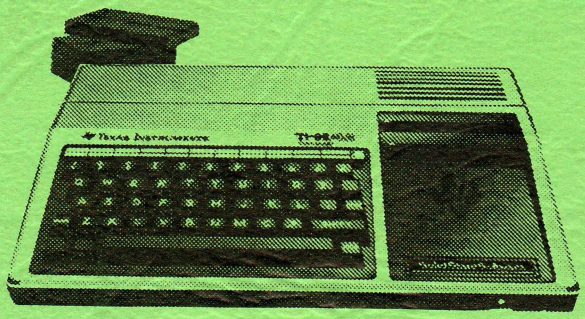
Norbert Dutz / Klaus Meier

# TI EXTENDED BASIC

## Deutsches Handbuch zum TI EXTENDED BASIC-Modul für TI 99/4A



HASSE - VERLAG

# H A N D B U C H

z u m


Deutsche Ausgabe

DUMESOFT

1982

## Impressum

Übertragung der amerikanischen Originalausgabe:

Robert E. Whitsitt u.a.
TI Extended BASIC for the TI-99/4 Home Computer
Texas Instruments Inc. 1981



DUMESOFT, 1982 / 1. Aufl.

==========

Im Endeffekt wird der Mikrocomputer nie
schlauer werden als das menschliche Gehirn -
Jedoch hat der Mikrocomputer dem Menschen
eines voraus. Er kennt nur die präzise
Entscheidung: "Ja" oder "Nein" ...
dem Menschen lastet er an, auch "Jein"
zu sagen und so zu handeln.

D. Nührmann

# Vorwort

Nachdem in der Zeit, in der wir uns selbst mit dem TI 99/4A und der Programmerstellung für diesen Computer befassen, von verschiedenen Seiten mehrfach das Fehlen eines deutschsprachigen Handbuchs für den EXTENDED BASIC-Modul beklagt wurde, haben wir uns entschlossen, das EXTENDED BASIC MANUAL ins Deutsche zu übertragen.

Dabei sprechen wir bewußt von einer "Übertragung" und nicht von einer "Übersetzung", denn schon beim ersten flüchtigen Vergleich zwischen der nun vorliegenden Ausgabe und dem Original werden deutliche Unterschiede formaler und inhaltlicher Art ersichtlich. Die Gründe resultieren aus unserem Bemühen, ein auf den deutschsprachigen Anwender zugeschnittenes Handbuch zu erstellen, das ihm nicht nur einen leichten ersten Zugang zum ERWEITERTEN BASIC verschaffen, sondern auch dem geübten Benutzer einen raschen und direkten Zugriff ermöglichen sollte. Dabei gingen wir u.a. davon aus, daß die Beispielprogramme für den deutschsprachigen Anwender durchsichtiger gestaltet werden mußten. Eine Verzögerungsschleife z.B. ist für den des Englischen Unkundigen in der Form

```
100 FOR DELAY=1 TO 1000 :: NEXT DELAY
```

eben nicht so leicht als solche erkennbar wie in der Form

```
100 FOR VERZOEG=1 TO 1000 :: NEXT VERZOEG
```

In dieser Konsequenz wurden nicht nur Programmteile, sondern z.T. ganze Beispielprogramme geändert oder ausgetauscht, weil wir glaubten, sie aktualisieren und damit durchsichtiger und anschaulicher gestalten zu können.

Wie die Originalausgabe (s.d. S.10) gingen wir davon aus, daß der Leser bereits grundlegende Erfahrung aus der Arbeit mit der Bedienungsanleitung zum TI-99/4(A) mitbringt. Unter dem Aspekt eines schnellen und direkten Zugriffs, der möglichst nur auf ein Handbuch beschränkt werden sollte, haben wir aber die Ausführungen der Bedienungsanleitung stärker berücksichtigt, als dies die Originalausgabe tut.

Für jeden Befehl - unter diesem Sammelbegriff werden von uns im weiteren Verlauf COMMANDS, STATEMENTS und FUNCTIONS zusammengefaßt - findet der Leser im Hauptteil dieses Handbuchs (Teil B) die Angabe über das Vorhandensein dieses Befehls in TI- und/oder EXTENDED BASIC. Andererseits glaubten wir, es in einigen wenigen Fällen verantworten zu können, gegenüber der Originalausgabe zu kürzen. So fehlt gegenüber der Originalausgabe (vgl. Appendix L,M, S.203f) die auf den Sprachsynthesizer bezogene Wortliste ebenso wie die Ausführung zur Endsilbenergänzung, weil wir davon ausgegangen sind, daß ein über amerikanisches Vokabular verfügender Synthesizer für den deutschsprachigen Benutzer nur von geringem Interesse sein dürfte und in einem solch speziellen Fall sicher auf die gesonderte Bedienungsanleitung für den Synthesizer zurückgegriffen wird. Auch auf die Liste der Beispielprogramme (Appendix A, S.192f) wurde verzichtet, da deren praktischer Nutzen bezweifelt werden kann. Doch sind zur Arbeitserleichterung für den Benutzer alle ausführlichen Beispielprogramme auf Cassette/Diskette erhältlich.

Insgesamt halten sich die hinsichtlich des Anhangs (Appendices) und des Hauptteils B (Reference Section, Kapitel 4) durchgeführten Änderungen in Grenzen, während die durch den Teil A vermittelte Übersicht der Originalausgabe (Kapitel 1 - 3) aus Gründen der Durchsichtigkeit und Schwerpunktsetzung, sowie unter Hinzufügung nachträglicher Produktinformationen seitens TEXAS INSTRUMENTS ("Important Product Information for TI Extended Basic") grundsätzlich neu konzipiert wurde.

## Vorwort

Das vorliegende Handbuch wurde mit Hilfe eines TI-99/4A, einer Diskettenstation, einem Drucker, der über eine RS232-Schnittstelle betrieben wurde, und einem eigenen Textverarbeitungsprogramm geschrieben. Die so erstellten Seiten wurden anschließend auf fototechnischem Wege verkleinert und vervielfältigt. Dieses Verfahren zeigt zum einen die Leistungsfähigkeit des Computersystems, zum anderen birgt es den Nachteil des etwas unübersichtlichen Druckbildes in sich, der aber auf Grund der wesentlichen Zeitersparnis in Kauf genommen wurde. Auch wurde die Anzahl der Zeichen in einer Zeile, seien es 28 auf dem Bildschirm oder 32 auf dem Thermodrucker, im Gegensatz zur Bedienungsanleitung aus Gründen der Übersichtlichkeit in den Programmdarstellungen nicht berücksichtigt.

Selbstverständlich sehen wir Anfragen und Anregungen seitens unserer Leser mit großem Interesse entgegen. Uns bleibt zunächst, dem Leser viel Vergnügen und einen Gewinn an Informationen bei der Lektüre dieses Handbuchs zu wünschen.

Im September 1982

Norbert Dutz
Klaus Meier

# Inhaltsverzeichnis

# Inhaltsverzeichnis

B. BEFEHLSLISTE

Die mit (D) gekennzeichneten Befehle können nur direkt ausgeführt werden, d.h. sie dürfen n i c h t (mit Programmzeilennummern versehen) in Programmen stehen. Die mit (U) gekennzeichneten Befehle sind Unterprogrammnamen und können nur in Verbindung mit dem Schlüsselwort CALL verwendet werden:

# Teil A

## A L L G E M E I N E  Ü B E R S I C H T

# Kapitel 1

## V O R Ü B E R L E G U N G E N

# Vorüberlegungen

## GEBRAUCH DES HANDBUCHS

Das Vorwort zu diesem Handbuch enthält die grundsätzlichen Erwägungen, die zu dem gegenüber der Originalausgabe teilweise veränderten Aufbau geführt haben und aus denen die folgende Dreigliedrigkeit resultiert:

TEIL A: Allgemeine Übersicht
(Kapitel 1 - 5)
TEIL B: Befehlsliste
TEIL C: Anhang

Der geübte und erfahrene Benutzer wird sich kaum mit allen Einzelheiten des einführenden Teils A befassen müssen. Da das Handbuch auf den TI 99/4A abgestimmt ist, sei der Besitzer des Vorgängermodells jedoch besonders auf den im 1. Kapitel vorhandenen Vergleich der beiden Computertypen verwiesen Für den Anfänger unerläßlich, für den Fortgeschrittenen zur Ausschöpfung aller Möglichkeiten nützlich, ist die Beschäftigung mit dem 4. Kapitel, in das auch nachträgliche Produktinformationen seitens TI eingearbeitet sind. Die Elemente des ERWEITERTEN BASIC werden im 3. Kapitel vorgestellt und beschrieben, wobei die Erweiterungen auch auf dem Hintergrund eines Vergleichs zum TI-BASIC verdeutlicht werden sollen. Unterschiedlich zum TI-BASIC sind auch einige wesentliche Konventionen (Übereinkünfte und Voraussetzungen), ohne deren Kenntnis im besten Fall die Möglichkeiten des EXTENDED BASIC nicht ausgeschöpft werden können, im schlechtesten Fall aber die Programme nicht laufen: deshalb sollte sich auch der erfahrene Benutzer mit den Kapiteln 3 bis 5 auf jeden Fall befassen, ehe er vielleicht seine neu erworbenen Kenntnisse anhand des am Ende von Kapitel 5 dargestellten Beispielprogramms überprüft.

Der Teil A des Handbuchs (Allgemeine Übersicht) steht in engem Zusammenhang mit Teil C (Anhang), in dem wesentliche Informationen tabellarisch zusammengefaßt sind, auf die der Benutzer immer wieder wird zurückgreifen müssen, ob er nun Zeichen selbst definiert oder Musik machen will. Der Hauptteil des Handbuchs (Teil B) stellt - alphabetisch geordnet - alle Befehle (Commands, Statements, Functions, Subprograms) im einzelnen ausführlich vor.

Dabei wird in der Kopfleiste unterschieden, für welches BASIC (TI- und/oder EXTENDED-) der Befehl Anwendung findet. Danach folgt das Eingabeformat des jeweiligen Befehls. Optionen (Möglichkeiten, die im Befehl nicht unbedingt angegeben werden müssen) sind in eckigen Klammern wiedergegeben. Runde Klammern müssen dagegen bei der Programmierung unbedingt berücksichtigt werden. Kann ein Befehl in mehreren Versionen benutzt werden, so sind die verschiedenen Möglichkeiten untereinander aufgeführt. Auf die Darstellung des Formats folgt eine allgemeine Beschreibung (Description) des Befehls, an die sich die Erklärung hinsichtlich der Wirkungsweise der verschiedenen Optionen (s.o.) anschließt, sofern solche vorhanden sind. Beispiele (Examples) und häufig komplette Beispielprogramme (Programs) schließen jeweils die Erläuterungen zu den einzelnen Befehlen ab.

Im Zusammenhang mit Zeilennummern taucht das Nummernzeichen in der Form ### auf (z.B. bei Fehlermeldungen). Damit ist dann eine beliebige in einem Programm auftretende Zeilennummer gemeint. Davon zu unterscheiden ist das Auftreten des #-Zeichens im Format-String des IMAGE-Befehls (s.d.) und im Zusammenhang mit #Dateinummern von externen Dateien.

# Vorüberlegungen

## TI-99/4 UND TI-99/4A IM VERGLEICH

Nicht ausgesprochen wie Zwillinge, doch wie Brüder, deren jüngerer (4A) an der Standdard-Schreibmaschinentastatur erkennbar ist, gleichen sich die beiden Computertypen 99/4 und 99/4A, was besagt, daß ihre Gemeinsamkeiten größer sind als ihre Unterschiede, wobei die in diesem Zusammenhang wesentlichste Gemeinsamkeit in der Tatsache zu sehen ist, daß das EXTENDED BASIC-Modul bei beiden Computertypen verwendet werden kann und beiden Typen auch die verbesserten Möglichkeiten dieses Moduls eröffnet, die im folgenden Kapitel vorgestellt werden. Zur Ausnutzung dieser Möglichkeiten müssen jedoch Unterschiede in der Anwendung bei beiden Computertypen beachtet werden, die in diesem Abschnitt aufgezeigt werden sollen.

### Spezielle Tastenfunktionen

Mit bestimmten Tasten lassen sich bei beiden Computertypen Sonderfunktionen aktivieren, die im einzelnen im Abschnitt "Spezielle Tastenfunktionen" erläutert werden. An dieser Stelle ist festzuhalten, daß die Aktivierung beim TI 99/4 durch gleichzeitiges Drücken der SHIFT-Taste und einer zugeordneten Buchstabentaste erfolgt, während beim TI-99/4A gleichzeitig die FCTN-Taste (mit grauem Punkt markiert) und eine zugeordnete Taste gedrückt werden müssen. Eine Ausnahme bildet lediglich die Eingabe(ENTER)-Taste, die bei beiden Computertypen allein zu drücken ist.

Über die FCTN-Taste werden beim TI-99/4A zudem die Symbole erreicht, die bei diesem Computertyp auf der Tastenvorderseite dargestellt sind. Die Shift-Taste dient bei diesem Computertyp dagegen im wesentlichen der Großschreibung, die wie bei einer normalen Schreibmaschine durch gleichzeitiges Drücken dieser und der entsprechenden Buchstabentaste bewirkt wird, aber auch alle anderen, in der oberen Tatenhälfte dargestellten Zeichen werden so erreicht (z.B."%"). Eine Dauergroßschreibung erreicht man durch Einrasten der ALPHA LOCK-Taste, durch die die Ziffern- und Interpunktionstasten nicht beeinflußt werden. Durch erneutes Drücken dieser Taste wird die Großschreibung wieder aufgehoben. Beim Gebrauch der Fernbedienung sollte die ALPHA LOCK-Taste gelöst sein, damit eine einwandfreie Funktion gewährleistet ist.

Für beide Computertypen ist bei der Benutzung aller Tasten die automatische Wiederholungsfunktion zu beachten. Diese bewirkt, daß ein Zeichen, dessen entsprechende Tate länger als eine Sekunde gedrückt wird, solange automatisch wiederholt wird, bis die Taste wieder losgelassen wird.

### Groß- und Kleinschreibung

Die Möglichkeit der Groß- und Kleinschreibung führt aber auch zu einem weiteren Unterschied zwischen beiden Computertypen, denn nur der TI-99/4A kann ein Programm einlesen, dessen Namen Kleinbuchstaben enthält. Für diesen Computer bezeichnen "PROGRAMM" und "Programm" zwei verschiedene Programme, während der TI 99/4 nicht in der Lage ist, ein mit dem Namen "Programm" bezeichnetes Programm einzulesen. Die erweiterte Möglichkeit, die die Verwendung von Kleinbuchstaben in DATA-Befehlen oder in Anführungszeichen gesetzte Strings hinsichtlich der Programmiertechniken beim TI 99/4A darstellt, ist beim TI 99/4, der diese Buchstaben zunächst nicht darstellen kann, jedoch nur teilweise eingeschränkt.Sofern nämlich genügend Speicherplatz zur Verfügung steht, kann man durch Einfügen der folgenden Programmzeilen auch den TI 99/4 veranlassen, Kleinbuchstaben eines TI 99/4A-Programms darzustellen, wobei die Programmschleife natürlich auch einen Zeitverlust hinsichtlich der Programmausführung bedeutet:

## Vorüberlegungen

```
100 REM ***** KLEINBUCHSTABEN FUER TI 99/4 *****
110 FOR CODE=65 TO 90
120 CALL CHARPAT(CODE,GROSS$)
130 KLEIN$="0000"&SEG$(GROSS$,1,4)&SEG$(GROSS$,7,4)&SEG$(GROSS$,13,4)
140 CALL CHAR(CODE+32,KLEIN$)
150 NEXT CODE
```

Externe Speicher müssen aber immer - auch beim TI-99/4A - mit Großbuchstaben (z.B. CS1) angesprochen werden, ein Gerätename in der Form "Dsk1" würde automatisch zu einer Fehlermeldung führen. Ebenso ist bei der sogenannten wissenschaftlichen Schreibweise numerischer Daten "E" immer als Großbuchstabe zu schreiben.

### Steuertasten

Nur der TI-99/4A verfügt über spezielle Steuertasten, die vor allem der Daten(fern)übertragung dienen (Telekommunikation). Zur Eingabe von Steuerzeichen sind gleichzeitig die CTRL-Taste (mit rotem Punkt markiert) und die jeweils zugeordnete Taste zu drücken. Diesen sind wie den Funktionstasten Codes zugeordnet, die der Vollständigkeit halber nicht unerwähnt bleiben sollen:

CODES FÜR STEUERTASTEN

| BASIC-Modus | Pascal-Modus | Kurzzeichen | Tasten | Anmerkungen |
|---|---|---|---|---|
| 129 | 1 | SOH | <CONTROL-A> | Anfang des Kopfes |
| 130 | 2 | STX | <CONTROL-B> | Anfang des Textes |
| 131 | 3 | ETX | <CONTROL-C> | Ende des Textes |
| 132 | 4 | EOT | <CONTROL-D> | Ende der Übertragung |
| 133 | 5 | ENQ | <CONTROL-E> | Stationsaufforderung |
| 134 | 6 | ACK | <CONTROL-F> | Positive Rückmeldung |
| 135 | 7 | BELL | <CONTROL-G> | Klingel |
| 136 | 8 | BS | <CONTROL-H> | Rückwärtsschritt |
| 137 | 9 | HT | <CONTROL-I> | Horizontal-Tabulator |
| 138 | 10 | LF | <CONTROL-J> | Zeilenvorschub |
| 139 | 11 | VT | <CONTROL-K> | Vertikal-Tabulator |
| 140 | 12 | FF | <CONTROL-L> | Formularvorschub |
| 141 | 13 | CR | <CONTROL-M> | Wagenrücklauf |
| 142 | 14 | SO | <CONTROL-N> | Dauerumschaltung |
| 143 | 15 | SI | <CONTROL-O> | Rückschaltung |
| 144 | 16 | DLE | <CONTROL-P> | Datenübertragungsumschaltung |
| 145 | 17 | DC1 | <CONTROL-Q> | Gerätesteuerzeichen (X-EIN) |
| 146 | 18 | DC2 | <CONTROL-R> | Gerätesteuerzeichen |
| 147 | 19 | DC3 | <CONTROL-S> | Gerätesteuerzeichen (X-AUS) |
| 148 | 20 | DC4 | <CONTROL-T> | Gerätesteuerzeichen |
| 149 | 21 | NAK | <CONTROL-U> | Negative Rückmeldung |
| 150 | 22 | SYN | <CONTROL-V> | Synchronisierung |
| 151 | 23 | ETB | <CONTROL-W> | Ende des Übertragungsblocks |
| 152 | 24 | CAN | <CONTROL-X> | Ungültig |
| 153 | 25 | EM | <CONTROL-Y> | Ende der Aufzeichnung |
| 154 | 26 | SUB | <CONTROL-Z> | Substitutionszeichen |
| 155 | 27 | ESC | <CONTROL-,> | Code-Umschaltung |
| 156 | 28 | FS | <CONTROL-;> | Hauptgruppen-Trennzeichen |
| 157 | 29 | GS | <CONTROL-=> | Gruppen-Trennzeichen |
| 158 | 30 | RS | <CONTROL-8> | Untergruppen-Trennzeichen |
| 159 | 31 | US | <CONTROL-9> | Teilgruppen-Kennzeichen |

## Vorüberlegungen

Die Steuercodes hängen vom im jeweiligen KEY-Unterprogramm gewählten Tastatur-Modus ab (vgl. CALL KEY-Befehl).

Mit Hilfe der mitgelieferten Tastaturschablone, die in die Klemmleiste über den Tasten des Computers befestigt werden kann, können die verschiedenen Tastenfunktionen in Verbindung mit FCTN und CTRL beim TI 99/4A besser erkannt werden, da sich die obere, mit einem roten Punkt gekennzeichnete Reihe auf die CTRL-Taste und die untere Reihe mit dem roten Punkt auf die FCTN-Taste bezieht (gleichzeitig drücken!).

### KEY-Unterprogramm

Wie oben schon angedeutet, sind den Steuer- und Funktionstasten Codes zugeordnet, die vom im KEY-Unterprogramm gewählten Tastaturmodus abhängen (vgl. CALL-KEY-Befehl). Die speziellen Tastaturmodi 3, 4 und 5 sind für den TI-99/4 nicht verfügbar.

### Assembler

Assembler-Unterprogramme können mit dem TI-99/4 nicht benutzt werden, der TI-99/4A benötigt dazu die Speichererweiterung (MEMORY EXPANSION UNIT).

### Zeichencodes

Das Vorhandensein von Kleinbuchstaben, die in der Bildschirmdarstellung als verkleinerte Großbuchstaben, bei der Druckerausgabe aber als echte Kleinbuchstaben erscheinen, macht deutlich, daß der Standardzeichensatz, der die vordefinierten Zeichen für den Computer enthält, für den TI-99/4A größer sein muß als für den TI-99/4.

Im Anhang finden Sie eine Tabelle, die die für den TI-99/4A definierten Zeichen und ihre Codes darstellt. Es handelt sich dabei um den sog. Standard-ASCII-Zeichensatz für die Codes von 32 bis 127 und die für den TI-99/4A zusätzlichen Zeichen für den Positionszeiger (Cursor) und das Randzeichen (ASCII-Codes 30 und 31). Für den TI-99/4 sind dagegen die Zeichen mit den Codes 32 bis 95 vorausdefiniert, d.h. dessen Zeichenumfang ist gegenüber dem TI-99/4A eingeschränkt.

Der Gesamtzeichensatz ist in Gruppen zu je 8 Zeichen aufgeteilt, wobei die Zeichen mit den Codes 30 und 31 die Gruppe 0 und die nicht definierten, d.h. frei bestimmbaren Zeichen mit den Codes 128 - 135 die Gruppe 13 und mit den Codes 136 - 143 die Gruppe 14 bilden. Eine solche Gruppe von Zeichen wird auch als Zeichensatz bezeichnet, wir bleiben jedoch hier bei der Bezeichnung Zeichengruppe. Damit verfügt der Computer also über 15 Zeichengruppen.

Wenn Sie sich jetzt - vom TI BASIC kommend - wundern, wo die nicht definierten Zeichen mit den Codes 144 - 159 geblieben sind, so lassen Sie sich auf Kapitel 2 vertrösten. Auch wenn Sie sich mit dem Erwerb des EXTENDED BASIC-Moduls den Verlust von zwei Zeichensätzen erkauft haben, so ist das kein Grund, den Modul jetzt wegzuwerfen, denn wie das folgende Kapitel zeigt, stehen diesem e i n e n Nachteil v i e l e Vorteile gegenüber. Der Vorbereitung Ihres Computers sollte also nach dem Studium dieser Zeilen nichts mehr im Wege stehen.

## Vorüberlegungen

### VORBEREITUNG DES COMPUTERS

Ehe der Computer TI-99/4(A) mit dem EXTENDED BASIC benutzt werden kann, muß der entsprechende Modul in die öffnung der rechten Konsolenseite eingeschoben werden, bis er fest an seinem Platz sitzt. Bei ausgeschaltetem Computer kann dies sofort geschehen. Danach schalten Sie alle angeschlossenen peripheren Geräte wie Drucker, Disketten-Laufwerk usw. ein, sofern solche benutzt werden sollen. Zum Schluß wird der Computer selbst eingeschaltet, und das Titelbild erscheint auf dem Bildschirm.

Sollte der Computer bereits eingeschaltet sein, dann kehren Sie durch Eingabe des BYE-Befehls (vgl. Teil B!) zum Titelbild zurück. Dies könnte auch durch Betätigung der Tasten <FCTN-=> (QUIT, s. spezielle Tastenfunktionen!) geschehen, doch dabei werden alle schon im Computer befindlichen Daten und Programme gelöscht, sowie alle offenen Dateien (vgl. OPEN-Befehl!) nicht geschlossen. Deshalb ist es grundsätzlich empfehlenswert, den BYE-Befehl und nicht <FCTN-=> (QUIT) zur Rückkehr zum Titelbild zu benutzen.

Ausgehend vom Titelbild wird durch Drücken einer beliebigen Taste die sogenannte Auswahlliste auf dem Bildschirm dargestellt, die hinter Wahlzahlen die möglichen Sprachen TI BASIC (1) oder EXTENDED BASIC (2) enthält. durch Betätigen der Taste <2> (ohne <ENTER>!) wählen Sie das "ERWEITERTE BASIC", und der Computer meldet sich mit * READY *.

Nun ist es soweit! Ihrer praktischen Programmiererfahrung und der Anwendung Ihrer Kenntnisse steht nichts mehr im Wege - oder doch fast nichts! überschlagen Sie ruhig das nächste Kapitel - dem einen oder anderen dürfte es ohnehin etwas zu unausgewogen ( e i n  Nachteil - v i e l e  Vorteile) erscheinen -, es sei denn, Sie trauern immer noch den beiden verlorengegangenen Zeichensätzen nach! Aber vielleicht blFttern Sie es auch später einmal durch, wenn Sie schon mit dem EXTENDED BASIC-Modul experimentiert haben. Vielleicht fällt Ihnen dann dabei die eine oder andere Möglichkeit auf, die sich mit der Anwendung dieses Moduls ergibt und die Sie vielleicht noch nicht oder noch zu wenig genutzt haben, und Sie können Ihre Programme noch schneller, noch schöner, noch überschaubarer - mit einem Wort: noch besser gestalten.

# Kapitel 2

## M ö G L I C H K E I T E N

## Möglichkeiten

### EIN NACHTEIL

Wie schon im 1. Kapitel erwähnt, führt die Benutzung des EXTENDED BASIC-Moduls zu einem Verlust der Zeichensätze 15 und 16 (Zeichencodes 144 - 159), weil der Computer deren Speicherplatz benötigt, um die als Sprites (Kobolde) bezeichneten bewegten Grafiksymbole zu steuern. Auch die gesamte Speicherkapazität des Computers wird bei Benutzung dieses Moduls um genau 864 Bytes gegenüber dem TI BASIC eingeschränkt. Um so viel kürzer müssen also Programme im EXTENDED BASIC im Vergleich zum TI BASIC sein, wenn man die volle Speicherkapazität nutzen will.

Dieser Verlust dürfte bei einer freien Speicherkapazität von weit über 11000 Bytes (bei Verwendung der Speichererweiterung MEMORY EXPANSION UNIT um fast 24500 Bytes) im Normalfall durchaus zu verschmerzen sein, zumal durch die Mehrfachbesetzung einer Programmzeile mit Befehlen (Multiple Statement Lines, s.u.) wieder Speicherplatz gespart werden kann. Wenn das allerdings nicht hilft, dann sollten die verwendeten Variablennamen und die REM-Befehle auf mögliche Kürzungen überprüft werden.

Unter Berücksichtigung dieses eingeschränkten Speicherplatzes laufen in TI BASIC geschriebene Programme im Normalfall auch im EXTENDED BASIC. Schwierigkeiten können sich dann ergeben, wenn Sie in einem in TI BASIC geschriebenen Programm als Variablenbezeichnung einen Namen gewählt haben, der im EXTENDED BASIC zu den reservierten Wörtern gehört, was aber trotz der im EXTENDED BASIC nicht unerheblich größeren Liste von reservierten Wörtern nicht sehr wahrscheinlich ist, da es sich bei den reservierten Wörtern um amerikanische Bezeichnungen handelt und Sie sicher im Interesse einer besseren Programmübersicht deutsche Namen als Variablenbezeichnungen gewählt haben.

Umgekehrt laufen im EXTENDED BASIC geschriebene Programme im TI BASIC normalerweise nicht, denn TI BASIC kann die nachfolgend beschriebenen erweiterten Möglichkeiten des EXTENDED BASIC nicht verarbeiten, von denen Sie in Ihren Programmen sicher Gebrauch gemacht haben, denn warum sonst sollten Sie den EXTENDED BASIC MODUL verwenden?

### VIELE VORTEILE

Zunächst mögen die hier vorgestellten Vorteile etwas theoretisch anmuten, doch bei ihrer Berücksichtigung in Programmen zeigt sich sehr schnell, wie leistungsfähig das EXTENDED BASIC ist und wie elegant sich damit arbeiten läßt. Vorteile ergeben sich insbesondere für:

Eingabe, Ausgabe (Input, Output)

ACCEPT, DISPLAY (USING), PRINT (USING), IMAGE

Der ACCEPT-Befehl erlaubt die Dateneingabe an jeder beliebigen Bildschirmstelle, wobei dieser vorher gelöscht oder die Art und Größe der Daten eingeschränkt werden kann. Auf die gleiche Weise können durch DISPLAY Daten an jeder beliebigen Bildschirmstelle angezeigt werden. Durch IMAGE und USING, wobei letzteres auch in Verbindung mit PRINT verwendet werden kann, ist es möglich, Daten zu formatieren, d.h. ihre äußere Form zu gestalten.

## Möglichkeiten

### Unterprogramme (Subprograms)

SUB, SUBEND, SUBEXIT, MERGE

Unterprogramme werden durch SUB und den Unterprogrammnamen (gegebenenfalls mit einer zusätzlichen Variablen-Liste) eingeleitet und mit SUBEND bzw. SUBEXIT abgeschlossen. In Unterprogrammen auftretende Variablenbezeichnungen können mit denen im Hauptprogramm oder anderen Unterprogrammen identisch sein, ohne daß sich die jeweils zugewiesenen Werte gegenseitig beeinflussen. Mit Hilfe des neu hinzugekommenen MERGE-Befehls und des auf die neuen Verhältnisse hin abgestimmten SAVE-Befehls lassen sich auf Diskette gespeicherte Programme mit anderen Programmen kombinieren.

### Grafik (Sprites)

COINC, CHARPAT, CHARSET, DELSPRITE, DISTANCE, LOCATE, MAGNIFY, MOTION, PATTERN, POSITION, SPRITE

Diese mit CALL aufzurufenden Unterprogramme sind hinzugekommen, um die als Sprites (Kobolde) bezeichneten Grafiksymbole zu befähigen, ihre Bahnen auf dem Bildschirm zu ziehen. Außerdem wurden die Unterprogramme COLOR und CHAR auf die neuen Möglichkeiten eingerichtet.

### Funktionen (Functions)

MAX, MIN, PI

Neu hinzugekommen sind die Funktionen MAX und MIN (die größere bzw. kleinere zweier Zahlen) und PI mit dem Zahlenwert 3.14159265359.

### Dimensionierung (Dimensions)

DIM

Datenfelder (Arrays) können jetzt bis zu 7 Dimensionen im Vergleich zu höchstens 3 bei TI BASIC enthalten.

### Zeichenketten (Strings)

RPT$

Mit dieser Funktion kann ein String mehrmals wiederholt werden.

### Fehlerbehandlung (Error Handling)

ON WARNING, ON ERROR, ON BREAK, ERR

Im TI BASIC erhält man bei einem geringfügigen Fehler eine Warnmeldung (Warning), während bei einem schwerwiegenden Fehler (Error) und bei einem BREAK-Befehl der Programmablauf unterbrochen wird. Im EXTENDED BASIC können Sie mit Hilfe der neuen, den obigen Situationen entsprechenden Befehlen ON WARNING, ON ERROR und ON BREAK bestimmen, wie das Programm in einer solchen Situation weiterlaufen soll. Auch der RETURN-Befehl wurde diesen neuen Verhältnissen angepaßt. Mit dem ERR-Unterprogramm können Sie die Art des auftretenden Fehlers bestimmen.

## Möglichkeiten

### Programmausführung (Program Execution)

RUN (als Programmbefehl)

Mit RUN als Direktbefehl wird die Ausführung eines im Arbeitsspeicher des Computers befindlichen Programms gestartet. Mit RUN als Programmbefehl läßt sich ein auf einer Diskette befindliches Programm während des Ablaufs eines schon im Computer befindlichen Programms laden und starten. Wenn man sich die jeweils vom Programm aus gestarteten Programme als Teil eines größeren Programms vorstellt, ist es also möglich, Programme von unbegrenzter Größe zu schreiben, wobei die einzelnen Programmteile jeweils den nächsten laden und starten.

Unter bestimmten Voraussetzungen ist auch ein Programm-Schnellstart (Powerup) möglich.

### Mehrfachbelegung von Programmzeilen (Multiple Statement Lines)

Im EXTENDED BASIC können sich mehrere Programmbefehle - getrennt durch das Symbol "::" - in einer Programmzeile befinden. Dadurch wird die Programmausführung beschleunigt, Speicherplatz gespart und das Programmlisting übersichtlicher gestaltet, indem logische Einheiten, z.B. FOR-NEXT-Schleifen in einer Programmzeile untergebracht werden können.

### Programmschutz (Save and List Protection)

SAVE ...., PROTECTED

Durch Eingabe des Schlüsselworts PROTECTED lassen sich Programme gegen unerwünschtes Kopieren und Auflisten schützen, wodurch auch fremde Programmänderungen unmöglich werden. In Verbindung mit dem Kopieschutzverfahren des DISK MANAGER Moduls läßt sich ein vollständiger Programmschutz erreichen.

### Vergleichs-Folgen (Consequences of Comparison)

IF ... THEN (Befehl) ELSE (Befehl)

Die aus Vergleichen resultierenden Folgen sind im Gegensatz zum TI BASIC nicht mehr allein auf Sprungbefehle in andere Programmzeilen begrenzt, sondern können nun auch ganze Befehle darstellen, wie Z.B.
IF X<4 THEN X=X+1 ELSE STOP

### Mehrfach-Wertzuweisungen (Multiple Assignments)

Im EXTENDED BASIC kann ein Wert gleichzeitig mehreren Variablen zugewiesen werden (z.B. A,B=5), wodurch Programmbefehle eingespart werden.

### Kommentare (Comments)

Über den REM-Befehl hinaus können Kommentare - abgetrennt durch das Symbol "!" - auch an das Ende einer Programmzeile angefügt werden, womit eine genauere Programm-Dokumentation möglich wird.

## Möglichkeiten

### Assembler (Assembly Language Support)

INIT, LOAD, LINK, PEEK

Unterprogramme in Maschinensprache (Assembler) können mit Hilfe der neuen Befehle geladen und ausgeführt werden - allerdings nur beim TI 99/4A und auch nur dann, wenn zusätzlich die Speichererweiterung (Memory Expansion Unit) verfügbar ist, die ihrerseits den EXTENDED BASIC-Modul voraussetzt.

### Information

SIZE, VERSION, CHARPAT

Die Computer-Informationen für den Benutzer sind erweitert worden. Durch den SIZE-Befehl läßt sich jederzeit der noch verfügbare Speicherplatz (mit oder ohne Speichererweiterung) feststellen. Das VERSION-Unterprogramm zeigt die augenblickliche BASIC-Version an und das CHARPAT-Unterprogramm stellt einen Zeichenstring dar, der seinerseits das Aussehen eines Zeichens definiert.

# Kapitel 3

## E L E M E N T E

## Elemente

Selbstverständlich ist das EXTENDED BASIC nicht auf die im vorhergehenden Kapitel angedeuteten Strukturelemente beschränkt, sondern umfaßt alle auch im TI BASIC existierenden, sprachgestaltenden Bestandteile und Möglichkeiten. Da dieses Handbuch nach Möglichkeit als umfassende Informationsquelle für den Benutzer gestaltet werden soll, ist es nur folgerichtig, die wesentlichen und immer wiederkehrenden Strukturen, die TI BASIC und EXTENDED BASIC gestalten, an dieser Stelle vorzustellen. Dabei soll diese "Vorstellung" keine Vorwegnahme der alphabetisch geordneten Befehlsliste (Teil B) darstellen, sondern durch ihre - übergeordneten Gesichtspunkten folgende - Gliederung helfen, die einzelnen sprachgestaltenden Elemente in ihrer Wirkungsweise besser zu durchschauen.

Inwieweit dabei eine Unterscheidung zwischen COMMANDS (Direktbefehlen) und STATEMENTS (Anweisungen) sinnvoll ist, mag dahingestellt bleiben, denn beides sind im weitesten Sinne Befehle, was auch dadurch ersichtlich wird, daß viele STATEMENTS als COMMANDS und umgekehrt Verwendung finden. Als COMMAND ist ein Befehl dann zu verstehen, wenn er den Computer anweist, diesen Befehl d i r e k t (= sofort) auszuführen, wohingegen STATEMENTS w ä h r e n d eines Programmablaufs (indirekt) ausgeführt werden.

Damit läßt sich natürlich mit einem gewissen Recht die Gruppe von Befehlen, die vorzugsweise dazu benutzt werden, den Computer zur sofortigen Ausführung zu veranlassen, die Direktbefehle (Commands) also, von den übrigen Befehlen abgrenzen.

### DIREKTBEFEHLE

| | |
|---|---|
| NEW | löscht ein im Computer befindliches Programm und macht ihn für ein neues Programm aufnahmebereit (new = neu). |
| NUM[BER] | numeriert die Programmzeilen bei der Programmeingabe automatisch (number = Nummer). |
| RES[EQUENCE] | numeriert die Programmzeilen nach Vorgabe neu (resequence = Neu-Anordnung). |
| LIST | listet ein im Computer befindliches Programm auf (list = liste auf!). |
| RUN | startet die Programmausführung (run = führe aus!). |
| SAVE | speichert Programme auf Diskette oder Cassette ab (save = bewahre auf!). |
| OLD | lädt gespeicherte Programme wieder in den Computer (old = alt). |
| MERGE | kombiniert bei Diskettenbetrieb mit der MERGE-Option gespeicherte Programme mit bereits im Computer befindlichen (merge = verschmelze!). |
| DELETE | löscht eine auf einer Diskette befindliche Datei oder ein Programm (delete = lösche!). |
| SIZE | gibt Auskunft über den noch vorhandenen Speicherplatz im Computer (size = Größe). |
| BYE | löscht das im Computer befindliche Programm und bewirkt das Verlassen von EXTENDED BASIC, offene Dateien werden dabei geschlossen (bye = tschüs!). |

## Elemente

(UN)TRACE
(UN)BREAK
CON[TINUE]

sind Befehle, die bei der Fehlersuche eine Rolle spielen und unter diesem Stichwort erläutert werden.

Bei der verbleibenden umfangreichen Gruppe von Befehlen wird der Übersichtlichkeit halber nach Anwendungsbereichen und nicht nach formalen Gesichtspunkten unterschieden, was z.B. zu einer Abtrennung der in den Computer eingebauten Unterprogramme geführt hätte.

Die Arbeit mit dem Computer folgt normalerweise einem "CALL AND RESPONSE"-Schema, stellt also eine Art Frage- und Antwortspiel zwischen Computer und Benutzer dar, das mit Hilfe von Befehlen abläuft, die der Eingabe und Ausgabe von Daten während des Programmablaufs dienen.

### EINGABE

INPUT — erlaubt die Eingabe von Daten am unteren Bildschirmrand mit oder ohne Eingabe-Dialog (input-prompt). Wird vom Computer eine Dateneingabe ohne Eingabe-Dialog erwartet, so zeigt er das durch ein Fragezeichen an. Sind mehrere Werte einzugeben, so sind diese durch Kommata zu trennen. String-Werte müssen bei der Eingabe dann in Anführungszeichen gesetzt werden, wenn sie ein Komma, ein einleitendes Anführungszeichen oder voran- bzw. nachgestellte Leerzeichen enthalten. INPUT kann für die Dateneingabe in Dateien verwendet werden. Eine Überprüfung der eingegebenen Daten auf formale Richtigkeit erfolgt nicht (input = Eingabe).

LINPUT — erlaubt die Zuweisung einer ganzen Zeile an eine String-Variable. Die bei INPUT erwähnte Einschränkung für das Setzen von Anführungszeichen gilt nicht (linput = line + input, line = Zeile).

ACCEPT — erlaubt die Dateneingabe an fast allen Bildschirmpositionen, das bei INPUT auftretende "Rollen" (scrolling) der Bildschirmzeilen nach oben entfällt. ACCEPT kann über die Option VALIDATE eine Überprüfung der eingegebenen Daten durchführen (accept = nimm an!).

Zwei in den Computer eingebaute Unterprogramme dienen ebenfalls der Dateneingabe, und zwar erfolgt diese dabei direkt, d.h. ohne Betätigung der <ENTER>-Taste. Beide Unterprogramme stellen die eingegebenen Daten nicht auf dem Bildschirm dar.

CALL JOYST — erlaubt die Dateneingabe über die als Zubehör erhältliche Fernbedienung (joystick = Steuerknüppel).

CALL KEY — erlaubt die Dateneingabe direkt von der Konsolentastatur, wobei jeweils nur eine einzelne Taste betätigt werden kann, z.B. <J> oder <N> bei Ja/Nein-Entscheidungen. Das Unterprogramm fragt die Tastatur oder einen Teil davon ab, um die betätigte Taste festzustellen. Der eingegebene Wert ist der Code des Zeichens der betätigten Taste und nicht das Zeichen selbst (key = Taste).

## Elemente

### AUSGABE

Die entsprechende Umkehrung der vorher erwähnten Befehle findet sich in den Ausgabe(Output)-Befehlen, deren Aufgabe normalerweise die Darstellung von Zeichen auf Bildschirm oder Drucker, aber auch ihr Abspeichern auf externe Speicher, wie Cassette oder Diskette, ist. Aber auch Farbveränderungen an Zeichen oder Bildschirm oder das Erzeugen von Geräuschen und Tönen stellt im eigentlichen Sinne einen "Output" dar.

PRINT — ist die Umkehrung von INPUT: Daten werden am unteren Bildschirmrand ausgegeben, der Bildschirm "rollt".Die Position kann durch die TAB-Funktion, ihre äußere Gestalt durch den IMAGE-Befehl im Zusammenhang mit USING beeinflußt werden. Bei Verwendung des PRINT USING-Befehls ist die TAB-Funktion nicht möglich. Nur mit dem PRINT- oder PRINT USING-Befehl können Daten auf externe Speicher ausgegeben werden (print = drucke!).

DISPLAY — ist die Umkehrung von ACCEPT: Daten können an fast allen Bildschirmpositionen ausgegeben werden, das "Rollen" entfällt. Der ACCEPT-Befehl ist in seiner Anwendung auf den Bildschirm beschränkt, eine Verbindung mit USING ist möglich. Darüberhinaus können mit diesem Befehl der ganze Bildschirm oder einzelne Zeilen(teilé) gelöscht und ein Signalton erzeugt werden (display = stelle dar!).

Mehrere in den Computer eingebaute Unterprogramme dienen der Grafik im weitesten Sinne, wobei wir unterscheiden wollen zwischen der Darstellung von Zeichen (characters) und der hochauflösenden Darstellung von nahezu übergangslos beweglichen Grafik-Symbolen (sprites). Das Strukturmuster (die äußere Form oder Gestalt) von Zeichen wird, wenn man nicht die voraus definierten Zeichen (Codes 30 - 127) verwenden will, ebenso wie das von Sprites mit Hilfe der sogenannten Mustercodierung (vgl. Teil C) bestimmt, wobei 8 mal 8 Einzelpunkte eines Zeichens festgelegt werden.

CALL CHAR — definiert Muster für Zeichen und Sprites (char = Zeichen).

CALL CHARSET — setzt Zeichen mit den Codes 32 bis 95 in ihre ursprüngliche Form zurück, falls diese neu definiert wurden (charset = character + set, setzen).

CALL HCHAR — plaziert ein voraus- oder selbstdefiniertes Zeichen an eine von 768 Bildschirmpositionen (24 Zeilen und 32 Spalten) und wiederholt es, wenn gewünscht, in Zeilenrichtung, also horizontal (hchar = horizontal + character, waagerechtes Zeichen).

CALL VCHAR — wirkt wie HCHAR, die Wiederholung des Zeichens geschieht jedoch, wenn gewünscht, in Spaltenrichtung, also vertikal (vchar = vertical + character, senkrechtes Zeichen).

Sprites werden, wie erwähnt, hinsichtlich ihres Aussehens (Musters) wie die selbstdefinierten Zeichen mit CALL CHAR gebildet. Auf den Bildschirm plaziert und bewegt werden sie jedoch durch CALL SPRITE und andere Unterprogramme, wobei ihr Vorteil gegenüber den oben erwähnten Zeichen in der Bewegung auf dem Bildschirm ist. Bei diesen Grafik-Symbolen sind nicht nur 768, sondern 49152 Positionen (192 Zeilen und 256 Spalten) ansprechbar, wodurch sich eine ruckfreie Bewegung auf dem Bildschirm ergibt.

## Elemente

CALL SPRITE — definiert Sprites bezüglich ihres Aussehens, das mit CALL CHAR bestimmt wurde, ihrer Farbe, ihrer Bildschirmposition und ihrer Bewegung (sprite = Kobold).

CALL MAGNIFY — kontrolliert die Auflösung und die Größe der Sprites (magnify = vergrößere!).

CALL LOCATE — bewegt Sprites in eine neue Richtung (locate = mache neue Lage ausfindig!).

CALL PATTERN — ändert das Zeichen, das ein Sprite in seinem Aussehen bestimmt (pattern = Muster).

CALL MOTION — ändert die Bewegung eines Sprites (motion = Bewegung).

CALL COINC
CALL DISTANCE
CALL POSITION — informieren während eines Programmablaufs über Sprites. Ihrem Charakter nach stellen sie aber Wertzuweisungen dar und werden daher im folgenden Abschnitt kurz vorgestellt.

CALL DELSPRITE — löscht Sprites (delsprite = delete + sprite, löschen).

Vor allem für Zeichen und Sprites, aber auch für den durch den Bildschirmhintergrund wesentlich beeinflußten Gesamteindruck spielt die Farbgebung eine große Rolle.

CALL SCREEN — bestimmt die Bildschirmfarbe (screen = Bildschirm).

CALL COLOR — bestimmt die Farbe eines Zeichens und seines Hintergrunds, sowie die Farbe eines Sprites (color = Farbe).

Neben der Farbgebung kann die Erzeugung von Tönen und Geräuschen während eines Programmablaufs von großem Interesse und auch der Auflockerung von Programmen dienlich sein.

CALL SOUND — erzeugt Töne, die in ihrer Klangfarbe, Lautstärke und Dauer beeinflußbar sind, sowie 8 verschiedene Geräusche. Bis zu drei Töne und ein Geräusch können gleichzeitig erzeugt werden (sound = Klang).

All dies kann man vergessen bei der Eingabe des Befehls CALL CLEAR, denn damit wird der gesamte Bildschirm gelöscht. Das ist sicher dienlich zu Beginn eines Programmablaufs, nicht aber an dieser Stelle, denn noch harren weitere Befehle ihrer Vorstellung.

CALL CLEAR — löscht den gesamten Bildschirm (clear = mache frei!).

### WERTZUWEISUNGEN

Alle im folgenden vorgestellte Befehle sind dadurch gekennzeichnet, daß sie Variablen einen oder mehrere Werte zuweisen.

[LET] — braucht nicht geschrieben zu werden und wird bei der Zuweisung von Werten oder Ausdrücken an Variable verwendet (let = lasse!).

READ — wird in Verbindung mit DATA und RESTORE bei der Zuweisung von Werten aus einer programm-internen Datei verwendet restore = setze wieder ein!).

## Elemente

Unter dem Stichwort "Grafik" sind folgende, auf Wertzuweisung beruhende Unterprogramme zu erwähnen, die dem Benutzer Auskunft über Zeichen oder Sprites erteilen:

CALL CHARPAT — gibt als Stringwert den Mustercode aus, durch den ein Zeichen in seinem Aussehen definiert ist (charpat = character + pattern, Zeichenmuster).

CALL GCHAR — gibt den ASCII-Code des Zeichens, das sich an einer beliebigen Bildschirmposition (bestimmt durch Zeile und Spalte) befindet, an (gchar = get + character, get = beschaffe!).

CALL COINC — gibt an, ob zwei Sprites oder ein Sprite und ein bestimmter Bildschirmpunkt zusammentreffen (coincidence = Zusammentreffen).

CALL DISTANCE — gibt die Entfernung zwischen zwei Sprites oder einem Sprite und einer bestimmten Bildschirmposition an (distance = Entfernung).

CALL POSITION — gibt die Bildschirmposition an, an der sich ein Sprite befindet (position = Ort).

Auch auf Wertzuweisung beruht das Unterprogramm CALL ERR, das unter dem Stichwort "Fehlerbehandlung" vorgestellt wird, und:

CALL VERSION — gibt die Art des BASIC an, das gerade verwendet wird (version = Art).

### PROGRAMMSCHLEIFEN

Unter Schleifen versteht man eine Gruppe von Programmzeilen, die bei der Programmausführung wiederholt durchlaufen werden. Der Schleifendurchlauf ist grundsätzlich durch Wertzuweisungen bedingt. Man unterscheidet sogenannte Zählschleifen und Unterprogrammschleifen:

1. Zählschleifen:

FOR-TO-[STEP] — Beginn einer Zählschleife, bei der einer Zählvariablen ein Wert zugewiesen wird. Mit dieser Variablen kann kontrolliert werden, wie oft die Schleife durchlaufen wird. Der Wert der Variablen erhöht sich dabei stets um 1, es sei denn, mit STEP wird eine andere Schrittweite vorgegeben (for = für, to = bis, step = Schritt).

NEXT — muß eine Zählschleife abschließen. An dieser Stelle wird der Variablenwert erhöht, und der Durchlauf der Schleife beginnt erneut, bis der durch TO bestimmte Wert erreicht ist. Danach fährt das Programm mit dem nächsten Befehl fort (next = nächster).

2. Unterprogrammschleifen:

Sollen während eines Programmablaufs beispielsweise für mehrere Variable die gleichen Berechnungen durchgeführt werden, dann wäre es unsinnig, die Befehle dafür für jede Variable gesondert im Programm aufzuführen. Die bessere Lösung für solche Fälle sind sogenannte Unterprogrammschleifen (subroutines) oder auch Unterprogramme (subprograms, vgl. auch den folgenden Abschnitt). Auch Unterprogrammschleifen werden mehrmals in einem Programmablauf verwendet ("angesprungen"). Mit ihnen erspart man sich eben-

falls Programmzeilen-Wiederholungen. Der Unterschied zu den Zählschleifen liegt darin, daß sie nicht mit Variablenwerten einer festen Schrittweite nacheinander durchlaufen werden und von beliebigen Programmstellen aus angesprochen werden können.

| | |
|---|---|
| [ON-]GOSUB | ruft eine Unterprogrammschleife auf. Bei Verwendung von ON wird die Unterprogrammschleife bei bestimmten Variablenwerten ausgeführt (on = bei, gosub = gehe hinunter!). |
| RETURN | beendet die Unterprogrammschleife und läßt das Programm mit dem dem GOSUB-Befehl folgenden Befehl fortfahren (return = kehre zurück!). |

Eine ähnliche Verfahrensweise wird bei den Unterprogrammen verwendet:

UNTERPROGRAMMTECHNIK

Dabei wollen wir uns nicht mit den im Computer eingebauten, d.h. bereits vorhandenen Unterprogrammen beschäftigen, von denen die meisten schon erwähnt wurden, sondern mit solchen, die der Benutzer selbst schreibt, und zwar normalerweise wohl aus den Gründen, die schon unter dem Stichwort "Schleifen" angedeutet wurden.

Unterprogramme sind nicht Teil des Hauptprogramms und können dessen Variablennamen unabhängig benutzen. Sie können ihrerseits andere Unterprogramme - nicht aber sich selbst - aufrufen und müssen immer nach dem Hauptprogramm stehen. Das bedeutet, daß ihre Programmzeilennummern immer größer als die größte Programmzeilennummer des Hauptprogramms sein. Unterprogramme werden im Hauptprogramm mit dem Befehl "CALL Unterprogrammname" aufgerufen.

| | |
|---|---|
| SUB Name | leitet ein Unterprogramm ein (sub = unten). |
| SUBEXIT | ermöglicht das vorzeitige Verlassen eines Unterprogramms (sub + exit = Ausgang). |
| SUBEND | beendet ein Unterprogramm (sub + end = Ende). |

SONSTIGE BEFEHLE

Eine Reihe von Befehlen läßt sich nur schwer den vorangehenden Stichwörtern unterordnen; diese sollen aber im Rahmen dieser "Vorstellung" nicht unerwähnt bleiben, wobei allerdings auf eine Erörterung der als sehr speziell erachteten Befehle im Zusammenhang mit dem Sprachsynthesizer und dem Assembler verzichtet wird.

| | |
|---|---|
| [ON-]GOTO | bewirkt einen uneingeshränkten Sprung in eine andere Programmzeile, wobei die Verwendung von ON eine Auswahl ermöglicht. ON-GOTO ist nicht für Unterprogramme geeignet (goto = gehe zu!). |
| IF-THEN[-ELSE] | dient ebenfalls der Programmverzweigung nach Vergleichsausdrücken (if = falls, then = dann, else = sonst). |
| RANDOMIZE | ermöglicht eine unvorhersagbare Folge von Zufallszahlen (randomize = erzeuge eine Zufallszahl!). |
| DIM | reserviert im Computerspeicher Platz für numerische oder String-Datenfelder (dimension = Dimension). |

## Elemente

OPTION BASE — setzt den kleinsten Index eines Datenfeldes (array) auf 0 oder 1 (option base = Wahlbasis). Der Befehl darf in einem Programm nur einmal auftreten und muß die niedrigste Zeilennummer aller Datenfelder betreffenden Programmzeilen haben.

OPEN
CLOSE — werden bei der Verarbeitung externer Dateien benötigt (open = öffne!, close = schließe!).

REM — erlaubt die Kommentierung von Programmen. REM-Befehle benötigen zwar Speicherplatz, werden aber bei der Programmausführung nicht berücksichtigt (remark = Bemerkung).

### FUNKTIONEN

Viele Funktionen sind mathematischer Natur (vgl. auch Anhang). Funktionen sind Unterprogramme, die bei ihrem Aufruf genau einen numerischen Wert an das Hauptprogramm übergeben. Dadurch, daß sie nicht mehr als einen numerischen Wert übergeben können und nicht am Ende eines Programms stehen müssen, unterscheiden sie sich von den anderen Unterprogrammen (vgl. auch Abschnitt "Unterprogrammtechnik"). Auch ist ihr Aufruf anders als bei den Unterprogrammen, sie werden nämlich nur mit ihrem Funktionsnamen (ohne CALL) und einer möglichen Variablenliste aufgerufen. Viele Funktionen sind bereits im Computer eingebaut (vgl. unten), man kann aber auch selbst Funktionen definieren:

DEF — definiert eigene Funktionen in Zeilenlänge. Längere Funktionen müssen unter Bezug auf vorangehende zusätzlich definiert werden. Bei sehr umfangreichen Funktionen kann die Verwendung eines Unterprogramms oder einer Unteprogrammschleife von Vorteil sein (definition = Definition, Festlegung).

Folgende Funktionen sind im Computer bereits eingebaut:

ABS Absolutwert eines numerischen Ausdrucks
ASC numerischer ASCII-Zeichencode des ersten Zeichens eines Strings
ATN Arcustangens eines numerischen Ausdrucks im Bogenmaß
CHR$ Zeichen, das dem jeweiligen ASCII-Code entspricht
COS Cosinus eines numerischen Ausdrucks im Bogenmaß
EOF überprüfung einer Datei auf ihr Ende
EXP Exponentialfunktion eines numerischen Ausrucks
INT ganzzahliger Anteil eines numerischen Ausdrucks
LEN Länge (Anzahl der Zeichen) eines String-Ausdrucks
LOG natürlicher Logarithmus eines numerischen Ausdrucks
MAX der größere Wert zweier numerischer Ausdrücke
MIN der kleinere Wert zweier numerischer Ausdrücke
PI der Wert 3.141592654
POS die erste Stelle des Auftretens eines Teilstrings in einem String
REC die Position eines Datensatzes in einer externen Datei
RND Zufallszahl zwischen 0 und 1
RPT$ Wiederholung eines String-Ausdrucks
SEG$ Teilstring von einem String-Ausdruck
SGN Vorzeichen eines numerischen Ausdrucks
SIN Sinus eines numerischen Ausdrucks im Bogenmaß
SQR positive Quadratwurzel eines numerischen Ausdrucks
STR$ Umformung eines numerischen Ausdrucks in einen String
TAB Position für den Ausdruck eines Wertes
TAN Tangens eines numerischen Ausdrucks im Bogenmaß
VAL Umformung eines Strings in eine numerische Konstante

## FEHLERSUCHE

Das Fatale an Programmierfehlern, seien es nun Schreibfehler oder logische Fehler, ist, daß man sie meist dann bemerkt, wenn man sich nach getaner mühevoller Programmierarbeit auf den ersten Programmablauf freut. Aus scheinbar völlig unerfindlichen Gründen geschehen dann häufig merkwürdige Dinge, wenn man nicht einen meist leicht zu behebenden Eingabefehler gemacht hat: der Bildschirm ändert seine Farbe (aus Ärger?) in kornblumenblau, mit den Zeichen gehen merkwürdige Veränderungen vor, und Sprites werden gelöscht - kurz gesagt, die Freude über das geschriebene Programm ist erst einmal verdorben. Aber ein Grund zum Verzweifeln ist das sicher nicht, denn wenn man nicht in die 2. Programmzeile GOTO 1000 geschrieben hat und in Programmzeile 1000 END steht, dann wird der Computer im Normalfall eine Fehlermeldung (vgl. auch Anhang) ausgeben, die, insbesondere wenn sie im Zusammenhang mit einer Zeilennummer steht, es ermöglichen sollte, den Fehler vergleichsweise schnell zu beheben. Und wenn sich die Fehlersuche einmal schwieriger gestalten sollte, dann kann sie doch durch eine Reihe von Befehlen erleichtert werden:

BREAK — unterbricht die Programmausführung, so daß die Variablen-Werte ausgegeben oder geändert werden können. Die Zeichen werden auf ihre Standardfarben (schwarz auf transparent) und der Bildschirm auf kornblumenblau zurückgesetzt. Die vordefinierten Zeichen erhalten ihre ursprüngliche Form, Sprites werden gelöscht (break = Unterbrechung).

ON BREAK — weist den Computer an, was bei einer Unterbrechung zu tun ist, z.B. diese Unterbrechung zu ignorieren und mit dem Programm fortzufahren: ON BREAK CONTINUE. Dann kann der Computer allerdings nicht mehr mit <FCTN-4> (CLEAR) angehalten werden.

CON[TINUE] — setzt die Programmausführung nach einer Unterbrechung wieder fort (continue = fahre fort!).

UNBREAK — hebt durch BREAK gesetzte Unterbrechungen wieder auf (unbreak = Aufhebung der Unterbrechung).

TRACE — veranlaßt den Computer, vor der Ausführung einer Programmzeile deren Zeilennummer auszugeben. Damit kann die Abfolge der einzelnen Programmschritte genau beobachtet werden (trace = spüre auf!).

UNTRACE — beendet die durch TRACE bewirkte Ausgabe der Zeilennummern (untrace = beende das Aufspüren!).

## FEHLERBEHANDLUNG

EXTENDED BASIC sieht für Fehlersuche und Fehlerbehandlung weitere - und auch elegante - Möglichkeiten vor: In einem Programm können Befehle enthalten sein, die dem Computer Anweisungen für den Fall erteilen, daß ein Fehler auftritt. Dazu kann zumindest für den Unterprogrammaufruf CALL ERR die im Anhang angegebene Fehlerliste verwendet werden:

CALL ERR — gibt an, wo ein Fehler aufgetreten ist und um was für eine Art Fehler es sich dabei handelt (error = Fehler).

ON ERROR — bestimmt, was der Computer beim Auftreten eines Fehlers machen soll (on = bei).

## Elemente

| | |
|---|---|
| ON WARNING | bestimmt, was der Computer beim Auftreten eines sogenannten leichten Fehlers, der normalerweise zur Ausgabe einer Warnmeldung (warning) - nicht aber zur Programmunterbrechung - führen würde, machen soll. |
| RETURN | kann in Ergänzung zu seiner Anwendung mit GOSUB auch in Verbindung mit ON ERROR verwendet werden: es wiederholt den Befehl, der zum Fehler führte und geht dann zum folgenden Befehl oder einem Programmteil über, der den aufgetretenen Fehler vermeidet. |

Nachdem Sie nun mit allen, die Programmiersprache EXTENDED BASIC gestaltenden Elementen vertraut sind, ist es nur noch ein ganz kleiner Schritt zum ersten Programm. Während Sie die im nächsten Kapitel beschriebenen "Arbeitsweisen" und "Spezielle Tastenfunktionen" in jedem Fall noch studieren sollten, kann eine Erarbeitung der unter "Programm-Abtastung" und "Speicherprobleme" beschriebenen Besonderheiten auch zu einem späteren Zeitpunkt erfolgen.

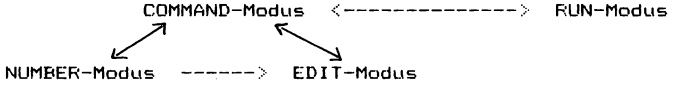
# Kapitel 4

## B E D I E N U N G

## Bedienung

BEFEHLSEBENEN: COMMAND, NUMBER, EDIT, RUN

Im EXTENDED BASIC gibt es verschiedene Befehlsebenen (Arbeitszustände), in denen sich der Computer befinden kann und die durch die verschiedenen Bildschirmfarben grün und blau angezeigt werden. Ist die Bildschirmfarbe blau, so wird im wesentlichen ein Programm eingegeben oder verändert (editiert), ist die Bildschirmfarbe grün, so läuft ein Programm automatisch ab, wobei sich natürlich die Bildschirmfarbe entsprechend den Befehlen im Programm ändern kann. Den letzteren Zustand nennen wir Programmablaufebene (RUN-Modus). Dies alles soll auch das folgende Diagramm etwas verdeutlichen:




BILDSCHIRMFARBE: b l a u g r ü n

Befindet sich der Computer nicht im RUN-Modus, so ist die Bildschirmfarbe automatisch kornblumenblau. In diesem Zustand gibt es verschiedene Befehlsebenen, in denen spezielle Tasten unterschiedliche Bedeutungen haben können: Da ist zunächst der Direktbefehls-(COMMAND-)Modus, in dem der Computer jeden Befehl direkt ausführt oder Programmzeilen übernimmt. Wird dabei der Direktbefehl NUMBER eingegeben, so werden die Programmzeilennummern automatisch erzeugt, und wir sprechen vom NUMBER-Modus. In diesem Modus kann unter Beachtung spezieller Tastenfunktionen auch editiert werden.

Der EDIT-Modus wird zur Ausgabe bereits existierender Programmzeilen zum Zwecke ihrer Änderung oder Kontrolle benötigt. Während dieser Modus im TI BASIC auch durch den von einer Zeilennummer gefolgten EDIT-Befehl erreicht werden kann und danach - als spezielle Funktionstaste - <FCTN-X> (TI 99/4: <SHIFT-X>) oder <FCTN-E> (TI 99/4: <SHIFT-E>) benötigt werden, ist dies im EXTENDED BASIC nur durch Eingabe einer Programmzeilennummer und der anschließenden Betätigung von <FCTN-X> oder <FCTN-E> möglich. Die im EDIT-Modus angesprochene Zeile wird - sofern vorhanden - auf dem Bildschirm ausgegeben und kann dann mit Hilfe spezieller Tastenfunktionen (s.d.) geändert, d.h. völlig oder teilweise überschrieben werden. Durch einfaches Betätigen der Tasten <ENTER>, <FCTN-X> oder <FCTN-E> kann eine solche Programmzeile auch unverändert im Programm verbleiben. Auch bei Auslösung der speziellen Tastenfunktion REDO (Wiederholung) bei Betätigung von <FCTN-8> (TI 99/4: <SHIFT-R>), durch die eine bereits eingegebene Programmzeile bzw. ein Befehl erneut auf den Bildschirm zurückgeholt wird, befindet sich der Computer im EDIT-Modus.

Wird im COMMAND-Modus unter der Voraussetzung, daß sich ein Programm im Computer befindet, der Direktbefehl RUN eingegeben, dann wird der RUN-Modus erreicht, d.h. das Programm wird ausgeführt (= "läuft ab"). Ein ablaufendes Programm kann durch die spezielle Tastenfunktion CLEAR bei Betätigung von <FCTN-4> (TI 99/4: <SHIFT-C>) unterbrochen werden, wobei die gleichen Folgen wie bei einprogrammierten Stoppstellen (breakpoints) eintreten. Wird während dieser Programmunterbrechung nicht editiert, dann kann der Programmablauf mit dem CONTINUE-Befehl wieder aufgenommen werden. Als Programmunterbrechung möglich, aber denkbar ungeeignet, ist die spezielle Tastenfunktion QUIT (endgültiges Ende), die durch <FCTN-=> (TI 99/4: <SHIFT-Q>) erreicht wird, weil damit nicht nur das Programm gelöscht wird, sondern auch Datenverluste eintreten können.

## SPEZIELLE TASTENFUNKTIONEN

Einige spezielle Tastenfunktionen wurden bereits im vorhergehenden Abschnitt im Zusammenhang mit den verschiedenen Befehlsebenen kurz beleuchtet, wobei auch angedeutet wurde, daß ein direkter Zusammenhang zwischen diesen Tastenfunktionen und der jeweilgen Befehlsebene (Modus) besteht. Aufgabe dieses Abschnitts ist es demnach, die einzelnen Tastenfunktionen in Abhängigkeit vom jeweils gewählten Modus zu erläutern, wobei zu beachten ist, daß alle speziellen Tastenfunktionen mit Ausnahme von ENTER durch gleichzeitiges Drücken der <FCTN>-Taste (TI 99/4: <SHIFT>) und einer jeweils zugehörigen Taste ausgelöst werden. Die jeweiligen Tasten werden durch die Symbole "<" und ">" mit der entsprechenden Tastenbezeichnung dargestellt, so bedeutet beispielsweise <FCTN-7> das gleichzeitige Betätigen der Tasten <FCTN> und <7>.

Die nachfolgende Übersicht enthält die Tastenfunktion, ihre Auslösung (für den TI 99/4 jeweils in Klammern angegeben) und ihre Wirkungsweise in den verschiedenen Befehlsebenen:

---

ENTER = <ENTER> — Übergabe

---

COMMAND: Der Inhalt eines Befehls oder einer vollständigen Programmzeile die bis zu 4 Bildschirmzeilen umfassen kann, wird an den Arbeitsspeicher des Computers übergeben.

NUMBER:
1) Nach einer schon erzeugten Programmzeilennummer wird der NUMBER-Modus aufgehoben, der Computer befindet sich wieder im COMMAND-Modus.
2) Nach einer vollständig eingegebenen Programmzeile wird diese an den Arbeitsspeicher des Computers übergeben und anschließend eine neue Zeilennummer ausgegeben.
3) Existiert die Programmzeile schon, so geschieht dasselbe wie unter 2).
4) Nach der speziellen Tastenfunktion ERASE bewirkt ENTER ebenfalls die Aufhebung des NUMBER-Modus, die Programmzeile wurde jedoch gelöscht.
5) Nach dem Editieren einer Programmzeile wird diese an den Arbeitsspeicher des Computers anstelle der ursprünglichen Programmzeile übergeben und eine neue Zeilennummer ausgegeben.

EDIT:
1) Nach dem Editieren einer Programmzeile wird diese an den Arbeitsspeicher des Computers übergeben, der Computer befindet sich anschließend im COMMAND-Modus.
2) Wird die Programmzeile durch ERASE gelöscht, so wird sie dann durch ENTER auch aus dem Arbeitsspeicher entfernt, und der Computer befindet sich im COMMAND-Modus.

RUN: Beendigung der Dateneingabe nach einem (L)INPUT- oder ACCEPT-Befehl und Fortsetzung des Programmablaufs

---

AID = <FCTN-7> (<SHIFT-A>) — Unterstützung

---

PROC'D = <FCTN-6> (<SHIFT-V>) — Fortsetzung

---

BEGIN = <FCTN-5> (<SHIFT-W>) — Beginn

---

BACK = <FCTN-9> (<SHIFT-Z>) — Rückkehr

---

Diese speziellen Tastenfunktionen werden im Zusammenhang mit einigen Software-Modulen verwendet und in deren Bedienungsanleitungen genauer erklärt.

## Bedienung

---

CLEAR = <FCTN-4> (<SHIFT-C>) — Unterbrechung

---

COMMAND: Eine eingegebene Zeile wird gelöscht.
NUMBER: Eine eingegebene Programmzeile wird nicht vom Computer übernommen und der NUMBER-Modus wird aufgehoben (Rückkehr zum COMMAND-Modus)
EDIT: Eine editierte Programmzeile wird nicht in den Arbeitsspeicher des Computers übernommen, die alte Programmzeile verbleibt also unverändert. Der Computer verläßt den EDIT-Modus und kehrt zum COMMAND-Modus zurück.
RUN: Das ablaufende Programm wird sofort unterbrochen. Die Stoppstelle wird durch ihre Zeilennummer angezeigt (BREAKPOINT IN ###). Das Programm kann, wenn nicht editiert wird, mit CONTINUE wieder fortgesetzt werden. Die Tasten <FCTN-4> sind solange zu betätigen, bis der Computer die Stoppstelle erkennt.

---

DELETE = <FCTN-1> (<SHIFT-F>) — Einzellöschung

---

COMMAND: DELETE erfolgt normalerweise nach Betätigung der Pfeiltasten (s. d.), mit deren Hilfe der Cursor (Positionszeiger) über ein bestimmtes Zeichen einer Programmzeile gebracht wurde. Durch DELETE wird dann das unter dem Positionszeiger befindliche Zeichen gelöscht, alle rechts vom Positionszeichen befindlichen Zeichen werden um eine Stelle nach links verschoben.
NUMBER: Wirkung wie im COMMAND-Modus.
EDIT: Wirkung wie im COMMAND-Modus.
RUN: Bei der Eingabe von Daten Wirkung wie im COMMAND-Modus.

---

INSERT = <FCTN-2> (<SHIFT-G>) — Einfügung

---

COMMAND: INSERT erfolgt normalerweise nach Betätigung der Pfeiltasten (s. d.), mit deren Hilfe der Cursor (Positionszeiger) über ein bestimmtes Zeichen einer Programmzeile grbracht wurde. An dieser Stelle lassen sich dann weitere Zeichen einfügen. Der Positionszeiger mit dem darunter befindlichen Zeichen und alle rechts vom Positionszeiger befindlichen Zeichen werden bei jedem eingefügten Zeichen jeweils um eine Stelle nach rechts verschoben. Das Einfügen läßt sich u.U. über die normale Zeilenlänge hinaus fortführen; dann kann jedoch der Computer eine solche überlange Zeile mit der Meldung LINE TOO LONG nicht mehr annehmen oder Zeichen am Zeilenende auslassen.
NUMBER: Wirkung wie im COMMAND-Modus.
EDIT: Wirkung wie im COMMAND-Modus.
RUN: Bei der Dateneingabe Wirkung wie im COMMAND-MODUS.

---

QUIT = <FCTN-=> (<SHIFT-Q>) — endgültiges Ende

---

COMMAND: Der Computer wird auf das Titelbild zurückgeschaltet, dabei werden gespeicherte Daten und das Programm gelöscht, offene Dateien werden nicht geschlossen. Verwenden Sie besser den BYE-Befehl!
NUMBER: Wirkung wie im COMMAND-Modus.
EDIT: Wirkung wie im COMMAND-Modus.
RUN: Wirkung wie im COMMAND-Modus.

## Bedienung

---

UNDO = <FCTN-8> (<SHIFT-R>) — Wiederholung

---

COMMAND: Der letzte vor dem Betätigen von <ENTER> eingegebene Befehl bzw. die letzte vorher eingegebene Programmzeile wird auf den Bildschirm zurückgeholt. Der Computer befindet sich dann im EDIT-Modus, und die Programmzeile kann - inklusive der Zeilennummer - geändert werden. Dadurch wird das Eingeben ähnlicher Programmzeilen umgangen, denn die ursprüngliche Programmzeile verbleibt im Arbeitsspeicher des Computers.
NUMBER: Der Computer geht in den EDIT-Modus über, sonst wie im COMMAND-Modus.
EDIT: Wirkung wie im COMMAND-Modus.
RUN: Keine Wirkung.

---

ERASE = <FCTN-3> (<SHIFT-T>) — Gesamtlöschung

---

COMMAND: Die gesamte eingegebene Programmzeile oder der eingegebene Befehl wird auf dem Bildschirm gelöscht und nicht in den Arbeitsspeicher des Computers übernommen. Der Positionszeiger kehrt an den linken Bildschirmrand zurück, und ein neuer Befehl oder eine neue Programmzeile kann eingegeben werden.
NUMBER: Die eingegebene Programmzeile - nicht aber die Zeilennummer - wird gelöscht. Der NUMBER-Modus bleibt erhalten.
EDIT: Wirkung wie im NUMBER-Modus.
RUN: Die über Tastatur eingegebenen Daten werden gelöscht.

---

ARROW LEFT = <FCTN-S> (<SHIFT-S>) — Pfeil links

---

COMMAND: Der Positionszeiger (Cursor) wird um eine Stelle nach links bewegt, wobei Zeichen, über die er hinweggeführt wird, unbeeinflußt bleiben. Die Bewegung endet am linken Bildschirmrand. Die Tastenfunktion wird normalerweise vor dem Ändern einer Zeile ausgeführt.
NUMBER: Wirkung wie im COMMAND-Modus.
EDIT: Wirkung wie im COMMAND-Modus.
RUN: Wirkung wie im COMMAND-Modus.

---

ARROW RIGHT = <FCTN-D> (<SHIFT-D>) — Pfeil rechts

---

Die Wirkung ist die gleiche wie bei ARROW LEFT, der Positionszeiger wird lediglich nach rechts bewegt. Die Bewegung endet mit dem Ende der Eingabezeile.

---

ARROW DOWN = <FCTN-X> (<SHIFT-X>) — Pfeil abwärts

---

COMMAND: 1) Gleiche Wirkung wie ENTER, d.h. die eingegebene Programmzeile wird an den Arbeitsspeicher des Computers übergeben.
2) Wird vor der Betätigung eine Zeilennummer eingegeben, so verläßt der Computer den COMMAND-Modus und geht in den EDIT-Modus über. Die angegebene Programmzeile wird dann auf dem Bildschirm ausgegeben, sofern sie schon im Arbeitsspeicher vorhanden ist.
NUMBER: Wirkung wie im COMMAND-Modus.
EDIT: Eine auf dem Bildschirm vorhandene Programmzeile wird in den Arbeitsspeicher der Computers übernommen (wie bei ENTER) und die nächsthöhere existierende Programmzeile wird auf dem Bildschirm

ausgegeben. Wenn keine Porgrammzeile mit einer höheren Zeilennummer im Arbeitsspeicher zu finden ist, geht der Computer in den COMMAND-Modus über. Für die Betätigung von <FCTN-X> ist die Stellung des Positionszeigers ohne Bedeutung.

RUN: Wirkung wie bei Betätigung von ENTER.

---

ARROW UP = <FCTN-E> (<SHIFT-E>) Pfeil aufwärts

---

Die Wirkung ist die gleiche wie bei ARROW DOWN, im EDIT-Modus wird lediglich die nächstniedrigere Programmzeile ausgegeben, und - wenn eine solche nicht vorhanden ist - verläßt der Computer denn EDIT-Modus, um in den COMMAND-Modus überzugehen.

Bei einigen dieser speziellen Tastenfunktionen wurde bereits auf ihre Verwendung im Zusammenhang mit Software-Paketen verwiesen. Die Bedienungsanleitungen für ältere Software-Pakete sind noch auf die speziellen Tastenfunktionen des TI 99/4 abgestimmt. Als TI 99/4A-Besitzer müssen Sie diese entsprechend der obigen Übersicht abändern.

## PROGRAMM-ABTASTUNG

Nach der Eingabe des Direktbefehls RUN zum Programmstart ist, bevor das Programm wirklich anläuft, eine Pause festzustellen, deren Dauer vom Umfang des zu startenden Programms direkt abhängt. Während dieser Pause tastet der Computer das Programm ab, um Speicherplatz für die Variablen, Datenfelder und Eingaben vorzusehen. Danach durchläuft er jede Anweisung, führt die entsprechenden Befehle aus und ordnet den Variablen Werte zu. Normalerweise ist diese Pause vertretbar kurz, bei umfangreichen Programmen kann sie jedoch so lange dauern, daß sich der Benutzer wünscht, der Computer würde nicht erst das ganze Programm abtasten.

Und genau das läßt sich im EXTENDED BASIC erreichen, und zwar durch die Abtast-Befehle !@P- und !@P+ , mit deren Hilfe die Anweisungen kontrolliert werden können, die nicht abgetastet werden sollen oder brauchen - und das sind mehr, als man zunächst glaubt! Denn da der Zweck der Programm-Abtastung im Einrichten von Speicherplatz für Variable liegt, brauchen nur diejenigen Anweisungen abgetastet zu werden, die zum ersten Mal auf eine Variable Bezug nehmen.

Die folgenden Befehle sollten durch die Programm-Abtastung erfaßt werden (!@P+), während sie für die übrigen Programmzeilen entfallen, d.h. ausgeschaltet werden kann (!@P-):

- der erste DATA-Befehl,
- der erste Befehl mit einer bestimmten Variablen,
- der erste Befehl mit einem bestimmten Datenfeld,
- der OPTION BASE-Befehl,
- alle DEF-Befehle für benutzerdefinierte Funktionen,
- der jeweils erste Bezug auf einen Unterprogramm-Aufruf (CALL),
- alle SUB- und SUBEND-Befehle.

Mit dem Einschluß der letztgenannten Befehle in die Programm-Abtastung ist sichergestellt, daß im Unterprogramm vorhandene Variable mit der gleichen Bezeichnung wie im Hauptprogramm ebenfalls von der Programm-Abtastung erfaßt werden.

## Bedienung

Wollen Sie die Abtast-Befehle zur Verkürzung der Zeit bis zum wirklichen Programmstart verwenden, dann stellen Sie zunächst sicher, daß Ihr Programm ohne Fehler läuft. Schalten Sie dann durch Einfügung entsprechender Programmzeilen, wie die nachfolgenden Beispiele zeigen, die Abtastung gemäß den o.a. Regeln ein (!@P+) oder aus (!@P-), wobei beim TI 99/4A auch der Kleinbuchstabe "p" eingegeben werden kann. Beachten Sie, daß das Ein- bzw. Ausschalten der Abtastung jeweils eine eigene Programmzeile beansprucht. Sollten Sie versehentlich erste Bezugnahmen auf Variablen (vgl.o.) nicht mit in die Programm-Abtastung einschließen, erhalten Sie die Fehlermeldung SYNTAX ERROR.

Betrachten Sie nun den folgenden Programmauszug:

```
100 CALL CLEAR
110 CALL CHAR(96,"FFFFFFFFFFFFFFFF")
120 CALL CHAR(42,"0F0F0F0F0F0F0F0F")
130 ...
140 ...
150 ...
160 CALL HCHAR(12,17,42)
170 CALL VCHAR(14,17,96)
180 VERZ=0
190 FOR VERZ=1 TO 1000
200 NEXT VERZ
210 DATA 3
220 ...
230 ...
```

Da die Programm-Abtastung zunächst eingeschaltet ist, sollte der DATA-Befehl der Zeile 210 an den Programmanfang gebracht werden, damit er von der Abtastung erfaßt wird:

```
10 DATA 3
```

Die Programmzeile 210 kann dann anderweitig besetzt oder gelöscht werden. Danach wird gemäß den o.a. Regeln die Abtastung mehrfach ein- bzw. ausgeschaltet:

```
125 !@P-
...
155 !@P+
...
185 !@P-
```

Dadurch wird der Programmstart schneller erfolgen. Man kann den Computer zudem veranlassen, Speicherplatz für CALL-Befehle und auf Variable bezogene Befehle zu reservieren, ohne daß diese Befehle auch wirklich durchgeführt werden, und zwar geschieht das mit Hilfe eines GOTO-Befehls, wodurch unser Programmausschnitt folgendes Aussehen erhält:

```
10 DATA 3
20 GOTO 100 :: VERZ :: CALL CLEAR ::
   CALL CHAR :: CALL VCHAR :: CALL HCHAR
30 !@P-
100 CALL CLEAR
110 CALL CHAR(96,"FFFFFFFFFFFFFFFF")
120 CALL CHAR(42,"0F0F0F0F0F0F0F0F")
130 ...
140 ...
150 ...
```

Fortsetzung des Programmauszugs auf der nächsten Seite!

```
160 CALL HCHAR(12,17,42)
170 CALL VCHAR(14,17,96)
190 FOR VERZ=1 TO 1000
200 NEXT VERZ
```

In Programmzeile 20 wird der erforderliche Speicherplatz reserviert, während die Befehle dieser Zeile nicht ausgeführt werden.

Wie auch das folgende Beispiel zeigt, können auf diese Weise alle Bezugnahmen auf Variable in einer Programmzeile zusammengefaßt werden. Auch die Unterprogramm-Aufrufe brauchen syntaktisch nicht korrekt zu sein:

```
100 GOTO 180 :: X,Y,ALPHA,BETA,Z=DELTA :: DIM B(10,10)
110 CALL KEY :: CALL HCHAR :: CALL CLEAR :: CALL UNTERPR
120 DATA 1,3,STRING
130 DEF F(X)=1-X*SIN(X)
140 ...
150 ...
160 !@P-
170 ...
180 ...
```

## SPEICHERPROBLEME

Ob Sie nun Besitzer eines Cassetten-Recorders oder eines Disketten-Laufwerks sind - eines werden Sie zur Aufbewahrung Ihrer Programme anhand der diesbezüglichen Anleitungen zum Computer bzw. zum DISK MANAGER MODUL gleich zu Anfang gelernt haben: das Speichern Ihrer Programme bzw. das Speichern von Daten. Denn wenn man erst einmal in mühevoller Kleinarbeit ein Programm erstellt hat, dann möchte man ja nicht bei nächster Gelegenheit mit derselben Arbeit von vorne beginnen, sondern auf das gespeicherte Programm zurückgreifen können.

Da die diesbezüglichen Bedienungsanleitungen sowohl für den Cassetten-Recorder (vgl. Bedienungsanleitung zum Computer, S. 15 - 18) als auch zum DISK MANAGER MODUL sehr ausführlich gestaltet sind, dürfte die grundsätzliche Bedienung dieser externen Speichergeräte keine Probleme verursachen. Deshalb sollen diese Bedienungshinweise hier auch nicht wiederholt werden, sondern dieser Abschnitt soll auf zwei Besonderheiten hinweisen, die unmittelbar in Zusammenhang mit der Verwendung des EXTENDED BASIC-Moduls stehen, das ja auch Voraussetzung zur Verwendung der Speichererweiterung (MEMORY EXPANSION UNIT) ist. Gerade bei Verwendung der Speichererweiterung ergeben sich nämlich sowohl für den Cassetten-Recorder-Benutzer als auch für den Besitzer eines Disketten-Laufwerks Besonderheiten, die am Ende dieses Kapitels nicht unerwähnt bleiben sollen.

### Cassetten-Recorder:

Trotz der Speichererweiterung um 32 k RAM kann das größte auf Cassette zu speichernde Programm höchstens 12 k umfassen, die Länge des Programms bleibt also bei Recorder-Betrieb bzgl. des Speicherns stärker begrenzt als bei Disketten-Betrieb. Lediglich durch das Programm erzeugte numerische Werte werden bei Verwendung der Speichererweiterung in dieser Einheit gespeichert, so daß der dafür benötigte Speicherplatz nicht im Arbeitsspeicher des Computers in Anspruch genommen wird, was ohne Speichererweiterung der Fall ist. Ohne Verwendung einer Speichererweiterung muß also das Programm kürzer sein, da neben den String-Daten auch die numerischen Daten im Arbeitsspeicher Platz finden müssen.

## Bedienung

Disketten-Laufwerk:

Einige in TI BASIC geschriebene Programme können (vgl. EIN NACHTEIL, Kapitel 2) zu groß für EXTENDED BASIC sein, was zu einem Systemausfall führen kann, der nur durch das Aus- und Wiedereinschalten des Computers zu beheben ist. Wird vor dem Laden des Programms der Befehl CALL FILES(1) oder CALL FILES(2) eingegeben (eine genaue Beschreibung des CALL FILES-Befehls findet sich in der Anleitung zum Disketten-Speicher-System), dann kann damit u.U. genügend Speicherplatz freigesetzt werden, um ein ursprünglich zu umfangreiches Programm auch im EXTENDED BASIC ablaufen zu lassen.

Wenn auch die Eingabe des CALL FILES-Befehls nicht genügend Speicherplatz freisetzt, dann hilft nur noch eine Programmkürzung, es sei denn, die Speichererweiterung steht zur Verfügung. Dann nämlich kann mit Hilfe der folgenden Verfahrensweise das gesamte Programm geladen werden:

1. Als Sicherheit wird zunächst eine Kopie des betreffenden TI BASIC-Programms auf Cassette oder Diskette angefertigt.

2. Bei eingeschalteter Speichererweiterung wird das Programm mit TI BASIC in den Computer geladen. Annschließend werden mehrere Befehle gelöscht und das so verkürzte Programm wird auf Cassette oder Diskette gespeichert. Schließlich wird das gekürzte und gespeicherte Programm in EXTENDED BASIC zu laden versucht. Gelingt dies nicht, müssen weitere Befehle gelöscht werden.

3. Die gelöschten Befehle werdenn wieder an den ursprünglichen Stellen in das Programm eingefügt, und das so ergänzte Programm wird erneut auf Diskette abgespeichert (wobei das Speichern allerdings nicht im PROGRAM-Format erfolgt).

4. Jetzt ist die Ausführung des Programms möglich, allerdings nur im EXTENDED BASIC und bei eingeschalteter Speichererweiterung.

Und nun ist es wirklich an der Zeit, die vielen theoretischen Kenntnisse auch einmal praktisch zu erproben, es ist Zeit für die "Programmierung" im EXTENDED BASIC!

# Kapitel 5

## P R O G R A M M I E R U N G

# Programmierung

## ÜBEREINKÜNFTE UND VORAUSSETZUNGEN

Die hier dargestellten Übereinkünfte und Voraussetzungen (Konventionen), die sich auf die äußere Gestaltung (Format) der Programmiersprache EXTENDED BASIC beziehen, sollten unbedingt beachtet werden, damit die im ERWEITERTEN BASIC geschriebenen Programme laufen und es nicht schon bei der Programm-Eingabe zu im Nachhinein oft nur schwer behebbaren Fehlern kommt.

### Programm-Schnellstart, Powerup

... ist nur bei Diskettenbetrieb möglich und setzt voraus, daß auf einer Diskette in der Diskettenstation 1 ein Programm mit dem Namen LOAD vorhanden ist. Ist das der Fall, so wird dieses Programm geladen und ausgeführt, sobald EXTENDED BASIC nach dem Einschalten und dem Erscheinen des Standardbildes gewählt worden ist, so als ob der Befehl

```
RUN "DSK1.LOAD"
```

eingegeben worden wäre. Ist ein Programm mit einem solchen Namen nicht auf der Diskette vorhanden, dann kommt es, weil zunächst danach gesucht wird, zu einer leichten Verzögerung.

### Extern gespeicherte Datengruppen, Files

... sind in der wohl am häufigsten vorkommenden Form auf Cassette oder Diskette gespeicherte Programme oder Daten. File wird in der Literatur häufig auch mit "Datensatz" wiedergegeben. Beachten Sie für Dateiverarbeitungen besonders die in der Befehlsliste (Teil B) unter dem Stichwort "OPEN" gegebenen Erläuterungen und die Tatsache, daß auch die Daten, die über Peripheriegeräte, wie z.B. das RS232-Interface oder den Thermodrucker ausgegeben werden, ebenfalls als Files betrachtet und diese Geräte demnach als Dateien angesprochen werden müssen.

### Programmzeilen-Nummern, Line Numbers

... sind erforderlich, da sie die Reihenfolge bestimmen, in der die Programmzeilen beim Ablauf eines Programms ausgeführt werden und damit auch die Zeilen festgestellt werden, die durch spezielle Befehle (Statements) angesprochen werden, wie z.B. durch IF-THEN-ELSE, GOTO, GOSUB, ON ERROR, ON-GOTO, ON-GOSUB. Die Verwendung von Programmzeilen-Nummern kann auch im Zusammenhang mit BREAK,LIST,NUM,RESTORE,RETURN und RUN erfolgen. Als Programmzeilen-Nummern sind alle ganzen Zahlen von 1 bis 32767 möglich.

Die Verwendung des NUM-Befehls führt zu einer automatischen Programmzeilen-Numerierung durch den Computer (vgl. NUM in der Befehlsliste, Teil B), die Verwendung des RES-Befehls bewirkt eine Neuordnung in der Programmzeilen-Numerierung (vgl. RES in der Befehlsliste, Teil B).

### Zeilen, Lines

... können inklusive der Programmzeilen-Nummern und der Zwischenräume (Spaces) insgesamt bis zu 140 Zeichen umfassen. Wenn das Zeilenende erreicht ist, ersetzen zusätzlich eingegebene Zeichen jeweils das 140. Zeichen, wobei ein Ton das Zeilenende signalisiert. Zeilen können aber bei Verwendung der Einfügungstaste (<FCTN-2>, INSERT) über 140 Zeichen hinaus verlängert werden (vgl. Kap.4, Spezielle Tastenfunktionen), doch sollte diese Methode nur in Ausnahmefällen angewandt werden, da der Computer nicht immer eine überlange Zeile mit der Meldung LINE TOO LONG (Zeile zu lang) zurückweist und es somit u.U. zu Datenverlusten kommen kann.

# Programmierung

## Separatoren, Special Symbols

... sind Trennsymbole. Im EXTENDED BASIC ist es möglich, mehrere Befehle (Statements) in eine Programmzeile zu schreiben, und außerdem kann am Zeilenende ein Schlußkommentar (Tail Remark) angefügt werden, der bei der Programmausführung überlesen wird und mit dessen Hilfe das Programm durchsichtiger gestaltet werden kann. Die einzelnen in einer Programmzeile befindlichen Befehle müssen voneinander, der Schlußkommentar muß vom Rest der Programmzeile abgetrennt werden. Dies geschieht durch die Separatoren, und zwar ist das Trennsymbol für das Abtrennen von Befehlen untereinander der doppelte Doppelpunkt (::), und das Trennsymbol für das Abtrennen eines Schlußkommentars ist das Ausrufezeichen (!). Die folgende Programmzeile zeigt die Verwendung dieser beiden Trennsymbole:

```
100 FOR A=1 TO 50 :: PRINT A,SQR(A) :: NEXT A !AUSGABE QUADRATWURZELN
```

Die bei der Programmabtastung (Pre-Scanning, vgl. Kap.4) verwendeten Zeichengruppen (!@P+ und !@P-) stellen eigenständige neue Befehle (Commands) dar und dürfen deshalb als "Reservierte Wörter" (vgl. die Wortliste unter dem Stichwort "Variable") nicht nach dem Trennsymbol "!" als Schlußkommentar verwendet werden.

## Zwischenräume, Spaces

... entstehen bei Betätigung der Leertaste und sind im EXTENDED BASIC erforderlich, um die einzelnen Teile eines Befehls voneinander abzutrennen. Nur dann kann der Computer zwischen den einzelnen Elementen der Programmiersprache unterscheiden. In einigen Fällen sind Zwischenräume nicht erforderlich, sie werden u.U., falls sie doch gesetzt werden, vom Computer wieder herausgekürzt, und zwar:

- vor und nach Vergleichssymbolen (=, >, <),
- vor und nach dem Schlußkommentar-Symbol (!),
- vor und nach dem Trennsymbol für Befehle (::).

Beim Auflisten (LIST) von Programmen können dagegen Zwischenräume vor und nach den Trennsymbolen entstehen.

## Numerische Konstanten, Numeric Constants

... sind jede beliebige reelle Zahl und können deshalb mit einer beliebig großen Anzahl von Stellen eingegeben werden. Sie werden aber durch den Computer gemäß einer internen Speichermethode immer auf 13 oder 14 Stellen gerundet. Ausgegeben werden sie mit höchstens 10 Stellen. Deshalb ist für sehr große oder sehr kleine Zahlen die wissenschaftliche Schreibweise die geeignetere Methode der Zahlendarstellung, die vom Computer auch bei der Ausgabe sehr großer oder sehr kleiner Zahlen angewandt wird. In dieser Schreibweise wird eine Basiszahl (Mantisse) mit einer Zehnerpotenz (Exponent) multipliziert, wobei der Großbuchstabe E für "* 10" steht (auch bei Verwendung von Kleinbuchstaben muß hier der Großbuchstabe E unbedingt verwendet werden!).

Beispiele:

$1500 = 1.5*10^{3} = 1.5E3$ $\quad -1500 = -1.5*10^{3} = -1.5E3$

$0.15 = 1.5*10^{-1} = 1.5E-1$ $\quad -0.15 = -1.5*10^{-1} = -1.5E-1$

## Programmierung

Numerische Konstanten sind für den Bereich von -9.9999999999999E127 bis -1E-128, 0, und von 1E-128 bis 9.9999999999999E127 definiert. Wenn der Exponent einer berechneten Zahl größer als 99 oder kleiner als -99 ist, dann wird normalerweise ** angezeigt, während der vollständige Exponent gespeichert wird und über USING in einer PRINT- oder DISPLAY-Anweisung ausgegeben werden kann.

### String-Konstanten, String Constants

... können im EXTENDED BASIC die Länge einer Eingabezeile haben. Sie werden aus einer Folge von Zeichen (Buchstaben, Leerstellen, Spezialsymbolen usw.) gebildet, die in Anführungszeichen gesetzt sind. Leerstellen in einer String-Konstanten werden berücksichtigt und als Zeichen mitgezählt. Auch die normalen Anführungszeichen dürfen in einer String-Konstanten verwendet werden, sie werden auch bei Verwendung von PRINT oder DISPLAY im Gegensatz zu den Anführungszeichen, mit denen die String-Konstante umschlossen wird, ausgegeben.

### Variable, Variables

... sind veränderliche Größen, denen in einem Programmablauf ein eindeutiger Wert zugewiesen wird. Variablen-Bezeichnungen können aus bis zu 15 Zeichen (inklusive des $-Zeichens bei String-Variablen) bestehen und müssen immer mit einem Buchstaben, dem "Klammeraffen" (@) oder dem Unterstrich (_) beginnen. Wird die Anzahl von 15 Zeichen überschritten, erscheint BAD NAME (falscher Name), und die Zeile wird nicht gespeichert. Eine String-Variable muß mit dem $-Zeichen abschließen. Variable können dimensioniert, d.h. als Datenfelder (Arrays) verwendet werden, wobei im EXTENDED BASIC bis zu 7 Dimensionen zulässig sind. In einem Programm darf der gleiche Name nicht wiederholt verwendet werden, ausgenommen sind hier lediglich die Unterprogramme (Subprograms), auch zwei Datenfelder mit unterschiedlichen Dimensionen dürfen nicht gleich benannt werden.

Z und Z(3) sind zusammen in einem Programm ebenso wenig zulässig wie X(3,4) und X(4,5,6,7). Zulässig ist aber der gleiche Name für eine numerische und eine String-Variable, da hier durch das $-Zeichen unterschieden wird: NAME und NAME$ dürfen also zusammen in einem Programm verwendet werden. Einige Beispiele für zulässige und unzulässige Variablen-Bezeichnungen:

| | numerisch | String |
|---|---|---|
| zulässig: | X,A4,BALL<br>TABU(2,X,9/3) | X$,A4$,BALL$<br>TABU$(1,2,3,4,5,6,7) |
| unzulässig: | X$,X/8,3Y | X$4,X4,4X$ |

Bestimmte Wörter, aus denen die Programmiersprache zusammengesetzt ist, das sind die Befehle (Commands, Statements, Functions) und andere Operatoren, wie etwa logische Ausdrücke (z.B. AND) gelten als reserviert und dürfen deshalb nicht als Variablen-Bezeichnungen benutzt werden. Sie können aber durchaus Teil einer Variablen-Bezeichnung sein. LIST ist unzulässig als Variablen-Bezeichnung, LIST$ dagegen durchaus zulässig.

# Programmierung

## Liste der reservierten Wörter im EXTENDED BASIC:

| | | | |
|---|---|---|---|
| ABS | EOF | NUMBER | SEQUENTIAL |
| ACCEPT | ERASE | NUMERIC | SGN |
| ALL | ERROR | OLD | SIN |
| AND | EXP | ON | SIZE |
| APPEND | FIXED | OPEN | SQR |
| ASC | FOR | OPTION | STEP |
| AT | GO | OR | STOP |
| ATN | GOSUB | OUTPUT | STR$ |
| BASE | GOTO | PERMANENT | SUB |
| BEEP | IF | PI | SUBEND |
| BREAK | IMAGE | POS | SUBEXIT |
| BYE | INPUT | PRINT | TAB |
| CALL | INT | RANDOMIZE | TAN |
| CHR$ | INTERNAL | READ | THEN |
| CLOSE | LEN | REC | TO |
| CON | LET | RELATIVE | TRACE |
| CONTINUE | LINPUT | REM | UALPHA |
| COS | LIST | RES | UNBREAK |
| DATA | LOG | RESEQUENCE | UNTRACE |
| DEF | MAX | RESTORE | UPDATE |
| DELETE | MERGE | RETURN | USING |
| DIGIT | MIN | RND | VAL |
| DIM | NEW | RPT$ | VALIDATE |
| DISPLAY | NEXT | RUN | VARIABLE |
| ELSE | NOT | SAVE | WARNING |
| END | NUM | SEG$ | XOR |

Reserviert ist auch die Zeichengruppe für die Programmabtastung (s.d.)

## Numerische Ausdrücke, Numeric Expressions

... werden aus numerischen Variablen, numerischen Konstanten und Funktionen, die die arithmetischen Operatoren für Addition (+), Subtraktion (-), Multiplikation (*), Division (/) und Potenzierung (^) anwenden, gebildet. Das Minuszeichen als Präfix-Operator bewirkt, daß der nachstehende Ausdruck nach den geltenden Regeln mit -1 multipliziert wird (z.B. -3^2 = -(3^2) = -9).

Für Berechnungen gelten die Standardregeln der mathematischen Hierarchie der Operatoren, nach der die numerischen Ausdrücke in der folgenden Reihenfolge berechnet werden:

1. Alle Ausdrücke in den Klammern
2. Potenzierungen in der Reihenfolge von links nach rechts
3. Verarbeitung der Präfix-Operatoren (s.o.)
4. Multiplikation und Division
5. Addition und Subtraktion

Beispiele zur Verdeutlichung:

| Ausdruck | Zwischenergebnisse | | Endergebnis |
|---|---|---|---|
| 4+2^2/2-6 | 4+4/2-6 | 4+2-6 | 0 |
| (4+2)^2/2-6 | 6^2/2-6 | 36/2-6 | 12 |
| 4+2^2/(2-6) | 4+4/(-4) | 4-1 | 3 |

# Programmierung

## String-Ausdrücke, String Expressions

... werden aus String-Variablen, String-Konstanten und String-Funktionen gebildet. Hinzu kommt der Verkettungsoperator (&), mit dessen Hilfe, wie das folgende Kurzprogramm zeigt, Strings kombiniert werden können:

```
100 A$="HALLO, "&"DU DA!"
110 PRINT A$
```

Bei Ablauf des Programms wird der String "HALLO, DU DA!" ausgegeben. Wenn die Verarbeitung eines String-Ausdrucks zu einem Wert führt, der die maximale String-Länge von 255 Zeichen übersteigt, wird der String auf der rechten Seite abgeschnitten!

## Vergleiche, Relational Expressions

... werden im allgemeinen im IF-THEN-ELSE-Befehl verwendet, können aber auch generell überall dort verwendet werden, wo numerische Ausdrücke erlaubt sind. Bei Verwendung von Vergleichen innerhalb numerischer Ausdrücke erhält die logische Variable den Wert -1, wenn der Ausdruck zutrifft und den Wert 0, wenn er nicht zutrifft. Vergleiche werden von links nach rechts vor der String-Verkettung (&) und nach Abschluß aller arithmetischer Operationen innerhalb des Ausdrucks durchgeführt. Die Vergleichsoperatoren sind:

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

---

Beispiele:

In Programmzeile 100 führt der Computer als nächstes den Befehl in Zeile 200 aus, wenn die numerische Variable X kleiner als Y ist. Andernfalls - ist X also größer oder gleich Y - erfolgt ein Sprung in die ab Zeile 420 beginnende Unterprogrammschleife.

```
100 IF X<Y THEN 200 ELSE GOSUB 420
```

In Programmzeile 120 wird die Variable C gleich S+1 gesetzt, wenn die Variable L(C) gleich 12 ist. Ist L(C) dagegen ungleich 12, dann wird die Variable ZAEHLER um 1 erhöht, und der Programmablauf geht mit Programmzeile 140 weiter.

```
120 IF L(C)=12 THEN C=S+1 ELSE
    ZAEHLER=ZAEHLER+1 :: GOTO 200
```

In Programmzeile 140 wird die numerische Variable A gleich -1 gesetzt, da der Vergleich 7<12 wahr ist.

```
140 A=7<12
```

In Programmzeile 160 wird 0 ausgegeben, da die beiden Strings "ROT" und "GRUEN" nicht gleich sind.

```
160 PRINT "ROT"="GRUEN"
```

In Programmzeile 180 erhält die logische Variable A den Wert -1 zugewiesen, wenn die numerische Variable B gleich 7 ist, andernfalls den Wert 0.

```
180 A=B=7
```

# Programmierung

## Logische Ausdrücke, Logical Expressions

... werden im Zusammenhang mit Vergleichen verwendet. Die dazu möglichen zusätzlichen logischen Operatoren sind AND, OR, XOR und NOT. Ist der logische Ausdruck wahr, so erhält er den Wert -1 und andernfalls den Wert 0 zugeordnet.

Die Hierarchie der logischen Operatoren (vergleichbar mit der Hierarchie der Rechenoperatoren, vgl. "Numerische Ausdrücke") ist

1. NOT
2. XOR
3. AND
4. OR.

---

Beispiele:

Die Variable L in Programmzeile 100 wird gleich 7 gesetzt, da sowohl 3 kleiner als 4 als auch 5 kleiner als 6 ist, aber in Programmzeile 120 nicht, da zwar 3 kleiner als 4, nicht aber 5 größer als 6 ist.

```
100 IF 3<4 AND 5<6 THEN L=7
120 IF 3<4 AND 5>6 THEN L=7
```

Ein logischer Ausdruck mit AND ist wahr, wenn beide Aussagen, die dem AND vorangehende u n d die dem AND folgende, wahr sind.

Die Variable L in Programmzeile 140 wird gleich 7 gesetzt, da 3 kleiner als 4 ist.

```
140 IF 3<4 OR 5>6 THEN L=7
```

Ein logischer Ausdruck mit OR ist wahr, wenn eine der beiden Aussagen, entweder die dem OR vorausgehende o d e r die dem OR folgende wahr ist (es können auch beide Aussagen wahr sein, eine genügt jedoch).

In Programmzeile 160 erhält die Variable L den Wert 7, da 3 kleiner als 4 und 5 nicht größer als 6 ist, in 180 jedoch nicht, da ja auch 5 kleiner als 6 ist.

```
160 IF 3<4 XOR 5>6 THEN L=7
180 IF 3<4 XOR 5<6 THEN L=7
```

Ein logischer Ausdruck mit XOR ist wahr, wenn genau eine der beiden Aussagen, e n t w e d e r die dem XOR vorangehende o d e r die dem XOR folgende wahr ist, nicht aber beide gleichzeitig.

In Programmzeile 200 wird die Variable L gleich 7 gesetzt, da 3 nicht gleich 4 ist.

```
200 IF NOT 3=4 THEN L=7
```

Ein logischer Ausdruck mit NOT ist wahr, wenn die dem NOT folgende Aussage n i c h t wahr ist.

Es folgen noch zwei Programmzeilen, die verdeutlichen sollen, daß gleichzeitig mehrere logische Ausdrücke zusammen verwendet werden können:

```
220 IF NOT 3=4 AND (NOT 6=5 XOR 2=2) THEN L=7
240 IF (A OR B) AND (C XOR D) THEN 500
```

## Programmierung

Die logischen Operatoren lasen sich auch direkt auf Zahlen anwenden. Dabei werden die Dezimalzahlen zunächst in Dualzahlen umgewandelt, anschließend wird die logische Operation bitweise ausgeführt und das Ergebnis wieder in eine Dezimalzahl umgewandelt. Eine genauere Erörterung der Anwendung logischer Operatoren auf Zahlen und Bitkombinationen ist der einschlägigen mathematischen Fachliteratur zu entnehmen, hier nur soviel:

AND setzt eine 1 an die entsprechende Bitstelle, wenn beide Dualzahlen an dieser Stelle eine 1 haben, andernfalls eine 0.

OR setzt an die entsprechende Bitstelle eine 1, wenn mindestens eine der beiden Dualzahlen an dieser Bitstelle eine 1 hat, andernfalls eine 0.

XOR setzt an der entsprechenden Bitstelle eine 1, wenn genau eine der beiden Dualzahlen an dieser Bitstelle eine 1 hat, andernfalls eine 0.

NOT wandelt jede Bitstelle einer Dualzahl um, und zwar wird aus jeder 0 eine 1 und aus jeder 1 eine 0.

## Programmierung

### EIN BEISPIEL

Nachdem Sie nun in Grundzügen mit den Möglichkeiten des EXTENDED BASIC vertraut sind, wird es Ihnen sicherlich Spaß machen, anhand eines Beispielprogramms die erworbenen Kenntnisse nicht nur zu überprüfen, sondern durch das Kennenlernen von zeitsparenden Verkürzungen bei der Programmeingabe auch noch zu vertiefen.

Das mit "Zahlencode-Spiel" betitelte Beispielprogramm ist eine Variante des allseits bekannten "Superhirn"- bzw. "Mastermind"-Spiels. Zunächst wird die Größe der zu erratenden Zahl, die 1 bis 8 Stellen umfassen kann, bestimmt. Danach wird eingegeben, wieviele verschiedene Ziffern in der Zahl vorkommen sollen. Wird z.B. "3" eingegeben, verwendet der Computer die Ziffern 0, 1 und 2 in der zu erratenden Zahl, bei "9" die Ziffern 0 bis 8, die höchste Eingabe ist demnach "10": dann verwendet der Computer alle Ziffern von 0 bis 9. Da der Computer innerhalb der zu erratenden Zahl keine Ziffer wiederholt, muß die Anzahl der Ziffern mindestens genauso groß sein wie die Anzahl der Stellen. Wenn für die Zahlengröße also "4" eingegeben wird, so muß für die Anzahl der Ziffern auch mindestens "4" eingegeben werden. Der Computer bestimmt dann die Zufallszahl unter den von Ihnen bestimmten Bedingungen und meldet sich mit:

```
VERSUCH:      RICHTIG:      STELLE:
```

Unter "VERSUCH:" geben Sie dann Zahlen in der von Ihnen gewählten Größe ein. Nachdem Sie <ENTER> gedrückt haben, gibt Ihnen der Computer unter "RICHTIG:" die Anzahl der richtig geratenen Ziffern an und unter "STELLE:" die Anzahl der zusätzlich an der richtigen Stelle der zu erratenden Zahl befindlichen Ziffern an.

Bedenken Sie, daß der Computer innerhalb seiner Zahl keine Ziffer wiederholt. Sollten Sie bei Ihren Versuchen Ziffern mehrfach eingeben, dann zählt der Computer sie auch jedesmal als richtig. Geben Sie unter "VERSUCH:" bei einer vierstelligen Zahl also beispielsweise "1111" ein, und die Ziffer "1" ist in der zu erratenden Zahl enthalten, dann meldet sich der Computer mit:

```
VERSUCH:      RICHTIG:      STELLE:
1111             4             1
```

Ziffernwiederholungen dürften bei Ihren Versuchen im Normalfall also nur wenig informativ sein. Sie gewinnen, wenn Sie die zu erratende Zahl korrekt, d.h. mit allen Ziffern an der richtigen Stelle unter "VERSUCH:" eingegeben haben. Wenn Sie aufgeben wollen, geben Sie unter "VERSUCH:" ein "X" ein, dann verrät Ihnen der Computer die gesuchte Zahl.

Ein Spiel könnte also wie folgt ablaufen, wenn man einmal annimmt, daß Sie die Zahlengröße mit 4 und die Anzahl der Ziffern mit 9 (Ziffern 0 - 8) bestimmt haben und der Computer die Zufallszahl 0743 gewählt hat:

```
VERSUCH:  RICHTIG:  STELLE:    (Bedeutung der Information)
0000         4         1     alle Ziffern richtig, eine an richtiger Stelle
1234         2         0     2 Ziffern richtig (3,4),keine an richtiger St.
5678         1         0     1 Ziffer richtig (7), keine an richtiger St.
2348         2         1     2 Ziffern richtig (3,4), eine an richtiger St.
0347         4         2     alle Ziffern richtig, zwei an richtiger Stelle
3047         4         1     alle Ziffern richtig, eine an richtiger Stelle
0734         4         2     alle Ziffern richtig, zwei an richtiger Stelle
0743         4         4     alle Ziffern richtig, alle an richtiger Stelle
```

## Programmierung

Nachdem die letzte Versuchszahl eingegeben ist, meldet der Computer den Gewinn und die Anzahl der Versuche, die zur Lösung benötigt wurden. Danach fragt er den Spieler, ob dieser weiterspielen möchte.

Um das Programm dieses Spiels in den Computer einzugeben, verfahren Sie zunächst so, wie unter "Vorbereitung des Computers" beschrieben: Schalten Sie die Peripheriegeräte ein, die an den Computer angeschlossen sind, bringen Sie das EXTENDED BASIC-Modul an seinen Platz und schalten Sie zuletzt den Computer ein. Das Titelbild erscheint. Drücken Sie nun eine beliebige Taste, anschließend die <2>, um das EXTENDED BASIC zu wählen. Der Computer zeigt seine Betriebsbereitschaft durch * READY * an, und Sie können mit der Eingabe des Programms beginnen.

Für die Fortgeschrittenen hier zunächst das vollständige Programm. Wie das Programm im einzelnen eingegeben wird, wird auf den folgenden Seiten genauestens erklärt.

```
100 REM ***** ZAHLENCODE-SPIEL *****
110 DIM ZAHL$(8),VERSUCH$(8)
120 RANDOMIZE
130 DISPLAY AT(11,7)BEEP ERASE ALL:"ZAHLENCODE-SPIEL"
140 DISPLAY AT(19,1)BEEP:"ZAHLENGROESSE(1-8 STELLEN)?"
150 DISPLAY AT(21,1)BEEP:"WIEVIELE ZIFFERN(1-10)?"
160 ACCEPT AT(19,28)VALIDATE(DIGIT):ZAHL
170 ACCEPT AT(21,24)VALIDATE(DIGIT):ZIFFER
180 IF ZAHL>ZIFFER THEN DISPLAY AT(24,2)BEEP:"ZU WENIG ZIFFERN!" :: GOTO 160
190 FOR A=1 TO ZAHL !WAHL DER ZUFALLSZAHLEN
200 ZAHL$(A)=STR$(INT(RND*ZIFFER))
210 FOR B=0 TO A-1 !KEINE GLEICHEN ZAHLEN
220 IF ZAHL$(A)=ZAHL$(B)THEN 200
230 NEXT B
240 NEXT A
250 ZEILE=2
260 DISPLAY AT(1,1)ERASE ALL:"VERSUCH:   RICHTIG:  STELLE:"
270 DISPLAY AT(24,1):"EINGABE <X> = LOESUNG!"
280 ACCEPT AT(ZEILE,1):V$
290 IF V$="X" THEN 470 !AUFGABE ODER NEUSTART
300 FOR D=1 TO ZAHL !RATEVERSUCH
310 VERSUCH$(D)=SEG$(V$,D,1)
320 NEXT D
330 RICHTIG,STELLE=0
340 FOR E=1 TO ZAHL !PRUEFUNG VERSUCH
350 FOR F=1 TO ZAHL
360 IF ZAHL$(E)=VERSUCH$(F)THEN RICHTIG=RICHTIG+1 :: IF E=F THEN STELLE=STELLE+1
370 NEXT F
380 NEXT E
390 DISPLAY AT(ZEILE,14):RICHTIG
400 DISPLAY AT(ZEILE,24):STELLE
410 IF STELLE<>ZAHL THEN ZEILE=ZEILE+1 :: IF ZEILE>22 THEN 470 ELSE 280
420 DISPLAY AT(23,1)BEEP:"DU GEWINNST:";ZEILE-1;"VERSUCHE"
430 DISPLAY AT(24,1)BEEP:"NEUES SPIEL(J/N)? J"
440 ACCEPT AT(24,19)SIZE(-1)BEEP VALIDATE("JN"):X$
450 IF X$="J" THEN 190
460 STOP
470 DISPLAY AT(23,1)BEEP:"DIE ZAHL IST:" !VERLOREN,AUFGABE,NEUSTART
480 FOR G=1 TO ZAHL
490 DISPLAY AT(23,14+G):ZAHL$(G)
500 NEXT G
510 DISPLAY AT(24,1)BEEP:"NEUES SPIEL(J/N)? J"
520 ACCEPT AT(24,19)SIZE(-1)BEEP VALIDATE("JN"):X$
530 IF X$="J" THEN 130
```

## Programmierung

Im folgenden wird die Eingabe des Programms im einzelnen erklärt, wobei der einzugebende Teil links und die danach zu betätigende Funktionstaste rechts über dem Kommentar stehen:

| EINGABE: | FUNKTIONSTASTE: |
|---|---|
| >NUM | <ENTER> |

Numeriert die Programmzeilen automatisch beginnend mit 100 in Zehnerschritten. Die eingeklammerten Zeilennummern werden vorgegeben.

| | |
|---|---|
| (100) REM ***** ZAHLENCODE-SPIEL ***** | <ENTER> |

Programmtitel als Kommentar (Zeile wird bei Programmausführung überlesen).

| | |
|---|---|
| (110) DIM ZAHL$(8),VERSUCH$(8) | <ENTER> |

Reserviert Speicherplatz für Zahlen und Rateversuche (beide als eindimensionale String-Datenfelder).

| | |
|---|---|
| (120) RANDOMIZE | <ENTER> |

Die Abfolge der vom Computer gewählten Zahlen wird zufällig.

| | |
|---|---|
| (130) DISPLAY AT(11,7)BEEP ERASE ALL:"ZAHLENCODE-SPIEL" | <ENTER> |

Löscht den Bildschirm, erzeugt einen Ton und schreibt den Titel "ZAHLENCODE-SPIEL" in die 11. Zeile beginnend ab der 7. Spalte auf dem Bilschirm.

| | |
|---|---|
| | <FCTN-8> |

Nach dem letzten Drücken von <ENTER> erschien gemäß dem NUM-Befehl die Zeilennummer 140. Durch gleichzeitiges Betätigen der Tasten <FCTN> und <8> (REDO) wird die Zeile 130 auf den Bildschirm zurückgeholt.

| | |
|---|---|
| | <FCTN-S>,<FCTN-D>,<FCTN-1>,<FCTN-2> |

Durch Gebrauch der Pfeiltasten und der Einfügungs- und Löschtaste wird diese Zeile an den unterstrichenen Stellen überschrieben und die Zeilennummer in 140 abgeändert. <FCTN-S>= Pfeil links, <FCTN-D>= Pfeil rechts, <FCTN-1>= Löschen, <FCTN-2>= Einfügen.

| | |
|---|---|
| 140 DISPLAY AT(19,1)BEEP:"ZAHLENGROESSE(1-8 STELLEN)?" | <ENTER> |

In der 19. Zeile wird, begleitet von einem Ton, "ZAHLENGROESSE (1-8 STELLEN?" angezeigt.

| | |
|---|---|
| | <FCTN-8> |

Durch REDO wird Zeile 140 auf den Bildschirm zurückgeholt.

| | |
|---|---|
| | <FCTN-S>,<FCTN-D>,<FCTN-1>,<FCTN-2> |

Wie oben wird mit Hilfe der angegebenen Funktionstasten die Zeile 140 an den unterstrichenen Stellen überschrieben und in Zeile 150 abgeändert.

| | |
|---|---|
| 150 DISPLAY AT(21,1)BEEP:"WIEVIELE ZIFFERN (1-10)?" | <ENTER> |

Zeigt in Zeile 21, begleitet von einem Ton, "WIEVIELE ZIFFERN (1-10)?" an.

## Programmierung

---

160 ACCEPT AT(19,28)VALIDATE(DIGIT):ZAHL <ENTER>

---

Weist der Variablen ZAHL den in der 19. Zeile und 28. Spalte eingegebenen Wert zu, der die Zahlenngröße (1-8 Stellen) bestimmt. Es werden nur Ziffern als Eingabe angenommen.

---

<FCTN-8>

---

Durch REDO wird die Zeile 160 auf den Bildschirm zurückgeholt.

---

<FCTN-S>,<FCTN-D>,<FCTN-1>,<FCTN-2>

---

Mit Hilfe der angegebenen Funktionstasten wird Zeile 160 an den unterstrichenen Stellen überschrieben und in Zeile 170 abgeändert.

---

170 ACCEPT AT(21,24)VALIDATE(DIGIT):ZIFFER <ENTER>

---

Weist der Variablen ZIFFER den in der 21. Zeile und 24. Spalte eingegebenen Wert zu, der die in der Zahl vorhandene Anzahl von verschiedenen Ziffern bestimmt. Es werden nur Ziffern als Eingabe angenommen.

---

LIST <ENTER>

---

Listet das bisher eingegebene Programm auf dem Bildschirm wie folgt auf:

```
100 REM ***** ZAHLENCODE-SPIEL *****
110 DIM ZAHL$(8),VERSUCH$(8)
120 RANDOMIZE
130 DISPLAY AT(11,7)BEEP ERASE ALL:"ZAHLENCODE-SPIEL"
140 DISPLAY AT(19,1)BEEP:"ZAHLENGROESSE(1-8 STELLEN)?"
150 DISPLAY AT(21,1)BEEP:"WIEVIELE ZIFFERN(1-10)?"
160 ACCEPT AT(19,28)VALIDATE(DIGIT):ZAHL
170 ACCEPT AT(21,24)VALIDATE(DIGIT):ZIFFER
```

---

RUN <ENTER>

---

Mit dem RUN-Befehl kann das Programm, soweit es bis jetzt eingegeben ist, ablaufen. Dabei wird zunächst der Bildschirm gelöscht, und dann erscheint:

```
                ZAHLENCODE-SPIEL


    ZAHLENGROESSE (1-8 STELLEN)?

    WIEVIELE ZIFFERN (1-10)?
```

Der Positionszeiger blinkt an der Stelle Ihrer ersten Eingabe. Wenn Sie versuchen, etwas anderes als eine Ziffer einzugeben, hören Sie einen Ton, und der Computer verweigert die Annahme. Geben Sie z.B. "4" ein, springt der Positionszeiger an die schraffierte Stelle. Wenn Sie jetzt wieder eine Zahl eingeben, z.B. "10", dann endet das Programm. Der Computer meldet sich mit * READY *, und Sie können mit der Programmeingabe fortfahren.

Solche "Probeläufe" dienen zur vorzeitigen Überprüfung, ob ein Programm auch fehlerfrei eingegeben wurde oder der bisherige Programmteil auch fehlerfrei abläuft.

## Programmierung

```
>NUM 180                                                          <ENTER>
```

Die automatische Zeilennumerierung wird mit Zeile 180 in Zehnerschritten fortgesetzt.

```
(180) IF ZAHL>ZIFFER THEN DISPLAY AT(24,2)BEEP:"ZU WENIG ZIFFERN!" ::
      GOTO 160                                                    <ENTER>
```

Die Zeile überprüft, ob die Anzahl der Ziffern für die Größe der gesuchten Zahl ausreicht (vgl Einleitung zum Programm). Wenn diese Anzahl kleiner oder gleich der Anzahl der Ziffern ist, wird das Programm mit der nächsten Zeile fortgeführt. Ist dies nicht der Fall, so wird in der letztenn Bildschirmzeile "ZU WENIG ZIFFERN!" angezeigt, und die Eingabe für Zahlengröße und Anzahl der Ziffern (ab Zeile 160) wird wiederholt.

```
(190) FOR A=1 TO ZAHL !WAHL DER ZUFALLSZAHLEN                     <ENTER>
```

Schleifenbeginn für die Ziffernauswahl für den vom Computer gebildeten Ziffernstring (= Geheimzahl). Kommentar hinter dem Ausrufezeichen.

```
(200) ZAHL$(A)=STR$(INT(RND*ZIFFER))                              <ENTER>
```

Die Zeile bewirkt, daß die Auswahl der Ziffern für den String zufällig erfolgt.

```
(210) FOR B=0 TO A-1 !KEINE GLEICHEN ZAHLEN                       <ENTER>
```

Beginn einer Schleife, durch die verhindert wird, daß in der Geheimzahl Ziffern mehrfach auftreten. Kommentar hinter dem Ausrufezeichen.

```
(220) IF ZAHL$(A)=ZAHL$(B) THEN 200                               <ENTER>
```

Wenn Ziffern doppelt gewählt wurden, wird durch Rücksprung auf Zeile 200 eine neue Ziffer statt der doppelten gewählt.

```
(230) NEXT B                                                      <ENTER>
```

Ende der Prüfschleife für doppelte Ziffern.

```
(240) NEXT A                                                      <ENTER>
```

Ende der Wahlschleife für die Ziffern, aus denen die Geheimzahl als String zusammengesetzt worden ist.

```
(250) ZEILE=2                                                     <ENTER>
```

Bestimmt die Variable ZEILE für die Stelle, von der ausgehend Informationen auf dem Bildschirm angezeigt werden.

```
(260) DISPLAY AT(1,1)ERASE ALL:"VERSUCH:   RICHTIG:  STELLE:"     <ENTER>
```

Der Bildschirm wird gelöscht, und in der 1. Zeile werden die Überschriften für die Versuchs- und Informationsspalten ausgegeben.

```
                                                                  <FCTN-8>
```

Zeile 260 wird auf den Bildschirm zurückgeholt (REDO).

## Programmierung

---

<FCTN-S>,<FCTN-D>,<FCTN-1>,<FCTN-2>

---

Mit Hilfe der angegebenen Tastenfunktionen wird die Zeile 260 an den unterstrichenen Stellen überschrieben und in Zeile 270 abgeändert.

---

```
270 DISPLAY AT(24,1):"EINGABE <X>= LOESUNG!"                    <ENTER>
```

---

In der letzten Bildschirmzeile wird die Information "EINGABE <X>= LOESUNG!" angezeigt.

---

```
>NUM 280                                                        <ENTER>
```

---

Numeriert wieder automatisch die Programmzeilen in Zehnerschritten ab Zeile 280.

---

```
(280) ACCEPT AT(ZEILE,1):V$                                     <ENTER>
```

---

Eingabe des Rateversuchs als Ziffern-String in der jeweils vorgesehenen Zeile (vgl. Zeilennummern 250 und 410).

---

```
(290) IF V$="X" THEN 470 !AUFGABE ODER NEUSTART                 <ENTER>
```

---

Wird kein Ziffern-String als Versuch, sondern "X" eingegeben, fährt das Programm mit Zeile 470 fort. Kommentar hinter Ausrufezeichen.

---

```
(300) FOR D=1 TO ZAHL !RATEVERSUCH                              <ENTER>
```

---

Beginn der Überprüfung auf Übereinstimmung zwischen Geheimzahl und Versuch 1. Schritt: Schleife zur Aufspaltung der Versuchszahl. Kommentar hinter Ausrufezeichen.

---

```
(310) VERSUCH$(D)=SEG$(V$,D,1)                                  <ENTER>
```

---

Der als Versuch eingegebene Ziffern-String wird in seine einzelnen Bestandteile (Ziffern) aufgeteilt.

---

```
(320) NEXT D                                                    <ENTER>
```

---

Ende der Schleife, in der die eingegebene Versuchszahl in ihre Ziffern aufgeteilt wird.

---

```
(330) RICHTIG,STELLE=0                                          <ENTER>
```

---

Die Variablen RICHTIG und STELLE, mit Hilfe derer der Computer das Ergebnis der Überprüfung auf Übereinstimmung mitteilt, werden gleich 0 gesetzt.

---

```
(340) FOR E=1 TO ZAHL !PRUEFUNG VERSUCH                         <ENTER>
```

---

Beginn der äußeren Schleife zum Vergleich zwischen Rateversuch und Geheimzahl und Überprüfung auf Übereinstimmung. Kommentar nach Ausrufezeichen.

---

```
(350) FOR F=1 TO ZAHL                                           <ENTER>
```

---

Beginn der inneren Schleife zur Überprüfung auf Übereinstimmung zwischen einzelner Versuchsziffer und einzelner Geheimziffer.

---

```
(360) IF ZAHL$(E)=VERSUCH$(F) THEN RICHTIG=RICHTIG+1 ::
      IF E=F THEN STELLE=STELLE+1                               <ENTER>
```

---

## Programmierung

Ergebnisauswertung. Bei fehlender Übereinstimmung fährt das Programm mit der nächsten Zeile fort. Stimmen Versuchsziffer und Geheimziffer überein, wird die Variable RICHTIG um 1 erhöht. Stimmt zudem die Stelle der Versuchsziffer mit der Stelle der Geheimziffer überein, wird auch die Variable STELLE um 1 erhöht. Die beiden Vergleiche und die Auswertungen werden entsprechend der Zahlengröße für alle Ziffern wiederholt.

---

(370) NEXT F <ENTER>

---

Ende der inneren Prüfschleife.

---

(380) NEXT E <ENTER>

---

Ende der äußeren Prüfschleife.

---

(390) DISPLAY AT(ZEILE,14):RICHTIG <ENTER>

---

Ergebnisanzeige der Überprüfung auf dem Bildschirm. Die Variable RICHTIG zeigt in der entsprechenden Zeile die Anzahl der Ziffern-Übereinstimmungen an.

---

<FCTN-8>

---

Durch REDO wird die Zeile 390 auf den Bildschirm zurückgeholt.

---

<FCTN-S>,<FCTN-D>,<FCTN-1>,<FCTN-2>

---

Mit Hilfe der angegebenen Funktionstasten wird Zeile 390 an den unterstrichenen Stellen überschrieben und in Zeile 400 abgeändert.

---

400 DISPLAY AT(ZEILE,24):STELLE <ENTER>

---

Die Variable STELLE zeigt in der entsprechenden Zeile die Anzahl der Versuchsziffern an, die in ihrer Position mit den Ziffern der Geheimzahl übereinstimmen.

---

>NUM 410 <ENTER>

---

Automatische Zeilennumerierung in Zehnerschritten, beginnend mit Zeile 410

---

```
(410) IF STELLE<>ZAHL THEN ZEILE=ZEILE+1 ::
      IF ZEILE>22 THEN 470 ELSE 280
```
<ENTER>

---

Überprüfung auf komplette Übereinstimmung zwischen Versuchszahl und Geheimzahl, die dann gegeben ist, wenn der Wert der Variablen STELLE gleich dem der Variablen ZAHL ist, da dann die Versuchszahl der Geheimzahl in allen Positionen entspricht. Ist das der Fall, fährt das Programm mit der nächsten Zeile fort. Andernfalls wird die Variable ZEILE um 1 erhöht, damit in der nächsten Bildschirmzeile ein neuer Rateversuch unternommen werden kann. Das Programm fährt dann mit Zeile 280 fort. Dies geschieht jedoch nicht, wenn nach dem Versuch die Variable ZEILE größer als 22 ist, da dann eine Bildschirmdarstellung in der vorgegebenen Weise nicht mehr möglich ist. In diesem Fall fährt das Programm mit Zeile 470 fort, d.h. die Geheimzahl wird ausgegeben, das Spiel wird als verloren gewertet, und der Spieler wird über einen eventuellen Neustart befragt.

---

(420) DISPLAY AT(23,1)BEEP:"DU GEWINNST:";ZEILE-1;"VERSUCHE" <ENTER>

---

Bei kompletter Übereinstimmung zwischen Versuchs- und Geheimzahl werden der Gewinnfall und die Anzahl der benötigten Versuche in der 23. Zeile angezeigt.

## Programmierung

---

<FCTN-8>

---

Durch REDO wird die Zeile 420 wieder auf den Bildschirm zurückgeholt.

---

<FCTN-S>,<FCTN-D>,<FCTN-1>,<FCTN-2>

---

Mit Hilfe der angegebenen Funktionstasten wird Zeile 420 an den unterstrichenen Stellen überschrieben und in Zeile 430 abgeändert.

---

```
430 DISPLAY AT(24,1)BEEP:"NEUES SPIEL(J/N)? J"                    <ENTER>
```

---

In der 24. Bildschirmzeile wird "NEUES SPIEL(J/N)? J" angezeigt.

---

```
>NUM 440                                                          <ENTER>
```

---

Automatische Zeilennumerierung in Zehnerschritten, beginnend mit Zeile 440

---

```
(440) ACCEPT AT(24,19)SIZE(-1)BEEP VALIDATE("JN"):X$              <ENTER>
```

---

Die Eingabe der ausschließlich zugelassenen Daten "J" oder "N" erfolgt in der 24. Zeile, 19.Spalte. Wegen des negativen Ausdrucks von SIZE steht an dieser Stelle das "J" aus der Programmzeile 430, das durch Drücken von <ENTER> sofort als Eingabewert übernommen wird. Will der Spieler nicht weiterspielen, muß vor dem Drücken von <ENTER> das "J" mit "N" überschrieben werden. Die Eingabebereitschaft wird durch einen Ton angezeigt.

---

```
(450) IF X$="J" THEN 190                                          <ENTER>
```

---

Wird die Fragestellung in Zeile 430 durch Eingabe von "J" in Zeile 440 bejaht, beginnt mit Zeile 190 ein neuer Durchgang mit der Wahl einer neuen Geheimzahl.

---

```
(460) STOP                                                        <ENTER>
```

---

Andernfalls endet das Programm mit Zeile 460, der Computer meldet sich wieder mit * READY *.

---

```
(470) DISPLAY AT(23,1)BEEP:"DIE ZAHL IST:" !VERLOREN,AUFGABE,NEUSTART
                                                                  <ENTER>
```

---

Zeigt in der 23. Zeile "DIE ZAHL IST:" an, wenn die Geheimzahl bis zur 22. Bildschirmzeile nicht erraten oder durch Eingabe von "X" vorher aufgegeben wurde.

---

```
(480) FOR G=1 TO ZAHL                                             <ENTER>
```

---

Beginn einer Schleife, in der die einzelnen Ziffern der Geheimzahl hintereinander angezeigt und somit als komplette Geheimzahl ausgegeben werden.

---

```
(490) DISPLAY AT(23,14+G):ZAHL$(G)                                <ENTER>
```

---

Anzeige der Geheimzahl-Ziffern in Bildschirmzeile 23, beginnend mit Spalte 15.

---

```
(500) NEXT G                                                      <ENTER>
```

---

Ende der Schleife zur Anzeige der Geheimzahl.

## Programmierung

```
                                                                    <ENTER>
-------------------------------------------------------------------------------
Der NUMBER-Modus (automatische Zeilennumerierung) wird verlassen.
-------------------------------------------------------------------------------
430                                                 <FCTN-X>(oder <FCTN-E>)
-------------------------------------------------------------------------------
```

Mit Hilfe der angegebenen Pfeiltasten wird die Zeile 430 auf den Bildschirm gebracht, der Computer befindet sich jetzt im EDIT-Modus.

```
-------------------------------------------------------------------------------
```

Die Zeile 430 wird durch <ENTER> wieder eingegeben, damit sie im Programm nicht verloren geht.

```
-------------------------------------------------------------------------------
                                                                    <FCTN-8>
-------------------------------------------------------------------------------
```

Durch REDO wird die Zeile 430 erneut auf den Bildschirm gebracht.

```
-------------------------------------------------------------------------------
                                                    <FCTN-S>,<FCTN-D>
-------------------------------------------------------------------------------
```

Mit Hilfe der angegebenen Funktionstasten wird Zeile 430 in Zeile 510 abgeändert.

```
-------------------------------------------------------------------------------
510 DISPLAY AT(24,1)BEEP:"NEUES SPIEL(J/N)? J"                      <ENTER>
-------------------------------------------------------------------------------
```

Dasselbe, d.h. Duplizieren von schon im Programm befindlichen Zeilen geschieht im folgenden mit der Zeile 440:

```
-------------------------------------------------------------------------------
520 ACCEPT AT(24,19)SIZE(-1)BEEP VALIDATE("JN"):X$
-------------------------------------------------------------------------------
```

und mit Zeile 450:

```
-------------------------------------------------------------------------------
530 IF X$="J" THEN 130
-------------------------------------------------------------------------------
```

Dem Starten des Programms sollte eine Überprüfung vorangehen, indem das eingegebene Programm sorgfältig mit dem am Anfang vollständig aufgeführten Programmlisting verglichen wird (mit Hilfe des LIST-Befehls).

Nun kann das Programm durch Eingabe von "RUN" gestartet werden (<ENTER> nicht vergessen!). Wählen Sie z.B. für die Zahlengröße "4" Stellen und alle "10" Ziffern als möglich für die Geheimzahl. Das Erraten dieser Geheimzahl in 6 Versuchen ist ausgezeichnet, 8 Versuche sind immer noch sehr gut.

Läuft Ihr Programm problemlos, dann sollten Sie es zur Wiederverwendung speichern, entweder auf Cassette ("SAVE CS1") oder auf Diskette ("SAVE DSK1.ZCODE"). Selbstverständlich können Sie beim Speichern auf Diskette auch andere Programmnamen wählen. Wenn Sie nach dem Speichern "NEW" eingeben, dann wird das Programm im Arbeitsspeicher des Computers gelöscht. Sie können dann ein neues Programm eingeben oder mit dem BYE-Befehl zum Titelbild zurückkehren, wenn Sie z.B. den Modul wechseln wollen.

Mit "OLD CS1" bzw. "OLD DSK1.ZCODE" können Sie das Programm jederzeit wieder in den Computer laden (auf EXTENDED BASIC-Modul achten!) und nach Eingabe von "RUN" erneut versuchen, die Geheimzahl zu enträtseln!

## Teil B

# B E F E H L S L I S T E

Die mit (D) gekennzeichneten Befehle können nur direkt ausgeführt werden, d.h. sie dürfen n i c h t (mit Programmzeilennummern versehen) in Programmen stehen. Die mit (U) gekennzeichneten Befehle sind Unterprogrammnamen und können nur in Verbindung mit dem Schlüsselwort CALL verwendet werden:

# ABS

Format:

ABS(numerischer Ausdruck)

Beschreibung:

Die ABS-Funktion gibt den absoluten Wert des numerischen Ausdrucks an. Für die Berechnung des numerischen Ausdrucks werden die üblichen Regeln angewendet. Ist der Wert des numerischen Ausdrucks positiv, so ergibt sie ABS-Funktion den Wert selbst, bei negativem numerischem Ausdruck den negativen Wert (also eine positive Zahl). Ist der numerische Ausdruck Null, so ergibt die ABS-Funktion den Wert 0. Das Ergebnis der ABS-Funktion ist also immer eine nichtnegative Zahl.

Beispiele:

In Programmzeile 100 wird 36.7 ausgegeben, in Programmzeile 120 ebenfalls.

```
100 PRINT ABS(36.7)
120 PRINT ABS(-36.7)
```

In Programmzeile 140 wird der numerischen Variablen BETRAG der Absolutwert der Differenz A-B zugeordnet.

```
140 BETRAG=ABS(A-B)
```

## ACCEPT

Format:

```
ACCEPT [[AT(Zeile,Spalte)] [VALIDATE(Typbezeichnung ,... )][BEEP]
       [ERASE ALL] [SIZE(numerischer Ausdruck)]:] Variable
```

Beschreibung:

Der ACCEPT-Befehl unterbricht die Programmausführung, bis der Variablenwert über die Tastatur eingegeben ist. Zu diesem Befehl sind viele Optionen vorgesehen (s.u.), wodurch er weitaus vielseitiger ist als der INPUT-Befehl: Daten können an jeder beliebigen Bildschirmstelle eingegeben werden, die Bereitschaft zur Dateneingabe kann durch einen Ton (BEEP) angezeigt und der Bildschirm vor der Dateneingabe gelöscht werden (ERASE ALL). Die einzugebenden Zeichen lassen sich hinsichtlich ihrer Anzahl (SIZE) und ihrer Art (VALIDATE) einschränken.

Optionen:

Die Reihenfolge der aufgeführten Optionen nach ACCEPT ist beliebig.

**AT(Zeile,Spalte)**

plaziert den Beginn der Eingabe an die durch (Zeile,Spalte) bestimmte Bildschirmstelle. Dabei sind die Zeilen 1 - 24 und die Spalten 1 - 28 möglich, wobei die 1. Spalte der 3. Spalte in den VCHAR-, HCHAR- und GCHAR-Unterprogrammen entspricht.
Bei fehlendem AT erfolgt die Eingabe in der 24. Zeile und 1. Spalte des Bildschirms.

**VALIDATE(Typbezeichnung)**

läßt bei der Eingabe nur bestimmte Zeichen zu, die durch die Typbezeichnung definiert werden. Dabei bedeuten:

| Typbezeichnung | erlaubte Zeichen |
|---|---|
| UALPHA | Großbuchstaben |
| DIGIT | Ziffern (0 ... 9) |
| NUMERIC | Ziffern,".","+","-" und "E" |
| "...." | jedes Zeichen zwischen den beiden Anführungszeichen |

Mehrere Typbezeichnungen hintereinander - durch Kommata getrennt - sind möglich.

**BEEP**

erzeugt einen Ton als Bereitschaftsanzeige vor einer Dateneingabe.

**ERASE ALL**

löscht den Bildschirm vor der Dateneingabe.

**SIZE(numerischer Ausdruck)**

begrenzt die Zahl der einzugebenden Zeichen auf die im numerischen Ausdruck angegebene Größe. Bei positivem Wert des numerischen Ausdrucks wird das Bildschirmfeld, in das die Daten eingegeben werden, vor der Eingabe gelöscht, bei negativem Wert nicht. Letzteres erlaubt, schon im Bildschirmfeld befindliche Daten mit <ENTER> zu übernehmen.
Bei fehlendem SIZE wird die Eingabezeile von der entsprechenden Spalte bis zum Zeilenende gelöscht.

## ACCEPT

Beispiele:

In Programmzeile 100 erfolgt die Eingabe des numerischen Variablenwertes Y in der 5. Zeile und der 7. Spalte des Bildschirms.

```
100 ACCEPT AT(5,7):Y
```

In der Programmzeile 120 wird als Eingabewert der Stringvariablen R$ nur "J", "N" oder "JN" zugelassen.

```
120 ACCEPT VALIDATE("JN"):R$
```

Vor der Eingabe des Wertes der numerischen Variablen B wird in Programmzeile 140 der Bildschirm gelöscht.

```
140 ACCEPT ERASE ALL:B
```

In Programmzeile 160 sind für die Eingabe des Stringvariablenwertes von X$ alle Ziffern und die Buchstaben A und B zugelassen. Dabei kann die Länge des Eingabestrings bis zu L betragen. Die Eingabe erfolgt in der Zeile Z und der Spalte S des Bildschirms, vor der Eingabe ertönt ein Signal.

```
160 ACCEPT AT(Z,S)SIZE(L)BEEP
    VALIDATE(DIGIT,"AB"):X$
```

Beispielprogramm:

Das Programm zeigt ein typisches Anwendungsbeispiel für den ACCEPT-Befehl. In ihm können bis zu 20 Namen mit Telefonnummern (als String mit "/") eingegeben werden, die nach der Eingabe alle zusammen auf dem Bildschirm ausgegeben werden.

Mit dem "Hardcopyprogramm" (vgl. GCHAR-Unterprogramm) kann der Bildschirminhalt anschließend ausgedruckt werden.

Das Programm endet mit <FCTN-4> (CLEAR).

```
100 REM ***** TELEFONREGISTER *****
110 DIM NAME$(20),TEL$(20)
120 DISPLAY AT(5,1)ERASE ALL:"NAME:"
130 DISPLAY AT(7,1):"TELEFON:"
140 DISPLAY AT(23,1):"<!> BEENDET DIE EINGABE"
150 FOR I=1 TO 20
160 ACCEPT AT(5,7)VALIDATE(UALPHA,"!")BEEP SIZE(13):NAME$(I)
170 IF NAME$(I)="!" THEN 210
180 ACCEPT AT(7,10)VALIDATE(DIGIT,"/")SIZE(12):TEL$(I)
190 DISPLAY AT(7,10):"      "
200 NEXT I
210 CALL CLEAR
220 DISPLAY AT(1,1):"NAME","TELEFON"
230 FOR J=1 TO I-1
240 DISPLAY AT(J+2,1):NAME$(J),TEL$(J)
250 NEXT J
260 GOTO 260
```

## ASC

Format:

```
ASC(String-Ausdruck)
```

Beschreibung:

Die ASC-Funktion ergibt den ASCII-Zeichencode des ersten Zeichens des String-Ausdrucks an. Eine Liste der ASCII-Zeichencodes für jedes Zeichen des Standardzeichensatzes befindet sich im Anhang. Die ASC-Funktion ist die Umkehrung der CHR$-Funktion.

Beispiele:

In Programmzeile 100 wird 67 ausgegeben.

```
100 PRINT ASC("C")
```

In Programmzeile 120 wird der numerischen Variablen EINS der Wert 49 zugewiesen.

```
120 EINS=ASC("1")
```

In Programmzeile 140 wird die Zahl 71 auf dem Bildschirm angezeigt.

```
140 DISPLAY ASC("GUTEN TAG")
```

# ATN

Format:

ATN(numerischer Ausdruck)

Beschreibung:

Die ATN-Funktion ergibt den Winkel (im Bogenmaß), dessen Wert gleich dem Tangens des numerischen Ausdrucks ist. Der numerische Ausdruck wird nach den üblichen Regeln berechnet. Der Wert aus der ATN-Funktion liegt im Bereich zwischen -PI/2 und PI/2 (Hauptwert). Die ATN-Funktion ist in diesem Bereich die Umkehrung der TAN-Funktion.

Will man den Winkel nicht im Bogenmaß sondern im Gradmaß haben, so muß der Wert der ATN-Funktion mit

180/PI im EXTENDED BASIC und
45/ATN(1) im TI BASIC

multipliziert werden.

Beispiele:

Programmzeile 100 liefert den Wert 0.

```
100 PRINT ATN(0)
```

In Programmzeile 120 erhält die Variable ALPHA den Wert 45 zugewiesen.

```
120 ALPHA=ATN(1)*180/PI
```

## BREAK

Format:

BREAK[Zeilennummern-Liste]

Beschreibung:

BREAK kann als Direktbefehl oder innerhalb eines Programms verwendet werden, wobei dies meist im Rahmen einer Fehlersuche geschieht, z.B. durch Ausdrucken von Variablenwerten an einer durch den BREAK-Befehl verursachten Stoppstelle (breakpoint), um herauszufinden, wie die Daten während des Programmablaufs verarbeitet werden.

Bei Verwendung von BREAK als Direktbefehl ist die Zeilennummern-Liste, die auch aus nur einer Zeilennummer bestehen kann, erforderlich, bei Verwendung als Programmbefehl ist sie möglich (optional). Wird BREAK als Programmbefehl mit Zeilennummern-Liste verwendet, dann wird der Programmablauf jedesmal vor der betreffenden Zeile unterbrochen. Fehlt die Zeilennummern-Liste, so stoppt der Programmablauf unmittelbar nach dem BREAK-Befehl. Die Stoppstelle kann nur durch Löschen des betreffenden BREAK-Befehls aufgehoben werden, auch bei Verwendung von ON BREAK NEXT (vgl. ON BREAK-Befehl). Bei Verwendung von BREAK als Direktbefehl wird die Stoppstelle dagegen automatisch beim nächsten Programmablauf aufgehoben. Enthält das Programm nicht eine in der Zeilennummern-Liste angegebene Programmzeile, so erscheint die Fehlermeldung BAD LINE NUMBER (falsche Zeilennummer), und der Programmablauf wird sofort unterbrochen.

Wird der Programmablauf mit einem BREAK-Befehl unterbrochen, so meldet der Computer * BREAKPOINT IN ### (Stoppstelle in Zeile ###). Der blinkende Positionszeiger zeigt dann an,daß ein neuer Direktbefehl eingegeben werden kann. Die im Programmablauf verarbeiteten Variablenwerte werden durch eine solche Stoppstelle nicht verändert, aber alle durch das CHAR-Unterprogramm neudefinierten Zeichen im Bereich der Codes 32 bis 95 werden in ihre ursprüngliche Form zurückgesetzt und erhalten wieder die Standardfarben. Desweiteren werden alle im Programm auftretenden Sprites gelöscht, und der Vergrößerungsfaktor für Sprites wird auf 1 gesetzt (vgl. CHAR-, SPRITE- und MAGNIFY-Unterprogramm).

Stoppstellen können auch durch den UNBREAK-Befehl (s.d.) aufgehoben werden. Wenn eine Behandlung von Stoppstellen nicht durch den ON BREAK-Befehl (s.d.) im Programm selbst geregelt ist, kann eine Fortsetzung des Programmablaufs auch durch Eingabe des CON TINUE -Befehls erreicht werden. Dies ist nur dann nicht möglich, wenn das Programm während einer Stoppstelle editiert wurde. In diesem Fall wird bei Verwendung des CON TINUE - Befehls die Fehlermeldung *CAN'T CONTINUE (Programmfortsetzung nicht möglich) ausgegeben.

Während des Programmablaufs kann eine Stoppstelle auch durch die spezielle Tastenfunktion CLEAR (durch Betätigung von <FCTN-4>) erzeugt werden. Dabei treten dieselben Wirkungen wie oben beschrieben ein.

Beispiel:

Durch Programmzeile 100 werden zwei Stoppstellen - vor der Ausführung der Programmzeilen 140 und 200 - erzeugt, die nur dadurch aufgehoben werden können, daß Programmzeile 100 gelöscht wird.

```
100 BREAK 140,200
```

## BREAK

Beispielprogramm:

Das Programm zeigt selbsterklärend die Wirkung des BREAK-Befehls insbesondere auch auf selbstdefinierte Zeichen (in Programmzeilen 210 und 240). Im 2. Durchlauf werden die mit dem Direktbefehl gesetzten Stoppstellen aufgehoben.

Geben Sie vor dem Programmstart <BREAK 280> ein!

```
100 REM ***** STOPPSTELLEN *****
110 FOR DURCHL=1 TO 2
120 CALL CLEAR
130 IF DURCHL=2 THEN DISPLAY AT(18,9)BEEP:"2. DURCHLAUF!"
140 GOSUB 430
150 FOR F=9 TO 11
160 CALL COLOR(F,13,1)
170 NEXT F
180 BEF$="programmbefehl" :: GOSUB 310
190 BREAK
200 CALL CLEAR :: GOSUB 430
210 CALL CHAR(42,"995A3C3C3C3C2424")
220 ZL=14 :: ZCH=42 :: GOSUB 400
230 CALL HCHAR(16,12,42,10)
240 CALL CHAR(128,"FF7E3C18183C7EFF")
250 CALL HCHAR(20,12,128,10)
260 ZL=18 :: ZCH=128 :: GOSUB 400
270 BEF$="direktbefehl" :: GOSUB 310
280 NEXT DURCHL
290 CALL CLEAR
300 END
310 REM ***** BEFEHLSANZEIGE *****
320 FOR VERZOEG=1 TO 500 :: NEXT VERZOEG
330 FOR Z=1 TO 8
340 DISPLAY AT(8,15)SIZE(14):""
350 DISPLAY AT(8,15)BEEP SIZE(14):BEF$
360 FOR VERZOEG=1 TO 200 :: NEXT VERZOEG
370 NEXT Z
380 FOR VERZOEG=1 TO 500 :: NEXT VERZOEG
390 RETURN
400 REM ***** ZEICHENANZEIGE *****
410 DISPLAY AT(ZL,9)BEEP:"ZEICHEN: ";ZCH
420 RETURN
430 DISPLAY AT(6,1)BEEP:"STOPPSTELLE PROGRAMMIERT MIT"
440 DISPLAY AT(7,1):"'BREAK' ALS"
450 DISPLAY AT(10,1):"FORTSETZUNG MIT <CON(TINUE)>"
460 RETURN
```

## BYE

Format:

BYE

Beschreibung:

Mit dem BYE-Befehl wird der Modus (TI-BASIC oder EXTENDED BASIC), in dem sich der Computer gerade befindet, abgeschaltet. Nach der Ausführung des BYE-Befehls erscheint das Standardbild auf dem Bildschirm. Zuvor werden alle noch offenen Dateien abgeschlossen sowie das eventuell im Computer befindliche Programm und alle Variablenwerte gelöscht.

Es ist besser, den augenblicklichen Modus mit dem BYE-Befehl zu verlassen als mit <FCTN-=> (QUIT), da mit QUIT offene Dateien nicht abgeschlossen werden, was zum Verlust von Daten auf externen Geräten führen kann.

## CALL

Format:

CALL Unterprogrammname[(Variablenliste)]

Beschreibung:

Mit Hilfe des CALL-Befehls wird ein Unterprogramm mit dem entsprechenden Namen aufgerufen. Das Unterprogramm ist entweder im Umfang des EXTENDED BASIC (oder des TI-BASIC) enthalten (z.B. CLEAR) oder mit Hilfe des SUB-Befehls selbst erstellt (s.d.). Der CALL-Befehl kann direkt ausgeführt werden oder in einem Programm enthalten sein. Selbst erstellte Unterprogramme können nur aus dem Hauptprogramm heraus aufgerufen werden, in dem sie eingebettet sind. Im Programmablauf wird nach der Abarbeitung des Unterprogramms der nächste Befehl im Hauptprogramm, der dem CALL-Befehl folgt, ausgeführt.

Die Variablenliste ist entsprechend dem aufgerufenen Unterprogramm definiert, wobei einige Unterprogramme auch keine Variablen benötigen. Jedes im EXTENDED BASIC enthaltene Unterprogramm ist in diesem Handbuch unter seinem Namen beschrieben. Dies sind:

| | | | |
|---|---|---|---|
| CHAR | | LINK | SAY |
| CHARPAT | GCHAR | LOAD | SCREEN |
| CHARSET | | LOCATE | SOUND |
| CLEAR | HCHAR | | SPGET |
| COINC | | MAGNIFY | SPRITE |
| COLOR | INIT | MOTION | |
| | | | VCHAR |
| DELSPRITE | JOYST | PATTERN | VERSION |
| DISTANCE | | PEEK | |
| | KEY | POSITION | |
| ERR | | | |

Beispielprogramme:

S. unter den verschiedenen Unterprogrammen (z.B. dem CLEAR-Unterprogramm).

## CHAR-Unterprogramm

Format:

```
CALL CHAR(Zeichencode,"Mustercode")
```

Beschreibung:

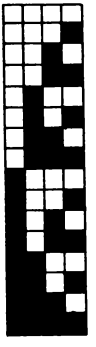
Mit dem CHAR-Unterprogramm wird die äußere Form (Muster) von Zeichen, aber auch Sprites (vgl. SPRITE-Unterprogramm) definiert, wobei sowohl die Zeichen des Standard-Zeichensatzes (Codes 32 - 127) als auch die frei definierbaren Zeichen (Codes 128 - 143 im EXTENDED BASIC, Codes 128 - 159 im TI BASIC) mit Hilfe der sogenannten Mustercodierung (vgl. auch Anhang) in ihrem Aussehen nach eigenen Wünschen bestimmt werden können.

Zeichencode ist ein numerischer Ausdruck, dessen ganzzahliger Wert den Code des Zeichens angibt, das definiert werden soll. Der Wert des Zeichencodes muß im EXTENDED BASIC zwischen 32 und 143 einschließlich liegen.

Der Mustercode ist ein normalerweise 16 Zeichen umfassender String-Ausdruck, mit dessen Hilfe die 64 Einzelpunkte, aus denen ein Zeichen in der Größe einer Schreibstelle besteht, codiert werden, wobei jeweils 4 dieser Einzelpunkte (= 1 Block) durch ein Zeichen des Mustercodes dargestellt sind (64/4 = 16). Der Mustercode beginnt mit dem Codezeichen für den linken Block der 1. Reihe und endet mit dem rechten Block der 8. Reihe, da die Reihen von links nach rechts und von oben nach unten beschrieben werden.

Zur Zeichendefinition ist es sinnvoll, sich das aus 8 mal 8 Positionen bestehende und in linke und rechte Blöcke unterteilte Rasterfeld aufzuzeichnen und das gewünschte Muster in diesem Rasterfeld zu entwerfen.

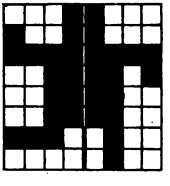
Die Codezeichen für die einzelnen Blöcke sind der nebenstehenden Darstellung zu entnehmen. Sie entsprechen der hexadezimalen Darstellung des Binärcodes des jeweiligen Blocks.



| BLOCK | Binärcode | Codezeichen |
|---|---|---|
| | 0000 | 0 |
| | 0001 | 1 |
| | 0010 | 2 |
| | 0011 | 3 |
| | 0100 | 4 |
| | 0101 | 5 |
| | 0110 | 6 |
| | 0111 | 7 |
| | 1000 | 8 |
| | 1001 | 9 |
| | 1010 | A |
| | 1011 | B |
| | 1100 | C |
| | 1101 | D |
| | 1110 | E |
| | 1111 | F |

Das im Beispielprogramm verwendete Muster für ein "Männchen" wird dann beispielsweise durch den folgenden Mustercode hervorgerufen:

Mustercode: 1898FF3D3C3CE404

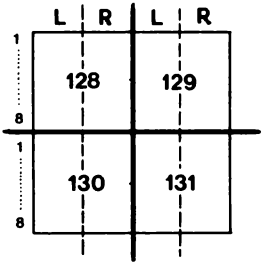


| L Blöcke R | Codezeichen |
|---|---|
| | 18 |
| | 98 |
| | FF |
| | 3D |
| | 3C |
| | 3C |
| | E4 |
| | 04 |

## CHAR-Unterprogramm

Mit dem CHAR-Unterprogramm wird das Zeichen lediglich definiert, zur Darstellung auf dem Bildschirm müssen andere Unterprogramme (wie HCHAR oder VCHAR) oder PRINT- bzw. DISPLAY-Befehle angewandt werden.

Ist das Zeichen in Schreibstellengröße für die grafische Darstellung zu klein, müssen mehrere Rasterfelder, wie die nebenstehende Skizze zeigt, kombiniert werden. Für das Gesamtrasterfeld sind dann mehrere Zeichen (z.B. mit den Codes 128, 129, 130, 131) zu definieren, die im Programmablauf hintereinander, z.B. durch eine Unterprogrammschleife, abgerufen werden.




Die ausgefüllten (eingeschalteten) Punkte des Rasterfeldes nehmen die Vordergrundfarbe, die leer belassenen (ausgeschalteten) Punkte die Hintergrundfarbe (vgl. COLOR-Unterprogramm) an.

Ein weniger als 16 Stellen umfassender Mustercode wird vom Computer mit dem Mustercode-Zeichen "0" bis zur 16. Stelle aufgefüllt. Das Leerzeichen z.B. ist also bereits durch den Mustercode "0" definiert, da die fehlenden Mustercode-Stellen ebenfalls mit "0" aufgefüllt werden. Ein Mustercode kann auch mehr als 16 Stellen umfassen. Dann werden entsprechend mehr Zeichen definiert, wobei ebenfalls fehlende Mustercode-Zeichen bis zur nächstfolgenden 16er-Länge mit "0" aufgefüllt werden. Es gilt also:

| Stellenzahl im Mustercode | Anzahl der definierten Zeichen |
|---|---|
| 1 - 16 | 1 |
| 17 - 32 | 2 |
| 33 - 48 | 3 |
| 49 - 64 | 4 |
| 65 und mehr | 4 (Mehrzeichen werden ignoriert) |

Reicht der Speicherplatz für die Zeichendefinition nicht aus, erscheint Meldung MEMORY FULL IN ### (Speicher belegt in Zeile Nummer ###) und der Programmablauf wird unterbrochen.

Das CHAR-Unterprogramm ist die Umkehrung des CHARPAT-Unterprogramms. Beachten Sie die Veränderungen, die voraus- oder neudefinierte Zeichen beim durch den BREAK-Befehl hervorgerufenen Programmstopp (breakpoint) oder beim Programmende erfahren.

---

Beispiele:

Programmzeile 100 definiert das vordefinierte Plus-Zeichen (Zeichencode 43) in einen "Balken" um

```
100 CALL CHAR(43,"0000FFFFFFFF0000")
```

Programmzeile 120 leistet das gleiche für ein frei definierbares Zeichen (Zeichencode 128), wobei die fehlenden Stellen des Mustercodes durch "0" ergänzt werden.

```
120 CALL CHAR(128,"000FFFFFFFFF")
```

## CHAR-Unterprogramm

Beispielprogramm:

In der Bildschirmmitte wird das o.a. "Männchen" dargestellt. Nach kurzer Zeit (Verzögerungsschleife) wird an dieselbe Stelle ein Zeichen mit ähnlichem Muster, ein zweites "Männchen", plaziert, dann erscheint wieder das erste usw. Damit entsteht insgesamt der Eindruck eines "tanzenden Männchens".

Das Programm endet mit <FCTN-4> (CLEAR).

```
100 REM ***** TANZMAENNCHEN *****
110 CALL CLEAR
120 CALL CHAR(96,"1898FF3D3C3CE404")
130 CALL CHAR(97,"1818FFBC3C3C2720")
140 CALL VCHAR(12,16,96)
150 FOR VERZOEG=1 TO 50 :: NEXT VERZOEG
160 CALL VCHAR(12,16,97)
170 FOR VERZOEG=1 TO 50 :: NEXT VERZOEG
180 GOTO 140
```

## CHARPAT-Unterprogramm

Format:

```
CALL CHARPAT(Zeichencode, String-Variable[,...])
```

Beschreibung:

Das CHARPAT-Unterprogramm weist der angegebenen String-Variablen den 16-stelligen Mustercode zu, durch den das Zeichen mit dem bezeichneten Zeichencode in seiner Form (Muster) definiert ist. Der CHARPAT-Befehl ist die Umkehrung des CHAR-Befehls.

Beispiel:

In Programmzeile 100 erhält die String-Variable MUSTER$ den String "0010101010001000" als Mustercode des Zeichens mit dem ASCII-Zeichencode 33 zu. Es handelt sich dabei um den Mustercode des Ausrufezeichens.

```
100 CALL CHARPAT(33,MUSTER$)
```

## CHARSET-Unterprogramm

Format:

```
CALL CHARSET
```

Beschreibung:

Das CHARSET-Unterprogramm setzt die Zeichen mit den Zeichencodes 32 bis 95 in ihre ursprüngliche, vorausdefinierte Form (Muster) und ihre Standardfarben zurück, was normalerweise nicht geschieht, wenn ein Programm durch ein anderes aufgerufen wird.

## CHR$

Format:

CHR$(numerischer Ausdruck)

Beschreibung:

Die CHR$-Funktion liefert das Zeichen, das dem durch den Wert des numerischen Ausdrucks bestimmten Speicherinhalt des Zeichensatzes entspricht. Im Normalfall ist dies ein ASCII-Zeichen. Der numerische Ausdruck wird zunächst nach den üblichen Regeln berechnet, wobei gegebenenfalls auf eine ganze Zahl gerundet wird. Eine Liste der ASCII-Zeichencodes des Standard-Zeichensatzes befindet sich im Anhang. Liegt der Wert des numerischen Ausdrucks zwischen einschließlich 32 und 127, so erhält man ein Standardzeichen. Wenn der Wert im EXTENDED BASIC zwischen einschließlich 128 und 143 liegt und für diesen Wert ein spezielles Zeichen definiert wurde, so wird dieses Zeichen angegeben. Im TI-BASIC liegt dieser Bereich zwischen 128 und 159 (s. auch "Vorüberlegungen"). Bei Werten des numerischen Ausdrucks kleiner 0 oder größer 32767 erscheint die Fehlermeldung BAD VALUE, und der Programmablauf wird unterbrochen. Die CHR$-Funktion ist die Umkehrung der ASC-Funktion.

Beispiele:

In Programmzeile 100 wird H ausgegeben.

```
100 PRINT CHR$(72)
```

In Programmzeile 120 wird der Stringvariablen AUSRUF$ das Zeichen "!" zugeordnet.

```
120 AUSRUF$=CHR$(33)
```

Beispielprogramm:

Das Programm erzeugt die ASCII-Zeichen der Codes 33 - 92 einschließlich ihrer Codes auf dem Bildschirm.

```
100 REM ***** ASCII-ZEICHENSATZ *****
110 CALL CLEAR
120 FOR NR=33 TO 52
130 PRINT NR;CHR$(NR);NR+20;CHR$(NR+20);NR+40;CHR$(NR+40)
140 NEXT NR
```

## CLEAR-Unterprogramm

Format:

```
CALL CLEAR
```

Beschreibung:

Das CLEAR-Unterprogramm verwendet man zum Löschen des gesamten Bildschirms Beim Aufruf des CLEAR-Unterprogramms werden alle Stellen des Bildschirms mit dem Leerzeichen (ASCII-Code 32) belegt. Ist das Leerzeichen mit Hilfe des CHAR-Unterprogramms umdefiniert worden, so wird der gesamte Bildschirm mit dem neudefinierten Zeichen gefüllt.

Beispielprogramm:

In dem Programm wird zunächst durch Zeile 110 der Bildschirm gelöscht. Anschließend werden auf den Bildschirm lauter "*" ausgegeben, die durch den 2. CALL CLEAR-Befehl in Zeile 140 nun mit Dreiecken überschrieben werden, da das Zeichen mit dem ASCII-Code 32 in Zeile 130 umdefiniert wurde.

Danach wird der Bildschirm wieder mit "*" gefüllt, und das Programm endet.

```
100 REM ***** BEISPIELE ZUM CLEAR-UNTERPROGRAMM *****
110 CALL CLEAR
120 GOSUB 170
130 CALL CHAR(32,"0103070F1F3F7FFF")
140 CALL CLEAR
150 GOSUB 170
160 STOP
170 FOR I=1 TO 24*28
180 PRINT "*";
190 NEXT I
200 RETURN
```

## CLOSE

**Format:**

```
CLOSE #Dateinummer[:DELETE]
```

**Beschreibung:**

Der CLOSE-Befehl schließt eine geöffnete Datei mit der angegebenen Dateinummer ab. Die entsprechende Datei kann dann im Programm nicht mehr angesprochen werden, es sei denn, sie wird wieder mit dem OPEN-Befehl geöffnet (vgl. OPEN-Befehl). Auch ist die Dateinummer nicht länger für die im OPEN-Befehl näher bezeichnete Datei reserviert und kann nun für andere Dateien verwendet werden.

Bei dem Versuch, eine Datei mit einer Dateinummer zu schließen, die vorher nicht unter derselben Dateinummer geöffnet wurde, gibt der Computer die Fehlermeldung FILE ERROR aus und unterbricht die Programmausführung.

Zum Schutz der Dateien werden vom Computer unter bestimmten Bedingungen automatisch alle offenen Dateien geschlossen:

- wenn das Programm editiert wird,
- bei Eingabe eines der folgenden Befehle:
  BYE
  NEW
  OLD
  RUN
  SAVE
  LIST,
- bei Programmunterbrechung mit <FCTN-4> (CLEAR).

Bei Betätigung der Tasten <FCTN-=> (QUIT) werden die offenen Dateien nicht geschlossen, was zu Datenverlusten führen kann. Der Befehl QUIT sollte daher unter allen Umständen vermieden werden.

Wenn der Computer den CLOSE-Befehl für eine Datei auf einer Cassette ausführt, erscheinen auf dem Bildschirm die entsprechenden Anweisungen für die Bedienung des Cassetten-Recorders.

**Option:**

Eine Datei auf einer Diskette kann mit dem Schlüsselwort :DELETE im CLOSE-Befehl gelöscht werden. Bei anderen Geräten, wie z.B. Cassetten-Recordern, ist der Zusatz :DELETE nicht gestattet.

**Beispiele:**

S. OPEN-Befehl

## COINC-Unterprogramm

Format:

```
CALL COINC(#Sprite-Nummer1,#Sprite-Nummer2,Toleranz,numerische Variable)
CALL COINC(#Sprite-Nummer,Punktzeile,Punktspalte,Toleranz,num. Variable)
CALL COINC(ALL,numerische Variable)
```

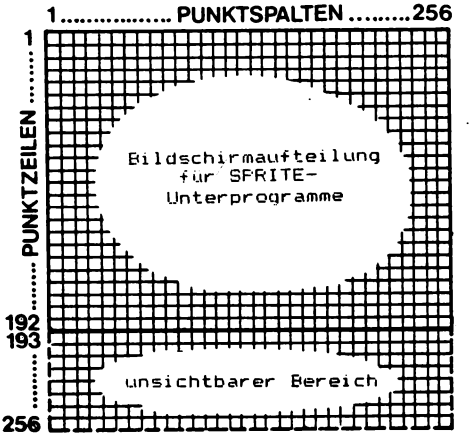
Beschreibung:

Mit dem COINC-Unterprogramm wird das etwaige Zusammentreffen zweier oder mehrerer Sprites oder eines Sprites mit einer durch Punktzeile und Punktspalte bestimmten Bildschirmposition festgestellt. Dabei wird der numerischen Variablen bei einem Zusammentreffen innerhalb einer angegebenen Toleranz der Wert -1 und andernfalls der Wert 0 zugeordnet.

Entweder werden für das festzustellende Zusammentreffen zweier Sprites mit Sprite-Nummer1 und Sprite-Nummer2 ihre Identität angegeben, oder es wird durch das Schlüsselwort ALL das etwaige Zusammentreffen zweier beliebiger Sprites festgestellt. Im letzteren Fall liegt ein Zusammentreffen immer dann vor, wenn einer oder mehrere Punkte, aus denen Sprites bestehen, dieselbe Bildschirmposition bedecken.

In den beiden anderen Fällen muß eine Toleranz (numerischer Ausdruck, der einen nach Punkten berechneten Abstand ausdrückt) angegeben werden, innerhalb dessen ein Zusammentreffen gültig sein soll. Ein Zusammentreffen liegt dann vor, wenn sich die beiden linken oberen Ecken der Sprites bzw. die linke obere Ecke eines Sprites und die angegebene Bildschirmposition in einem Abstand zueinander befinden, der innerhalb der durch die Toleranz bestimmten Grenze liegt. Je nach Größe dieser Toleranz ist es also für die Feststellung eines Zusammentreffens nicht unbedingt erforderlich, daß sich die Sprites bzw. ein Sprite und die Bildschirmposition tatsächlich in ihrer Darstellung auf dem Bildschirm überlappen, d.h. dieselben Punkte bedecken. Der Wert für die Toleranz muß zwischen 0 und 255 liegen.

Zur Festlegung der Bildschirmposition durch Punktzeilen und Punktspalten ist die nebenstehende Darstellung zu beachten.




Zur Feststellung eines Zusammentreffens mittels des COINC-Unterprogramms bedarf es allerdings einiger Voraussetzungen: die Sprites dürfen sich z.B. nicht zu schnell bewegen, denn wenn gerade ein Programmteil ausgeführt wird, in dem dieses Unterprogramm nicht aufgerufen wird, kann ein mögliches Zusammentreffen auch übersehen, d.h. nicht festgestellt werden.

# COINC-Unterprogramm

Beispielprogramm:

Im Programm werden 5 Sprites gezeigt, die sich zufällig auf dem Bildschirm bewegen. Beim Zusammentreffen zweier beliebiger Sprites ertönt aufgrund der Programmzeile 170 ein Signal.

Das Programm endet mit <FCTN-4> (CLEAR).

```
100 REM ***** ZUSAMMENTREFFEN VON SPRITES *****
110 RANDOMIZE
120 CALL CLEAR
130 CALL MAGNIFY(2)
140 FOR ANZAHL=1 TO 5
150 CALL SPRITE(#ANZAHL,ASC("*"),2,85,127,RND*20-10,RND*20-10)
160 NEXT ANZAHL
170 CALL COINC(ALL,J)
180 IF NOT J THEN 170
190 CALL SOUND(10,1000,0)
200 GOTO 170
```

## COLOR-Unterprogramm

Format:

```
CALL COLOR(#Sprite-Nummer, Vordergrund-Farbcode[,...])
CALL COLOR(Zeichengruppen-Nummer,Vordergrund-Farbcode,Hintergrund-Farbcode
```

Beschreibung:

Das COLOR-Unterprogramm ermöglicht es, die Farben für Sprites oder Zeichen einer Zeichengruppe und deren Hintergrund festzulegen. In einem COLOR-Unterprogramm-Aufruf können die Farben entweder nur von Sprites oder nur von Zeichengruppen, nicht aber beide gleichzeitig verändert werden. Das bedeutet, daß in einem CALL COLOR-Befehl entweder Sprite-Nummern (mit vorangestelltem #-Zeichen) oder Zeichengruppen-Nummern stehen.

Jedes auf dem Bildschirm dargestellte Zeichen besitzt zwei Farben: die Farbe der Punkte, die das Zeichen bilden, ist die Vordergrundfarbe; für die restlichen Punkte gilt die Hintergrundfarbe. Bei Sprites ist der Hintergrund immer transparent (Farbcode 1), so daß die Farbe des Bildschirms, die über das SCREEN-Unterprogramm bestimmt wird, immer durchscheint.

Die Farbcodes sind numerische Ausdrücke mit ganzzahligen Werten von 1 bis 16. Die nebenstehende Farbcode-Tabelle gilt für die Vordergrundfarbe und für die Hintergrundfarbe, wie auch für die Bildschirmfarbe (vgl. SCREEN-Unterprogramm).

| FARBCODE | FARBE |
|---|---|
| 1 | transparent |
| 2 | schwarz |
| 3 | mittelgrün |
| 4 | hellgrün |
| 5 | dunkelblau |
| 6 | hellblau |
| 7 | dunkelrot |
| 8 | kornblumenblau |
| 9 | mittelrot |
| 10 | hellrot |
| 11 | dunkelgelb |
| 12 | hellgelb |
| 13 | dunkelgrün |
| 14 | magentarot |
| 15 | grau |
| 16 | weiß |

Bis zum COLOR-Unterprogramm-Aufruf in einem Programm ist die Standard-Vordergrundfarbe für alle Zeichen schwarz (Farbcode 2) und die Standard-Hintergrundfarbe transparent (Farbcode 1). Die Sprites erhalten durch den entsprechenden Unterprogramm-Aufruf ihre Farbe. Durch einen BREAK-Befehl werden alle Farben der Zeichen auf die Standardfarbe zurückgesetzt.

Um das COLOR-Unterprogramm für Zeichen einer Zeichengruppe zu benutzen, muß man wissen, zu welcher Zeichengruppe das entsprechende Zeichen gehört. Eine Liste der ASCII-Zeichencodes für die Standardzeichen befindet sich im Anhang. Die Zeichengruppen-Nummern unterscheiden sich in TI BASIC und EXTENDED BASIC geringfügig voneinander. Sie sind der folgenden Tabelle zu entnehmen.

# COLOR-Unterprogramm

| ZEICHENGRUPPEN-NUMMER TI BASIC | ZEICHENGRUPPEN-NUMMER EXTENDED BASIC | ZEICHENCODE |
|---|---|---|
| | 0 | 30 - 31 |
| 1 | 1 | 32 - 39 |
| 2 | 2 | 40 - 47 |
| 3 | 3 | 48 - 55 |
| 4 | 4 | 56 - 63 |
| 5 | 5 | 64 - 71 |
| 6 | 6 | 72 - 79 |
| 7 | 7 | 80 - 87 |
| 8 | 8 | 88 - 95 |
| 9 | 9 | 96 - 103 |
| 10 | 10 | 104 - 111 |
| 11 | 11 | 112 - 119 |
| 12 | 12 | 120 - 127 |
| 13 | 13 | 128 - 135 |
| 14 | 14 | 136 - 143 |
| 15 | | 144 - 151 |
| 16 | | 152 - 159 |

Beispiele:

In Programmzeile 100 erhält die Zeichengruppe mit der Nummer 4, d.h. die Zeichen mit den Zeichencodes 56 bis 63, die Farbe dunkelblau (Farbcode 5) und der Hintergrund die Farbe hellgelb (Farbcode 12).

```
100 CALL COLOR(4,5,12)
```

In Programmzeile 120 wird die Farbe des Sprites #3 auf mittelrot (Farbcode 9) geändert. Dieses Sprite #3 ist zuvor mit dem SPRITE-Unterprogramm definiert worden.

```
120 CALL COLOR(#3,9)
```

## CONTINUE

Format:

CONTINUE
CON

Beschreibung:

Mit dem CONTINUE-Befehl kann ein Programm in seinem Ablauf fortgesetzt werden, wenn es vorher durch einen BREAK-Befehl oder durch die spezielle Tastenfunktion CLEAR (<FCTN-4>) unterbrochen worden ist. Der CONTINUE-Befehl darf mit CON abgekürzt werden.

Die Fortsetzung eines Programmablaufs kann nicht mit dem CONTINUE-Befehl erfolgen, wenn das Programm vorher editiert worden ist, d.h. Programmzeilen geändert, eingefügt oder gelöscht worden sind. Es erscheint dann die Fehlermeldung CAN'T CONTINUE (Fortsetzung nicht möglich).

Nach einer Programmunterbrechung und einer anschließenden Programmfortführung werden selbstdefinierte Zeichen nicht mehr weiter verwendet, es sei denn, es folgen neue Zeichendefinitionen mit Hilfe von CHAR- oder SPRITE-Unterprogrammen. In jeder anderen Hinsicht - vor allem was die Variablenbelegungen angeht - läuft ein Programm nach dem CONTINUE-Befehl normal weiter.

## COS

Format:

COS(numerischer Ausdruck)

Beschreibung:

Die COS-Funktion berechnet den Cosinus des numerischen Ausdrucks. Dabei entspricht der Wert des numerischen Ausdrucks einem Winkel im Bogenmaß. Der numerische Ausdruck wird zunächst nach den üblichen Regeln berechnet.

Stellt der Wert des numerischen Ausdrucks einen Winkel in der Einheit Grad dar, und es soll davon der Cosinus berechnet werden, so muß dieser Wert zuerst mit

| | |
|---|---|
| PI/180 | im EXTENDED BASIC und |
| ATN(1)/45 | im TI BASIC |

multipliziert werden, um den entsprechenden Winkel im Bodenmaß zu erhalten

Achtung:

Ist der Absolutbetrag des numerischen Ausdrucks größer als 1.5707963266375E10, so erscheint die Fehlermeldung * BAD ARGUMENT, und der Programmablauf wird unterbrochen.

Beispielprogramm:

Das Programm berechnet den Cosinus verschiedener Winkel, wobei ALPHA und BETA im Bogenmaß und GAMMA im Gradmaß gegeben ist.

```
100 REM ***** BEISPIELE ZUR COS-FUNKTION *****
110 ALPHA=.7635 !BOGENMASS
120 BETA=-8.348 !BOGENMASS
130 GAMMA=65.39 !GRADMASS
140 PRINT ALPHA,COS(ALPHA)
150 PRINT BETA,COS(BETA)
160 PRINT GAMMA;" GRAD",COS(GAMMA*PI/180)
```

# DATA

Format:

DATA Datenliste

Beschreibung:

Der DATA-Befehl erlaubt die Abspeicherung von Daten innerhalb eines Programms (programm-interne Dateien). Die Daten können numerische oder String-Konstanten sein. Alle Datenwerte werden in der Datenliste durch Kommata getrennt aufgeführt. Während der Programmausführung werden die Datenwerte den Variablen der Variablenliste des READ-Befehls zugeordnet.

DATA-Befehle können in einem Programm an beliebigen Stellen stehen. Lediglich ihre Reihenfolge ist von Bedeutung, da sie in der Reihenfolge ihrer Zeilennummern und innerhalb eines DATA-Befehls von links nach rechts den Variablen des READ-Befehls zugeordnet werden. DATA-Befehle müssen immer in einer eigenen Programmzeile stehen.

Die Typen der Konstanten der Datenliste müssen mit den Typen der entsprechenden Variablen der Variablenliste des READ-Befehls übereinstimmen. Enthält eine Datenliste aufeinanderfolgende Kommata, so nimmt der Computer die Zuweisung des Leerstrings (String ohne Zeichen) an. Da eine Zahl auch als String aufgefaßt werden kann, können Zahlen sowohl numerischen als auch String-Variablen zugeordnet werden. Bei dem Versuch, einer numerischen Variablen einen String zuzuweisen, erscheint die Fehlermeldung DATA ERROR IN ### (Datenfehler in Zeile ###), und der Programmablauf wird unterbrochen.

Werden in einem DATA-Befehl String-Konstanten gespeichert, so können diese in Anführungszeichen eingeschlossen werden. Enthalten die String-Konstanten vorangestellte oder nachgestellte Leerzeichen oder Kommata, so müssen sie in Anführungszeichen stehen. Sollen die String-Konstanten selbst in Anführungszeichen stehen, so müssen diese doppelt geschrieben werden.

---

Beispielprogramm:

Mit dem READ-Befehl in Programmzeile 130 werden den Variablen WORT$(I) nacheinander die Datenelemente der Programmzeilen 200 bis 220 zugewiesen. Dabei sind einzelne Datenelemente auch Leerstrings, da sonst die Zahl der Zuweisungen in Programmzeile 120 nicht stimmt. Fehlt z.B. das letzte Komma in Zeile 220, so erscheint die Fehlermeldung DATA ERROR IN 130.
(Vgl. auch das Beispielprogramm zum READ-Befehl!)

Das Programm endet mit <FCTN-4> (CLEAR).

```
100 REM ***** LISTE DER UNTERPROGRAMME *****
110 DIM WORT$(42)
120 FOR I=1 TO 42
130 READ WORT$(I)
140 NEXT I
150 PRINT "LISTE DER UNTERPROGRAMMNAMEN":""
160 FOR I=1 TO 21
170 PRINT WORT$(I);TAB(15);WORT$(I+21)
180 NEXT I
190 DATA CHAR,CHARPAT,CHARSET,CLEAR,COINC,COLOR,,DELSPRITE
200 DATA DISTANCE,,ERR,,GCHAR,,HCHAR,,INIT,,JOYST,,KEY,,LINK
210 DATA LOAD,LOCATE,,MAGNIFY,MOTION,,PATTERN,PEEK,POSITION,,SAY
220 DATA SCREEN,SOUND,SPGET,SPRITE,,VCHAR,VERSION,
230 GOTO 230
```

# DEF

Format:

```
DEF Funktionsname [(Variable)] = Ausdruck
```

Beschreibung:

Der DEF-Befehl ermöglicht es, Funktionen selbst zu definieren, wobei der Funktionsname eine gültige Variablenbezeichnung ist. Das bedeutet, daß bei einer String-Funktion im Gegensatz zu einer numerischen Funktion an den Funktionsnamen das $-Zeichen angefügt werden muß. Besitzt die Funktion ein Argument, das ist eine Variable, deren Wert beim Aufruf der Funktion zur Berechnung an sie übergeben wird, so ist dieses in Klammern an den Funktionsnamen anzufügen. Der Ausdruck ist je nach Funktion entweder ein numerischer Ausdruck oder ein String-Ausdruck.

Der DEF-Befehl muß in einer Zeile stehen, deren Zeilennummer niedriger ist als die, die den ersten Funktionsaufruf im Programm enthält. Der DEF-Befehl darf nicht in einem IF-THEN-ELSE-Befehl auftreten. Die Programmzeile mit dem DEF-Befehl nimmt keinen weiteren Einfluß auf den Programmablauf.

Der Funktionsaufruf innerhalb des Programms erfolgt durch Nennung des Funktionsnamens und - gegebenenfalls - einer Variable, deren Bezeichnung nicht mit der Bezeichnung des Funktionsargumentes übereinstimmen muß. Lediglich die Variablentypen (numerische oder String-Variable) müssen übereinstimmen. Der in der Funktion berechnete Wert wird dann während des Programmablaufs an die Variable, die im Funktionsaufruf bezeichnet ist, übergeben.
Ein DEF-Befehl kann auf andere Funktionen, nicht aber auf sich selbst Bezug nehmen, weder direkt (z.B.: DEF B=B*2) noch indirekt (z.B. DEF F=G :: DEF G=F). Desweiteren ist es nicht möglich, den Wert einer Funktion mit PRINT als Direktbefehl auszugeben, wenn die Speichererweiterung angeschlossen ist.

Beispiele:

In Programmzeile 100 wird eine numerische Funktion mit Namen BRUTTO definiert, die zu jeder eingegebenen Zahl 13% dazuaddiert, d.h. bei einem Funktionsaufruf BRUTTO(X) wird, wenn X vorher den Wert 100 besaß, der Wert 113 zugeordnet.

```
100 DEF BRUTTO(NETTO)=NETTO+
    .13*NETTO
```

In Programmzeile 120 wird eine Funktion mit Namen RND6 ohne Argument definiert, so daß bei einem Funktionsaufruf RND6 eine Zufallszahl zwischen 1 und 6 erzeugt wird.

```
120 DEF RND6=INT(RND*6+1)
```

In Programmzeile 140 wird mit Hilfe der angegebenen Funktion der erste Teil eines Namens bis zum Leerzeichen abgespalten.

```
140 DEF VORNAME$(NAME$)=
   SEG$(NAME$,1,POS(NAME$," ",1)-1)
```

## DELETE

Format:

DELETE Gerätebezeichnung.Dateiname

Beschreibung:

Der DELETE-Befehl ermöglicht das Löschen von Daten oder ganzen Programmen aus externen Speichern. Gerätebezeichnung und Dateiname müssen gültige String-Ausdrücke sein. Bei String-Konstanten sind Anführungszeichen zu setzen. Der DELETE-Befehl kann auch innerhalb eines Programms benutzt werden.

Dateien können auch durch die Benutzung des Wortes "DELETE" im Zusammenhang mit dem CLOSE-Befehl gelöscht werden.

Dateien auf Cässetten können nicht mit dem DELETE-Befehl gelöscht werden. Weitere Informationen sind den Bedienungsanleitungen der entsprechenden Peripherie-Geräte zu entnehmen.

Beispiel:

Der nebenstehende Befehl löscht eine Datei mit dem Dateinamen "DATEI" auf der Diskette im Diskettenlaufwerk 1.

```
DELETE "DSK1.DATEI"
```

Beispielprogramm:

Mit Hilfe des folgenden Programms kann eine Datei oder ein Programm gelöscht werden, wenn es sich auf einer Diskette im Diskettenlaufwerk 1 befindet.

Das Programm endet mit <FCTN-4> (CLEAR).

```
100 REM ***** LOESCHUNG EINES FILES AUF DISKETTE *****
110 CALL CLEAR
120 INPUT "PROGRAMM-/DATEINAME: ":NAME$
130 DELETE "DSK1."&NAME$
140 GOTO 120
```

# DELSPRITE-Unterprogramm

Format:

```
CALL DELSPRITE(#Sprite-Nummer[,...])
CALL DELSPRITE(ALL)
```

Beschreibung:

Mit dem DELSPRITE-Unterprogramm können während des Programmablaufs einzelne oder alle Sprites gelöscht werden. Sie stehen dann für das weitere Programm nicht mehr zur Verfügung und können nur durch das SPRITE-Unterprogramm wieder neu definiert werden.

Will man einzelne Sprites löschen, dann werden nach dem Nummernzeichen (#) deren jeweilige Sprite-Nummern angegeben, wobei mehrere Sprite-Nummern durch Kommata getrennt werden müssen. Sollen alle Sprites gelöscht werden, so genügt in den Klammern das Schlüsselwort ALL.

Beispiele:

Mit Programmzeile 100 wird Sprite #3 gelöscht.

```
100 CALL DELSPRITE(#3)
```

Mit Programmzeile 120 werden die beiden Sprites mit den Nummern 4 und 3*C gelöscht.

```
120 CALL DELSPRITE(#4,#3*C)
```

Mit Programmzeile 140 werden alle bis zu dieser Zeile im Programm auftretenden Sprites gelöscht.

```
140 CALL DELSPRITE(ALL)
```

Beispielprogramm:

S. SPRITE-Unterprogramm!

# DIM

Format:

```
DIM Datenfeldname(ganze Zahl 1[,ganze Zahl 2][,...][,ganze Zahl 7])[,...]
```

Beschreibung:

Der DIM-Befehl reserviert im Computer Speicherplatz für numerische und String-Datenfelder. In einem Programm kann ein Datenfeld nur einmal dimensioniert werden. Wenn dies geschieht, muß der DIM-Befehl im Programm vor jedem anderen Befehl, der das Datenfeld betrifft, erscheinen.

Bei der Dimensionierung von mehr als einem Datenfeld in einem einzigen DIM-Befehl müssen die Datenfeldnamen durch Kommata getrennt werden. Der Datenfeldname kann ein beliebiger gültiger Variablenname sein. Der DIM-Befehl darf nicht in einem IF-THEN-ELSE-Befehl stehen.

In TI-BASIC kann mit bis zu dreidimensionalen Datenfeldern gearbeitet werden, in EXTENDED BASIC mit bis zu siebendimensionalen Datenfeldern. Die Anzahl der ganzen Zahlen nach einem Datenfeldnamen zeigt die Zahl der Dimensionen des Datenfeldes an, die ganze zahl selbst gibt die maximale Anzahl der Elemente einer Dimension an. Wird ein Datenfeld nicht mit einem DIM-Befehl dimensioniert, ordnet der Computer automatisch den Wert 10 der ganzen Zahl 1 zu (und bei Bedarf ebenfalls für weitere Dimensionen). Dies gilt für jedes in einem Programm auftretende Datenfeld. Ohne Verwendung des OPTION BASE-Befehls ist der kleinste Index eines Elementes 0, so daß beispielsweise ein Feld, das mit DIM A(6) definiert ist, ein eindimensionales Feld mit 7 Elementen ist.

Nach der Ausführung des DIM-Befehls wird vom Computer automatisch für jedes Element eines String-Datenfeldes der Leerstring und für jedes numement eines numerischen Datenfeldes der Wert 0 gesetzt bis dem jeweiligen Element während des Programmablaufs ein anderer Wert zugewiesen wird. Kann der Computer ein Datenfeld mit der angegebenen Dimension nicht verarbeiten, erscheint die Meldug MEMORY FULL (Speicher voll) und das Programm wird nicht ausgeführt.

Beispiele:

Mit Programmzeile 100 reserviert der Computer in seinem Speicher Platz für 21 Elemente eines String-Datenfeldes mit Namen FELD$.

```
100 DIM FELD$(20)
```

In Programmzeile 120 wird ein eindimensionales numerisches Datenfeld von 100 Elementen und ein zweidimensionales Datenfeld von 10*20 Elementen deniert.

```
120 DIM A(99),B(9,19)
```

Beispielprogramm:

S. Beispielprogramm unter dem RND-Befehl.

# DISPLAY

Format:

```
DISPLAY [[AT(Zeile,Spalte)] [BEEP] [ERASE ALL] [SIZE(numerischer Ausdruck)];]
                                                              Variablenliste
```

Beschreibung:

Der DISPLAY-Befehl ermöglicht die Ausgabe von Daten an der durch (Zeile, Spalte) bestimmten Bildschirmstelle. Zu diesem Befehl sind viele Optionen (s.u.) vorgesehen, wodurch er weitaus vielseitiger ist als der PRINT-Befehl: Zusätzlich zur Ausgabe an einer beliebigen Bildschirmstelle kann vorher ein Ton (BEEP) erzeugt werden und/oder der Bildschirm völlig gelöscht werden (ERASE ALL).

Optionen:

| | |
|---|---|
| AT(Zeile,Spalte) | plaziert den Beginn der Ausgabe an die durch (Zeile,Spalte) bestimmte Bildschirmstelle. Dabei sind die Zeilen 1 - 24 und die Spalten 1 - 28 möglich, wobei die 1. Spalte der 3. Spalte in den VCHAR-, HCHAR- und GCHAR-Unterprogrammen entspricht. Bei fehlendem AT erfolgt die Ausgabe in der 24. Zeile und 1. Spalte des Bildschirms, genau wie beim PRINT-Befehl. |
| BEEP | erzeugt bei der Datenausgabe einen Ton. |
| ERASE ALL | löscht den Bildschirm vor der Datenausgabe. |
| SIZE(numerischer Ausdruck) | gibt eine dem Wert des numerischen Ausdrucks entsprechende Anzahl von Leerzeichen beginnend an der durch (Zeile,Spalte) bestimmten Bildschirmstelle aus. Bei fehlendem SIZE wird vor der Datenausgabe der Rest der entsprechenden Bildschirmzeile gelöscht. |

Beispiele:

| | |
|---|---|
| In Programmzeile 100 wird der Wert numerischen Variablen Y beginnend in der 5. Zeile und 7. Spalte des Bildschirms ausgegeben. | `100 DISPLAY AT(5,7):Y` |
| In Programmzeile 120 wird der Wert numerischen Variablen B in der 24. Zeile und 1. Spalte ausgegeben, zuvor wird der ganze Bildschirm gelöscht. | `120 DISPLAY ERASE ALL:B` |
| Vor der Ausgabe des Wertes der nume- Stringvariablen X$ an der durch Z und S bestimmten Bildschirmstelle werden dort L Leerzeichen ausgegeben, und es ertönt ein Signal. | `140 DISPLAY AT(Z,S)SIZE(L)BEEP:X$` |

Beispielprogramm:

S. nächste Seite!

# DISPLAY

Beispielprogramm:

In dem Programm werden an den (durch zweistellige Zahlen) bestimmten Bildschirmstellen Blocks ausgegeben, wodurch eine Figur oder ein Muster erzeugt werden kann. Die Bildschirmpositionen werden nach Eingabe über das KEY-Unterprogramm mit Hilfe der DISPLAY-Befehle in den Programmzeilen 190 und 240 auch auf dem Bildschirm ausgegeben.

Das Programm endet mit <FCTN-4> (CLEAR).

```
100 REM ***** BLOCKGRAFIK *****
110 CALL CLEAR
120 CALL COLOR(9,5,5)
130 DISPLAY AT(23,1):"EINGABE VON ZEILE UND SPALTE"
140 DISPLAY AT(24,1):"ZEILE:"
150 DISPLAY AT(24,16):"SPALTE:"
160 FOR STELLE=1 TO 2
170 CALL KEY(0,ZEILE(STELLE),STATUS)
180 IF STATUS<=0 THEN 170
190 DISPLAY AT(24,6+STELLE)SIZE(1):STR$(ZEILE(STELLE)-48)
200 NEXT STELLE
210 FOR STELLE=1 TO 2
220 CALL KEY(0,SPALTE(STELLE),STATUS)
230 IF STATUS<=0 THEN 220
240 DISPLAY AT(24,23+STELLE)SIZE(1):STR$(SPALTE(STELLE)-48)
250 NEXT STELLE
260 ZEILE1=10*(ZEILE(1)-48)+ZEILE(2)-48
270 SPALTE1=10*(SPALTE(1)-48)+SPALTE(2)-48
280 DISPLAY AT(ZEILE1,SPALTE1)SIZE(1):CHR$(96)
290 GOTO 140
```

# DISPLAY USING

**Format:**

```
DISPLAY USING String-Ausdruck:Variablen-Liste
DISPLAY USING Zeilennummer:Variablen-Liste
```

**Beschreibung:**

Der DISPLAY USING-Befehl ist genauso aufgebaut wie der DISPLAY-Befehl. Hinzu kommt die USING-Bedingung, die das Format der Daten in der Variablen-Liste angibt. Entweder gibt der String-Ausdruck das Format an, oder die Zeilennummer bezieht sich auf einen IMAGE-Befehl. Unter IMAGE ist erklärt, wie das Format definiert wird.

**Beispielprogramm:**

In den Programmzeilen 140, 160 und 190 wird mit dem String "####" das Format der auszugebenden Variablen im DISPLAY-Befehl definiert.

```
100 REM ***** ADDITION DER ZAHLEN VON 1 BIS 100 *****
110 CALL CLEAR
120 ERGEBNIS=0
130 ZAHL=1
140 DISPLAY AT(5,10):USING "####":ERGEBNIS
150 DISPLAY AT(7,12):"+"
160 DISPLAY AT(9,10):USING "####":ZAHL
170 DISPLAY AT(11,9):"------"
180 ERGEBNIS=ERGEBNIS+ZAHL
190 DISPLAY AT(13,10):USING "####":ERGEBNIS
200 ZAHL=ZAHL+1
210 IF ZAHL>100 THEN STOP
220 GOTO 140
```

## DISTANCE-Unterprogramm

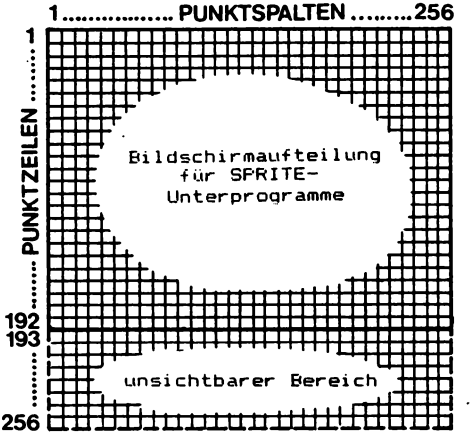
Format:

```
CALL DISTANCE(#Sprite-Nummer1,#Sprite-Nummer2,numerische Variable)
CALL DISTANCE(#Sprite-Nummer,Punktzeile,Punktspalte,numerische Variable)
```

Beschreibung:

Das DISTANCE-Unterprogramm gibt die Quadratzahl der Entfernung zwischen zwei Sprites oder einem Sprite und einer bestimmten Bildschirmposition als Wert einer numerischen Variablen aus. Die dabei zu betrachtenden Sprites werden durch ihre Sprite-Nummer angegeben. Die Position eines Sprites ist durch seine linke obere Ecke gegeben.

Die Numerierung der Punktzeilen und -spalten auf dem Bildschirm ist der nebenstehenden Darstellung zu entnehmen. Beachten Sie, daß Bildschirmpunkte mit Punktzeilen, die größer als 192 sind, nicht sichtbar sind.




Um die tatsächliche Entfernung zwischen zwei Sprites oder einem Sprite und einer Bildschirmposition zu erhalten, ist die Quadratwurzel aus der numerischen Variablen zu ziehen. Daß die ausgegebene Zahl das Quadrat der Entfernung darstellt, hängt mit der internen Berechnungsweise des Computers zusammen: Zuerst wird der Punktzeilen-Unterschied zwischen den beiden Sprites oder dem Sprite und der Bildschirmposition berechnet und - wegen einer möglichen negativen Differenz - quadriert, dann wird dasselbe mit dem Punktspalten-Unterschied gemacht, und anschließend werden die beiden Quadratzahlen addiert. Ist die Summe größer als 32767, so wird 32767 als Quadrat der Entfernung ausgegeben.

Beispiele:

In Programmzeile 100 wird der numerischen Variablen ENTFERNUNG das Quadrat der Entfernung zwischen den Sprites #3 und #4 zugeordnet, in Programmzeile 120 ist Sprite #4 durch die Bildschirmposition (18,89) ersetzt.

```
100 CALL DISTANCE(#3,#4,ENTFERNUNG)

120 CALL DISTANCE(#3,18,89,
    ENTFERNUNG)
```

## END

Format:

END

Beschreibung:

Mit dem END-Befehl wird eine Programmausführung beendet. Er kann wechselweise mit dem STOP-Befehl benutzt werden. Der END-Befehl kann zwar an beliebiger Stelle im Programm erscheinen, erhält aber in der Regel die höchste Zeilennummer und beendet damit physikalisch und logisch das Programm. Es können zwar in einem Programm mehrere END-Befehle vorkommen, doch ist es üblich, statt deren den STOP-Befehl zu verwenden, wenn ein Programm an bestimmten Stellen in seinem Ablauf unterbrochen werden soll. Letztlich ist weder in TI-BASIC noch in EXTENDED BASIC der END-Befehl unbedingt erforderlich, da der Programmablauf automatisch nach der höchsten Zeilennummer beendet wird.

## EOF

Format:

EOF(Dateinummer)

Beschreibung:

Die EOF-Funktion überprüft, ob das Ende einer Datei, die unter der angegebenen Dateinummer geöffnet worden ist, erreicht ist. Die Dateinummer ist ein numerischer Ausdruck, dessen ganzzahliger Wert vorher nach den üblichen Regeln berechnet wird.

Die EOF-Funktion kann nicht im Zusammenhang mit Dateien auf Cassetten benutzt werden. Die angegebene Datei kann auch vom Typ RELATIVE sein.

Der Wert der EOF-Funktion hängt von der Position ab, die man gerade in der Datei erreicht hat:

| POSITION | WERT |
|---|---|
| Datei-Ende nicht erreicht | 0 |
| logisches Datei-Ende | 1 |
| physikalisches Datei-Ende | -1 |

Das logische Ende einer Datei ist nach der Verarbeitung ihres letzten Datensatzes erreicht. Von einem physikalischen Ende spricht man, wenn kein Speicherplatz für weitere Datensätze in der Datei verfügbar ist.

Beispiele:

Programmzeile 100 bewirkt einen bedingten Sprung nach Zeile 200, wenn das logische oder physikalische Ende der Datei #5 erreicht ist.

```
100 IF EOF(5)<>0 THEN 200
```

Nach der Verarbeitung des letzten Datensatzes der Datei #3 wird in Programmzeile 120 die Zeile 300 ausgeführt.

```
120 IF EOF(3) THEN 300
```

## ERR-Unterprogramm

Format:

```
CALL ERR(Fehlercode,Fehlertyp[,Fehlergewicht,Zeilennummer])
```

Beschreibung:

Das ERR-Unterprogramm gibt den Fehlercode und den Fehlertyp des letzten unbereinigten Fehlers an. Ein Fehler gilt als bereinigt, wenn bei seinem Auftreten das ERR-Unterprogramm aufgerufen wurde oder ein neuer Fehler auftritt oder der Programmablauf beendet wird.

Fehlercode ist eine zwei- oder dreistellige Zahl, deren Bedeutung im Anhang erklärt ist.

Fehlertyp: Bei einer negativen Zahl wurde der Fehler bei der Programmausführung entdeckt. Im Zusammenhang mit dem Fehlercode 130 (I/O-ERROR, Ein-/Ausgabefehler) ist der Fehlertyp eine positive Zahl und gibt die Nummer der Datei an, in der der Fehler auftrat.

Wenn kein Fehler auftritt, sind alle Werte des ERR-Unterprogramms 0. Das ERR-Unterprogramm wird im Zusammenhang mit dem ON ERROR-Befehl benutzt.

Optionen:

Fehlergewicht: Hier wird immer der Wert 9 zugeordnet.

Zeilennummer: Das ist die Nummer derjenigen Programmzeile, die gerade ausgeführt wurde, als der Fehler auftrat. Sie muß nicht identisch sein mit der Programmzeile, die den Fehler verursacht hat.

Beispiele:

In Programmzeile 100 wird der numerischen Variablen A der Fehlercode und der numerischen Variablen B der Fehlertyp zugeordnet.

```
100 CALL ERR(A,B)
```

In Programmzeile 120 werden zusätzder numerischen Variablen X der Wert 9 als Fehlergewicht und der numerischen Variablen Y die Zeilennummer der Programmzeile, in der der Fehler auftrat, zugeordnet.

```
120 CALL ERR(A,B,X,Y)
```

Beispielprogramm:

S. ON ERROR-Befehl

## EXP

Format:

EXP(numerischer Ausdruck)

Beschreibung:

Die EXP-Funktion gibt den Exponentialwert ($e^X$) des Wertes des numerischen Ausdrucks an. Dabei ist e=2.718281828459. Für die Berechnung des numerischen Ausdrucks gelten die üblichen Regeln. Die EXP-Funktion ist die Umkehrung der LOG-Funktion, daher gilt

X=EXP(LOG(X)).

Beispiele:

In Programmzeile 100 wird 10 1096.633158429 ausgegeben, in Programmzeile 120 die Zahl 2.

```
100 PRINT EXP(7)
120 PRINT EXP(LOG(2))
```

In Programmzeile 140 wird der Variablen X der Exponentialwert von -3*Y zugewiesen.

```
140 X=EXP(-3*Y)
```

# FOR - TO - STEP

Format:

FOR Kontrollvariable=Startwert TO Endwert [STEP Schrittweite]

Beschreibung:

Die Ausführung der Programmbefehle zwischen dem FOR-TO-STEP-Befehl und dem NEXT-Befehl werden solange wiederholt, bis der Wert der Kontrollvariablen außerhalb des Bereichs zwischen Startwert und Endwert liegt. Man spricht in diesem Zusammenhang auch von einer "Zählschleife". Der FOR-TO-STEP-Befehl darf nicht in einem IF-THEN-ELSE-Befehl auftreten.

Die Kontrollvariable kann irgendeine numerische Variable sein. Sie fungiert als Zähler für die Schleife. Startwert und Endwert sind numerische Ausdrücke. Die Schrittweite ist eine positive oder negative Zahl. Sie gibt an, um welchen Wert die Kontrollvariable bei jeder Ausführung des NEXT-Befehls (s.d.) geändert wird. Dies geschieht solange, bis der Wert der Kontrollvariablen den Endwert über- bzw. unterschritten hat. Werden die Schrittweite und das Schlüsselwort STEP nicht angegeben, so nimmt der Computer automatisch STEP 1 an.

Der Wert der Kontrollvariablen kann innerhalb der Zählschleife durch Programmbefehle geändert werden. Dies sollte jedoch mit aller Vorsicht geschehen, um unerwartete Effekte, z.B. eine sog. Endlosschleife, aus der der Computer nur noch mit CLEAR (<FCTN-4>) oder QUIT (<FCTN-=>) "herauskommt", zu vermeiden. Die Zählschleife kann vorzeitig, d.h. vor Erreichen des Endwertes der Kontrollvariablen, mit GOTO-, GOSUB- oder IF-THEN-ELSE-Befehlen verlassen werden. Dann behält die Kontrollvariable den letzten Wert, den sie vor Verlassen der Schleife hatte, bei.

Es kann auch passieren, daß eine Zählschleife überhaupt nicht durchlaufen wird und das Programm mit dem nächsten Befehl hinter dem NEXT-Befehl fortgesetzt wird. Dies ist der Fall, wenn der Startwert bei positiver Schrittweite größer und bei negativer Schrittweite kleiner als der Endwert ist.

FOR-NEXT-Schleifen können ineinandergreifen, d.h. Zählschleifen können andere - allerdings nur vollständige - Zählschleifen enthalten sein. Dabei müssen folgende Regeln beachtet werden:

- Jeder FOR-TO-STEP-Befehl muß mit einem entsprechenden NEXT-Befehl gepaart sein.
- Für jede ineinandergeschachtelte Zählschleife sind verschiedene Kontrollvariablen zu verwenden.
- In einer Zählschleife muß die gesamte nächste Zählschleife enthalten sein, und nicht nur ein Teil.

Andernfalls wird der Programmablauf mit der Fehlermeldung CAN'T DO THAT IN ### (Durchführung in Zeile ### nicht möglich) unterbrochen.

Beispiele:

Mit Programmzeile 100 wird eine Zählschleife mit den Werten 1,3,5 und 7 für die Kontrollvariable UNGERADE durchlaufen. Anschließend hat UNGERADE den Wert 9.

```
100 FOR UNGERADE=1 TO 7 STEP 2
```

Programmzeile 120 stellt den Anfang einer Zählschleife dar, die mit den Werten ABFALL=5,4.5,...,0.5,0,-0.5,...,-2.5,-3 durchlaufen wird.

```
120 FOR ABFALL=5 TO -3 STEP -.5
```

# FOR — TO — STEP

Beispielprogramm:

Das Programm berechnet die kleinste dreistellige Zahl, die gleich der Summe der Kubikzahlen ihrer Ziffern ist. Dabei sind 3 Zählschleifen ineinander geschachtelt, deren Kontrollvariablen HUNDERTER, ZEHNER und EINER genannt wurden. Ist die gewünschte Zahl gefunden, wird sie ausgegeben und es erfolgt ein Sprung aus der innersten Schleife ans Programmende.

```
100 REM ***** BEISPIEL FUER *****
110 REM ***** INEINANDERGESCHACHTELTE ZAEHLSCHLEIFEN *****
120 FOR HUNDERTER=1 TO 9
130 FOR ZEHNER=0 TO 9
140 FOR EINER=0 TO 9
150 SUMME=100*HUNDERTER+10*ZEHNER+EINER
160 IF SUMME<>HUNDERTER^3+ZEHNER^3+EINER^3 THEN 190
170 PRINT SUMME;"=";HUNDERTER;"^3 +";ZEHNER;"^3 +";EINER;"^3"
180 GOTO 220
190 NEXT EINER
200 NEXT ZEHNER
210 NEXT HUNDERTER
220 END
```

## GCHAR-Unterprogramm

Format:

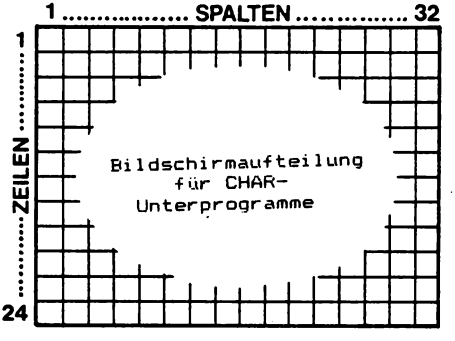
CALL GCHAR(Zeilennummer, Spaltennummer, numerische Variable)

Beschreibung:

Das GCHAR-Unterprogramm ermöglicht das Ablesen eines Zeichens von der durch Zeilennummer und Spaltennummer angegebenen Stelle auf dem Bildschirm. Dabei wird der numerischen Variablen der ASCII-Code des entsprechenden Zeichens zugeordnet.

Zeilennummer und Spaltennummer sind numerische Ausdrücke, die nach den üblichen Regeln berechnet werden. Führt ihre Auswertung nicht zu einem ganzzahligen Wert, so wird dieser entsprechend gerundet.

Die Numerierung der Zeilen und Spalten auf dem Bildschirm ist der nebenstehenden Darstellung zu entnehmen.




Beispiel:

In Programmzeile 100 wird der numerischen Variablen CODE der ASCII-Code des Zeichens, das sich in der 12. Zeile und 16. Spalte (ungefähr Bildschirmmitte) befindet, zugeordnet.

```
100 CALL GCHAR(12,16,CODE)
```

Beispielprogramm:

In dem folgenden Hardcopy-Programm wird die gesamte Bildschirmseite auf den Thermodrucker (als Zubehör erhältlich) kopiert. Dieses Programm eignet sich vor allem als Unterprogramm von Grafikprogrammen.

```
100 REM ***** HARDCOPY *****
110 OPEN #1:"TP.U.S",OUTPUT
120 FOR ZEILE=1 TO 24
130 FOR SPALTE=1 TO 32
140 CALL GCHAR(ZEILE,SPALTE,CODE)
150 PRINT #1:CHR$(CODE);
160 NEXT SPALTE
170 NEXT ZEILE
180 CLOSE #1
190 END
```

## GOSUB

Format:

GOSUB Zeilennummer
GO SUB Zeilennummer

Beschreibung:

Der GOSUB-Befehl bewirkt während des Programmablaufs einen unbedingten Sprung in die mit der angegebenen Zeilennummer beginnende Unterprogrammschleife. Der erste Befehl in der bezeichneten Zeile wird nach dem GOSUB-Befehl ausgeführt, anschließend alle weiteren - einschließlich GOTO- und weiteren GOSUB-Befehlen - bis zu einem RETURN-Befehl. Nach dem RETURN-Befehl wird der dem GOSUB-Befehl folgende Befehl ausgeführt.

Unterprogrammschleifen sind nützlich, wenn gleiche oder ähnliche Programmteile an mehreren Stellen in einem Programm auftreten (vgl. auch ON-GOSUB-Befehl). Unterprogrammschleifen können sich im EXTENDED BASIC im Gegensatz zu Unterprogrammen (vgl. SUB-Befehl) selbst aufrufen (rekursive Programmtechnik); in TI BASIC bedingt dies die Fehlermeldung MEMORY FULL (Speicher belegt), und der Programmablauf wird unterbrochen. Existiert die angegebene Zeilennummer in dem Programm nicht, erscheint die Fehlermeldung BAD LINE NUMBER (falsche Zeilennummer), und der Programmablauf wird ebenfalls unterbrochen.

Beispielprogramm:

Das Programm berechnet die sog. Binomialkoeffizienten nach der Formel

$$\binom{X}{Y} = \frac{X!}{Y!*(X-Y)!}$$

Mit X=49 und Y=6 kann man dann beispielsweise die Zahl der Möglichkeiten beim Zahlenlotto (6 aus 49) bestimmen.

```
100 REM ***** BINOMIALKOEFFIZIENTEN *****
110 CALL CLEAR
120 INPUT "X = ":X
130 INPUT "Y = ":Y
140 PRINT
150 IF X>Y OR X>=0 OR Y>=0 THEN 170
160 PRINT "FALSCHE EINGABE" :: GOTO 120
170 IF X<70 OR Y<70 THEN 190
180 PRINT "ZAHL ZU GROSS!" :: GOTO 120
190 FAKTOR=X :: GOSUB 270
200 BINKOEFF=FAKULT
210 FAKTOR=Y :: GOSUB 270
220 BINKOEFF=BINKOEFF/FAKULT
230 FAKTOR=X-Y :: GOSUB 270
240 BINKOEFF=BINKOEFF/FAKULT
250 PRINT "ERGEBNIS = ";BINKOEFF
260 STOP
270 REM *** FAKULTAET BERECHNEN ***
280 IF FAKTOR<2 THEN FAKULT=1 :: GOTO 330
290 FAKULT=FAKTOR
300 FAKTOR=FAKTOR-1
310 FAKULT=FAKULT*FAKTOR
320 IF FAKTOR>1 THEN 300
330 RETURN
```

## GOTO

Format:

GOTO Zeilennummer
GO TO Zeilennummer

Beschreibung:

Der GOTO-Befehl ermöglicht während des Programmablaufs einen unbedingten Sprung nach der angegebenen Zeilennummer. Der erste Befehl in der bezeichneten Zeile wird nach dem GOTO-Befehl ausgeführt. Existiert die angegebene Zeile nicht in dem ablaufenden Programm, so wird dieses unterbrochen und die Fehlermeldung BAD LINE NUMBER (falsche Zeilennummer) ausgegeben. Der GOTO-Befehl sollte keinen Sprung in ein Unterprogramm bewirken.

Beispielprogramm:

Jedesmal, wenn der Programmablauf zur Zeile 220 gelangt, fährt der Computer mit der Zeile 140 fort.

```
100 REM ***** ADDITION DER ZAHLEN VON 1 BIS 100 *****
110 CALL CLEAR
120 ERGEBNIS=0
130 ZAHL=1
140 DISPLAY AT(5,10):USING "####":ERGEBNIS
150 DISPLAY AT(7,12):"+"
160 DISPLAY AT(9,10):USING "####":ZAHL
170 DISPLAY AT(11,9):"------"
180 ERGEBNIS=ERGEBNIS+ZAHL
190 DISPLAY AT(13,10):USING "####":ERGEBNIS
200 ZAHL=ZAHL+1
210 IF ZAHL>100 THEN STOP
220 GOTO 140
```

## HCHAR-Unterprogramm

Format:

```
CALL HCHAR(Zeilennummer, Spaltennummer, Zeichencode[,Wiederholungszahl])
```

Beschreibung:

Das HCHAR-Unterprogramm setzt ein Zeichen mit dem angegebenen Zeichencode an die Stelle des Bildschirms, die durch Zeilennummer und Spaltennummer angegeben wird. Die zusätzlich mögliche Wiederholungszahl bewirkt, daß das Zeichen entsprechend oft in der jeweiligen Zeile gesetzt wird, also vertikal wiederholt wird, wobei Zeilennummer und Spaltennummer die Startstelle auf dem Bildschirm darstellen. Gegebenenfalls werden die Wiederholungen in den nächsten Zeilen fortgesetzt.

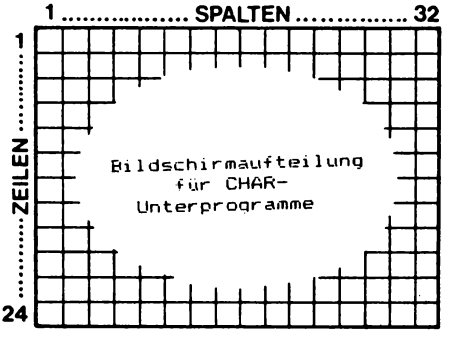
Zeilennummer, Spaltennummer, Zeichencode und Wiederholungszahl sind numerische Ausdrücke, deren Werte nach den üblichen Regeln berechnet werden. Führt die Auswertung der numerischen Ausdrücke nicht zu ganzen Zahlen, so werden sie entsprechend gerundet.

Die Numerierung der Zeilen und Spalten auf dem Bildschirm ist der nebenstehenden Darstellung zu entnehmen.

Soll der ganze Bildschirm mit einem Zeichen mit dem Zeichencode CODE beschrieben werden, so ist der Befehl

```
CALL HCHAR(1,1,CODE,768)
```

zu verwenden.




Beispiele:

In Programmzeile 100 wird das Zeichen 33 (Ausrufezeichen) in der 12. Zeile und 16. Spalte (etwa Bildschirmmitte) gesetzt, in Programmzeile 120 wird es zusätzlich 24 mal wiederholt.

```
100 CALL HCHAR(12,16,33)
120 CALL HCHAR(12,16,ASC("!"),24)
```

Beispielprogramm:

Das Programm zeigt eine etwas komplizierte Anwendung der HCHAR- und VCHAR-Unterprogramme, um den Bildschirm mit einem Muster zu belegen.

```
100 REM ***** SPIRALE *****
110 CALL CLEAR
120 FOR Z1=1 TO 11 STEP 2
130 S1=Z1
140 W1=34-2*Z1
150 CALL HCHAR(Z1,S1,42,W1)
160 S2=S1+W1-1
170 W2=26-2*Z1
180 CALL VCHAR(Z1,S2,42,W2)
190 Z3=Z1+W2-1
200 CALL HCHAR(Z3,S1+2,42,W1-2)
210 CALL VCHAR(Z1+2,S1+2,42,W2-2)
220 NEXT Z1
230 GOTO 230
```

# IF — THEN — ELSE

Format:

```
IF logischer Ausdruck THEN Zeilennummer/Befehlsliste
                      [ELSE Zeilennummer/Befehlsliste]
```

Beschreibung:

Mit dem IF-THEN-ELSE-Befehl können bedingte Programmverzweigungen ausgeführt werden, die vom Wert des logischen Ausdrucks abhängen. Dieser Wert ist entweder -1, wenn der logische Ausdruck wahr ist, oder 0, wenn der logische Ausdruck falsch ist (vgl. auch "Logische Ausdrücke" im Kapitel "Programmierung").

In der Programmverzweigung werden hinter den Schlüsselwörtern THEN und ELSE Programmzeilennummern oder Befehlslisten angegeben. Befehlslisten bestehen aus einem Befehl oder mehreren Befehlen, die durch das Trennungssymbol "::" voneinander getrennt sind. Ist der logische Ausdruck wahr (Wert -1), so werden die Befehle hinter dem Schlüsselwort THEN ab der angegebenen Zeilennummer oder der angegebenen Befehlsliste ausgeführt. Ist der logische Ausdruck falsch (Wert 0), so werden die Befehle hinter dem Schlüsselwort ELSE ab der angegebenen Zeilennummer oder der angegebenen Befehlsliste ausgeführt. Ist der logische Ausdruck falsch und fehlt das Schlüsselwort ELSE, so werden die Befehle der nächsten Programmzeile nach dem IF-THEN-Befehl ausgeführt.

Im TI-BASIC sind nach dem Schlüsselwörtern THEN und ELSE nur Zeilennummern zugelassen. Im IF-THEN-ELSE-Befehl des EXTENDED BASIC dürfen die folgenden Befehle n i c h t verwendet werden:

DATA, DEF, DIM, FOR, NEXT, OPTION BASE, SUB, SUBEND.

Beispiele:

Wenn die numerische Variable V in Programmzeile 100 einen größeren Wert als 5 hat, dann wird die in Zeile 300 beginnende Unterprogrammschleife ausgeführt, andernfalls wird der Wert von V um 5 erhöht und das Programm mit der nächsten Programmzeile fortgesetzt.

```
100 IF V>5 THEN GOSUB 300
    ELSE V=V+5
```

Ist der numerische Ausdruck NA in Programmzeile 120 ungleich 0, dann wird der Wert von V um 1 erhöht und das Programm mit Zeile 500 fortgesetzt, andernfalls ist Z=Z/V und das Programm geht in Zeile 300 weiter.

```
120 IF NA THEN V=V+1 :: GOTO 500 ::
    ELSE Z=Z/V :: GOTO 300
```

Ist der Wert der String-Variablen ANTWORT$ ungleich "J", wird das Programm mit der nächsten Zeile fortgesetzt. Ist ANTWORT$="J", dann wird der Wert der numerischen Variablen ZAEHLER um 1 erhöht und in der letzten Bildschirmzeile eine Nachricht ausgegeben und das Programm in Zeile 300 fortgesetzt.

```
140 IF ANTWORT$="J" THEN ZAEHLER=
    ZAEHLER+1 :: DISPLAY AT (24,1):
    "NOCH EIN VERSUCH!" :: GOTO 300
```

# IF — THEN — ELSE

Wenn die numerische Variable A in Programmzeile 160 ungleich 1 ist, wird E gleich 5 gesetzt, und das Programm wird in der nächsten Zeile fortgesetzt. Wenn A=1 ist, wird überprüft, ob B=2 ist. Ist das nicht der Fall, dann wird D gleich 4 gesetzt. Ist A=1 und B=2, dann wird C gleich 3 gesetzt.

```
160 IF A=1 THEN IF B=2 THEN C=3
    ELSE D=4 ELSE E=5
```

---

Beispielprogramm:

Das folgende "Lottozahlen-Programm" veranschaulicht den Gebrauch des IF-THEN-ELSE-Befehls, u.a. in dem Programmteil, in dem die Zahlen nach ihrer Größe sortiert werden. Für Sortierverfahren ist dieser Befehl unerläßlich.

```
100 REM ***** LOTTOZAHLEN *****
110 CALL CLEAR
120 INPUT "MITTWOCHSLOTTO(J/N)?":FR$
130 INPUT "ANZAHL TIPREIHEN(<17)?":TIP
140 CALL CLEAR :: IF FR$="N" THEN 160
150 DISPLAY AT(1,4)BEEP:"MITTWOCHSLOTTO 7 AUS 38" :: GOTO 170
160 DISPLAY AT(1,4)BEEP:"SAMSTAGSLOTTO 6 AUS 49"
170 IF FR$="N" THEN SYST=6 ELSE SYST=7
180 DISPLAY AT(2,8):TIP;"TIPREIHEN"
190 DISPLAY AT(5,9):"Z A H L E N"
200 REM ***** ZAHLENERMITTLUNG *****
210 RANDOMIZE
220 FOR REIHEN=1 TO TIP
230 FOR Z=1 TO SYST
240 ZAHL(Z)=INT(RND*49+1)
250 IF SYST=7 AND ZAHL(Z)>38 THEN 240
260 FOR KONTR=Z-1 TO 1 STEP -1
270 IF ZAHL(KONTR)=ZAHL(Z)THEN 240
280 NEXT KONTR
290 NEXT Z
300 REM ***** SORTIERUNG *****
310 FOR S1=1 TO SYST-1
320 FOR S2=S1+1 TO SYST
330 IF ZAHL(S1)>ZAHL(S2)THEN X=ZAHL(S1):: ZAHL(S1)=ZAHL(S2):: ZAHL(S2)=X
340 NEXT S2
350 NEXT S1
360 FOR Z=1 TO SYST
370 DISPLAY AT(7+REIHEN,1+3*Z)BEEP:ZAHL(Z)
380 NEXT Z
390 NEXT REIHEN
400 END
```

## IMAGE

Format:

IMAGE Format-String

Beschreibung:

Der IMAGE-Befehl ist mit der Programmzeilennummer versehen, die in den folgenden beiden Befehlen angegeben wird, auf die er sich bezieht:

```
PRINT ... USING Zeilennummer
DISPLAY ... USING Zeilennummer
```

Mit dem IMAGE-Befehl wird das Format bestimmt, in dem Zahlen oder Strings gedruckt bzw. auf dem Bildschirm ausgegeben werden. Er muß allein in einer Programmzeile stehen, die ihrerseits ein ablaufendes Programm nicht beeinflußt, auf die aber durch die o.a. Befehle während des Programmablaufs Bezug genommen wird.

Der für den IMAGE-Befehl erforderliche Format-String kann auch direkt hinter PRINT (DISPLAY) ... USING verwendet werden (z.B.: 100 PRINT USING "##.##":V), er bezieht sich dann aber nur auf den betreffenden PRINT- bzw. DISPLAY-Befehl. Soll ein Format-String in einem Programm mehrfach Verwendung finden, dann ist es sinnvoller, ihn mit einem IMAGE-Befehl zu verbinden und die jeweilige Programmzeile aufzurufen.

Der Format-String kann aus jedem beliebigen Zeichen mit Ausnahme des #-Zeichens, aber insgesamt aus nicht mehr als 254 Zeichen bestehen. Das #-Zeichen wird durch die im PRINT- (DISPLAY-) USING-Befehl vorhandenen Werte der Variablenn ersetzt, wobei jeweils ein #-Zeichen für je eine Stelle des Wertes vorzusehen ist. Bei numerischen Variablen ist ein eventuell auftretendes Minuszeichen mit zu berücksichtigen. Ein Format-String in der Form ### würde also ganze Zahlen im Bereich von -99 bis 999, aber auch dreibuchstabige Wörter korrekt ausgeben.

Wenn der auszugebende Wert (String) in seiner Länge die durch den Format-String vorgegebenen Grenzen überschreitet (für das obige Beispiel z.B. die Zahl -100 oder der String TEXAS), dann wird für jedes #-Zeichen des Format-Strings ein "*" ausgegeben (in unserem Beispiel also: ***). Zu viele Nachkomma-(Dezimal-)Stellen werden gerundet, bis sie mit Hilfe des Format-Strings ausgegeben werden können. Werden die Grenzen des Format-Strings unterschritten, so werden die überzähligen Stellen bei String-Variablenwerten mit dem Leerzeichen, bei Nachkommastellen von numerischen Variablenwerten mit Nullen aufgefüllt. Dies läßt den IMAGE-Befehl besonders für die Ausgabe von Tabellen interessant werden.

Für einen Format-String der Form ####.### ergeben sich folgende Ausgaben:

| WERT | AUSGABE |
|---|---|
| -999.999 | -999.999 |
| -34.5 | -34.500 |
| 0 | 0.000 |
| 12.4565 | 12.457 |
| 6312.9991 | 6312.999 |
| 999999999 | ******** |

Der Dezimalpunkt wird also dort ausgegeben, wo er in einem IMAGE-Befehl auftritt. Soll eine Zahl in wissenschaftlicher Schreibweise ausgegeben werden, dann müssen für das Zeichen E und die möglichen Ziffern des Exponenten im Format-String Potenzierungszeichen (^) angegeben werden. Dabei dürfen nur 10 oder weniger Zeichen (Minuszeichen, #-Zeichen, Dezimalpunkt) verwendet werden.

## IMAGE

Der Format-String kann in Anführungszeichen eingeschlossen werden. Wenn er nicht in Anführungszeichen eingeschlossen ist dann werden voran- oder nachgestellte Leerzeichen ignoriert. Wird der Format-String in direkter Verbindung mit dem PRINT-(DISPLAY-)USING-Befehl verwendet, dann muß er in Anführungszeichen eingeschlossen sein.

Wenn die Anzahl der Werte im PRINT-(DISPLAY-)USING-Befehl die Möglichkeiten des Format-Strings überschreitet, dann wird dieser erneut von seinem Beginn an benutzt, wie das folgende Kurzprogramm als Beispiel zeigt:

```
100 IMAGE ###.##, ##.#
110 PRINT USING 100: 50.34,50.34,37.26,37.26
120 END
```

Dieses Programm liefert die Ausgabe:

```
 50.34, 50.3
 37.26, 37.3
```

---

Beispiele:

Programmzeile 100 ermöglicht die Ausgabe zweier Zahlen, die erste im Bereich von -99 bis 999, die zweite von -9.99 bis 99.99.

```
100 IMAGE BETRAG ### MWST ##.##
```

Programmzeile 120 erlaubt die Ausgabe eines 4 Zeichen umfassenden Strings (z.B. FRAU), dem der String SEHR GEEHRTE vorangestellt wird.

```
120 IMAGE SEHR GEEHRTE ####
```

---

Beispielprogramm:

Das Beispielprogramm, in dem Beträge addiert und mit ihrer Summe ausgegeben werden, zeigt die beiden grundsätzlichen Anwendungsarten des IMAGE-Befehls:

```
100 REM ***** KASSENZETTEL *****
110 CALL CLEAR
120 IMAGE DM#####.##
130 IMAGE "  ####.##"
140 REM ***** EINGABE *****
150 DISPLAY AT(24,1):"0=ENDE"
160 DISPLAY AT(10,8):"BETRAG:"
170 Z=Z+1
180 ACCEPT AT(10,15)BEEP SIZE(7):BETR(Z)
190 SUMME=SUMME+BETR(Z)
200 IF BETR(Z)=0 THEN 220
210 GOTO 170
220 REM ***** AUSGABE *****
230 CALL CLEAR
240 PRINT USING 120:BETR(1)
250 FOR I=2 TO Z-1
260 PRINT USING 130:BETR(I)
270 NEXT I
280 PRINT "----------"
290 PRINT USING "DM#####.##":SUMME
300 END
```

## INIT-Unterprogramm

Format:

```
CALL INIT
```

Beschreibung:

Das INIT-Unterprogramm wird zusammen mit LINK, LOAD und PEEK benötigt, um Assembler-Unterprogramme zu benutzen.

Das INIT-Unterprogramm überprüft, ob die Speichererweiterung angeschlossen ist und bereitet den Computer auf die Benutzung von Assembler-Unterprogrammen durch einige weitere unterstützende Unterprogramme vor.

Das INIT-Unterprogramm muß vor LOAD und LINK aufgerufen werden. dabei werden alle vorher in die Speichererweiterung geladenen Unterprogramme gelöscht. Die Wirkungen des INIT-Unterprogramms bleiben bestehen, bis die Speichererweiterung ausgeschaltet wird. Es braucht nicht von jedem Programm, das das entsprechende Unterprogramm benutzt, aufgerufen zu werden.

Ist die Speichererweiterung nicht angeschlossen, wird eine entsprechende Fehlermeldung abgegeben.

Genauere Informationen zur Anwendung der INIT-, LINK-, LOAD- und PEEK-Unterprogramme sind in den Dokumentationen zu den Assembler-Unterprogrammen (auf Diskette oder Cassette erhältlich) zu finden.

# INPUT

Format:

```
INPUT Eingabe-Dialogzeile : Variablenliste
INPUT [#Dateinummer[,REC Datensatznummer]:]Variablenliste
```

Beschreibung:

Mit dem INPUT-Befehl können numerische oder String-Werte über die Tastatur oder eine durch #Dateinummer bezeichnete externe Datei in ein ablaufendes Programm zur weiteren Bearbeitung eingegeben werden.

Bei der Eingabe über Tastatur wird die Programmausführung solange unterbrochen, bis die Dateneingabe mit <ENTER> beendet ist.

Die Variablenliste enthält die numerischen und/oder String-Variablen, denen bei der Ausführung des INPUT-Befehls die jeweiligen Werte zugeordnet werden. Die einzelnen Variablen werden durch Kommata getrennt, ebenso wie die Variablenwerte bei der Eingabe über die Tastatur. Sollen einzugebende String-Variablewerte voran- oder nachgestellte Leerzeichen oder Kommata enthalten, dann müssen sie in Anführungszeichen eingeschlossen werden, sonst nicht.

Mit der Eingabe-Dialogzeile können bei der Eingabe über Tastatur auf dem Bildschirm die einzugebenden Daten beschrieben werden. Die Eingabe-Dialogzeile ist ein String-Ausdruck, der hier mit einem Doppelpunkt abgeschlossen werden muß. Die Eingabe-Dialogzeile wird in der letzten Bildschirmzeile ausgegeben, wenn der INPUT-Befehl ausgeführt wird. Fehlt die Eingabe-Dialogzeile, dann werden in der letzten Bildschirmzeile ein Fragezeichen und ein Leerzeichen ausgegeben, bei einer vorhandenen Eingabe-Dialogzeile nimmt diese den Platz des Fragezeichens und des Leerzeichens ein.

Sollen mit dem INPUT-Befehl Daten von einer externen Datei (z.B. auf einer Cassette oder Diskette) eingelesen werden, dann muß die entsprechende Datei unter der angegebenen Dateinummer im INPUT- oder UPDATE-Modus geöffnet worden sein (vgl. OPEN-Befehl). Ist der Datentyp der Datei DISPLAY, dann dürfen die Datensätze nicht aus mehr als 160 Zeichen bestehen.

Die Datensatznummer kann im Zusammenhang mit RELATIVE-Dateien auf Diskette verwendet werden. Sie gibt die Nummer des einzulesenden Datensatzes der externen Datei an, auf den damit direkt zugegriffen werden kann (im Gegensatz zu sequentiellen Dateien).

Die über den INPUT-Befehl eingegebenen Daten werden vom Computer auf ihre Korrektheit überprüft. Versucht man, einen String einzugeben, wenn ein numerischer Wert erwartet wird, so erscheint die Warnung *WARNING: INPUT ERROR IN ### (Eingabefehler in Zeile ###), und die Eingabe muß wiederholt werden. Im umgekehrten Fall erscheint keine Warnung, da eine Zahl vom Computer auch als String aufgefaßt werden kann. Ist eine eingegebene Zahl zu groß, erscheint die Warnung *WARNING: NUMERIC OVERFLOW IN ### (numerischer Überlauf in Zeile ###), und die Eingabe muß ebenfalls wiederholt werden.

Beispiele:

Programmzeile 100 ermöglicht die Eingabe einer Zahl, die der numerischen Variable X zugeordnet wird.

```
100 INPUT X
```

# INPUT

In Programmzeile 120 werden eine Zahl und ein String als Eingabe erwartet. Beide Variablenwerte werden bei der Eingabe durch ein Komma getrennt.

```
120 INPUT X,Y$
```

Mit der Eingabe-Dialogzeile in Programmzeile 140 wird deutlich gemacht, daß zwei Zahlen eingegeben werden sollen.

```
140 INPUT "EINGABE VON 2 ZAHLEN:":
    A,B
```

Mit Programmzeile 160 wird die Eingabe eines mit dem Wert von I zu bestimmenden Elementes eines numerischen Datenfeldes A ermöglicht.

```
160 INPUT I,A(I)
```

In Programmzeile 180 wird der Wert für X von der Datei #3 eingelesen, in Programmzeile 200 kommt noch ein Stringwert für Y$ hinzu.

```
180 INPUT #1:X
200 INPUT #3:X,Y$
```

In Programmzeile 220 wird aus dem Datensatz mit der Nummer 5 (6. Datensatz) der Datei #7 der 1. numerische Wert eingelesen und der Variablen X zugewiesen.

```
220 INPUT #7,REC 5:X
```

---

Beispielprogramme:

Die INPUT-Befehle in den Programmzeilen 140 und 150 haben unterschiedliche Funktionen: Im ersten Programm werden damit Namen über Tastatur eingegeben, im zweiten Programm von der Datei auf Cassette eingelesen.

```
100 REM ***** ABSPEICHERN EINES NAMENREGISTERS *****
110 REM *************** AUF CASSETTE ***************
120 OPEN #1:"CS1",INTERNAL,OUTPUT,FIXED 64
130 FOR I=1 TO 10
140 INPUT "NAME: ":NAME$
150 INPUT "VORNAME: ":VORNAME$
160 PRINT #1:NAME$
170 PRINT #1:VORNAME$
180 NEXT I
190 CLOSE #1
200 END
```

```
100 REM ***** EINLESEN EINES NAMENREGISTERS *****
110 REM ************* VON CASSETTE **************
120 OPEN #1:"CS1",INTERNAL,INPUT ,FIXED 64
130 FOR I=1 TO 10
140 INPUT #1:NAME$
150 INPUT #1:VORNAME$
160 PRINT NAME$
170 PRINT VORNAME$
180 NEXT I
190 CLOSE #1
200 END
```

# INT

Format:

INT(numerischer Ausdruck)

Beschreibung:

Die INT-Funktion gibt die größte ganze Zahl an, die kleiner als der Wert des numerischen Ausdrucks ist. Das bedeutet, daß bei positiven Zahlen der Dezimalteil (alles nach dem Dezimalpunkt) weggelassen wird, während bei negativen Zahlen die nächstkleinere ganze Zahl genommen wird. Der Wert des numerischen Ausdrucks wird nach den üblichen Regeln berechnet.

Beispiele:

In Programmzeile 100 wird die Zahl 5 uns in Programmzeile 120 die Zahl -5 ausgegeben.

```
100 PRINT INT(5.87353)
120 PRINT INT(-4.35212)
```

Der Variablen X wird in Programmzeile 140 der Wert 0 und der Variablen Y in Programmzeile 160 der Wert -1 zugewiesen.

```
140 X=INT(.99999999)
160 Y=INT(-.00000001)
```

## JOYST-Unterprogramm

Format:

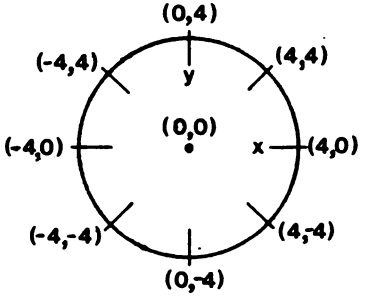
```
CALL JOYST(Tastaturmodus, X-Wert, Y-Wert)
```

Beschreibung:

Mit dem JOYST-Unterprogramm können Daten, basierend auf der Position der Hebel der kabelgebundenen Fernbedienungen (Zubehör) eingegeben werden.

Der Tastaturmodus ist ein numerischer Ausdruck, dessen Wert nach seiner Berechnung (nach den üblichen Regeln) eine ganze Zahl zwischen 1 und 4 (einschließlich) sein muß. Die Werte 1 und 2 entsprechen den beiden Fernbedienungen, die Werte 3 und 4 sind für einen möglichen späteren Gebrauch reserviert.

Der X-Wert und der Y-Wert hängen von der Hebelposition der bezeichneten Fernbedienung ab. Sie betragen -4, 0 oder +4 gemäß der folgenden Abbildung (die erste Zahl in den Klammern stellt den X-Wert und die zweite den Y-Wert dar).




Beispiel:

In Programmzeile 100 werden die Werte X und Y der Fernbedienung 1 an den Computer übergeben.

```
100 CALL JOYST(1,X,Y)
```

Beispielprogramm:

Das Programm zeigt eine typische Anwendung des JOYST-Unterprogramms. Die beiden Zeichen * und O werden mit Hilfe der beiden Fernbedienungen bewegt. Das Programm endet, wenn beide Zeichen genau zusammentreffen.

ACHTUNG: Um die Fernbedienung voll funktionsfähig zu haben, darf die ALPHA LOCK-Taste nicht gedrückt sein!

```
100 REM ***** HASCH MICH *****
110 CALL CLEAR
120 CALL SPRITE(#1,42,13,96,100)
130 CALL SPRITE(#2,48,5,96,156)
140 CALL JOYST(1,X1,Y1)
150 CALL JOYST(2,X2,Y2)
160 CALL MOTION(#1,-2*Y1,2*X1)
170 CALL MOTION(#2,-2*Y2,2*X2)
180 CALL COINC(ALL,TREFFER)
190 IF TREFFER=-1 THEN STOP
200 GOTO 140
```

## KEY-Unterprogramm

Format:

```
CALL KEY(Tastaturmodus,Rückmeldevariable,Statusvariable)
```

Beschreibung:

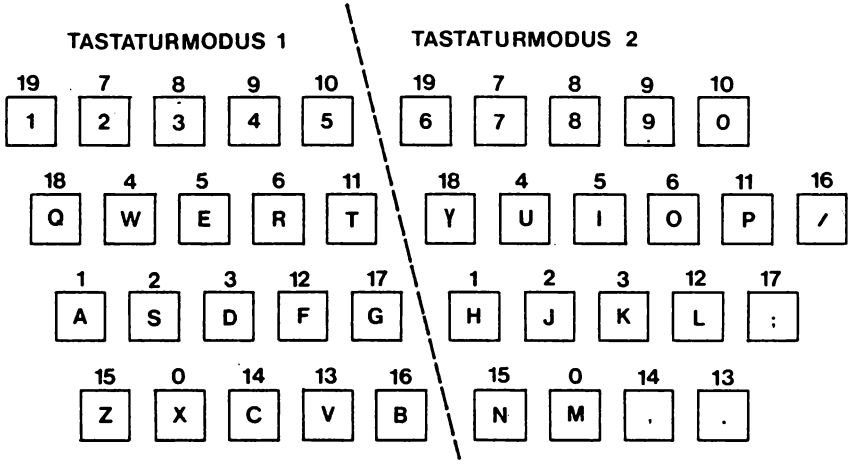
Das KEY-Unterprogramm ermöglicht die direkte Eingabe der Tastaturzeichen, ohne daß dabei die Taste <ENTER> betätigt werden muß. Dabei wird nicht das Tastaturzeichen selbst, sondern sein Code an die Rückmeldevariable übergeben. Der Zeichencode hängt wiederum vom verwendeten Tastaturmodus ab. Über die Statusvariable kann festgestellt werden, ob eine Taste betätigt worden ist.

Der Tastaturmodus ist ein numerischer Ausdruck, der nach seiner Auswertung eine ganze Zahl zwischen 0 und 5 einschließlich sein muß. Dabei bedeuten:

| TASTATURMODUS | BEDEUTUNG |
|---|---|
| 0 | Konsolentastatur |
| 1 | linke Seite der Konsolentastatur oder Fernbedienungseinheit 1 |
| 2 | rechte Seite der Konsolentastatur oder Fernbedienungseinheit 2 |
| 3,4,5 | spezielle Tastaturmodi für spätere Anwendungen reserviert |

Ist der Tastaturmodus 0, so wird der Rückmeldevariablen der jeweilige ASCII-Code des Tastenzeichens zugeordnet (vgl. Anhang). Wird keine Taste betätigt, dann erhält die Rückmeldevariable den Wert -1. Die Rückmeldevariable muß eine numerische Variable sein.

Die Teilung der Tastatur in den Tastaturmodi 1 und 2 sowie die dabei der Rückmeldevariablen zugeordneten Werte sind der folgenden Abbildung zu entnehmen:




Zur Fernbedienung s. JOYST-Unterprogramm!

## KEY-Unterprogramm

Die Statusvariable ist ein numerischer Ausdruck, dessen Wert angibt, ob eine Taste betätigt worden ist. Ist der Wert

1, dann ist nach dem letzten CALL KEY-Befehl eine n e u e Taste betätigt worden,

-1, dann ist d i e s e l b e Taste wie beim letzten CALL KEY-Befehl betätigt worden,

0, dann ist k e i n e Taste betätigt worden.

---

Beispielprogramme:

Im ersten Programm ist der Tastaturmodus 0 (Konsolentastatur) gewählt worden. Der ASCII-Code wird der numerischen Variablen CODE zugeordnet, dieser wird dann auf dem Bildschirm ausgegeben.

Das Programm endet mit <FCTN-4> (CLEAR).

```
100 REM ***** TASTATUR-ZEICHENCODE *****
110 CALL CLEAR
120 CALL KEY(0,CODE,STATUS)
130 IF STATUS=0 THEN 120
140 PRINT TAB(14);CODE
150 GOTO 120
```

Das zweite Programm ähnelt dem Beispielprogramm zum JOYST-Unterprogramm, im Gegensatz dazu werden die Sprites hier über die Tastatur gesteuert. Die Steuertasten sind <X>, <S>, <D> und <E> für Sprite #1 und <M>, <J>, <K> und <I> für Sprite #2. Das Programm endet, wenn beide Sprites zusammentreffen.

```
100 REM ***** HASCH MICH *****
110 CALL CLEAR
120 CALL SPRITE(#1,42,13,96,100)
130 CALL SPRITE(#2,48,5,96,156)
140 CALL KEY(1,CODE1,STATUS1)
150 CALL KEY(2,CODE2,STATUS2)
160 IF CODE1=0 THEN X1=0 :: Y1=-4
170 IF CODE2=0 THEN X2=0 :: Y2=-4
180 IF CODE1=2 THEN X1=-4 :: Y1=0
190 IF CODE2=2 THEN X2=-4 :: Y2=0
200 IF CODE1=3 THEN X1=4 :: Y1=0
210 IF CODE2=3 THEN X2=4 :: Y2=0
220 IF CODE1=5 THEN X1=0 :: Y1=4
230 IF CODE2=5 THEN X2=0 :: Y2=4
240 CALL MOTION(#1,-2*Y1,2*X1)
250 CALL MOTION(#2,-2*Y2,2*X2)
260 CALL COINC(ALL,TREFFER)
270 IF TREFFER=-1 THEN STOP
280 GOTO 140
```

# LEN

Format:

LEN(String-Ausdruck)

Beschreibung:

Die LEN-Funktion gibt die Anzahl der Zeichen des String-Ausdrucks an. Dabei zählt das Leerzeichen als Zeichen.

Beispiele:

In Programmzeile 100 wird 5 ausgegeben.

```
100 PRINT LEN("ABCDE")
```

In Programmzeile 120 erhält die Variable L den Wert 18 zugewiesen.

```
120 L=LEN("DIES IST EIN SATZ!")
```

Programmzeile 140 gibt 0 und Programmzeile 160 die Zahl 1 aus.

```
140 DISPLAY LEN("")
160 DISPLAY LEN(" ")
```

# LET

Format:

[LET]numerische Variable[,numerische Variable][,...] =,numerischer Ausdruck
[LET]String-Variable[,String-Variable][,...] = String-Ausdruck

Beschreibung:

Mit dem LET-Befehl wird der Wert eines Ausdrucks einer oder mehreren Variablen zugewiesen. Der Computer wertet den Ausdruck rechts vom Gleichheitszeichen nach den üblichen Regeln aus und ordnet den Wert der/den links vom Gleichheitszeichen stehenden Variablen zu. Stehen links vom Gleichheitszeichen mehr als eine Variable (nur im EXTENDED BASIC möglich), so müssen diese durch Kommata getrennt werden. In dem numerischen Ausdruck können auch Vergleichsoperatoren und logische Operatoren verwendet werden. Ist der Wahrheitswert des Vergleichs oder des logischen Ausdrucks wahr (true), so erhält die numerische Variable den Wert -1 zugewiesen, ist der Wahrheitswert falsch (false), den Wert 0.

Das Wort "LET" ist eine Option und kann weggelassen werden, was üblicherweise geschieht. So fehlt es auch in allen Anweisungen dieses Handbuchs.

Beispiele:

In Programmzeile 100 wird der numerischen Variablen A der Wert 3.7 zugewiesen, in 120 der Variablen B der Wert -1.

```
100 A=3.7
120 B=2<3
```

Die Variablen X, Y und Z in Programmzeile 140 erhalten den Wert 1.

```
140 X,Y,Z=1
```

Die String-Variablen X$, Y$ und Z$ in Programmzeile 160 bekommen den String "ABC" zugewiesen.

```
160 X$,Y$,Z$="ABC"
```

## LINK-Unterprogramm

Format:

```
CALL LINK(Unterprogrammname[,Variablenliste])
```

Beschreibung:

Das LINK-Unterprogramm wird zusammen mit INIT, LOAD unnd PEEK benötigt, um Assembler-Unterprogramme zu benutzen.

Der LINK-Befehl übergibt den Programmablauf und - falls vorhanden - eine Variablenliste vom Hauptprogramm in EXTENDED BASIC einem Assembler-Unterprogramm.

Unterprogrammname ist die Bezeichnung des aufzurufenden Unterprogramms. Dieses muß zuvor mit Hilfe des LOAD-Befehls in die Speichererweiterung geladen worden sein. Die Variablenliste ist eine Liste von Ausdrücken und Variablen, deren Werte an das aufgerufene Assembler-Unterprogramm übergeben werden sollen.

Genauere Informationen zur Anwendung der INIT-, LINK-, LOAD- und PEEK-Unterprogramme sind in den Dokumentationen zu den Assembler-Unterprogrammen (auf Diskette oder Cassette erhältlich) zu finden.

## LINPUT

Format:

```
LINPUT [#Dateinummer [,REC Datensatznummer]:]String-Variable
LINPUT[Eingabe-Dialogzeile:]String-Variable
```

Beschreibung:

Der LINPUT-Befehl ermöglicht die Eingabe einer vollständigen Zeile oder eines vollständigen Datensatzes oder - bei einer "schwebenden" Eingabebedingung (vgl. INPUT-Befehl) - des restlichen Teils eines Datensatzes mit der anschließenden Zuweisung an die angegebene String-Variable. Der einmal eingebene String kann im Nachhinein nicht mehr editiert werden. Im Gegensatz zum INPUT-Befehl werden auch Zeichen wie Komma, Semikolon, Doppelpunkt, Fragezeichen und voran- bzw. nachgestellte Leerzeichen der String-Variablen zugeordnet.

Optionen:

Beim LINPUT-Befehl kann die Dateinummer einer - vorher zu öffnenden - Datei angegeben werden (vgl. OPEN-Befehl). Der Datentyp muß DISPLAY sein. Ist die Datei zusätzlich vom Typ RELATIVE, so können einzelne Datensätze über REC Datensatznummer angesprochen werden.

Ist keine Datei angegeben, so erfolgt die Eingabe über die Tastatur. Dabei ist die Darstellung einer Eingabe-Dialogzeile (vgl. INPUT-Befehl) möglich.

Beispiele:

In Programmzeile 100 können beliebige Zeichenfolgen über die Tastatur eingegeben und der String-Variablen NAME$ zugeordnet werden, bis die Taste <ENTER> gedrückt wird, in Zeile 120 ist zusätzlich der Eingabedialog "NAME: " vorangestellt.

```
100 LINPUT NAME$
120 LINPUT "NAME$: ":NAME$
```

In Programmzeile 140 wird der String-Variablen ZEILE$(N) der N-te Datensatz einer zuvor mit #1 geöffneten Datei zugeordnet.

```
140 LINPUT #1,REC N:ZEILE$(N)
```

Beispielprogramme:

Mit den beiden Programmen kann ein Text auf Diskette abgespeichert und wieder in den Computer geladen werden. Einmal bezieht sich der LINPUT-Befehl auf die Tastatur, das andere Mal auf die externe Datei DSK1.TEXT.

```
100 REM ***** TEXTEINGABE *****
110 OPEN #1:"DSK1.TEXT",OUTPUT,FIXED 80,DISPLAY
120 LINPUT ZEILE$
130 IF ZEILE$="XXX" THEN 160
140 PRINT #1:ZEILE$
150 GOTO 120
160 CLOSE #1 :: STOP
```

```
100 REM ***** TEXTAUSGABE *****
110 OPEN #1:"DSK1.TEXT",INPUT ,FIXED 80,DISPLAY
120 IF EOF(1)THEN CLOSE #1 :: STOP
130 LINPUT #1:ZEILE$
140 PRINT ZEILE$
150 GOTO 120
```

# LIST

Format:

```
LIST["Gerätename":][Liste der Zeilennummern]
```

Beschreibung:

Mit dem LIST-Befehl lassen sich komplette Programme oder auch Programmteile bis hin zur einzelnen Programmzeile entweder auf dem Bildschirm oder einem mit Gerätename bezeichneten Peripheriegerät ausgeben ("auflisten"). Das betreffende Programm muß sich zu diesem Zweck im Arbeitsspeicher des Computers befinden.

Die Liste der Zeilennummern kann aus einer Einzelzahl (###), einer Einzelzahl mit voran- oder nachgestelltem Bindestrich (-### oder ###-) oder aus zwei mit Bindestrich verbundenen Zahlen (###1-###2) bestehen. Danach ergeben sich folgende Möglichkeiten:

| BEFEHL | AUFGELISTETER PROGRAMMTEIL |
|---|---|
| LIST | vollständiges Programm |
| LIST ### | nur die Programmzeile mit der angegebenen Zeilennummer |
| LIST ###- | von der angegebenen Zeilennummer bis zum Programmende |
| LIST -### | vom Programmanfang bis zur angegebenen Zeilennummer |
| LIST ###1-###2 | alle Programmzeilen mit den Zeilennummern von ###1 bis ###2 |

Werden die Programmzeilen auf dem Bildschirm ausgegeben, "laufen" die Programmzeilen von unten nach oben durch (Bildschirm-Scrolling). Durch Betätigung einer beliebigen Taste kann dies unterbrochen werden, ein erneutes Drücken einer beliebigen Taste setzt die Ausgabe der Programmzeilen fort. Eine endgültige Unterbrechung, und damit ein Abbruch des LIST-Befehls, kann durch CLEAR (<FCTN-4>) erreicht werden.

Die Programmauflistung kann auch über einen Thermodrucker oder die RS232-Schnittstelle erfolgen. Im ersten Fall ist der Gerätename TP, im zweiten z.B. RS232.BA=9600. Die Gerätenamen müssen in Anführungszeichen stehen. Fehlt der Gerätename, dann erfolgt die Ausgabe auf dem Bildschirm.

Beispiele:

Das gesamte Programm wird auf dem Bildschirm ausgegeben.

```
>LIST
```

Alle Programmzeilen zwischen den Zeilennummern 100 und 300 einschließlich werden auf dem Bildschirm ausgegeben.

```
>LIST 100-300
```

Alle Programmzeilen mit Zeilennummern kleiner oder gleich 300 werden über den Thermodrucker ausgegeben.

```
>LIST "TP":-300
```

Alle Programmzeilen mit Zeilennummern größer oder gleich 300 werden über einen Drucker, der an die RS232-Schnittstelle angeschlossen ist, ausgegeben.

```
>LIST "RS232.BA=9600":300-
```

## LOAD-Unterprogramm

Format:

```
CALL LOAD("Dateibezeichnung" [,Adresse,Byte1][,...][,Dateifeld][,...] )
```

Beschreibung:

Das LOAD-Unterprogramm wird zusammen mit INIT, LINK und PEEK benötigt, um Assembler-Unterprogramme zu benutzen.

Das LOAD-Unterprogramm lädt Assembler-Objekt-Dateien von Diskette oder Cassette oder speichert Direkt-Daten in die Speichererweiterung ("POKE"), um sie für eine spätere Benutzung mit Hilfe des LINK-Unterprogramms bereitzustellen.

Mit einer gültigen Dateibezeichnung wird eine zu ladende Direkt-Objekt-Datei angegeben. In einem LOAD-Befehl können mehrere solcher Dateien benannt werden.

Sollen eine Reihe von Direkt-Daten mit dem LOAD-Unterprogramm eingegeben werden, so erfolgt zunächst die Angabe der Adresse der ersten Speicherstelle und anschließend die Datenbytes. Adresse und Datenbytes, letztere auch untereinander, werden durch Kommata getrennt.

Mehrere Objekt-Daten müssen durch je ein Dateifeld getrennt werden. Ein Dateifeld ist ein String-Ausdruck, der angibt, von welcher Datei der Assembler-Objektcode geladen wird. Das Dateifeld ist ein Leerstring (String ohne Zeichen), wenn Direkt-Daten voneinander getrennt werden sollen.

Unsachgemäßer Gebrauch von LOAD kann den Computer "sperren", was ein Ausschalten und anschließendes Wiedereinschalten des Computers für den weiteren Gebrauch erforderlich macht.

Genauere Informationen zur Anwendung der INIT-, LINK-, LOAD- und PEEK-Unterprogramme sind in den Dokumentationen zu den Assembler-Unterprogrammen (auf Diskette oder Cassette erhältlich) zu finden.

Beispiele:

In Programmzeile 100 wird der Objektcode der Datei OBJDAT, die sich auf der Diskette im Laufwerk 1 befindet, geladen.

```
100 CALL LOAD("DSK1.OBJDAT")
```

In Programmzeile 120 werden die Direkt-Daten 7, 3 und 4 in die Speicherstellen mit den Adressen 8192, 8193 und 8194 geladen (ge"POKE"t).

```
120 CALL LOAD(8192,7,3,4)
```

## LOCATE-Unterprogramm

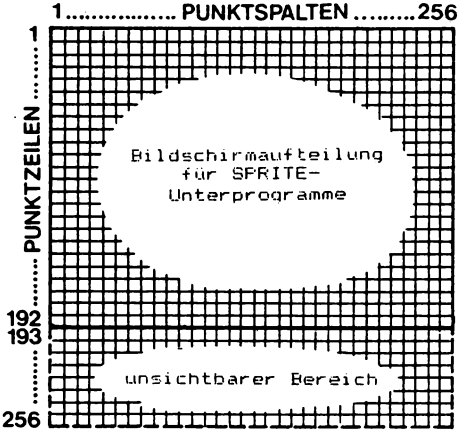
Format:

```
CALL LOCATE(#Sprite-Nummer, Punktzeile, Punktspalte [,...])
```

Beschreibung:

Das LOCATE-Unterprogramm wird dazu verwendet, um Sprites mit der angegebenen Sprite-Nummer an die durch Punktzeile und Punktspalte bestimmte Bildschirmposition zu bringen. Dabei wird die linke obere Ecke des Sprites an die angegebene Bildschirmposition gebracht.

Die Numerierung der Punktzeilen und -spalten auf dem Bildschirm ist der nebenstehenden Darstellung zu entnehmen. Beachten Sie, daß Bildschirmpunkte mit Punktzeilen, die größer als 192 sind, nicht sichtbar sind.




Beispiel:

Programmzeile 100 bringt den Sprite #1 an die Bildschirmposition (200,200), d.h. sie "versteckt" den Sprite im nicht sichtbaren Bildschirmbereich.

```
100 CALL LOCATE(#1,200,200)
```

Beispielprogramm:

Es werden zwei Sprites an zufällige Bildschirmpositionen gebracht, von denen sich Sprite #1 eine Zeit lang bewegt.

Das Programm endet mit <FCTN-4> (CLEAR).

```
100 REM *****HUEPF-SPRITE *****
110 RANDOMIZE
120 CALL CLEAR
130 CALL SPRITE(#1,33,5,1,1,15,15)
140 CALL SPRITE(#2,42,2,1,1)
150 PZEILE=INT(RND*192+1)
160 PSPALTE=INT(RND*256+1)
170 CALL LOCATE(#1,PZEILE,PSPALTE,#2,PZEILE,PSPALTE)
180 DISPLAY AT(24,6)BEEP:"POSITION:";PZEILE;PSPALTE
190 FOR VERZOEG=1 TO 300 :: NEXT VERZOEG
200 GOTO 150
```

## LOG

Format:

LOG(numerischer Ausdruck)

Beschreibung:

Die LOG-Funktion gibt den natürlichen Logarithmus des Wertes des numerischen Ausdrucks an. Der Wert, von dem der Logarithmus bestimmt werden soll, heißt Numerus und darf nicht negativ sein, sonst erscheint die Fehlermeldung BAD ARGUMENT, und der Programmablauf wird unterbrochen. Für die Berechnung des numerischen Ausdrucks werden die üblichen Regeln angewendet. Die LOG-Funktion ist die Umkehrung der EXP-Funktion, daher gilt:

$$X = LOG(EXP(X)).$$

Zur Berechnung des Logarithmus von einem Numerus zu einer anderen Basis B als der natürlichen Basis e gilt die Formel:

$$\log_B(X) = \log_e(X)/\log_e(B)$$

(s. Beispielprogramm).

Beispiele:

In Programmzeile 100 wird der Logarithmus von 3.4 ausgegeben, dieser beträgt 1.223775431622.

```
100 PRINT LOG(3.4)
```

Die Variable X in Programmzeile 120 erhält den Wert 5.

```
120 X=LOG(EXP(5))
```

Der Variablen Y in Programmzeile 140 wird der natürliche Logarithmus der positiven Quadratwurzel aus Z zugeordnet.

```
140 Y=LOG(SQR(Z))
```

Beispielprogramm:

Das Programm berechnet den Logarithmus einer positiven Zahl zu einer beliebigen vorher einzugebenden Basis.

Das Programm endet mit <FCTN-4> (CLEAR).

```
100 REM ***** BERECHNUNG DER LOGARITHMEN *****
110 REM ***** ZU EINER BELIEBIGEN BASIS ******
120 CALL CLEAR
130 INPUT "BASIS: ":BASIS
140 IF BASIS<0 THEN 130
150 INPUT "NUMERUS: ":ZAHL
160 ERGEBNIS=LOG(ZAHL)/LOG(BASIS)
170 PRINT "LOG(";ZAHL;")=";ERGEBNIS
180 PRINT
190 GOTO 130
```

## MAGNIFY-Unterprogramm

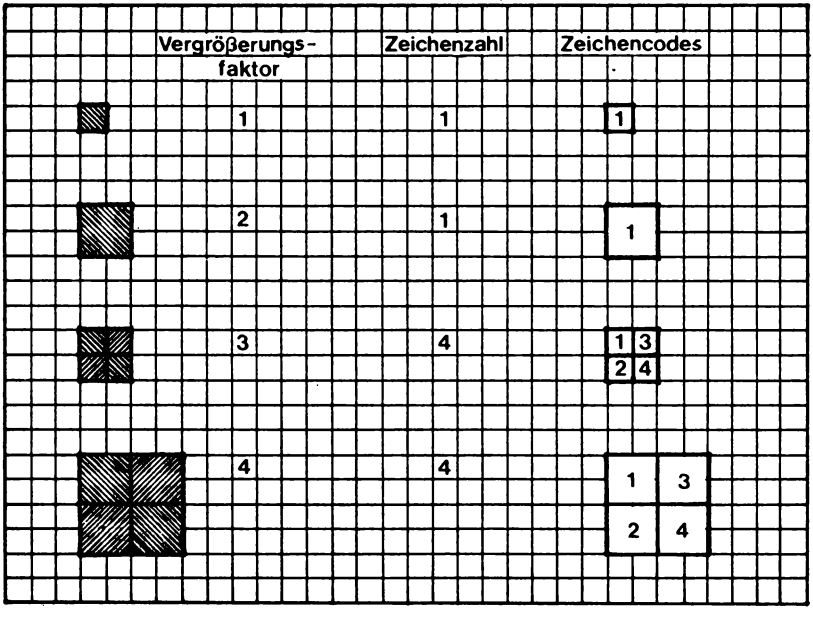
Format:

CALL MAGNIFY(Vergrößerungsfaktor)

Beschreibung:

Mit dem MAGNIFY-Unterprogramm kann die Größe von Sprites und auch die Anzahl der Zeichen, die ein Sprite bilden, beeinflußt werden. Der CALL MAGNIFY-Befehl wirkt sich immer auf alle im Programm befindlichen Sprites aus. Der Vergrößerungsfaktor ist ein numerischer Ausdruck, dessen ganzzahliger Wert zwischen 1 und 4 einschließlich liegen muß. Wird das MAGNIFY-Unterprogramm nicht aufgerufen, ist der Vergrößerungsfaktor 1.

Die folgende Abbildung zeigt die Wirkung des Vergrößerungsfaktors:




Ist der Vergrößerungsfaktor 1, so erscheint das Sprite in einfacher Größe auf dem Bildschirm.

Ist der Vergrößerungsfaktor 2, so erscheint das Sprite in doppelter Größe (4 Schreibstellen) auf dem Bildschirm.

Ist der Vergrößerungsfaktor 3, so erscheint das Sprite in einfacher Größe. Es besteht jedoch aus 4 Zeichen, deren Zeichencodes 4 aufeinanderfolgende Zahlen sind, deren Zuordnung der Abbildung zu entnehmen ist. Das Sprite füllt damit 4 Schreibstellen aus.

Ist der Vergrößerungsfaktor 4, so erscheint das aus 4 Zeichen bestehende Sprite in doppelter Größe auf dem Bildschirm. Das Sprite umfaßt dabei also 16 Schreibstellen.

## MAGNIFY-Unterprogramm

Beispielprogramm:

Es erscheint die aus dem Beispielprogramm zum CHAR-Unterprogramm (s.d.) bekannte Figur, die nach einiger Zeit (Verzögerungsschleife) vergrößert wird. Anschließend wird sie durch das linke obere Viertel einer größeren Figur ersetzt, die schließlich in voller Größe auf dem Bildschirm erscheint. Nachdem sie sich eine kurze Zeit auf dem Bildschirm bewegt hat, wird sie verkleinert, hat aber immer noch die Größe von 4 Schreibstellen im Gegensatz zur Anfangsfigur von nur einer Schreibstelle.

```
100 REM ***** WACHSTUM *****
110 CALL CLEAR
120 CALL CHAR(96,"1898FF3D3C3CE404")
130 CALL SPRITE(#1,96,5,50,50)
140 GOSUB 270
150 CALL MAGNIFY(2)
160 GOSUB 270
170 Z$="0103C3417F3F07070707077E7C400000B0C0C0B0FCFEE2E3E0E0E0606060607"
180 CALL CHAR(96,Z$)
190 GOSUB 270
200 CALL MAGNIFY(4)
210 GOSUB 270
220 CALL MOTION(#1,5,5)
230 GOSUB 270
240 CALL MAGNIFY(3)
250 GOSUB 270
260 STOP
270 REM ***** VERZOEGERUNG *****
280 CALL SOUND(200,440,4,220,4,660,4)
290 FOR VERZOEG=1 TO 750 :: NEXT VERZOEG
300 RETURN
```

## MAX

Format:

MAX(numerischer Ausdruck 1, numerischer Ausdruck 2)

Beschreibung:

Die MAX-Funktion ergibt den größeren der beiden Werte des numerischen Ausdrucks 1 und des numerischen Ausdrucks 2. Sind beide Werte gleich, so erhält man den gemeinsamen Wert. Die numerischen Ausdrücke werden zunächst nach den üblichen Regeln berechnet.

Beispiele:

In Programmzeile 100 wird die Zahl 5 ausgegeben, in 120 der Wert -3.

```
100 PRINT MAX(2,5)
120 PRINT MAX(-3,-5)
```

In Programmzeile 140 wird der Variablen G der größere der beiden Werte von A und B zugewiesen.

```
140 G=MAX(A,B)
```

## MERGE

Format:

MERGE Gerätename.Programmname

Beschreibung:

Der MERGE-Befehl bewirkt die Übernahme der Programmzeilen des mit Gerätename.Programmname bezeichneten Programms in den Arbeitsspeicher des Computers (merge = einbetten). Im Gegensatz zum OLD-Befehl wird dabei das eventuell schon im Arbeitsspeicher befindliche Programm nicht gelöscht. Eine schon im Arbeitsspeicher befindliche Programmzeile mit derselben Zeilennummer wird dabei von der neu geladenen Programmzeile überschrieben. Ansonsten werden alle Programmzeilen nach Zeilennummern geordnet. Durch den MERGE-Befehl werden die eventuell vorhandenen Stoppstellen nicht aufgehoben.

Der MERGE-Befehl kann nur im Zusammenhang mit Disketten benutzt werden. Dabei können nur solche Programme in ein im Arbeitsspeicher befindliches Programm eingebettet werden, die zuvor mit dem Schlüsselwort MERGE abgespeichert wurden (vgl. auch SAVE-Befehl).

Der MERGE-Befehl kann vor allem dazu benutzt werden, um häufiger benötigte Unterprogramme in andere Programme einzubetten. Diese Programme müssen vorher mit dem Schlüsselwort MERGE abgespeichert sein. Ihre Zeilennummern sollten (notfalls mit RES) weit über den Zeilennummern der Programme liegen, in die sie eingebunden werden sollen, um keine Überschneidungen von Programmteilen und damit nicht beabsichtigte Überschreibungen von schon bestehenden Zeilen zu erhalten.

Beispiel:

Das Programm, das unter dem Programmnamen UPROGR und dem Zusatz MERGE abgespeichert wurde, wird in das im Arbeitsspeicher befindliche Programm eingebettet.

```
>MERGE DSK1.UPROGR
```

## MIN

Format:

MIN(numerischer Ausdruck 1, numerischer Ausdruck 2)

Beschreibung:

Die MIN-Funktion ergibt den kleineren der beiden Werte des numerischen Ausdrucks 1 und des numerischen AUsdrucks 2. Sind beide Werte gleich, so erhält man den gemeinsamen Wert. Die numerischen Ausdrücke werden zunächst nach den üblichen Regeln berechnet.

Beispiele:

In Programmzeile 100 wird die Zahl 2 ausgegeben, in 120 der Wert -5.

```
100 PRINT MIN(2,5)
120 PRINT MIN(-3,-5)
```

Inn Programmzeile 140 wird der Variablen K der kleinere der beiden Werte von A und B zugewiesen.

```
140 K=MIN(A,B)
```

## MOTION-Unterprogramm

Format:

```
CALL MOTION(#Sprite-Nummer,Zeilengeschwindigkeit,Spaltengeschwindigkeit
                                                              [,...])
```

Beschreibung:

Mit dem MOTION-Unterprogramm wird die Bewegung eines oder mehrerer Sprites durch die Angabe der Geschwindigkeiten in Zeilen- und Spaltenrichtung geändert. Zeilengeschwindigkeit und Spaltengeschwindigkeit sind numerische Ausdrücke, deren Werte zwischen -128 und +127 liegen dürfen. Je weiter die Werte der numerischen Ausdrücke von 0 entfernt sind (positiv wie negativ), um so schneller bewegt sich ein Sprite und um so langsamer, je näher die angegebenen Geschwindigkeiten bei 0 liegen. Die Bewegung der Sprites wird also letztlich durch die Kombination der beiden numerischen Ausdrücke für die Geschwindigkeiten in Zeilen- und Spaltenrichtung bestimmt. So bewegt sich ein Sprite mit der Zeilengeschwindigkeit 10 und der Spaltengeschwindigkeit -10 auf dem Bildschirm langsam schräg nach links unten. Gelangt ein Sprite an den Bildschirmrand, verschwindet er und taucht an der entsprechenden Stelle am gegenüberliegenden Bildschirmrand wieder auf und setzt von dort seine Bewegung in der angegebenen Richtung fort.

Beispielprogramm:

Ein vergrößertes Ausrufezeichen bewegt sich mit zunehmender Geschwindigkeit nach rechts unten und anschließend mit abnehmender Geschwindigkeit nach links oben.

Das Programm endet mit <FCTN-4> (CLEAR).

```
100 REM ***** BEWEGUNG *****
110 CALL CLEAR
120 CALL SPRITE(#1,33,3,170,200)
130 CALL MAGNIFY(2)
140 GOSUB 210
150 FOR GESCHW=-30 TO 30
160 IF GESCHW=0 THEN GOSUB 210
170 DISPLAY AT(24,1):"GESCHWINDIGKEIT:";GESCHW
180 CALL MOTION(#1,GESCHW,GESCHW)
190 NEXT GESCHW
200 GOTO 140
210 DISPLAY AT(24,1):"RUHE!"
220 CALL MOTION(#1,0,0)
230 CALL SOUND(500,220,4,440,4,880,4)
240 FOR VERZOEG=1 TO 750
250 NEXT VERZOEG
260 RETURN
```

## NEW

Format:

```
NEW
```

Beschreibung:

Der NEW-Befehl löscht den Speicher und den Bildschirm und bereitet den Computer für die Eingabe eines neuen Programms vor. Alle Variablenwerte werden gelöscht, ebenso alle selbstdefinierten Zeichen. Alle offenen Dateien werden geschlossen (s. auch OPEN-Befehl). Die Wirkung des TRACE-Befehls und des BREAK-Befehls sind aufgehoben. Nach Eingabe des NEW-Befehls meldet der Computer sich mit TI BASIC READY oder * READY *, je nach der von Ihnen beim Erscheinen der Titelgrafik gewählten Sprache.Das Dialogzeichen (>) und der blinkende Positionszeiger zeigen an, daß nun ein anderer Befehl eingegeben werden kann.

Speichern Sie gegebenenfalls ein im Computer befindliches Programm v o r der Eingabe des NEW-Befehls ab, da es sonst unwiderruflich verloren ist.

# NEXT

Format:

NEXT Kontrollvariable

Beschreibung:

Der NEXT-Befehl tritt in einem Programm immer zusammen mit dem FOR-TO-STEP -Befehl auf. Er ist der letzte Befehl (mit der höchsten Programmzeilennummer) in einer Zählschleife. Die Kontrollvariable des NEXT-Befehls muß mit der Kontrollvariablen des FOR-TO-STEP-Befehls, auf den er sich bezieht, übereinstimmen. Der NEXT-Befehl darf nicht in einem IF-THEN-ELSE-Befehl auftreten.

Der NEXT-Befehl kontrolliert, ob eine Zählschleife wiederholt wird. Jedesmal, wenn der NEXT-Befehl ausgeführt wird, wird die Kontrollvariable um den Wert verändert, der hinter dem STEP-Schlüsselwort des FOR-TO-STEP-Befehls steht, oder um 1, wenn das STEP-Schlüsselwort fehlt. Wenn der Wert der Kontrollvariablen zwischen dem Anfangs- und dem Endwert liegt, wird die Zählschleife wiederholt, andernfalls wird der Programmablauf mit dem Befehl, der dem NEXT-Befehl folgt, fortgesetzt. Nach Beendigung der Zählschleife besitzt die Kontrollvariable den ersten Wert außerhalb des Bereichs des FOR-TO-STEP-Befehls.

Im Zusammenhang mit dem ON BREAK-, ON WARNING- und RETURN-Befehl kann NEXT auch als Schlüsselwort auftreten (vgl. den jeweiligen Befehl).

Beispielprogramm:

S. FOR-TO-STEP-Befehl!

## NUMBER

Format:

```
NUMBER[Startzeile][,Schrittweite]
NUM[Startzeile][,Schrittweite]
```

Beschreibung:

Durch Eingabe des Direktbefehls NUMBER wird der COMMAND-Modus verlassen und der NUMBER-Modus erreicht (s. "Bedienung", Kap.4). Im NUMBER-Modus werden die Programmzeilen-Nummern automatisch vom Computer erzeugt. Dies stellt eine Erleichterung für den Benutzer dar, da bei der Eingabe von Programmzeilen nicht jedesmal die Zeilennummer eingegeben werden muß.

Durch die Angabe einer Startzeile wird die erste vom Computer zu erzeugende Zeilennummer benannt, mit der Schrittweite die folgenden, die durch Addition der Schrittweite zur jeweils aktuellen Zeilennummer entsteht.

Werden Startzeile und Schrittweite nicht angegeben, dann wird die automatische Zeilennumerierung beginnend mit der Startzeilennummer 100 in Zehnerschritten durchgeführt: jedesmal, wenn eine vollständige Programmzeile mit <ENTER> an den Computer übergeben wird, erscheint eine um 10 höhere Programmzeilen-Nummer. Sollte bei diesem Vorgehen eine bereits im Arbeitsspeicher des Computers vorhandene Programmzeile angesprochen werden, dann wird diese zusammen mit der betreffenden Zeilennummer auf dem Bildschirm ausgegeben. Sie läßt sich dann editieren oder - durch Betätigung von <ENTER> - unverändert übernehmen. Der NUMBER-Modus kann verlassen werden, indem nach Erzeugung einer Programmzeilen-Nummer sofort <ENTER> gedrückt wird, sofern diese Zeile leer, d.h. nicht bereits im Computer vorhanden ist, oder durch die spezielle Tastenfunktion CLEAR (<FCTN-4>).

Beispiele:

Beachten Sie auch die Ausführungen zur Erstellung des Beispielprogramms im Kapitel "Programmierung"!

Die folgenden Zeilen bilden kein Programm, sondern veranschaulichen die Erstellung eines solchen mit Hilfe des NUMBER-Befehls. Die selbst zu schreibenden Eingaben sind unterstrichen, nach der Fertigstellung einer jeden Zeile ist <ENTER> zu drücken:

```
Der NUMBER-Modus wird aufgerufen        >NUM
(Startzeile 100, Schrittweite 10).
                                        100 X=4
                                        110 Z=5
                                        120
Der NUMBER-Modus wird verlassen.
Erneuter Aufruf: Startzeile 110,        >NUM 110
die editiert wird.                      110 Z=11
                                        120 PRINT (Y+X)/Z
                                        130
Der NUMBER-Modus wird verlassen und
erneut, diesmal mit Startzeile 105      >NUM 105,5
und Schrittweite 5, aufgerufen.         105 Y=7
Programmzeile 110 wird unverändert      110 Z=11
belassen.
                                        115
Der NUMBER-Modus wird verlassen.
                                        100 X=4
Vollständiges Programm:                 105 Y=7
(mit LIST-Befehl erstellt)              110 Z=11
                                        120 PRINT (Y+X)/Z
```

## OLD

Format:

```
OLD  Gerätename[.Programmname]
```

Beschreibung:

Der OLD-Befehl lädt ein Programm von dem mit Gerätename bezeichneten Gerät in den Arbeitsspeicher des Computers. Dieses Programm muß vorher mit dem SAVE-Befehl auf dem entsprechenden Gerät abgespeichert worden sein. Bei der Ausführung des OLD-Befehls werden vom Computer sämtliche noch offenen Dateien geschlossen und ein eventuell im Arbeitsspeicher befindliches Programm gelöscht.

Ist der Gerätename CS1 oder CS2, also einer der beiden möglichen Cassetten-Recorder, dann muß kein Programmname angegeben werden. Es wird das Programm in den Arbeitsspeicher geladen, das sich gerade auf der Cassette befindet. Die Anweisungen zur Bedienung des Cassetten-Recorders erscheinen auf dem Bildschirm.

Beispiele:

Der Befehl OLD CS1 bewirkt das Laden eines auf der Cassette befindlichen Programms.

```
>OLD CS1
```

Beim Laden eines Programms von einer Diskette muß der entsprechende Programmname angegeben werden, hier beispielsweise PROGR1.

```
>OLD DSK1.PROGR1
```

## ON BREAK

Format:

```
ON BREAK STOP
ON BREAK NEXT
```

Beschreibung:

Mit dem ON BREAK-Befehl wird bestimmt, was geschehen soll, wenn während der Programmausführung eine Stoppstelle erreicht wird. Fehlt der ON BREAK-Befehl, dann wird der Programmablauf an einer Stoppstelle unterbrochen und die Standardmeldung BREAKPOINT in *** (Stoppstelle in Zeile mit der Nummer ***) ausgegeben. Dies geschieht auch mit dem Befehl ON BREAK STOP. Eine Alternative dazu ist der Befehl ON BREAK NEXT, bei dem die Programmausführung ohne Unterbrechung mit der nächsten Programmzeile fortgesetzt wird.

Der ON BREAK NEXT-Befehl kann dazu benutzt werden, Stoppstellen zu übergehen, die in ein Programm zur Fehlersuche eingefügt worden sind. Dabei dürfen die Stoppstellen nur mit dem Befehl "BREAK Zeilennummer" erzeugt werden, da der ON BREAK NEXT-Befehl auf einen BREAK-Befehl ohne Zeilennummer unwirksam ist. Ist allerdings der ON BREAK NEXT-Befehl wirksam, so kann ein Programm bis zu seiner Aufhebung oder dem Programmende nicht mehr mit <FCTN-4> (CLEAR) unterbrochen werden. In diesem Fall kann das Programm lediglich mit <FCTN-=> (QUIT) unterbrochen werden, was jedoch gleichzeitig zum Verlust des Programms und zu Schwierigkeiten mit externen Dateien führt.

Beispielprogramm:

In Programmzeile 120 wird eine Stoppstelle in Zeile 170 gesetzt. Diese wird jedoch durch den Befehl in Zeile 130 aufgehoben. Der BREAK-Befehl in Zeile 150 wird wirksam, da er keine Zeilennummer enthält. Das Programm kann mit dem CONTINUE-Befehl fortgesetzt werden. Während des Ablaufs der Zählschleife (Zeilen 160 bis 180) ist die Tastenfunktion <FCTN-4) wegen der Programmzeile 130 wirkungslos. Dies wird durch Programmzeile 190 wieder aufgehoben.

```
100 REM ***** BEISPIEL ZUM ON BREAK-BEFEHL *****
110 CALL CLEAR
120 BREAK 170
130 ON BREAK NEXT
140 PRINT "DIE FOLGENDE STOPPSTELLE IST VORPROGRAMMIERT"
145 PRINT "(WEITER MIT <CON(TINUE)>"
150 BREAK
160 FOR A=1 TO 100
170 PRINT "<FCTN-4> IST OHNE WIRKUNG"
180 NEXT A
190 ON BREAK STOP
200 PRINT
210 PRINT "DIE TASTENFUNKTION <FCTN-4> IST JETZT WIEDER WIRKSAM."
```

## ON ERROR

Format:

```
ON ERROR STOP
ON ERROR Zeilennummer
```

Beschreibung:

Mit dem ON ERROR-Befehl wird bestimmt, was geschehen soll, wenn während der Programmausführung ein Fehler auftritt. Ohne diesen Befehl erscheint bei einem Fehler die Standardfehlermeldung, und die Programmausführung wird unterbrochen. Dies geschieht auch bei dem Befehl ON ERROR STOP. Die Alternative dazu ist die Angabe einer Zeilennummer, in der die Programmausführung im Falle eines Fehlers fortgesetzt werden soll. Die angegebene Zeilennummer muß dann den Beginn einer Unterprogrammschleife darstellen, entsprechend dem GOSUB-Befehl. Diese Unterprogrammschleife muß mit einem RETURN-Befehl enden (vgl. RETURN-Befehl). Dabei muß wie bei den GOTO- und GOSUB-Befehlen der Sprung in ein oder aus einem Unterprogramm ausgeschlossen werden.

Ist einmal in einem Programm ein Fehler eingetreten und mit einem ON ERROR -Befehl in einer Unterprogrammschleife abgearbeitet worden, so werden alle folgenden Fehler wieder normal behandelt, d.h. wie mit dem Befehl ON ERROR STOP. Sollen weitere auftretende Befehle anders behandelt werden, muß erneut ein ON ERROR-Befehl mit einer entsprechenden Zeilennummer ausgeführt werden.

Beispielprogramm:

In dem Programm ist in Zeile 140 ein Fehler eingebaut (vgl. VAL-Befehl). Aufgrund der Programmzeile 120 erfolgt während des Programmablaufs nach dem Auftreten dieses Fehlers einen Sprung in die Unterprogrammschleife ab Zeile 170.

```
100 REM ***** BEISPIEL ZUM ON ERROR-BEFEHL *****
110 CALL CLEAR
120 ON ERROR 170
130 X$="A"
140 X=VAL(X$)
150 PRINT X;" ZUM QUADRAT IST ";X*X
160 STOP
170 REM ***** FEHLER-UNTERPROGRAMMSCHLEIFE *****
190 CALL ERR(CODE,TYP,GEWICHT,ZEILE)
200 IF ZEILE<>140 THEN RETURN 240
210 IF CODE<>74 THEN RETURN 240
220 X$="5"
230 RETURN
```

# ON — GOSUB

Format:

ON numerischer Ausdruck GOSUB Zeilennummer [,...]
ON numerischer Ausdruck GO SUB Zeilennummer [,...]

Beschreibung:

Der ON-GOSUB-Befehl bewirkt nach der Auswertung des numerischen Ausdrucks einen Sprung in eine Unterprogrammschleife, die mit der jeweils angegebenen Zeilennummer beginnt. Dabei hängt die Unterprogrammschleife vom Wert des numerischen Ausdrucks ab. Ist dieser 1, so wird der Programmablauf in der 1. angegebenen Zeilennummer fortgesetzt, ist der Wert 2, in der 2. angegebenen Zeilennummer usw. Im Gegensatz zum GOSUB-Befehl kann also mit dem ON-GOSUB-Befehl eine Auswahl von Unterprogrammschleifen getroffen werden. Hierbei werden meistens weniger Programmzeilen benötigt als bei einer Auswahl mit dem IF-THEN-ELSE-Befehl.

Ist bei der Auswertung der gegebenenfalls gerundete Wert des numerischen Ausdrucks kleiner als 1 oder größer als die Anzahl der angegebenen Zeilennummern, so erscheint die Fehlermeldung BAD VALUE IN ### (falscher Wert in Zeile ###). Existiert die angegebene Zeilennummer nicht in dem Programm, so erscheint die Fehlermeldung BAD LINE NUMBER (falsche Zeilennummer). In beiden Fällen wird der Programmablauf unterbrochen.

Beispiel:

```
100 ON X-3 GOSUB 200,500,300
```

Für X=4 springt der Programmablauf in die Unterprogrammschleife, die in Programmzeile 200 beginnt, für X=5 in der Zeile 500 und für X=6 in der Zeile 300. Für andere Werte von X erscheint eine Fehlermeldung.

# ON — GOTO

Format:

ON numerischer Ausdruck GOTO Zeilennummer [,...]
ON numerischer Ausdruck GO TO Zeilennummer [,...]

Beschreibung:

Der ON-GOTO-Befehl bewirkt nach der Auswertung des numerischen Ausdrucks einen Sprung in die angegebene Zeilennummer. Dabei hängt die Zeilennummer vom Wert des numerischen Ausdrucks ab. Ist dieser 1, so wird der Programmablauf in der 1. angegebenen Zeilennummer fortgesetzt, ist der Wert 2, in der 2. angegebenen Zeilennummer usw. Im Gegensatz zum GOTO-Befehl kann also mit dem ON-GOTO-Befehl eine Auswahl von Zeilennummern getroffen werden. Hierbei werden meistens weniger Programmzeilen benötigt als bei einer Auswahl mit dem IF-THEN-ELSE-Befehl.

Ist bei der Auswertung der gegebenenfalls gerundete Wert des numerischen Ausdrucks kleiner als 1 oder größer als die Anzahl der angegebenen Zeilennummern, so erscheint die Fehlermeldung BAD VALUE IN ### (falscher Wert in Zeile ###). Existiert die angegebene Zeilennummer nicht in dem Programm, so erscheint die Fehlermeldung BAD LINE NUMBER (falsche Zeilennummer). In beiden Fällen wird der Programmablauf unterbrochen.

Beispiel:

Für X=4 springt der Programmablauf in Programmzeile 200, für X=5 in Zeile 500 und für X=6 in Zeile 300. Für andere Werte von X erscheint eine Fehlermeldung.

```
100 ON X-3 GOTO 200,500,300
```

Beispielprogramm:

Dieses Programm entspricht dem Beispielprogramm zum RUN-Befehl, es nutzt jedoch die Möglichkeit der Auswahl durch den ON-GOTO-Befehl in Programmzeile 210.

```
100 REM ***** MENUETECHNIK *****
110 CALL CLEAR
120 PRINT "     PROGRAMMUEBERSICHT"
130 PRINT "     ------------------"
140 PRINT "           1   PROGR1"
150 PRINT "           2   PROGR2"
160 PRINT "           3   PROGR3"
170 PRINT "           4   ENDE"
180 PRINT
190 INPUT "           W A H L ? ":WAHL
200 IF WAHL<1 OR WAHL>4 THEN 110
210 ON WAHL GOTO 230,240,250,220
220 STOP
230 RUN "DSK1.PROGR1"
240 RUN "DSK1.PROGR2"
250 RUN "DSK1.PROGR3"
```

## ON WARNING

Format:

```
ON WARNING PRINT
ON WARNING STOP
ON WARNING NEXT
```

Beschreibung:

Mit dem ON WARNING-Befehl wird bestimmt, was geschehen soll, wenn während der Programmausführung ein leichter Fehler, der eine Warnung - aber keine Programmunterbrechung! - verursacht, auftritt.

Ohne ON WARNING-Befehl wird beim Auftreten eines leichten Fehlers eine Warnmeldung ausgegeben. Dies geschieht auch bei dem Befehl ON WARNING PRINT. Eine Alternative ist ON WARNING STOP, die nach der Ausgabe der Warnmeldung eine Programmunterbrechung bewirkt. Soll beim Auftreten eines leichten Fehlers keine Warnmeldung ausgegeben werden, so lautet der Befehl ON WARNING NEXT. Der Computer geht dann einfach zur nächsten Programmzeile über.

Beispielprogramm:

Programmzeile 120 bewirkt, daß der leichte Fehler in Programmzeile 130 (durch 0 darf nicht dividiert werden) unbeachtet bleibt, während der Fehler in Zeile 150 aufgrund des Befehls in Zeile 140 als Warnung ohne Programmunterbrechung ausgegeben wird. Programmzeile 160 bewirkt eine Programmunterbrechung in Zeile 170, so daß die Programmzeile 180 nicht erreicht wird.

```
100 REM ***** BEISPIEL ZUM ON WARNING-BEFEHL *****
110 CALL CLEAR
120 ON WARNING NEXT
130 PRINT 130,5/0
140 ON WARNING PRINT
150 PRINT 150,5/0
160 ON WARNING STOP
170 PRINT 170,5/0
180 PRINT 180
```

# OPEN

Format:

OPEN #Dateinummer: Dateibezeichnung
[,Eröffnungsmodus][,Dateiorganisation][,Datentyp][,Datensatztyp]

Beschreibung:

Der OPEN-Befehl bereitet die Benutzung einer externen Datei, das ist eine Datei auf einem Peripheriegrät oder das Peripheriegerät selbst (z.B. der Drucker), in einem Programm vor. Dabei wird eine Verbindung zwischen der Dateinummer, die in den anderen Befehlen zur Dateiverwaltung (CLOSE, PRINT, INPUT, EOF, RESTORE) angegeben wird, und der Dateibezeichnung mit möglichen Informationen über die Art der Datei (s. Optionen) hergestellt. Stimmen die Dateibezeichnung und die weiteren Optionen nicht mit den Eigenschaften einer schon existierenden Datei überein, so wird die Datei nicht geöffnet, und es wird ein Eingabe/Ausgabe-Fehler (I/O-ERROR) ausgegeben und der Programmablauf unterbrochen.

#Dateinummer: Die Dateinummer wird als Nummernzeichen (#) gefolgt von einem numerischen Ausdruck eingegeben. Der Wert des numerischen Ausdrucks muß eine ganze Zahl zwischen 0 und 255 sein, er wird gegebenenfalls gerundet. Es dürfen keine zwei Dateien mit derselben Dateinummer geöffnet werden. Die Dateinummer 0 ist für Bildschirm und Tastatur reserviert und kann nicht für andere Dateien benutzt werden, sie ist immer geöffnet.

Dateibezeichnung = Gerätename[.Dateiname]
Als Dateibezeichnung kann jeder beliebige String-Ausdruck verwendet werden, dessen Wert eine gültige Dateibezeichnung ergibt. Gültige Gerätenamen sind u.a.
"CS1", "DSK1", "TP", "RS232".
Die möglichen Cassetten-Recorder werden mit den Gerätenamen "CS1" und "CS2" angesprochen. Dies sind auch gleichzeitig die vollständigen Dateibezeichnungen, da der Dateiname bei Cassettenbetrieb entfällt. Wenn der Computer den OPEN-Befehl für eine Datei auf einer Cassette ausführt, erscheinen auf dem Bildschirm die entsprechenden Anweisungen für die Bedienung des Cassetten-Recorders.

Die möglichen Diskettenlaufwerke werden mit "DSK1", "DSK2" und "DSK3" angesprochen. Hierbei ist als Gerätename auch die Bezeichnung
DSK.Diskettenname
möglich, wobei der Diskettenname die Bezeichnung der Diskette ist, die diese bei ihrer Initialisierung bzw. Umbenennung (vgl. DISK MANAGER Modul) erhalten hat. Der Computer sucht dann, beginnend bei Laufwerk 1, alle Laufwerke nach der angegebenen Diskette ab.

Bei Diskettenbetrieb sucht der Computer als letztes nach dem angegebenen Dateinamen auf der mit Gerätename bezeichneten Diskette, oder - bei Eröffnung einer neuen Datei - er trägt diesen Namen in das Inhaltsverzeichnis der Diskette (Diskettenkatalog) ein. Als Dateiname kann jeder String-Ausdruck verwendet werden, dessen Wert einen zulässigen Dateinamen ergibt. Bei Anwendung von String-Konstanten muß er in Anführungszeichen gesetzt werden.

Dateinummer und Dateibezeichnung müssen im OPEN-Befehl erscheinen, die anderen Optionen können in beliebiger Reihenfolge angegeben werden. Fehlen Optionen, so werden bestimmte Standardoptionen vom Computer angenommen.

## OPEN

Optionen:

Eröffnungsmodus: Der Computer wird angewiesen, die Datei im
INPUT
OUTPUT
UPDATE
APPEND
-Modus zu verarbeiten:
INPUT(Eingabe)-Dateien können nur gelesen werden.
OUTPUT(Ausgabe)-Dateien können nur beschrieben werden.
UPDATE(Änderungs)-Dateien können gelesen und beschrieben werden.
APPEND(Anfügungs)-Dateien sind nur vom Typ RELATIVE (vgl. unten) und können nur beschrieben werden, und zwar nur an ihrem Ende. In diesem Modus kann nicht auf bereits vorhandene Datensätze der Datei zugegriffen werden.

Ist eine Datei geschützt, dann kann sie nur gelesen werden (im INPUT-Modus). Fehlt der Eröffnungsmodus, so setzt der Computer automatisch den UPDATE-Modus voraus.

Dateiorganisation: Die Dateien können
SEQUENTIAL oder RELATIVE
organisiert sein.
SEQUENTIAL-Dateien werden nacheinander (sequentiell) verarbeitet, d.h. beschrieben oder gelesen.
RELATIVE-Dateien können in beliebiger Reihenfolge verarbeitet werden (wahlfreier Direktzugriff). Eine sequentielle Verarbeitung ist ebenfalls möglich.

Der Angabe der Dateiorganisation kann ein numerischer Ausdruck folgen, dessen Wert die Anfangszahl der zu verarbeitenden Datensätze angibt. Fehlt die Angabe der Dateiorganisation, so setzt der Computer automatisch eine SEQUENTIAL-Datei voraus.

Datentyp: Der Datentyp gibt das Format der Daten in der Datei an:
DISPLAY oder INTERNAL.
DISPLAY-Daten bestehen aus ASCII-Zeichen und sind vom Benutzer lesbar.
INTERNAL-Daten bestehen aus Binärzeichen und sind oft nur vom Computer lesbar. Sie benötigen weniger Speicherplatz und sind vom Computer schneller zu verarbeiten.

Sollen die Daten lesbar, z.B. über Drucker, ausgegeben werden, so ist der Datentyp DISPLAY vorzuziehen. Bei Dateien auf Cassetten bzw. Disketten ist der bessere Datentyp INTERNAL. Fehlt die Angabe des Datentyps, so setzt der Computer automatisch DISPLAY-Daten voraus.

Datensatztyp: Der Datensatztyp gibt Auskunft über die Länge (Anzahl der Bytes) der Datensätze in der Datei:
FIXED oder VARIABLE.
FIXED-Datensätze haben alle dieselbe Länge, sie werden gegebenenfalls abgeschnitten oder mit Leerzeichen aufgefüllt.
VARIABLE-Datensätze können in ihrer Länge unterschiedlich sein.

## OPEN

Dem Datensatztyp kann ein numerischer Ausdruck folgen, dessen Wert die maximale Länge des Datensatzes angibt. Wird keine Datensatzlänge angegeben, so nimmt der Computer automatisch als Datensatzlänge

80 für Disketten,
64 für Cassetten,
80 für RS 232-Schnittstelle und
32 für Thermodrucker an.

SEQUENTIAL-Dateien können VARIABLE- oder FIXED-Datensätze besitzen, bei RELATIVE-Dateien muß der Datensatztyp FIXED sein. Fehlt der Datensatztyp, setzt der Computer automatisch VARIABLE bei SEQUENTIAL-Dateien und FIXED bei RELATIVE-Dateien voraus.

---

Beispielprogramm:

Hierbei handelt es sich nicht um ein "echtes" Programm, sondern es sollen die Awendungen des OPEN- und CLOSE-Befehls verdeutlicht werden:

```
100 OPEN #1:"CS2",INPUT,FIXED
 :
 :
200 OPEN #2:"TP",OUTPUT
 :
 :
300 CLOSE #1
 :
 :
400 OPEN #3:"DSK1.HILFE",INTERNAL,RELATIVE,FIXED 100
 :
 :
500 CLOSE #2
 :
 :
600 CLOSE #3:DELETE
```

In Programmzeile 100 wird die Datei #1 geöffnet, die sich auf einer Cassette im Cassetten-Recorder 2 befindet. Von ihr können nur Datensätze fester Länge (hier: 64) einngelesen werden. Die Dateiorganisation ist SEQUENTIAL und der Datentyp DISPLAY.

In Programmzeile 200 wird zusätzlich als Datei #2, der Thermodrucker, zur Ausgabe der Daten geöffnet (OUTPUT). Der Dateityp ist SEQUENTIAL, der Datentyp DISPLAY und die maximale Datensatzlänge 32.

In Programmzeile 300 wird die Datei #1 (auf dem Cassetten-Recorder) wieder geschlossen, in den nachfolgenden Programmzeilen können keine Daten mehr von dieser Datei eingelesen werden.

In Programmzeile 400 wird die Datei #3 auf der Diskette im Laufwerk 1 mit Namen HILFE eröffnet. Die Datei ist im wahlfreien Direktzugriff (RELATIVE) organisiert, ihre Datensätze sind vom Typ INTERNAL und haben eine maximale Länge von 100 (Bytes).

In Programmzeile 500 wird die Datei #2 (der Thermodrucker) geschlossen und in Zeile 600 die Datei auf der Diskette. Die letzte Datei (#3) wird auf Grund des DELETE-Schlüsselworts gleichzeitig wieder gelöscht.

# OPTION BASE

Format:

```
OPTION BASE 0
OPTION BASE 1
```

Beschreibung:

Der OPTION BASE-Befehl gibt die Möglichkeit, in einem Datenfeld den Index des ersten Elementes einer Dimension auf 1 anstatt auf 0 zu setzen. Fehlt der OPTION BASE-Befehl, gilt automatisch OPTION BASE 0. Die Programmzeile mit dem OPTION BASE-Befehl muß eine niedrigere Zeilennummer als alle Datenfelder betreffende Programmzeilen haben, einschließlich des DIM-Befehls In einem Programm ist nur ein OPTION BASE-Befehl zulässig, und dieser gilt für alle Datenfelder des Programms. Der OPTION BASE-Befehl darf nicht in einem IF-THEN-ELSE-Befehl erscheinen.

Beipiel:

Programmzeile 100 setzt den kleinsten Index der Elemente aller im Programm auftretenden Datenfelder auf 1.

```
100 OPTION BASE 1
```

## PATTERN-Unterprogramm

Format:

```
CALL PATTERN(#Sprite-Nummer,Zeichencode[,...])
```

Beschreibung:

Das PATTERN-Unterprogramm ändert das Aussehen (Muster) eines Sprites mit der angegebenen Sprite-Nummer, ohne seine sonstigen Eigenschaften zu beeinflussen. Der PATTERN-Befehl ist gleichzeitig auf mehrere Sprites anwendbar. Der Sprite-Nummer des zu ändernden Sprites ist das Nummernzeichen (#) vorangestellt. Der Zeichencode ist der des neuen Zeichens, das das Aussehen des Sprites im weiteren Programmablauf bestimmt, und ist ein numerischer Ausdruck, dessen Wert zwischen 32 und 143 einschließlich liegt (vgl. CHAR-Befehl).

Beispielprogramm:

Es wird zunächst ein Boden gebildet, auf dem sich ein einem Rad ähnliches Sprite nach rechts zu bewegen beginnt, dessen "Speichen" sich zu drehen scheinen.

Das Programm endet mit <FCTN-4> (CLEAR).

```
100 REM ***** RAD *****
110 CALL CLEAR
120 REM ***** BODEN *****
130 CALL COLOR(12,16,16)
140 FOR ZEILE=19 TO 24
150 CALL HCHAR(ZEILE,1,120,32)
160 NEXT ZEILE
170 REM ***** ZEICHENDEFINITION *****
180 R1$="0107182121414 1FFFF4141212119070080E09884848282FFFF8282848498E000"
190 R2$="0106182030 5C468181424624 2C1807008060183424624281816 23A0C0418E000"
200 R3$="0106182C2446428181465C3020180700806018040C3A628181426224 3418E000"
210 CALL CHAR(96,R1$)
220 CALL CHAR(100,R2$)
230 CALL CHAR(104,R3$)
240 CALL SPRITE(#1,96,5,130,1,0,8)
250 CALL MAGNIFY(3)
260 REM ***** BEWEGUNG *****
270 FOR ZEICHEN=96 TO 104 STEP 4
280 CALL PATTERN(#1,ZEICHEN)
290 FOR VERZOEG=1 TO 5 :: NEXT VERZOEG
300 NEXT ZEICHEN
310 GOTO 270
```

## PEEK-Unterprogramm

Format:

CALL PEEK(Adresse, numerische Variablenliste)

Beschreibung:

Das PEEK-Unterprogramm wird zusammen mit INIT, LINK und LOAD benötigt, um Assembler-Unterprogramme zu benutzen.

Mit dem PEEK-Unterprogramm können die Inhalte von Speicherstellen im Arbeitsspeicher des Computers direkt gelesen werden. Dabei werden dann den numerischen Variablen der Variablenliste die Werte der Bytes in der Speicherstelle, die mit Adresse bezeichnet wurde, und den folgenden Speicherstellen zugeordnet. Die Werte reichen von 0 bis 255. Mit dem PEEK-Befehl sollte nicht auf den gesamten Speicherbereich des Computers zurückgegriffen werden, sondern nur auf den Bereich ab Speicheradresse 16383 (HEX 2000 bis HEX 3FFF). Außerhalb dieses Bereiches können Probleme entstehen.

Der PEEK-Befehl kann auch ohne Assembler-Unterprogramme benutzt werden, die dann dabei gewonnenen Informationen sind jedoch von nur geringem Nutzen. Unsachgemäßer Gebrauch von PEEK kann den Computer "sperren", was ein Ausschalten und anschließendes Wiedereinschalten des Computers für den weiteren Gebrauch erforderlich macht.

Genauere Informationen zur Anwendung der INIT-, LINK-, LOAD- und PEEK-Unterprogramme sind in den Dokumentationen zu den Assembler-Unterprogrammen (auf Diskette oder Cassette erhältlich) zu finden.

Beispiel:

Programmzeile 100 ordnet den numerischen Variablen X1, X2, X3 und X4 die Bytewerte in den Speicherstellen 8912, 8913, 8914 und 8915 zu.

```
100 CALL PEEK(8192,X1,X2,X3,X4)
```

# PI

## Format:

PI

## Beschreibung:

Die PI-Funktion setzt die Variable PI zu

PI = 3.14159265359

fest.

## Beispiel:

In Programmzeile 100 wird der Variablen UMFANG der numerische Ausdruck 2*PI*R zugeordnet, was dem Umfang eines Kreises mit dem Radius R entspricht.

```
100 UMFANG=2*PI*R
```

## POS

Format:

POS(String 1, String 2, numerischer Ausdruck)

Beschreibung:

Die POS-Funktion gibt die Stelle des ersten Auftretens von String 2 in String 1 an. Die Suche nach String 2 beginnt an der Stelle, die dem Wert des numerischen Ausdrucks entspricht. Der numerische Ausdruck wird nach den üblichen Regeln ausgewertet und - wenn nötig - auf eine ganze Zahl gerundet. Wird String 2 in String 1 nicht gefunden, so gibt die POS-Funktion den Wert 0 an, ebenso, wenn der Wert des numerischen Ausdrucks größer ist als die Anzahl der Zeichen in String 1. Ist der Wert des numerischen Ausdrucks kleiner als Null, erscheint die Fehlermeldung BAD VALUE (falscher Wert) und der Programmablauf wird unterbrochen.

Beispiel:

In Programmzeile 100 wird der Variablen Z die Stelle des ersten Auftretens des Strings Y$ im String X$ als ganze Zahl zugewiesen.

```
100 Z=POS(X$,Y$,LEN(Y$))
```

Beispielprogramm:

S. SEG$-Befehl!

# POSITION-Unterprogramm

Format:

```
CALL POSITION(#Sprite-Nummer, Punktzeile, Punktspalte[,...])
```

Beschreibung:

Mit Hilfe des POSITION-Unterprogramms kann die Bildschirmposition eines oder mehrerer durch die Sprite-Nummer bestimmten Sprites festgestellt werden. Dabei werden den numerischen Variablen Punktzeile und Punktspalte vom Computer Werte zwischen 1 und 256 einschließlich zugeordnet. Die Ausgabe kann für mehrere Sprites gleichzeitig erfolgen. Als Position eines Sprites gilt seine linke obere Ecke.

Ist ein Sprite nicht definiert, so wird den Variablen Punktzeile und Punktspalte der Wert 0 zugewiesen. Die Bewegung eines Sprites wird durch die Abfrage seiner Bildschirmposition mit dem POSITION-Unterprogramm nicht beeinflußt, d.h. zum Zeitpunkt der Ausgabe seiner Position kann er sich je nach seiner Geschwindigkeit inzwischen um eine bestimmte Entfernung fortbewegt haben.

Beispiel:

Programmzeile 100 fragt die Bildschirmposition der Sprites #1 und #2 ab, die dann in den numerischen Variablen X1, Y1 und X2, Y2 zur Verfügung stehen.

```
100 CALL POSITION(#1,Y1,X1,#2,Y2,X2)
```

Beispielprogramm:

S. SPRITE-Unterprogramm!

# PRINT

Format:

PRINT [#Dateinummer[,REC Datensatznummer]:]Ausgabeliste

Beschreibung:

Der PRINT-Befehl ermöglicht die Ausgabe der Werte der Ausgabeliste auf dem Bildschirm oder einer mit #Dateinummer bezeichneten externen Datei.

Die Elemente der Ausgabeliste können sein:
- numerische Variable/Konstanten,
- String-Variable/-Konstanten,
- numerische oder String-Ausdrücke,
- TAB-Funktionen.

Mehrere Elemente der Ausgabeliste müssen mit einem sog. Ausgabeseparator, das ist ein Komma, Semikolon oder Doppelpunkt, voneinander getrennt werden:

- Das Semikolon bewirkt, daß benachbarte Ausgabeelemente unmittelbar aufeinanderfolgend, d.h. ohne zusätzliche Leerzeichen ausgegeben werden.
- Der Doppelpunkt bewirkt, daß das nächste Ausgabeelement am Anfang der nächsten Zeile oder des nächsten Datensatzes ausgegeben wird.
- Das Komma bewirkt, daß das nächste Ausgabeelement am Anfang der nächsten Ausgabezone oder des nächsten Datensatzes ausgegeben wird. Auf dem Bildschirm beispielsweise sind die insgesamt 28 Schreibstellen in 2 Zonen zu je 14 Schreibstellen aufgeteilt. Ist bei der Ausgabe eines Elementes die nächste Zone schon erreicht, dann wird es in der nächsten noch nicht begonnenen Zone (Datensatz) ausgegeben.

Zahlen werden unabhängig vom Ausgabeseparator mit einer anschließenden Leerstelle ausgegeben. Positive Zahlen erhalten zusätzlich eine vorangehende Leerstelle (anstelle des positiven Vorzeichens). Strings werden ohne voran- und nachgestellte Leerzeichen ausgegeben.

Fehlt die Druckliste, dann wird eine Leerzeile ausgegeben. Mehrere Leerzeilen werden durch eine entsprechende Anzahl von Doppelpunkten hinter dem PRINT-Befehl erreicht. Dabei muß aber je ein Leerzeichen zwischen den Doppelpunkten sein, damit die Ausgabeseparatoren nicht mit den Befehlsseparatoren (::) verwechselt werden.

Wenn eine neue Zeile auf dem Bildschirm ausgegeben wird, erfolgt dies mit dem PRINT-Befehl immer in der untersten (24.) Bildschirmzeile. Alles - außer Sprites -, was sich schon auf dem Bildschirm befindet, wird dann um eine Zeile nach oben verschoben (Bildschirm-Scrolling). Die oberste Bildschirmzeile verschwindet dabei.

Optionen:

Die #Dateinummer ist ein numerischer Ausdruck, deren ganzzahliger Wert die Datei angibt, auf die mit Hilfe des PRINT-Befehls ausgegeben werden soll. Die Datei muß vorher geöffnet worden sein (vgl. OPEN-Befehl). Fehlt die Dateinummer oder ist sie #0, dann erfolgt die Ausgabe auf dem Bildschirm, der nicht als Datei geöffnet werden muß.

Die Datensatznummer ist ein numerischer Ausdruck, dessen Wert den Datensatz angibt, in den die Elemente der Ausgabeliste gebracht werden sollen. Vor der Datensatznummer muß das Schlüsselwort REC stehen. Dies kann nur im Zusammenhang mit Dateien vom Typ RELATIVE geschehen (vgl. OPEN-Befehl).

Ist der Datentyp einer Datei INTERNAL, werden die Elemennte der Ausgabeliste bei Verwendung eines Kommas oder Semikolons direkt nacheinander ausgegeben, beim Datentyp DISPLAY nur bei einem Semikolon.

# PRINT

Beispiele:

Programmzeile 100 erzeugt eine Leerzeile auf dem Bildschirm.

```
100 PRINT
```

Ist ANTWORT$ eine String-Variable, deren Wert "JA" ist, so wird mit Programmzeile 120 der String DIE ANTWORT IST JA auf dem Bildschirm ausgegeben. Beachten Sie, daß dazu das letzte Zeichen in der String-Konstanten ein Leerzeichen sein muß.

```
120 PRINT "DIE ANTWORT IST ";
    ANTWORT$
```

Mit Programmzeile 140 werden die Werte der numerischen Variablen X und Y - der letztere halbiert -, in zwei aufeinanderfolgenden Zeilen auf dem Bildschirm ausgegeben.

```
140 PRINT X;Y/2
```

Programmzeile 160 bewirkt die Ausgabe des Wertes der numerischen Variablen A auf die unter #3 geöffneten Datei, z.B. einen Drucker, während Programmzeile 180 sich auf eine Datei #4 bezieht, die nur vom Typ RELATIVE sein kann, z.B. ein Diskettenlaufwerk.

```
160 PRINT #3:A
180 PRINT #4,REC8:A
```

Bei Programmzeile 200 entsteht eine sog. schwebende Ausgabebedingung durch das letzte Komma. Das bedeutet, daß der nächste PRINT-Befehl in Programmzeile 220 eine Ausgabe in denselben Datensatz (mit Nummer 5) bewirkt.

```
200 PRINT #7,REC5:A,B,
220 PRINT #7:C,D
```

---

Beispielprogramm:

Das Programm demonstriert die unterschiedlichen Wirkungen der Ausgabeseparatoren bei ihrer Anwendung auf dem Bildschirm.

```
100 REM ***** BILDSCHIRMAUSGABE MIT PRINT-BEFEHL *****
110 CALL CLEAR
120 PRINT 1;2;3;4;5;6;7;8;9
130 PRINT
140 PRINT 1,2,3,4,5,6;
150 PRINT 7;8;9
160 PRINT
170 PRINT 1,,2,,3,,
180 PRINT ,4,,5,,6
190 PRINT 1:,2: :3: : :,4
200 END
```

## PRINT USING

Format:

```
PRINT [#Dateinummer,[REC Datensatznummer]]USING String-Ausdruck:Ausgabeliste
PRINT [#Dateinummer,[REC Datensatznummer]]USING Zeilennummer:Ausgabeliste
```

Beschreibung:

Der PRINT USING-Befehl ist genauso aufgebaut wie der PRINT-Befehl. Hinzu kommt die USING-Bedingung, die das Format der Daten in der Ausgabeliste bestimmt. Der String-Ausdruck definiert das Format, wie es im IMAGE-Befehl beschrieben wird (vgl. d.). Die Zeilennummer bezieht sich auf einen IMAGE-Befehl.

Beispiele:

Programmzeile 100 liefert die Ausgabe von 14.30 auf dem Bildschirm.

```
100 PRINT USING "###.##":14.3
```

In Programmzeile 120 werden die Werte der numerischen Variablen A und B auf eine Datei #2, das kann auch ein Drucker sein, in einem Format, das in Programmzeile 80 mit einem IMAGE-Befehl definiert ist, ausgegeben.

```
120 PRINT #2,USING 80:A,B
```

# RANDOMIZE

Format:

RANDOMIZE numerischer Ausdruck

Beschreibung:

Der RANDOMIZE-Befehl wird in Verbindung mit der RND-Funktion verwendet. Fehlt in einem Programm der RANDOMIZE-Befehl, so wird mit der RND-Funktion bei jedem Programmablauf dieselbe Folge von Pseudo-Zufallszahlen erzeugt. Verwendet man den RANDOMIZE-Befehl ohne numerischen AUsdruck, so ist die bei jedem Programmablauf erzeugte Folge von Pseudo-Zufallszahlen unbestimmt.

Enthält der RANDOMIZE-Befehl einen numerischen Ausdruck, so wird immer dann dieselbe Folge von Pseudo-Zufallszahlen erzeugt, wenn der Wert des numerischen Ausdrucks derselbe ist, verschiedene Werte bedingen also verschiedene Folgen von Zufallszahlen - um genau zu sein: solange die ersten beiden Bits der internen Darstellung des Wertes des numerischen Ausdrucks nicht übereinstimmen.

Beispielprogramm:

Das folgende Programm zeigt eine Anwendung des RANDOMIZE-Befehls. Die Folge der 10 erzeugten Zufallszahlen hängt von dem STARTWERT ab.

Das Programm endet mit <FCTN-4> (CLEAR).

```
100 REM ***** ZUFALLSZAHLEN BEI VERSCHIEDENEM STARTWERT *****
110 CALL CLEAR
120 INPUT "STARTWERT: ":STARTWERT
130 RANDOMIZE STARTWERT
140 FOR I=1 TO 10 :: PRINT I;RND :: NEXT I
150 GOTO 120
```

# READ

Format:
-------

READ Variablenliste

---

Beschreibung:
-------------

Der READ-Befehl ermöglicht die Zuweisung von Daten, die programm-intern mit Hilfe von DATA-Befehlen abgespeichert sind, an die Variablen der Variablenliste. Die Variablenliste kann String-Variable und/oder numerische Variable enthalten, die alle durch Kommata getrennt werden müssen.

Mit dem READ-Befehl werden die Datenwerte nach den DATA-Befehlen von links nach rechts in der Reihenfolge der Zeilennummern den Variablen zugewiesen. Dabei wird also immer der nächste Datenwert der nächsten Variablen zugeordnet. Wird der READ-Befehl mehrmals ausgeführt, können einer oder mehreren Variablen nacheinander verschiedene Datenwerte der Datenliste zugeordnet werden. Mit Hilfe des RESTORE-Befehls kann die Reihenfolge verändert werden.

Die Typen der Variablen der Variablenliste müssen mit den Typen der entsprechenden Datenelemente des DATA-Befehls übereinstimmen. Enthält eine Datenliste aufeinanderfolgende Kommata, so nimmt der Computer die Zuweisung eines Leerstrings (String ohne Zeichen) an. Da eine Zahl auch als String aufgefaßt werden kann, können Zahlen sowohl numerischen als auch String-Variablen zugeordnet werden. Bei dem Versuch, einer numerischen Variablen einen String zuzuweisen, erscheint die Fehlermeldung DATA ERROR (Datenfehler), und der Programmablauf wird unterbrochen.

Enthält die Variablenliste mehr Variable als Datenwerte in den DATA-Befehlen vorhanden sind, erscheint die Fehlermeldung DATA ERROR (Datenfehler) und der Programmablauf wird unterbrochen.

---

Beispielprogramm:
-----------------

Bei der Ausführung des READ-Befehls in Programmzeile 130 werden den String-Variablen W1$(I) und W2$(I) jeweils ein Datenelement zugeordnet, so daß nach 21 Durchläufen der Zählschleife (Zeilen 120 bis 140) alle 42 Datenelemente den Variablen zugeordnet sind. Vgl. auch Beispielprogramm zum DATA-Befehl!

Das Programm endet mit <FCTN-4> (CLEAR).

```
100 REM ***** LISTE DER UNTERPROGRAMME *****
110 DIM W1$(21),W2$(21)
120 FOR I=1 TO 21
130 READ W1$(I),W2$(I)
140 NEXT I
150 PRINT "LISTE DER UNTERPROGRAMMNAMEN":""
160 FOR I=1 TO 21
170 PRINT W1$(I);TAB(15);W2$(I)
180 NEXT I
190 DATA CHAR,,CHARPAT,LINK,CHARSET,LOAD,CLEAR,LOCATE,COINC,,COLOR
200 DATA MAGNIFY,,MOTION,DELSPRITE,,DISTANCE,PATTERN,,PEEK,ERR,POSITION,,
210 DATA GCHAR,SAY,,SCREEN,HCHAR,SOUND,,SPGET,INIT,SPRITE,,
220 DATA JOYST,VCHAR,,VERSION,KEY,
230 GOTO 230
```

# REC

Format:

REC(Dateinummer)

Beschreibung:

Die REC-Funktion gibt die Nummer des Datensatzes an, der als nächstes mit einem PRINT-, INPUT- oder LINPUT-Befehl in der Datei angesprochen wird, die unter der anzugebenden Dateinummer geöffnet worden ist. Die Numerierung der Datensätze einer Datei beginnt mit 0, so ist beispielsweise der Datensatz mit der Nummer 5 der 6. Datensatz in der Datei.

Beispiel:

In Programmzeile 100 wird die Datensatznummer des nächsten anstehenden Datensatzes der Datei, die unter der Nummer 5 geöffnet worden ist, ausgeben.

```
100 PRINT REC(5)
```

## REM

Format:

REM Zeichenkette

Beschreibung:

Der REM-Befehl erlaubt es, Anmerkungen in ein Programm einzufügen, um das Programm näher zu erläutern. Für die Zeichenkette kann jedes druckbare Zeichen verwendet werden. Die Zeichenkette hat keinerlei Auswirkungen auf den Programmablauf, sie benötigt lediglich Speicherplatz.

Der REM-Befehl sollte hauptsächlich dazu benutzt werden, ein Programm in Abschnitte zu unterteilen und den jeweils folgenden Abschnitt zu erläutern.

Beispiel:

Programmzeile 100 zeigt den Beginn eines Unterprogramms an.

```
100 REM  U N T E R P R O G R A M M
```

# RESEQUENCE

Format:

RESEQUENCE [Anfangszeile][,Zuwachszahl]
RES [Anfangszeile][,Zuwachszahl]

Beschreibung:

Der RES-Befehl ändert die Zeilennummern in einem Programm entsprechend der eventuell angegebenen Anfangszeile und Zuwachszahl. Fehlt die Anfangszeile, so erhält die erste Programmzeile die Nummer 100. Fehlt die Zuwachszahl, so wird sie automatisch zu 10 angenommen, d.h. alle der Anfangszeile folgenden Programmzeilen erhalten eine um jeweils 10 höhere Zeilennummer. Durch den RES-Befehl werden auch die Nummern der Bezugszeilen in den Befehlen BREAK, DISPLAY USING, GOSUB, GOTO, IF-THEN-ELSE, ON ERROR, ON-GOSUB, ON-GOTO, PRINT USING, RESTORE, RETURN und RUN geändert, so daß sie sich auf dieselben Zeilen wie vor der Ausführung des RES-Befehls beziehen.

Wird in einer Programmzeile eine Zeilennummer angegeben, die im vorliegenden Programm nicht existiert, so erhält sie die Zeilennummer 32767, ohne daß der Computer eine Fehlermeldung abgibt. Sind für Anfangszeile und Zuwachszahl Werte gewählt worden, nach denen einige neue Zeilennummern größer als 32767 werden, erscheint die Fehlermeldung BAD LINE NUMBER (falsche Zeilennummer), und der RES-Befehl wird nicht ausgeführt.

Beispiele:

| | |
|---|---|
| Die Programmzeilen werden in 10er-Schritten ab Zeile 100 numeriert. | >RES |
| Hier werden die Programmzeilen ab 1000 in 10er-Schritten numeriert. | >RES 1000 |
| Die Anfangszeile erhält die Nummer 1000, bei allen folgenden Programmzeilen erhöht sich die Zeilennummer um jeweils 15. | >RES 1000,15 |
| Bei fehlender Anfangszeile erhält die erste Programmzeile die Nummer 100, alle weiteren Zeilennummern sind um jeweils 15 größer. | >RES ,15 |

# RESTORE

Format:

RESTORE [Zeilennummer]
RESTORE [#Dateinummer [,REC Datensatznummer]]

Beschreibung:

Der RESTORE-Befehl kann entweder im Zusammenhang mit dem DATA-Befehl bei programminternen Dateien oder bei externen Dateien (auf Peripheriegeräten), die unter der entsprechenden Dateinummer geöffnet worden sind, benutzt werden. Grundsätzlich wird der Computer mit dem RESTORE-Befehl auf den nächsten zu verarbeitenden Datensatz hingewiesen.

Im Zusammenhang mit dem READ-Befehl weist der RESTORE-Befehl den Computer an, auf welchen DATA-Befehl er zurückgreifen soll. Fehlt die Zeilennummer, so ist dies automatisch der DATA-Befehl mit der niedrigsten Zeilennummer. Es wird immer auf das erste Datenelement nach dem DATA-Befehl zurückgegriffen.

Bei externen Dateien wird durch den RESTORE-Befehl der Datensatz angegeben, der durch den nächsten im Programm folgenden PRINT-, INPUT- oder LINPUT-Befehl in der unter der Dateinummer geöffneten Datei verarbeitet werden soll.

Dabei kann nach dem Schlüsselwort REC die entsprechende Datensatznummer angegeben werden. Fehlt sie, so wird sie automatisch zu 0 gesetzt. Die Anwendung des RESTORE-Befehls im Zusammenhang mit Dateien auf Cassetten ist nicht möglich.

Beispiele:

Programmzeile 100 bewirkt, daß das nächste Datenelement, auf das im Programm zurückgegriffen wird, dasjenige ist, das hinter dem DATA-Befehl mit der niedrigsten Zeilennummer steht.

```
100 RESTORE
```

In Programmzeile 120 hingegen wird auf das erste Datenelement der Zeile 200 zurückgegriffen.

```
120 RESTORE 200
```

Durch den Befehl in Programmzeile 140 wird die Datei #3 zurückgesetzt, d.h. es wird der Zugriff auf den ersten Datensatz der Datei #3 vorbereitet.

```
140 RESTORE #3
```

In Programmzeile 160 hingegen ist dies der Datensatz mit der Nummer NR-1 der Datei #4.

```
160 RESTORE #4,REC NR-1
```

# RETURN

Format:

```
RETURN
RETURN  Zeilennummer
RETURN  NEXT
```

Beschreibung:

Steht in einer Unterprogrammschleife, die mit dem GOSUB-Befehl oder dem ON-GOSUB-Befehl aufgerufen wird, der RETURN-Befehl, so wird der Programmablauf mit dem nächsten Befehl hinter dem Unterprogrammschleifen-Aufruf fortgesetzt. Der RETURN-Befehl schließt also die Unterprogrammschleife ab.

Im Zusammenhang mit dem ON ERROR-Befehl gibt es verschiedene Versionen des RETURN-Befehls: Ohne angefügte Zeilennummer oder NEXT wird der Programmablauf zurück zu der Zeile springen, in der der Fehler auftrat und diese Programmzeile erneut ausführen. Bei einer angegebenen Zeilennummer wird der Programmablauf mit der entsprechenden Programmzeile fortgesetzt. Bei dem Schlüsselwort NEXT wird der Programmablauf mit dem nächsten Befehl fortgesetzt, der dem den Fehler verursachenden Befehl folgt.

Beispielprogramme:

S. GOSUB-Befehl und ON ERROR-Befehl.

## RND

Format:

```
RND
```

Beschreibung:

Die RND-Funktion erzeugt die nächste Pseudo-Zufallszahl in der momentanen Folge der Pseudo-Zufallszahlen. Diese Zufallszahl ist größer oder gleich 0 und kleiner als 1. Die Zahlenfolge, die mit der RND-Funktion erzeugt wird, ist bei jedem Programmablauf gleich, wenn sich nicht im Programm ein RANDOMIZE-Befehl befindet.

Beispiele:

In Programmzeile 100 wird der Variablen WERT als Zufallszahl eine ganze Zahl zwischen 1 und 10 zugeordnet.

```
100 WERT=INT(RND*10)+1
```

Will man eine Zufallszahl ZUFALL zwischen A und B erzeugen, so kann man wie in Programmzeile 120 vorgehen.

```
120 ZUFALL=INT(RND*(B-A+1))+A
```

Beispielprogramm:

In diesem Programm "würfelt" der Computer 100 mal und zählt, wie oft er jede Augenzahl insgesamt "geworfen" hat (Test des Zufallszahlengenerators)

```
100 REM ***** TEST DES ZUFALLSZAHLEN-GENERATORS *****
110 OPTION BASE 1
120 DIM ANZAHL(6)
130 RANDOMIZE
140 CALL CLEAR
150 REM ***** WUERFELN UND ZAEHLEN *****
160 FOR WURF=1 TO 100
170 AUGEN=INT(RND*6)+1
180 ANZAHL(AUGEN)=ANZAHL(AUGEN)+1
190 NEXT WURF
200 REM ***** AUSGABE DER ERGEBNISSE *****
210 IMAGE "     #      ####"
220 PRINT "   ZAHL    ANZAHL"
230 PRINT "--------------------"
240 FOR AUGEN=1 TO 6
250 PRINT USING 210:AUGEN,ANZAHL(AUGEN)
260 NEXT AUGEN
```

## RPT$

Format:

RPT$(String-Ausdruck, numerischer Ausdruck)

Beschreibung:

Die RPT$-Funktion liefert einen String, der den String-Ausdruck entsprechend dem Wert des numerischen Ausdrucks wiederholt. Für die Berechnung des numerischen Ausdrucks gelten die üblichen Regeln. Enthält der durch Anwendung der RPT$-Funktion entstandene String mehr als 255 Zeichen, so werden die darüber hinausgehenden Zeichen nicht berücksichtigt.

Beispiele:

In Programmzeile 100 wird der Stringvariablen NEU$ der String "ALTALTALT" zugeordnet.

```
100 NEU$=RPT$("ALT",3)
```

Ist in Programmzeile 120 der Variablen ZAHL ein Wert zugeordnet, so dieser Wert als ganze Zahl mit maximal 7 Stellen ausgegeben.

```
120 PRINT USING RPT$("#",7):ZAHL
```

## RUN

Format:

```
RUN ["Gerätename.Programmname"]
RUN [Zeilennummer]
```

Beschreibung:

Der RUN-Befehl, der im EXTENDED BASIC im Gegensatz zum TI BASIC auch in einer Programmzeile auftreten kann, startet die Programmausführung entweder eines schon im Arbeitsspeicher des Computers befindlichen Programms (eventuell ab einer angegebenen Zeilennummer) oder eines unter "Gerätename.Programmname" abgespeicherten Programms (z.B. auf einer Diskette). Im letzteren Fall wird dazu das bezeichnete Programm zunächst in den Arbeitsspeicher geladen.

Vor der Programmausführung überprüft der Computer das zu startende Programm auf bestimmte Fehler, die er gegebenenfalls anzeigt. Auch wird allen numerischen Variablen der Wert 0 und allen String-Variablen der Leerstring (String ohne Zeichen) zugewiesen.

Optionen:

Bei der Angabe von "Gerätename.Programmname" (die Anführungszeichen müssen eingegeben werden) wird vor dem Start der Programmausführung das entsprechend bezeichnete Programm in den Arbeitsspeicher geladenn. Dabei werden alle zuvor im Speicher befindlichen Daten und Programme gelöscht.

Wird eine Zeilennummer angegeben, so wird das im Arbeitsspeicher befindliche Programm ab der angegebenen Zeile gestartet.

Beispiele:

Der Direktbefehl startet das im Arbeitsspeicher befindliche Programm.

```
>RUN
```

Programmzeile 100 bewirkt den Start der Programmausführung ab Zeile 200.

```
100 RUN 200
```

Programmzeile 120 bewirkt das Laden des Programms mit Namen BANANE, das sich auf der Diskette im Laufwerk 1 befindet, in den Arbeitsspeicher und die anschließende Ausführung des Programms.

```
120 RUN "DSK1.BANANE"
```

Beispielprogramm:

Sind auf einer Diskette verschiedene Programme abgespeichert, die hier einmal mit PROGR1, PROGR2 und PROGR3 bezeichnet sein sollen, so können diese Programme mit Hilfe eines weiteren Programms gestartet werden, das selbst unter dem Programmnamen LOAD auf der Diskette abgespeichert ist, und beim Einschalten direkt geladen und ausgeführt wird (vgl. Schnellstart). Man spricht in diesem Zusammenhang von "Menütechnik".

```
100 REM ***** MENUETECHNIK *****
110 CALL CLEAR
120 PRINT "     PROGRAMMUEBERSICHT"
130 PRINT "     ------------------"
140 PRINT "           1   PROGR1"
150 PRINT "           2   PROGR2"
160 PRINT "           3   PROGR3"
170 PRINT "           4   ENDE"
180 PRINT
190 INPUT "           W A H L ? ":WAHL
200 IF WAHL=1 THEN RUN "DSK1.PROGR1"
210 IF WAHL=2 THEN RUN "DSK1.PROGR2"
220 IF WAHL=3 THEN RUN "DSK1.PROGR3"
230 IF WAHL=4 THEN STOP
240 GOTO 120
```

# SAVE

Format:

```
SAVE Gerätename[.Programmname][,PROTECTED]
SAVE Gerätename.Programmname[,MERGE]
```

Beschreibung:

Mit dem SAVE-Befehl kann das Programm, das sich gerade im Arbeitsspeicher des Computers befindet, auf das mit Gerätename bezeichnete Gerät abgespeichert werden. Mit Hilfe des OLD- oder RUN-Befehls kann das Programm später wieder in den Arbeitsspeicher zurückgeholt werden. Der SAVE-Befehl löscht alle Stoppstellen in dem Programm, er bewirkt keine Löschung des Programms im Arbeitsspeicher.

Ist der Gerätename CS1, also ein Cassetten-Recorder, dann muß kein Programmname angegeben werden. Die Anweisungen zur Bedienung des Cassetten-Recorders erscheinen auf dem Bildschirm. Soll ein Programm auf Diskette abgespeichert werden, so muß zusätzlich zum Gerätename (z.B. DSK1) ein Programmname angegeben werden.

Optionen:

Mit dem Schlüsselwort PROTECTED wird sichergestellt, daß ein Programm nur zur Programmausführung in den Arbeitsspeicher gebracht werden kann. Dieses Programm kann jedoch weder aufgelistet noch editiert noch erneut abgespeichert werden (Programmschutz). Da der Programmschutz nicht aufhebbar ist, sollte man sicherheitshalber immer eine ungeschützte Kopie des Programms aufbewahren. Um ein Programm auch vor dem Kopieren zu schützen, ist die Anleitung zum DISK MANAGER Modul zu beachten.

Um ein Programm in ein anderes Programm einzubinden, muß das Programm mit dem Schlüsselwort MERGE abgespeichert werden (vgl. MERGE-Befehl). Das MERGE-Schlüsselwort ist nicht im Zusammenhang mit dem Abspeichern auf Cassetten möglich.

Beispiele:

Mit diesem Befehl wird ein im Arbeitsspeicher befindliches Programm auf Cassette abgespeichert.

```
>SAVE CS1
```

Zur Abspeicherung eines Programms auf Diskette ist der Gerätename (DSK1) und der Programmname (PROGR1) erforderlich, bei zusätzlichem Programmschutz das Schlüsselwort PROTECTED.

```
>SAVE DSK1.PROGR1
```

```
>SAVE DSK1.PROGR1,PROTECTED
```

Das Programm mit Namen UPROGR wird auf der Diskette im Laufwerk 1 abgespeichert. Eine spätere Einbindung dieses Programms in andere Programme ist möglich.

```
>SAVE DSK1.UPROGR,MERGE
```

## SAY-Unterprogramm

Format:

```
CALL SAY(Wortstring[,Wiedergabestring][,...])
```

Beschreibung:

Das SAY-Unterprogramm veranlaßt den Computer zur Ausgabe des Wortstrings oder des Wertes des Wiedergabestrings über den SPRACHSYNTHESIZER. Eine vollständige Beschreibung des SAY-Unterprogramms ist in der Bedienungsanleitung zum SPRACHSYNTHESIZER zu finden, der als Zusatzgerät erhältlich ist.

## SCREEN-Unterprogramm

Format:

```
CALL SCREEN(Farbcode)
```

Beschreibung:

Mit dem SCREEN-Unterprogramm kann die Bildschirmfarbe gemäß dem eingegebenen Farbcode verändert werden. Der Farbcode ist ein numerischer Ausdruck mit einem Wert zwischen 1 und 16. Die entsprechenden Farben sind der folgenden Tabelle zu entnehmen:

| FARBCODE | FARBE |
|---|---|
| 1 | transparent |
| 2 | schwarz |
| 3 | mittelgrün |
| 4 | hellgrün |
| 5 | dunkelblau |
| 6 | hellblau |
| 7 | dunkelrot |
| 8 | kornblumenblau |
| 9 | mittelrot |
| 10 | hellrot |
| 11 | dunkelgelb |
| 12 | hellgelb |
| 13 | dunkelgrün |
| 14 | magentarot |
| 15 | grau |
| 16 | weiß |

Beispielprogramm:

Das folgende Programm zeigt alle Bildschirmfarben mit Ausnahme von transparent und schwarz (Farbcode 1 und 2). Die Bildschirmfarbe wird jeweils in Programmzeile 140 verändert. Programmzeile 160 ist eine Verzögerungsschleife.

```
100 REM ***** BILDSCHIRMFARBEN *****
110 CALL CLEAR
120 FOR CODE=3 TO 16
130 READ FARBE$
140 CALL SCREEN(CODE)
150 DISPLAY AT(12,5):CODE;"  ";FARBE$
160 FOR VERZOEG=1 TO 500 :: NEXT VERZOEG
170 NEXT CODE
180 DATA mittelgruen,hellgruen,dunkelblau,hellblau,dunkelrot
190 DATA kornblumenblau,mittelrot,hellrot,dunkelgelb,hellgelb
200 DATA dunkelgruen,magentarot,grau,weiss
210 END
```

## SEG$

Format:

SEG$(String-Ausdruck, numerischer Ausdruck 1, numerischer Ausdruck 2)

Beschreibung:

numerischer Ausdruck 1 = Position
numerischer Ausdruck 2 = Länge

Mit der SEG$-Funktion erhält man ein Stringsegment (Substring) aus dem durch den String-Ausdruck bezeichneten String (Originalstring). Die Position (= numerischer Ausdruck 1) gibt die Stelle des Zeichens im Originalstring an, das gleichzeitig das erste Zeichen des Stringsegments darstellt. Der Wert des numerischen Ausdrucks 2 gibt die Länge des Stringsegments an. Für die Berechnung der numerischen Ausdrücke gelten die üblichen Regeln.

Ist der Wert des numerischen Ausdrucks 1 (Position) größer als die Länge des Originalstrings, so wird der Leerstring ausgegeben, ist der Wert des numerischen Ausdrucks 2 (Länge) größer als die restliche Länge des Originalstrings, so erhält man den Rest des Originalstrings ab der angegebenen Position. Ist der Wert von Position kleiner oder gleich Null und/oder der Wert von Länge kleiner Null, so erscheint die Fehlermeldung BAD VALUE (falscher Wert) und der Programmablauf wird unterbrochen.

Beispiele:

Aus dem vorgegebenen String "TEXAS INSTRUMENTS" wird in Programmzeile 120 der Variablen W1$ der String "TEXAS", in 140 der Variablen W2$ der String "INSTRUMENTS" und in 160 der String "TEST" zugeordnet.

```
100 FIRMA$="TEXAS INSTRUMENTS"
120 W1$=SEG$(FIRMA$,1,5)
140 W2$=SEG$(FIRMA$,7,11)
160 W3$=SEG$(FIRMA$,1,2)&SEG$
    (FIRMA$,9,2)
```

Beispielprogramm:

In dem Programm wird in einem eingegebenen Satz nach allen Leerzeichen gesucht, wodurch die einzelnen Worte getrennt werden können, um sie anschließend einzeln zeilenweise auszugeben.

```
100 REM ***** BEISPIEL ZUM SEG$- UND POS-BEFEHL *****
110 CALL CLEAR
120 PRINT "EINGABE EINES SATZES:":" <ENTER> = PROGRAMMENDE"
130 INPUT SATZ$ :: PRINT
140 IF SATZ$="" THEN 200
150 STELLE=POS(SATZ$," ",1)
160 IF STELLE=0 THEN PRINT SATZ$ :: PRINT :: GOTO 120
170 WORT$=SEG$(SATZ$,1,STELLE):: PRINT WORT$
180 SATZ$=SEG$(SATZ$,STELLE+1,LEN(SATZ$))
190 GOTO 150
200 END
```

## SGN

Format:

SGN(numerischer Ausdruck)

Beschreibung:

Die SGN-Funktion gibt das algebraische Vorzeichen des Wertes des numerischen Ausdrucks an. Ist der numerische Ausdruck positiv, so gibt die SGN-Funktion den Wert 1 an, bei negativem Ausdruck den Wert -1. Ist der numerische Ausdruck 0, so nimmt auch die SGN-Funktion diesen Wert an. Für die Berechnung des numerischen Ausdrucks gelten die üblichen Regeln.

Beispiele:

In Programmzeile 100 wird der Wert 1 in 120 der Wert -1 und in 140 der Wert 0 ausgegeben.

```
100 PRINT SGN(5.8)
120 PRINT SGN(-5)
140 PRINT SGN(0)
```

In Programmzeile 160 springt das Programm nach Zeile 200, wenn X negativ ist, nach Zeile 300, wenn X=0 ist, und nach Zeile 400, wenn X positiv ist.

```
160 ON SGN(X)+2 GOTO 200,300,400
```

# SIN

Format:

SIN(numerischer Ausdruck)

Beschreibung:

Die SIN-Funktion berechnet den Sinus des numerischen Ausdrucks. Dabei entspricht der Wert des numerischen Ausdrucks einem Winkel im Bogenmaß. Der numerische Ausdruck wird zunächst nach den üblichen Regeln berechnet.

Stellt der Wert des numerischen Ausdrucks einen Winkel in der Einheit Grad dar, und es soll davon der Sinus berechnet werden, so muß dieser Wert zuerst mit

| | |
|---|---|
| PI/180 | im EXTENDED BASIC und |
| ATN(1)/45 | im TI BASIC |

multipliziert werden, um den entsprechenden Winkel im Bogenmaß zu erhalten

Achtung:

Ist der Absolutbetrag des numerischen Ausdrucks größer als 1.5707963266375E10, so erscheint die Fehlermeldung * BAD ARGUMENT und der Programmablauf wird unterbrochen.

Beispielprogramm:

Das Programm berechnet den Sinus verschiedener Winkel, wobei ALPHA und BETA im Bogenmaß und GAMMA im Gradmaß gegeben ist.

```
100 REM ***** BEISPIELE ZUR SIN-FUNKTION *****
110 ALPHA=.7635 !BOGENMASS
120 BETA=-8.348 !BOGENMASS
130 GAMMA=65.39 !GRADMASS
140 PRINT ALPHA,SIN(ALPHA)
150 PRINT BETA,SIN(BETA)
160 PRINT GAMMA;" GRAD",SIN(GAMMA*PI/180)
```

# SIZE

Format:

SIZE

Beschreibung:

Der SIZE-Befehl zeigt den freien Platz im Arbeitsspeicher des Computers (in Bytes) an. Ein Byte ist der Speicherplatz, den ein Zeichen oder ein EXTENDED BASIC-Schlüsselwort beansprucht.

Ist keine Speichererweiterung als peripheres Gerät hinzugeschaltet, so ist der freie Speicherplatz derjenige, den man erhält, wenn man den Speicherplatz für das Programm, die Bildschirmausgabe, die Zeichendefinition, die Farbtabelle, die String-Werte und ähnliches von dem ursprünglich vorhandenen Speicherplatz (16384 Bytes) abzieht.

Ist eine Speichererweiterung angeschlossen, so wird der noch verfügbare Speicherplatz im Stackregister und im Programmspeicher getrennt angezeigt. Im Stackregister befinden sich die Stringwerte, die Informationen über Variable und ähnliches, während im Programmspeicher das Programm und die im Programm erscheinenden numerischen Variablenwerte abgelegt werden.

# SOUND-Unterprogramm

Format:

```
CALL SOUND(Dauer, Frequenz1, Lautstärke1[,...][,Frequenz4, Lautstärke4])
```

Beschreibung:

Mit dem SOUND-Unterprogramm können vom Computer Töne erzeugt werden, deren Dauer, Frequenz (Tonhöhe) und Lautstärke angegeben werden müssen. Dauer, Frequenz und Lautstärke sind numerische Ausdrücke, deren jeweils gültigen Bereiche für ihre ganzzahligen Werte sind:

| WERT | BEREICH |
|---|---|
| Dauer | 1 ... 4250 und -1 ... -4250 |
| Frequenz | Ton: 110 ... 44733 |
| | Geräusch: -1 ... -8 |
| Lautstärke | 0 (größte Lautstärke) ... 30 (kleinste Lautstärke) |

Dauer: Die Dauer wird in Millisekunden (tausendstel Sekunden) angegeben. Die Länge eines Tones reicht also von 0.001 bis 4.25 Sekunden. Dabei kann die tatsächliche Dauer um etwa 1/60 Sekunde variieren. Die angegebene Dauer gilt für j e d e n Ton, der in e i n e m SOUND-Befehl erzeugt wird.

Während ein Ton abgespielt wird, setzt der Computer die Programmausführung schon fort. Wird dabei erneut das SOUND-Unterprogramm aufgerufen, wartet der Computer den Abschluß der vorangehenden Töne ab. Bei Angabe einer negativen Dauer werden die vorangehenden Töne allerdings gestoppt und sofort mit den neuen begonnen.

Frequenz: Mit der angegebenen Frequenz wird entweder bei positiven Werten ein Ton oder bei negativen Werten ein Geräusch erzeugt. Bei den Tönen werden die Frequenzen in Hertz (Schwingungen pro Sekunde) angegeben, wobei der höchste mögliche Wert weit über der Grenze des menschlichen Hörbereichs liegt. Die Frequenzen für einige übliche Musiktöne sind dem Anhang zu entnehmen, sie können jedoch je nach Wert um bis zu 10 % schwanken.

Die Eigenschaften der Geräusche sind der folgenden Tabelle zu entnehmen:

| FREQUENZWERT | EIGENSCHAFT |
|---|---|
| -1 | periodisches Rauschen Typ 1 |
| -2 | periodisches Rauschen Typ 2 |
| -3 | periodisches Rauschen Typ 3 |
| -4 | periodisches Rauschen mit Variation in der Frequenz des 3. angegebenen Tones |
| -5 | weißes Rauschen Typ 1 |
| -6 | weißes Rauschen Typ 2 |
| -7 | weißes Rauschen Typ 3 |
| -8 | weißes Rauschen mit Variation in der Frequenz des 3. angegebenen Tones |

Es können bis zu drei Töne und ein Geräusch gleichzeitig mit einem SOUND-Befehl erzeugt werden.

Beispielprogramme:

S. nächste Seite

## SOUND-Unterprogramm

Mit dem ersten Programm kann jeweils ein Ton beliebiger Dauer, Tonhöhe und Lautstärke erzeugt werden (in Programmzeile 150).

Das Programm endet mit <FCTN-4>.

```
100 REM ***** TONTEST *****
110 CALL CLEAR
120 INPUT "DAUER:        ":DAUER
130 INPUT "FREQUENZ:     ":FREQUENZ
140 INPUT "LAUTSTAERKE: ":LAUTSTAERKE
150 CALL SOUND(DAUER,FREQUENZ,LAUTSTAERKE)
160 GOTO 120
```

In dem zweiten Programm werden die dem mittleren C folgenden Töne der Oktave erzeugt (Programmzeile 160) und deren Frequenz gleichzeitig auf dem Bildschirm angezeigt (Zeile 170). Die Tonfolge wird mit dem C-Dur-Akkord in Programmzeile 190 abgeschlossen.

Das Programm endet mit <FCTN-4>

```
100 REM ***** TONLEITER *****
110 CALL CLEAR
120 IMAGE FREQUENZ = ####
130 X=2^(1/12)
140 FOR A=0 TO 12
150 TON=262*X^A
160 CALL SOUND(1000,TON,0)
170 DISPLAY AT(12,7):USING 120:TON
180 NEXT A
190 CALL SOUND(2000,196,0,262,0,330,0)
200 CALL CLEAR
210 GOTO 140
```

Im letzten Programm zum SOUND-Unterprogramm werden alle 8 möglichen Geräusche der Reihe nach erzeugt (Programmzeile 140), wobei ihre Dauer von 0.5 bis 2 Sekunden gesteigert wird:

```
100 REM ***** GERAEUSCHE *****
110 CALL CLEAR
120 FOR DAUER=500 TO 2000 STEP 500
130 FOR GERAEUSCH=-1 TO -8 STEP -1
140 CALL SOUND(DAUER,GERAEUSCH,0)
150 DISPLAY AT(12,7):"GERAEUSCH = ";GERAEUSCH
160 NEXT GERAEUSCH
170 NEXT DAUER
```

## SPGET-Unterprogramm

Format:

```
CALL SPGET(Wortstring, Wiedergabestring)
```

Beschreibung:

Das SPGET-Unterprogramm ordnet dem Wiedergabestring das Sprachmuster zu, das dem Wortstring entspricht. Eine vollständige Beschreibung des SPGET-Unterprogramms ist in den Bedienungsanleitungen zum SPRACHSYNTHESIZER und SPRACHAUSGABE COMMAND Modul zu finden, die als Zusatzgeräte erhältlich sind.

# SPRITE-Unterprogramm

Format:

```
CALL SPRITE(#Sprite-Nummer, Zeichencode, Farbcode,
            Punktzeile, Punktspalte
            [,Zeilengeschwindigkeit, Spaltengeschwindigkeit][,...])
```

Beschreibung:

Das SPRITE-Unterprogramm dient zur Erzeugung von Sprites (Kobolden). Dabei handelt es sich um Grafiksymbole, die durch ihren Zeichencode bestimmt sind. Sie können damit Zeichen des Standardzeichensatzes (s. Anhang) oder mit Hilfe des CHAR-Unterprogramms selbstdefinierte Zeichen sein. Sie besitzen eine Farbe, die durch den Farbcode bestimmt wird, und werden auf dem Bildschirm an die durch Punktzeile und Punktspalte definierte Stelle plaziert. Sie können mit unterschiedlichen Geschwindigkeiten in alle Richtungen in Bewegung versetzt werden. Sprites ändern ihren Zustand nur mit Hilfe verschiedener Unterprogramme, am Programmende oder an einer Stoppstelle (breakpoint) verschwinden sie ganz.

Sprite-Nummer ist ein numerischer Ausdruck, dessen ganzzahliger Wert zwischen 1 und 28 einschließlich liegen muß. Wird eine Sprite-Nummer gewählt, für die im Programm schon ein Sprite existiert, dann wird der alte Sprite gelöscht und durch den neuen ersetzt. Werden für den neuen Sprite keine neuen Geschwindigkeiten in Zeilen- und Spaltenrichtung angegeben, übernimmt er die für den gelöschten Sprite angegebenen Geschwindigkeitswerte. Sprites bewegen sich über auf dem Bildschirm bereits vorhandene Zeichen. Treffen zwei oder mehr Sprites aufeinander, überdeckt der Sprite mit der kleinsten Sprite-Nummer die anderen Sprites. Befinden sich fünf oder mehr Sprites in derselben Bildschirmzeile, dann verschwinden die Sprites mit den höchsten Sprite-Nummern.

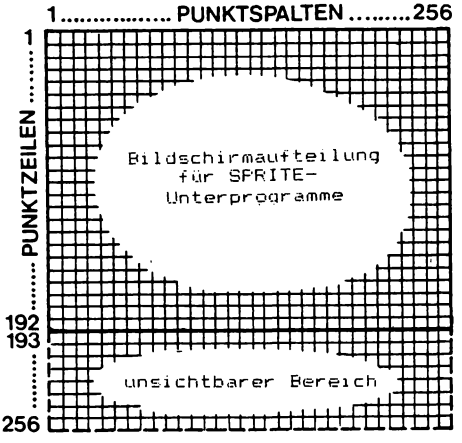
Zeichencode ist ein numerischer Ausdruck, dessen ganzzahliger Wert zwischen 32 und 143 einschließlich liegen muß. Er bestimmt das Aussehen (Muster) des Sprites (vgl. CHAR-Unterprogramm). Mit Hilfe des PATTERN-Unterprogramms kann dieses Aussehen verändert werden (vgl. PATTERN-Unterprogramm). Sprites, die aus einem Zeichen gebildet werden, können mit Hilfe des MAGNIFY-Unterprogramms auf 4 Schreibstellen vergrößert werden, belegen dann aber immer noch nur einen Zeichencode. Sprites, die aus vier Zeichen bestehen - angegeben wird dann nur der kleinste der vier aufeinanderfolgenden Zeichencodes -, können entsprechend auf 16 Schreibstellen vergrößert werden (vgl. MAGNIFY-Unterprogramm).

Farbcode ist ein numerischer Ausdruck, dessen ganzzahliger Wert zwischen 1 und 16 einschließlich liegen darf. Er bestimmt die Vordergrundfarbe der Sprites, deren Hintergrundfarbe immer transparent (Farbcode 1) ist (vgl. COLOR-Unterprogramm). Allerdings sind Sprites mit Farbcode 1 (transparent) nicht sichtbar. Die Farben sind der nebenstehenden Farbcode-Tabelle zu entnehmen.

| FARBCODE | FARBE |
|---|---|
| 1 | transparent |
| 2 | schwarz |
| 3 | mittelgrün |
| 4 | hellgrun |
| 5 | dunkelblau |
| 6 | hellblau |
| 7 | dunkelrot |
| 8 | kornblumenblau |
| 9 | mittelrot |
| 10 | hellrot |
| 11 | dunkelgelb |
| 12 | hellgelb |
| 13 | dunkelgrün |
| 14 | magentarot |
| 15 | grau |
| 16 | weiß |

## SPRITE-Unterprogramm

Punktzeile und Punktspalte sind numerische Ausdrücke, deren ganzzahlige Werte zwischen 1 und 256 liegen müssen, und die die Bildschirmposition der Sprites bestimmen. Zur Festlegung dieser Bildschirmpostion beachten Sie bitte die nebenstehende Darstellung.




Zeilengeschwindigkeit und Spaltengeschwindigkeit setzen einen Sprite in Bewegung und bestimmen seine Richtung und seine Geschwindigkeit. Sie sind numerische Ausdrücke, deren Werte zwischen -128 und +127 einschließlich liegen müssen. Informationen über die Wirkungsweise der Zeilen- und Spaltengeschwindigkeit sind unter der Beschreibung zum MOTION-Unterprogramm zu finden.

Folgende Unterprogramme beeinflussen den Zustand der Sprites oder informieren über sie:

| WIRKUNG | UNTER-PROGRAMM | KOMMENTAR | BEZUG |
|---|---|---|---|
| Muster | CHAR | bestimmt das Aussehen von Sprites | |
| | PATTERN | ändert das Muster von Sprites | Zeichencode |
| | MAGNIFY | ändert die Größe von Sprites | |
| Farbe | COLOR | ändert die Farbe von Sprites | Farbcode |
| Position | LOCATE | bestimmt neue Sprite-Positionen | |
| Bewegung | MOTION | ändert die Sprite-Bewegung | Geschwindigkeit |
| Auskunft | POSITION | informiert über Sprite-Position | |
| | COINC | informiert über Zusammentreffen von mehreren Sprites | Punktzeile/ Punktspalte |
| | DISTANCE | informiert über Sprite-Abstände | |
| Existenz | (SPRITE) | (erzeugt Sprites) | |
| | DELSPRITE | löscht Sprites | |

Zur Beachtung bei der Verwendung von Sprites in einem Programm:

1. Verwenden Sie keine Leerzeichen (Zeichencode 32 oder 143)!
2. Sprite-Farbe muß von Bildschirmfarbe abweichen.
3. Sprite-Farbe darf nicht transparent sein (Farbcode 1).
4. Nur bis Punktzeile 192 sind Sprites auf dem Bildschirm sichtbar.

## SPRITE-Unterprogramm

Beispiel:

Programmzeile 100 erzeugt ein weißes Ausrufezeichen als Sprite, das sich langsam nach rechts unten bewegt.

```
100 CALL SPRITE(#1,33,16,12,48,1,1)
```

Beispielprogramme:

Im ersten Programm werden in Programmzeile 120 in der Bildschirmmitte nacheinander 28 Sprites (Sternchen) erzeugt, deren Farbe durch INT(A/3)+3 bestimmt wird. Die Geschwindigkeiten in Zeilen- und Spaltenrichtung werden zufällig ermittelt.

```
100 REM ***** STERNCHEN *****
110 CALL CLEAR
120 FOR A=1 TO 28 :: CALL SPRITE(#A,42,INT(A/3)+3,92,124,A*INT(RND*4.5)-2.25+A/2
*SGN(RND-.5),A*INT(RND*4.5)-2.25+A/2*SGN(RND-.5)):: NEXT A
130 FOR VERZOEG=1 TO 10000 :: NEXT VERZOEG
140 END
```

Das folgende Beispielprogramm verwendet alle im Zusammenhang mit Sprites gebräuchlichen Unterprogramme mit Ausnahme von COLOR und DISTANCE. Wegen der besonderen Bedeutung, die Sprites im EXTENDED BASIC besitzen, werden die wesentlichen Programmzeilen zunächst kommentiert. Das vollständige Programm finden Sie auf der nächsten Seite.

Das Programm erzeugt zwei vier Zeichen umfassende, auf 16 Schreibstellen vergrößerte Sprites in Form von Männchen, die sich auf einem Boden fortbewegen. Durch ein Hindernis geht das eine Männchen hindurch, das andere springt hindurch und geht nach jedem Sprung etwas schneller, bis es das andere Männchen einholt. Dann "schrumpfen" beide Männchen auf 4 Schreibstellen und gehen weiter. Treffen sie ein zweites Mal aufeinander, verschwindet das schnellere, und das andere setzt seine Bewegung fort bis das Programmende erreicht wird.

Kurzkommentar zu wesentlichen Programmzeilen:

```
120      Zähler für Treffen wird gleich Null gesetzt.
130-160  Definition der Sprite-Muster
200-210  Aufruf und Vergrößerung der Sprites
220      Startgeschwindigkeit des schneller werdenden Sprites
240      Sprites werden in Bewegung gesetzt.
250      Geh-Bewegung wird erzeugt.
270      Überprüfung hinsichtlich eines Zusammentreffens
290-300  Überprüfung des Erreichens des Hindernisses durch Sprite #1
320-430  Verfahren bei Treffen der Sprites
340      Sprites werden angehalten.
370      Überprüfung auf 1. oder 2. Zusammentreffen
390      Feststellung der Sprite-Positionen
400      Sprites werden verkleinert.
410      Sprites werden auf Boden gestellt, das schnellere erhält Vorgabe.
420      Bewegung beginnt erneut.
440-510  Der schnellere Sprite springt durch das Hindernis.
460      Sprite wird angehalten.
470      Seine Position wird festgestellt.
480      Neue Position hinter dem Hindernis
490-500  Geschwindigeit wird erhöht und Sprite in Bewegung gesetzt.
520-660  Verfahren bei 2. Zusammentreffen
550      Der langsamere Sprite wird in Bewegung gesetzt.
560      Der schnellere Sprite wird gelöscht.
570-620  Der langsamere Sprite geht noch 20 Schritte.
```

## SPRITE-Unterprogramm

```
100 REM ***** NACHLAUFEN *****
110 CALL CLEAR
120 ZAEHLER=0
130 M1$="0103030103030303030303030303038080C0C080C0C0C0C0C0C0C0C0C0C0C0E0"
140 M2$="0103030103070F1B1B03030306080C0C0E80C0C080C0E0F0D8CCC0C0C060303038"
150 CALL CHAR(96,M1$)
160 CALL CHAR(100,M2$)
170 CALL SCREEN(14):: CALL COLOR(13,16,16):: CALL COLOR(14,13,13)
180 FOR ZEILE=19 TO 24 :: CALL HCHAR(ZEILE,1,136,32):: NEXT ZEILE
190 FOR SPALTE=22 TO 24 :: CALL VCHAR(14,SPALTE,128,6):: NEXT SPALTE
200 CALL SPRITE(#1,96,5,113,129,#2,96,7,113,9)
210 CALL MAGNIFY(4)
220 GS=4
230 FORM=2
240 CALL MOTION(#1,0,GS,#2,0,3)
250 CALL PATTERN(#1,98+FORM,#2,98-FORM)
260 FORM=-FORM
270 CALL COINC(ALL,TREFFEN)
280 IF TREFFEN<>0 THEN 320
290 CALL POSITION(#1,PZ1,PS1)
300 IF PS1>136 AND PS1<192 THEN 440
310 GOTO 250
320 REM ***** TREFFEN *****
330 CALL SOUND(500,550,3,770,6,990,2)
340 CALL MOTION(#1,0,0,#2,0,0)
350 GOSUB 670
360 CALL PATTERN(#1,96,#2,96)
370 IF ZAEHLER>0 THEN 520
380 ZAEHLER=ZAEHLER+1
390 CALL POSITION(#1,PZ1,PS1,#2,PZ2,PS2)
400 CALL MAGNIFY(3)
410 CALL LOCATE(#1,PZ1+16,PS1+32,#2,PZ2+16,PS2)
420 CALL MOTION(#1,0,GS,#2,0,4)
430 GOTO 290
440 REM ***** #1 BEI HINDERNIS *****
450 CALL SOUND(300,-3,3)
460 CALL MOTION(#1,0,0)
470 CALL POSITION(#1,PZ1,PS1)
480 CALL LOCATE(#1,PZ1,193)
490 GS=GS+1
500 CALL MOTION(#1,0,GS)
510 GOTO 250
520 REM ***** 2. TREFFEN *****
530 CALL SOUND(500,220,4,440,2,660,4)
540 GOSUB 670
550 CALL MOTION(#2,0,4)
560 CALL DELSPRITE(#1)
570 FOR SCHRITTE=1 TO 20
580 CALL PATTERN(#2,100)
590 FOR VERZOEG=1 TO 20 :: NEXT VERZOEG
600 CALL PATTERN(#2,96)
610 FOR VERZOEG=1 TO 20 :: NEXT VERZOEG
620 NEXT SCHRITTE
630 CALL DELSPRITE(#2)
640 GOSUB 670
650 CALL CLEAR
660 END
670 FOR VERZOEG=1 TO 500 :: NEXT VERZOEG :: RETURN
```

## SQR

Format:

SQR(numerischer Ausdruck)

Beschreibung:

Die SQR-Funktion berechnet die positive Quadratwurzel des Wertes des numerischen Ausdrucks. Fur die Berechnung des numerischen Ausdrucks werden die üblichen Regeln angewendet.Ist der Wert des numerischen Ausdrucks negativ, so erscheint die Fehlermeldung * BAD ARGUMENT,und der Programmablauf wird unterbrochen. SQR(X) entspricht X^(1/2).

Beispiele:

| | |
|---|---|
| Programmzeile 100 bewirkt die Ausgabe der Zahl 2. | `100 PRINT SQR(4)` |
| In Programmzeile 120 wird der Variablen X der Wert 506.9516742 zugewiesen. | `120 X=SQR(2.57E5)` |
| Die Variable Z in Programmzeile 140 erhält den Wert der positiven Quadratwurzel aus der Summe der Quadrate von X und Y. | `140 Z=SQR(X^2+Y^2)` |

## STOP

Format:

```
STOP
```

Beschreibung:

Der STOP-Befehl beendet die Ausführung eines Programms. Er kann wechselweise mit dem END-Befehl benutzt werden. Der STOP-Befehl darf an beliebiger Stelle in einem Programm stehen, jedoch nicht nach einem Unterprogramm. Auch mehrere STOP-Befehle in einem Programm sind möglich.

## STR$

Format:

STR$(numerischer Ausdruck)

Beschreibung:

Die STR$-Funktion wandelt den Zahlenwert des numerischen Ausdrucks in einen String um. Der numerische Ausdruck wird zunächst nach den üblichen Regeln berechnet. Die STR$-Funktion ist die Umkehrung der VAL-Funktion.

Beispiele:

In Programmzeile 100 wird der Stringvariablen ZAHL$ der String "37.8" zugeordnet.

```
100 ZAHL$=STR$(37.8)
```

In Programmzeile 120 wird der Wert der numerischen Variablen A in einen String umgewandelt und der Stringvariablen A$ zugewiesen.

```
120 A$=STR$(A)
```

In Programmzeile 140 wird der dreifache Wert der numerischen Variablen C als String ausgegeben, mit C=-8 ist dies beispielsweise der String "-24".

```
140 PRINT STR$(3*C)
```

## SUB

Format:

```
SUB Unterprogrammname [(Variablenliste)]
```

Beschreibung:

Der SUB-Befehl ist der erste Befehl in einem selbstdefinierten Unterprogramm. Ein solches Unterprogramm wird benutzt, wenn man eine Gruppe von Programmzeilen vom Hauptprogramm abtrennen will, um sie beispielsweise in einem Programm mehrmals ausführen zu lassen. Damit wird dann Speicherplatz im Computer eingespart. Auch können Unterprogramme im Zusammenhang mit dem MERGE-Befehl in verschiedene Programme eingebettet werden, ohne daß es Komplikationen mit den Variablenbezeichnungen gibt. Der SUB-Befehl darf nicht in einem IF-THEN-ELSE-Befehl stehen.

Unterprogramme werden mit CALL Unterprogrammname [(Variablenliste)] aufgerufen. Sie enden mit dem SUBEND-Befehl. Während des Programmablaufs werden sie mit dem SUBEXIT-Befehl verlassen, und es wird dann der nächste Befehl nach dem Unterprogrammaufruf im Hauptprogramm fortgesetzt.

Unterprogramme müssen an das Ende des Hauptprogramms angefügt werden. Die Programmstruktur mit Unterprogrammen sieht daher folgendermaßen aus:

```
BEGINN des Hauptprogramms
 :
 :
Unterprogrammaufrufe über CALL Unterprogrammname
 :
 :
ENDE des Hauptprogramms
BEGINN des 1. Unterprogramms mit SUB Unterprogrammname
 :
 :
(ev. Aufruf anderer Unterprogramme)
 :
 :
ENDE des 1. Unterprogramms mit SUBEND
BEGINN des 2. Unterprogramms
 :
 :
ENDE des 2. Unterprogramms
 :
 :
Weitere Unterprogramme
 :
 :
PHYSIKALISCHES ENDE des Programms
```

Nach dem Ende eines Unterprogramms dürfen außer dem END-Befehl oder Kommentare mit dem REM-Befehl keine anderen Befehle stehen.

Optionen:

Alle im Unterprogramm benutzten Variablen mit Ausnahme von denen in der Variablenliste gelten lokal für das jeweilige Unterprogramm. Das bedeutet, daß dieselben Variablennamen im Hauptprogramm und im Unterprogramm benutzt werden können, ohne daß sich ihre Werte gegenseitig beeinflussen.

## SUB

Sollen Daten zwischen Hauptprogramm und Unterprogramm übergeben werden, so ist in dem SUB-Befehl eine Variablenliste mit derselben Anzahl und demselben Typ (numerisch oder String) von Variablen erforderlich. Die Variablen müssen auch in derselben Reihenfolge wie im CALL-Befehl erscheinen, sie brauchen aber nicht denselben Variablennamen zu haben. Wenn Variable, deren Werte an ein Unterprogramm übergeben wurden, dort ihre Werte ändern, dann werden ihre Werte auch im Hauptprogramm entsprechend geändert. Wenn ein Variablenwert des Hauptprogramms im Unterprogramm zwar verwendet, aber im Hauptprogramm nicht verändert werden soll, so muß diese Variable in der Variablenliste in Klammern stehen.

Soll ein ganzes Datenfeld an ein Unterprogramm übergeben werden, dann geschieht das in der Form

SUB Unterprogrammname(Feldname([,...])),

d.h., nach dem Feldnamen folgen die beiden Klammern () und für jede Dimension größer als 1 ein Komma. Innerhalb eines Unterprogramms ist ein zusätzlicher DIM-Befehl nicht erforderlich, da der entsprechende DIM-Befehl des Hauptprogramms für das ganze Programm gilt.

Wird ein Unterprogramm mehr als einmal aufgerufen, so behalten seine Variablen ihre Werte von einem Aufruf zum anderen bei.

---

Beispiele:

| | |
|---|---|
| Programmzeile 100 kennzeichnet den Beginn eines Unterprogramms mit dem Namen MENUE, alle Programmzeilen des Hauptprogramms müssen dann kleinere Zeilennummern haben. In Programmzeile 120 wird zusätzlich ein numerischer Wert an das Unterprogramm übergeben oder vom Hauptprogramm übernommen. | 100 SUB MENUE<br><br>120 SUB MENUE(WAHL) |
| Programmzeile 140 könnte der Beginn eines Unterprogramms zur Matrizenaddition sein, an das die beiden Matrizen M1 und M2 (zweidimensionale Datenfelder) übergeben werden, und das die Summenmatrix SUM an das Hauptprogramm zurückgibt. | 140 SUB MATADD(M1(,),M2(,),SUM(,)) |

---

Beispielprogramm:

In diesem Programm, das zur Berechnung der Hypothenuse und des Flächeninhalts eines rechtwinkligen Dreiecks dient, werden zwei selbstdefinierte Unterprogramme in den Programmzeilen 140 und 210 aufgerufen. Die Variablenwerte von X und Y werden dabei an das jeweilige Unterprogramm übergeben. Beachten Sie, daß innerhalb der Unterprogramme mit anderen Variablen weitergerechnet wird.

Programm s. nächste Seite!

## SUB

```
100 REM ***** RECHTWINKLIGES DREIECK *****
110 CALL CLEAR
120 INPUT "1. KATHETE = ":X
130 INPUT "2. KATHETE = ":Y
140 CALL HYPOTHENUSE(X,Y,Z)
150 CALL CLEAR
160 PRINT "DAS DREIECK MIT DEN KATHETEN":""
170 PRINT "      A = ";X
180 PRINT "      B = ";Y:""
190 PRINT "HAT DIE HYPOTHENUSE":""
200 PRINT "      C = ";Z:""
210 CALL FLAECHE(X,Y,Z)
220 PRINT "UND DEN FLAECHENINHALT":""
230 PRINT "      F = ";Z
240 STOP
1000 SUB HYPOTHENUSE(A,B,C)
1010 C=SQR(A^2+B^2)
1020 SUBEND
2000 SUB FLAECHE(G,H,F)
2010 F=G*H/2
2020 SUBEND
```

## SUBEND

Format:

SUBEND

Beschreibung:

Der SUBEND-Befehl kennzeichnet das Ende eines Unterprogramms. Im Programmablauf wird nach dem SUBEND-Befehl derjenige Befehl ausgeführt, der dem Unterprogramm-Aufruf folgt. Der SUBEND-Befehl muß der letzte Befehl (mit der höchsten Zeilennummer) im Unterprogramm sein.

Der SUBEND-Befehl darf nicht in einem IF-THEN-ELSE-Befehl auftreten. Die einzigen Befehle, die unmittelbar einem SUBEND-Befehl folgen dürfen, sind der REM- oder END-Befehl oder der SUB-Befehl des nächsten Unterprogramms.

Beispielprogramm:

S. SUB-Befehl!

## SUBEXIT

Format:

SUBEXIT

Beschreibung:

Der SUBEXIT-Befehl erlaubt das vorzeitige Verlassen eines Unterprogramms, bevor also das Ende des Unterprogramms, das durch den SUBEND-Befehl gekennzeichnet ist, erreicht worden ist. Im Programmablauf wird dann der nächste Befehl nach dem Unterprogramm-Aufruf ausgeführt. Der SUBEXIT-Befehl muß nicht in einem Unterprogramm erscheinen.

# TAB

FORMAT:

TAB(numerischer Ausdruck)

Beschreibung:

Die TAB-Funktion gibt die Anfangsposition der nächsten Datenausgabe in einem PRINT-, PRINT USING-, DISPLAY- oder DISPLAY USING-Befehl mit Hilfe des Wertes des numerischen Ausdrucks an. Ist der Wert des numerischen Ausdrucks größer als die Länge des Datensatzes (der Zeile) des Gerätes, auf dem die Ausgabe erfolgen soll (z.B. 28 für den Bildschirm oder 32 für den Thermodrucker), dann wird der Wert solange um die Länge reduziert, bis er zwischen 1 und der zulässigen Ausgabelänge liegt.

Ist in der laufenden Ausgabe die Zahl der ausgegebenen Zeichen größer oder gleich dem Wert des numerischen Ausdrucks, erfolgt die nächste Datenausgabe an der Stelle des nächsten Datensatzes, die dem Wert des numerischen Ausdrucks entspricht (beim Drucker in der nächsten Zeile).

Die TAB-Funktion wird wie ein Ausgabezeichen behandelt. Das bedeutet, daß vor und/oder nach ihm ein Ausgabeseparator (Doppelpunkt, Semikolon oder Komma) erscheinen muß. Normalerweise werden dazu Semikolons benutzt.

Beispiele:

In Programmzeile 100 wird die Zahl 17 an der 12. Stelle der Zeile ausgegeben.

```
100 PRINT TAB(10);17
```

In Programmzeile 120 wird in der 5. Bildschirmzeile am Anfang die Zahl 1 ab der 5. Stelle der String "NAME" und ab der 15. Stelle "ADRESSE" angezeigt.

```
120 DISPLAY AT(5,1):1;TAB(5);"NAME"
    TAB(15);"ADRESSE"
```

Der String "XYZ" in Programmzeile 140 wird erst in der nächsten Zeile ausgegeben, da davor der String "ABCDEFG HIK" steht, der länger als 5 ist.

```
140 PRINT "ABCDEFGHIK";TAB(5);"XYZ"
```

Beispielprogramm:

Das Beispiel zeigt eine kleine Anwendung der TAB-Funktion, das zu einer formatierten Ausgabe von Zahlen führt.

```
100 REM ***** BEISPIEL FUER TAB-BEFEHL *****
110 FOR I=1 TO 10
120 PRINT TAB(I);I;TAB(28-I);10-I
130 NEXT I
```

# TAN

Format:

TAN(numerischer Ausdruck)

Beschreibung:

Die TAN-Funktion berechnet den Tangens des numerischen Ausdrucks. Dabei entspricht der Wert des numerischen Ausdrucks einem Winkel im Bogenmaß. Der numerische Ausdruck wird zunächst nach den üblichen Regeln berechnet.

Stellt der Wert des numerischen Ausdrucks einen Winkel in der Einheit Grad dar, und es soll davon der Tangens berechnet werden, so muß dieser Wert zuerst mit

| | |
|---|---|
| PI/180 | im EXTENDED BASIC und |
| ATN(1)/45 | im TI BASIC |

multipliziert werden, um den entsprechenden Winkel im Bogenmaß zu erhalten

Achtung:

Ist der Absolutbetrag des numerischen Ausdrucks größer als 1.5707963266375E10, so erscheint die Fehlermeldung * BAD ARGUMENT, und der Programmablauf wird unterbrochen.

Beispielprogramm:

Das Programm berechnet den Tangens verschiedener Winkel, wobei ALPHA und BETA im Bogenmaß und GAMMA im Gradmaß gegeben ist.

```
100 REM ***** BEISPIELE ZUR TAN-FUNKTION *****
110 ALPHA=.7635 !BOGENMASS
120 BETA=-8.348 !BOGENMASS
130 GAMMA=75.39 !GRADMASS
140 PRINT ALPHA,TAN(ALPHA)
150 PRINT BETA,TAN(BETA)
160 PRINT GAMMA;" GRAD",TAN(GAMMA*PI/180)
```

# TRACE

Format:

TRACE

Beschreibung:

Der TRACE-Befehl bewirkt die Ausgabe der Programm-Zeilennummern auf dem Bildschirm, bevor die Befehle der jeweiligen Programmzeile während des Programmablaufs ausgeführt werden. Damit ist es möglich, den Programmablauf zu beobachten, um eventuell Fehler zu finden (bespielsweise eine unendlich oft durchlaufene Programmschleife).

Der TRACE-Befehl kann auch in einer Programmzeile auftreten. Seine Wirkung wird durch den NEW- oder UNTRACE-Befehl aufgehoben.

Beispielprogramm:

Die Unterprogrammschleife (Zeilennummern 200 bis 230) des Programms wird zweimal durchlaufen. Beim ersten Mal werden aufgrund des TRACE-Befehls in Zeile 120 die Zeilennummern mit ausgegeben; beim zweiten Mal aufgrund des UNTRACE-Befehls in Zeile 140 nicht.

```
100 REM ***** BEISPIEL ZUM TRACE UND UNTRACE *****
110 CALL CLEAR
120 TRACE
130 GOSUB 200
140 UNTRACE
150 PRINT
160 GOSUB 200
170 END
200 FOR I=1 TO 5
210 PRINT I
220 NEXT I
230 RETURN
```

## UNBREAK

Format:

UNBREAK[Zeilenliste]

Beschreibung:

Der UNBREAK-Befehl hebt die Stoppstellen im Programm auf, die durch den BREAK-Befehl gesetzt worden sind. Wird zusätzlich eine Zeilenliste angegeben, so werden nur die in der Zeilenliste aufgeführten Stoppstellen aufgehoben. Der UNBREAK-Befehl kann auch im Programm auftreten.

Beispiele:

Sind in einem Programm Stoppstellen gesetzt worden, so werden diese durch Programmzeile 100 alle wieder aufgehoben.

```
100 UNBREAK
```

Befinden sich in einem Programm z.B. Stoppstellen in den Zeilen 200, 300 und 400, so werden die ersten beiden Stoppstellen durch Programmzeile 120 aufgehoben, nicht aber die letzte.

```
120 UNBREAK 200,300
```

## UNTRACE

Format:

UNTRACE

Beschreibung:

Der UNTRACE-Befehl hebt die Wirkung des TRACE-Befehls auf. Er kann auch in einer Programmzeile auftreten.

Beispielprogramm:

S. TRACE-Befehl!

# VAL

Format:

VAL(String-Ausdruck)

Beschreibung:

Die VAL-Funktion wandelt den String-Ausdruck in einen Zahlenwert um. Dabei muß der String-Ausdruck eine gültige Darstellung einer numerischen Konstanten sein, andernfalls erscheint die Fehlermeldung * BAD ARGUMENT, und der Programmablauf wird unterbrochen. Die VAL-Funktion ist die Umkehrung der STR$-Funktion.

Beispiele:

In Programmzeile 100 wird der numerischen Variablen ZAHL der Wert 37.8 zugewiesen.

```
100 ZAHL=VAL("37.8")
```

Die numerische Variable A in Programmzeile 120 erhält den Wert von A$, sofern A$ eine gültige Zahlendarstellung ist.

```
120 A=VAL(A$)
```

Ist in einem Programm der String-Variablen N$ der Wert "14.7" zugeordnet worden, so wird in Programmzeile 140 die Zahl .0147 ausgegeben.

```
140 PRINT VAL(N$&"E"&"-3")
```

# VCHAR-Unterprogramm

Format:

```
CALL VCHAR(Zeilennummer, Spaltennummer, Zeichencode[,Wiederholungszahl])
```

Beschreibung:

Das VCHAR-Unterprogramm setzt ein Zeichen mit dem angegebenen Zeichencode an die Stelle des Bildschirms, die durch Zeilennummer und Spaltennummer angegeben wird. Die zusätzlich mögliche Wiederholungszahl bewirkt, daß das Zeichen entsprechend oft in die jeweilige Spalte gesetzt wird, also vertikal wiederholt wird, wobei Zeilennummer und Spaltennummer die Startstelle auf dem Bildschirm darstellen. Gegebenenfalls werden die Wiederholungen in den nächsten Spalten fortgesetzt.

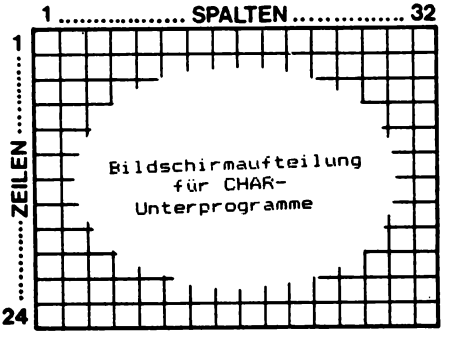
Zeilennummer, Spaltennummer, Zeichencode und Wiederholungszahl sind numerische Ausdrücke, deren Werte nach den üblichen Regeln berechnet werden. Führt die Auswertung der numerischen Ausdrücke nicht zu ganzen Zahlen, so werden sie entsprechend gerundet.

Die Numerierung der Zeilen und Spalten auf dem Bildschirm ist der nebenstehenden Darstellung zu entnehmen.

Soll der ganze Bildschirm mit einem Zeichen mit dem Zeichencode CODE beschrieben werden, so ist der Befehl

```
CALL VCHAR(1,1,CODE,768)
```

zu verwenden.




Beispiele:

In Programmzeile 100 wird das Zeichen 33 (Ausrufezeichen) in der 12. Zeile und 16. Spalte (etwa Bildschirmmitte) gesetzt, in Programmzeile 120 wird es zusätzlich 24 mal wiederholt.

```
100 CALL VCHAR(12,16,33)
120 CALL VCHAR(12,16,ASC("!"),24)
```

Beispielprogramm:

S. HCHAR-Unterprogramm!

## VERSION-Unterprogramm

Format:

```
CALL VERSION(numerische Variable)
```

Beschreibung:

Das VERSION-Unterprogramm ordnet der angegebenen numerischen Variablen einen Wert für die verwendete BASIC-Version zu. Dieser Wert beträgt 110 für EXTENDED BASIC.

Beispielprogramm:

Im Programm wird der Wert 110 an die numerische Variable TYP übergeben und anschließend über den Bildschirm ausgegeben.

```
100 REM ***** AUFRUF DER BASIC-VERSION *****
110 CALL CLEAR
120 CALL VERSION(TYP)
130 PRINT TYP
```

# Anhang

## A N H A N G

# Anhang 1

## ASCII-ZEICHENCODES

| CODE | ZEICHEN | CODE | ZEICHEN | CODE | ZEICHEN |
|---|---|---|---|---|---|
| 30 | | 63 | ? | 96 | ` |
| 31 | | 64 | § | 97 | a |
| 32 | | 65 | A | 98 | b |
| 33 | ! | 66 | B | 99 | c |
| 34 | " | 67 | C | 100 | d |
| 35 | # | 68 | D | 101 | e |
| 36 | $ | 69 | E | 102 | f |
| 37 | % | 70 | F | 103 | g |
| 38 | & | 71 | G | 104 | h |
| 39 | ' | 72 | H | 105 | i |
| 40 | ( | 73 | I | 106 | j |
| 41 | ) | 74 | J | 107 | k |
| 42 | * | 75 | K | 108 | l |
| 43 | + | 76 | L | 109 | m |
| 44 | , | 77 | M | 110 | n |
| 45 | - | 78 | N | 111 | o |
| 46 | . | 79 | O | 112 | p |
| 47 | / | 80 | P | 113 | q |
| 48 | 0 | 81 | Q | 114 | r |
| 49 | 1 | 82 | R | 115 | s |
| 50 | 2 | 83 | S | 116 | t |
| 51 | 3 | 84 | T | 117 | u |
| 52 | 4 | 85 | U | 118 | v |
| 53 | 5 | 86 | V | 119 | w |
| 54 | 6 | 87 | W | 120 | x |
| 55 | 7 | 88 | X | 121 | y |
| 56 | 8 | 89 | Y | 122 | z |
| 57 | 9 | 90 | Z | 123 | ä |
| 58 | : | 91 | Ä | 124 | ö |
| 59 | ; | 92 | Ö | 125 | ü |
| 60 | < | 93 | Ü | 126 | ß |
| 61 | = | 94 | ^ | 127 | |
| 62 | > | 95 | _ | 128 | |

| ZEICHENGRUPPEN-NUMMER TI BASIC | ZEICHENGRUPPEN-NUMMER EXTENDED BASIC | ZEICHENCODE |
|---|---|---|
| | 0 | 30 - 31 |
| 1 | 1 | 32 - 39 |
| 2 | 2 | 40 - 47 |
| 3 | 3 | 48 - 55 |
| 4 | 4 | 56 - 63 |
| 5 | 5 | 64 - 71 |
| 6 | 6 | 72 - 79 |
| 7 | 7 | 80 - 87 |
| 8 | 8 | 88 - 95 |
| 9 | 9 | 96 - 103 |
| 10 | 10 | 104 - 111 |
| 11 | 11 | 112 - 119 |
| 12 | 12 | 120 - 127 |
| 13 | 13 | 128 - 135 |
| 14 | 14 | 136 - 143 |
| 15 | | 144 - 151 |
| 16 | | 152 - 159 |

# Anhang 2

## CODES FÜR STEUERTASTEN

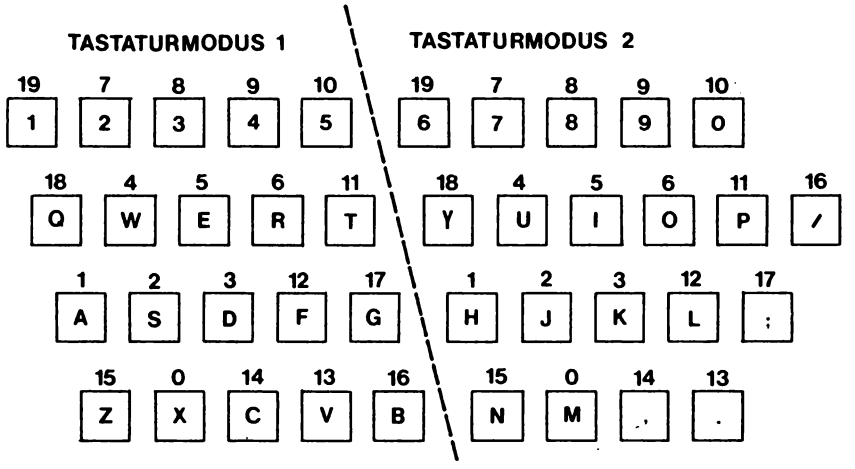
| BASIC-Modus | Pascal-Modus | Kurzzeichen | Tasten | Anmerkungen |
|---|---|---|---|---|
| 129 | 1 | SOH | <CONTROL-A> | Anfang des Kopfes |
| 130 | 2 | STX | <CONTROL-B> | Anfang des Textes |
| 131 | 3 | ETX | <CONTROL-C> | Ende des Textes |
| 132 | 4 | EOT | <CONTROL-D> | Ende der Übertragung |
| 133 | 5 | ENQ | <CONTROL-E> | Stationsaufforderung |
| 134 | 6 | ACK | <CONTROL-F> | Positive Rückmeldung |
| 135 | 7 | BELL | <CONTROL-G> | Klingel |
| 136 | 8 | BS | <CONTROL-H> | Rückwärtsschritt |
| 137 | 9 | HT | <CONTROL-I> | Horizontal-Tabulator |
| 138 | 10 | LF | <CONTROL-J> | Zeilenvorschub |
| 139 | 11 | VT | <CONTROL-K> | Vertikal-Tabulator |
| 140 | 12 | FF | <CONTROL-L> | Formularvorschub |
| 141 | 13 | CR | <CONTROL-M> | Wagenrücklauf |
| 142 | 14 | SO | <CONTROL-N> | Dauerumschaltung |
| 143 | 15 | SI | <CONTROL-O> | Rückschaltung |
| 144 | 16 | DLE | <CONTROL-P> | Datenübertragungsumschaltung |
| 145 | 17 | DC1 | <CONTROL-Q> | Gerätesteuerzeichen (X-EIN) |
| 146 | 18 | DC2 | <CONTROL-R> | Gerätesteuerzeichen |
| 147 | 19 | DC3 | <CONTROL-S> | Gerätesteuerzeichen (X-AUS) |
| 148 | 20 | DC4 | <CONTROL-T> | Gerätesteuerzeichen |
| 149 | 21 | NAK | <CONTROL-U> | Negative Rückmeldung |
| 150 | 22 | SYN | <CONTROL-V> | Synchronisierung |
| 151 | 23 | ETB | <CONTROL-W> | Ende des Übertragungsblocks |
| 152 | 24 | CAN | <CONTROL-X> | Ungültig |
| 153 | 25 | EM | <CONTROL-Y> | Ende der Aufzeichnung |
| 154 | 26 | SUB | <CONTROL-Z> | Substitutionszeichen |
| 155 | 27 | ESC | <CONTROL-.> | Code-Umschaltung |
| 156 | 28 | FS | <CONTROL-;> | Hauptgruppen-Trennzeichen |
| 157 | 29 | GS | <CONTROL-=> | Gruppen-Trennzeichen |
| 158 | 30 | RS | <CONTROL-8> | Untergruppen-Trennzeichen |
| 159 | 31 | US | <CONTROL-9> | Teilgruppen-Kennzeichen |

## CODES FÜR FUNKTIONSTASTEN

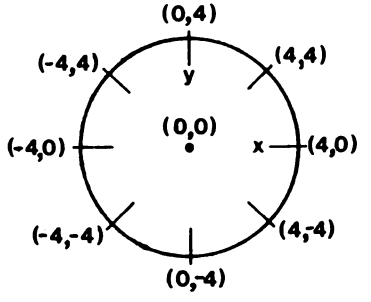
| BASIC-Modus | Pascal-Modus | Tasten | Tastenfunktion |
|---|---|---|---|
| 1 | 129 | <FCTN-7> | AID |
| 2 | 130 | <FCTN-4> | CLEAR |
| 3 | 131 | <FCTN-1> | DEL(ETE) |
| 4 | 132 | <FCTN-2> | INS(ERT) |
| 5 | 133 | <FCTN-=> | QUIT |
| 6 | 134 | <FCTN-8> | REDO |
| 7 | 135 | <FCTN-3> | ERASE |
| 8 | 136 | <FCTN-S> | ARROW LEFT |
| 9 | 137 | <FCTN-D> | ARROW RIGHT |
| 10 | 138 | <FCTN-X> | ARROW DOWN |
| 11 | 139 | <FCTN-E> | ARROW UP |
| 12 | 140 | <FCTN-6> | PROC'D |
| 13 | 141 | <ENTER> | ENTER |
| 14 | 142 | <FCTN-5> | BEGIN |
| 15 | 143 | <FCTN-9> | BACK |

# Anhang 3

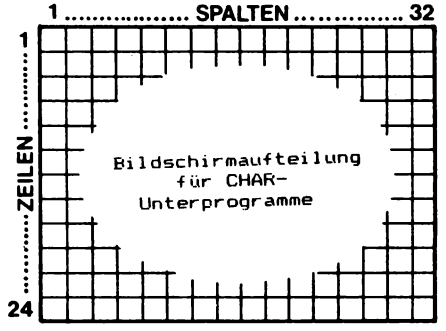
GETEILTE TASTATUR

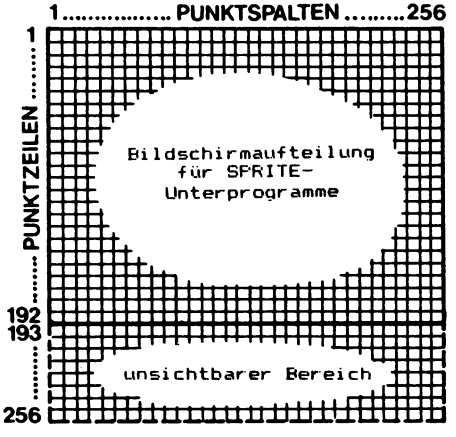



FERNBEDIENUNG



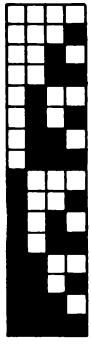
## Anhang 4


1 ..... SPALTEN ..... 32
1
ZEILEN
24
Bildschirmaufteilung
für CHAR-
Unterprogramme


1 ..... PUNKTSPALTEN ..... 256
1
PUNKTZEILEN
192
193
256
Bildschirmaufteilung
für SPRITE-
Unterprogramme
unsichtbarer Bereich

# Anhang 5

MUSTERCODIERUNG

| BLOCK | Binärcode | Codezeichen |
|---|---|---|
|  | 0000 | 0 |
| | 0001 | 1 |
| | 0010 | 2 |
| | 0011 | 3 |
| | 0100 | 4 |
| | 0101 | 5 |
| | 0110 | 6 |
| | 0111 | 7 |
| | 1000 | 8 |
| | 1001 | 9 |
| | 1010 | A |
| | 1011 | B |
| | 1100 | C |
| | 1101 | D |
| | 1110 | E |
| | 1111 | F |

| FARBCODE | FARBE |
|---|---|
| 1 | transparent |
| 2 | schwarz |
| 3 | mittelgrün |
| 4 | hellgrün |
| 5 | dunkelblau |
| 6 | hellblau |
| 7 | dunkelrot |
| 8 | kornblumenblau |
| 9 | mittelrot |
| 10 | hellrot |
| 11 | dunkelgelb |
| 12 | hellgelb |
| 13 | dunkelgrün |
| 14 | magentarot |
| 15 | grau |
| 16 | weiß |

## Anhang 6

MUSIK-TONFREQUENZEN

| FREQUENZ | NOTE |
|---|---|
| 110 | A |
| 116 | A# |
| 123 | H |
| 130 | C |
| 138 | C# |
| 146 | D |
| 155 | D# |
| 164 | E |
| 174 | F |
| 184 | F# |
| 195 | G |
| 207 | G# |
| 220 | A |
| 233 | A# |
| 246 | H |
| 261 | C |
| 277 | C# |
| 293 | D |
| 311 | D# |
| 329 | E |
| 349 | F |
| 369 | F# |
| 391 | G |
| 415 | G# |
| 440 | A |
| 466 | A# |
| 493 | H |
| 523 | C |
| 554 | C# |
| 587 | D |
| 622 | D# |
| 659 | E |
| 698 | F |
| 739 | F# |
| 783 | G |
| 830 | G# |
| 880 | A |
| 932 | A# |
| 987 | H |
| 1046 | C |
| 1108 | C# |
| 1174 | D |
| 1244 | D# |
| 1318 | E |
| 1396 | F |
| 1479 | F# |
| 1567 | G |
| 1661 | G# |

# Anhang 7

## MATHEMATISCHE FUNKTIONEN

Im folgenden sind einige der am häufigsten verwendeten mathematischen Funktionen zusammengestellt, die n i c h t unter ihrem Funktionsnamen im Sprachumfang des EXTENDED BASIC enthalten sind. Sie können z.B. mit Hilfe des DEF-Befehls in Programme eingebaut werden:

| FUNKTION | DEFINITION |
|---|---|
| Secans | DEF SEC(X)=1/COS(X) |
| Cosecans | DEF CSC(X)=1/SIN(X) |
| Cotangens | DEF COT(X)=1/TAN(X) |
| Arcussinus | DEF ARCSIN(X)=ATN(X/SQR(1-X*X)) |
| Arcuscosinus | DEF ARCCOS(X)=-ATN(X/SQR(1-X*X))+PI/2 |
| Arcussecans | DEF ARCSEC(X)=ATN(SQR(X*X-1))+(SGN(X)-1)*PI/2 |
| Arcuscosecans | DEF ARCCSC(X)=ATN(1/(SQR(X*X-1))+(SGN(X)-1)*PI |
| Arcuscotangens | DEF ARCCOT(X)=PI/2-ATN(X) oder =PI/2+ATN(X) |
| Sinushyperbolicus | DEF SINH(X)=(EXP(X)-EXP(-X))/2 |
| Cosinushyperbolicus | DEF COSH(X)=(EXP(X)+EXP(-X))/2 |
| Tangenshyperbolicus | DEF TANH(X)=-2*EXP(-X)/(EXP(X)+EXP(-X))+1 |
| Cotangenshyperbolicus | DEF COTH(X)=2*EXP(-X)/(EXP(X)-EXP(-X))+1 |
| Secanshyperbolicus | DEF SECH(X)=2/(EXP(X)+EXP(-X)) |
| Cosecanshyperbolicus | DEF CSCH(X)=2/(EXP(X)-EXP(-X)) |
| Arcussinushyperbolicus | DEF ARCSINH(X)=LOG(X+SQR(X*X+1)) |
| Arcuscosinushyperbolicus | DEF ARCCOSH(X)=LOG(X+SQR(X*X-1)) |
| Arcustangenshyperbolicus | DEF ARCTANH(X)=LOG(1+X)/(1-X))/2 |
| Arcuscotangenshyperbolicus | DEF ARCCOTH(X)=LOG((X+1)/(X-1))/2 |
| Arcussecanshyperbolicus | DEF ARCSECH(X)=LOG(1+SQR(1-X*X))/X) |
| Arcuscosecanshyperbolicus | DEF ARCCSCH(X)=LOG((SGN(X)*SQR(X*X+1)+1)/X) |

## Anhang 8

Die folgende Übersicht listet alle Fehlermeldungen auf, die im EXTENDED BASIC auftreten können, und zwar zunächst nach Fehler-Nummern, wie sie im ERR-Unterprogramm auftreten, und dann alphabetisch geordnet.

10 **NUMERIC OVERFLOW**
14 **SYNTAX ERROR**
16 **ILLEGAL AFTER SUBPROGRAM**
17 **UNMATCHED QUOTES**
19 **NAME TOO LONG**
20 **UNRECOGNIZED CHARACTER**
24 **STRING-NUMBER MISMATCH**
25 **OPTION BASE ERROR**
28 **IMPROPERLY USED NAME**
36 **IMAGE ERROR**
39 **MEMORY FULL**
40 **STACK OVERFLOW**
43 **NEXT WITHOUT FOR**
44 **FOR-NEXT NESTING**
47 **MUST BE IN SUBPROGRAM**
48 **RECURSIVE SUBPROGRAM CALL**
49 **MISSING SUBEND**
51 **RETURN WITHOUT GOSUB**
54 **STRING TRUNCATED**
56 **SPEECH STRING TOO LONG**
57 **BAD SUBSCRIPT**
60 **LINE NOT FOUND**
61 **BAD LINE NUMBER**
62 **LINE TOO LONG**
67 **CAN'T CONTINUE**
69 **COMMAND ILLEGAL IN PROGRAM**
70 **ONLY LEGAL IN A PROGRAM**
74 **BAD ARGUMENT**
78 **NO PROGRAM PRESENT**
79 **BAD VALUE**
81 **INCORRECT ARGUMENT LIST**
83 **INPUT ERROR**
84 **DATA ERROR**
97 **PROTECTION VIOLATION**
109 **FILE ERROR**
130 **I/O ERROR**
135 **SUBPROGRAM NOT FOUND**

Beachten Sie, daß diese Übersicht und die folgende Fehlerliste für TI-BASIC k e i n e Gültigkeit besitzt!

Wenn ein Fehler während des Programmablaufs auftritt, folgt der Fehlermeldung häufig eine Zeilennummer, die die Fehlersuche oft erleichtert.

In einem Unterprogramm auftretende Fehlermeldungen haben in vier Fällen eine andere Bedeutung. Dies sind:

| | | |
|---|---|---|
| 74 BAD ARGUMENT | bedeutet: | BAD VALUE |
| 57 BAD SUBSCRIPT | bedeutet: | BAD LINE NUMBER |
| 79 BAD VALUE | bedeutet: | INCORRECT ARGUMENT LIST |
| 14 SYNTAX ERROR | bedeutet: | STRING-NUMBER MISMATCH |

Folgen Sie dann den unter den rechts aufgeführten Meldungen stehenden Erläuterungen. Im letztgenannten Fall (Fehler-Nr. 14) kann es zu einem Systemausfall kommen, der nur durch Ausschalten und späteres Wiedereinschalten des Computers zu beheben ist.

## Anhang 8



### ALPHABETISCHE FEHLERLISTE

---

BAD ARGUMENT (74) FALSCHES ARGUMENT

---

1. Falscher Wert in ASC,ATN,COS,EXP,INT,LOG,SIN,SOUND,SQR,TAN,VAL.
2. Datenfeld-Element in einem SUB-Statement aufgeführt.
3. Falscher erster Parameter oder zu viele Parameter verwendet.

---

BAD LINE NUMBER (61) FALSCHE ZEILENNUMMER

---

1. Programmzeilennummern kleiner als 1 oder größer als 32767.
2. Programmzeilennummer nicht vorhanden.
3. RES(EQUENCE) erzeugt eine Programmzeilennummer kleiner als 1 oder größer als 32767.

---

BAD SUBSCRIPT (57) FALSCHER INDEX

---

1. Zu großer oder zu kleiner Index in einem Datenfeld (indizierte Variable).
2. Falscher Index bei DIM.

---

BAD VALUE (79) FALSCHER WERT

---

1. Falscher Wert in AND, CHAR, CHR$, CLOSE, EOF, FOR, GOSUB, GOTO, HCHAR, INPUT, MOTION, NOT, OR, POS, PRINT, PRINT USING, REC, RESTORE, RPT$, SEG$, SIZE, VCHAR, XOR.
2. Wert des Datenfeld-Indexes größer als 32767.
3. Dateinummer kleiner als 0 oder größer als 255.
4. Mehr als drei Töne und ein Geräusch bei SOUND aufgeführt.
5. Ein an Unterprogramm übergebener Wert kann von diesem nicht aufgenommen werden (z.B. Wert für Sprite-Geschwindigkeit kleiner als -128 oder Wert für ein Zeichen größer als 143).
6. Wert für die Zeilennummer bei ON GOTO oder ON GOSUB liegt nicht im Definitionsbereich.
7. Falsche Position nach AT bei ACCEPT oder DISPLAY.

---

CAN'T CONTINUE (67) FORTSETZUNG UNMÖGLICH

---

1. Nach einer Stoppstelle (breakpoint) im Programm wurde editiert.
2. Befehl wurde ohne vorhergehende Stoppstelle eingegeben.

---

COMMAND ILLEGAL IN PROGRAM (69) IM PROGRAMM UNZULÄSSIGER BEFEHL

---

Die reinen Commands BYE, CON, LIST, MERGE, NEW, NUM, OLD, RES, SAVE, wurden als Statements in einem Programm verwendet.

---

DATA ERROR (84) DATENFEHLER

---

1. READ oder RESTORE auf nicht vorhandene Daten bezogen oder Zuordnung eines String-Wertes zu einer numerischen Variablen.
2. Zeilennummern bei RESTORE ist größer als die größte Programmzeilennummer.
3. Fehler in durch LOAD angesprochener Datei.

## Anhang 8

FILE ERROR (109) DATEI-FEHLER

1. Falscher Datentyp wird durch READ zu lesen versucht.
2. Versuch, CLOSE, EOF, INPUT, OPEN, PRINT, PRINT USING, REC, RESTORE, für eine nicht existierende oder mit falschen Eigenschaften ausgestattete Datei zu verwenden.
3. Speicherplatz für Dateibenutzung zu gering.

FOR-NEXT NESTING (44) SCHLEIFEN-FEHLER

1. FOR- und NEXT-Befehle stimmen bei Schleifen nicht überein.
2. NEXT fehlt.

I/O ERROR (130) EIN-/AUSGABE-FEHLER

1. Fehler wurde beim Versuch, CLOSE, DELETE, LOAD, MERGE, OLD, OPEN, RUN, SAVE auszuführen entdeckt.
2. Speicherplatz reicht zum Auflisten eines Programmes nicht aus.

ILLEGAL AFTER SUBPROGRAM (16) NACH UNTERPROGRAMM UNZULÄSSIG

Etwas anderes als END, REM, SUB ist nach SUBEND vorhanden.

IMAGE ERROR (36) DARSTELLUNGSFEHLER

1. Fehler beim Gebrauch von DISPLAY USING, IMAGE, PRINT USING.
2. Mehr als 10 (E-Format) oder 14 (numerisches Format) maßgebliche Stellen in dem Format-String.
3. IMAGE-String umfaßt mehr als 254 Zeichen.

IMPROPERLY USED NAME (28) UNZULÄSSIGE BEZEICHNUNG

1. Unzulässige Variablen-Bezeichnung bei CALL, DEF, DIM.
2. Reserviertes Wort bei [LET] benutzt.
3. Indizierte oder String-Variable bei FOR benutzt.
4. Datenfeld mit falscher Anzahl von Dimensionen benutzt.
5. Benutzung einer nicht vorgesehenen Variablen-Bezeichnung. Nur Datenfelder, numerische oder String-Variablen oder ein benutzerdefinierter Funktionsname können Variablen sein.
6. Datenfelder mit unterschiedlichen Dimensionen haben gleiche Bezeichnung
7. Eine benutzerdefinierte Funktionsbezeichnung steht links vom Gleichheitszeichen.
8. In der Parameter-Liste eines Unterprogramms wird dieselbe Variable zweimal benutzt.

INCORRECT ARGUMENT LIST (81) UNKORREKTE ARGUMENT-LISTE

Falsche Zuordnung von Argumenten bei CALL und SUB.

LINE NOT FOUND (60) ZEILE NICHT GEFUNDEN

1. Falsche Zeilennummer bei BREAK, GOSUB, GOTO, ON ERROR, RUN, UNBREAK oder nach THEN oder ELSE.
2. Auszugebende (zu editierende) Zeile nicht gefunden.

LINE TOO LONG (62) ZEILE ZU LANG

Zeile ist für die Eingabe im Programm zu lang.

## Anhang B

MEMORY FULL (39) SPEICHER BELEGT

1. Programm für die Ausführung von DEF, DELETE, DIM, GOSUB, LET , LOAD, ON GOSUB, OPEN oder SUB zu umfangreich.
2. Programm zu umfangreich für das Hinzufügen einer weiteren Zeile, für das Einfügen einer Zeile, für den Ersatz einer Zeile oder für die Berechnung eines Ausdrucks.

MISSING SUBEND (49) SUBEND-BEFEHL FEHLT

Unterprogramm wird nicht durch SUBEND abgeschlossen.

MUST BE IN SUBPROGRAM (47) FÜR UNTERPROGRAMM NÖTIG

Im Unterprogramm fehlt SUBEND oder SUBEXIT.

NAME TOO LONG (19) BEZEICHNUNG ZU LANG

Variablen-Bezeichnung oder Unterprogramm enthält mehr als 15 Zeichen.

NEXT WITHOUT FOR (43) NEXT OHNE FOR

FOR fehlt am Schleifenbeginn, NEXT steht vor FOR, falsche FOR-NEXT-Zuordnung oder Verzweigung in eine Schleife.

NO PROGRAM PRESENT (78) KEIN PROGRAMM VORHANDEN

Programm fehlt für LIST, RES(EQUENCE), RESTORE, RUN, SAVE.

NUMERIC OVERFLOW (10) KAPAZITÄTSÜBERLAUF

1. Der nach einer mathematischen Operation oder bei ACCEPT, ATN, COS, EXP, INPUT, INT, LOG, SIN, SQR, TAN, VAL entstehende Wert einer Zahl ist zu groß oder zu klein für die Anzeige durch den Computer.
2. Eine Zahl außerhalb des Definitionsbereichs von -32768 bis 32767 bei PEEK oder LOAD.

ONLY LEGAL IN A PROGRAM (70) NUR IM PROGRAMM ZULÄSSIG

Reine Statements (DEF, GOSUB, GOTO, IF, IMAGE, INPUT, ON BREAK, ON ERROR, ON GOSUB, ON GOTO, ON WARNING, OPTION BASE, RETURN, SUB, SUBEND, SUBEXIT) wurden als Direkt-Befehle (commands) verwendet.

OPTION BASE ERROR (25) OPTION BASE-FEHLER

OPTION BASE mehr als einmal ausgeführt oder mit einem von 1 oder 0 abweichenden Wert gebraucht.

PROTECTION VIOLATION (97) PROGRAMMSCHUTZ VERLETZT

Versuch, ein geschütztes Programm zu speichern, aufzulisten oder auszugeben.

RECURSIVE SUBPROGRAM CALL (48) UNTERPROGRAMM-SELBSTAUFRUF

Unterprogramm ruft sich selbst direkt oder indirekt auf.

RETURN WITHOUT GOSUB (51) RETURN OHNE GOSUB

RETURN ohne vorhergehendes GOSUB oder Fehler bei vorangehender Fehlerbehandlung durch den ON ERROR-Befehl.

# Anhang 8

SPEECH STRING TOO LONG (56) SPRACH-STRING ZU LANG

Der durch SPGET ausgegebene Sprach-String umfaßt mehr als 255 Zeichen.

STACK OVERFLOW (40) SPEICHERÜBERLAUF

1. Zuviele Klammern () gesetzt.
2. Nicht genug Speicherplatz zur Ausdrucksberechnung oder Wertzuweisung.

STRING TRUNCATED (54) STRING-VERKÜRZUNG

1. Ein durch RPT$, Verkettung (&) oder durch benutzerdefinierte Funktion gebildete String umfaßt mehr als 255 Zeichen.
2. Ein String-Ausdruck bei VALIDATE umfaßt mehr als 254 Zeichen.

STRING-NUMBER MISMATCH (24) STRING/ZAHL-FEHLZUORDNUNG

1. Bei einer Funktion oder einem Unterprogramm wurde zur Berechnung, die einen numerischen Wert erfordert, wurde ein String vorgegeben.
2. Ein String wird einem numerischen Wert zugewiesen.
3. Versuch, eine Zahl zu verketten (&).
4. String wurde als Index benutzt.

SUBPROGRAM NOT FOUND (135) UNTERPROGRAMM NICHT GEFUNDEN

Ein aufgerufenes Unterprogramm existiert nicht oder ein durch LINK angesprochenes Unterprogramm wurde nicht geladen.

SYNTAX ERROR (14) SYNTAX-FEHLER

1. Formaler oder sprachlogischer Fehler wie: fehlende bzw. überzählige Kommata oder Klammern, fehlende oder falsch geordnete Parameter, falsche Reihenfolge von Befehlsbestandteilen, Rechtschreibefehler bei reservierten Wörtern usw.
2. DATA oder IMAGE nicht allein in und am Beginn einer Programmzeile.
3. Weitere Informationen nach abschließender ")".
4. Fehlendes "#" bei Sprites.
5. Fehlendes ENTER, fehlendes Trennsymbol (!) für Schlußkommentare, fehlendes Trennsymbol (::) bei mehrfach besetzten Programmzeilen.
6. Fehlendes THEN nach IF.
7. Fehlendes TO nach FOR.
8. Fehlende Informationen nach CALL, SUB, FOR, THEN oder ELSE.
9. Zweimal E bei einer numerischen Konstanten.
10. Falsche Parameterliste in einem Unterprogramm.
11. Aufruf oder Verlassen eines Unterprogramms mit GOTO, GOSUB, ON ERROR.
12. Aufruf von INIT ohne angeschlossene Speichererweiterung.
13. Aufruf von LINK oder LOAD ohne vorheriges INIT.
14. Benutzung einer Konstanten, wo Variable erforderlich.
15. Mehr als 7 Dimensionen in einem Datenfeld.

UNMATCHED QUOTES (17) ANFÜHRUNGSZEICHEN FALSCH

Beie einer Eingabe-Zeile stimmt die Zahl der Anführungszeichen nicht.

UNRECOGNIZED CHARACTER (20) ZEICHEN NICHT ERKANNT

1. Zeichen wie ? oder % wurden nicht innerhalb von Anführungszeichen verwendet.
2. Eine durch LOAD angesprochene Datei enthält falsches Dateifeld.

## Register

REGISTER

Im folgenden sind die wichtigsten Stichworte dieses Handbuchs mit den Seitenzahlen der Seiten, in denen sie erklärt oder angewendet werden, alphabetisch aufgeführt.

Ein Stern (*) hinter einem Stichwort bedeutet, daß es sich hierbei um ein Befehlswort handelt. Ein Stern hinter einer Seitenzahl verweist auf eine Seite in der Befehlsliste (teil B). Ein P hinter einer Seitenzahl verweist auf ein nicht unwichtiges Anwendungsbeispiel des Befehls in einem Beispielprogramm.

## Register

## Register

## Register

## Register

## Register

## Register

## Register


***Thanks to 99'er:***

***Peter Schaub***

*for the scan of this document.*

*- Scanned and Reworked by:*

*TI99 Italian User Club* *in the year 2016*

*(info@ti99iuc.it)*