für Farbe kann 0 (Hintergrundfarbe oder löschen), 1 (Zeichenfarbe) oder 2 (inverse Darstellung) gewählt werden. Bei der inversen Darstellung wird ein Punkt, falls er auf ein bereits besetztes Feld trifft, in der Farbe des Hintergrundes gezeichnet, sonst in der Zeichenfarbe.

Der Wert für X muß im Bereich von 0 bis 319 und für Y von 0 bis 199 liegen.

## PULSE (Soundbefehl)

**Abkürzung:** P<SHIFT>U

**Funktion:** Festlegen der Impulsbreite

Wenn für eine bestimmte Stimme die Rechteckschwingung (siehe WAVE) eingestellt ist, kann der Befehl PULSE die Impulsbreite dieses Tons einstellen:

```
PULSE Stimme,Impulsbreite
```

Für eine gewählte Stimme kann die Impulsbreite von 0 bis 4095 variieren.

## REPEAT (Programmierhilfe)

**Abkürzung:** <SHIFT>R oder R<SHIFT>E

**Funktion:** Tastenwiederholung

Der Befehl REPEAT kann alle Tasten, die außer der Leertaste, Cursortasten und INST/DEL-Tasten normalerweise keine Dauerfunktion haben, umstellen:

| | |
|---|---|
| `REPEAT 0` | keine Taste hat Dauerfunktion |
| `REPEAT 1` | alle Tasten haben Dauerfunktion |
| `REPEAT` | Grundeinstellung |

## RESET (Programmierhilfe)

**Abkürzung:** RE<SHIFT>S

**Funktion:** Kaltstart

Dieser Befehl veranlaßt den C64, wieder in den Zustand, in dem er nach dem Einschalten ist, zu gehen. Alles, was sichtbar ist, wird initialisiert, einschließlich des RAM-Speichers. Vorsichtshalber verlangt Honey.Aid deshalb vorher eine Bestätigung:

```
System reset-ist das okay j/n?
```

## SOUND (Soundbefehl)

**Abkürzung:** S<SHIFT>O

**Funktion:** Klang festlegen

Dieser Befehl kann benutzt werden, um eine einzelne Note zu spielen:

```
SOUND Stimme,Oktave,Note,Dauer
```

Die Parameter können folgende Werte annehmen:

| | |
|---|---|
| Stimme | Werte von 1 bis 3 |
| Oktave | Werte von 0 bis 7 |
| Note | Werte von 0 bis 12 |
| Dauer | Werte von 1 bis 8 |

Bei der Note erzeugt die Null eine Pause, die anderen Werte, 1 bis 12, entsprechen den Tönen C, C#, D, D#, E, F, F#, G, G#, A, A# und H.

Bei der Dauer stehen die Werte von 8 bis 1 für Ganze Note, Halbe Note, Viertelnote usw.

Der Befehl SOUND ohne Parameter wiederholt den Ton, der zuvor definiert wurde:

```
SOUND 1,4,1,6
SOUND
```

Die Parameter von SOUND können auch wahlweise Variablen oder Ausdrücke sein.

## TEMPO (Soundbefehl)

**Abkürzung:** TE<SHIFT>M

Der Befehl TEMPO legt für alle Variablen fest, wie viele Viertelnoten pro Minute gespielt werden können:

```
TEMPO Zahl
```

Dabei ist Zahl irgendein positiver Wert.

```
TEMPO
```

ohne Parameter schaltet auf das Standardtempo von 120 Viertelnoten pro Minute. Das Tempo 1 liefert pro Minute eine Viertelnote.

## VOL (Soundbefehl)

**Abkürzung:** V<SHIFT>O

**Funktion:** Einstellen der Lautstärke

Mit diesem Befehl wird für alle Stimmen die Lautstärke mit Werten von 0 bis 15 festgelegt:

```
VOL Lautstärke
```

VOL ohne eine Angabe bedeutet volle Lautstärke (15).

## WAVE (Soundbefehl)

**Abkürzung:** <SHIFT>W oder W<SHIFT>A

**Funktion:** Festlegen der Stimme und der Wellenform

Mit WAVE wird die Stimme und die Wellenform festgelegt:

```
WAVE Stimme,Wellenform
```

Die Stimme kann Werte von 1 bis 3 annehmen und der Parameter für die Wellenform Werte von 0 bis 8.

## Anhang D

# BASIC-Befehlserweiterung SUPER EXPANDER 64

Eine der Befehlserweiterungen, die den umfangreichen Befehlsvorrat von BASIC 3.5 fast vollständig mit gleichlautenden Befehlswörtern bietet, ist SUPER EXPANDER 64 von Commodore.

Dieses Steckmodul enthält gegenüber dem BASIC 2.0 auf dem C64 21 neue Befehle und 11 neue Funktionen.

Hier ist nun die alphabetische Übersicht mit Erläuterungen:

### BOX (Grafikbefehl)

**Funktion:** Ausgefülltes Rechteck zeichnen

**Syntax:** `BOX Farbquelle,X1,Y1,X2,Y2,Winkel,0/1`

Ein Rechteck wird mit der angegebenen Farbe gezeichnet und, falls der letzte Parameter 1 ist, auch damit ausgefüllt. Dabei markiert X1,Y1 die linke obere Ecke und X2,Y2 die rechte untere.

Die Farbquelle kann Werte von 0 bis 3 annehmen (Hintergrundfarbe, Vordergrundfarbe, Mehrfarben). Wenn ein Winkel angegeben ist, gibt er die Gradzahl der Drehung im Uhrzeigersinn an. Denken Sie daran, vor dem BOX-Befehl mit GRAPHIC den Grafikmodus einzuschalten.

### CHAR (Grafikbefehl)

**Funktion:** Text im Grafikmodus ausgeben

**Syntax:** `CHAR Farbquelle,Spalte,Zeile,"Text",0/1`

In der angegebenen Spalte (max. 39) und Zeile (max. 24) wird der Text auf den hochauflösenden Schirm geschrieben. Wenn der letzte Parameter 1 ist, wird der Text revers ausgegeben.

## CIRCLE (Grafikbefehl)

**Funktion:** Ellipse zeichnen

**Syntax:** `CIRCLE Farbquelle,X1,Y1,RADX,RADY,Start, Ende,Winkel,Z`

Mit CIRCLE wird eine Ellipse (Sonderform Kreis für RADX=RADY) gezeichnet. Außer dem Radius in X-Richtung, RADX, sind alle Angaben wahlweise und können weggelassen werden.

| | |
|---|---|
| Farbquelle | 0 - Hintergrundfarbe<br>1 - Vordergrundfarbe (Standard)<br>2 - Mehrfarbenmodus 1<br>3 - Mehrfarbenmodus 2 |
| X1,Y1 | Kreismittelpunkt (Standard Cursorposition) |
| RADX | Radius in X-Richtung |
| RADY | Radius in Y-Richtung (Standard RADX) |
| Start | Startpunkt für Kreisbogen (Standard 0 Grad) |
| Ende | Endpunkt für Kreisbogen (Standard 360 Grad) |
| Winkel | Drehung im Uhrzeigersinn in Grad (Standard 0 Grad) |
| Z | Erhöhung in Grad (Standard 2) |

## COLINT (Spritebefehl)

**Funktion:** Verzweigung nach Sprite-Kollision

**Syntax:** `COLINT Modus,Zeilennummer`

Wenn eine Sprite-Kollision stattgefunden hat, verzweigt das Programm zu der in der Zeilennummer angegebenen Unterroutine. Dabei hat der Parameter Modus folgende Funktion:

| Modus | Sprite/Sprite-Kollision |
|---|---|
| 1 | Sprite/Bitmuster-Kollision |
| 2 | Lichtstift-Berührung |

## COLOR (Grafikbefehl)

**Funktion:** Farbgebung

**Syntax:** `COLOR Hintergrund,Vordergrund,Mehrfarben1, Mehrfarben2,Rahmen`

Die fünf angegebenen Parameter können jeweils die Werte von 0 bis 15 annehmen, um die entsprechenden Farben festzulegen.

## DRAW (Grafikbefehl)

**Funktion:** Figur zeichnen

**Syntax:** `DRAW Farbquelle,X1,Y1 TO X2,Y2 TO ...`

DRAW zeichnet eine beliebige Figur in der angegebenen Zeichenfarbe, ausgehend vom Startpunkt X1,Y1 (Standard Cursorposition) zum Punkt X2,Y2 usw.

## FILTER (Soundbefehl)

**Funktion:** Filter setzen für Sound

**Syntax:** `FILTER Frequenz,Tief,Band,Hoch,Resonanz`

Mit FILTER können Sie die Klangfilter für die Töne setzen. Die Parameter bedeuten im einzelnen:

| | |
|---|---|
| Frequenz | Frequenzgrenze (0 bis 2048), ab der der Filter wirksam ist |
| Tief | Tiefpaßfilter (0 aus/1 an) |
| Band | Bandpaßfilter (0 aus/1 an) |
| Hoch | Hochpaßfilter (0 aus/1 an) |
| Resonanz | Werte von 0 bis 15 |

## GRAPHIC (Grafikbefehl)

**Funktion:** Grafikmodus einstellen

**Syntax:** `GRAPHIC Modus,Löschen`

Mit GRAPHIC wird der Grafikmodus eingestellt. Außerdem kann gleichzeitig mit dem zweiten Parameter festgelegt werden, ob der Bildschirm gelöscht werden soll. Ist er 1, wird gelöscht. Wenn er 0 ist oder weggelassen wird, wird nicht gelöscht.

Die Grafikmodi sind im einzelnen:

0 Text
1 Mehrfarben-Grafik
2 hochauflösende Grafik
3 Grafik/Text gemischt. Die unteren 5 Zeilen sind für den Text.

## GSHAPE (Grafikbefehl)

**Funktion:** Grafik auf Bildschirm plazieren

**Syntax:** `GSHAPE Zeichenkettenvariable,X1,Y1,Modus`

Die in der Zeichenkettenvariablen mit SSHAPE abgelegten Grafik kann an der Position X1,Y1 (linke obere Ecke) auf dem Bildschirm angezeigt werden. Standard ist Cursorposition. Der Modus kann dabei variieren:

**Modus** 0 - Grafik normal darstellen
1 - Grafik invertiert darstellen
2 - Grafik mit Schirm über OR verknüpft darstellen
3 - Grafik mit Schirm über AND verknüpft darstellen
4 - Grafik mit Schirm über XOR verknüpft darstellen

## KEY (Programmierhilfe)

**Funktion:** Funktionstasten belegen

**Syntax:** `KEY Tastennummer,Zeichenkettenausdruck`

Die Funktionstasten können mit geeigneten Befehlen und Befehlssequenzen belegt werden, die dann auf Tastendruck bequem ausgeführt werden. Der Befehl KEY ohne Parameter zeigt die aktuelle Belegung an.

## LOCATE (Grafikbefehl)

**Funktion:** Cursor setzen

**Syntax:** `LOCATE X,Y`

Der Cursor wird an die mit X (Spalte) und Y (Zeile) angegebene Position gebracht.

## MOVSPR (Spritebefehl)

**Funktion:** Sprite setzen

**Syntax:** `MOVSPR Spritenummer,X,Y`

Das Sprite mit der angegebenen Nummer wird an die mit X (Spalte) und Y (Zeile) definierte Position gesetzt.

## PAINT (Grafikbefehl)

**Funktion:** Fläche farbig ausfüllen

**Syntax:** `PAINT Farbquelle,X,Y,Modus`

Mit PAINT lassen sich umschlossene Flächen in der angegebenen Farbe ausfüllen. Es wird immer die Fläche gefärbt, in die sich die Koordinaten X und Y befinden. Modus kann 0 oder 1 gesetzt werden und bewirkt dann die Färbung wie das gewählte Farbregister (0) oder jede andere Vordergrundfarbe (1).

## RBUMP (Spritebefehl)

**Funktion:** Sprite-Kollision abfragen

**Syntax:** `RBUMP (Modus)`

RBUMP registriert eine Sprite-Kollision mit einem anderen Sprite (Modus=0) oder mit einem Hintergrundzeichen (Modus=1).

## RCLR (Grafikbefehl)

**Funktion:** Farbwerte ausgeben

**Syntax:** `RCLR Farbquelle`

Mit RCLR kann man die mit COLOR gesetzten Farbwerte abfragen, und zwar kann die Farbquelle sein:

**Farbquelle**
0 - Hintergrundfarbe
1 - Vordergrundfarbe
2 - Mehrfarbenmodus1
3 - Mehrfarbenmodus2
4 - Rahmenfarbe

## RDOT (Grafikbefehl)

**Funktion:** Farbwert und Koordinaten des Grafikcursors angeben

**Syntax:** `RDOT (Modus)`

Die Funktion RDOT liefert den Farbwert und die Koordinaten des Grafikcursors.

**Modus**
0 - X-Koordinate des Grafikcursors
1 - Y-Koordinate des Grafikcursors
2 - RDOT(2)=0 bedeutet Hintergrundfarbe
RDOT(2)=1 bedeutet Vordergrundfarbe
RDOT(2)=2 bedeutet Mehrfarbenmodus1
RDOT(2)=3 bedeutet Mehrfarbenmodus2

## RGR (Grafikbefehl)

**Funktion:** Eingestellten Grafikmodus angeben

**Syntax:** `RGR(0)`

Der Grafikmodus, der mit GRAPHIC eingestellt wurde, kann mit RGR(0) abgefragt werden. Dabei können wie beim Einstellen Werte von 0 bis 3 auftreten.

## RJOY (Eingabebefehl)

**Funktion:** Joystick-Position ermitteln

**Syntax:** `RJOY (Joysticknummer)`

Je nach eingegebener Joysticknummer (1 oder 2) wird der Positionswert des Joysticks von 0 bis 8 ermittelt. Falls der Feuerknopf gedrückt ist, erhöht sich dieser Wert um 128.

## RPEN (Eingabebefehl)

**Funktion:** Koordinaten des Lichtstifts ermitteln

**Syntax:** `RPEN (Koordinate)`

Wenn man die Koordinaten des Lichtstifts wissen will, kann man diese mit RPEN (0) für die X-Koordinate und mit RPEN(1) für die Y-Koordinate erfahren.

## RPOT (Eingabebefehl)

**Funktion:** Paddle-Position ermitteln

**Syntax:** `RPOT (Paddle)`

Ein Wert von 0 bis 255 gibt an, in welcher Position sich die Drehregler befinden. Wenn der Feuerknopf gedrückt ist, wird zu dem Wert 256 addiert. Es können 4 Drehregler abgefragt werden:

RPOT(0) - Position von Regler 1
RPOT(1) - Position von Regler 2
RPOT(2) - Position von Regler 3
RPOT(3) - Position von Regler 4

## RSPCOL (Spritebefehl)

**Funktion:** Spritefarbe ermitteln

**Syntax:** `RSPCOL (Modus)`

Die Farbe eines Sprites im Mehrfarbenmodus wird ermittelt. Dabei kann sich ein Farbwert von 0 bis 15 ergeben.

RSPCOL(0) - Mehrfarbenmodus1
RSPCOL(1) - Mehrfarbenmodus2

## RSPPOS (Spritebefehl)

**Funktion:** Position und Geschwindigkeit eines Sprites ermitteln

**Syntax:** `RSPPOS (Spritenummer,Modus)`

Mit dem Befehl RSPPOS läßt sich die aktuelle Position und die Geschwindigkeit des Sprites mit der angegebenen Nummer (0 bis 7) feststellen. Dabei gibt das zweite Argument, der Modus, folgendes an:

**Modus** 0 - X-Koordinate angeben
1 - Y-Koordinate angeben
2 - Geschwindigkeit angeben

## RSPR (Spritebefehl)

**Funktion:** Sprite-Parameter ermitteln

**Syntax:** `RSPR (Spritenummer,Modus)`

Die für ein Sprite mit der angegebenen Nummer gesetzten Werte können über die Funktion RSPR abgefragt werden. Dabei gibt der Modus an, für welche Eigenschaft des Sprites man sich interessiert:

**Modus** 0 Sprite aus (0) oder an (1)
1 Vordergrundfarbe (0 bis 15)
2 Priorität (0 - hoch, 1 - tief)
3 Dehnung in X-Richtung (0 - nein, 1 - ja)
Dehnung in Y-Richtung (0 - nein, 1 - ja)
5 Spritemodus (0 - hochauflösend, 1 - Mehrfarben)

## SCALE (Grafikbefehl)

**Funktion:** Skalierung einstellen

**Syntax:** `SCALE 0/1`

Im Standard-Koordinatensystem mit SCALE 0 hängt die Skalierung des Koordinatensystems vom eingestellten Grafikmodus ab. Wenn Sie SCALE 1 eingeben, reichen die Grenzen von 0 bis 1023 in jedem beliebigen Grafikmodus.

## SCNCLR (Bildschirm-Anweisung)

**Funktion:** Bildschirm löschen

**Syntax:** `SCNCLR`

Der Bildschirm wird in jedem beliebigen Modus gelöscht.

## SPRCOL (Spritebefehl)

**Funktion:** Farben im Mehrfarbenmodus setzen

**Syntax:** `SPRCOL Farbe1,Farbe2`

Die beiden zusätzlichen Farben für den Mehrfarbenmodus können mit SPRCOL und zwei Farbwerten von jeweils 0 bis 15 eingestellt werden.

## SPRDEF (Spritebefehl)

**Funktion:** Spritedefinition

**Syntax:** `SPRDEF`

Der Anwender kann über die Steuerung der folgenden Tasten ein Sprite erstellen:

| | |
|---|---|
| 0 bis 7 | Spritenummer vergeben |
| 1 bis 4 | Farbquelle vergeben |
| CTRL 1 bis 8 | Zeichenfarbe 0 bis 7 einstellen |
| C= 1 bis 8 | Zeichenfarbe 8 bis 15 einstellen |
| 2 bis 3 | Punkt setzen |
| 0 | Punkt löschen |
| RETURN | neue Zeile |
| Cursor | hoch, runter, rechts, links |
| HOME | Ausgangsposition |
| CLR | löschen |
| STOP | Änderungen ignorieren |
| RETURN+RETURN | Spritegenerator verlassen |
| SHIFT RETURN | Sprite speichern und Exit |
| X | Sprite in X-Richtung dehnen |
| Y | Sprite in Y-Richtung dehnen |
| M | Sprite im Mehrfarbenmodus zeigen |

## SPRITE (Spritebefehl)

**Funktion:** Sprite aktivieren

**Syntax:**
```
SPRITE Spritenummer,Sprite ein/aus,Farbe,
Priorität,X-Vergrößerung,Y-Vergrößerung,
Multicolor ein/aus
```

Die Anweisung SPRITE ruft ein vorher definiertes Sprite auf. Die Spritenummer (0 bis 7) muß dabei unbedingt angegeben werden. Die anderen Parameter können wahlweise weggelassen werden. Sie haben folgende Bedeutung:

| | |
|---|---|
| Spritenummer | Nummer des Sprites (0 bis 7) |
| Sprite ein/aus | 0 - Sprite wird gelöscht |
| | 1 - Sprite erscheint auf dem Schirm |
| Farbe | Zeichenfarbe (0 bis 15) |
| Priorität | 0 - Sprite liegt im Vordergrund |
| | 1 - Sprite liegt im Hintergrund |
| X-Vergrößerung | 0 - normal |
| | 1 - Sprite wird in X-Richtung verdoppelt |
| Y-Vergrößerung | 0 - normal |
| | 1 - Sprite wird in Y-Richtung verdoppelt |
| Multicolor ein/aus | 0 - Mehrfarbenmodus ist ausgeschaltet |
| | 1 - Mehrfarbenmodus ist eingeschaltet |

## SPRSAV (Spritebefehl)

**Funktion:** Grafikinformation speichern

**Syntax:**
```
SPRSAV Spritenummer,Zeichenkettenvariable
SPRSAV Zeichenkettenvariable,Spritenummer
```

Wenn wir mit Hilfe von SSHAPE Grafikinformationen in einer Zeichenkettenvariablen abgelegt haben, können wir diese mit SPRSAV als Sprite speichern. Umgekehrt kann man ein Sprite in einer Zeichenkettenvariablen ablegen.

## SSHAPE (Grafikbefehl)

**Funktion:** Grafikinformationen als Zeichenkette abspeichern

**Syntax:**
```
SSHAPE Zeichenkettenvariable,X1,Y1,X2,Y2
```

Eine Grafik, die innerhalb des Rechtecks mit den diagonal gegenüberliegenden Punkten X1,Y1 und X2,Y2 (Standard Cursorposition) liegt, kann mit der Anweisung SSHAPE einer Zeichenkettenvariablen zugeordnet werden.

## TEMPO (Soundbefehl)

**Funktion:** Spieldauer der Noten festlegen

**Syntax:** `TEMPO Dauer`

Jede Note wird in der angegebenen Dauer abgespielt. Das Tempo reicht von 0 - langsam - bis 255 - sehr schnell.

## TUNE (Soundbefehl)

**Funktion:** Hüllkurve für die Töne festlegen

**Syntax:**
```
TUNE Hüllkurvennummer,Anschwellzeit,
Abschwellzeit,Dauer,Ausklingzeit,
Wellenform,Impulsbreite
```

Mit TUNE werden bei der Tonerzeugung die Hüllkurven definiert. Die Parameter variieren den Ton in der folgenden Bandbreite:

| | |
|---|---|
| Hüllkurvennummer | Werte von 0 bis 9 |
| Anschwellzeit | Werte von 0 bis 15 |
| Abschwellzeit | Werte von 0 bis 15 |
| Dauer | Werte von 0 bis 15 |
| Ausklingzeit | Werte von 0 bis 15 |
| Wellenform | 0 - Dreieckschwingung |
| | 1 - Sägezahnschwingung |
| | 2 - Rechteckschwingung |
| | 3 - Rauschen |
| | 4 - Ringmodulation |
| Impulsbreite | Werte von 0 bis 4095 |
| | (nur für Wellenform 2 - Rechteck) |

## Anhang E

# Alphabetische Befehlsübersicht
## mit Crossreference aller BASIC-Versionen

| | BASIC 2.0 | BASIC 4.0 | BASIC 3.5 | BASIC 7.0 |
|---|---|---|---|---|
| ABS | X | X | X | X |
| APPEND | - | X | - | X |
| ASC | X | X | X | X |
| ATN | X | X | X | X |
| AUTO | - | - | X | X |
| BACKUP | - | X | X | X |
| BANK | - | - | - | X |
| BEGIN | - | - | - | X |
| BEND | - | - | - | X |
| BLOAD | - | - | - | X |
| BOOT | - | - | - | X |
| BOX | - | - | X | X |
| BSAVE | - | - | - | X |
| BUMP | - | - | - | X |
| CATALOG | - | X | - | X |
| CHAR | - | - | X | X |
| CHR$ | X | X | X | X |
| CIRCLE | - | - | - | X |
| CLOSE | X | - | X | X |
| CLR | X | X | X | X |
| CMD | X | X | X | X |
| COLLECT | - | X | X | X |
| COLLISION | - | - | - | X |
| COLOR | - | - | X | X |
| CONCAT | - | X | - | X |
| CONT | X | X | X | X |
| COPY | X* | X | X | X |
| COS | X | X | X | X |
| DATA | X | X | X | X |
| DCLEAR | - | - | - | X |
| DCLOSE | - | X | - | X |
| DEC | - | - | X | X |
| DEF FN | X | X | X | X |
| DELETE | X* | X* | X | X |

| | BASIC 2.0 | BASIC 4.0 | BASIC 3.5 | BASIC 7.0 |
|---|---|---|---|---|
| DIM | X | X | X | X |
| DIRECTORY | - | X | X | X |
| DLOAD | - | X | X | X |
| DO LOOP EXIT | - | - | X | X |
| DO LOOP UNTIL | - | - | X | X |
| DO LOOP WHILE | - | - | X | X |
| DOPEN | - | X | - | X |
| DRAW | - | - | X | X |
| DS | - | - | X | X |
| DS$ | - | - | X | X |
| DSAVE | - | X | X | X |
| DVERIFY | - | - | - | X |
| END | X | X | X | X |
| ENVELOPE | - | - | - | X |
| ER EL ERR$ | - | - | X | X |
| EXP | X | X | X | X |
| FAST | - | - | - | X |
| FETCH | - | - | - | X |
| FILTER | - | - | - | X |
| FOR NEXT | X | X | X | X |
| FRE | X | X | X | X |
| GET | X | X | X | X |
| GET# | X | X | X | X |
| GETKEY | - | - | X | X |
| GO 64 | - | - | - | X |
| GOSUB RETURN | X | X | X | X |
| GOTO | X | X | X | X |
| GRAPHIC | - | - | X | X |
| HEADER | - | X | X | X |
| HELP | - | - | X | X |
| HEX$ | - | - | X | X |
| IF THEN | X | X | X | X |
| IF THEN ELSE | - | - | X | X |
| INPUT | X | X | X | X |
| INPUT# | X | X | X | X |
| INSTR | - | - | X | X |
| INT | X | X | X | X |
| JOY | - | - | X | X |
| KEY | - | - | X | X |
| LEFT$ | X | X | X | X |
| LEN | X | X | X | X |
| LET | X | X | X | X |

| | BASIC 2.0 | BASIC 4.0 | BASIC 3.5 | BASIC 7.0 |
|---|---|---|---|---|
| LIST | X | X | X | X |
| LOAD | X | X | X | X |
| LOCATE | - | - | X | X |
| LOG | X | X | X | X |
| MID$ | X | X | X | X |
| MONITOR | - | - | X | X |
| MOVSPR | - | - | - | X |
| NEW | X | X | X | X |
| ON GOSUB | X | X | X | X |
| ON GOTO | X | X | X | X |
| OPEN | X | X | X | X |
| PAINT | - | - | X | X |
| PEEK | X | X | X | X |
| PEN | - | - | - | X |
| PLAY | - | - | - | X |
| POINTER | - | - | - | X |
| POKE | X | X | X | X |
| POS | X | X | X | X |
| POT | - | - | - | X |
| PRINT | X | X | X | X |
| PRINT# | X | X | X | X |
| PRINT USING | - | - | X | X |
| PUDEF | - | - | X | X |
| RCLR | - | - | X | X |
| RDOT | - | - | X | X |
| READ | X | X | X | X |
| RECORD# | - | X | - | X |
| REM | X | X | X | X |
| RENAME | X* | X | X | X |
| RENUMBER | - | - | X | X |
| RESTORE | X | X | X | X |
| RESUME | - | - | X | X |
| RGR | - | - | X | X |
| RIGHT$ | X | X | X | X |
| RLUM | - | - | X | X |
| RND | X | X | X | X |
| RREG | - | - | - | X |
| RSPCOLOR | - | - | - | X |
| RSPPOS | - | - | - | X |
| RSPRITE | - | - | - | X |
| RUN | X | X | X | X |
| RWINDOW | - | - | - | X |

| | BASIC 2.0 | BASIC 4.0 | BASIC 3.5 | BASIC 7.0 |
|---|---|---|---|---|
| SAVE | X | X | X | X |
| SCALE | - | - | X | X |
| SCNCLR | - | - | X | X |
| SCRATCH | X* | X | X | X |
| SGN | X | X | X | X |
| SIN | X | X | X | X |
| SLEEP | - | - | - | X |
| SLOW | - | - | - | X |
| SOUND | - | - | X | X |
| SPC | X | X | X | X |
| SPRCOLOR | - | - | - | X |
| SPRDEF | - | - | - | X |
| SPRITE | - | - | - | X |
| SPRSAV | - | - | - | X |
| SQR | X | X | X | X |
| SSHAPE/GSHAPE | - | - | X | X |
| ST | X | X | X | X |
| STASH | - | - | - | X |
| STOP | X | X | X | X |
| STR$ | X | X | X | X |
| SWAP | - | - | - | X |
| SYS | X | X | X | X |
| TAB | X | X | X | X |
| TAN | X | X | X | X |
| TEMPO | - | - | - | X |
| TI | X | X | X | X |
| TI$ | X | X | X | X |
| TRAP | - | - | X | X |
| TRON/TROFF | - | - | X | X |
| USR | X | X | X | X |
| VAL | X | X | X | X |
| VERIFY | X | X | X | X |
| VOL | - | - | X | X |
| WAIT | X | X | X | X |
| WIDTH | - | - | - | X |
| WINDOW | - | - | - | X |
| XOR | - | - | - | X |

Dabei bedeutet das Zeichen * folgendes: Alle mit * gekennzeichneten Befehle können nicht direkt verwendet werden, sondern müssen über den Befehlskanal

15 gesendet werden, der zu diesem Zweck geöffnet werden muß. Beispielsweise wird der Befehl SCRATCH in BASIC 2.0 so angewendet:

```
OPEN 1,8,15
PRINT#1,"SCRATCH:DATEI1"
CLOSE 1
```

# Anhang F

# Alphabetische Befehlsübersicht
## nach Funktionen

Die mit * gekennzeichneten Schlüsselwörter sind Funktionen.

### Trigonometrische Funktionen

```
*ATN
*COS
*SIN
*TAN
```

### Andere numerische Funktionen

```
*ABS
*ASC
*DEC
*EXP
*INT
*LEN
*LOG
*RND
*SGN
*SQR
*VAL
```

### Zeichenkettenfunktionen

```
*CHR$
*HEX$
*INSTR
*LEFT$
*MID$
*RIGHT$
*STR$
```

### Programmierhilfen

```
AUTO
CONT
```

```
DELETE
FAST
HELP
KEY
LIST
RENUMBER
SLOW
STOP
TROFF
TRON
```

### Befehle zur Fehlerbehandlung

```
RESUME
TRAP
```

### Systemvariablen

```
DS
DS$
EL
ER
ERR$
ST
TI
TI$
```

### Systembefehle

```
*FRE
GO 64
MONITOR
NEW
*PEEK
*POINTER
POKE
*RGR
RREG
```

### System-Befehle für Dienstprogramme des Diskettenlaufwerks

```
APPEND
BACKUP
CATALOG
COLLECT
```

CONCAT
COPY
DIRECTORY
HEADER
RENAME

### Befehle und Systembefehle für Ein- und Ausgabe

BLOAD
BOOT
BSAVE
CLOSE
CMD
DCLEAR
DCLOSE
DLOAD (auch für Ablaufsteuerung (Overlay) zu verwenden)
DOPEN
DSAVE
DVERIFY
GET
GET#
GETKEY
INPUT
INPUT#
*JOY
LOAD
OPEN
*PEN
*POT
PRINT
PRINT#
PRINT USING
PUDEF
READ
RECORD#
SAVE
SCRATCH
VERIFY
WAIT

### Speicherverwaltungsbefehle

*BANK
FETCH
STASH
SWAP

### Befehle zur Bildschirmsteuerung und Ausgabeformatierung

```
CLR
*POS
*RWINDOW
SCNCLR
*SPC
*TAB
WINDOW
```

### Befehle zur Programmstruktur und Ablaufsteuerung

```
BEGIN
BEND
DEF FN
DO LOOP EXIT
DO LOOP UNTIL
DO LOOP WHILE
END
FOR NEXT
GOSUB RETURN
GOTO
IF THEN
IF THEN ELSE
ON GOSUB
ON GOTO
REM
RUN      (wird auch häufig als Systembefehl eingesetzt)
SLEEP
SYS      (wird auch häufig als Systembefehl eingesetzt)
USR      (wird auch häufig als Systembefehl eingesetzt)
```

### Grafikbefehle

```
BOX
CHAR
CIRCLE
COLOR
DRAW
GRAPHIC
LOCATE
PAINT
*RCLR
*RDOT
*RLUM
```

SCALE
WIDTH

### Sprite-Befehle

*BUMP
COLLISION
GSHAPE
MOVSPR
*RSPCOLOR
*RSPPOS
*RSPRITE
SPRCOLOR
SPRDEF
SPRITE
SPRSAV
SSHAPE

### Sound-Befehle

ENVELOPE
FILTER
PLAY
SOUND
TEMPO
VOL

### Sonstige Befehle

DATA
DIM
LET
RESTORE
*XOR

# Programmverzeichnis

# Stichwortverzeichnis

# Die SYBEX Bibliothek

## SPRACHEN

### BASIC

**BASIC COMPUTER SPIELE/Band 1**
**herausgegeben von David H. Ahl** – die besten Mikrocomputerspiele aus der Zeitschrift „Creative Computing" in deutscher Fassung mit Probelauf und Programmlisting. 208 Seiten, 56 Abbildungen, Best.-Nr. **3009**

**BASIC COMPUTER SPIELE/Band 2**
**herausgegeben von David H. Ahl** – 84 weitere Mikrocomputerspiele aus „Creative Computing". Alle in Microsoft-BASIC geschrieben mit Listing und Probelauf. 224 Seiten, 61 Abbildungen, Best.-Nr.: **3010**

**BASIC PROGRAMME – MATHEMATIK, STATISTIK, INFORMATIK**
**von Alan Miller** – eine Bibliothek von Programmen zu den wichtigsten Problemlösungen mit numerischen Verfahren, alle in BASIC geschrieben, mit Musterlauf und Programmlisting. 352 Seiten, 147 Abbildungen, Best.-Nr.: **3015** (1983)

**PLANEN UND ENTSCHEIDEN MIT BASIC**
**von X. T. Bui** – eine Sammlung von interaktiven, kommerziell-orientierten BASIC-Programmen für Management- und Planungsentscheidungen. 200 Seiten, 53 Abbildungen, Best.-Nr.: **3025** (1983)

**BASIC FÜR DEN KAUFMANN**
**von D. Hergert** – das BASIC-Buch für Studenten und Praktiker im kaufmännischen Bereich. Enthält Anwendungsbeispiele für Verkaufs- und Finanzberichte, Grafiken, Abschreibungen u.v.m. 208 Seiten, 85 Abbildungen, Best.-Nr.: **3026** (1983)

**GRUNDKURS IN BASIC**
**von U. Ströbel** – die Einführung in die meistgenutzte Programmiersprache für Lehrer, Schüler und das Selbststudium (Reihe SYBEX Informatik). 208 Seiten, mit Abb., Best.-Nr. **3058** (1985), Lehrerbegleitheft Best.-Nr. **3091** ( 1985)

### Pascal

**EINFÜHRUNG IN PASCAL UND UCSD/PASCAL**
**von Rodnay Zaks** – das Buch für jeden, der die Programmiersprache PASCAL lernen möchte. Vorkenntnisse in Computerprogrammierung werden nicht vorausgesetzt. Eine schrittweise Einführung mit vielen Übungen und Beispielen. 535 Seiten, 130 Abbildungen, Best.-Nr.: **3004** (1982)

**DAS PASCAL HANDBUCH**
**von Jacques Tiberghien** – ein Wörterbuch mit jeder Pascal-Anweisung und jedem Symbol, reservierten Wort, Bezeichner und Operator, für beinahe alle bekannten Pascal-Versionen. 480 Seiten, 270 Abbildungen, Format 23 x 18 cm, Best.-Nr.: **3005** (1982)

**GRUNDKURS IN PASCAL Bd. 1**
**von K.-H. Rollke** – der sichere Einstieg in Pascal, speziell für Schule und Fortbildung (Reihe SYBEX Informatik). 224 Seiten, mit Abb., Format 17,5x25 cm, Best.-Nr. **3046** (1984), Lehrerbegleitheft Best.-Nr. **3059**

**GRUNDKURS IN PASCAL BAND 2**
**von K.H. Rollke** – Mit diesem Buch wird der Pascal-Grundkurs aus der Reihe SYBEX Informatik abgerundet. Für Lehrer, Schüler, Teilnehmer an Pascal-Kursen, Studenten und Autodidakten. 224 Seiten, mit Abb., Best.-Nr. **3061** (1985), Lehrerbegleitheft Best.-Nr. **3090**

**DAS TURBO PASCAL BUCH**
**von Karl-Hermann Rollke** – Sie lernen die Arbeitsweise des Turbo-Editors und des Systems kennen und werden mit Programmierkonzepten, Daten- und Kontrollstrukturen für den Programmfluß vertraut gemacht. 288 Seiten, mit Abbildungen, Best.-Nr.: **3608** (1985)

## Assembler

**PROGRAMMIERUNG DES Z80**
**von Rodnay Zaks** – ein umfassendes Nachschlagewerk zum Z80-Mikroprozessor – jetzt in einer durch Lösungen ergänzten Ausgabe. 2., erweiterte Ausgabe. 640 Seiten, 176 Abbildungen, Best.-Nr.: **3099** (1985)

**PROGRAMMIERUNG DES 6502 mit 6510/65C02/65SC02**
**von Rodnay Zaks** – Programmierung in Maschinensprache mit dem Mikroprozessor 6502 und anderen Mitgliedern der 65xx Familie, von den Grundkonzepten bis hin zu fortgeschrittenen Informationsstrukturen. 3. überarbeitete und erweiterte Ausgabe. 440 Seiten, 170 Abbildungen, Best.-Nr.: **3600** (1985)

**PROGRAMMIERUNG DES 8086/8088**
**von J. W. Coffron** – lehrt Sie Programmierung, Kontrolle und Anwendung dieses 16-Bit-Mikroprozessors; vermittelt Ihnen das notwendige Wissen zu optimaler Nutzung Ihrer Maschine, von der internen Architektur bis hin zu fortgeschrittenen Adressierungstechniken. 312 Seiten, 107 Abbildungen, Best.-Nr.: **3050** (1984)

**PROGRAMMIERUNG DES 68000**
**von C. Vieillefond** – macht Sie mit dem 32-bit-Prozessor von leistungsstarken Rechnern wie Macintosh, Amiga, ATARI ST und Sinclair QL vertraut; erläutert die Struktur des 68000, den Aufbau des Speichers, die Adressierungsarten und den Befehlssatz. 456 Seiten, 150 Abb., Best.-Nr. **3060** (1985)

## Andere Programmiersprachen

**ERFOLGREICH PROGRAMMIEREN MIT C**
**von J. A. Illik** – ein unentbehrliches Handbuch für jeden, der mit der universellen Sprache C erfolgreich programmieren will. Aussagekräftige Beispiele, auf verschiedenen Mini- und Mikrocomputern getestet. 408 Seiten, Best.-Nr.: **3055** (1984)

## Spezielle Geräte

### Apple

**WAS IST WO IM APPLE**
**von William F. Luebbert** – eine systematische Arbeitshilfe für Besitzer eines Apple II, II+ oder IIe mit vielen alphabetisch und numerisch sortierten Apple-Routinen und Speicheradressen, die Ihnen hilft, viel Arbeit und Zeit zu sparen. 496 Seiten, mit 84 Abb., Best.-Nr. **3098** (erscheint 1985)

**DAS ProDOS HANDBUCH**
**von Karen und Timothy Rice** – Anfänger und Fortgeschrittene lernen alles Wissenswerte über das starke Betriebssystem für Apple und Apple-Kompatible. 272 Seiten, 29 Abbildungen, Best.-Nr. **3617** (1985)

### Atari

**ATARI BASIC HANDBUCH**
**von J. Reschke** – das vollständige ABC der BASIC-Programmierung für den Atari mit Erläuterung durch viele praktische Beispiele. 208 Seiten, mit Abb., Best.-Nr. **3083** (1984)

**DAS ATARI PROFIBUCH**
**von Julian Reschke und Andreas Wiethoff** – Das richtige Buch für Sie, wenn Sie einen Atari 400, 800, 600 XL, 800 XL oder 130 XE besitzen. 296 Seiten, mit Abbildungen, Best.-Nr. **3605** (1986)

**ATARI STARTEXTER**
**von Toni Schwaiger** – Das Spitzen-Textverarbeitungs-Programm für die ATARI-Heimcomputer 400/800/600XL/800XL/130XE (400 und 600XL mit Speichererweiterung). Komplett mit ausführlichem Trainingsbuch. Best.-Nr. **3414** (1985)

### Commodore

**FARBSPIELE MIT DEM COMMODORE 64**
**von W. Black und M. Richter** – 20 herrliche Farbspiele für Ihren C64, mit Beschreibung, Programmlisten und Bildschirm-Darstellungen. Für mehr Freizeit-Spaß mit Ihrem Commodore! 176 Seiten, 58 Abbildungen, Best.-Nr.: **3044** (1984)

**COMMODORE 64 – GRAFIK + DESIGN**
**von Ch. Platt** – Eine Schritt-für-Schritt-Einführung in die Grafik-Programmierung Ihres C 64. Tips, die Sie in keinem Handbuch finden. 280 S., 150 Abb., Best.-Nr. **3073** (1984)

**COMMODORE 64 PROGRAMMSAMMLUNG**
**von S. R. Trost** – mehr als 70 getestete Anwenderprogramme, die direkt eingegeben werden können. Erläuterungen gewährleisten eine optimale Nutzung. 192 Seiten, 160 Abbildungen, Best.-Nr.: **3051** (1983)

**SPASS AN MATHE MIT DEM COMMODORE 64**
**von H. Danielsson** – zeigt Ihnen mit vielen Beispielen, wie der C 64 für schulische oder private Berechnungen genutzt werden kann. 280 Seiten mit Abb., Best.-Nr. **3072** ( 1985)

**COMMODORE 64 BASIC-KURS MIT HONEY-AID**
**Reihe MISTER MICRO** – BASIC auf dem C64 durch Praxis lernen; mit dem integrierten Lernpaket (Buch + Software). Außer vielen Übungsprogrammen: Honey-Aid – eine universell einsetzbare BASIC-Erweiterung mit 28 zusätzlichen Befehlen. 352 Seiten, Buch und Kassette, Best.-Nr. **3400**, Buch und Diskette, Best.-Nr. **3401** (1984)

**COMMODORE 64 ASSEMBLER-KURS**
**Reihe MISTER MICRO** – zeigt in Theorie und Praxis, wie Sie den 6510-Prozessor Ihres C64 programmieren. Der mitgelieferte Assembler ist universell einsetzbar. 296 Seiten, Buch und Kassette, Best.-Nr. **3402**, Buch und Diskette, Best.-Nr. **3403** (1984)

**COMMODORE 64 STARTEXTER**
**Textverarbeitungs-Kurs aus der Reihe MISTER MICRO** – StarTexter ist ein Textverarbeitungskurs mit Doppelnutzen: das Buch führt Sie in die Textverarbeitung mit Ihrem C64 ein, die Diskette bietet Ihnen ein exzellentes Programm – komplett zu einem erstaunlichen Preis! 112 Seiten, Handbuch und Diskette, Best.-Nr.: **3411** (1985)

**COMMODORE 64 STARDATEI**
**von Toni Schwaiger** – Der universelle Karteikasten für den C 64 aus der Reihe MISTER MICRO, voll kompatibel zu StarTexter. Diskette und ausführliches Trainingsbuch (ca. 120 Seiten) Best.-Nr. **3413** (1985)

## Schneider

**MEIN SCHNEIDER CPC**
**von Norbert und Christoph Hesselmann** – von der Hardware-Umgebung über ein umfangreiches BASIC-Lexikon, Grafikentwurf und Musikerzeugung, Mikroprozessorentechnik und Maschinensprache, Arbeiten mit dem Disketten-Laufwerk bis hin zu den Betriebssystemen AMSDOS und CP/M 2.2. 376 Seiten, 124 Abbildungen, Best.-Nr.: **3602** (1985)

**ARBEITEN MIT DEM SCHNEIDER CPC**
**von Hans Lorenz Schneider** – eine umfassende und didaktisch aufbereitete Arbeitshilfe für Anfänger, aber auch Fortgeschrittene finden ein Bündel von Tips und Tricks. 288 Seiten, 113 Abbildungen, Best.-Nr.: **3603** (1985)

SYBEX

# Das große Commodore BASIC Handbuch

**Darauf haben Commodore-Fans schon lange gewartet: BASIC komplett für alle Rechner von VC20 bis C128!**

Grundlage des Buches ist BASIC 7.0 – das außerordentlich leistungsstarke und komfortable BASIC des Commodore 128 mit ca. 150 Befehlen. Von diesen Befehlen ausgehend, zeigt das Buch im alphabetisch gegliederten Hauptteil Übereinstimmungen mit und Unterschiede zu den bisherigen, aufwärts kompatiblen BASIC-Versionen. So erhalten Sie detaillierte Informationen zu

- **BASIC 2.0** (VC20, C64, Rechner der 2000-/3000-Serien)
- **BASIC 4.0** (Rechner der 4000-/8000-/600-/700-Serien)
- **BASIC 3.5** (C16, C116, Plus/4)
- **BASIC 7.0** (C128)

Dabei lernen Sie auch, in einer bestimmten Version nicht vorhandene Befehle so zu simulieren, daß die gleiche Funktion erfüllt wird.

Zusätzliche Arbeitshilfen für Anfänger und Fortgeschrittene bieten sorgfältig erarbeitete Fakten zu Themen wie:

- Interpreter und Editor
- Programmsteuerung und -strukturierung
- Ein-/Ausgabefunktionen und -geräte
- Dateiverwaltung

Und vor allem: Praktische Tips, wie Sie Programme zwischen den verschiedenen Geräten übertragen und lauffähig machen können!

Richtig rund wird das Buch durch die ausführlichen Anhänge: Hier finden Sie alphabetische Befehlsübersichten mit Erläuterungen zu den BASIC-Erweiterungen SIMON'S BASIC, EXBASIC und HONEY.AID, die das BASIC 2.0 quasi auf die Ebene des 7.0 anheben.

**Ein Muß für alle Commodore-Anwender!**

ISBN 3-88745-615-7

**DM 58,–**
**öS 452,–**
**sfr 53,40**

9 783887 456153