M20

PERSONAL COMPUTER

BASIC 8000, Band II

Anwendung der Sprachelemente

olivetti

VORWORT

Das vorliegende Handbuch ist Band II der Dokumentation zu BASIC 8000 unter PCOS-Betriebssystem.

In Band I finden Sie die Einzelbeschreibungen der BASIC-Sprachelemente in alphabetischer Ordnung.

Wir empfehlen bei der Lektüre der BASIC 8000-Dokumentation vorab das Kapitel 4.1.2 durchzulesen. Dieses beschreibt, welche Bedeutung die Symbole und Darstellungsmittel (z.B. eckige/geschweifte Klammer, Negativdarstellung, Fettdruck) haben.

Die folgenden zusätzlichen Dokumentationen können im Papierlager der

Deutschen Olivetti DTS GmbH
Schmickstraße 14
6000 Frankfurt/M-Osthafen

bestellt werden:

* M20, Bedienungs- und Installationshandbuch
* PCOS, Betriebssystem
* V24-Peripherie, Programmierhandbuch
* IEC-Schnittstelle, Programmierhandbuch
* ISAM, indexsequentielle Dateiverwaltung
* Befehlsliste PCOS/BASIC 8000
* Bedienungsanleitungen für die verschiedenen Drucker

Zur Anwendung von Standardprogrammen, zusätzlichen Programmiermöglichkeiten oder zum Erlernen von BASIC können - ebenfalls beim Papierlager - deutsche Dokumentation mit den folgenden Themen bestellt werden:

* Multiplan
* Oliword
* Olisort
* Oliterm
* Olicom
* Bedienungsanleitung Fakturiersystem
* Bedienungsanleitung Finanzbuchführung
* Bedienungsanleitung Lohn-/Gehaltsabrechnung
* Datenverarbeitung mit BASIC (Programmierte Unterweisung)
* M20, Software-Katalog

Die gültigen Drucknummern entnehmen Sie bitte dem aktuellen M20-Software-Katalog.

Bei der

Deutschen Olivetti DTS GmbH
Lyoner Straße 34
6000 Frankfurt/M 71 (Niederrad)

finden außerdem laufend Kurse statt.

Themenkreise:

* Programmierung
* Betriebssysteme
* Textverarbeitung
* Betriebskalkulation mit Personal-Computer
* Dialog mit Anwenderprogrammen am Personal-Computer

Die Kurse erfordern unterschiedliche Vorkenntnisse.

Von Anwendern, die sich neu in ein Programm oder System einarbeiten wollen, bis hin zu Fachleuten, die sich Spezialkenntnisse aneignen wollen ——— alle finden etwas in diesem breiten Schulungsangebot.

Bitte fordern Sie dazu - unverbindlich und kostenlos - im Ausbildungszentrum an:

* den Ausbildungsplan des Ausbildungszentrums
* das Kursangebot der Olivetti-Personal-Computer-Schule.

BASIC 8000, BAND II, INHALTSVERZEICHNIS

1. DIE PROGRAMMIERSPRACHE BASIC

Personal-Computer werden nur durch ihre Programmierbarkeit in einer höheren Programmiersprache universell einsetzbar. Dabei hat BASIC wegen seiner leichten Erlernbarkeit und großen Flexibilität im kommerziellen und technischen Bereich für Personal-Computer besondere Bedeutung erlangt.

1.1 Definition BASIC

BASIC ist der Name der Programmiersprache, die aus den Anfangsbuchstaben von Beginners' All Purpose Symbolic Instruction Code gebildet wurde. Dieses Handbuch bezieht sich auf die Sprachform von BASIC, die für den Olivetti L1 M 20 Personalcomputer realisiert wurde.

Die Bezeichnung lautet:

L1.M20 BASIC-8000 Rev.5.20

Alle Beschreibungen beziehen sich auf diese von Olivetti verwendete Sprachversion.

Programme der Programmiersprache werden interpretierend ausgeführt. Dadurch wird große Flexibilität in den Sprachmöglichkeiten und bei der Erstellung und beim Testen von Programmen erreicht.

Bei Übernahme von Programmen anderer Sprachformen von BASIC auf M 20 muß geprüft werden, ob die zulässigen Regeln für das BASIC des L1 M 20 eingehalten wurden.

1.2 Anwahl von BASIC in PCOS

Nach dem Laden des Betriebssystems befindet sich das System in der PCOS-Ebene.

Um BASIC-Programme erstellen oder ausführen zu können, muß der BASIC-Interpreter aktiviert werden. Dazu stehen folgende Möglichkeiten zur Verfügung:

1. Eingabe des PCOS-Befehles **ba**
2. Vorhandensein eines Programmes INIT.BAS
3. Vorwahl während der Ausführung der Selbstprüfung durch Eingabe von **b**

Jede dieser Möglichkeiten bewirkt aus der PCOS-Ebene die Aktivierung und die Übergabe der Kontrolle an den BASIC-Interpreter. Am Bildschirm wird die jeweils geltende Fassung des aktuellen Betriebssystems angezeigt. Der dem Anwender verbleibende Speicherplatz hängt ab von

- der Speicherausrüstung und Hardwarekonfiguration des M 20,
- dem aktuellen Betriebssystem,
- der Anzahl von PCOS-Befehlen, die mit dem PCOS-Befehl **pl** in den Speicher geladen wurden.

zu 1. Der PCOS-Befehl **ba** (basic) bewirkt den Übergang in BASIC. Wird **ba** und dahinter ein Programmname angegeben, wird das entsprechende Programm gesucht, in den Arbeitsspeicher geladen und die Ausführung des Programms gestartet.

zu 2. Der M 20 sieht einen Selbststart vor. Enthält eine Diskette das Programm INIT.BAS, wird nach dem Laden des Systems in die BASIC-Ebene verzweigt und mit der Ausführung des Programmes INIT.BAS begonnen.

zu 3. Nach dem Einschalten des Systems oder physischem Reset, nicht aber logischem Reset (vgl. PCOS-Handbuch), wird vom System ein Selbsttest durchgeführt. Wird während der Ausführung dieses Tests die Taste **b** gedrückt gehalten, wird ebenfalls sofort der BASIC-Interpreter aktiviert. Ist ein Programm INIT.BAS vorhanden, wird es in diesem Fall nicht ausgeführt.

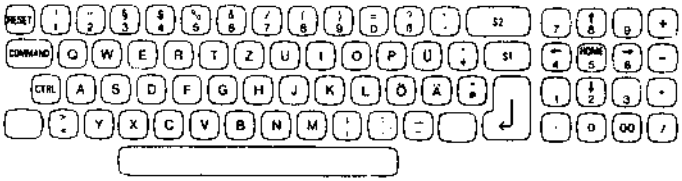
Der Betriebszustand BASIC kann verlassen werden durch:

- Eingabe des Befehles **SYSTEM**
- Drücken von RESET. Dies bewirkt ein Neuladen des Systems.

1.3 Zulässiger Zeichensatz in BASIC

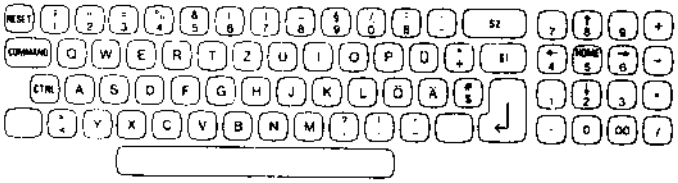
Die Tastatur des M 20 sieht einige Tasten vor, die in BASIC eine besondere Bedeutung haben. Ebenso hat die Programmiersprache BASIC Vorschriften über die Zulässigkeit von bestimmten Zeichen.

Der nachfolgende Abschnitt faßt die Regeln über die Verwendung der Tastatur zusammen und erklärt die Schreibweise, die für bestimmte Tastenkombinationen in diesem Handbuch gewählt wurde.

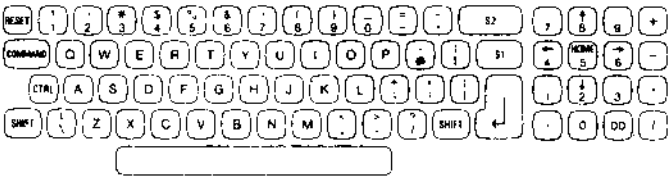


Deutsche Tastatur des M20

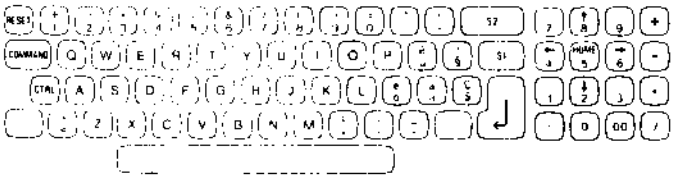
Mit Hilfe des PCOS-Befehles **sl** können andere nationale Tastaturen ausgewählt werden, z.B. die zweite deutsche Standard-Tastatur (**sl** 15),



der US-ASCII-Zeichensatz (**sl** 4)



oder die Schweizer Tastatur (deutschsprachige Version; **sl** 14)



1.3.1 Eingabelänge und Abschlußtasten

Alle Eingaben werden im aktiven Window angezeigt. Übersteigt die Anzahl der eingegebenen Zeichen die Länge der Zeile im Window, werden Eingabe und Anzeige in der nächsten Zeile fortgesetzt.

Die maximale Länge einer Eingabezeile beträgt 255 Zeichen. Da die Eingabe im allgemeinen früher abgeschlossen wird, sieht das System 3 Abschlußtasten vor:

↵ , S1, S2

Alle drei Tasten erzeugen das Abschlußzeichen CR (Carriage-Return, ISO-Code 13). Die Bezeichnung CR steht in der Folge generell für das Drücken einer der drei Abschlußtasten.

Anmerkung:

Auf Systemebene und beim Arbeiten mit dem Interpreter haben alle drei Abschlußtasten die gleiche Wirkung.

In Anwenderprogrammen kann durch Aufruf des PCOS-Befehles **lt** geprüft werden, welche der Abschlußtasten gedrückt wurde.

Wird ständig ohne CR eingegeben, so wird nach dem 256. Zeichen automatisch die Eingabe unterbrochen.

Das gilt sowohl bei der Durchführung von Eingaben im Command-Mode des Interpreters als auch bei der Ausführung von Programmen.

1.3.2 Die Umschalttasten

Am linken Rand der Tastatur sind drei Umschalttasten vorgesehen. Sie ermöglichen die volle Verwendung aller 255 möglichen Textzeichen, die jedoch nicht alle druckbar sind und teilweise eine spezielle Bedeutung haben.

Die Umschalttaste SHIFT befindet sich in der untersten Tastaturreihe links. Sie ist wie bei Schreibmaschinen auch auf der rechten Seite der untersten Tastenreihe vorhanden und bewirkt den Wechsel zwischen Groß- und Kleinschreibung bzw. den Wechsel zu dem Zeichen, das bei doppelt beschrifteten Tasten oben angegeben ist.

Darüber liegt die blaue Umschalttaste. Sie wird nachfolgend als Taste CONTROL bezeichnet. Die Taste CONTROL bewirkt die Erzeugung von Steuerzeichen. Im Handbuch wird die Verwendung einer Taste gemeinsam mit CONTROL durch **Negativdarstellung** dieser Taste dargestellt.

z.B. **H** Die Taste H wird gemeinsam mit CONTROL gedrückt.

Die gelbe (bzw. rote) Umschalttaste wird mit COMMAND bezeichnet. Sie wird überwiegend zur freien Belegung von Tasten (z.B. innerhalb von Programmen) verwendet, so daß diese Taste gemeinsam mit COMMAND die Wirkung einer Funktionstaste hat.

1.3.3 Groß- und Kleinschreibung

Mit der Tastatur des M20 können sowohl Groß- als auch Kleinbuchstaben erzeugt werden. Um für die Eingabe von Programmen die Tastaturhandhabung zu erleichtern, werden Kleinbuchstaben in den sinnvollen Fällen automatisch in Großbuchstaben umgewandelt.

Diese Verwandlung erfolgt in der BASIC-Eben für

alle BASIC-Schlüsselwörter und

alle Variablennamen.

Sie erfolgt nicht für Stringkonstanten (d.h. in Anführungszeichen eingeschlossene Strings) und auch nicht für Filenamen, die in Form von Stringkonstanten definiert sind.

Um bei der Eingabe von Folgen von Großbuchstaben nicht ständig die Umschalttaste drücken zu müssen, sieht das System eine Feststellfunktion vor.

Mit COMMAND und gleichzeitig der Taste - (unten rechts auf der Buchstabentastatur) wird auf Großbuchstaben umgeschaltet.

Diese Funktion gilt nicht für die Tasten mit Zweitfunktion (z.B. 1/;).

Die Großschreibung bleibt solange erhalten, bis erneut die Tasten COMMAND und - gleichzeitig gedrückt werden.

1.3.4 Spezielle Kontrollzeichen

Wesentliche Steuerfunktionen werden über Kontrollzeichen erreicht. Sie werden durch Drücken der blauen Taste (CONTROL) gemeinsam mit einer anderen Taste erzeugt und durch Negativdarstellung dieses Zeichens dargestellt.

C Bricht die Ausführung des Programmes oder eines unterbrechbaren Befehles ab und löscht den Inhalt des Tastaturpuffers.

G Erlaubt die verdeckte Eingabe von Zeichen. Durch Eingabe von G werden nachfolgend eingegebene Zeichen so lange nicht auf dem Bildschirm angezeigt, bis erneut G oder CR gedrückt wird. Aus der Form des Cursors kann ersehen werden, ob G aktiviert ist.

H Löscht das jeweils zuletzt eingegebene Zeichen. Der Cursor wird dabei um eine Position zurückgesetzt.

S Unterbricht die Ausführung einer Textausgabe am Bildschirm (Pausenfunktion). Das Drücken einer beliebigen anderen Taste bewirkt die Fortsetzung der Ausgabe.

1.3.5 Bildschirmcursor-Steuertasten

Im numerischen Teil der Tastur befinden sich die Tasten

→
←
↓
↑
HOME.

Nachstehend ist die Bedeutung der Cursor-Steuertasten für Olivetti-Programme aufgeführt, die mit normierter Programmierung erstellt wurden.

→ Cursor um eine Position nach rechts

← Cursor um eine Position nach links

↑ Vorbereitung zum Einfügen von Zeichen an der Cursor-Position

↓ Löschen eines Zeichens an der Cursor-Position

HOME Löschen des gesamten angezeigten Eingabefeldes

1.3.6 Programmierbare Tasten

Mit den Umschalttasten (SHIFT, CONTROL, COMMAND) und den Tasten des alphanumerischen Teiles der Tastatur können alle Zeichen erzeugt werden.

PCOS sieht über dem PCOS-Befehl **pk** die Möglichkeit vor, jeder Taste eine beliebige Textfolge zuzuordnen. Jedes Drücken der entsprechenden Taste oder Tastenkombination (z.B. COMMAND und Taste +) erzeugt die Zeichenfolge, die mit **pk** zugeordnet wurde.

Mit dem PCOS-Befehl **ck** können u.a. auch die Wirkungen der Sonderfunktionstasten (z.B. [H]) aufgehoben, geändert oder anderen Tasten zugewiesen werden.

Mit den PCOS-Befehlen **rf** und **wf** können - unter Zuhilfenahme des Full-Screen-Editors - eigene Zeichensätze für den Bildschirm erstellt werden.

Üblicherweise wird diese Tastenbelegung so lange aufrecht erhalten, bis das System neu geladen wird. Es kann jedoch über **ps** eine vom Anwender gewählte Tastenbelegung im Systemfile gespeichert werden, so daß bei jedem Laden des Systems diese Tastenbelegung wieder verfügbar ist. Dies gilt auch für die mit **sl** zuletzt ausgewählte nationale Tastatur.

2. ELEMENTE DER SPRACHE BASIC

Bei der Erstellung eines Programms wird vom Ersteller ein Ablauf festgelegt. Dieser Ablauf wird in Abhängigkeit von den aktuellen Daten bei jeder Ausführung des Programms ausgeführt.

Im Folgenden werden alle Sprachelemente beschrieben, die

- zur Eingabe und zum Testen eines Programms benötigt werden,
- die ein BASIC-Programm bilden können.

2.1 Aufbau eines BASIC-Programms

Ein BASIC-Programm besteht aus einer Folge von Anweisungen. Sie bilden Programmzeilen, die mit einer Zeilennummer versehen sind. Enthält eine Programmzeile mehrere Anweisungen, so sind die Anweisungen durch : (Doppelpunkt) getrennt. Eine Programmzeile kann maximal 255 Zeichen lang sein.

Eine Programmzeile hat daher folgendes Format:

Zeilennr. Anweisung [:Anweisung] ...

Zeilennumer: ganze, positive Zahl im Bereich von Ø bis 65529

Anweisung: Eine Folge von Sprachelementen, die nach den Vorschriften von Abschnitt 2.3 aufgebaut sind.

Die Zeilennummern bestimmen die Reihenfolge, in der die Programmzeilen im Arbeitsspeicher abgestellt werden. Dieser Programmaufbau legt auch den normalen Ablauf der Verarbeitung fest. Die Reihenfolge der Verarbeitung kann durch Sprachelemente verändert werden.

Innerhalb einer Programmzeile erfolgt die Ausführung der Anweisungen von links nach rechts.

Die Ablaufsteuerung kann sowohl durch Anweisungen im Programm als auch während der Ausführung, durch Eingabe von Befehlen im Direkt-Mode (vgl. Kapitel 3) erfolgen.

Das ordnungsgemäße Ende eines Programms kann durch:

- die Ausführung der Anweisung **END,**
- die Verzweigung in ein anderes Programm oder
- Erreichen der höchsten Zeilennumer im Programm

festgelegt werden.

2.2 Befehle

Befehle sind BASIC-Elemente, die eine unmittelbare Aktion des Computers bewirken. Befehle arbeiten mit Programmen oder Datenfiles, aber nicht mit Daten.

Befehle werden üblicherweise verwendet für

- die Eingabe von Programmen,
- das Ändern, Speichern, Dokumentieren,
- einzelne Operationen mit Datenbeständen.

Beispiel:

Nachdem ein Programm eingegeben wurde, soll zur Kontrolle und Dokumentation ein Protokoll am Drucker erstellt werden.

Durch Eingabe von

```
LLIST
```

werden alle im Arbeitsspeicher vorhandenen Programmzeilen in der Reihenfolge der aufsteigenden Zeilennummern gedruckt.

Anmerkung: Einige Befehle können auch sinnvoll innerhalb von Programmen verwendet werden.

2.3 Anweisungen

Ein Programm ist die Festlegung eines Ablaufs. Welche Aktion bei der Ausführung eines Programms durchgeführt wird, ist durch Anweisungen bestimmt.

Eine Anweisung besteht aus

- einem oder mehreren Schlüsselwörtern, die den Typ der Anweisung bestimmen,
- den Parametern der Anweisung. Sie bestimmen, mit welchen Daten die Anweisung auszuführen ist.

1. Anweisungen können als Bestandteil von Programmzeilen verwendet werden.

   Beispiel:

   10 **PRINT** A ist eine Anweisungszeile.
   10 ist die Zeilennummer der Programmzeile
   PRINT ist das Schlüsselwort. Die Anweisung bewirkt eine Ausgabe am Bildschirm.
   A ist der Datenteil der Anweisung. Er bestimmt, daß der Wert der Variablen A am Bildschirm anzuzeigen ist.

   Anweisungen müssen den syntaktischen Regeln entsprechend eingegeben werden (vgl. Kapitel 4.1.2).

Da sie nicht inmittelbar ausgeführt werden, erfolgt keine unmittelbare Überprüfung auf ihr gültiges Format. Diese Überprüfung erfolgt erst bei der Ausführung.

2. Anweisungen zur direkten Eingabe:

Beispiel:

Ein Programm soll auf seine Korrektheit geprüft werden. Dann soll der Wert von Daten an einer bestimmten Stelle geprüft werden.

Das Programm wird ausgeführt und an der gewünschten Stelle durch CONTROL in Verbindung mit der Taste C (C) ohne Eingabe der Abschlußtaste angehalten. Der Wert der Variablen A (siehe Abschnitt 2.5) soll überprüft werden.

PRINT A bewirkt das gewünschte Ergebnis. Es handelt sich hierbei um eine Eingabe, die unmittelbar ausgeführt wird, aber nicht der Definition des Befehls aus 2.2 entspricht.

Die Unterscheidung zwischen den Fällen 1. und 2. wird in Kapitel 3 erläutert.

2.4 Funktionen

Eine Funktion ist eine Berechnungsvorschrift, die in Abhängigkeit von Parametern (Argumenten) genau einen Wert als Ergebnis liefern. Viele Funktionen sind in BASIC standardmäßig erhalten.

Diese Funktionen heißen Standardfunktionen. Sie sind mit einem reservierten Namen versehen.

Zusätzlich hat der Programmierer die Möglichkeit, in einem Programm selbst Funktionen zu vereinbaren und an beliebigen nachfolgenden Stellen im Programm zu verwenden (siehe Abschnitt 4.3.7).

Eine Funktion kann immer nur innerhalb von Anweisungen aufgerufen werden.

Sie kann nicht allein stehen. Es muß in einer Anweisung festgelegt werden, was mit dem Ergebnis der Funktion gemacht werden soll.

Bei den vom Programmierer definierten Funktionen ist die Definition der Funktion ("der Vorschrift") eine Anweisung.

Der Aufruf der Funktion (mit ihrem Namen und aktuellen Parameter) ist Bestandteil jener Anweisungen, in denen der Funktionsname aufgerufen wird.

2.5 Daten

Die Anweisungen bilden die Vorschrift, nach der die Daten in einem Programm behandelt werden.

Die Daten sind in unterschiedliche Typen nach verschiedenen Gesichtspunkten gegliedert. Ihre interne Darstellung unterliegt bestimmten Vorschriften, die in den folgenden Abschnitten erläutert werden.

Daten können in einem Programm konstant, (während der Ausführung nicht veränderbar) oder variabel (während der Ausführung veränderbar) sein.

Daten kommen daher vor als:

- Konstanten
- Variablen.

Eine wesentliche Unterscheidung ist die Art der Daten. Daten können "Zahlen" sein, mit denen im Programm arithmetische Operationen ausgeführt werden (oder ohne Umwandlung ausführbar wären).

Solche Daten sind Numerische Daten.

Im Unterschied dazu sind beliebige Zeichenfolgen als Daten zulässig. Die Bedeutung solcher Zeichenfolgen wird nur durch die Interpretation des Anwenders bestimmt.

Beispiel:

Die Zeichenfolge "12345" wird vom Anwender als Zahl interpretiert. Das Programm behandelt sie jedoch nur als Folge von 5 Zeichen. Die Zeichenfolge "17.12.1984" ist mit Sicherheit eine Zeichenfolge, da sie zwei Punkte enthält und deshalb nicht mehr als Zahl verwendet werden kann.
Solche in einem Programm vorkommende Zeichenfolgen heißen Strings. Strings sind deshalb als Konstanten im BASIC immer in "" eingeschlossen, im Gegensatz zu Zahlen.

2.5.1 Numerische Daten

Numerische Daten sind Daten, mit denen ohne Umwandlung arithmetische Operationen (z.B. Addition) ausgeführt werden können. Solche Daten werden im Speicher des Rechners in spezieller Form gespeichert. Um eine möglichst optimale Verwendung des Speicherplatzes zu gewährleisten und unnötige Rechenzeiten zu vermeiden, werden numerische Daten entsprechend ihrem zulässigen Wertebereich in unterschiedliche Gruppen eingeteilt:

- Integer (ganzzahlig)
- Single Precision (einfache Genauigkeit)
- Double Precision (doppelte Genauigkeit)

Die Längenvereinbarung (Vereinbarung der Anzahl möglicher Ziffern bzw. belegter Bytes) für numerische Größen erfolgt einzig durch Definition der Gruppenzugehörigkeit ("Typisierung").

Es gilt:

Integer-Werte belegen zwei Bytes.
Werte in einfacher Genauigkeit belegen vier Bytes.
Werte in doppelter Genauigkeit belegen acht Bytes.

Numerische Daten können auf zwei Arten dargestellt werden:

- im Festkommaformat
- im Gleitkommaformat (floating-point-Format)

Die Bezeichnungen sind irreführend, da am M20 anstelle des Dezimalkommas stets der Dezimalpunkt verwendet werden muß.

Zahlen in Festkommaformat bestehen aus einem Vorzeichen, den Vorkommastellen, dem Dezimalpunkt und den Nachkommastellen.

Beispiele: 3.14 -68.2 -.523 1.001

Zahlen im Gleitkommaformat bestehen aus einer Mantisse incl. Vorzeichen, der Basis 10 und einer Hochzahl incl. Vorzeichen (Exponent). Der Wert der Zahl ergibt sich aus der Multiplikation von Mantisse mit 10 hoch Exponent.

Beispiele:

$-6.3*10^{2} = -630$ Mantisse: -6.3
Exponent: 2

$5.29*10^{4} = 52900$

$2.3*10^{-2} = .023$ 10 hoch -2 = 1/(10 hoch 2) = 1/100 = .01

$-68.94*10^{-3} = -.06894$

Ist der Exponent n positiv, wird in der Mantisse der Dezimalpunkt um n Stellen nach rechts versetzt. Ist n negativ, wird der Dezimalpunkt in der Mantisse um n Stellen nach links versetzt.

Es ist besonders zu beachten, daß somit für die gleiche Festkommazahl mehrere Gleitkommadarstellungen möglich sind.

Beispiel: $6.34 = .634*10^{1} = 634.0*10^{-2} = .00634*10^{3}$ etc.

Zur Ausgabe von Zahlen im Standardformat bzw. über Masken vgl. Kapitel 4.3.9.1.

2.5.1.1 Der Typ "Integer"

Numerische Daten vom Typ Integer sind immer ganzzahlig und haben einen Wertebereich von

$-32768 \leq \text{Integer} \leq 32767$

Ihre interne Darstellung erfolgt in zwei Bytes. Zur Unterscheidung von anderen Typen werden Integer-Daten mit dem Zeichen % gekennzeichnet.

Beispiel: -12%
VARIABLE%
sind numerische Daten vom Typ Integer.

Anmerkung:

Integer-Daten sparen sowohl Speicherplatz als auch Rechenzeit. Ein Integer-Wert belegt genau zwei Bytes.

2.5.1.2 Der Typ "Einfache Genauigkeit"

Ein numerischer Wert einfacher Genauigkeit ("single precision") wird bestimmt durch

- maximal 7 signifikante Ziffern
- einen Exponenten, der mit **E** angegeben wird,
- Konstanten oder Variablen, die am Ende mit ! (Rufzeichen) gekennzeichnet sind.

Der Bestandteil E steht für "*10 hoch" und kann auch bei der Eingabe bzw. Ausgabe eines Single-Precisions-Wertes verwendet werden.

Der Wertebereich für numerische Daten in einfacher Genauigkeit lautet:

$-3.40282E38 \leq X1 \leq 3.40282E38$

Achtung:

Eingaben, die diese Grenze überschreiten, führen zum Overflow. Es wird aber - nach Fehlermeldung - mit dem niedrigsten bzw. höchsten intern darstellbaren Wert sofort weitergearbeitet.

Numerische Daten, für die kein eigener Typ angegeben wird, werden als Single-Precision interpretiert.
Ein Single-Precision-Wert belegt genau 4 Bytes.

2.5.1.3 Der Typ "Doppelte Genauigkeit"

Ein numerischer Wert hat doppelte Genauigkeit ("double precision"), wenn

- er mehr als 7 signifikante Ziffern hat oder
- der Exponent mit **D** dargestellt wird oder
- am Ende ein #-Zeichen als Kennzeichen verwendet wird.

Der Bestandteil **D** steht für "*10 hoch" und kann auch bei der Eingabe bzw. Ausgabe eines numerischen Wertes verwendet werden. Es sind maximal 15 signifikante Zeichen möglich.

Der Wertebereich für numerische Daten in doppelter Genauigkeit lautet

-1.79769313486231D308 ≤ X# ≤ 1.79769313486231D308

Es sind zwar Eingaben im Intervall

-35953862697246D308 ≤ X# ≤ 3.5953862697246D308

möglich, intern wird jedoch mit den obigen Werten weitergearbeitet. Der Rest ist Signifikanzverlust. Ein Double-Precision-Wert belegt genau 8 Bytes.

Für Overflows gilt die Anmerkung am Ende von 2.5.1.2.

2.5.1.4 Typenumwandlung

Falls erforderlich, führt BASIC automatisch Typenumwandlungen durch. Dabei bestehen folgende Möglichkeiten:

1. Falls eine numerische Konstante eines Typs einer numerischen Variablen eines anderen Typs zugewiesen wird, erfolgt die Zuweisung entsprechend dem Typ der Variablen.

   Beispiel:
   ```
   10 A%=23.42
   20 PRINT A%
   RUN
    23
   ```

2. Bei Rechenoperationen mit Werten unterschiedlichen Typs erfolgt eine Umwandlung aller Operanden in den Typ der höchsten vorkommenden Genauigkeit. Das Ergebnis hat ebenfalls diese Genauigkeit.

   Beispiel: a)
   ```
   10 D#=6#/7%
   20 PRINT D#
   RUN
    .8571428571428571
   ```

   b)
   ```
   10 D=6#/7%
   20 PRINT D
   RUN
    .857143
   ```

   In Beispiel a) wurde die Division in doppelter Genauigkeit ausgeführt und der doppelt genauen Variablen **D#** zugewiesen.

   In Beispiel b) wurde die Division ebenfalls in doppelter Genauigkeit ausgeführt, die Zuweisung an die einfach genaue Variable **D** hat jedoch eine neuerliche Typenumwandlung in einfache Genauigkeit bewirkt.

3. Logische Operatoren (siehe Abschnitt 2.6.3) erfordern Werte vom Typ Integer. Bei der Umwandlung wird nur der ganzzahlige Wert (nach kaufmännischer Rundung) berücksichtigt.

4. Wird eine Gleitkommazahl (Single- oder Double-Precision) in einen Integer-Wert umgewandelt, wird der Wert auf Ganzzahligkeit kaufmännisch gerundet.

```
10 C%=55.88
20 PRINT C%
RUN
 56
```

5. Wird einer doppelt genauen Variablen ein Wert in einfacher Genauigkeit zugewiesen, sind nur die ersten sieben Ziffern signifikant. Die nachfolgenden Stellen sind nicht sicher genau. Die Differenz zwischen dem ursprünglichen Wert in einfacher Genauigkeit und dem Inhalt der doppelt genauen Variablen ist in jedem Fall kleiner als 6.3E-8 mal dem ursprünglichen Wert.

Beispiel:

```
10 A=2.04
20 B#=A
30 PRINT A;B#
RUN
 2.04  2.03999961853027
```

Die Ursache hierfür liegt in der internen binären Darstellung der Zahlen, sowie der immer möglichen Ungenauigkeit bei der Errechnung von gebrochenen Zahlen.

Wird in Anweisung 20 B#=**VAL(STR$(**A**))** gesetzt, sind beide Variableninhalte gleich, da auf dezimaler Ebene gerechnet und zugewiesen wird.

6. Werden doppelt genaue Werte außerhalb des für einfach genaue Werte zulässigen Intervalls (vgl. Kapitel 2.5.1.2) in einfach genaue Werte verwandelt, sind zwei Fälle zu unterscheiden:

   a) Liegt der zu konvertierende doppelt genaue Wert außerhalb des Intervalls $-1.80925139D75 \leq X\# \leq 1.80925139D75$

   wird die Meldung "Overflow" gegeben. Es wird jedoch mit den verschiedensten völlig falschen Werten weitergerechnet!!!

   b) Wenn der zu konvertieren doppelt genaue Wert zwar unterhalb des Intervalls zu a), aber über dem Intervall

   $$-3.40282D38 \leq X\# \leq 3.40282D38$$

   liegt, erfolgt keine Overflow-Meldung. Intern wird mit dem niedrigsten bzw. höchsten in einfacher Genauigkeit intern darstellbaren Wert (vgl. Intervall) weitergerechnet!

   Anmerkungen:

   - Werden numerische Konstanten bei der Eingabe eines Programmes am Ende mit % definiert, wird dieses Kennzeichen beim Programm-Listing weggelassen (Ausnahme: **DATA**-Anweisung).
   - Ein Underflow existiert nicht; für extrem nahe an Ø liegende Daten wird genau Ø angesetzt und weitergearbeitet.

7. Bei wiederholten Rechenvorgängen selbst innerhalb von einfacher oder doppelter Genauigkeit können ebenfalls Ungenauigkeiten auftreten.

Beispiele:

```
1 CLEAR:GOSUB 1001:T1!=PH!
10 X#=0:'1000 x doppelt genaue .1 addieren
20 FOR I%=1 TO 1000:X#=X#+.1#:NEXT I%:GOSUB 1001
30 PRINT X#;"...Zeitdauer:"PH!-T1!"Sekunden":END
1000 'Berechnen Sekunden
1001 PH!=VAL(RIGHT$(TIME$,2))+60*VAL(MID$(TIME$,3,2))
+3600*VAL(LEFT$(TIME$,2)):RETURN
RUN
 99.999999999986 ...Zeitdauer: 3 Sekunden

1 CLEAR:GOSUB 1001:T1!=PH!
10 X#=0:'1000 x einfach genaue .1 addieren
20 FOR I%=1 TO 1000:X#=X#+.1:NEXT I%:GOSUB 1001
30 PRINT X#;"...Zeitdauer:"PH!-T1!"Sekunden":END
1001 PH!=VAL(RIGHT$(TIME$,2))+60*VAL(MID$(TIME$,3,2))
+3600*VAL(LEFT$(TIME$,2)):RETURN
RUN
 100.000001490116 ...Zeitdauer: 3 Sekunden
```

Werden die Zahlen vor Rechenoperationen über **STR$** und **VAL** (Stringverarbeitung) konvertiert, wird auf dezimaler Basis gerechnet. Die Wahrscheinlichkeit für Genauigkeitsverluste ist dann äußerst minimal. Die Verarbeitungszeit steigt allerdings erheblich.

Beispiele:

```
1 CLEAR:GOSUB 1001:T1!=PH!
10 X#=0:'1000 x einfach genaue .1 über Stringverarbeitung
20 FOR I%=1 TO 1000:X#=VAL(STR$(X#))+VAL(STR$(.1))
25 NEXT I%:GOSUB 1001
30 PRINT X#;"...Zeitdauer:"PH!-T1!"Sekunden":END
1001 PH!=VAL(RIGHT$(TIME$,2))+60*VAL(MID$(TIME$,3,2))
+3600*VAL(LEFT$(TIME$,2)):RETURN
RUN
 100 ...Zeitdauer: 13 Sekunden
```

2.5.1.5 Rundungen

Müssen, z.B. aufgrund interner Typumwandlungen, automatische Rundungen erfolgen, gilt:

1. Es wird kaufmännisch gerundet. Bei einigen Operationen wird von diesem Prinzip abgewichen; dies ist dann bei der Beschreibung der Operationen vermerkt.

2. Für die Rundung der 5 gilt:

   a) Wird eine Rundung von ! oder # nach % vorgenommen, wird stets auf den nächstgrößeren Wert (rechts auf der Zahlenskala) gerundet, sofern hinter der 5 nicht noch eine Ziffer folgt.

   Beispiele:

```
A%=3.5:B%=3.49:C%=3.51:?A%;B%;C%
 4  3  4
A%=-3.5:B%=-3.49:C%=-3.51:?A%;B%;C%
-3 -3 -4
```

   b) Wird eine Rundung aufgrund einer Umwandlung von # nach ! oder von ! nach # oder aufgrund einer zu hohen Anzahl signifikanter Stellen für den Typ vorgenommen, wird stets auf den im Betrag größeren Wert gerundet (echte kaufmännische Rundung).

   Beispiele:

```
A!=12345.65:B!=-12345.65
?A!;B!
 12345.7 -12345.7
C#=12345678Ø123455:D#=-12345678Ø123455
?C#;D#
 12345678Ø12346 -12345678Ø12346
```

2.5.2 Strings

Strings sind eine beliebige Folge von Zeichen.

Die Anzahl der Zeichen, die eine Zeichenfolge bilden, heißt Länge des Strings. Die maximale Länge eines Strings beträgt 255 Zeichen. Ein String, der keine Zeichen enthält, heißt "Leerstring". Der Leerstring wird durch **""** (2 Anführungszeichen direkt hintereinander) dargestellt. Der Leerstring hat die Länge Ø. Nach einmaliger Betätigung der Leertaste entsteht ein Blank. Das Blank ist ein Zeichen wie alle anderen auch. Es hat den ISO-Code 32. Gelegentlich wird es in diesem Handbuch durch das Zeichen ß (anstelle der Leertaste) dargestellt.

Die Speicherung von Strings erfolgt entsprechend ihrer Länge. Ein String braucht daher umsomehr Speicherplatz, je mehr Zeichen er enthält. Bei der Ausführung von Programmen wird der Speicherplatz für Strings entsprechend ihrer aktuellen Länge dynamisch verwaltet. (Der Rechner belegt intern nur soviel Platz, wie für den String benötigt wird.)

2.5.3.1 Konstanten

Numerische Konstanten werden in Programmen mit ihrem Wert dargestellt. Sie haben ohne explizite Angabe eines Typenkennzeichens einfache Genauigkeit, wenn die Konstante weniger als 7 Ziffern lang ist oder der Exponent mit **E** angegeben wurde. Sie haben doppelte Genauigkeit, wenn sie mehr als 7 Ziffern haben oder der Exponent mit **D** bezeichnet wurde oder das Typenkennzeichen # angehängt ist. Sie sind vom Typ Integer, wenn das Typenkennzeichen % angehängt ist (nur bei **DATA**-Anweisung).

Numerische Konstanten können auch oktal oder hexadezimal angegeben werden.

Hexadezimale Konstanten sind mit dem Präfix **&H** anzugeben. Beispiele für hexadezimale Konstanten sind:

&H76 (Dezimal 118)
&H3F2 (Dezimal 1Ø1Ø)

Oktale Konstanen werden mit dem Präfix **&O** angegeben. Beispiele für oktale Konstanten sind:

&O347 (Dezimal 231)
&O12 (Dezimal 1Ø)

In BASIC kann statt **&O** auch nur **&** vorausgesetzt sein, um oktale Konstanten zu definieren.

Die Typumwandlung von Konstanten in die internen Datenformate wird automatisch durchgeführt.

Stringkonstanten sind Zeichenfolgen, die im Programm durch Anführungszeichen am Anfang und am Ende des Strings gekennzeichnet werden.

Beispiele: Numerische Konstanten:	Stringkonstanten:
12%	"15.1Ø.1982"
157.25	"MAX"
-3.15#	"Basic-Zeile"
1ØE57	"125.-"
3.141592765#	"4711"
1D-15	

2.5.3.2 Variablen

Daten, die im Ablauf des Programms veränderbar sind, heißen variable Daten. Sie werden in Variablen gespeichert.

Eine Variable ist ein Speicherplatz, der durch einen Namen identifiziert wird.

Eine Variable wird charakterisiert durch:

- ihren Namen
- ihren Typ
- ihren aktuellen Inhalt.

Variablennamen: Der Name einer Variablen ist eine Folge von maximal 40 Zeichen, wobei das erste Zeichen ein Buchstabe sein muß.

Folgende Zeichen sind als eventuelle weitere Zeichen (bis zu 38) zulässig:

- Buchstaben (A bis Z, keine Umlaute) und Ziffern (0 bis 9). Kleinbuchstaben können zwar eingegeben werden, sie werden jedoch vom M20 automatisch als Großbuchstaben interpretiert und bei Listings in solche verwandelt.
- Punkt (.) und Unterstreichungszeichen (_).

Das Typkennzeichen **!**, **%**,**#** , **$**, kann an der letzten Stelle, gegebenenfalls auch als 40. Zeichen, eingesetzt werden.

Unzulässig ist die Verwendung von reservierten Wörtern als Variablennamen.

Reservierte Wörter sind:

- die Schlüsselwörter der Befehle und Anweisungen (stets **fett** gedruckt)
- die Namen der Standardfunktionen (stets **fett** gedruckt)
- **DATE$, TIME$** als spezielle Variablen zur Steuerung und Abfrage des Datums und der Uhr
- **ERR** und **ERL** für Fehlerroutinen als reservierte Variablen
- Ebenfalls vom System benötigt und deshalb gesperrt sind die Variablennamen **USR, PEEK** und **POKE.**
- Variablennamen dürfen nicht mit der Buchstabenfolge **FN** beginnen, da diese für Aufrufe von selbstdefinierten Funktionen reserviert ist.
- Wird mit indexsequentiellen Files unter Zuhilfenahme von ISAM-Assembler-Programmroutinen gearbeitet, sind im betreffenden BASIC-Programm die Array ISAM$(1Ø) und ISAM%(1Ø) zu dimensionieren und nur für bestimmte Arbeiten im Zusammenhang mit der Verwaltung der indexsequentiellen Files zu verwenden.
- Wird mit der "normierten Programmierung" der DEUTSCHEN OLIVETTI gearbeitet, sind die Anfangsbuchstaben O, P und Q für die dort verwendeten Variablennamen reserviert, da die in den Modulen verwendeten Variablen stets mit O, P oder Q beginnen. (Es ist standardmäßig ein **DEFSNG** O und ein **DEFSNG** P und ein **DEFDBL** Q erfolgt; von dieser Vereinbarung abweichende Variablennamen sind durch Typkennzeichen zu kennzeichnen).
- Eine Zusammenstellung aller gesperrten Variablennamen findet sich im Kapitel 4.2.2.

Typenvereinbarung von Variablen

Variablen können auf folgende Arten in ihrem Typ festgelegt werden:

1. Durch Angabe des Typkennzeichens am Ende und als Bestandteil des Namens
   % legt eine Variable als Integer-Variable fest
   ! legt eine Variable in einfacher Genauigkeit fest
   # legt eine Variable in doppelter Genauigkeit fest
   $ legt eine Stringvariable fest

2. Implizit werden numerische Variablen in einfacher Genauigkeit festgelegt, falls keine Typvereinbarung erfolgte.

3. Durch Verwendung der Anweisungen **DEFINT, DEFSNG, DEFDBL, DEFSTR.** Es kann durch diese Anweisungen für Gruppen von Anfangsbuchstaben von Variablennamen ein bestimmter Typ festgelegt werden.

   Beispiel:

   10 **DEFINT** I-N bewirkt, daß alle Variablen, deren Namen den Anfangsbuchstaben I bis N haben und die kein Typkennzeichen am Ende haben, als Integer-Variable vereinbart sind.

Die Vereinbarung von Variablen

Variablen werden durch ihr erstes Auftreten im Programm vereinbart; dabei wird numerischen Variablen der Wert Ø, Stringvariablen der Leerstring zugewiesen.

Bei der Verbindung mehrerer Programme können die Variablen des vorangehenden Programms mit ihren aktuellen Werten in das nachfolgende Programm übernommen werden.

2.5.3.3 Arrays

Arrays sind eine Form von Variablen, bei der einem Name mehrere Werte zugeordnet sind. Zur Identifikation jedes einzelnen Wertes müssen zusätzlich zum Namen noch ein oder mehrere Indices angegeben werden. Die Indices werden in Klammern eingeschlossen und durch Komma getrennt. Für die Namen von Arrays gelten dieselben Vorschriften wie in Abschnitt 2.5.3.2 beschrieben. Arrays können in allen zulässigen Variablentypen gebildet werden. Die Dimension und die maximalen Werte der Indices jeder Dimension muß für jeden Array in einer **DIM**-Anweisung festgelegt werden.

Beispiele für Variablennamen:

A! oder A	Num. Variable in einfacher Genauigkeit
PI#	Num. Variable in doppelter Genauigkeit
MAXIMUM%	Integervariable
BEZEICHNUNG$	Stringvariable
KUNDEN.NAME$	Stringvariable
KUNDEN_ADRESSE$	Stringvariable
MONAT%(3)	Bezeichnet das dritte Element des Arrays MONAT%(), in dem jedes Array-Element einen Integer-Wert annehmen kann.
KOEFFIZIENTENMATRIX (ZEILE%,SPALTE%)	Bezeichnet das Element in Zeile 'ZEILE%' und Spalte 'SPALTE%' des zweidimensionalen Arrays KOEFFIZIENTENMATRIX(), in dem jedes Array-Element einen einfach genauen Wert annehmen kann.

2.6 Operatoren

Operatoren werden zur Verknüpfung von Daten verwendet. Sie werden nach den Ergebnissen, die sie liefern, klassifiziert.

2.6.1 Arithmetische Operatoren

Mit arithmetischen Operatoren können numerische Daten verknüpft werden, wobei Berechnungsvorschriften festgelegt werden.

Arithmetische Operatoren werden mit unterschiedlicher Priorität ausgeführt, wenn sie gemeinsam verwendet werden.

Es stehen folgende arithmetische Operatoren in absteigener Priorität zur Verfügung:

Operator	Operation	Beispiel
^	Potenzierung	X^Y
-	Negation (Vorzeichen)	-X
*, /	Multiplikation, Division	X*Y, X/Y
\	Integerdivision	X\Y (Das Zeichen \ (backlash) wird im deutschen Zeichensatz durch den Großbuchstaben Ö dargestellt.)
MOD	Rest einer Intger-division (Modulo)	X **MOD** Y
+, -	Addition, Subtraktion	X+Y, X-Y

Regeln für arithmetische Operatoren

- Potenzierung

B^EXPO Die Basis B wird zur Potenz EXPO erhoben.

Sonderfälle:

Basis B	Exponent EXPO	Ergebnis B^EXPO
beliebig	Ø	1
Ø	< Ø	"Division by zero"
Ø	> Ø	Ø
< Ø	≠ INT(EXPO)	"Illegal function call"

- Multiplikation und Addition:

A*B A wird mit B multipliziert

A+B B wird zu A addiert

Regeln:

Es gilt das kommutative Gesetz:

A*B = B*A

A+B = B+A

Das transitive Gesetz gilt nicht uneingeschränkt. Durch die Reihenfolge der Operationen können minimale Abweichungen entstehen:

A*(B*C) ≈ (A*B)*C

A+(B+C) ≈ (A+B)+C

Es entstehen gelegentlich Signifikanz-Unterschiede bei der internen Abspeicherung von Zwischenergebnissen (vgl. Beispiele Kapitel 2.5.1.4).

Wird keine Klammer gesetzt, erfolgt die Berechnung des Ergebnisses von links nach rechts, sofern keine Prioritätsunterschiede bestehen.

- Division und Subtraktion

A/B A wird durch B dividiert
A-B B wird von A subtrahiert

Die Division durch Null führt zur Fehlermeldung "Division by zero".

Erlaubte Vorzeichenkombinationen sind zum Beispiel:

-B+(-A)+C-(-Z)
A+-B
A*-B
A++B

- Integer-Division

Die Operation ist durch das Sprachelement (backslash) gekennzeichnet und liefert den ganzzahligen Teil einer Division. (Das Zeichen wird auf der deutschen Tastatur des M20 durch den Großbuchstaben Ö dargestellt.) Die **Operanden** werden auf Ganzzahligkeit **kaufmännisch gerundet** und müssen im Bereich von -32768≤X≤32767 sein. Das **Ergebnis** wird auf seinen ganzzahligen Teil **reduziert.**

Beispiele:

10 Ö 4 liefert den Wert 2
25.68Ö6.99 liefert den Wert 3, denn 26 dividiert durch 7 ist 3 Rest 5
20.51Ö7.49 liefert den Wert 3, denn 21 dividiert durch 7 ist 3 Rest 0

- Modulo-Berechnung

Die Operation ist durch das Sprachelement **MOD** gekennzeichnet und liefert den Divisionsrest einer ganzzahligen Division. Die Operanden müssen ebenfalls im Intervall $-32768 \leq X \leq 32767$ liegen.

Beispiele:

11 **MOD** 4 liefert den Wert 3 (11Ö4 hat das ganzzahlige Ergebnis 2 mit Divisionsrest 3)

-25.68 **MOD** 6.99 liefert den Wert -5 (-26Ö7 hat das ganzzahlige Ergebnis -3 mit Rest -5)

- Prioritätsregeln

Die numerischen Operatoren haben in nachstehender Reihenfolge Priorität

$$\hat{} \quad - \quad \begin{matrix} * \\ / \end{matrix} \quad \text{Ö} \quad \textbf{MOD} \quad \begin{matrix} + \\ - \end{matrix}$$

höchste Priorität ⟶ niedrigste Priorität

Die Operatoren * und / sowie + und - haben jeweils die gleiche Priorität. Operatoren gleicher Priorität werden von links nach rechts verarbeitet, es sei denn, daß durch Klammern etwas anderes definiert wird; Klammern durchbrechen die Prioritätsregeln.

Beispiele: -3^2 liefert -9, bzw. (-3)^2 liefert 9
8/4*2 liefert 4, bzw. 8/(4*2) liefert 1

- Overflow und Division durch Null

  Bei Division durch Ø oder einer Potenzierung von Ø mit negativem Exponenten wird die Meldung "Division by zero" ausgegeben. Als Ergebnis wird die größtmögliche darstellbare Zahl mit richtigem Vorzeichen gesetzt und die Ausführung **fortgesetzt.**

  Falls ein Overflow auftritt, wird ebenfalls die größtmögliche darstellbare Zahl mit richtigem Vorzeichen als Ergebnis angenommen. Es wird die Meldung "Overflow" angezeigt und die Ausführung **fortgesetzt.**

2.6.2 Vergleichsoperatoren

Vergleichsoperatoren erlauben den Vergleich von 2 numerischen Werten oder 2 Strings.

Das Ergebnis kann "wahr" oder "falsch" sein. Vergleiche werden zur Steuerung des Programmablaufes verwendet.

Operator	Prüfung	Beispiel
=	Gleichheit	X = Y
<>	Ungleichheit	X<>Y
<	kleiner als	X < Y
>	größer als	X > Y
< =	kleiner oder gleich	X < =Y
> =	größer oder gleich	X > =Y

Der Vergleich von numerischen Werten erfolgt entsprechend ihrem Wert. Beim Vergleich von Strings wird zeichenweise von links beginnend der Vergleich durchgeführt.

2 Strings sind gleich, wenn sie in allen ihren Zeichen gleich sind. String 1 ist kleiner als String 2, wenn das erste verschiedene Zeichen in String 1 einen niedrigeren ASCII-Code hat als das entsprechende Zeichen in String 2.

Sind zwei Strings bis zum Ende des einen String gleich, ein String aber länger, so ist der längere String der größere.

2.6.3 Logische Operatoren

Logische Operatoren erlauben die Verknüpfung von mehreren Vergleichen oder die Verknüpfung auf Bit-Ebene mit Boole'schen Operationen. Sie liefern eine Bitfolge als Ergebnis, in der jedes Bit entweder den Wert "wahr" oder "falsch" darstellt.

Logische Operatoren haben unterschiedliche Prioritäten. Die nachfolgende Liste führt die logischen Operatoren in absteigender Priorität auf.

Logische Operatoren in absteigender Priorität, dargestellt in Wahrheitstafeln (1 = wahr; Ø = falsch):

NOT

X	NOT X	
1	Ø	NEGATION
Ø	1	(KOMPLEMENT)

AND

X	Y	X **AND** Y	
1	1	1	
1	Ø	Ø	
Ø	1	Ø	KONJUNKTION
Ø	Ø	Ø	(logisches Produkt)

OR

X	Y	X **OR** Y	
1	1	1	
1	Ø	1	DISJUNKTION
Ø	1	1	IMPLIZITES ODER
Ø	Ø	Ø	(logische Summe)

XOR

X	Y	X **XOR** Y	
1	1	Ø	
1	Ø	1	
Ø	1	1	EXPLIZITES ODER
Ø	Ø	Ø	(symmetrische Differenz)

IMP

X	Y	X **IMP** Y	
1	1	1	
1	Ø	Ø	
Ø	1	1	IMPLIKATION
Ø	Ø	1	(SUBJUNKTION)

EQV

X	Y	X **EQV** Y	
1	1	1	
1	Ø	Ø	
Ø	1	Ø	ÄQUIVALENZ
Ø	Ø	1	(BIJUNKTION)

- Ausführung von Vergleichen

Für die Ausführung von Entscheidungen können mehrere Vergleiche erforderlich sein.

Die Verknüpfungen können folgendermaßen anschaulich gemacht werden:

AND	SOWOHL Bedingung 1 ALS AUCH Bedingung 2
OR	ENTWEDER Bedingung 1 ODER Bedingung 2 (oder beide)
XOR	GENAU Bedingung 1 ODER GENAU Bedingung 1 (also nicht beide)
EQV	Bedingung 1 MUSS SEIN WIE Bedingung 2
IMP	Bedingung 1 WIE Bedingung 2 (oder nur Bedingung 2); Vertauschungsgesetz nicht gültig!

Beispiele:

	Der Ausdruck
A>B **AND** A>C	ist genau dann wahr, wenn sowohl A größer als B ist als auch A größer als C ist.
A>B **OR** A>C	ist dann wahr, wenn entweder nur A größer als B ist oder nur A größer als C ist oder wenn sowohl A größer als B und A größer als C ist.

A>B **XOR** A>C ist genau dann wahr, wenn nur A größer als B ist oder aber wenn nur A größer als C ist. Er ist falsch, wenn beide Bedingungen erfüllt oder nicht erfüllt sind.

A>B **EQV** C>D ist genau dann wahr, wenn entweder A größer als B ist und C größer als D ist, oder wenn A<=B und C<=D ist.

A>B **IMP** C>D Falls A größer als B ist, muß auch C größer als D sein. Die Abfrage ist aber auch dann wahr, wenn nur C größer als D ist, oder wenn beide Bedingungen nicht erfüllt sind.

- Interne Speicherung von Integer-Werten im Arbeitsspeicher

Integer-Daten werden in dualem Format in 2 Bytes = 16 Bit abgespeichert, wobei das Vorzeichen im höchstwertigen - ganz links außen stehenden - Bit festgehalten wird; Ø bedeutet dort + und 1 bedeutet -.

Beispiel:

$69_{(10)} = 2_{(10)}^{6} + 2_{(10)}^{2} + 2_{(10)}^{0} = 0100\ 0101_{(2)}$

Abspeicherung von $+69_{(10)}$ (zwei Byte): 0000 0000 0100 0101

Dies erklärt z.B., warum sich in zwei Bytes nur bis +32767 (interne Darstellung: 0111 1111 1111 1111) darstellen läßt.

Negative Zahlen werden durch Bildung des Komplements der positiven Dualzahl - Ø wird zu 1 und 1 wird zu Ø - und anschliessender Addition einer dualen 1 zum Komplement entwickelt.

Beispiel:

-69 erst $+69_{(10)}$ = 0000 0000 0100 0101$_{(2)}$

dann Komplementbildung: 1111 1111 1011 1010 (Einerkomplement)

dann duale 1 addieren: + 1

1111 1111 1011 1011 (Zweierkomplement)

Dies ist die interne Abspeicherung von -69 als Integerzahl in zwei Byte.

Eine andere Möglichkeit, die Abspeicherung negativer Integer-Zahlen zu entwickeln, ist die folgende:

. positive Zahl in 2 Bytes dual ausdrücken,

. alle Einzelbits von links kommend im Wert umdrehen (0 wird zu 1 und 1 wird zu 0). Wenn die letzte 1 erscheint, wird abgebrochen. Diese und alle folgenden werden nicht umgedreht.

Beispiel:

- 116 erst $+116_{(10)}$ = 0000 0000 0111 0100 → Stop

dann 1111 1111 1000 1100

Dies ist die interne Abspeicherung von -116 als Integerzahl in zwei Byte.

- Ausführung von logischen Operationen

Die Operanden von logischen Operationen müssen numerisch sein und werden stets gemäß den Rundungsregeln in Kapitel 2.5.1.5 in den Typ Integer umgewandelt.

Das Bitmuster der zwei Bytes langen, internen Darstellung des Integerwertes der beiden Operanden wird bitweise entsprechend der Regeln für die Operation verknüpft. Das Ergebnis ist der dem Bitmuster entsprechende numerische Wert.

Beispiele:

63 **AND** 16

$63_{(10)}$ entspricht: ØØØØ ØØØØ ØØ11 1111$_{(2)}$

$16_{(10)}$ entspricht: ØØØØ ØØØØ ØØØ1 ØØØØ$_{(2)}$

63 **AND** 16 : ØØØØ ØØØØ ØØØ1 ØØØØ$_{(2)}$

daher: 63 **AND** 16 liefert $16_{(10)}$

4 **OR** 2

$4_{(10)}$: ØØØØ ØØØØ ØØØØ Ø1ØØ$_{(2)}$

$2_{(10)}$: ØØØØ ØØØØ ØØØØ ØØ1Ø$_{(2)}$

4 **OR** 2: ØØØØ ØØØØ ØØØØ Ø11Ø

daher: 4 **OR** 2 liefert $6_{(10)}$

-1 **OR** -2

$-1_{(10)}$: 1111 1111 1111 1111$_{(2)}$

$-2_{(10)}$: 1111 1111 1111 111Ø$_{(2)}$

-1 **OR** -2:1111 1111 1111 11111$_{(2)}$

daher: -1 **OR** -2 liefert $-1_{(10)}$

NOT X

ist das Komplement von X. Mathematisch ist das -(X+1)

$5_{(10)}$: ØØØØ ØØØØ ØØØØ Ø1Ø1$_{(2)}$

Komplement: 1111 1111 1111 1Ø1Ø$_{(2)}$

daher: **NOT** 5 liefert $-6_{(10)}$

2.6.4 Stringoperatoren

Als einzige zulässige Operation auf Strings ist die Verkettung von Strings zulässig. Diese Operation wird mit dem Zeichen + dargestellt.

Beispiel:

```
10  VORNAME$=  "MAXL"
20   ZUNAME$=  "HUBER"
30  GESNAME$=  VORNAME$ + " " + ZUNAME$
40     PRINT  GESNAME$
RUN
MAXL HUBER
```

In diesem Beispiel müssen zwei Verkettungen durchgeführt werden, um das Blank zwischen Vornamen und Zunamen einzufügen.

Anmerkungen:

Für die Verarbeitung von Strings stehen eine Reihe von Anweisungen und Funktionen zur Verfügung, die sehr komplexe Verarbeitungen von Strings erlauben.

Die Vergleichsoperatoren sind, wie in Abschnitt 2.6.2 beschrieben, auch auf Strings anwendbar.

2.7 Ausdrücke

Jede Operation ist eine Verbindung aus Operator/en und Operand/en und liefert ein Datenelement (Datum) eines bestimmten Typs.

Da innerhalb von vielen Anweisungen nur die Verwendung eines bestimmten Datentyps vorgeschrieben ist, nicht aber seine Darstellungsform, können an solchen Stellen verwendet werden:

- Konstanten
- Variablen
- die Verbindung von Konstanten und Variablen durch Operationen
- Standard- oder definierte Funktionen.

2.7.1 Numerische Ausdrücke

Numerische Ausdrücke sind numerische Konstanten oder numerische Variablen oder das Ergebnis von numerischen Funktionen oder Vergleichsausdrücke bzw. logische Ausdrücke (vgl. Kapitel 2.7.3) oder die Verknüpfung solcher numerischer Werte durch arithmetische Operationen.

Prioritätsregeln können durch die Verwendung von Klammern verändert werden.

Beispiele:

```
-1
LAENGE# + BREITE#
SQR(UMFANG)
HOEHE!
(3*A+C*D)/(SIN(A%*B)^SQR(A%)-D)
RADIUS^2*3.141592#
```

Anmerkung:

Bei der Berechnung numerischer Ausdrücke von numerischen Werten verschiedenen Typs sind Typenumwandlungen erforderlich.

Dabei können erhebliche Signifikanzverluste auftreten. Es werden alle Operanden in die höchste vorkommende Genauigkeitsstufe umgewandelt. Dadurch werden in der Operation auch nicht signifikante Ziffern verarbeitet, die das Ergebnis verfälschen können.

Beispiel:

Um den Inhalt der Variablen A! (Typ: einfache Genauigkeit) mit dem konstanten Wert 1.Ø auf Gleichheit zu testen, ist folgende Formulierung nötig:

```
IF  ABS(A!-1.Ø)<1.ØE-6 THEN.....
```

2.7.2 Stringausdrücke

Für Stringausdrücke können verwendet werden:

- Stringkonstanten
- Stringvariablen
- Standard- oder definierte Funktionen, die als Ergebnis einen String liefern
- Die Verbindung von solchen Elementen derart, daß als Ergebnis ein String entsteht.

Beispiele:

Stringausdrücke sind:

```
"MAXL"
FRAGE$
""                          (der Leerstring)
ZEICHENSATZ$+CHR$(I%)
VORNAME$+" "+ZUNAME$
```

2.7.3 Vergleichsausdrücke und logische Ausdrücke

Vergleichsausdrücke sind:

1. Numerischer Ausdruck Vergleichsoperator numerischer Ausdruck
2. Stringausdruck Vergleichsoperator Stringausdruck,

wobei Vergleichsoperatoren die in Abschnitt 2.6.2 definierte Operatoren sind.

Vergleichsausdrücke liefern als Ergebnis den Wahrheitswert -1 (="wahr"), wenn die Vergleichsbedingung erfüllt ist. Sie liefern den Wahrheitswert Ø (="falsch"), wenn die Vergleichsbedingung nicht erfüllt ist. Dieser Wahrheitswert wird als Integer dargestellt und kann somit als numerischer Wert behandelt werden.

Beispiele:

```
VORNAME$=""
ZUNAME$="HANS"
R$<CHR$(I%)
MID$(ADRESSE$,I%,L!)>=STRASSE$+NUMMER$
```

Das Gleichheitssymbol = steht hier nicht für Zuweisung, sondern bedeutet: Führe einen Vergleich durch und errechne den Wahrheitswert.

Solche Vergleiche (bzw. deren Wahrheitswert) können, ähnlich wie Funktionen, nicht allein stehen. Es muß in einer Anweisung festgelegt werden, was mit dem Ergebnis (dem Wahrheitswert der Vergleichsoperation) geschehen soll.

Beispiele:

```
11 INPUT B
12 PRINT TAB(-4Ø*(B>Ø)+5);B
```

Vergleichsausdrücke können zu logischen Ausdrücken verbunden werden.

Logische Ausdrücke sind Vergleichsausdrücke oder die Verknüpfung von Vergleichsausdrücken durch logische Operatoren (siehe Abschnitt 2.6.3). Da logische Operatoren Prioritätsregeln genügen, ist die Verwendung von Klammern zur Änderung der Reihenfolge zulässig.

Beispiele:

```
MONAT$="DEZEMBER" AND TAG%=24
A=1Ø AND (B>D  OR B<E) AND B<= Ø
```

Solche Ausdrücke können wiederum nur in Anweisungen eingekleidet vorkommen.

Die zweite Form von logischen Ausdrücken wird durch direkte Verknüpfung von numerischen Ausdrücken mit Hilfe von logischen Operatoren gebildet.

Beispiele:

```
A% AND B% OR C%
(A=B) OR NOT X%
```

Logische Ausdrücke liefern als Ergebnis immer einen Integer-Wert. Ist dieser Wert nicht Ø, wird bei Verzweigungen der Fall "Bedingung erfüllt" unterstellt.

Die Kombinationen solcher Werte können zur Steuerung von Abläufen im Programm verwendet werden. Wegen der Darstellung als Integer ergeben sich durch solche Operationen auf Bitebene numerische Werte, die somit auch in numerischen Ausdrücken verwendet werden können.

Beispiel:

```
10 INPUT "Bitte Zahl eingeben ";ZAHL%
20 PRINT ZAHL%;(ZAHL%)0);(ZAHL%(0);(ZAHL%=0)
30 IF ZAHL%=0 THEN END ELSE 10
RUN
 5 -1  0  0
-2  0 -1  0
 0  0  0 -1
```

```
100 CLEAR:OPTION BASE 0:DIM T$(1)
200 T$(0)="falsch":T$(1)="richtig"
300 LINE INPUT "Bitte 1 Buchstaben eingeben (a-e) ";B$
350 IF LEN(B$))1 THEN PRINT T$(0);:GOTO 800
400 PRINT T$(1+  ( (B$("a") OR (B$)"e") )    );
800 PRINT "e Eingabe":END
```

```
2534 REM  Verwandlung von Groß- in Kleinbuchstaben
2536 T$=INPUT$(1):IF T$="ß" THEN T$="SS":GOTO 2550
2538 IF T$)="a" AND T$(="Ü" THEN T$=CHR$(-33 IMP ASC(T$))
2550 PRINT T$;:GOTO 2536
```

3. ARBEITSWEISE DES INTERPRETERS

Das Betriebssystem des M20 enthält neben seinem Befehlsvorrat zur Ausführung verschiedener Operationen auf System-, Disketten- und Fileebene (PCOS-Befehle) einen BASIC-Interpreter zur Verwaltung und Ausführung von Programmen. Dem Anwender stehen somit PCOS-Befehle und - bei aktiviertem BASIC-Interpreter - sowohl BASIC-Befehle als auch BASIC-Anweisungen zur Verfügung.

Der BASIC-Interpreter kann in verschiedenen Betriebszuständen arbeiten, die nachfolgend einzeln beschrieben werden:

- Der Command- bzw. Direkt-Mode erlaubt die Eingabe von Programmzeilen und die Eingabe von unmittelbar auszuführenden BASIC-Befehlen und -Anweisungen.
- Der Execute-Mode
  wird bei der Ausführung von Programmen und von direkt auszuführenden Zeilen oder BASIC-Befehlen erreicht.
- Der Edit-Mode
  wird für die Korrektur von Programmzeilen aufgerufen.

3.1 Eingabe von BASIC-Befehlen

BASIC-Befehle werden hauptsächlich für die Eingabe, Korrektur, Dokumentation und Speicherung von Programmen angeboten.

Der Betriebszustand des Interpreters, der die Eingabe (und unmittelbare Ausführung) von Befehlen zuläßt, heißt Command-Mode bzw. Direkt-Mode.

3.1.1 Übergang in den Command-Mode

Der BASIC-Command-Mode wird erreicht,

- falls sich das Betriebssystem in der PCOS-Ebene befindet und der (PCOS)-Befehl **ba** eingegeben wird;
- falls während der Ausführung eines BASIC-Programmes (oder einer direkt auszuführenden Zeile) **C** eingegeben wird;
- falls bei der Korrektur einer Programmzeile (Edit-Mode) eine der nachfolgende Edit-Subbefehle eingegeben wird: **E** (Exit), **Q** (Quit Editing) oder eine Eingabeschlußtaste (CR).

Der BASIC-Command-Mode wird selbständig aktiviert, falls die Ausführung eines Programms, eines BASIC-Befehls oder einer direkt auszuführenden Zeile beendet wird bzw. falls ein Fehler erkannt wird und keine Fehlerbehandlungsroutine aktiv ist (Ausnahmen: Syntax error und gewisse Fälle des Overflows).

Beim Übergang aus einem beliebigen anderen Betriebszustand in den BASIC-Command-Mode erfolgt im allgemeinen die Meldung "Ok".

3.1.2 Eingaben im Command-Mode

Der BASIC-Command-Mode erlaubt die Eingabe von BASIC-Befehlen, direkt auszuführenden Zeilen und von Programmzeilen.

Der Command-Mode und der Direkt-Mode haben keinen Einfluß auf ein im Arbeitsspeicher befindliches Programm. Eingaben von Programmzeilen im Command-Mode löschen jedoch sofort den gesamten Datenbereich im Arbeitsspeicher sowie alle Stack-Speicher.

Jede Eingabe wird im BASIC-Command-Mode erst verarbeitet, wenn sie mit einer Eingabeabschlußtste (CR) beendet wurde.

Bei jeder Eingabe werden führende Blanks ignoriert. Die Interpretation richtet sich nach dem ersten eingegebenen Zeichen, das von Blank verschieden ist.

- Ist das erste von Blank verschiedene Zeichen eine Ziffer, wird die Eingabe als Programmzeile behandelt und im Arbeitsspeicher abgelegt. Dabei werden alle Daten im Datenbereich des Arbeitsspeichers sofort gelöscht.
- Ist das erste von Blank verschiedene Zeichen keine Ziffer, wird die Zeile als direkt auszuführend interpretiert. Dabei werden die Daten im Datenbereich des Arbeitsspeichers nicht gelöscht.

Für die Erstellung eines Programms werden somit die einzelnen Programmzeilen, mit einer Zeilennummer beginnend, eingegeben. Diese Zeilen werden nach dem Drücken von CR übernommen und im Arbeitsspeicher abgelegt. Die weitere Behandlung dieser Programmzeilen wird durch BASIC-Befehle bestimmt (so kann z.B. durch Eingabe von **RUN** die Ausführung dieser Zeilen verlangt werden).

Ist einer Eingabezeile keine Zeilennummer vorangestellt, wird sie unmittelbar nach dem Drücken von CR ausgeführt. Neben den BASIC-Befehlen können auch die meisten Anweisungen direkt ausgeführt werden. Die Unterscheidung wird in Abschnitt 3.2 erläutert.

Korrektur von Eingaben im Command-Mode

Eingabezeilen können vor dem Drücken der Abschlußtaste (CR) korrigiert werden.

Durch (wiederholtes) Drücken von H können die Zeichen von der aktuellen Cursorposition bis zur ersten fehlerhaften Stelle in der Zeile gelöscht werden. Es ist ab dieser Stelle die Eingabe richtig fortzusetzen.

3.1.3 Verlassen des Command-Modes

Der Command-Mode wird in vielen Fällen automatisch durch die Ausführung des gewünschten Befehls verlassen (z.B. bei Eingabe von **RUN** zur Ausführung eines Programms).

In folgenden Fällen wird der Command-Mode explizit verlassen:

SYSTEM als Befehl bewirkt den Übergang in die PCOS-Ebene unter Löschen des Speichers (**SYSTEM** kann auch im Programm verwendet werden).

RESET löscht den Inhalt des Speichers und bewirkt nachfolgend das Neuladen des Betriebssystems.

EDIT als Befehl dient zum Aufruf des Edit-Modes zur Korrektur von Programmzeilen (siehe Abschnitt 3.4).

3.2 Direkt-Mode

Der BASIC-Interpreter kann neben Befehlen auch die meisten Anweisungen direkt ausführen. Umgekehrt können die meisten Befehle auch innerhalb von Programmen vorkommen und sinnvoll angewendet werden.

Befehle, die unmittelbar zum Umgang mit Programmen (z.B. **LIST, LLIST**) vorgesehen sind, werden zwar innerhalb von Programmen ausgeführt, bewirken jedoch danach einen automatischen Übergang in den BASIC-Command-Mode bzw. Direkt-Mode.

Kommt daher ein solcher Befehl in einem Programm vor, wird die Programmausführung nach der Ausführung dieses Befehls abgebrochen. Andere Befehle, wie z.B. **KILL, TRON, TROFF,** können sinnvoll innerhalb von Programmen verwendet werden.

BASIC-Anweisungen, die ohne vorangehende Zeilennumer eingegeben werden, werden als direkt auszuführend interpretiert. Dadurch ist eine sehr komfortable Möglichkeit verfügbar, kleinere Aufgaben direkt zu lösen oder Programme sehr wirkungsvoll zu testen und zu korrigieren.

Einige Gruppen von Anweisungen sind nicht im Direkt-Mode ausführbar und haben entweder keine Wirkung oder führen zu einer Fehlermeldung. Das sind z.B.:

- **DATA**
- **DEF FN**

Sinnvoll im Direkt-Mode anwendbar sind jene Anweisungen, die zu direkten Ergebnissen (z.B. von Berechnungen) führen oder die Aufschluß über den Zustand von Variablen eines aktuellen Programms bieten bzw. diese zu Testzwecken verändern (z.B. **PRINT, ?, LPRINT, LET** usw.).

3.3 Erstellen eines Programms

Wie bereits in Kapitel 2 beschrieben, besteht ein Programm aus einer Folge von Programmzeilen, die mit einer Zeilennummer beginnen und eine oder mehrere durch : getrennte Anweisungen enthalten.

Aus Abschnitt 3.1 ist bekannt, daß im Command-Mode des BASIC-Interpreters eingegebene Zeilen, die mit einer Ziffer beginnen, als Programmzeilen interpretiert werden.

Dieser Abschnitt befaßt sich jetzt mit den Bedienungselementen, die zur Eingabe eines neuen Programms notwendig bzw. möglich sind.

3.3.1 Vorbereiten des Arbeitsspeichers

Um ein neues Programm eingeben zu können, muß der Arbeitsspeicher mit dem Befehl **NEW** gelöscht werden. Es ist dadurch gewährleistet, daß eventuell im Arbeitsspeicher enthaltene Programmzeilen oder Daten von früheren Arbeiten gelöscht sind.

Nach Ausführung von **NEW** kann unmittelbar mit der Eingabe der ersten Programmzeile begonnen werden. Zusätzlich bietet der Interpreter die Möglichkeit der selbständigen Vergabe der Zeilennummern mit dem Befehl **AUTO**.

3.3.2 Eingabe der Programmzeilen

Die Programmzeilen werden entsprechend der BASIC-Regeln eingegeben.

Sobald eine der Abschlußtasten CR gedrückt wird, wird die Zeile entsprechend ihrer Zeilennummern den bereits bestehenden Programmzeilen hinzugefügt und dabei eingeordnet, wenn nötig.

Wird eine Zeilennummer doppelt vergeben, wird die bestehende Zeile durch die neue Programmzeile vollständig ersetzt. Erzeugt der Befehl **AUTO** eine bereits bestehende Zeilennummer, wird nach der Zeilennummer ein * (Sternchen) angezeigt, um den Anwender auf die bereits bestehende Zeile hinzuweisen. Wird unmittelbar CR gedrückt, bleibt die alte Zeile erhalten; erfolgen Eingaben, wird die alte Programmzeile wie gewohnt durch die neue Programmzeile ersetzt. Die automatische Zeilennumerierung ist durch [C] zu beenden.

Die Korrektur von Programmzeilen während der Eingabe kann vor dem Drücken der Abschlußtaste CR mit [H] erfolgen.

Unter Zuhilfenahme von [H] sind alle Zeichen bis zur fehlerhaften Stelle zu löschen und anschließend die Zeile korrekt einzugeben.

Wurde die Abschlußtaste CR bereits gedrückt, kann die Korrektur auf zwei Arten erfolgen:

- durch erneute Eingabe der Zeile mit der gleichen Zeilennummer,
- durch die Möglichkeiten des Edit-Modes (siehe Abschnitt 3.4)

3.3.3 Möglichkeiten nach dem Ende der Eingabe

Die Programmzeilen werden während der Eingabe im Arbeitsspeicher abgelegt. Sind alle Programmzeilen eingegeben, muß durch Befehle bestimmt werden, was mit dem soeben erstellen Programm geschehen soll.

Jede im BASIC-Command-Mode zulässige Eingabe kann jetzt durchgeführt werden.

Es ist jedoch zu empfehlen, das Programm mit dem Befehl **SAVE** zuvor auf eine Diskette zu speichern und mit **LLIST** einen Ausdruck des Programms auf dem Drucker zu erstellen.

3.4 Korrektur bestehender Programme

Zur Korrektur eines bestehend Programms muß dieses Programm im Arbeitsspeicher vorhanden sein. Das ist der Fall, wenn das Programm unmittelbar vorher eingegeben oder z.B. mit **RUN** ausgeführt wurde. Ist das Programm noch nicht im Arbeitsspeicher, muß es mit **LOAD** von der Diskette geladen werden.

Wurden Programme mit **SAVE** und Parameter **P** geschützt gespeichert, ist eine nachträgliche Korrektur nicht möglich. Alle Befehle, die eine Änderung des bestehenden Programms oder Einsicht in das Programm bewirken, führen zu einer Fehlermeldung. Es handelt sich um die folgenden Befehle:

LIST, LLIST, DELETE, EDIT, RENUM,

sowie das Einfügen von neuen Zeilen.

3.4.1 Einfügen zusätzlicher Programmzeilen

Zusätzliche Programmzeilen können durch einfache Eingabe dieser Zeilen mit den entsprechenden Zeilennumern eingegeben werden. Diese Zeilen werden entsprechend einsortiert oder angefügt.

3.4.2 Ersetzen von Programmzeilen

Soll eine bestehende Programmzeile vollständig durch eine andere ersetzt werden, ist die neue Programmzeile mit der alten Zeilennummer einzugeben.

3.4.3 Löschen von Programmzeilen

Eine oder mehrere Programmzeilen können mit dem Befehl **DELETE** gelöscht werden. Eine Programmzeile wird auch gelöscht, wenn man nur die Zeilennumer eingibt und sofort CR drückt.

3.4.4 Ändern bestehender Programmzeilen (Edit-Mode)

Für die Korrektur bestehender Programmzeilen stellt der BASIC-Interpreter einen eigenen Betriebszustand, den Edit-Mode, zur Verfügung.

Der Edit-Mode wird auf zwei Arten erreicht:

1. Durch Anwendung des Befehls **EDIT**
2. Falls bei der Ausführung eines Programms ein Syntax-Fehler erkannt wird, wird dieser Fehler gemeldet und automatisch der Edit-Mode für die fehlerhafte Zeile aufgerufen.

Innerhalb des Edit-Modes stehen eine Reihe von Edit-Subbefehlen zur Verfügung, die nachfolgend tabellarisch angeführt und anhand zweier Beispiele erläutert werden.

Diese Subbefehle sind ohne CR einzugeben, wenn Sie nicht gerade in einer Einfüge-Phase (Subbefehle **I, X** oder **H**) oder Auswechselphase (Subbefehl **C**) sind und werden nicht angezeigt. Steht vor einem Subbefehl ein n, kann eine Zahl zwischen 1 und 255 eingegeben werden, die eine n-malige Ausführung des direkt danach getippten Subbefehls bewirkt.

Edit-Befehl	Bedeutung
EDIT nnnnn	Es wird die Programmzeile mit der Nummer nnnnn aufgerufen und die Zeilennumer im Bildschirm angezeigt. Der Cursor befindet sich an der ersten Stelle nach der Zeilennumer. Die Zeile wird nicht angezeigt.
EDIT .	Der Befehl **EDIT** bezieht sich auf die aktuelle Zeilennummer, das ist z.B. die zuletzt eingegebene oder ausgeführte Zeile.

Edit-Subbefehl	Bedeutung
L	Die Zeile wird im Display angezeigt. In der nächsten Zeile erscheint wieder die Zeilennummer und der Cursor an der ersten Position der Zeile.
A	Die laufende Edit-Operation wird unterbrochen und es bleibt die ursprüngliche Zeile erhalten. Die Zeilennummer wird erneut angezeigt und der Cursor befindet sich an der ersten Stelle der Zeile.
nBlank	der Cursor wird um n Stellen nach rechts bewegt und das entsprechende Zeichen wird angezeigt.

H	Der Cursor wird um eine Stelle nach links (zurück) bewegt. Das vorherige Zeichen wird jedoch nicht gelöscht! Ausnahme: H wird bei Gültigkeit des Subbefehls I oder H oder X verwendet.
I	ab der gegenwärtigen Cursorposition sollen Zeichen eingefügt werden. Die Zeichen sind nach I einzugeben und entweder mit CR oder HOME 5 abzuschließen. Die eingefügten Zeichen werden angezeigt und der Cursor befindet sich an der ersten Stelle nach dem letzten eingefügten Zeichen. H wirkt hier löschend.
X	Erlaubt das Anfügen von Zeichen an das Ende einer bestehenden Zeile. Wird X eingegeben, wird die bestehende Zeile angezeigt. Der Cursor ist an der ersten freien Stelle und es können weitere Zeichen eingegeben (wie nach dem Subbefehl I) oder mit H die letzten Zeichen gelöscht werden.
HOME 5	Bewirkt das Ende des Einfügens, der Edit-Mode bleibt jedoch erhalten. Wird CR gedrückt, wird zwar ebenfalls das Ein- (bzw. An-)fügen beendet, jedoch auch der Edit-Mode verlasen.

nD löscht ab der Cursorposition die nächsten n Zeichen. Die gelöschten Zeichen werden zwischen "Ö" angezeigt, der Cursor steht an der ersten Stelle nach dem letzten gelöschten Zeichen. Ist n größer als die Anzahl der rechts vom Cursor stehenden Zeichen, wird der Rest der Zeile ab der Cursor-Position gelöscht.

H löscht den Rest der Zeile ab der Cursorposition und erlaubt anschließend das Eingeben von Zeichen wie nach **I**.

nSZeichen Dieser Subbefehl sucht das n-te Auftreten des nach **S** angegebenen Zeichens (Groß- und Kleinbuchstaben sind für 'Zeichen' zu unterscheiden).
Das Suchen beginnt an der ersten Position rechts vom Cursor. Wird das Zeichen gefunden, wird der Cursor unmittelbar vor dem gefundenen Zeichen angezeigt. Wird das Zeichen nicht gefunden, steht der Cursor am Ende der Zeile. Alle während des Suchens durchlaufenen Zeichen werden angezeigt.

nKZeichen — Dieser Subbefehl ist dem Subbefehl **S** ähnlich, mit dem Unterschied, daß alle während des Suchens durchlaufenen Zeichen gelöscht werden (Groß- und Kleinbuchstaben sind für 'Zeichen' zu unterscheiden). Die gelöschten Zeichen werden zwischen "Ö" angezeigt. Der Cursor steht an der Stelle unmittelbar vor dem im Subbefehl angegebenen Zeichen.

nCZeichen — Mit **C** kann das nächste Zeichen durch das eingegebene Zeichen ersetzt werden. Wird n**C** gewählt, so können ab der Cursorposition die nächsten n Zeichen durch die eingegebenen Zeichen ersetzt werden. Erst wenn genau n Zeichen ausgewechselt wurden, wird automatisch wieder der Edit-Mode angewählt. Es müssen immer n Zeichen zum Auswechseln eingetippt werden.

CR — Das Drücken der Abschlußtaste bwirkt die Anzeige der modifizierten Zeile und die Rückkehr in den Command-Mode.

E — Bewirkt ebenfalls die Rückkehr in den Command-Mode, die modifizierte Zeile wird jedoch nicht angezeigt.

Q — Die Eingabe von **Q** bewirkt die Rückkehr in den Command-Mode, es bleibt jedoch der ursprüngliche Inhalt der Zeile gespeichert und alle durchgeführten Änderungen bleiben unberücksichtigt! Auch der Datenbereich im Arbeitsspeicher bleibt ungelöscht.

Anmerkungen:

Im Edit-Mode werden Zeichen unmittelbar übernommen. Normalerweise ist also die Abschlußtasten nicht zu drücken. (Durch Drücken der Abschlußtaste wird der Edit-Mode verlassen.)

Bei Verwendung des Edit-Modes zum Ändern einer Zeile werden alle noch bestehenden Variableninhalte gelöscht. Sollen also im Zuge des Testen eines Programms noch Variableninhalte geprüft werden (z.B. weil der Edit-Mode automatisch nach einem Syntaxfehler aufgerufen wird), ist unbedingt der Subbefehl **Q** einzugeben. Dann wird der Edit-Mode verlassen und die Inhalte der Variablen können überprüft werden. Jeder, eine Zeile verändernde, Subbefehl löscht die Inhalte der Variablen (CR gilt immer als ändernd)!

Mit dem Parameter **P** des BASIC-Befehls **SAVE** geschützte Programme können nicht verändert werden. Die Anwahl des Edit-Modes bewirkt die Meldung "Illegal function call".

Beispiele:

1. Die Zeile 5ØØ **FOR** I=1 **TO** 15 **STEP** 2

   soll korrigiert werden auf:

   5ØØ **FOR** I=2 **TO** 16 **STEP** 2

Schritt	Eingabe	Anzeige
1	**EDIT** 5ØØ CR	500 ■
2	**L**	500 FOR I=1 TO 15 STEP 2
		500 ■
3	Blank (6 mal)	500 FOR I=■
4	**C2**	500 FOR I=2■
5	**5** Blank	500 FOR I=2 TO 1■
6	**C6**	500 FOR I=2 TO 16■
7	CR	500 FOR I=2 TO 16 STEP 2
		■

2. Die Zeile 510 LET A(I)=I*SIN(X)

soll geändert werden auf:

510 **LET** A(I)=I***COS(**X**):PRINT** A(I)

Schritt	Eingabe	Anzeige
1	**EDIT** 51Ø CR	51Ø ■
2	L	51Ø LET A(I)=I*SIN(X)
		51Ø ■
3	SS	51Ø LET A(I)=I*■
4	3CCOS	51Ø LET A(I)=I*COS■
5	**X:**PRINT A(I) CR	51Ø LET A(I)=I*COS(X):PRINT A(I)
		■

Eine weitere, sehr komfortable Möglichkeit, ganze Programme am gesamten Bildschirm zu editieren, bietet der Full-Sreen-Editor. Er kann nur angewendet werden, wenn das zu editierende Programm im ASCII-Format abgespeichert wurde (**SAVE** Filename,**A**) und die Programmzeilen (incl. Zeilennummer) nicht länger als 80 Zeichen sind.

Der Full-Screen-Editor wird mit Hilfe des PCOS-Befehls **ed** (bei BASIC-Programmen mit folgendem Parameter **%c**) aufgerufen.

3.5 Ausführen von Programmen

3.5.1 Programmstart

Für die Ausführung von Programmen stehen mehrere BASIC-Befehle und -Anweisungen zur Verfügung.

1. Das Programm befindet sich im Arbeitsspeicher.

   Ist das Programm bereits im Arbeitsspeicher, weil es z.B. unmittelbar zuvor eingegeben wurde oder zu Edit-Zwecken geladen war, wird die Ausführung mit **RUN** gestartet.

   RUN ohne Parameter bewirkt die Ausführung des im Arbeitsspeicher befindlichen Programms, beginnend bei der niedrigsten Zeilennumer.

   Soll das Programm ab einer bestimmten Zeile ausgeführt werden, gilt:

   RUN nnnnn startet die Ausführung bei der Zeilennumer nnnnn. Selbstverständlich muß nnnnn so gewählt werden, daß eine sinnvolle Programmausführung ab dieser Stelle möglich ist.

2. Das Programm befindet sich als File auf der Diskette.

   In diesem Fall stehen folgende Möglichkeiten zur Verfügung:

   a) **RUN** Filename
   b) **LOAD** Filename,**R**
   c) **ba** Filename, falls das System in der PCOS-Ebene ist ('Filename' ist hier nicht in Anführungszeichen einzugeben!)
   d) Start des Programms als INIT.BAS
   e) Aufruf des Programms durch ein anderes Programm, das die Anweisung **CHAIN** verwendet.

   'Filename' ist dabei ein nach den PCOS-Regeln aufgebauter Filename.

Beispiele für Programmnamen:

"Ø:EINGABE"
"1:PROGRAMM25"
"DISKA:ENDE/PASSWORD"

zu a) **RUN** Filename

Das Programm 'Filename' wird auf der angegebenen Diskette gesucht, in den Arbeitsspeicher geladen und mit der Ausführung bei der niedrigsten Zeilennummer begonnen.

Wird zusätzlich der Parameter **R** angegeben (**RUN** Filename, **R**) bleiben alle Datenfiles offen, die zum Zeitpunkt des **RUN**-Befehls offen sind.

zu b) **LOAD** Filename,R

LOAD mit Parameter R ist vollkommen gleichbedeutend mit **RUN**.

Anmerkung:

LOAD alleine bewirkt das Laden des Programms ohne Start der Ausführung.

zu c) **ba** Filename

Befindet sich der M20 in der PCOS-Ebene, so kann unmittelbar die Ausführung eines Programms mit **ba** Filename gestartet werden. (Der Filename ist hier nicht in Anführungszeichen eingeschlossen anzugeben, da es sich um einen PCOS-Befehl handelt.) Es wird dann sofort in die BASIC-Ebene gewechselt, das Programm auf Diskette gesucht und mit seiner Ausführung begonnen.

zu d) Start als INIT.BAS

Der M20 sucht nach dem Laden des Betriebssystems automatisch nach dem Programm namens INIT.BAS. Ist ein BASIC-Programm mit diesem Namen auf Diskette vorhanden, wird ohne weitere Bedienungseingriffe mit der Ausführung dieses Programms begonnen. INIT.BAS kann selbstverständlich auch mit einer der unter a) - c) angegebenen Möglichkeiten gestartet werden.

zu e) Programmaufruf aus anderen Programmen heraus

Ein Programm kann durch Anweisungen selbst andere Programme aufrufen. Es wird dann das Programm im Arbeitsspeicher durch das aufrufende Programm ersetzt und mit der Ausführung dieses Programm begonnen (vgl. Anweisung **CHAIN**).

Anmerkung:

Für die erfolgreiche Ausführung eines Programms müssen bestimmte Voraussetzungen hinsichtlich der Systemvorbereitung (z.B. **sb, ss, sf** in der PCOS-Ebene) und des Zustands des Arbeitsspeichers gewährleistet sein (vgl. **CLEAR-, COMMON-** und **CHAIN**-Anweisungen). Führt der Start eines Programms, dessen Funktionsfähigkeit bereits bekannt ist oder vorausgesetzt werden kann, zu Fehlermeldungen, so ist zu überprüfen, ob die Ursache in solchen fehlerhaften Voraussetzungen liegen kann (z.B. das vorangehende Programm wurde vorzeitig beendet oder es ist unbedingt die reguläre Ausführung vorangehender Programme nötig).

3.5.2 Unterbrechen der Programmausführung

Üblicherweise sollte der vorgesehene Programmablauf eingehalten werden.

Besonders zum Testen von neu erstellten Programmen ist jedoch die abschnittweise Ausführung von Programmen oder die Unterbrechung der Ausführung sinnvoll und notwendig.

Durch Eingabe von

C kann die Programmausführung an der aktuellen Stelle unterbrochen werden. Das System meldet "Ok". Die Datenfiles bleiben offen, die Inhalte der Variablen unverändert; der Tastaturpuffer wird geleert.Es können alle Befehle und direkten Anweisungen ausgeführt werden, die Variablen haben ihren aktuellen Wert. Variablenwerte können somit mit **PRINT** Variablenname oder **?** Variablenname abgefragt oder z.B. mit **LET** verändert werden.
Die Ausführung kann aber an der Stelle der Unterbrechung durch Eingabe von **CONT** fortgesetzt werden.

Durch Eingabe von

S wird die Ausführung einer Bildschirmanzeige (z.B. eines Listings) unterbrochen, es wird jedoch nicht in den Command-Mode verzweigt. S wirkt nicht bei internen Rechenvorgängen oder Ausgabe von Graphik.
Das Drücken einer beliebigen Taste bewirkt die Fortsetzung des Programms.

3.5.3 Testen und Korrektur von Programmen

Für den Test von Programmen stehen verschiedene Möglichkeiten zur abschnittweisen Ausführung zur Verfügung.

1. Unterbrechnung des Programmablaufs durch den Bediener.

   Durch Eingabe von C während der Ausführung kann an der aktuellen Stelle unterbrochen werden.

Es stehen alle Möglichkeiten des Direkt-Modes zur Verfügung. Die Ausführung kann mit **CONT** fortgesetzt werden.

2. Unterbrechungen aus den Programmen heraus

   Soll an bestimmten Stellen unterbrochen werden, so sind an diesen Stellen **STOP**-Anweisungen im Programm aufzunehmen. Die Ausführung wird unterbrochen. Es stehen alle Möglichkeiten des Direkt-Modes zur Verfügung.

   Die Ausführung kann mit **CONT** mit der auf **STOP** folgenden Anweisung fortgesetzt werden.

3. Überprüfung des Programmweges

   Der aufgrund von Schleifen, Bedingungen oder Kontrollanweisungen aufgeführte Ablauf in der Ausführung kann durch Einsatz der Befehle **TRON** und **TROFF** verfolgt werden. **TRON** und **TROFF** können auch im Programm verwendet werden.

4. Fortsetzung an beliebigen Stellen

   Mit **CONT** kann nach **C** oder **STOP** ab der Stelle der Unterbrechung fortgesetzt werden.

   Soll mit einer anderen Anweisung fortgesetzt werden, kann mit **RUN** Zeilennummer oder mit der direkt auszuführenden Anweisung **GOTO** Zeilenr. angegeben werden, ab welcher Anweisung die Ausführung wieder gestartet wird.

5. Fortsetzung nach Fehlermeldungen

   Werden Syntaxfehler gemeldet, erfolgt automatisch der Übergang in den Edit-Mode.

Editieren bewirkt das Löschen aller Variablenwerte. Sollen also nach Auftreten des Syntaxfehlers Variablewenwerte überprüft werden, so ist der Edit-Mode mit **Q** (Quit) zu verlassen. Das Drücken jeder anderen Taste bewirkt das Löschen der Variablen!

Bei der Fortsetzung nach solchen Fehlern ist zu beachten, daß eventuell die Variablenwerte nicht mehr zur Verfügung stehen.

Bei anderen Fehlermeldungen ist nach der jeweiligen Situation zu entscheiden.

Einige Meldungen lassen die Fortsetzung mit **CONT** zu, andere erfordern vor der Fortsetzung die Behebung der Fehlerursache (siehe Meldungen des Interpreters; Kapitel 6).

3.5.4 Ende einer Programmausführung

Das reguläre Ende einer Programmausführung wird mit Ausführung der Anweisung **END** erreicht.

Es werden alle offenen Files geschlossen und in den BASIC-Command-Mode verzweigt. Die Meldung "Ok" wird angezeigt.

Nach der Ausführung von **END** stehen die Variablenwerte weiterhin zur Verfügung und können im Direkt-Mode verarbeitet werden.

Hinweis:

Jedes andere Programmende außer der Ausführung von **END** bewirkt, daß Datenfiles offen bleiben, es sei denn, diese wurden mit einer **CLOSE**-Anweisung geschlossen.

3.5.5 Verbindung mehrerer Programme

Das M20-BASIC erlaubt die Verbindung von Programmen auf mehrere Arten:

1. Mit **RUN** oder **LOAD**
2. Mit **CHAIN** oder **CHAIN MERGE**

Der Hauptunterschied dieser Möglichkeiten besteht in der Behandlung der zu übergebenden DATEN.

1. **RUN** und **LOAD**

   Werden Programme mit **RUN** oder **LOAD** verbunden, werden
   - alle Datenfiles geschlossen (außer, es wurde der Parameter **R** angegeben)
   - grundsätzlich alle numerischen Variablen mit Ø und alle Stringvariablen mit dem Leerstring ("") vorbelegt und alle Feldvereinbarungen für Random-Files aufgehoben.

2. **CHAIN**

   Die Anweisung **CHAIN** erlaubt unterschiedliche Möglichkeiten zur Weitergabe von Daten zwischen einzelnen Programmen:

   CHAIN läßt die Datenfiles offen;

   CHAIN unter Verwendung von einer oder mehreren **COMMON**-Anweisungen erlaubt die Übergabe einzelner Variablen;

   CHAIN mit Parameter **ALL** bewirkt, daß alle Variablen des aufrufenden Programms mit ihren Werten an das aufgerufene Programm übergeben werden.

3. **CHAIN MERGE**

   Mit **RUN** bzw. **CHAIN** können mehrere Programme sequentiell verbunden werden.

CHAIN MERGE erlaubt hingegen, daß innerhalb eines Programms Overlays von Programmteilen gebildet werden.

CHAIN MERGE bewirkt, daß ein als ASCII-File gespeicherter Programmteil in das im Arbeitsspeicher befindliche Programm eingebunden wird. Die Ausführung wird bei der Zeile begonnen, die dabei vorgeschrieben wird.

Alle Daten des aufrufenden Programms bleiben erhalten.

Wird zusätzlich der Parameter **DELETE** angegeben, werden vor dem Laden des Overlays die angegebenen Zeilennumern gelöscht.

4. DIE VERWENDUNG DER SPRACHELEMENTE

4.1 Bildung von Variablennamen, Formatbeschreibung, Syntax

4.1.1 Bildung von Variablennamen

(vgl. auch Abschnitt 2.5.3.2)

Für jeden einzelnen Variablennamen können von 1 bis zu 39 Zeichen verwendet werden. Variablennamen, die sich in mindestens einem Zeichen bzw. dessen Position unterscheiden, sind verschieden. Zulässig sind neben dem Punkt und dem Unterstreichungszeichen alle Groß- und Kleinbuchstaben (außer Ä, Ö, Ü, ä, ü, ö und ß) sowie die Ziffern Ø bis 9.

Das erste Zeichen eines Variablennamens muß ein Buchstabe sein.

Für Variablennamen oder Befehle oder Anweisungen oder Funktionen gilt:

Groß- und Kleinbuchstaben werden vom M20 nicht unterschieden. Werden Kleinbuchstaben eingegeben, ersetzt der M20 diese beim Listen sofort durch die entsprechenden Großbuchstaben. Das Gesagte gilt nicht bei Stringkonstanten (vgl. Abschnitt 2.5.3.1) und Filenamen.

Bei Variablenbezeichnungen können zusätzlich zum Namen am Ende noch die Zeichen **%, !,#** oder **$** angehängt sein, um den Typ der Variablen zu bestimmen, gegebenenfalls sogar als 40. Zeichen (vgl. Abschnitt 2.5).

Die Zeichen (und) dienen dazu, Elemente von Arrays zu spezifizieren (vgl. Abschnitt 2.5.3.3). Arrays können ebenfalls von jedem Typ sein (z.B. NAMEN$() oder KENN.NR!()).

Variablennamen, die sich nur durch das Typkennzeichen unterscheiden, sind verschieden. Ebenso sind Namen, die durch () als Array gekennzeichnet sind, verschieden von solchen ohne ().

Schlüsselwörter sowie reservierte Variablen (in diesem Handbuch stets **fett** gedruckt) dürfen nicht als Variablennamen, wohl aber als Bestandteil von Variablennamen verwendet werden (vgl. Kapitel 2.5.3.2 und 4.2.2).

4.1.2 Formatbeschreibung; Syntax

Um alle möglichen Fälle der Formulierung eines Befehls oder einer Anweisung möglichst knapp, aber umfassend darstellen zu können, werden bei der FORMAT-Beschreibung die folgenden Hilfssymbole und Darstellungsmittel verwendet:

[,] , { , } , ... ,

Großschreibung, Kleinschreibung, Unterstreichung (___), N (Negativdarstellung) und Fettdruck.

Diese Hilfssymbole sind bei der Eingabe des Befehls, der Anweisung oder der Funktion wegzulassen.

Bedeutung der Symbole

{ } einer der in den Klammern untereinander angeführten Parameter muß verwendet werden (Alternativ-Auswahl)

[ ]	der/die Parameter in den Klammern kann verwendet werden
. . .	der/die Parameter in der zuletztgenannte/n [ ] kann beliebig oft erneut verwendet werden
___	Default-Wert = Wert wird vom System selbständig angesetzt, sofern der Parameter vom Anwender nicht eingegeben wird. Solche Default-Werte stehen immer in [ ].
SPRACHELEMENT (GROSSSCHREIBUNG)	feste Vorgabe als Sprachelement; muß genau so eingegeben werden.
FETTDRUCK	Reservierte Wörter und jeweils erforderliche Sonderzeichen werden **fett** gedruckt.
Begriff (Kleinschreibung)	Platzhalter, ist durch einen aktuellen Parameter zu ersetzen. Dieser Parameter ist eine vom Anwender zu bestimmende Zeichenfolge, die dem 'Begriff' gerecht wird.
X (Negativdarst.)	Betätigen der Taste, die das in Negativdarstellung angegebene Zeichen (hier z.B. X) erzeugt, in Verbindung mit der blauen Taste (sog. CONTROL-Taste), aber ohne Eingabeschlußtaste.
, od. () od. / od. #	und andere Sonderzeichen, die in der Syntaxbeschreibung nicht in [ ] stehen, müssen gesetzt werden, auch wenn ein folgender Ausdruck entfällt, aber noch etwas folgt (Platzhalterfunktion). Wenn gar nichts mehr folgt, muß das Zeichen entfallen. () müssen voll gesetzt werden, wenn nötig.

Ø Blank; dieses Symbol wird nur an einigen Stellen ausdrücklich gesetzt, um die benötigte Anzahl Leertasten sofort anzuzeigen. Im allgemeinen ist diese jedoch durch Abzählen der an der entsprechenden Position der Folgezeile stehenden Buchstaben zu ermitteln.

ACHTUNG:

Die Eingabe von Sprachelementen ist nicht "formfrei". Das heißt, daß Sprachelemente und Parameter nicht einfach hintereinander eingegeben und trennende Blanks dabei weggelassen werden können. Schlüsselwörter müssen durch Blanks von sonstigen Parametern getrennt sein.

Beispiele:

1. Die Eingabe
   30 **PRINT**A,B(1) führt zur Fehlermeldung "Syntax error". Es muß
   30 **PRINT** A,B(1) eingegeben werden.

2. 37521**GOSUB**6500:C=**ASC(**Q$**)** führt auf die Fehlermeldung "Syntax error".
   Es muß
   37521**GOSUB** 6500:C=**ASC(**Q$**)** eingegeben werden.

Von besonderer Bedeutung bei der Syntaxbeschreibung (FORMAT-Beschreibung der Befehle, Anweisungen und Funktionen) ist, auf die angegebene Art von Parametern zu achten.

So muß z.B. bei

$$\left\{\begin{matrix}\text{Stringkonstante}\\ \text{num.Konstante}\end{matrix}\right\}$$

entweder nur eine Stringkonstante, z.B. "NAME", oder aber nur eine numerische Konstante, z.B. -7.21, eingegeben werden, aber keine Variablenbezeichnung oder Ergebnisse von Funktionen oder Operationen (dafür müßte dann 'Ausdruck' statt 'Konstante' definiert sein).

Beispiele:

1. Die **DATA**-Anweisung hat das FORMAT

   DATA {num.Konst. / Stringkonst.} [, {num.Konst. / Stringkonst.}] ...

   Die Anweisung

   52 **DATA** "JAN",31,"FEB",28,"MRZ",31

   ist laut Formatbeschreibung syntaktisch korrekt, nicht aber

   47 N=31

   48 **DATA** "JAN",N,"FEB",N-3,"MRZ",N

   falsch falsch falsch

   N ist keine num.Konstante, ebensowenig N-3

2. Die **OPEN**-Anweisung hat das Format

   OPEN Zugriffsart,[#] Filenr., Filename [, Record-Länge]

   Zugriffsart: Stringausdruck (Ergebnis: "I", "O", "A" oder "R")

   Filenr.: num.Ausdruck (Ergebnis: 1 - 15)

   Filename: Stringausdruck (Ergebnis muß einen Begriff ergeben, der den Regeln zur Bildung von Filenamen entspricht; vgl. Kapitel 4.1.3)

   Record-Länge: num.Ausdruck (Ergebnis: 1 - beliebig)

   Nach dieser Formatschreibung ist die Anweisung

   42316 **OPEN** M$(I%),F%(I%),**MID$**(FILEN$,A%(I%),E%(I%)),R%(I%)

   syntaktisch korrekt.

M$(I%) darf nur den Inhalt "R" haben (da die Inhalte "I", "A" oder "O", bei Angabe der 'Record-Länge' den Fehler "Illegal function call" liefern!)

F%(I%) darf nur ganze Zahlen zwischen 1 und 15 enthalten; die String-Funktion **MID$(**FILEN$,A%(I%),E%(I%)**)** muß als Ergebnis einen String liefern, der den auf Diskette vorhandenen, gewünschten Filenamen beschreibt;

R(I%) muß eine ganze Zahl größer oder gleich 1 ergeben.

Zulässig wäre auch die Anweisung

4117 **OPEN** "I",2,"**Ø:**DAFILE"

3. Die Erklärung

H löscht das zuletzt am Bildschirm abgebildete Zeichen.

Wird die Taste H in Verbindung mit der blauen Taste (CONTROL-Taste), aber ohne Eingabeabschlußtaste, gedrückt, wird das zuletzt am Bildschirm erzeugte Zeichen wieder gelöscht.

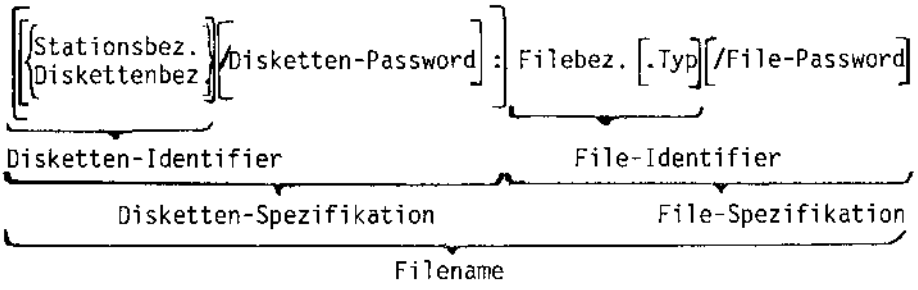
4.1.3 Bildung von Filenamen

Zur vollständigen Identifikation eines Files ist anzugeben

- auf welcher Diskette oder Diskettenstation das File sich befindet,
- welches File auf dieser Diskette gemeint ist.

Ein Filename besteht somit aus der Diskettenspezifikation und der Filespezifikation.

Der gesamte Filename hat folgendes FORMAT:



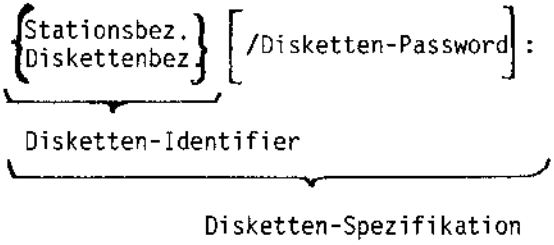
1. Die Diskettenspezifikation

   Zur vollständigen Identifikation einer Diskette ist anzugeben entweder:

   - in welcher Diskettenstation sich die Diskette befindet oder
   - wie die Diskette heißt

   Am Ende dieser Diskettenspezifikation ist stets der : zu setzen (kein Blank davor).

   Die gesamte Diskettenspezifikation hat folgendes FORMAT:



   Die Diskettenspezifikation enthält also:
   - Die Bezeichnung, die einer Diskette gegeben wurde ("Name der Diskette") oder
   - die Bezeichnung der Station, in der die Diskette liegt;
   - den Doppelpunkt am Ende

Beispiele:

Die folgenden Diskettenspezifikationen sind in der BASIC-Ebene zulässig:

a) "**Ø**/DKENNW:"
b) "DISKNAME:"
c) "**1**:"
d) "DISK1/KWDISK1:"

Bemerkungen:

- Groß- und Kleinbuchstaben sind bei Diskettenbezeichnungen zu unterscheiden!
- Wird keine 'Diskettenspezifikation' angegeben, wird (außer bei der Suche nach PCOS-Befehlen) ausschließlich auf der aktuellen Station gesucht. Das ist die zuletzt angesprochene Station. Wird die 'Diskettenbezeichnung' angegeben, wird die Suche nach der Diskette stets auf Station **Ø** begonnen und auf Station **1** fortgesetzt
- Sind Disketten-Passwords vergeben worden, sind diese stets anzugeben.
- Disketten-Passwords können mit dem PCOS-Befehl **vp** (Aufhebung durch **vd**) vergeben werden. Ist ein Disketten-Password einmal angegeben worde, kann bis zum nächsten Laden des Systems die Angabe unterbleiben. Dies gilt nicht mehr, wenn ein Diskettenwechsel erfolgt ist.
- Disketten können mit Schreibschutz versehen werden. Dies geschieht durch Überkleben der Ausstanzung am oberen rechten Rand der Diskette mit einem Stück Aluminiumfolie.

2. Die Filespezifikation

Sie enthält die Bezeichnung des Files und ggf. eine Angabe über den Typ des Files. Enthält eine Filespezifikation eine Typangabe, so muß diese stets als Bestandteil angegeben werden.

Zusätzlich zur Filebezeichnung [.Typ] kann einem File ein Password zugeordnet werden, ohne dessen Kenntnis ein Zugriff auf das File nicht möglich ist. Das File-Password ist mit / sofort an den File-Identifier anzuschließen.

Bestandteile der Filespezifikation sind im einzelnen:

Filebez.:	eine Folge von 1 bis zu 14 druckbaren Zeichen; ausgenommen sind alle am Ende des Kapitels aufgeführten unzulässigen Zeichen.
Typ: (optional)	eine Folge von mindestens einem druck baren Zeichen mit vorangehendem .; die maximale Anzahl Zeichen ergibt sich aus 13 minus Länge der Filebezeichnung; ausgenommen sind alle am Ende des Kapitels aufgeführten unzulässigen Zeichen. Einige Typ-Zeichenfolgen haben für PCOS spezielle Bedeutung und zwar: .cmd, .sav, .SAV, .BAS und .bas.
File-Password: (optional)	eine Folge von 1 bis zu 14 druckbaren Zeichen; mit vorangehendem /; ausgenommen sind alle am Ende des Kapitels aufgeführten unzulässigen Zeichen.

Beispiele:

Die folgenden Filenamen sind in der BASIC-Ebene zulässig:

a) "**Ø**/DKENNW:ANWPROG.BAS/FPW"
b) "DISKNAME:DAFILE.DAT"
c) "**1**:ASCFILE"
d) "**1**:MODUL1.ASC/FILEPASS"
e) "KUNDEN"

Bemerkungen:

- Groß- und Kleinbuchstaben sind bei Filenamen zu unterscheiden!
- Enthält ein File-Identifier eine Typangabe (durch Anhängen von .Typ), ist diese Erweiterung fester Bestandteil und muß beim Aufruf des Filenamens unbedingt angegeben werden.
- Sind Disketten- oder File-Passwords vergeben worden, sind diese stets anzugeben.
- Disketten-Passwords können mit dem PCOS-Befehl **vp** (Aufhebung durch **vd**) vergeben werden ; File-Passwords können mit dem PCOS-Befehl **fp** (Aufhebung durch **fd**) vergeben werden.
- Wird beim 'Filenamen' die 'Disketten-Spezifikation' weggelassen, wird das File ausschließlich auf der aktuellen Station gesucht. Das ist die zuletzt angesprochene Station. Wird hingegen die 'Diskettenbezeichnung' angegeben, wird die Suche nach der betreffenden Diskette auf Station **Ø:** begonnen und anschließend auf Station **1:** fortgesetzt.
- Auch Files können mit Schreibschutz versehen werden. Dies wird unter Verwendung des PCOS-Befehls **fw** (Aufhebung **fu**) erreicht.

- Es empfiehlt sich, Filenamen stets mit Typangabe zu versehen, da beim Inhaltsverzeichnis (PCOS-Befehle **vl** bzw. **vq** und BASIC-Befehl **FILES**) keine Angaben über den Typ des Files gemacht werden. Außerdem können ganze Gruppen von Files leichter kopiert werden, wenn sie gemeinsame Eigenschaften am Namen haben (siehe PCOS-Handbuch "Filenamen" und PCOS-Befehl **fc**).
- Sollen BASIC-Programmfiles mit dem 'Full-Screen-Editor' bearbeitet werden (PCOS-Befehl **ed** mit Parameter **%c**), sollte der File-Identifier nicht länger als 10 Zeichen (incl. .) sein, da der Full-Screen-Editor bei Angabe des Parameters **%b** automatisch eine Sicherungskopie erstellt. Der Name dieses Files wird durch automatisches Anhängen von **.bak** an den Filenamen des zu editierenden Files gebildet.

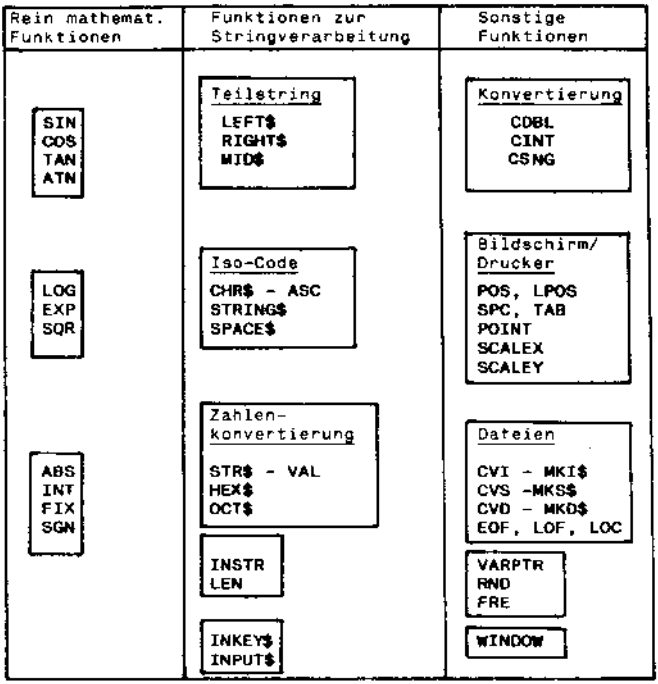
Unzulässige Zeichen

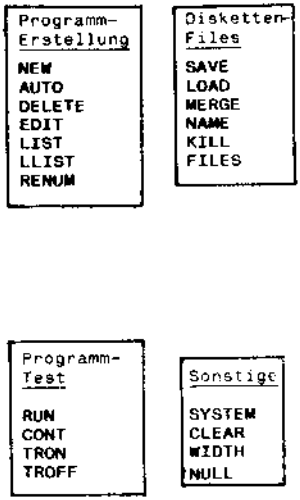
: Doppelpunkt (außer bei Diskettenspezifikation)	? Fragezeichen (außer bei Filenamenauswahl)
. Punkt (außer bei Typangabe)	! Ausrufezeichen
/ Slash (außer bei Passwords)	§ Paragraph-Zeichen
, Komma	= Gleichheitszeichen
+ Plus	Ö
* Sternchen (außer bei Filenamenauswahl)	₤ Pfundzeichen
" Anführungszeichen	␣ Blank
- Bundestrich oder Minus	; Strichpunkt
# Nummernzeichen	$ Dollarzeichen
& Ampersand	@ at-Zeichen
> größer-als-Zeichen	\ Backslash
' Hochkomma	

4.2 Gegliederte Zusammenstellung der Befehle, Anweisungen und Funktionen

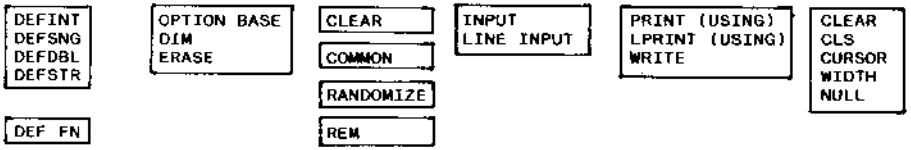
4.2.1 Zusammenstellungen

BASIC - STANDARD - FUNKTIONEN





BASIC-ANWEISUNGEN



Funktion: FRE

Funktionen: INPUT$, INKEY$
POS, LPOS
SPC, TAB

PCOS-Befehl: sb

PCOS-Befehle: sf, lt, ls, +/- s/d ...

3. Wertzuweisungen

LET
MID$ =
SWAP
LSET
RSET
ERROR

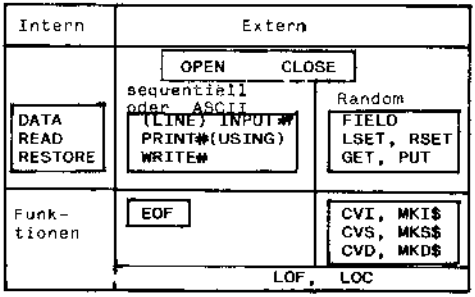
4. Logische Programmsteuerung

unbedingte Sprünge	bedingte Sprünge	Schleifen
GOTO	ON...GOTO	FOR/NEXT
GOSUB	ON...GOSUB	WHILE/WEND
RETURN	ON ERROR GOTO	
RESUME	IF...THEN...ELSE	

STOP	CALL
END	EXEC
CHAIN	

Spezielle Variablen: ERR, ERL

5. Dateien



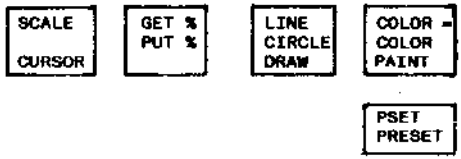
PCOS-Befehle: sb, ss und alle f-Befehle
Programmpaket ISAM zur Verwaltung indexsequentieller Filestrukturen

6. Stringverarbeitung

LSET
MID$=
RSET

Funktionen: FRE und alle Funktionen zur Stringverarbeitung

7. Graphik und Farbe



Funktionen: SCALEX, SCALEY, POINT
PCOS-Befehle: la, sp, rf, wf

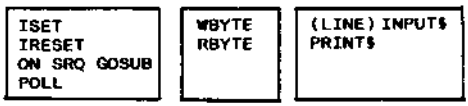
8. Bildschirmsteuerung

Funktionen: WINDOW , POS, TAB, SPC
PCOS-Befehle: sb, ss, rf, wf

9. V24-Schnittstelle

PCOS-Befehle: sd, rs, sc, ci

10. IEC-Schnittstelle



PCOS-Befehl: ie

11. selbstdefinierte Funktionen / selbsterstellte Assemblerroutinen

Funktionen: FN

4.2.2 Liste der für Variablennamen gesperrten Sprachelemente

ABS	DEFSTR	INT	OPEN	SIN
ALL	DELETE	IRESET	OPTION	SPACE$
AND	DIM	ISET	OR	SPC()
ASC	DRAW	KILL	PAINT	SQR
ATN	EDIT	LEFT$	PEEK	SRQ
AUTO	ELSE	LEN	POINT	STEP
CALL	END	LET	POKE	STOP
CDBL	EOF	LINE	POLL	STR$
CHAIN	EQV	LIST	POS	STRING$
CHR$	ERASE	LLIST	PRESET	SWAP
CINT	ERL	LOAD	PRINT	SYSTEM
CIRCLE	ERR	LOC	PSET	TAB()
CLEAR	ERROR	LOF	PUT	TAN
CLOSE	EXEC	LOG	RANDOMIZE	THEN
CLS	EXP	LPOS	RBYTE	TIME$
COLOR	FIELD	LPRINT	READ	TO
COMMON	FILES	LSET	REM	TROFF
CONT	FIX	MERGE	RENUM	TRON
COS	FN...	MID$	RESTORE	USING
CSNG	FOR	MKD$	RESUME	USR
CURSOR	FRE	MKI$	RETURN	VAL
CVD	GET	MKS$	RIGHT$	VARPTR
CVI	GOSUB	MOD	RND	WBYTE
CVS	GOTO	NAME	RSET	WEND
DATA	HEX$	NEW	RUN	WHILE
DATE$	IF	NEXT	SAVE	WIDTH
DEFDBL	IMP	NOT	SCALE	WINDOW
DEF	INKEY$	NULL	SCALEX	WRITE
DEFINT	INPUT	OCT$	SCALEY	XOR
DEFSNG	INSTR	ON	SGN	OUT

Die angegebenen reservierten Wörter können nicht allein bzw. nur durch Typkennzeichen bzw. ( ) erweitert als Variablennamen verwendet werden. Sie können aber durch ein oder mehrere zulässige Zeichen erweitert und dann als Variablenname verwendet werden. (Ausnahme: **FN**. Alle Variablennamen, die mit **FN** beginnen, sind unzulässig.)

4.3 Funktioneller Zusammenhang der Befehle und Anweisungen

4.3.1 Erstellen von Programmen

Programme für den M20 werden in der BASIC-Ebene erstellt (zur Auswahl der Ebene vgl. Kapitel 1.2). Nach Eingabe des Befehls **NEW** können BASIC-Anweisungen in den Arbeitsspeicher eingegeben werden. Dabei kann die automatische Zeilennumerierung (siehe Befehl **AUTO**) eingesetzt werden. Die Zeilennummern können auch per Hand eingetippt werden, wobei führende Nullen weggelassen werden dürfen. Die (Mehrfach-)Anweisungen werden in dem Moment im Arbeitsspeicher abgestellt, indem eine Eingabeschlußtaste (CR) betätigt wird. Die Reihenfolge der Erfassung von Anweisungszeilen ist beliebig; die (Mehrfach-)Anweisungen werden jedoch in der Reihenfolge der Zeilennummern im Arbeitsspeicher abgestellt.

Zur Änderung von (Mehrfach-)Anweisungen stehen die Möglichkeiten des Edit-Modes (vgl. Befehl **EDIT** und Kapitel 3.4.4) zur Verfügung.

(Mehrfach-)Anweisungen können in der BASIC-Ebene zur sofortigen Ausführung gebracht werden, indem sie ohne Zeilennummer eingegeben werden. Solche Anweisungen werden jedoch nicht im Arbeitsspeicher gespeichert (vgl. Kapitel 3.2)! Der Direkt-Mode wird z.B. folgendermaßen eingesetzt:

Ein im Arbeitsspeicher befindliches Programm wird durch **RUN** zum Laufen gebracht. (Damit werden alle Variableninhalte und Stack-Speicher des vorherigen Programms gelöscht; alle (logisch) offenen Files werden geschlossen). Durch eine BASIC-Anweisung **STOP** im Programm oder durch Betätigen von **C** wird das Programm unterbrochen; im Direkt-Mode werden die Inhalte bestimmter Variablen abgefragt oder Testwerte auf Variablen zugewiesen. Anschließend wird das Programm mit dem Befehl **CONT** fortgesetzt.

Achtung:

Wird, z.B. wegen eines Fehlers, der Edit-Mode zum Ändern von Anweisungen eingesetzt oder im Command-Mode neue Programmzeilen aufgenommen bzw. vorhandene gelöscht, gehen dabei der vorher geschaffene Datenbereich und alle Stacks verloren!!

Eine zweite Möglichkeit, Programme zu erstellen, besteht darin, bereits früher in Form von ASCII-Files auf Diskette gespeicherte Programm-Module mit dem aktuellen Inhalt des Arbeitsspeichers zu verbinden (vgl. **SAVE, CHAIN, MERGE** und Kapitel 3.5.5).

4.3.2 Vereinbarungsteil

Nach den das Programm beschreibenden **REM**s folgt der Vereinbarungsteil. In diesem werden festgelegt:

1. die im Arbeitsspeicher für das betreffende Programm (einschließlich Daten) zu reservierende Anzahl Bytes - sofern nötig, (nur Verringerung der durch PCOS-Befehl **sb** definierten Grösse ist möglich), sowie Anzahl Stack-Register - sofern sie von der Standard-Anzahl abweichen soll -; für beides ist die Anweisung **CLEAR** einzusetzen;

2. die Typen der Variablen, sofern diese nicht durch Typkennzeichen im Gesamtprogramm vereinbart werden (**DEFINT, DEFSNG, DEFDBL, DEFSTR**);
3. der niedrigstmögliche Wert für alle Array-Indices (Ø oder 1; **OPTION BASE**);
4. die Dimension von Arrays; es wird definiert, wieviele Indices zur Spezifizierung eines Elements des jeweiligen Arrays nötig sind (**DIM**);
5. der höchstmögliche Wert für die einzelnen Indices der jeweiligen Arrays (**DIM**).

Beispiel:

```
10 OPTION BASE 1
20 DEFINT I-J:DEFDBL X-Z:DEFSTR S-T
30 INDEX1=5:INDEX2=5:INDEX3=5
40 DIM D3.1(INDEX1,INDEX2,INDEX3),TEXTE(INDEX1)
50 DIM X(5),Y(5),Z(5)
```

1Ø legt fest, daß alle Array-Elemente als niedrigsten Index 1 haben dürfen.

2Ø legt fest, daß im folgenden alle ohne Typkennzeichen versehenen Variablennamen, die mit I oder J anfangen, vom Typ Integer, diejenigen mit X, Y oder Z am Anfang doppelt genaue numerische Variablen und die mit S oder T anfangende Strings sein sollen. Mit I, J, S, T, X, Y und Z beginnende Variablen können jedoch durch Typkennzeichen am Ende von dieser Vereinbarung abweichend definiert werden.

3Ø setzt die Integer-Variablen INDEX1, INDEX2 und INDEX3 alle auf 5.

40 legt fest, daß der einfach genaue Array D3.!() dreidimensional ist, wobei als höchster Wert für alle drei Indices 5 gilt. Da der niedrigste Wert 1 ist, wird für D3.!() somit für 5x5x5 Elemente Platz im Arbeitsspeicher reserviert. Auch der String-Array TEXTE() wird dreidimensional mit 5x5x5 Elementen definiert.

50 legt fest, daß die doppelt genauen Arrays X(), Y() und Z() je eindimensional sind und jeder als höchsten zulässigen Index 5 hat.

Default-Werte:

Numerische Variablen ohne jede Typvereinbarung sind von einfacher Genauigkeit. Die niedrigsten Indices von Array-Elementen sind 0. Die höchsten Indices sind 10.

Die numerischen Variablen haben beim ersten Aufruf ohne vorherige Wertzuweisung den Inhalt 0, alle Strings den Inhalt Leerstring.

Arrays können bis zu jedem beliebig hohen Index dimensioniert werden. Der Index muß im Bereich Integer liegen. Die Dimension (Anzahl Indices) kann bis 255 gehen. Für die Anzahl Arrayelemente muß ausreichend Platz im Arbeitsspeicher sein.

Bei der Durchführung der **DIM**-Anweisung werden für jedes Element 2 Byte zur Adressierung belegt; für numerische Elemente wird überprüft, ob ausreichend Platz für alle Elemente ist. Arrays können dynamisch dimensioniert werden (Begrenzung nur durch Kapazität des Arbeitsspeichers); vgl. Anweisung 40. Nach **ERASE**-Anweisung können sie redimensioniert werden. Der alte Inhalt und die alte Dimensionierung gehen dabei verloren. Dies ist auch verwendbar, um schnell einen gesamten Array auf 0 zu setzen (danach neues **DIM** erforderlich).

Strings werden im Speicher dynamisch verwaltet, so daß es nicht nötig ist, für sie Maximallängen zu definieren. Das Gesagte gilt auch für String-Arrays. Der für Strings benötigte Platz im Arbeitsspeicher ist somit abhängig von der aktuellen Länge der einzelnen Strings bzw. String-Array-Elemente. Um die bei Stringmanipulationen auftretenden "Fehlreservierungen" im Arbeitsspeicher aufzuheben und ggf. die Rechengeschwindigkeit zu erhöhen, empfiehlt sich im Programmablauf von Zeit zu Zeit eine "Bereinigung" des Arbeitsspeichers unter Anwendung der Funktion **FRE (Stringausdr.)**! Diese Bereinigung wird vom BASIC-Interpreter selbst durchgeführt, wenn sie unumgänglich wird, und kostet dann erheblich Rechenzeit.

Es empfiehlt sich, im Vereinbarungsteil auch alle im Programm benötigten selbstdefinierten Funktionen (vgl. Kapitel 4.3.7 und Anweisung **DEF FN**) anzuführen, denn sie müssen stets vor Aufruf definiert worden sein.

4.3.3 Wertändernde Anweisungen

Für erste Wertzuweisung bzw. Wertänderungen von Variablen bieten sich folgende Möglichkeiten an:

1. mit Hilfe der **LET...**=-Anweisung (Zuweisungsanweisung)
   Das Sprachelement **LET** kann dabei weggelassen werden. Eine Mehrfachzuweisung existiert nicht; es müssen mehrere Einzelzuweisungen erfolgen.
   Array-Zuweisungen (z.B. Setzen aller Elemente eines numerischen Arrays auf 1) sind nicht möglich. Dies muß in Form einer (ggf. verschachtelten) Schleife erfolgen.
2. mit Hilfe der Anweisung **MID$...**= , um in einer Stringvariablen einen Teil durch einen anderen String zu ersetzen.

3. mit Hilfe der Anweisung **SWAP**, um Variableninhalte zu vertauschen.
4. mit Hilfe der Anweisung **LSET** und **RSET** auf Feldvariablen eines Random-File-Puffers bzw. zum links- oder rechtsbündigen Einsetzen von String-Inhalten in vorhandene String-Variablen-Inhalte.
5. mit Hilfe der **FOR...NEXT**-Anweisung auf deren Laufvariable.
6. mit Hilfe der Anweisung **ERROR** auf die reservierte Variable **ERR**.
7. mit Hilfe von Tastatur-Inputs (**INPUT, LINE INPUT**).
8. mit Hilfe von Inputs aus einem internen Datenfile (**READ**).
9. mit Hilfe von Inputs aus einem sequentiellen externen Datenfile (**INPUT#, LINE INPUT#**)
10. mit Hilfe von Inputs aus einem Random-File-Puffer (**GET**).

Bei allen Wertzuweisungen numerischer Größen auf numerische Variablen ist auf eine ggf. notwendige Typumwandlung (vgl. Kapitel 2.5.1.4) und daraus resultierende Signifikanzverluste zu achten - (vgl. z.B. Kapitel 2.7.1)!!

Solche Signifikanzverluste lassen sich auch durch Anwendung der Funktionen **CDBL** u.ä. nicht vermeiden.

Die Problematik kann jedoch mit Hilfe einer Anweisung der Art

D# = **VAL(STR$(G!))**

umgangen werden, also unter Zuhilfenahme der Stringverarbeitung.

4.3.4 Stringverarbeitung

Es werden in diesem Kapitel nur die folgenden Stringmanipulationen besprochen:

1. Stringverkettung
2. Teilstringbildung und Ersetzungen in Strings
3. Verwandlung von numerischen Werten in Strings und umgekehrt (Konvertierungen)

Formatierungsmöglichkeiten (Masken) werden in Kapitel 4.3.9.2 erörtert, Fragen im Zusammenhang mit Inputs über Tastatur in Kapitel 4.3.8 und die Verarbeitung von Strings beim Schreiben und Lesen auf externen Datenfiles im Kapitel 4.3.10.2.

4.3.4.1 Stringverkettung (String-Addition)

Zur Verkettung von Stringausdrücken dient der String-Operator **+**.

Beispiel:

```
1 CLEAR:DEFSTR T:OPTION BASE 1:DIM T(5)
10 A$="Personal":T="Computer":T(1)="M":T(2)="20"
20 E$=A$+"-"+T+" "+T(1)+" "+T(2)
30 PRINT E$
RUN
Personal-Computer M 20
```

Es ist generell zu beachten, daß die Gesamtlänge eines Strings nie mehr als 255 Zeichen betragen darf!

4.3.4.2 Teilstringbildung (Substrings) und Ersetzungen in Strings

Zur Bildung von Teilstrings können die Funktionen **LEN, INSTR, LEFT$, RIGHT$** und **MID$** (als Funktion!) eingesetzt werden.

Beispiel: (der obige Programmteil wird übernommen)

```
40 POSITION%=INSTR(E$,"-"):T(3)=LEFT$(E$,POSITION%-1)
60 T(4)=RIGHT$(E$,LEN(E$)-POSITION%)
70 T(5)=MID$(E$,POSITION%+1,INSTR(E$," ")-POSITION%)
80 PRINT POSITION%;"**";T(3);"**";T(4);"**";T(5)
RUN
Personal-Computer M 20
 9 **Personal**Computer M 20**Computer
```

Ersetzungen von Teilen eines Strings durch einen anderen String können entweder durch Verkettung der betroffenen Teilstrings oder aber mit Hilfe der Anweisung **MID$...=** durchgeführt werden.

Beispiel: (das obige Programm wird fortgesetzt)

```
90 MID$(E$,1)="     Ihr  " PRINT "*****" E$;"**";
92 E$=RIGHT$(E$,LEN(E$)-5):PRINT E$
RUN
Personal-Computer M 20
 9 **Personal**Computer M 20**Computer
*****     Ihr Computer M 20**Ihr Computer M 20
```

Wie man sieht, läßt sich bei Verwendung der **MID$**-Anweisung die aktuelle Länge des Ausgangsstrings nicht verändern.

Um in Strings eine bestimmte Anzahl von Blanks einzubringen, kann die Funktion **SPACE$** eingesetzt werden.

Mit der Funktion **STRING$** lassen sich Strings erzeugen, die ausschließlich aus einem gewählten Zeichen bestehen (z.B. Bindestriche für das Unterstreichen von Textzeilen im Bildschirm).

Beispiel: (obiges Programm wird fortgesetzt)

```
96 ANZAHL%=9:PRINT SPACE$(ANZAHL%/2) E$
97 PRINT STRING$(LEN(E$)+ANZAHL%,"*")
100 PRINT STRING$(40,7);:'erzeugt 40 x akustisches Signal
E$="Ihr Computer M 20":GOTO 96
     Ihr Computer M 20
**************************
```

Die Funktionen **SPACE$** und **STRING$** können auch eingesetzt werden, um für Stringvariablen eine Länge festzulegen (vgl. Anweisungen 2Ø und 3Ø im nächsten Beispiel). Es ist zu beachten, daß Strings stets nur in aktueller Länge verwaltet werden. Eine Stringsvariable, die noch keinen Inhalt zugewiesen bekommen hat, hat den Inhalt Leerstring.

(Für das Arbeiten mit Random-Files existiert eine eigene Anweisung - **FIELD** - mit der die Feldlängenvereinbarung für die Variablen des Random-Files getroffen wird.)

Mit Hilfe der Anweisungen **LSET** und **RSET** lassen sich in Strings mit einer vorher festgelegten Länge (z.B. über **SPACE$** für Ausgabefelder) rechtsbündige oder linksbündige Ersetzungen vornehmen. Nicht von der Ersetzung betroffene Zeichen werden durch Blanks ersetzt. Ist der Ausgangsstring der Leerstring, verändert er sich auch nach **LSET** bzw. **RSET** nicht.

Beispiel: (neues Programm)

```
1 PRINT STRING$(40,"*"):FLX(1)=17:FLX(2)=20
20 F$(1)=STRING$(FLX(1),"?"):F$(2)=STRING$(FLX(2),"#")
30 PRINT F$(1);"**";F$(2);"!!!!!!"
40 RSET F$(1)="1. FELD":LSET F$(2)="2. FELD"
70 PRINT F$(1),"**";F$(2);"!!!!!!"
RUN
????????????????**####################!!!!!!
          1. FELD**2. FELD             !!!!!!
```

4.3.4.3 Konvertierungen

Für die Umwandlung von numerischen Werten zu Strings bieten sich folgende Möglichkeiten:

1. Konvertierung von ISO-Codes (vgl. Kapitel 4.3.17) zu den durch sie definierten Zeichen mit Hilfe der Funktionen **CHR$** (Umkehrung: **ASC**) und **STRING$**;
2. direkte Konvertierung von dezimal, oktal oder hexadezimal dargestellten numerischen Werten in die optisch deckungsgleiche Stringdarstellung mit Hilfe der Funktion **STR$** (Umkehrung:**VAL**);
3. Konvertierung von dezimal dargestellten numerischen Werten in die oktale oder hexadezimale Darstellung in String-Form mit Hilfe der Funktionen **OCT$** und **HEX$** (Umkehrung über Stringverkettung und **VAL**).

Beispiel zu 1.:

```
 1 AF$=CHR$(34):Z%(1)=77:Z%(2)=32:Z%(3)=50:Z%(4)=48
 5 PRINT AF$+CHR$(Z%(1))+CHR$(Z%(2))+CHR$(Z%(3))+CHR$(Z%(4))+AF$
 7 PRINT STRING$(LEN(M$),34)
RUN
"M 20"
""""""
```

Beispiel zu 2.:

```
10 A(1)=7.430:A(2)=-60.93:A(3)=-.27:A(4)=3E-20:A(5)=&072:A(6)=&HB7D
20 PRINT STRING$(32,"*"):I%=1
40 PRINT STR$(A(I%));:IF I%=6 THEN PRINT "&&":STOP
60 I%=I%+1:GOTO 40
RUN
**********************************
 7.43-60.93-.27 3E-20 58 2941&&
```

Die über **STR$** gewonnenen Strings enthalten also die Zeichenfolge der dezimal dargestellten numerischen Werte im Standardformat (vgl. dazu die Anmerkung bei **PRINT** sowie Kapitel 4.3.9). Führende und hinter dem Komma stehende nicht-signifikante Nullen werden nicht konvertiert. Im Gegensatz zum Standardformat wird jedoch kein Blank am Ende des Strings erzeugt!!

Beispiel zu 3.:

```
10 DEFINT A,I:OPTION BASE 1:DIM A(3)
20 A(1)=635:A(2)=&H6B:A(3)=&O30
30 FOR I=1 TO 3
40 PRINT A(I),OCT$(A(I)),HEX$(A(I))
50 NEXT
RUN
 635          1173          27B
 107          153           6B
 24           30            18
```

Die über **OCT$** bzw. **HEX$** gewonnenen Strings enthalten also die Zeichenfolge der oktal bzw. hexadezimal dargestellten numerischen Werte ohne führende Blanks wie beim Standardformat. Die Präfixe **&O** bzw. **&H**, die zur Darstellung oktaler oder hexadezimaler numerischer Konstanten verwendet werden (vgl. Kapitel 2.5.3.1), werden somit nicht vorangestellt.

Für die Umwandlung von Strings in numerische Werte bieten sich folgende Möglichkeiten:

a) die Funktion **ASC** als Umkehrung von **CHR$**;
b) die Funktion **VAL** als Umkehrung von **STR$**.

Beispiel zu a):

```
Ok
T$="Test-Text":PRINT ASC(T$)
 84
Ok
```

Nur vom ersten Zeichen von **T$** wird der ISO-Code ermittelt.

Beispiel zu b):

```
10 T$="34, Straße 25":PRINT VAL(T$)
RUN
 34
```

Der numerische Wert muß, ebenso wie ein eventuell vorhandenes Vorzeichen, am Anfang des Arguments stehen, sonst liefert die Funktion **VAL** das Ergebnis Ø.

Beispiel:

```
10 T$="20-04-84"
20 PRINT VAL(MID$(T$,3))
RUN
-4
```

Bei der Rückkonvertierung eines Strings, der eine oktale oder hexadezimale Zahlendarstellung enthält, ist wie folgt vorzugehen:

Beispiel:

```
10 A#=98#:PRINT A#;
20 T$=OCT$(A#):PRINT T$;
40 REM Zuerst in eine oktale num. Konstante verwandeln
50 T$="&O"+T$:B#=VAL(T$):PRINT B#;T$
RUN
 98 142 98 &O142
```

Bei der Rückumwandlung von Strings mit hexadezimaler Zahlendarstellung als Inhalt ist statt **OCT$** die Funktion **HEX$** und statt des Präfixes **&O** das Präfix **&H** vor den zurückzukonvertierenden String zu setzen.

Bemerkung:

Bei häufigen Stringmanipulationen empfiehlt sich, im Programm hin und wieder mit Hilfe der Funktion **FRE(**Stringausdr.**)** den Arbeitsspeicher zu bereinigen (neu zu ordnen) (vgl. **FRE**).

4.3.5 Ablaufsteuerung

Um in einem Programm vom linearen Ablauf, der durch die Ordnung der Programm-Anweisungen im Arbeitsspeicher nach aufsteigender Zeilennummer festgelegt ist, abzuweichen, stehen Sprunganweisungen zur Verfügung. Die Zeilennummer des Sprungzieles kann grundsätzlich nicht als Variable definiert werden.

Es ist nicht möglich, bestimmte Anweisungen in einer Programmzeile anzuspringen, die aus durch : getrennten Anweisungen besteht (Ausnahmen: **RETURN** und **RESUME**).

4.3.5.1 Unbedingte Sprünge

Mit den Anweisungen **GOTO, GOSUB, RETURN** oder **RESUME** (diese wird im Zusammenhang der Fehlerbehandlungsroutine in Kapitel 4.3.12 besprochen) werden Sprünge verlangt, die immer ausgeführt werden, sobald die betreffende Anweisung erkannt wird.

Bei **GOTO** wird die Zeilennummer des Absprungs nicht gespeichert. Mit einer **GOSUB**-Anweisung wird sofort ein Stack-Speicher errichtet, in dem festgehalten wird, von wo der **GOSUB**-Aufruf erfolgte. Trifft das Programm auf eine **RETURN**-Anweisung, so wird die Absprungadresse aus dem letzten errichteten Stack-Speicher wieder gelöscht und automatisch zu der Anweisung gesprungen, die der Anweisung mit der gerade ermittelten Adresse folgt. Dies muß nicht die nächste Zeilennummer sein (bei durch : verbundenen Anweisungen, von denen eine ein **GOSUB** ist!).

Werden zuviele **GOSUB**-Anweisungen verschachtelt oder wurden mit **GOSUB**-Anweisungen zu viele Stack-Speicher errichtet, ohne daß sie durch **RETURN**-Anweisungen wieder gelöscht wurden, kann ein Overflow im Arbeitsspeicher auftreten. Die Anzahl der möglichen Stack-Speicher kann mit Hilfe der Anweisung **CLEAR** geändert werden.

Beispiel:

```
1 I%=0
10 I%=I%+1:GOSUB 2000
2000 I%=I%+1:GOSUB 10
```

Die führt zur Fehlermeldung "Out of memory in 2ØØØ". Wird I% abgefragt (**?I%**), wird bei Standard-Arbeitsspeicher 71 ausgegeben. Es konnten also nur 7Ø **GOSUB**-Stack-Speicher gleichzeitig verwaltet werden (Default-Wert).

Es existiert keine BASIC-Anweisung, mit der die Anzahl der angelegten Stack-Speicher direkt abgefragt werden kann oder mit der alle Stack-Speicher gelöscht werden können.

Beispiel zu **GOTO** und **GOSUB**:

```
0      'Zwischen den folgenden Zeilen seien Verarbeitungszeilen
1      REM Es folgt die Programmbeschreibung
11     GOSUB 29001:REM Vereinbarungsteil aufrufen
500    GOTO 15001:REM Hauptverarbeitung Steuerleiste
15000 REM Steuerleiste
15672 GOSUB 20201
15900 REM Endverarbeitung Steuerleiste
19999 END
20200 REM Datumsprüfung
20218 GOSUB 25901:RETURN
20219 RETURN
25900 REM Ausgabe eines Datums auf dem Drucker
25999 RETURN
29000 REM Vereinbarungsteil
29999 RETURN
```

4.3.5.2 Bedingte Sprünge

Häufig werden Programmverzweigungen nur dann gewünscht, wenn bestimmte Bedingungen erfüllt bzw. nicht erfüllt sind. Solche Möglichkeiten bieten die Anweisungen **IF...THEN...ELSE, ON...GOTO, ON...GOSUB** und **ON ERROR GOTO** (diese wird im Zusammenhang der Fehlerbehandlungsroutine, Kapitel 4.3.12, erörtert). Außerdem leisten die Schleifenanweisungen **FOR...NEXT** und **WHILE...WEND** ähnliches.

Beispiel zu **IF...THEN...ELSE:**

```
 1 REM ZIEHUNG 6 AUS 49 ohne Zusatzzahl
 5 RANDOMIZE:LPRINT:LPRINT
 10 DIM Q%(49)
 20 ZIEHUNG%=1
 30 ZAHL%=INT(RND*49)
 40 IF ZAHL%=0 OR ZAHL%)49 GOTO 30
 45 IF Q%(ZAHL%) THEN 30
 46 'gleiche Bedeutung wie IF Q%(ZAHL%)<>0 ...
 48 'Q%(ZAHL%) enthält Nr. der Ziehung, falls Zahl
 49 'bereits gezogen ist, ansonsten 0 (Default-Wert)
 50 LPRINT "Ziehung Nr.";ZIEHUNG%;"ist";ZAHL%
 60 Q%(ZAHL%)=ZIEHUNG%:ZIEHUNG%=ZIEHUNG%+1
 80 IF ZIEHUNG%<7 THEN 30 ELSE I%=0:GOSUB 1001:END
1000 REM Ausgabe in aufsteigender Reihenfolge
1001 LPRINT:LPRINT
1005 I%=I%+1:IF Q%(I%) THEN LPRINT "Ziehung Nr.";Q%(I%);"war";I%
1010 IF I%=49 THEN RETURN ELSE 1005
RUN
Ziehung Nr. 1 ist 20
Ziehung Nr. 2 ist 43
Ziehung Nr. 3 ist 28
Ziehung Nr. 4 ist 22
Ziehung Nr. 5 ist 1
Ziehung Nr. 6 ist 7


Ziehung Nr. 5 war 1
Ziehung Nr. 6 war 7
Ziehung Nr. 1 war 20
Ziehung Nr. 4 war 22
Ziehung Nr. 3 war 28
Ziehung Nr. 2 war 43
```

Beispiel zu **ON...GOTO** und **ON...GOSUB**:

```
1 REM Wahrheitswerte ermitteln
2 DEFSTR T:DEFDBL W
3 OPTION BASE 0
5 T(0)="wahr":T(1)="falsch!"
10 INPUT "# Wert 1, # Wert 2:";W1,W2
20 PRINT "Vergleich Wert 1 mit Wert 2:"
22 PRINT "1:=   2:<   3:>   4:<>   5:<=   6:>="
27 INPUT "Ihre Wahl (1 - 6): ";Q%
30 IF Q%<1 OR Q%>6 THEN 20
35 LPRINT W1;
40 ON Q% GOSUB 110,210,310,410,510,610:GOTO 9901
100 REM =
110 LPRINT "= "W2"ist "T((W1=W2)+1):RETURN
200 REM <
210 LPRINT "< "W2"ist "T((W1<W2)+1):RETURN
300 REM >
310 LPRINT "> "W2"ist "T((W1>W2)+1):RETURN
400 REM <>
410 LPRINT "<> "W2"ist "T((W1<>W2)+1):RETURN
500 REM <=
510 LPRINT "<= "W2"ist "T((W1<=W2)+1):RETURN
600 REM >=
610 LPRINT ">= "W2"ist "T((W1>=W2)+1):RETURN
9900 'Ende oder neue Werte oder Vertauschen aktuelle Werte
9901 PRINT "Neuer Vergleich (ja=1 nein=2 vertauschen=3)"
9902 INPUT "Ihre Wahl (1 - 3): ";Q%
9910 IF Q%<1 OR Q%>3 THEN 9901
9920 ON Q% GOTO 10,9999
9930 SWAP W1,W2
9940 GOTO 20
9999 END
```

```
 7.28 < -5.2 ist falsch!
 2.2208 >=  2.2028 ist wahr
 2.2028 <  2.2208 ist wahr
 2.2208 <  2.2028 ist falsch!
 5D-123 < -5D+123 ist wahr
-5D+123 <=  5D-123 ist wahr
```

4.3.5.3 Programmende und Programmunterbrechung

Ein BASIC-Programm im M20 kann eine oder mehrere **END**-Anweisungen an beliebigen Stellen im Programm haben. Diese Anweisung bewirkt das sofortige Programmende unter Schließung alle Datenfiles und die Meldung "Ok". Die Variableninhalte im Arbeitsspeicher bleiben vorhanden.

Es kann aber auch ohne **END**-Anweisung gearbeitet werden. Erreicht das Programm die höchste Zeilennummer ohne **END**, wird ebenfalls "Ok" gemeldet. Wurden die Datenfiles vorher nicht mit **CLOSE** geschlossen, bleiben sie ohne **END**-Anweisung offen.

Ein vorzeitiger Programmabruch kann durch **C** jederzeit erzwungen werden (vgl. Kapitel 3). Die Files werden nicht geschlossen und die Variableninhalte im Arbeitsspeicher bleiben vorhanden. Wird ein Programm durch die **STOP**-Anweisung verlassen, befindet sich das System danach im Direkt-Mode (vgl. Kapitel 3); alle im Programm geöffneten Datenfiles bleiben offen und die Variableninhalte erhalten. Das Programm kann nach **STOP** oder **C** mit Hilfe des Befehls **CONT** fortgesetzt werden, und zwar wird ab der nächsten Anweisung weitergearbeitet.

Bitte beachten:

Werden durch den Befehl **EDIT** Programmzeilen nach einer Programmunterbrechung verändert oder neue Zeilennummern aufgenommen bzw. vorhandene gelöscht, gehen die Inhalte aller Variablen, ebenso wie alle Stacks, verloren. Der Befehl **CONT** bewirkt danach, daß mit Default-Werten für die Variablen weitergearbeitet wird (Ø für num. und "" für String-Variablen).

Zur Verkettung von Programmen mit Hilfe von **CHAIN** vgl. Kapitel 4.3.13.

4.3.6 Schleifen

Programmteile, die eine Folge von sich gleichenden oder ähnelnden aufeinanderfolgenden Vorgängen darstellen, werden in Schleifen programmiert. Dazu bieten sich folgende Möglichkeiten:

1. eine Kombination von **GOTO** und **IF...THEN**-Anweisungen,
2. eine **IF...THEN...ELSE**-Anweisung,
3. eine aus den Anweisungen **FOR** und **NEXT** gebildete zählergesteuerte Schleife,
4. eine aus den Anweisungen **WHILE** und **WEND** gebildete, von einem Vergleichs- oder logischen Ausdruck abhängige Schleife.

Beispiel zu 1.:

```
1 REM Einlesen von Daten aus internem Datenfile
30 INPUT "JAHR ";NO%
40 IF NO%<>0 AND (NO%<1980 OR NO%>1983) THEN 30
45 IF NO%=0 THEN END
50 ON NO%-1979 GOSUB 980,981,982,983
60 I%=1
65 LPRINT:LPRINT "JAHR:";NO%
66 LPRINT "==========="
70 READ U!:LPRINT U!;
80 IF I%<12 THEN I%=I%+1:GOTO 70
99 GOTO 30
980 RESTORE 1980:RETURN
981 RESTORE 1981:RETURN
982 RESTORE 1982:RETURN
983 RESTORE 1983:RETURN
1980 DATA 10,20,40,30,20,80,70,80,60,20,40,10
1981 DATA 15,25,40,25,30,60,50,70,60,10,50,80
1982 DATA 50,60,20,40,30,80,90,120,110,70,65,80
1983 DATA 40,,,10,100,105,130,40,70,25,30,90
1995 ' , als Platzhalter für 0 bzw. "" möglich

JAHR: 1982
==========
 50  60  20  40  30  80  90  120  110  70  65  80
JAHR: 1980
==========
 10  20  40  30  20  80  70  80  60  20  40  10
```

Beispiel zu 2.: (gleiches Programm wie oben, nur geänderte Programmzeilen 7Ø bis 8Ø)

```
70 I%=I%+1
75 READ U!:LPRINT U!;
80 IF I%=12 THEN 99 ELSE 70

JAHR: 1983
==========

 40  0  0  10  100  105  130  40  70  25  30  90

JAHR: 1982
==========

 50  60  20  40  30  80  90  120  110  70  65  80
```

Beispiel zu 3.: (**DATA**- und **RESTORE**-Anweisungen von oben werden übernommen)

```
1 REM Einlesen von Daten aus internem Datenfile
20 INPUT;"von ",N0%
30 IF N0%<1980 OR N0%>1983 THEN 20
40 INPUT" bis ",N9%
50 IF N9%<1980 OR N9%>1983 OR N9%<N0% THEN 40
70 FOR I1%=N9% TO N0% STEP -1
75 LPRINT "JAHR:";I1%
76 LPRINT "==========":LPRINT
79 ON I1%-1979 GOSUB 980,981,982,983
80 FOR I2%=1 TO 12
90 READ U!:LPRINT U!;
100 NEXT I2%
110 LPRINT:LPRINT
120 NEXT
199 END
```

Bei den **FOR...NEXT**-Schleifen handelt es sich um zählergesteuerte "pre-checked loops", also Schleifen, bei denen die Endwert-Über- bzw. -Unterschreitung am Beginn der Schleife geprüft wird. Für jede **FOR...NEXT**-Schleife wird ein Stack-Speicher benötigt. Ein vorzeitiger Aussprung aus der Schleife (z.B. aufgrund einer Bedingung) mit **IF...THEN** oder mit **GOTO** ist zwar möglich, löscht aber nicht den Stack-Speicher.

Das heißt, daß ein Rücksprung mit anschließender Weiterabarbeitung der Schleife möglich ist. Vor allem heißt das auch, daß häufige vorzeitige Aussprünge ohne Stack-Löschung bald zum Overflow im Stack-Bereich des Arbeits-Speichers (der standardmäßig nur eine begrenzte Anzahl von Stack-Speichern gleichzeitig ermöglicht) führen. Dies könnte durch Verwendung der Anweisung **CLEAR** geändert werden. Besser ist ein Verfahren wie im folgenden Beispiel demonstriert (Programmzeilen 80 bis 99):

```
0 'Untersuchung eines Strings; zulässige Zeichen sind:
2 'alle Groß- und Kleinbuchstaben außer ß; alle Ziffern, Blanks
3 'Bindestriche, Punkte, Kommas sowie / . Alle Kleinbuchstaben
4 'werden z.B. zum späteren Sortieren der Eingaben in Groß-
5 'buchstaben verwandelt.
10 PRINT "Eingabe der Zeichenfolge ";:LINE INPUT EINGABE$
15 LPRINT "EINGABE: ";EINGABE$
30 FALSCH%=0:FOR I%=1 TO LEN(EINGABE$)
40 C%=ASC(MID$(EINGABE$,I%,1))
50 IF C%=32 OR (C%)43 AND C%(58) OR (C%)64 AND C%(94) THEN 99
70 IF C%)96 AND C%(126 THEN MID$(EINGABE$,I%)=CHR$(C%-32):GOTO 99
80 FALSCH%=-1:'Schalter für vorzeitigen Aussprung
90 I%=LEN(EINGABE$):' Erhöhung des Laufindex auf Endwert ==> bei
92                  ' NEXT sofortiger Aussprung
98 LPRINT EINGABE$;"ist eine unzulässige Eingabe"
99 NEXT I%:IF FALSCH% THEN 10
110 LPRINT EINGABE$ "ist o.k.!"   'umgewandelte Eingabe

EINGABE: Ist die Eingabe in Kleinbuchstaben zulässig?
? ist unzulässig!
EINGABE: Die Eingabe in Kleinbuchstaben ist zulässig.
DIE EINGABE IN KLEINBUCHSTABEN IST ZULÄSSIG. ist o.k.!
```

Wie sich am Beispiel zeigt, kann (Anfangs- und) Endwert über Variablen einmalig festgelegt werden. Es gelten die Werte der Variablen im Moment der erstmaligen Abarbeitung der **FOR**-Anweisung (vgl. Programmzeile 3Ø).

Nach vollständiger, mindestens einmaliger Abarbeitung hat die 'Laufvariable' einer **FOR...NEXT**-Schleife den Wert 'Endwert' + 'Schrittweite'. (Ist 'Schrittweite' negativ, ist selbstverständlich der Betrag von 'Schrittweite' zu subtrahieren.) Wird eine solche Schleife nie abgearbeitet (da von Anfang an 'Endwert' über/unterschritten ist), hat die 'Laufvariable' nach dem Abarbeiten der **FOR**-Anweisung den Wert 'Anfangswert'.

Wird aus einer Schleife vorzeitig direkt ausgesprungen, ohne den Stack-Speicher zu löschen (siehe Zeilennr. 9Ø), gibt die Laufvariable Aufschluß darüber, wie oft die **FOR...NEXT**-Schleife durchgearbeitet wurde (Rücksprung ist dann möglich!).

Beispiel zu 4.:

```
5115 'Bubble Sort für den Array A$() mit J% Elementen
5118 MARKE%=1 'erzwingt mindestens einen Durchlauf der Schleife!
5120 WHILE MARKE% 'kurz statt: WHILE MARKE()0
5122       MARKE%=0
5124       FOR I%=1 TO J%-1
5126           IF A$(I%)<=A$(I%+1)
                   THEN   5128
                   ELSE   SWAP A$(I%),A$(I%+1):MARKE%=1
5128       NEXT I%
```

WHILE...WEND-Schleifen benötigen ebenfalls Stack-Speicher und bieten sich vor allem anstelle von **FOR...NEXT**-Schleifen an, wenn die Anzahl der Durchläufe einer Schleife nicht festliegt, sondern abhängt von bestimmten Bedingungen, die sich innerhalb der Schleife ergeben.

4.3.7 Unterprogramme und selbstdefinierte Funktionen

4.3.7.1 Unterprogramme (Subroutinen)

Unterprogramme sind Bestandteile eines Hauptprogramms, die Routinen beinhalten, die in unregelmäßigen Abständen im Hauptprogramm häufiger benötigt werden. Sie sind im allgemeinen stark parametrisiert, d.h. mit Variablen versehen. Unterprogramme werden mit **GOSUB** aufgerufen; mit **RETURN** wird die Kontrolle an das Hauptprogramm zurückgegeben (weitere Abarbeitung ab der auf das betreffende **GOSUB** folgenden Anweisung).

Beispiel:

```
1 REM Datumsprüfung
20 LINE INPUT "Datum 1 ";D1$:D$=D1$
30 GOSUB 705:IF O.K.% THEN TAG1%=T%:MONAT1%=M%:JAHR1%=J% ELSE 20
40 LINE INPUT "Datum 2 ";D2$:D$=D2$
50 GOSUB 705:IF O.K.% THEN TAG2%=T%:MONAT2%=M%:JAHR2%=J% ELSE 40
699 GOTO 5010
700 REM Datumsprüfung, nur Eingaben der Form TTMMJJ oder TMMJJ
702 REM 01<=JJ<=99
705 O.K.%=-1    'Schalter setzen
710 D!=VAL(D$):IF MID$(STR$(D!),2)<>D$ THEN O.K.%=0:RETURN
720 IF D!<=0 OR D!<>FIX(D!) OR D!>999999! THEN O.K.%=0:RETURN
725 J%=D!-INT(D!/100)*100 'MOD nur für Intergerwerte verwendbar!
730 IF J%=0 THEN O.K.%=0:RETURN
740 T%=INT(D/10000)
760 M%=INT(D!/100) MOD 100  ' Rest der Integerdivision
765 IF M%<1 OR M%>12 THEN O.K.%=0:RETURN
770 IF T%> 31 + ( (M%=4) OR (M%=6) OR (M%=9) OR (M%=11) )
              + 2 *  ((M%=2) AND (J% MOD 4 =0))
              + 3 *  ((M%=2) AND (J% MOD 4<>0)) OR T%=0
    THEN O.K.%=0
780 RETURN
5000 REM Hauptverarbeitung
5010 LPRINT:LPRINT D1$,TAG1%,MONAT1%,JAHR1%
5030 LPRINT D2$,TAG2%,MONAT2%,JAHR2%
9900 STOP

280283          28              2               83
290284          29              2               84
```

Wie sich im Beispiel zeigt, sind bei der Arbeit mit Unterprogrammen i.a. folgende Schritte nötig:

1. Zuweisung(en) der Daten, die das Unterprogramm benötigt, auf die im Unterprogramm dafür verwendeten Variablen.

2. Setzen von Schaltervariablen im Unterprogramm, falls es im Hauptprogramm später nötig ist zu wissen, welche Abläufe im Unterprogramm stattgefunden haben. (Das Unterprogramm muß ja stets über **RETURN** verlassen werden!)

3. Sichern von Variableninhalten (Ergebnissen) aus dem Unterprogramm in Hilfsvariablen, falls diese Ergebnisse im Hauptprogramm auch benötigt werden, wenn das Unterprogramm später nochmals aufgerufen wird. (Der nächste Aufruf des Unterprogramms verändert sämtliche darin vorkommenden Variablen!)

4. Abgrenzung des Unterprogramms vom Hauptprogramm, z.B. durch **STOP**-Anweisung vor Beginn des Unterprogramms. (Läuft das Hauptprogramm bei linearen Abarbeiten ohne **GOSUB**-Anweisung direkt in ein Unterprogramm, wird dies erst entdeckt, wenn die **RETURN**-Anweisung erkannt ist, für die ja kein **GOSUB**-Stack-Speicher errichtet wurde (Fehler: RETURN without GOSUB). Im schlimmsten Fall wird sogar ein Stack-Speicher zum Rücksprung herangezogen, der von einem vorherigen Unterprogramm errichtet und bisher noch nicht mit **RETURN** gelöscht wurde.)

4.3.7.2 Selbstdefinierte Funktionen

Der M20 kennt zahlreiche Standardfunktionen (festverdrahtete Funktionen, Bibliotheksfunktionen).

Zu unterscheiden sind

a) numerische Funktionen und

b) String-Funktionen

zu a) Numerische Funktionen liefern als Ergebnis stets einen numerischen Wert, unabhängig vom Typ des Arguments bzw. der Argumente, die in die Funktion eingereicht werden.

zu b) String-Funktionen liefern als Ergebnis stets einen String, unabhängig vom Typ des Arguments bzw. der Argumente, die in die Funktion eingereicht werden. String-Funktionen sind stets am **$**-Zeichen am Ende des Funktionsnamens zu erkennen. Beispiele: **STRING$, CHR$,** Funktion **MID$**.

Neben diesen Standardfunktionen besteht die Möglichkeit, öfter auftretende Operationen als selbstdefinierte Funktionen im Programm selbst zu definieren und diese unter ihrem bei der Definition vergebenen Namen aufzurufen. Das Argument/die Argumente der Funktion können von beliebigem Typ sein; Typ und Anzahl der Argumente werden in der Definition festgelegt und müssen beim Aufruf der Funktion mit der Definition übereinstimmen.

Beispiel: (selbstdefinierte numerische Funktion)

```
10 DEFDBL A,M
20 DEF FNMITTELWERT(ARG1,ARG2)=(ARG1+ARG2)/2
30 M1=FNMITTELWERT(3,5)
40 LPRINT "Der Mittelwert aus";3;"und";5;"ist";M1
50 INPUT "Wert 3,Wert 4";W3#,W4#
60 M2=FNMITTELWERT(W3#,W4#)
65 LPRINT "Der Mittelwert aus ";W3#;"und ";W4#;"ist ";M2
70 LPRINT "Der Mittelwert aus 3, 5,";W3#;"und ";W4#;"ist ";
80 LPRINT FNMITTELWERT(M1,M2)
999 END
```

```
Der Mittelwert aus 3 und 5 ist 4
Der Mittelwert aus  5 und -3 ist  1
Der Mittelwert aus 3, 5, 5 und -3 ist  2.5
```

Der Name der selbstdefinierten Funktion ist **FN**Mittelwert. **FN** ist einleitender Bestandteil einer selbstdefinierten Funktion und darf deshalb nicht am Anfang von Variablennamen verwendet werden! Mit **FN**Mittelwert unter Angabe der aktuellen Argumente (in Klammern) kann jederzeit die unter **DEF FN**Mittelwert rechts vom =-Zeichen vorher einmal im Programm definierte Operation abgerufen werden und steht als Funktionsergebnis zur Verfügung.

Besonders wichtig ist die Tatsache, daß die in der Liste der Argumente bei **DEF FN** verwendeten Variablennamen nicht den Inhalt gleichnamiger Variablen im Hauptprogramm verändern (sog. 'lokale' Variablen)!!

Selbstdefinierte Funktionen können nur einzeilig sein, daß heißt, die mit den Argumenten der Liste vorzunehmenden Operationen müssen hinter dem Gleichheitszeichen in Form eines Ausdrucks vorgenommen werden. Rechts vom =-Zeichen können zusätzlich zu den in der Argumente-Liste verwendeten Argumenten noch (Konstanten oder) Variablen verwendet werden, die für das Gesamtprogramm gelten (sog. 'globale' Variablen).

Beispiel: (selbstdefinierte numerische Funktion mit globaler Variable)

```
10 'kfm. Rundung auf die n-te Nachkommastelle
15 DEF FNR#(X#)=FIX(X#*10^NK.STELLEN%+.5*SGN(X#))/10^NK.STELLEN%
20 INPUT "gewünschte Anzahl NK-Stellen";NK.STELLEN%
25 LPRINT "Es wird auf";NK.STELLEN%;"Nachkommastellen gerundet."
30 INPUT "zu rundende Zahl";ZRZAHL#
40 LPRINT:LPRINT ZRZAHL#,FNR#(ZRZAHL#)
50 GOTO 30
Es wird auf 2 Nachkommastellen gerundet.

 2.755          2.76

-2.755         -2.76
```

Beispiel: (selbstdefinierte numerische Funktion)

```
 1 REM Berechnung der Anfangsposition zum Zentrieren eines
       Strings in ein Feld von F% Zeichen Länge
10 DEF FNSP0%(ZENTR$,FL%)=INT((FL%-LEN(ZENTR$))/2)+1
100 INPUT "Feldlänge";F%
120 INPUT "zu zentrierender String";S$
130 IF LEN(S$)>F% THEN PRINT "String zu lang":GOTO 120
135 LPRINT S$:LPRINT STRING$(F%,42)
140 LPRINT TAB(FNSP0%(S$,F%));S$
150 GOTO 100
in 60-Zeichen-Feld zentriert!
************************************************************
               in 60-Zeichen-Feld zentriert!
```

Beispiel: (selbstdefinierte String-Funktion)

```
 1 REM In S1$ wird S2$ durch S3$ ersetzt.
 2 REM Nur das erste S2$ von links wird ersetzt; S2$ muß in
       S1$ vollständig vorhanden sein, sonst Syntax error!
 3 REM S3$ darf durch das Ersetzen nicht länger als 255
 4 REM Zeichen lang werden!
10 DEF FNREP$(S1$,S2$,S3$)=LEFT$(S1$,INSTR(S1$,S2$)-1)+S3$
   +MID$(S1$,INSTR(S1$,S2$)+LEN(S2$))
15 A$="Lieferung mit Matrixdrucker":B$="Matrix":C$="Schnell"
20 LPRINT FNREP$(A$,B$,C$)

Lieferung mit Schnelldrucker
```

Beispiel: (selbstdefinierte String-Funktion mit Wahrheitswerten)

```
1 ' Aussortieren aller Strings aus einem String-Array T$(), die
    mit einer bestimmten Zifernfolge beginnen.
2 ' Für die Auswahl sind 1 bis 4 numerische Zeichen einzugeben.
    Die Strings, die mit diesen Zeichen beginnen, werden mit *
    am Ende markiert.
3 ' Die Auswahl geschieht mit Hilfe eines logischen Ausdrucks
    (Wahrheitswertes) in einer selbstdefinierten Funktion (eine
    Zeile')
4 '
5 ' hier: maximal 20 Orte
10 DIM T$(20):ZAEHLER%=0
15 DEF FNTSUCHE$(T$,X%)=T$+"      "+STRING$(-(VAL(LEFT$(T$,LEN(ST
R$(X%))-1))=X%),"*")
20 E$=CHR$(0)   'Anfangswert für WHILE/WEND-Bedingung
30 WHILE (LEN(E$) AND ZAEHLER%(21)
40       PRINT ZAEHLER%+1;:LINE INPUT", Eingabe (CR=Ende) ";E$
45       IF E$="" THEN 90      ' vorzeitiger Aussprung; Löschen
                                 des Stacks durch Setzen auf
                                 "Bedingung nicht erfüllt" und
                                 Sprung zu WEND
50       IF VAL(E$)(1000 OR VAL(E$))8999 THEN 40
60       ZAEHLER%=ZAEHLER%+1 : T$(ZAEHLER%)=E$
90 WEND
95 INPUT "zu markierende Anfangsziffern ";PLZ%
96 IF PLZ%(1 OR PLZ%)8999 THEN 95
99 LPRINT "auszugeben sind alle Orte, die mit";PLZ%;"beginnen:":
LPRINT:LPRINT
100 FOR I%=1 TO ZAEHLER%
110 LPRINT FNTSUCHE$(T$(I%),PLZ%)
120 NEXT
999 END
```

```
auszugeben sind alle Orte, die mit 6 beginnen:


6000 Frankfurt      *
6050 Offenbach      *
5000 Köln
6200 Wiesbaden      *
1000 Berlin
2000 Hamburg
```

4.3.8 Inputs über Tastatur

Daten können auf mehrere Arten über Tastatur eingegeben werden. Die Positionierung des Bildschirm-Cursors auf die gewünschte Eingabeposition am Bildschirm kann mit der Anweisung **CURSOR** erreicht werden. **CURSOR** POSITIONIERT NUR ; die Ausgabe bzw. Eingabe ab der gewünschten Position ist durch **PRINT** bzw. (**LINE**) **INPUT**-Anweisung zu programmieren.

1. **INPUT** und **LINE INPUT**

   Mit Hilfe der Anweisung **INPUT** können über Tastatur einer oder mehrerer Variablen einer Liste Werte zugewiesen werden. Die einzelnen Werte sind durch Kommas getrennt einzugeben. Dabei ist es nicht möglich, Anführungszeichen zu Bestandteilen von Stringeingaben zu machen. Im String enthaltene Kommas oder führende oder am Schluß stehende Blanks können nur zu Stringbestandteilen werden, wenn die Eingabe in Anführungszeichen eingeschlossen wird.

   Mit Hilfe der Anweisung **LINE INPUT** hingegen kann auf genau eine Stringvariable ein Tastatur-Input gemacht werden, wobei alle Zeichen - auch die oben erwähnten - übernommen werden. Die Eingabe kann danach selbstverständlich auf bestimmte Eigenschaften hin untersucht werden (Trennzeichen, rein numerische Eingabe) und via Stringverarbeitung zerlegt und ggf. in numerische Variablen verwandelt werden.

Beispiel:

```
10 'Folgende Eingabe soll erfolgen:
20 '     DM 20,70
30 'Dabei werden je 4 Blanks vor DM und nach 20,70 getippt
40 INPUT "Eingabe über INPUT: ",A$,B$
50 LINE INPUT "Eingabe über LINE INPUT: ";C$
60 LPRINT A$,LEN(A$):LPRINT B$,LEN(B$):LPRINT C$,LEN(C$)
70 'Beide Male wird dasselbe in " " eingeschlossen getippt
80 INPUT "über INPUT: ";E$
90 LINE INPUT "über LINE INPUT: ";F$
100 LPRINT E$,LEN(E$):LPRINT F$,LEN(F$)
999 END
DM 20          5
70             2
    DM 20,70                16
    DM 20,70                16
"    DM 20,70    "          18
```

Es ist besonders zu beachten, daß bei **INPUT** und **LINE INPUT** keine festen Begrenzungen der Eingabefelder möglich ist (keine Festlegung der maximal eingebbaren Zeichen), was dazu führen kann, daß ein gegebener Bildschirmaufbau vom Bediener zerstört werden kann (insbesondere bei Ausnutzung des Tastaturpuffers von 64 Zeichen!).

Ähnliches gilt für numerische Eingaben: Sind diese zu groß, um auf die betreffenden Variablen zugewiesen werden zu können, erscheint die Meldung "Overflow" in der nächsten Zeile. Der Overflow selbst kann in einer Fehlerbehandlungsroutine abgefangen werden (vgl. Kapitel 4.3.12), nicht aber die Meldung.

Bei Eingaben von nicht formatgerechten oder zu wenig Daten bei **INPUT** kann die Meldung "Redo from start?" ebenfalls nicht unterdrückt werden. Bei dieser Meldung muß die gesamte Eingabe(liste) für den betreffenden **INPUT** wiederholt werden.

Über **INPUT** und **LINE INPUT** angeforderte Eingabe(liste)n können erst nach Auslösen mit der Eingabe-Abschlußtaste (CR) verarbeitet werden.

Alle programmgesteuerten Tastatur-Inputs werden rein über Tastatur abgewickelt; eine Übernahme von am Bildschirm angezeigten Daten in den Tastaturpuffer ist nicht möglich.

Sollen am Bildschirm angezeigte Werte über Tastatureingaben des Bedieners geändert werden, müssen sie, falls mit **INPUT** oder **LINE INPUT** gearbeitet wird, vollständig eingegeben werden.

2. **INKEY$** und **INPUT$**

Die oben beschriebenen Probleme lassen sich durch Verwendung der Funktionen **INKEY$** und **INPUT$** lösen:

Mit **INKEY$** wird abgefragt, ob sich seit der letzten Abfrage des Tastaturpuffers durch **INPUT, LINE INPUT, INKEY$** oder **INPUT$(...)** der Inhalt des Tastaturpuffers geändert hat. Zu den abfragbaren Zeichen gehören z.B. auch CR und **H**. Ist bis zum Moment der Funktionsabfrage keine Veränderung des Tastaturpuffers erfolgt, ist das Ergebnis der Funktion der Leerstring (""). Anderenfalls ist es dasjenige Zeichen, das nach der letzten Abfrage als erstes im Tastaturpuffer gespeichert wurde. Die nächste Abfrage des Tastaturpuffers wird ggf. das nächste Zeichen liefern. (Der Tastaturpuffer kann max. 64 Zeichen fassen, alle weiteren gehen verloren. Nach Empfang des 58. Zeichens ertönt ein automatisches Signal.)

Bei **INKEY$** muß in einer Schleife abgefragt werden, wann eine Taste betätigt wurde, da das System nicht abwartet, bis eine Taste betätigt wurde. Das Ergebnis von **INKEY$** wird nicht automatisch am Bildschirm angezeigt, ebensowenig wie das von **INPUT$.**

Beispiel:

```
' REM Eingabe einer bis zu 5stelligen ganzen Zahl
  REM Löschen von Eingabezeichen mit CTRL H hier nicht möglich
  ) FOR I%=1 TO 5
40 K$=INKEY$
50 IF K$="" THEN 40
60 IF ASC(K$)=13 THEN I%=5:GOTO 90
62 IF K$<"0" OR K$>"9" THEN 40
70 PRINT K$;
80 S$=S$+K$
90 NEXT I%
95 PRINT
100 PRINT VAL(S$)
999 END
```

Das Ergebnis von **INKEY$** ist das Zeichen, das sich als erstes seit der letzten Abfrage im Tastaturpuffer befindet.

Bei **INPUT$** muß eine genau festgelegte Anzahl von Zeichen (mindestens eins) eingegeben werden, wobei auch CR und ■H mitzählen. Neben der Tatsache, daß mehr als ein Zeichen festgelegt werden kann, unterscheidet sich **INPUT$** von **INKEY$** dadurch, daß das System abwartet, bis die vorgeschriebene Anzahl Zeichen eingegeben wurde und erst dann mit der weiteren Abarbeitung des Programms fortfährt.

Beispiel:

```
1 REM Eingabe einer bis zu 5stelligen positiven ganzen Zahl
2 CLS:CURSOR(1,2):PRINT "Ihre Eingabe: ";
5 'vorzeitiges Ende durch CR; nach 5 Zeichen automatisch Ende
20 KEINAUSSPRZ=-1
30 WHILE LEN (S$)<5 AND KEINAUSSPRZ
40 K$=INPUT$(1)
60 IF ASC(K$)=13 THEN KEINAUSSPRZ=0:GOTO 90
62 IF ASC(K$)<>8 THEN 66 ' CHR(8) IST CONTROL H
64 IF LEN(S$) THEN S$=LEFT$(S$,LEN(S$)-1):CURSOR(15,2):PRINT SPA
CE$(6):CURSOR(15,2):PRINT S$
66 IF K$<"0" OR K$>"9" THEN 40
68 CURSOR(15,2):S$=S$+K$:PRINT S$
90 WEND:PRINT:PRINT VAL(S$):END
```

INPUT$ kann auch zum Einlesen von Zeichen aus externen Datenfiles oder von einer V.24-Schnittstelle bei Wechsel des Ein/Ausgabe-Mediums verwendet werden (vgl. Kapitel 4.3.10.2, PCOS-Handbuch und V.24-Programmierhandbuch).

Sollen vor **INPUT$, INKEY$** oder **INPUT** bzw. **LINE INPUT** eventuelle Vorauseingaben im Tastaturpuffer gelöscht werden, bietet sich eine Routine der Art

```
7140 IF INKEY$<>"" THEN 7140
```

an.

3. Sonder-Funktionstasten und Eingabeabschlußtasten

Einige Tasten haben besondere Funktionen, z.B. [H] und die normale Eingabeabschlußtaste sowie die Tasten S1 und S2.

Außerdem existieren die Cursor-Steuertasten (vgl. Kapitel 1.3.5). Diese Tasten steuern den Cursor selber nicht, sondern erzeugen nur bestimmte ISO-Codes. In Abhängigkeit von diesen Codes ist per BASIC-Programm jeweils die betreffende **CURSOR**-Positionierung vorzunehmen (**CURSOR**-Anweisung).

Unter Verwendung der Funktionen **INKEY$** bzw. **INPUT$** lassen sich alle Tasten abfragen.

Hinweis:

Alle drei Eingabeabschlußtasten haben den ISO-Code 13 (CR). Die ISO-Codes, die den übrigen Tasten zugeordnet sind, können der Tabelle in Kapitel 4.3.17 entnommen werden.

Alle über **INKEY$** bzw. **INPUT$** erzeugten Zeichen werden nicht automatisch am Bildschirm angezeigt und müssen deshalb - falls nötig - mit **PRINT** am Bildschirm ausgegeben werden.

Soll abgefragt werden, welche der drei Eingabeabschlußtasten betätigt wurde, muß nach der betreffenden **INPUT** oder **LINE INPUT** Anweisung bzw. **INKEY$**- oder **INPUT$**-Funktion die Assembler-Routine **lt** mit Hilfe der Anweisung **CALL** aufgerufen werden. Als Argument in der Klammer ist - nach dem Präfix **§** - eine Integer-Variable anzugeben, die nach Ausführung der **CALL**-Anweisung Aufschluß darüber gibt, mit welcher Abschlußtaste die Eingabe abgeschlossen wurde, bzw. mit welcher Taste das Ergebnis CR(=**CHR$(13)**) bei Anwendung einer der Funktionen **INKEY$** bzw. **INPUT$** erzeugt wurde.

Die Integer-Variable enthält:

0 bei Abschluß mit ↵
1 bei Abschluß mit S1
2 bei Abschluß mit S2

Bei DES-Versionen enthält die Variable (nach Verwendung des PCOS-Befehls **sl** 16) außerdem:

3 bei Abschluß mit S3
4 bei Abschluß mit S5
5 bei Abschluß mit S6

Beispiel:

```
1000 'Verzweigung in Abhängigkeit von der Abschlußtaste
1001 CLS:TASTE%=-1 'Adresse für Variable im RAM schaffen
1005 PRINT TAB(20)"Bitte S,S1 oder S2 drücken"
1020 K$=INKEY$:IF K$="" OR K$()CHR$(13) THEN 1020
1030 CALL "lt"(@TASTE%):PRINT TASTE%;"==) ",
1050 ON TASTE%+1 GOSUB 2010,3010,4010:GOTO 1005
2010 PRINT "normale Abschlußtaste CR":RETURN
3010 PRINT "S1":RETURN
4010 PRINT "S2":RETURN
```

Es ist zu beachten, daß die hinter § angeführte Integervariable vor dem ersten Aufruf durch eine wertzuweisende Anweisung einen beliebigen Wert erhalten (z.B.
TASTE%=Ø im Vereinbarungsteil).

Beispiel für eine andere Form des Text-Cursors:

```
1 CLEAR ' neue Form des Cursors
10 OPTION BASE 1
20 DIM A%(6)
30 DATA 4152          :' 00010000
31                     ' 00111000
32 REM      ---------              ----
33 DATA 31998         :' 01111100
34                     ' 11111110
35 REM      ---------              ----
36 DATA 14392         :' 00111000
37                     ' 00111000
38 REM      ---------              ----
39 DATA 14392         :' 00111000
40                     ' 00111000
41 REM      ---------              ----
42 DATA 0             :' 00000000
43 REM                 ' 00000000
44 REM      ---------              ----
45 DATA 0             :' 00000000
46 REM      ---------  ' 00000000 ----
50 FOR I%=1 TO 6:READ A%(I%):NEXT I%
54 ' beide Cursor-Punktzeilen = -1
55 ' unterste Cursor-Punktzeile = 255
60 CURSOR(31,10)1,15,A%(1)
68 WHILE A$()CHR$(13)
70 A$=INPUT$(1):PRINT A$;
80 WEND
90 ' keine neue Cursor-Form bei (LINE) INPUT !!
999 END 'END und CLEAR heben neue Form auf.
```

Sollen im Bildschirm angezeigte Daten direkt geändert werden, wird wie folgt vorgegangen:

1. Das angezeigte, zu ändernde Datum in einer Stringvariablen speichern;
2. Jede einzelne Tastenbetätigung abfragen (Zuweisung auf eine Hilfsvariable über **INKEY$** bzw. **INPUT$**);
3. Unterschieden, ob eine Cursor-Steuertaste gedrückt wurde oder CR oder ein echtes Zeichen zur Änderung eingegeben wurde;
4. Je nach dem Ergebnis von 3.:
   - den Inhalt der Stringvariablen, in der das zu ändernde Datum steht, entsprechend anpassen (Stringverarbeitung);
   - die geänderte Stringvariable am Bildschirm ausgeben (nach **CURSOR**-Anweisung);
   - den Cursor auf die erforderliche neue Stelle positionieren;
   - Änderungsroutine verlassen (bei CR) bzw. zurück zu 2.
5. Eventuell abfragen, mit welcher Abschlußtaste schließlich der Aussprung aus der Routine 1 - 4 bewirkt wurde.

Die Cursor-Steuertasten besitzen die folgenden ISO-Codes:

Steuertasten	ISO-Code (dezimal)
↑	158
↓	154
→	157
←	155
HOME	156

Zur Übersicht aller ISO-Codes vgl. Tabelle in Kapitel 4.3.17.

Jm kontrollierte Eingaben genauer steuern zu können (Feldbegrenzung, Vorgabemöglichkeit, Cursorsteuerung, gewisse Zulässigkeitsprüfungen schon während numerischer Eingaben - z.B. Vor/Nachkommastellen und Intervalle) existieren zwei Möglichkeiten:

a) ein BASIC-Modul in Form eines ASCII-Files namens T2Ø5ØØ (beginnend bei Zeilennr. 2Ø5ØØ bis 2Ø599), das mit **MERGE** in jedes BASIC-Programm eingebunden werden kann.
   Für jeden Input sind durch Wertzuweisungen die entsprechenden Eingangsparameter zu setzen, mit **GOSUB** 2Ø5ØØ die Routine aufzurufen und anschließend die Ausgangsparameter abzusichern. Zur genaueren Beschreibung vgl. Kapitel 7, Anhang, Punkt I .

b) eine Assemblerroutine in Form eines PCOS-Befehls namens kboard.cmd, der mit dem PCOS-Befehl **pl** im Nicht-BASIC-Bereich des Speichers resident gemacht werden kann. Der Aufruf geschieht aus BASIC mit Hilfe der **CALL**-Anweisung, wobei in Klammern die Ein- und Ausgangsparameter in Form von Variablen zu definieren sind. Vgl. dazu PCOS-Handbuch, Beschreibung von **kb**.

4.3.9 Outputs

4.3.9.1 **Bildschirm** (alphanumerisches Arbeiten)

Die Ausgabe von Tastatureingaben über **INPUT** bzw. **LINE INPUT** erfolgt direkt am Bildschirm. (Diese Ausgaben werden auch von einer **WIDTH**-Anweisung nicht beeinflußt.) Die Ergebnisse der Funktionen **INKEY$** bzw. **INPUT$** werden nicht angezeigt. Darüberhinaus können Daten mit Hilfe der Anweisungen **PRINT** und **WRITE** am Bildschirm angezeigt werden.

Beispiel zu **PRINT** und **WRITE**

```
5000 REM PRINT / WRITE (Unterschied)
5001 CLS
5010 'eingegeben wird: DM 20,70
5020 INPUT "INPUT: ";A$,B
5030 LINE INPUT "LINE INPUT: ";C$
5040 PRINT STRING$(25,"*")
5050 PRINT A$;B;C$
5060 WRITE A$;B;C$
5099 END

INPUT: ? DM 20,70
LINE INPUT: DM 20,70
*************************
DM 20** 70 **DM 20,70**
"DM 20","**",70,"**","DM 20,70","**"
```

Der hier sichtbare Unterschied ist wesentlich zur Unterscheidung der File-Anweisungen **PRINT#** und **WRITE#** in Kapitel 4.3.10.2.

WRITE bewirkt, im Gegensatz zu **PRINT**, die automatische Einfügung eines Trennzeichens (,) zwischen den auszugebenden Daten. Darüber hinaus werden alle Strings automatisch dadurch gekennzeichnet, daß sie in Anführungszeichen ("") eingeschlossen werden. Die Anführungszeichen werden jedoch nicht zu eigentlichen Bestandteilen der Strings.

Bei Verwendung von **PRINT** ist es möglich, eine Zeilenschaltung (LF) dadurch zu unterdrücken, daß am Ende der Liste der Ausgabedaten ein Strichpunkt (;) gesetzt wird.

Wird in der letzten Zeile des Bildschirms bzw. des aktiven Windows eine Ausgabe mit **PRINT** angefordert, wird immer ein Roll-Up (Scrolling) des Bildschirms auftreten, es sein denn, am Ende der **PRINT**-Anweisung steht ein ;.

Eine Formatierung der Ausgaben, die weitergeht als mit **TAB**, (z.B. rechtsbündige Darstellung von numerischen Daten) ist möglich entweder durch Stringverarbeitung oder aber durch Verwendung des Anweisungsbestandteils **USING** (vgl. Ende dieses Kapitels).

Durch Ausgabe des ISO-Codes 7 über den Bildschirm (**PRINT CHR$(7)**) kann ein akustisches Signal erzeugt werden. Steht am Ende einer solchen Anweisung kein ;, bewirkt sie natürlich noch zusätzlich eine Zeilenschaltung (LF).

Beim Arbeiten mit Texten (alphanumerisches Arbeiten) ist der Bildschirm - in Abhängigkeit von im PCOS-Befehl **ss** gesetzten Parameter 'Bildschirmformat'-wie folgt aufgeteilt:

Version 1: 64x16 (16 Textzeilen à 64 Spalten) (Default)
Version 2: 80x25 (25 Textzeilen à 80 Spalten)

Das Format kann auch mit Hilfe der **WINDOW**-Funktion geändert werden.

Die Darstellung der Zeichen basiert auf der Einteilung des Bildschirms in ein Punktraster von Elementarpunkten.

Bei Version 1 liegen 512 Punktspalten und 256 Punktzeilen zugrunde, bei Version 2 hingegen 480 Punktspalten und 256 Punktzeilen.

Mit der **WIDTH**-Anweisung kann die Ausgabebreite auf dem Bildschirm (Anzahl Textspalten) festgelegt werden. Die **WIDTH**-Anweisung wird weder durch **NEW** noch durch **RUN** oder **CHAIN** aufgehoben, sondern nur durch eine neue **WIDTH**-Anweisung! Außerdem läßt sich der Bildschirm per Programm in bis zu 16 verschiedene rechteckige Windows einteilen, von denen jedes einzelne getrennt angesprochen werden kann. Jedes einzelne Window kann mit bestimmten Attributen (Zeilenhöhe, Spaltenbreite, Hinter- und Vordergrundfarbe) ausgestattet werden. Jedes Window hat einen selbständigen Text-Cursor mit einer eigenen Spalten-Zeilen Adressierung bzw. einen eigenen Graphik-Cursor und eigene Skalierung. Zur Arbeit mit Windows vgl. Kapitel 4.3.18; zur graphischen Arbeit vgl. Kapitel 4.3.14. Bei der Arbeit mit Windows ist zu beachten:

- Nur die Anweisungen **CLOSE WINDOW** oder **CLEAR** lösen eine vorherige Aufteilung des Bildschirms in Windows auf.
- Auch für Windows werden Default-Werte durch PCOS-Befehle gesetzt, und zwar durch **sb** und **ss**.

Mit der Anweisung **CURSOR** kann der Text-Cursor auf bestimmte Textspalten und -zeilen innerhalb des gesamten Bildschirms (WindowNr. 1) oder innerhalb eines Windows gesetzt werden.

Beispiel:

```
10 CALL "ss %n,,,,,1":CLEAR:W%=WINDOW(0,150)
20 INPUT "Window-Nr.(1 oder 2), Zeichenbreite(6 oder 8) ",W%,Z%
21 IF W%<1 OR W%>2 OR Z%<>6 AND Z%<>8 THEN 20
24 WINDOW%W%:CLS:W%=WINDOW(0,0,,Z%):CURSOR(1,6)
25 INPUT "Druckposition (Spalte) ";D%:IF D%<0 THEN 25
26 IF (D%>60 AND Z%=8) OR (D%>80 AND Z%=6) THEN 25
30 'Da wegen ss (80er-Format) nur 480 Punkte zur Verfügung
    stehen, gehen nur 60 Zeichen (60*8) in eine Zeile!
40 FOR I%=1 TO 6
45 CURSOR(D%,I%)
50 PRINT "Dies ist Window Nr.";W%;"Ausgabe ab Spalte";D%;
60 NEXT I%:PRINT:CURSOR(1,6):GOTO 20
```

Zur Positionierung des Graphik-Cursors vgl. Kapitel 4.3.14.

Es ist zu beachten, daß bei einer Window-Aufteilung des Bildschirms die Text-Cursor-Positionen sich auf das jeweilige Window beziehen. Jedes Window kann somit als ein "eigener kleiner Bildschirm" interpretiert werden.
Die Position des Text-Cursors ist stets in der Form (Textspalte, Textzeile) anzugeben. Die oberste Zeile des Bildschirms (bzw. des betreffenden Windows) hat die Nr. 1, die Spalte am weitesten links ebenfalls die Nr. 1. Tiefere bzw. weiter rechts liegende Positionen haben entsprechend höhere Nummern.
Mit Hilfe der Funktion **POS** kann die aktuelle Position (Zeile oder Spalte) des Text-Cursors im Bildschirms bzw. in einem Window abgefragt werden.

Alle Ausgaben, deren Länge den rechten Rand des Bildschirms bzw. eines Windows überschreiten, werden automatisch in der nächsten Zeile ab dem linken Rand fortgesetzt.

Zur Berechnung der zu reservierenden Ausgabeposition bei nicht über **USING** formatierten numerischen Daten, ist das Standardformat zugrundezulegen.

Dabei gilt:

1. Kommazahlen werden stets mit einem Dezimalpunkt ausgegeben.
2. Bei Zahlen im Intervall $-1 < X < 1$ wird keine Ø vor dem Dezimalpunkt ausgegeben.
3. Nullen nach dem Dezimalpunkt werden nie ausgegeben, falls auf sie keine anderen Ziffern mehr folgen.
4. Hinter der letzten ausgegebenen Ziffer wird grundsätzlich zusätzlich ein Blank ausgegeben.

5. Zahlen, die zuviele signifikante Stellen für ihren Typ haben, werden automatisch im Gleitkomma- (bzw. Gleitpunkt)-Format ausgegeben, also in der Form

Hochzahl zur Basis 10, mit der die Mantisse multipliziert werden muß

Beispiel:

```
0 'Test auf "normale" numerische Eingabe
10 CLS:LINE INPUT"numerische Eingabe ";A$:GOSUB 51
20 PRINT "!";A$;"!!";VAL(A$);"!":K$=INPUT$(1):GOTO 10
51 IF A$="" THEN A$="0":RETURN
52 T$=STR$(VAL(A$)):IF LEFT$(T$,1)=" " THEN T$=MID$(T$,2)
53 FOR I%=1 TO LEN(A$)
54 IF INSTR("dDeE&hHoO",MID$(A$,I%,1)) THEN I%=LEN(A$)+1
56 NEXT 'I% = LEN(A$)+2 bei Fehler, sonst I% = LEN(A$)+1
58 IF I%=LEN(A$)+1 AND A$=T$ THEN RETURN
60 IF LEFT$(A$,1)="+" AND MID$(A$,2)=T$ THEN RETURN
61 PRINT "unzulässig":RETURN
```

Werden numerische Daten mit der Funktion **STR$** konvertiert, gilt für das Ergebnis das eben Gesagte, mit der Ausnahme, daß 4. entfällt!

Werden am Bildschirm Strings ausgegeben, die nicht druckbare Zeichen (ISO-Codes Ø bis 31 bzw. über 126) enthalten, werden diese übergangen (nicht angezeigt). Die auf sie folgenden druckbaren Zeichen werden unmittelbar darauf ausgegeben. Die Länge der Anzeige weicht dann von der aktuellen Länge des Strings (**LEN**) ab.

Einige dieser Zeichen haben jedoch Wirkung am Bildschirm, wenn sie per **WRITE** oder **PRINT** ausgegeben werden:

ISO-Code	Wirkung
7	akustisches Signal (ohne Zeichenausgabe)
8	Löschen des vorherigen Zeichens und Positionierung des Cursors um eine Stelle nach links. Im Gegensatz zu ■ werden die Zeichen, um die rückpositioniert wurde, nicht gelöscht.
9	Tabulation auf die nächste der Spalten 1, 9, 17, 25 etc.
10	(LF) Positionierung des Cursors auf gleiche Spaltenposition eine Zeile tiefer
12	Löschen Bildschirm bzw. Window und Positionierung des Cursors auf Position (1,1)
13	CR (Zeilenrücklauf)

Ab Release 2.Ø ist der Bildschirmzeichensatz für die ISO-Codes 32 ff. abhängig von der gewählten nationalen Tastatur (vgl. PCOS-Befehl **sl**) und einem eventuell zugrundeliegendem Tastaturfile (vgl. PCOS-Befehle **rf** und **wf**).

Zur Formatierung von Daten (numerisch oder String) dient der Anweisungsbestandteil **USING**. Dabei ist in einem Formatstring, der sich aus bestimmten "Platzhalter"symbolen zusammensetzt, vorzuschreiben, wie die Daten zu formatieren sind (Definition der "Masken"). Die zu formatierenden Daten sind am Ende der Anweisung in einer Liste anzugeben.

Beispiel:

```
16 WIDTH LPRINT 64:Z$="1234567890":Z$=Z$+Z$+Z$+Z$+Z$+Z$+Z$+"1234"
20 A#=123.456:B#=-123.456:C#=.45a:D#=1.5752350+22
30 A$="Alpha-Feld 1 ":B$="2. Alpha-Feld":LPRINT Z$
40 LPRINT USING "######.#### = A_# und ######.#### = B_#";A#;B#
60 LPRINT Z$:LPRINT USING "###.# = A_# und ###.# = B_#";A#;B#
70 LPRINT Z$
80 LPRINT USING "#####,###.##(D_#);#####.#####^^^^(D_#)";D#,D#;
100 LPRINT USING"+#.###### = C_# und ö          ö = B_$";C#;B$
101 INPUT "Wieviele * sollen eingefügt werden? ",N%
102 IF N%<0 OR N%>35 THEN 101
103 LPRINT USING "-----&-----";MID$(Z$,6,54)
110 LPRINT USING "1234566          601&Das vorige Feld ist vari
abel lang -- hier: ## Zeichen";A$;STRING$(N%,"*");N%


12345678901234567890123456789012345678901234567890123456789012345678901234
  123.4560 = A# und  -123.4560 = B#
12345678901234567890123456789012345678901234567890123456789012345678901234
123.5 = A# und %-123.5 = B#
12345678901234567890123456789012345678901234567890123456789012345678901234
% 1575235.00D+16(D#): 1575.23500D+019(D#) +.456000 = C# und 2. A
lpha-Fel = B$
-----67890123456789012345678901234567890123456789012345678901234567890-----
1234566Alpha-Feld 1 61**********************Das vorige Feld ist variabel
 lang -- hier  15 Zeichen
```

Eine Liste zu formatierenden Daten kann durch ; abgeschlossen sein. Dadurch wird die Folgeausgabe ab der nächsten freien Position bewirkt.

Der Formatstring kann auch mit Hilfe von Variablen zusammengesetzt sein (z.B. über Stringverarbeitung oder mit Hilfe von selbstdefinierten Stringfunktionen).

Beispiel:

```
1 PRINT "Artikelnummer (5 Zeichen), Bezeichnung (bis zu 25"
2 PRINT "Zeichen), Lagernummer (1-10)"
3 INPUT "",ART$,BEZ$,LNR%
10 PRINT "Bitte Maske eingeben ":LINE INPUT MASKE$
20 REM Prüfung entfällt hier
30 LPRINT MASKE$
40 LPRINT ART$;"/";BEZ$;"/";LNR%
50 LPRINT USING MASKE$;ART$;BEZ$;LNR%
99 END
```

```
Artikel ö    ö Name ö                              ö  Lager ##
abcde/Strümpfe/ 9
Artikel abcde Name Strümpfe                          Lager  9
```

Außerdem kann derselbe Formatstring gleichzeitig zur Ausgabe mehrerer, gleich zu formatierender Daten benutzt werden.

Beispiel:

```
0 CLS:DATA 5100.5 97.603,2700,25,60,138,85,50,97,7416
2 DEFINT I-K:CALL "ss %n,,,,,0"
10 MASKE$="##,### <--> ##,### * "
12 FOR KA=1 TO 2
15 FOR K=1 TO 3:READ I(K),J(K):NEXT K
20 PRINT USING MASKE$;I(1),J(1),I(2),J(2),I(3),J(3)
30 NEXT KA:WIDTH LPRINT 64:EXEC "ls" 'Hardcopy des Textes
```

```
 5,100 <-->       5 *     97 <-->     603 *  2,700 <-->      25 *
    60 <-->     138 *     85 <-->      50 *     97 <-->   7,416 *
```

4.3.9.2 Drucker

Daten können auch am Drucker ausgegeben werden. Es ist dann statt **PRINT** die Anweisung **LPRINT** und statt **PRINT USING** die Anweisung **LPRINT USING** anzuwenden.

Die am Drucker auszugebenden Zeichen können sich von den Zeichen auf dem Bildschirm unterscheiden, falls der Zeichensatz des Druckers von dem des Bildschirms abweicht. Die Art des Drucks (Fettdruck, 10-Zeichen-Zoll-Druck etc.) sowie Vorschriften über Zeilenschaltung u.ä. können über die Ausgabe von Steuerzeichen (vgl. jeweilige Druckerhandbücher) mit **LPRINT** vorgeschrieben werden.

Beispiel:

```
1 CALL "sf %n,,pr1471,0":WIDTH LPRINT 64
2 CLS:T$="Dieser Text ist auszugeben"
8 A$=MKS$(150):LPRINT "A=150; MKS$(A) wird ausgegeben als !";A$;
"! . Die aktuelle Länge ist jedoch";LEN(A$)
9 'über MKI$,MKS$,MKD$ gewonnene Strings sind immer 2, 4 oder
  8 Bytes lang, aber nicht sinnvoll druckbar. Sie beinhalten
  das NUMERISCHE Bitmuster des Wertes, das sich NICHT mit den
  ASCII-Codes der einzelnen Ziffern deckt!!
10 LPRINT T$; 'bleibt wegen ;im Druckpuffer!
15 PRINT "Bitte CR drücken"
20 K$=INPUT$(1):LPRINT CHR$(10) 'Ausgabe Druckpuffer
25 LPRINT CHR$(27)+CHR$(61)'Schalten auf 12 Zeichen/Zoll
                            + Zeilenschaltung (kein :)
30 LPRINT "Nach diesem Text erfolgt ein Seitenvorschub"+CHR$
   (12)
```

```
A=150; MKS$(A) wird ausgegeben als !ÇW
! . Die aktuelle Länge ist jedoch 4
Dieser Text ist auszugeben


Nach diesem Text erfolgt ein Seitenvorschub
```

Über eine Hardware-Schaltung kann festgelegt werden, ob für die übrigen nicht druckbaren Zeichen ||||| ausgegeben werden soll oder ihre Ausgabe (wie am Bildschirm) übergangen werden soll.

Es ist zu beachten, daß auch für den Drucker mit Hilfe der Anweisung **WIDTH LPRINT** die Zeilenbreite per Programm festgelegt werden kann, die wiederum mit **NEW, RUN** oder **CHAIN** nicht aufgehoben werden kann.

Eine Hardcopy (Abbildung des aktuellen Bildschirmtextes und aller Graphik auf dem Drucker) kann mit Hilfe der Anweisung

CALL "sp"

erzeugt werden. Dies gilt jedoch nur für graphikfähige Drucker (z.B. nicht PR 1471).

Die Ausgabe nur der Texte im Bildschirm kann für alle Drucker mit **CALL "ls"** abgerufen werden.

Mit dem PCOS-Befehl **sf** werden verschiedene Default-Werte für den Drucker gesetzt.

Weitere Informationen über die Drucker finden Sie in Kapitel 5.1.

4.3.10 Externe Datenfiles

Neben den Programmfiles und ASCII-Files (vgl. BASIC-Befehl **SAVE**) stehen mehrere Arten von Datenfiles zur Verfügung.

Datenfiles können mit dem PCOS-Befehl **fn** angelegt werden. Die gewünschte Größe (Anzahl dafür zu reservierende 256-Byte-Sektoren auf Diskette) ist dabei anzugeben.

Es ist aber auch möglich, Datenfiles nur mit Hilfe der BASIC-Anweisung **OPEN** (z.B. im Programm) anzulegen. Die Anzahl der Sektoren, die dabei reserviert werden, ist abhängig von der im PCOS-Befehl **ss** definierten Anzahl Sektoren, die bei automatischer Anlage bzw. Vergrößerung zugrundegelegt wird.

Wird es im Verlauf von Programmen notwendig, die physische Länge von Datenfiles zu vergrößern, erledigt dies das System selbsttätig. Es ist allerdings Voraussetzung, daß noch genügend Platz auf der Diskette frei ist. Die Anzahl der Sektoren, um die das System versucht, das File physisch zu vergrößern, ist ebenfalls gleich dem im PCOS-Befehl **ss** gesetzten Parameter 'anzulegende Selektoren'. Ist eine Vergörßerung um diese Anzahl Sektoren (bzw. ein Vielfaches davon) nicht möglich, wird "Disk full" gemeldet, auch wenn für die abzuspeichernde Information selbst der Platz nicht ausreichen würde.

Optimale Blockungsfaktoren brauchen nicht errechnet zu werden, da die logischen Records von den Grenzen der physischen Sektoren oder Spuren in keiner Weise beeinflußt werden. Es ist also möglich, beliebige Record-Längen (maximal 4096 Bytes) festzulegen, ohne dabei Speicherplatzverluste auf Diskette befürchten zu müssen. In einem Programm können gleichzeitig maximal 15 Files verwaltet werden. Die Recordlänge von sequentiellen Files ist - von der Disketten-Kapazität abgesehen - nicht von Bedeutung. Die Record-Länge von Random-Files beträgt 1 bis max. 4096 Bytes. Das Betriebssystem richtet im Arbeitsspeicher selbständig physische Puffer ein. Anzahl und Größe können durch den PCOS-Befehl **sb** definiert werden.

Die maximale File-Anzahl in jedem Programm und die maximale Record-Länge der Random-Files (eigentlich die Anzahl und die physische Größe der File-Puffer) wird mit dem PCOS-Befehl **sb** festgelegt. Es ist zu beachten, daß bei einer in **sb** definierten maximalen Record-Länge vor Programmstart sofort für jedes Random-File ein eigener Puffer in der angegebenen Länge reserviert ist, wodurch sich der für BASIC zur Verfügung stehende Speicherplatz im RAM entsprechend vermindert.

Im Arbeitsspeicher werden sowohl für alle angesprochenen Disketten als auch für alle geöffneten Files Tabellen verwaltet, in denen der Zustand der Disketten bzw. Files festgehalten ist.

Das System prüft bei geöffneten Files in regelmäßigen Abständen, ob die Diskette gewechselt worden ist (ca. alle 3 Sekunden). Wird dies festgestellt, wird der Fehler "Illegal disk change" gemeldet (Fehler-Code 74). Wird die betroffene Diskette wieder eingelegt, kann weitergearbeitet werden.

Nach dem Öffnen von Files wird vom Betriebssystem kein Byte auf der Diskette gesetzt, das das File als physisch offen kennzeichnet.

4.3.10.1 Organisationsmöglichkeiten

Folgende File-Typen stehen als externe Datenfiles zur Verfügung:

1. Textzeilen mit Zeilennummer, die in Form von ASCII-Files abgespeichert sind;
2. Sequentielle Datenfiles mit fester oder variabler Recordlänge;
3. Random-Files mit fester Record-Länge, deren Records unter ihrer Nummer aufgrufen werden (sog. Direct-Files)

4. Index-sequentielle Files; für die Verarbeitung von indexsequentiellen Files wird das Zusatzpaket ISAM benötigt. Es wird eine Assembler-Routine eingesetzt, die die Verwaltung der Keys übernimmt. Dabei bestehen folgende Möglichkeiten:
   - Länge des alphanumerischen Zugriffskriteriums (Key) bis 110 Zeichen (variable Länge möglich)
   - Sequentielles Lesen in logischer Folge
   - Suche des nächsten oder des vorherigen Keys
   - Löschungsmöglichkeit des Records und/oder des Keys (ohne Reaktivierungsmöglichkeit)
   - partieller Key
   - sekundäre Keys (bis zu 5)
   - doppelte Keys auf Wunsch

1. ASCII-Files

ASCII-Files sind wie sequentielle Files zu behandeln. Sie können wie diese gelesen und beschrieben werden. Allerdings können sie auch im Arbeitsspeicher editiert werden (Textzeilen mit vorangestellten Zeilennummern) und dann mit **SAVE "Filename",A** gespeichert werden. Am Ende des Records (mit Zeilennummer versehene Zeile) werden bei Speichern vom System selbst solche Trennzeichen gesetzt, die bei **LINE INPUT#** dazu führen, daß jede Textzeile als Record erkannt wird. Somit kann jede Zeile sehr einfach einer Stringvariablen zugewiesen und gegebenenfalls mit **PRINT#** in ein anderes sequentielles File geschrieben werden.

2. Sequentielle Files

In sequentiellen Files sind die Daten lückenlos aufeinander folgend gespeichert. Am Ende des geschriebenen Records werden vom System automatisch die Trennzeichen **CHR$(13)** und eventuell **CHR$(10)** gesetzt.

Die Datenelemente eines Records müssen, um identifiziert werden zu können, durch Trennzeichen voneinander getrennt geschrieben werden. Die Abbildung der Daten auf das File geschieht völlig analog zur entsprechenden Bildschirmanweisung (**PRINT - PRINT#; WRITE-WRITE#; PRINT USING - PRINT# USING**). Sequentielle Files können vom Programm in jeweils einem der folgenden Arbeitsmodi verwendet werden:

a) Lesen von Anfang an (Input-Mode, Lese-Mode)
b) Schreiben von Anfang an (Output-Mode, Schreib-Mode)
c) Schreiben anschließend an bestehenden Inhalt (Append-Mode)

Beim Öffnen des Files wird festgelegt, in welchem dieser Modi gearbeitet wird. Eine Änderung ist über Schließen des Files und neuerliches Öffnen im gewünschten Modus möglich.

Während ein sequentielles File geöffnet ist, wird vom System ein Pointer für dieses File geführt, der entsprechend dem Modus auf die Stelle im File zeigt, ab der beim nächsten Zugriff gelesen bzw. geschrieben wird. Eine frei wählbare Änderung der Pointerposition innerhalb des Files ist beim Schreiben nicht und beim Lesen nur durch Setzen an den Anfang mit Schließen und neuerlichem Öffnen möglich.

zu a) Lesen des sequentiellen Files (Input-Mode)

Nach dem Öffnen des Files zeigt der Pointer auf das erste Datenelement im File. Die Daten können in der Reihenfolge ihrer Speicherung, entsprechend ihrem Typ, gelesen werden.

Das Zurücksetzen des Pointers an die erste Stelle im File ist mit **CLOSE** und neuem **OPEN** möglich. Das Betriebssystem erkennt beim Lesen das Ende eines sequentiellen Files an einem bestimmten Trennzeichen. Dazu kann die Funktion **EOF** eingesetzt werden. Mit der Funktion **LOC** kann der Sektor abgefragt werden, in dem zuletzt gelesen oder geschrieben wurde. Mit der Funktion **LOF** kann ermittelt werden, in welchem Sektor das jeweils zuletzt geschriebene Datenelement steht (**EOF**-Kennzeichen).

zu b) Schreiben in sequentielle Files von Beginn an (Output-Mode)

Nach dem Öffnen des Files wird der Pointer an die erste Stelle im File gesetzt und der bisher bestehende Inhalt dadurch nicht mehr verfügbar. Beim Schreiben von Datenelementen wird jedes Element an das vorangehende angeschlossen und der Pointer an die nächste freie Stelle gesetzt.

zu c) Anhängen an bestehende Daten (Append-Mode)

Dieser Arbeitsmodus erlaubt das Hinzufügen von Daten an ein bestehendes File. Es wird nach dem Öffnen der Pointer an die erste freie Stelle im File gesetzt und anschließend wie im Schreibmodus verfahren.

Das Schreiben des Puffers aufs File geschieht erst, wenn der Puffer gefüllt ist. Nur die Anweisung **CLOSE** stellt sicher, daß sofort der gesamte Pufferinhalt auf das File geschrieben wird!!

3. Random-Files

Random-Files erlauben den direkten Zugriff auf Daten. Um diesen direkten Zugriff zu ermöglichen, sollten die Daten in Records logisch organisiert werden. Die Records bilden die Einheit, über die vom Programm auf Daten der Diskette zugegriffen werden kann.

Der Zugriff erfolgt auf gleiche Weise ("Random"), unabhängig davon, ob nachfolgend Daten gelesen oder verändert werden sollen. Der Zugriff auf einen Record des Files erfolgt über Angabe der Filenummer und die Ordnungsnummer des Records innerhalb des Files. Die Daten werden über einen Random-File-Puffer in Form von Variablen aus dem Arbeitsspeicher auf Diskette (bzw. umgekehrt) übertragen.

Es existiert kein EOF, so daß unter Umständen über das "eigentliche" Ende des Files gelesen wird! Die Funktion **LOC** kann eingesetzt werden, um zu ermitteln, welche Record-Nummer im Programm zuletzt gelesen oder beschrieben wurde. Die Funktion **LOF** gibt Aufschluß über die bisher höchste Record-Nr.

Zwischen den Records werden keine automatischhen Trennzeichen gesetzt. Die Informationen stehen somit als Kette von Bytes ("Byte-Stream") auf Diskette. Je nachdem, wie der Puffer definiert wird, wird ab einer bestimmten Stelle (abhängig von Record-Nummer und Record-Länge) eine bestimmte Anzahl von Bytes in den Puffer gelesen oder aus dem Puffer geschrieben.

Innerhalb des Programms wird ein Record (genauer: der Puffer) durch eine Feldvereinbarung beschrieben (**FIELD**-Anweisung). Feldvereinbarungen können innerhalb des Programms verändert werden. Durch eine neue Feldvereinbarung für einen bestimmten Random-File-Puffer wird die zuletzt gültige aufgehoben. Unterschiedliche Feldvereinbarungen für das gleiche File sind also möglich. Bei unterschiedlicher Recordstruktur im gleichen File kann mit einer Füllvariable ("Filler") die Recordlänge angepaßt werden. Wurde auf einen Record zugegriffen, so stehen seine Daten in den Feldvariablen der für das File zuletzt gültigen **FIELD**-Anweisung zur Verfügung und können vom Programm weiterverarbeitet werden.

Beim Schreiben in ein Random-File müssen zunächst die Feldvariablen im Random-File-Puffer die entsprechenden Werte enthalten. Anschließend kann der Inhalt des Random-File-Puffers unter einer Ordnungsnummer als Record in das File geschrieben werden.

Beim Lesen oder Schreiben kann die Angabe der Ordnungsnummer unterbleiben. In diesem Fall wird der Record im File gelesen bzw. geschrieben, dessen Ordnungsnummer auf die zuletzt - durch Lesen oder Schreiben - angesprochene Ordnungsnummer folgt.

Haben die Records eines Random-Files einen Aufbau wie Datenelemente eines sequentiellen Files (Trennzeichen!) ist es möglich, Random-Files als sequentielle Files zu bearbeiten.

4. Index-sequentielle Files

In verschiedenen Anwendungen ist es nötig, häufig auf Records zuzugreifen, die als Zugriffskriterium keine Recordnummer verwenden. Dies kann mit dem Programmpaket ISAM verwirklicht werden. Informationen darüber entnehmen Sie bitte dem Handbuch "ISAM, indexsequentielle Dateiverwaltung".

Beispiel:

Die Artikel einer Artikeldatei sind mit einer alphanumerischen Artikelnummer versehen, mit der auf die benötigten Records zugegriffen wird. Es soll außerdem möglich sein, die Artikel jederzeit - ohne vorherigen Sortierlauf - sofort sequentiell so auszulisten, wie sie aufgrund der logischen Reihenfolge der Artikelnummern vorhanden sind.

4.3.10.2 Befehle, Anweisungen und Funktionen

1. Befehle zum Arbeiten mit Datenfiles

Mit dem Befehl **NAME** kann ein auf Diskette vorhandenes Datenfile umbenannt werden, mit **KILL** kann es gelöscht werden (der Platz auf Diskette wird wieder frei). **KILL** löscht das File nur aus dem Inhaltsverzeichnis, stellt aber nicht die Default-Belegung auf Diskette wieder her!

Mit **FILES** kann das Inhaltsverzeichnis einer Diskette auf dem Bildschirm angezeigt werden.

Mit dem PCOS-Befehl **sb** wird die Anzahl von Datenfiles festgelegt, die sich in einem Programm gleichzeitig verwalten lassen. Außerdem wird angegeben, bis zu welcher maximalen Recordlänge (Puffergröße) bei Random-Files gearbeitet werden kann. Vor Programmstart werden automatisch soviele File-Puffer in dieser Größe im Arbeitsspeicher reserviert, wie durch 'max. Fileanzahl' vereinbart wurde.

Mit dem PCOS-Befehl **fn** kann ein File auf Diskette angelegt werden (Reservierung einer bestimmten Anzahl von Sektoren zu 256 Bytes auf Diskette). Mit den PCOS-Befehlen **fp** und **fd** können File-Passwords vergeben bzw. wieder gelöscht werden.

Mit **fc** können Datenfiles kopiert werden, mit **fw** mit Schreibschutz versehen. Mit **fu** kann ein Schreibschutz wieder aufgehoben werden.

2. Öffnen und Schließen von Datenfiles

Alle Datenfiles müssen mit der Anweisung **OPEN** auf Zugriffe vorbereitet werden. Dabei ist für jedes geöffnete File eine Nummer zu vergeben, unter der das File von da an bei allen Befehlen anzusprechen ist. Ist das betreffende File unter dem 'Filenamen' noch nicht angelegt, erledigt dies **OPEN** ebenfalls, sofern bei **OPEN** die Zugriffsarten "O" oder "R" angegeben sind.

Mit der Anweisung **CLOSE** können bestimmte offene Files oder alle aktuell offenen Files geschlossen werden. Auch die Anweisung **END** sowie der Befehl **NEW** schließen alle offenen Files.

Mit **OPEN** dürfen Files nur geöffnet werden, wenn:

a) die 'Filenr.' noch nicht aufgrund einer vorherigen **OPEN**-Anweisung vergeben ist, es sei denn, die vergebene Filenummer würde vorher mit **CLOSE** wieder freigegeben;
b) die 'Filenr.' im Rahmen der mit **sb** gesetzten maximalen Anzahl liegt;
c) der 'Filename' vorher nicht in einer **OPEN**-Anweisung verwendet wurde, die noch nicht durch **CLOSE** aufgehoben ist;
d) noch ausreichend Platz (falls das File noch nicht angelegt ist) auf der Diskette ist, um das File in soviel Sektoren anzulegen, wie durch **ss** festgelegt.

3. Arbeit mit sequentiellen Files

Zuerst ist das File in der entsprechenden Zugriffsart ("I", "O" oder "A") zu öffnen (**OPEN**-Anweisung). Die Angabe von 'Record-Länge' unterbleibt. Ist das File leer, haben "A" und"O" die gleiche Wirkung.

Für "I" können die folgenden Leseanweisungen eingesetzt werden: **INPUT#** und **LINE INPUT#**; für "A" bzw. "O" die Schreibanweisungen **PRINT#** bzw. **PRINT#...USING** und **WRITE#**.

Es ist zu beachten, daß die einzelnen Datenelemente im Record mit entsprechenden Trennzeichen voneinander getrennt geschrieben werden müssen, um sie später beim Lesen wieder voneinander trennen zu können. Dazu bietet sich besonders das Komma (**CHR$(44)**) an, das bei **INPUT#** als Trennzeichen erkannt wird (vgl. dazu die Bildschirmanweisungen **INPUT** und **LINE INPUT**!).

Die numerischen Daten werden im ASCII-Format auf das File geschrieben. Bei unformatierten Records wird das Standardformat zugrundegelegt (vgl. Kapitel 4.3.9), daraus resultiert eine uneinheitliche Länge, auch bei gleichem Datentyp. Strings werden in aktueller Länge geschrieben, also ebenfalls mit uneinheitlicher Länge (vgl. die Bildschirm-Anweisungen **PRINT** und **WRITE**). Unter Verwendung von **PRINT#...USING** können alle Records auf gleiche Länge "maskiert" werden. Mit der Funktion **INPUT$** kann der Inhalt des Files Zeichen für Zeichen eingelesen werden.

Beispiele:

```
1 CLEAR   ' Schreiben sequentiell ab Beginn
10 OPEN "O",1,"10:DATEN.seq":REM File auf HDD
20 PRINT "Artikelnr. (CR oder 0 = Ende)"
21 INPUT "",ANR%:IF ANR%=0 THEN 990
25 IF ANR%=0 THEN 990
30 PRINT "Bezeichnung"
31 LINE INPUT BEZ$:IF BEZ$="" THEN 30
40 PRINT#1,ANR%
50 PRINT#1,BEZ$
60 PRINT ANR%;" ist erfaßt":PRINT:GOTO 20
990 CLOSE 1:CLS:PRINT "ENDE":END
```

```
1 CLEAR   ' Lesen und Ausgeben Werte aus vorherigem Programm
10 OPEN "I",1,"10:DATEN.seq"
20 Z%=0:WHILE NOT EOF(1):Z%=Z%+1
30 INPUT#1,ANR%
40 LINE INPUT#1,BEZ$
50 M$="lfd. Nr. ###### Art.-Nr. ######## Bezeichnung &"
55 PRINT USING M$;Z%,ANR%,BEZ$
60 WEND
990 CLOSE 1:PRINT STRING$(64,"*")"FILE-ENDE"
```

```
0 REM Erfassen, Anhängen oder Ändern (Update) von Records
1 'Mindestens 2 Files nötig (+ genug Platz auf Diskette 0:)
2 CLEAR:OPTION BASE 1:DIM FELD$(5)
10 PRINT "n(eu) ab Beginn, a(nhängen), v(erändern) CR=ENDE ";
15 LINE INPUT E$:IF E$="" THEN 990
16 E%=ASC(LEFT$(E$,1)) AND 223 '1. Zeichen wird Großbuchstabe
20 IF INSTR("NAV",CHR$(E%))=0 THEN 10
30 IF E%=ASC("A") THEN ZA$="A"
40 IF E%=ASC("N") THEN ZA$="O"
50 IF E%=ASC("V") THEN ZA$="I"
60 OPEN ZA$,1,"0:RECORDS.seq"
70 IF E%=ASC("V") THEN OPEN "O",2,"0:COPY.seq"
80 FOR I%=1 TO 5
90 PRINT USING "#. zu erfassendes alphanumerisches Feld ";I%;
95 LINE INPUT FELD$(I%) ' ergibt variable Feldlängen!!
99 NEXT
100 ON INSTR("NAV",CHR$(E%)) GOSUB 1010,1010,2010
110 PRINT " *****   OK  *****":PRINT SPACE$(255):CLOSE:GOTO 10
990 CLOSE 1,2:CLS:PRINT "ENDE":END
1000 REM neuen Record schreiben
1010 FOR I%=1 TO 5:PRINT#1,FELD$(I%):NEXT:RETURN
2000 REM vorhandenen Satz überschreiben
2010 CLS:LINE INPUT "Wievielter Satz ";RN$
2020 IF VAL(RN$)(=0 OR VAL(RN$))32767 THEN 2010
2025 ENDE%=0:SATZNR%=1
2030 WHILE NOT EOF(1) AND ENDE%=0
2031 IF SATZNR%=VAL(RN$) THEN ENDE%=-1:GOTO 2099
2033     FOR J%=1 TO 5
2034         LINE INPUT#1,PSEUDO$
2036         PRINT#2,PSEUDO$     ' übertragen in anderes File
2038     NEXT J%
2040 SATZNR%=SATZNR%+1
2099 WEND
2100 IF ENDE%=0 THEN PRINT "nur";SATZNR%-1"Sätze da!":RETURN
2110 FOR J%=1 TO 5:PRINT#2,FELD$(J%):NEXT
2120 FOR J%=1 TO 5:LINE INPUT#1,PSEUDO$:NEXT
2130 WHILE NOT EOF(1)
2140     FOR J%=1 TO 5
2142         LINE INPUT#1,PSEUDO$
2144         PRINT#2,PSEUDO$
2146     NEXT
2148 WEND
2200 CLOSE:KILL "0:RECORDS.seq"
2220 NAME "0:COPY.seq" AS "RECORDS.seq":RETURN
```

```
0 'Lesen eines beliebigen sequentiellen Files
10 CLS:ON ERROR GOTO 1010
20 LINE INPUT "Filename (mit Diskettenspezifikation) ";F$
21 P%=INSTR(F$,":"):IF P%=0 THEN 20
25 IF INSTR("010",LEFT$(F$,P%-1))=0 THEN 20
30 OPEN "I",1,F$
35 WHILE NOT EOF(1)
36   K$=INPUT$(1,#1):PRINT K$;
50 WEND:CLOSE 1:END
1000 'Fehlerbehandlungsroutine
1010 IF ERR=53 AND ERL=30 THEN PRINT "nicht gefunden":RESUME 20
1020 IF ERR=57 THEN PRINT "Fehler beim öffnen: Disk(etten)-Stati
on leer oder":PRINT "nicht vorhanden oder Lesefehler.":RESUME 20
1030 IF ERR=64 THEN PRINT "falsche  Filespezifikation":RESUME 20
1040 ON ERROR GOTO 0:STOP
```

4. Arbeit mit Random-Files

Random-Files sind unter einer Filenummer mit Angabe des Parameters "R" für 'Zugriffsart' und unter Angabe der 'Record-Länge' des Random-Files zu öffnen. Diese 'Record-Länge' definiert die aktuelle gewünschte (logische) Länge des Puffers. Die durch **sb** gesetzte physische Länge kann durch **OPEN** auf eine andere logische Länge reduziert werden. Ist keine 'Recordlänge' angegeben, wird die Länge unterstellt, die in **sb** für den Parameter 'File-Puffer-Größe' gerade gültig ist.

Anschließend wird in einer **FIELD**-Anweisung für die betreffende 'Filenr.' die logische Struktur des Records (des Puffers) festgelegt, mit der von da an gearbeitet werden soll. Dabei werden die einzelnen Felder durch Feldvariablen benannt und deren Länge im Puffer in Bytes festgelegt. Die Felder sind ausschließlich in Form von Stringvariablen zu definieren (feste Feldlängen).

- Für Strings gilt:
  Für jedes Zeichen ist ein Byte zu reservieren;

- Für numerische Werte gilt:
  Für Integer-Werte ist ein String mit 2 Bytes zu reservieren;
  für jeden einfach genauen Wert ein String mit 4 Bytes;
  für jeden doppelt genauen Wert ein String mit 8 Bytes.

a) Lesen eines Records in den Random-File-Puffer

In der Anweisung **GET** ist die Filenr. und die Nummer des einzulesenden Records zu spezifizieren. Nach Durchführung der **GET**-Anweisung stehen in den Feldvariablen der betreffenden **FIELD**-Anweisung die Feldinhalte aus dem File im Random-File-Puffer zur Verfügung. Die Variableninhalte können ausgegeben werden, z.B. durch **PRINT**. Sollen diese Variableninhalte später wieder auf das File gebracht werden (z.B. nach Veränderung), dürfen dazu die Feldvariablen nur über die Anweisungen **LSET** bzw. **RSET** verändert werden. Sie sollten auch nicht als Argument in Funktionen oder innerhalb von Vergleichsausdrücken verwendet werden!! Sie müssen dazu vorher auf Programmvariablen zugewiesen werden. Numerische Daten, die über **MKI$**, **MKS$** oder **MKD$** in Strings verwandelt wwurden, bevor sie in den Puffer übertragen wurde, müssen zur Ausgabe bzw. bei der Zuweisung auf Programmvariablen mit den Funktionen **CVI**, **CVS** oder **CVD** - je nach Typ - rückverwandelt werden.

b) Schreiben eines Records auf das File

Zuerst ist auf alle Feldvariablen eine Wertzuweisung durchzuführen. Diese darf nur über die Anweisungen **LSET** oder **RSET** erfolgen! Diese Anweisungen bringen die zugewiesenen Inhalte automatisch an die durch **FIELD** definierte Position des Feldes in den Random-File-Puffer.

Numerische Daten sind - je nach Typ - mit Hilfe der Funktionen **MKI$, MKS$** bzw. **MKD$** in Strings mit festen Längen zu verwandeln.

Nach **LSET** bzw. **RSET** ist nur der Random-File-Puffer gefüllt. Mit Hilfe der **PUT**-Anweisung kann nun der Inhalt des Random-File-Puffers auf das File gebracht werden, wofür die betreffende 'Filenr.' und die Nummer des gewünschten Records anzugeben ist.

Beispiele:

```
200 CLS:OPEN "R",#1,"0:pers.rnd",38:GOTO 15020
2006  ' Die Personal-Nummer entspricht der Record-Nr. und
      bräuchte eigentlich nicht aufs File geschrieben werden.
2007  ' Es handelt sich um ein Neuerfassungsprogramm, nur noch
nicht belegte Personal-Nummern sollen aufs File gebracht werden.
2010  ' Folgender Record-Aufbau liegt zugrunde:
2020  ' von  bis  I Laenge I Progr.var. I Var.Typ I Feldvar.
2030  '  incl.,   Iin BytesI           I         I
2040  '  1    4    I   4    I PERS.NR!   I single  I PNR$
2050  '  5    24   I   20   I GES.NAME$  I string  I N$
2060  '  25   32   I   8    I LOHN#      I double  I LO$
2070  '  33   34   I   2    I ST.KLASSE% I integer I KL$
2080  '  35   38   I   4    I  (offen)   I (offen) I FILLER$
2096 'Feldvereinbarungen
2097 REM a) mit Einzelvariablen
2100 FIELD #1,4 AS PNR$,20 AS N$,8 AS LO$,2 AS KL$,4 AS FILLER$
2110 RETURN
2120 REM b) mit Feldvariable für alle Felder
2130 FIELD #1,38 AS GESFELD.RANDFILE$ : RETURN
```

Fortsetzung s. nächstes Blatt

```
7000 ' gesamter Record in einer Feldvariablen
7010 GOSUB 2130
7020 ' wie 15030-15070
7030 'wenn nicht vorhanden, ist GESFELD.RANDFILE$=STRING$(38,0)
7040 IF GESFELD.RANDFILE$<>STRING$(38,0) THEN PRINT "vorhanden"
7045 REM  Da Programmvariablen vom Typ String immer nur von
          aktueller Länge sind  wird es hier nötig, die String-
          Variable GES.NAME$ zur Länge von 20 aufzufüllen
          (hier: mit Blanks, rechtsbündig)
7050 LSET GESFELD.RANDFILE$=MKS$(PERS.NR!)+GES.NAME$+SPACE$(20-L
EN(GES.NAME$))+MKD$(LOHN#)+MKI$(ST.KLASSE%)
7060  'ACHTUNG: Der Filler würde hier automatisch mit Blanks
       belegt'
7070 'wie 15140
7080 GOTO 7020
15000 REM Hauptverarbeitung
15020 GOSUB 2100 'Feldvereinbarungen einzeln (gesamt s. 7000ff.)
15030 PRINT "Pers.Nr.(1-500), Name, Brutto, St.-Kl.(1-6) (0,0,0,
0=ENDE)":INPUT "",PERS.NR!,GES.NAME$,LOHN#,ST.KLASSE%
15031 IF PERS.NR!=0 THEN 16910
15035 IF PERS.NR!<0 OR FIX(PERS.NR!)<>PERS.NR! OR PERS.NR!>500
THEN 15030
15040 IF LEN(GES.NAME$)>20 THEN 15030
15050 IF ST.KLASSE% <=0 OR ST.KLASSE%>6 THEN 15030
15060 'Feststellen, ob Persnr. bereits vorhanden
15070 GET #1,PERS.NR!
15080 'wenn Record noch nie beschrieben, ist PNR$=STRING$(4,0)
15090 IF PNR$<>CHR$(0)+CHR$(0)+CHR$(0)+CHR$(0) THEN PRINT "Pers.
-Nr.";PERS.NR!;" ist bereits vorhanden!";SPACE$(255):GOTO 15030
15095 REM Vorbereitung zum Schreiben (PUT) --> Zuweisung der
      Programmvariablen auf die Feldvariablen
15100 RSET PNR$=MKS$(PERS.NR!) 'RSET bei num. Werten
15110 LSET N$=GES.NAME$
15120 RSET LO$=MKD$(LOHN#) 'RSET bei num. Werten
15130 RSET KL$=MKI$(ST.KLASSE%) 'RSET bei num. Werten
15135 ' Filler wird nicht verändert. Da vorher kein GET erfolgt
      ist (!), ist der Wert undefiniert (FIELD stellt nur die
      richtige Länge her, aber KEINE sinnvolle Vorbelegung!)
15140 PUT #1,PERS.NR!
15145 PRINT PERS.NR!;"ist jetzt erfasst":GOTO 15030
16910 CLOSE #1:END :REM Endverarbeitung
```

```
0 CLEAR
1 ' Lesen und Ändern von mit dem Vorprogramm erfaßten Records
10 OPEN "R",1,"0:pers.rnd",38
20 FIELD 1,4 AS PNR$,20 AS N$,8 AS LO$,2 AS KL$,4 AS FILLER$
30 PRINT "Welche Record-Nr. (0=ENDE)":LINE INPUT H$
40 IF H$="0" THEN CLOSE:CLS:PRINT "ENDE":END
50 IF VAL(H$)<=0 OR VAL(H$)>500 THEN 30 ELSE PNR%=VAL(H$)
100 GET 1,PNR%
110 IF PNR$=STRING$(4,0) THEN PRINT "nicht vorhanden!":GOTO 30
120 MASKE$="Nr.: ### Name: &                    & Brutto: ######.#
# STK: #"
130 PERS.NR!=CVI(PNR$)
132 GNAME$=N$:WHILE RIGHT$(GNAME$,1)=" ":GNAME$=LEFT$(GNAME$,LEN
(GNAME$)-1):WEND     ' Eliminieren unnötiger Strings
133 Z!=FRE("A") ' garbage collection
140 LOHN#=CVD(LO$)
150 STK%=CVI(KL$)
170 LPRINT USING MASKE$;PNR%,GNAME$,LOHN#,STK%
175 PRINT USING MASKE$;PNR%,GNAME$,LOHN#,STK%:PRINT
180 PRINT "Wähle Änderung (1=Name, 2=Brutto, 3=Steuerkl, 0=ok)"
190 LINE INPUT Q$:IF LEN(Q$)<>1 THEN 180 ELSE IF INSTR("1230",Q$
)=0 THEN 180
195 IF Q$="0" THEN GOSUB 5010:GOTO 30
200 ON VAL(Q$) GOSUB 1010,2010,3010
210 GOTO 180
1000 REM Name
1010 PRINT GNAME$:LINE INPUT H$
1020 IF LEN(H$)>20 THEN 1010 ELSE GNAME$=H$
1030 LSET N$=GNAME$:RETURN
2000 REM Brutto
2010 PRINT LOHN#:LINE INPUT H$
2015 IF VAL(H$)=0 AND H$<>"0" THEN 2010
2020 RSET LO$=MKD$(LOHN#):RETURN
3000 REM Steuerklasse
3010 PRINT STK%:LINE INPUT H$
3020 IF VAL(H$)<=0 OR VAL(H$)>6 THEN 3010 ELSE STK%=VAL(H$)
3030 RSET KL$=MKI$(STK%):RETURN
5000 REM Update
5010 PUT 1,PNR%:RETURN
```

Bemerkungen:

- Sowohl beim Lesen als auch beim Schreiben kann die Angabe der Record-Nr. unterbleiben. Es wird dann der Record bearbeitet, der dem zuletzt gelesenen oder geschriebenen folgt.
- Wurden Records auf dem File noch nicht beschrieben, können sie trotzdem gelesen werden. Jedes Byte dieser Records ist mit dem Zeichen **CHR$(Ø)** belegt. Dies gilt nicht uneingeschränkt. Nur der PCOS-Befehl **vf** stellt sicher, daß alle Positionen für Files mit **CHR$(Ø)** vorbelegt sind. Wurden Disketten mit **vn** gelöscht oder der BASIC-Befehl **KILL** bzw. der PCOS-Befehl **fk** eingesetzt, werden die betreffenden File-Positionen nicht mit **CHR$(Ø)** vorbelegt, sondern nur das Inhaltsverzeichnis wird betroffen. Auch **OPEN** bzw. **fn** stellt diese Vorbelegung nicht her.
- Unterscheidet sich die Summe der Längen aller Feldvariablen in der **FIELD**-Anweisung von der Recordlänge, sind zwei Fälle zu unterscheiden:
  a) die Feldvereinbarung ist zu niedrig ausgelegt:
     Nur die nicht belegten Zeichen in den definierten Feldvariablen werden beim Schreiben des Records wie üblich mit Blanks belegt; der nicht von der Feldvereinbarung betroffene Rest des Records (rechter Teil) auf dem File bleibt unverändert.
  b) die Feldvereinbarung ist zu groß:
     Es wird der Fehler "FIELD overflow" (Fehler-Code 5Ø) gemeldet.
- Wird ein beliebiges File mit der Zugriffsart "I" geöffnet, ist aber nicht vorhanden, wird es nicht automatisch angelegt, sondern der Fehler "File not found" (Fehler-Code 53) gemeldet. Dieser Fehler kann in einer Fehlerbehandlungsroutine verarbeitet werden.

4.3.11 Spezielle und reservierte Variablen

Die speziellen Variablen **DATE$** und **TIME$** werden vom System auf PCOS-Ebene verwaltet und bieten die Möglichkeit, den M20 eine Echtzeit führen zu lassen.

Zuweisungen auf diese Variablen können erfolgen durch

a) den PCOS-Befehl ss;

b) eine Wert-Zuweisung in der BASIC-Ebene (Dabei muß **LET** weggelassen werden). Eine Zuweisung über **INPUT** o.ä. ist nur über Hilfsvariablen möglich; bei **INPUT DATE$** z.B. wird zwar keine Fehlermeldung beim Input gegeben, der Inhalt von **DATE$** zum Zeitpunkt der Eingabe wird jedoch durch diese nicht verändert.

Ist der zugewiesene String nicht formgerecht, wird der aktuelle Wert von **DATE$** bzw. **TIME$** nicht verändert (keine Fehlermeldung).

Die Werte von **DATE$** bzw. **TIME$** werden über die Programme hinweg intern ständig vom Betriebssystem angepaßt. Sie gehen beim Ausschalten oder physischen Reset verloren, jedoch nicht beim logischen Reset mit RESET.

Mit den reservierten Variablen **ERR** und **ERL** kann der Fehler-Code des zuletzt aufgetretenen BASIC-Fehlers (vgl. folgendes Kapitel) bzw. die Zeilennr. abgefragt werden, in der der letzte BASIC-Fehler auftrat. Wertänderung von **ERL** bzw. von **ERR** sind mit Hilfe der Anweisung **ERROR** möglich.

4.3.12 Fehlerbehandlungsroutine

Durch Fehler bei der Erstellung des Programms oder durch unzulässige oder unvorhergesehene Datenkonstellationen können Situationen entstehen, die das System meldet.

Befindet sich das System in der PCOS-Ebene oder wurden PCOS-Routinen vom BASIC-Interpreter aufgerufen, erfolgt die Meldung vom Betriebssystem PCOS. Diese Meldungen werden im weiteren kurz PCOS-Fehler genannt.

Befindet sich das System in der BASIC-Ebene, erfolgen die Meldungen vom BASIC-Interpreter (Ausnahme: Aufruf von PCOS-Routinen in einem BASIC-Programm durch **CALL**- oder **EXEC**-Anweisung). Diese Meldungen werden im weiteren kurz BASIC-Fehler genannt. Die einzelnen BASIC-Fehlermeldungen werden in Kapitel 6 erläutert.

Mit Hilfe der Anweisungen **ON ERROR GOTO, RESUME** sowie der reservierten Variablen **ERR** und **ERL** kann in Programmen in einer Fehlerbehandlungsroutine definiert werden, was geschehen soll, wenn der Interpreter einen BASIC-Fehler erkennt. Fehler, die von PCOS gemeldet werden, werden dort nicht erkannt. Allerdings erhält **ERR** bei den meisten PCOS-Fehlern den Wert 40.

Ohne Fehlerbehandlungsroutine werden alle Fehler, die das System beim Abarbeiten einer Anweisung oder eines Befehls entdeckt, in der nächsten Bildschirm- bzw. Window-Zeile gemeldet.

BASIC-Fehler werden dabei mit ihrer englischen Bedeutung ausgegeben, PCOS-Fehler hingegen mit ihrem Fehler-Code.

Treten BASIC-Fehler bei der Abarbeitung von Anweisungen im Rahmen eines Programms auf, wird immer die Zeilennummer mit angegeben, in der der Fehler erkannt wurde. Treten BASIC-Fehler während der Abarbeitung von Anweisungen im Direkt-Mode auf, wird keine Zeilennummer angegeben. **ERL** hat aber den Wert 65535.

In Fehlerbehandlungsroutinen können BASIC-Fehler abgefangen werden. Ist im Programm eine Fehlerbehandlungsroutine aktiviert (**ON ERROR GOTO**), verzweigt das Programm selbsttätig zu der bei **ON ERROR GOTO** festgelegten Zeilennummer, sobald der Interpreter einen Fehler erkennt. Der Fehler wird nicht angezeigt und kann in der Fehlerbehandlungsroutine behandelt werden (z.B. durch korrigierende Wertzuweisung oder Ausgabe einer Meldung, was vom Bediener zu tun ist). Die Behandlung kann von Fehler zu Fehler verschieden definiert werden, und zwar in Abhängigkeit von den Inhalten der Variablen **ERR** und/oder **ERL**.

Der Rücksprung ins Hauptprogramm (z.B. nach Behebung der Fehlerursache) wird durch **RESUME** erreicht. Es kann dabei vorgeschrieben werden, daß

a) die Anweisung, bei der der Fehler erkannt wurde, nochmals abgearbeitet werden soll (**RESUME** oder **RESUME Ø**)

b) diejenige Anweisung als nächste abgearbeitet werden soll, die der Anweisung unmittelbar folgt, bei der der Fehler verursacht wurde (**RESUME NEXT**);

c) ab einer bestimmten Zeilennumer weitergearbeitet werden soll. Diese Zeilennumer muß außerhalb der Fehlerbehandlungsroutine liegen.

Im Fall a) ist zu beachten, daß gewisse Fehler nicht behoben werden können, ohne das Programm selbst zu verändern (z.B. Syntaxfehler); erfolgt dies nicht, verzweigt das System sofort wieder in die Fehlerbehandlungsroutine, so daß das Programm in eine Art Endlosschleife gerät. Im Falle des Fehler 7 ("Overflow") gilt bei **INPUT** bzw. **LINE INPUT** auf numerische Werte: Der durch eine zu große Eingabe verursachte "Overflow" kann zwar in der Fehlerbehandlungsroutine bearbeitet werden und die den Fehler verursachende Anweisung wiederholt werden; in diesem Fall wird die Meldung "Overflow" jedoch in der nächsten Zeile trotzdem ausgegeben.

Durch **RESUME** erhält die reservierte Variable **ERR** automatisch den Wert Ø zugewiesen; der Inhalt von **ERL** hingegen bleibt unverändert.

Beispiel:

```
10 CLEAR:ON ERROR GOTO 10010:CLIN%=1
20 CLS:CURSOR(1,4):INPUT "Zähler ";ZAEHLER%
25 CURSOR(1,CLIN%+1):PRINT SPACE$(LEN("Division by zero"));
26 'Löschen Zeile mit Fehlermeldung
30 CURSOR(1,5):CLIN%=POS(1):INPUT "Nenner ";NENNER%
40 INTDIVERGEBNIS%=ZAEHLER% \ NENNER% 'Div. durch 0 möglich!
50 PRINT "Die Integerdivision von ";ZAEHLER%;"und ";NENNER%;
60 PRINT "liefert ";INTDIVERGEBNIS%
99 K$=INPUT$(1):GOTO 20
10000 'Fehlerbehandlungsroutine
10010 IF ERL=40 THEN RESUME 25 'erst ERR=11, dann ERR=6;
      beides wird nach RESUME angezeigt, deshalb wird die
      Zeile, in der Meldung erfolgen würde, nach RESUME
      erst gelöscht!
10099 ON ERROR GOTO 0:STOP
```

```
0 ' Test auf richtige Station, richtige Disk, richtiges File
    und darauf, ob Schreiben in das File möglich ist.
20 CLEAR:ON ERROR GOTO 2010
25 OPEN "I",1,"0:KONTR.seq" ' richtige Station ?
26 ' KONTR.seq muß bei Zugriffsart "I" vorhanden sein!
30 CLOSE 1:OPEN "I",1,"DATEN1_A:KONTR.seq" 'Diskettenname ok?
35 ' Disk-Name kann über CALL "pl vr" (Systemdisk muß einliegen)
     und CALL "vr 0:,DATEN1_A" (zu benennende Disk in Station 0)
     und CALL "pu vr" vergeben werden.
40 CLOSE 1:OPEN "R",1,"DATEN1_A:KONTR.seq",1 'Schreiben möglich?
41 SLF%=0 ' Schalter für Error 57 beim Schreibversuch
42 FIELD 1,1 AS F1$:GET 1,1:PUT 1,1
50 CLOSE 1:IF SLF% THEN SLF%=0:GOTO 40 'Schreiben erfolgt ?
60 ' Der Fehler 57 entsteht beim CLOSE. Vorher wird das File
     jedoch LOGISCH geschlossen, so daß beim 2. Versuch kein
     Fehler mehr entsteht, wenn nicht nochmals geöffnet wird.
99 PRINT "Disk o.k., Schreiben mit Sicherheit erfolgt.":END
2000 ' Fehlerbehandlungsroutine
2010 IF ERL=25 OR ERL=30 OR ERL=40 OR ERL=50 THEN CLS:COLOR 0,1:
PRINT "Bitte Diskette in Station 0: einlegen, die gelesen werden
 kann. ".GOTO 2016
2015 ON ERROR GOTO 0:STOP ' sonstige Fehler --> Abbruch!!!
2016 IF ERR=57 THEN GOSUB 2030:GOTO 2080
2017 IF ERR=69 THEN GOSUB 2040:GOTO 2080
2018 IF ERR=53 THEN GOSUB 2050:GOTO 2080
2030 PRINT "Schreib/Lese-Fehler ":SLF%=-1:RETURN
2040 PRINT "Falscher Diskettenname ":RETURN
2050 PRINT "Falsche Diskette, Datei nicht vorhanden ":RETURN
2080 PRINT "Weiter mit beliebiger Taste. ";:COLOR 1,0:K$=INPUT$(
1):CLS:RESUME 0
```

Es ist immer die zuletzt aktivierte Fehlerbehandlungsroutine aktiv. Diese Routine bleibt auch beim Aufsuchen des Direkt-Modes aktiv, kann aber durch **ON ERROR GOTO Ø** "abgeschaltet" werden.

PCOS-Fehler können in Fehlerbehandlungsroutinen nicht abgefangen werden, die Fehlermeldung erscheint immer. Allerdings hat **ERR** nach Meldung eines PCOS-Fehlers innerhalb eines BASIC-Programms (bei **CALL**- oder **EXEC**-Anweisungen) meistens den Wert 4Ø.

Mit Hilfe der Anweisung **ERROR** kann auf **ERR** ein Wert zugewiesen werden. (Davon wird indirekt auch **ERL** betroffen!) Dies läßt sich zu Testzwecken ausnutzen (Simulation von Fehler-Konditionen).

Es bietet sich aber auch eine weitere Möglichkeit: Da die dem Interpreter bekannten Fehler-Codes nur von 1 bis 78 gehen (Ø kann nicht auf **ERR** zugewiesen werden), können "künstliche" Fehler vom Anwender definiert und erzeugt werden, die sich z.B. zur Programmsteuerung aus der Fehlerbehandlungsroutine heraus einsetzen lassen.

Beispiel:

```
0 ON ERROR GOTO 22800  ' mit simulierten Fehlern
5 I%=1:LGR#=-50:RGR#=50
10 INPUT "Eingabe eines Wertes (Intervall -50 <= 50)";WERT#(I%)
20 GOSUB 800
25 I%=2:LGR#=-.999:RGR#=.999
30 INPUT "Eingabe eines Wertes (Intervall -.999 <= +.999)";
WERT#(I%)
40 GOSUB 800 ' ...
799 END
800 IF WERT#(I%)<LGR# OR WERT#(I%)>RGR# THEN ERROR 100+I%
809 RETURN
22800 IF ERR<80 THEN PRINT "Programmfehler":ON ERROR GOTO 0:STOP
22810 PRINT "Bitte Eingabe im korrekten Intervall"
22820 ON ERR-100 GOTO 22821,22822
22821 RESUME 10
22822 RESUME 30
```

4.3.13 Verkettung von Programmen

Aus verschiedenen Gründen kann es nötig oder empfehlenswert sein, Programme zu verbinden.

1. Job-Streams

   In einem automatischen Startprogramm (BASIC-Programm namens INIT.BAS, das über BASIC selbst erstellt werden kann und unter dem festen Namen INIT.BAS auf der Diskette abgespeichert werden muß, auf der sich der BASIC-Interpreter -PCOS-Befehl **ba**befindet) werden z.B. **DATE$** und **TIME$** eingegeben und gesetzt, bestimmte Systemparameter definiert (z.B. PCOS-Befehle **sf** für Drucker, **ss** für das Betriebssystem, **pl** zum Laden benötigter PCOS-Befehle) und Vorarbeiten für das ISAM-Programmpaket erledigt (**pl is** sowie Anwenden der **MI**-Funktion - vgl. ISAM-Handbuch). Aus diesem Programm wird am Ende ein Programmauswahlprogramm aufgerufen, das vom Programmpaket abhängig ist. Nach Abarbeitung jedes Einzelprogramms wird sofort wieder zum Auswahlprogramm zurückverzweigt.

2. Verbindung von Programmteilen zu einem Gesamtprogramm

   Diese Möglichkeit kann angewendet werden, weil entweder das Gesamtprogramm als Ganzes (+ Datenbereich) zu groß für den Arbeitsspeicher ist und deshalb segmentweise abgearbeitet werden muß, oder weil bestimmte Routinen (Module) in allen Programmen gleich sind und der Speicherplatz auf Diskette besser ausgelastet werden soll. Die Module werden isoliert abgespeichert und bei Bedarf in ein laufendes Programm zugeladen ("Overlay"), wobei evtl. andere alte Teile gelöscht werden.

Zum Verketten von Programmen gibt es folgende Möglichkeiten:

1. BASIC-Befehl **MERGE**

   Beim Erstellen eines Programms im Command-Mode wird ein zuvor im ASCII-Format gespeichertes Modul (**SAVE** "Modulfilename",**A**) mit **MERGE** "Modulfilename" zum aktuellen Inhalt des Arbeitsspeichers hinzugeladen. Später wird das Gesamtprogramm als BASIC-File abgespeichert (**SAVE** "Programmfilename").

   Beispiel:

```
Ok
load "10:KOPF.prg"
Ok
merge "10:T20500" :REM Modul
Ok
save "10:GESAMT.prg"
Ok
```

2. BASIC-Anweisung **CHAIN**

   Ein im Arbeitsspeicher vorhandenes Programm wird durch ein anderes vollständig ersetzt. Das neue Programm wird sofort zum Laufen gebracht (**CHAIN** "neuer Programmname"). Dies entspricht dem BASIC-Befehl **RUN** "Filename". In dieser Form gehen alle Daten des ersten Programms verloren, offene Files werden aber nicht geschlossen. Wird der BASIC-Befehl **RUN** "Filename",**R** eingesetzt, bleiben ebenfalls die Files offen.

   Beispiel:

```
0 REM INIT.BAS (Startprogramm)
10 CALL "sf %n,,pr1471,0" 'Drucker setzen
20 CALL "ss %n,,,,,,1" ' Bildschirm-Format
30 CHAIN "0:MENU.prg"
```

Sollen in das laufende Programm ein oder mehrere BASIC-Overlays eingebracht werden, ist die **CHAIN**-Anweisung in der Form **CHAIN MERGE** "Overlay-Filename" zu verwenden. Das Overlay-File muß im ASCII-Format auf Diskette gespeichert sein (**SAVE** mit Parameter **A**). Der Prozeß des Zuladens kann etwas länger dauern. Dabei ist es möglich, durch zusätzliche Verwendung des Parameters **DELETE** in der **CHAIN MERGE**-Anweisung vor dem Zuladen bestimmte Teile des Arbeitsspeicherinhalts zu löschen.

Sollen bei **CHAIN MERGE** alle bisherigen Variableninhalte an das neugewonnene Programm übergeben werden, ist zusätzlich noch der Parameter **ALL** anzugeben.

Beispiel:

```
0 REM Anfangsprogramm ("ROOT")
25 OPTION BASE 0:
30 DIM T%(12):FOR I%=1 TO 12:READ T%(I%):NEXT
40 CHAIN MERGE "0:PROG1.asc",50,ALL,DELETE 9000-9010
50 PRINT "HALLO"
9000 DATA 31,28,31,30,31,30
9010 DATA 31,31,30,31,30,31


60 ' einzubindendes Programm PROG1.asc (SAVE "0:PROG1.asc",a)
61 FOR I%=6 TO 12:PRINT USING "##    ";T%(I%),:NEXT


Ok
run "0:ROOT"
HALLO
30    31    31    30    31    30    31
Ok
```

4.3.14 Graphisches Arbeiten

In Abschnitt 4.3.18 wird beschrieben, auf welche Weise der Bildschirm in mehrere Teile (Windows) unterteilt werden kann. Die nachfolgende Beschreibung bezieht sich auf das Arbeiten innerhalb eines Windows.

In jedem Window kann über die Anweisung **SCALE** ein gültiger Wertebereich in problembezogenen Koordinaten definiert werden. Diese Zuweisung erfolgt durch Angabe der gültigen Minimal- und Maximalkoordinaten. Dabei werden der linken unteren Ecke die Minimal- und der rechten oberen Ecke die Maximalwerte zugeordnet. Aus der Größe des Windows (das heißt der Anzahl Elementarpunkte) und dem gültigen Wertebereich ergibt sich ein Maßstab, der für horizontale und vertikale Richtung unterschiedlich sein kann.

Die horizontale Richtung von links nach rechts entspricht der positiven X-Richtung, die vertikale Richtung von unten nach oben entspricht der positiven Y-Richtung.

Wird keine **SCALE**-Anweisung gegeben, entspricht der Punkt unten links der Koordinate (Ø,Ø) und jede Koordinatenangabe bezieht sich auf Elementarpunkt-Koordinaten (vgl. Kapitel 4.3.18).

Für die graphische Darstellung stehen eine Reihe von Anweisungen zur Verfügung, die sich auf das für das Window gültige **SCALE** beziehen.

Mit **LINE** können Geraden oder Rechtecke entweder in Absolut- oder Relativkoordinaten definiert werden. **CIRCLE** erlaubt das Zeichnen von Kreisen oder Ellipsen.

Durch diese Anweisungen können Darstellungen graphischer Elemente sehr einfach erfolgen. Zusätzliche Möglichkeiten ergeben sich durch das Verändern von Vorder- und Hintergrund- bzw. der für die Darstellung gültigen Farbe (vgl. Kapitel 4.3.18.3). Sowohl **LINE** als auch **CIRCLE** erlauben über einen eigenen Operanden die Ausführung von logischen Operationen auf den durch die Darstellung angesprochenen Elementarpunkten.

Mit **PSET** und **PRESET** können einzelne Punkte des Windows angesprochen werden. Um die Elementarpunkt-Koordinaten aus einer Anwenderskalierung zu erhalten, gibt es die Funktionen **SCALEX** und **SCALEY**. Diese sind besonders wichtig bei Aufruf des PCOS-Befehls **la** in BASIC zur Erzeugung von vergrößerter Schrift, da dort die Anfangsposition des Strings in Elementarpunkt- und nicht in Anwenderkoordinaten definiert werden muß.

Mit der Anweisung **CURSOR POINT** läßt sich der Graphik-Cursor - unabhängig vom Text-Cursor - positionieren.

Beispiele:

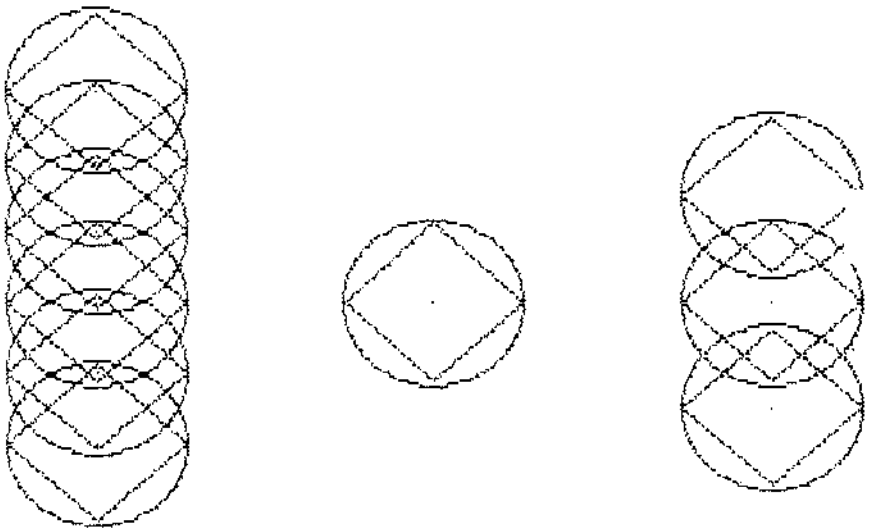
```
0 ' Balkendiagramm; Summe: 100%
10 OPTION BASE 1:DIM P!(10):MINIMUM!=0:MAXIMUM!=0
20 CALL "ss %n,,,,,0"'voller Schirm
30 INPUT "wie viele Teile";A%:IF A%>10 THEN 30
40 S!=0:FOR I%=1 TO A%
50 PRINT USING "##. %-Satz ";I%;
51 INPUT P!(I%)
55 IF S!+P!(I%)>100 THEN 50
60 S!=S!+P!(I%):IF P!(I%)>MAXIMUM! THEN MAXIMUM!=P!(I%)
70 IF S!=100 THEN I%=A%
99 NEXT
100 SCALE 0,500,0,MAXIMUM!:CLS
110 PSET(0,0)
115 FOR I%=1 TO A%
120 LINE -((I%*40),P!(I%)),,B
125 PSET((I%*40),0)
130 NEXT
```

```
1 DEF FNPARABEL!(X!)=X!^3-X!^2-2*X!
2 CALL "pl %n la"
10 XL!=-3:XR!=3
15 SCALE XL!,XR!,FNPARABEL!(XL!),FNPARABEL!(XR!)
20 CLS:FOR X!=-3 TO 3 STEP .01
30 PSET(X!,FNPARABEL!(X!))
40 NEXT
50 CALL "la"("X**3 - X**2 - 2*X",SCALEX(-2.5),0,4,0)
60 PRINT T$
```

```
0 REM Kreis
10 CALL "es %n,,,,,0"
20 PRINT "Randwerte des Bildes"
30 INPUT "links ";XMIN!
40 INPUT "rechts ";XMAX!:IF XMAX!<=XMIN! THEN 40
50 INPUT "unten ";YMIN!
60 INPUT "oben ";YMAX!:IF YMAX!<=YMIN! THEN 50
100 SCALE XMIN!,XMAX!,YMIN!,YMAX!
110 PRINT "Koordinaten des Mittelpunkts: X,Y":INPUT MX!,MY!
120 INPUT "Radius ";R!
125 CLS
130 CIRCLE(MX!,MY!),R!
199 END
```

```
1 CALL "sf %n.,pr2400,0":CALL "pl sp":CLEAR
2 ON ERROR GOTO 2010
3 S%=1:LPRINT "@@@@@@"
10 XMIN%=0:XMAX%=100:YMIN%=0:YMAX%=100
20 SCALE XMIN%,XMAX%,YMIN%,YMAX%
30 MX%=(XMIN%+XMAX%)/2:MY%=(YMIN%+YMAX%)/2
40 X%=MX%:Y%=MY%:GOSUB 1010
55 CURSOR(1,16):PRINT SPACE$(63);:CURSOR(1,16)
60 PRINT "Mittelpunkt: X,Y (0<X<100; 0<Y<100 ";:IF S% THEN PRI
NT " 0,0-->Hardcopy ";
61 INPUT ;"",X%,Y%
62 CURSOR(1,16):IF S%=2 THEN PRINT "LOCAL ! ":STOP
66 IF (X%=0)*(Y%=0) AND S% THEN LPRINT:IF S% THEN CALL "sp ,p"
70 IF (X%>=100)+(X%<=0)+(Y%>=100)+(Y%<=0) THEN 55
99 GOSUB 1010:GOTO 55
1000 REM Figur zeichnen
1010 REM FIGUR$="BJ=X%=,=Y%= BL8":DRAW FIGUR$ 'ohne Mittelpunkt
1011 PSET (X%,Y%):DRAW "BL8" 'andere Version mit Mittelpunkt
1020 DRAW FIGUR$
1022 FIGUR$="M8,-11 M8,11 M-8,11 M-8,-11":DRAW FIGUR$
1030 CIRCLE(X%,Y%),8,,8/11
1099 RETURN
2010 'Drucker nicht angeschlossen
     LOCAL-Stellung bringt PC05-Fehler 110, kann nicht abge-
     fangen werden
2020 IF ERR=79 AND ERL=3  THEN S%=0:RESUME NEXT
2030 IF ERR=79 AND ERL=66 THEN  S%=2:RESUME 62
2099 ON ERROR GOTO 0:STOP
```

Hardcopy s. nächstes Blatt

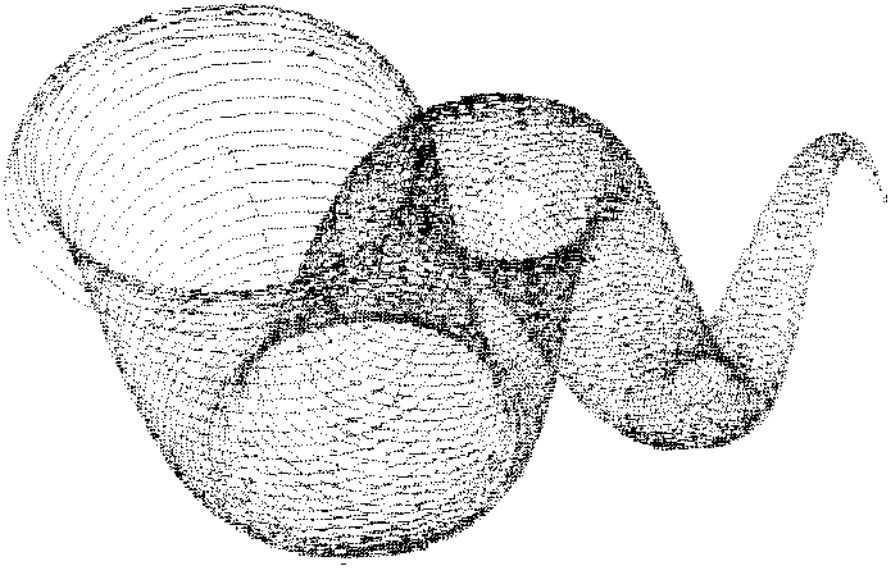


Mittelpunkt: X,Y (0<X<100; 0<Y<100) 0,0-->Hardcopy ? 0,0

```
0 REM GET% PUT%
1 CLEAR:GOSUB 2:GOTO 100
2 DIM A1%(1900),A2%(1700),A3%(2500),A4%(700)
10 RANDOMIZE VAL(RIGHT$(TIME$,2))
30 DEF FNZ(V)=1/16*(429*V^7-693*V^5+315*V^3-35*V)
58 SCALE -.74,1.5,-.8,.65:RETURN
100 R=.005
105 X=-.69
106 DELTA=.1
107 PSET (X,FNZ(X))
110 WHILE X<.91
160 CIRCLE(X,FNZ(X)),R,,.58
165 LINE (X,FNZ(X))
166 IF X>1.3 THEN R=R+.001:GOTO 180
170 R=R+.003
180 X=X+.01
190 DELTA=DELTA+RND*.1
195 DELTA1=DELTA+RND*.04
199 WEND
200 GOSUB 901:GOSUB 921:GOSUB 931:GOSUB 941
490 CURSOR(1,16):PRINT "Bitte beliebige Taste drücken ";
500 K$=INPUT$(1):CLS
510 GOSUB 1010:GOSUB 1020:GOSUB 1030:GOSUB 1040
600 GOTO 600
900 ' GET%
901 RESTORE 2010:READ X11,Y11,X12,Y12
910 GET (X11,Y11)-(X12,Y12),A1%(0):RETURN
921 ' GET%
922 RESTORE 2020:READ X21,Y21,X22,Y22
923 GET (X21,Y21)-(X22,Y22),A2%(0)
929 RETURN
931 ' GET%
932 RESTORE 2030:READ X31,Y31,X32,Y32
933 GET (X31,Y31)-(X32,Y32),A3%(0)
939 RETURN
941 ' GET%
942 RESTORE 2040:READ X41,Y41,X42,Y42
943 GET (X41,Y41)-(X42,Y42),A4%(0)
949 RETURN
```

Fortsetzung s. nächstes Blatt

```
1000 ' PUT%
1010 PUT (X11,Y11)-(X12,Y12),A1%(0):RETURN
1019 ' PUT%
1020 PUT (X21,Y21)-(X22,Y22),A2%(0):RETURN
1030 ' PUT%
1031 PUT (X31,Y31)-(X32,Y32),A3%(0):RETURN
1040 ' PUT%
1041 PUT (X41,Y41)-(X42,Y42),A4%(0):RETURN
2010 DATA -.70,-.48,+.12,.4
2020 DATA .12,-.69,.57,.65
2030 DATA .57,-.77,1.20,.62
2040 DATA 1.2,-.68,1.5,.1
```



Die Anweisung **PAINT** erlaubt das Ausfüllen eines beliebigen umschlossenen Gebietes in einer bestimmten Farbe. Dabei wird - ausgehend von einem Punkt innerhalb dieses Gebietes - die gewählte Farbe für alle Punkte gesetzt, bis ein Elementarpunkt mit einem anderen Farbcode erkannt wird.
Es ist möglich, Teilbereiche eines Windows in einem Array zu speichern oder in einem Array gespeicherte Bilder auf dem Bildschirm darzustellen. Die Anweisung **GET %** erlaubt die Auswahl dieses Bereiches und die Angabe, in welchem Array die Speicherung erfolgen soll. **PUT %** erlaubt die Darstellung eines gespeicherten Bildes an einer Stelle des Bildschirms.
Über **DRAW** können sehr komplexe Darstellungen von in Form eines Stringausdrucks gespeicherten Figuren erreicht werden. Durch die Angabe in Relativkoordinaten können solche Figuren leicht an beliebigen Stellen dargestellt und ggf. auch eine Vergrößerung ("Zoom") oder eine Drehung programmiert werden.

4.3.15 Aufruf von Assembler-Routinen bzw. PCOS-Befehlen in BASIC

Neben der Möglichkeit, BASIC-Overlays zu bilden (**CHAIN MERGE**-Anweisung), kann auch eine Assembler-Routine aus dem BASIC-Programm aufgerufen werden, ohne daß dabei das BASIC-Programm oder dessen Daten verändert werden. Diese Assembler-Routine (wie auch alle PCOS-Befehle) kann auf Diskette gesucht und dann kurzfristig - also nur für die Abarbeitung - zugeladen werden oder aber ständig im Arbeitsspeicher vorhanden (resident) sein. Letzteres beschleunigt den Aufruf, kostet aber Speicherplatz im Nicht-BASIC-Bereich des Arbeitsspeichers. Mit dem PCOS-Befehl **pl** können Assemblerroutinen resident gemacht werden, die ständig - über alle Programme hinaus - im Arbeitsspeicher vorhanden bleiben. Mit dem PCOS-Befehl **pu** können sie wieder aus dem Arbeitsspeicher entfernt werden.

Zum Aufrufen existieren die BASIC-Anweisungen **EXEC** und **CALL. CALL** beinhaltet alle Möglichkeiten von **EXEC**, ist aber leistungsfähiger als diese Anweisung.

Hinter **EXEC** ist der Name des abzuarbeitenden Assemblerfiles in Form eines Stringausdrucks anzugeben. Gegebenenfalls ist nach dem Namen der Routine die Liste der benötigten Parameter anzugeben. Alle Angaben müssen in Form eines Strings erfolgen.

Beispiel:

```
0 'Setzen Drucker
1 CLEAR:GOSUB 6000:END
6000 W=WINDOW(0,0,,6):CURSOR(1,1):COLOR 0,1
6001 PRINT"Welcher Drucker? (pr2400, pr1450, pr1471, pr1481, pr2
300, keiner) ";
6002 COLOR 1,0:KONFIGURIERTER_PRINTER$=""
6003 CURSOR(70,1)
6004 IF LEN(KONFIGURIERTER_PRINTER$)=6 THEN 6005 ELSE KONFIGURIE
RTER_PRINTER$=KONFIGURIERTER_PRINTER$+CHR$(-33 IMP ASC(INPUT$(1)
)):PRINT RIGHT$(KONFIGURIERTER_PRINTER$,1);:GOTO 6004
6005 IF INSTR("pr2400pr1450pr1471pr1481pr2300keiner",KONFIGURIER
TER_PRINTER$)=0 THEN 6000
6006 CURSOR(INSTR("Welcher Drucker? (pr2400, pr1450, pr1471, pri
481, pr2300, keiner",KONFIGURIERTER_PRINTER$),1):PRINT KONFIGURI
ERTER_PRINTER$;
6007 IF INKEY$()"" THEN 6007 ELSE COLOR 0,1:CURSOR(70,1):PRINT "
Ok? (j/n) ";
6008 IF CHR$(ASC(INPUT$(1)) AND 223)="J" THEN 6009 ELSE 6000
6009 IF KONFIGURIERTER_PRINTER$="keiner" THEN 6012
6010 EXEC "sf %n,,"+PR$+",0":LPRINT:LPRINT
6012 IF INKEY$()"" THEN 6012 ELSE COLOR 1,0:CURSOR(1,1)0:PRINT S
PACE$(80);:RETURN
6014 REM EXEC-Aufruf in 6010 !!
```

Wie man aus der letzten Anweisungszeile sieht, kann auch mit Stringvariablen und Stringoperationen bzw. -Funktionen gearbeitet werden.

Diese Möglichkeiten gelten auch bei der Anwendung der **CALL**-Anweisung. Sie unterscheidet sich von **EXEC** dadurch, daß die von der Assembler-Routine benötigten Parameter ggf. auch in Form einer Parameterliste (in Klammern) eingereicht werden können. Variablen müssen dabei vom Typ Integer oder String sein.

Beispiele:

```
 1 CLEAR:ON ERROR GOTO 3020 ' Ermitteln Diskettenname
 10 FU$="getvolname":V$=SPACE$(14)
 20 LINE INPUT "Station (0 oder 1) ";SN$
 30 IF SN$()"1" AND SN$()"0" THEN 20
 40 CALL "bv"(FU$,$V$):'nur Residentmachen bv!
 50 EXEC "-dcons:":FILES SN$+":":EXEC "+dcons:"
 55 'Aktivieren richtige Station!
 60 CALL "bv"(FU$,$V$)
 65 IF V$=SPACE$(14) THEN PRINT "kein ";
 70 PRINT "Diskettenname: ";V$:END
 3010 REM Fehlerbehandlungsroutine
 3020 IF ERL=40 AND ERR=40 THEN PRINT "Systemdiskette einlegen":R
ESUME
 3030 ON ERROR GOTO 0:STOP
```

```
 10 UN%=0:UA%=-1:T$="ck &64,&AF"
 11 CALL T$:CALL "ck"("%f",UN%,$UA%) 'Aufruf aus BASIC-Programm
 12 'CTRL C desaktivieren + Ändern Tastaturfeststellfunktion
 20 IF UA%()UN% THEN PRINT "Feststellfunktionen geändert, beide i
naktiv!"
```

Soll die Assembler-Routine Ergebnisse in bestimmte BASIC-Variablen bringen, kann nur die Form

CALL"Assemblerfilename"(Liste der Parameter)

verwendet werden. Vor den Namen für die einzureichenden Variableninhalte ist in diesem Fall stets das Präfix **§** zu setzen (kein ∅ zwischen **§** und Variablenname). Außerdem müssen diese Variablen vor ihrem ersten Aufruf durch eine wertzuweisende Anweisung einen Wert erhalten haben (z.B. durch eine Wertzuweisung im Vereinbarungsteil); für Strings muß stets sichergestellt werden, daß sie ausreichend lang sind, um später von PCOS den Wert zu übernehmen.

```
10 CALL "pl %n  la":CALL "ss %n,,,,,1":CLEAR
100 SCALE 0,100,0,100
110 PRINT "Folgende Eingaben sind vorzunehmen: "
120 PRINT "1.  Ausserste linke horizontale Punktkoordinate für M
 20             (0 - 100)"
130 PRINT "2.  Unterste vertikale Punktkoordinate für M 20
               (0 - 100)"
140 PRINT "3.  Vergrösserung von M 20 (1 - 16)"
150 PRINT "4.  2. Text (nur CR -->  Spitze!! )
160 PRINT "5.  Ausserste hor. Pktkoord. für 2. Text (0-100)
170 PRINT "6.  unterste vert. Pktkoord. für 2. Text (0-100)
180 PRINT "7.  Vergrösserung von 2. Text (1-16)
200 GOSUB 500:GOSUB 510:GOSUB 520:GOSUB 530:GOSUB 540
210 GOSUB 550:GOSUB 560
270 GOSUB 590
300 CLS:CALL "la"("M 20",CINT(SCALEX(VAL(X1$))),CINT(SCALEY(VAL(
Y1$))),CINT(VAL(V1$)),0)
310 CALL "la"(TEXT$,CINT(SCALEX(VAL(X2$))),CINT(SCALEY(VAL(Y2$))
),CINT(VAL(V2$)),0)
320 K$=INPUT$(1):OHO%=0:CALL "lt"(&OHO%):IF OHO%=1 THEN END ELSE
 RUN
500 CURSOR(1,11):PRINT "1.                                        ";:CU
RSOR(4,11):LINE INPUT X1$
501 IF VAL(X1$)<0 OR VAL(X1$)>100 THEN 500 ELSE RETURN
510 CURSOR(32,11):PRINT "2.                                       ";:CU
RSOR(35,11):LINE INPUT Y1$
511 IF VAL(Y1$)<0 OR VAL(Y1$)>100 THEN 510 ELSE RETURN
520 CURSOR(1,12):PRINT "3.                                        ";:CURSOR(4,12)
:LINE INPUT V1$
521 IF VAL(V1$)<1 OR VAL(V1$)>16 THEN 520 ELSE RETURN
530 CURSOR(1,13):PRINT "4.                                        ";:CURSOR(
4,13):LINE INPUT TEXT$
531 IF TEXT$="" THEN TEXT$="Spitze!!":CURSOR(4,13):PRINT TEXT$;
539 RETURN
540 CURSOR(1,14):PRINT "5.                                        ";:CURSOR(
4,14):LINE INPUT X2$
541 IF VAL(X2$)<0 OR VAL(X2$)>100 THEN 540 ELSE RETURN
550 CURSOR(32,14):PRINT "6.                                       ";:CURSOR(
35,14):LINE INPUT Y2$
551 IF VAL(Y2$)<0 OR VAL(Y2$)>100 THEN 550 ELSE RETURN
560 CURSOR(1,15):PRINT "7.                                        ";:CURSOR
(4,15):LINE INPUT V2$
561 IF VAL(V2$)<1 OR VAL(V2$)>16 THEN 560 ELSE RETURN
```

Fortsetzung s. nächstes Blatt

```
590 CURSOR(1,16):PRINT "ok? (j/n)                    ";:CURSOR(11,1
6):LINE INPUT ;"",JN$
591 IF JN$="j" OR JN$="J" THEN RETURN ELSE IF JN$="n" OR JN$="N"
 THEN 595  ELSE 590
595 CURSOR(1,16):PRINT "Nr.     (0=ok)                    ";:CURSOR
(15,16):LINE INPUT ;"",NR$
596 IF VAL(NR$)<0 OR VAL(NR$)>7 THEN 595 ELSE IF VAL(NR$)=0 THEN
 RETURN ELSE ON VAL(NR$) GOSUB 500,510,520,530,540,550,560:GOTO
595
```

4.3.16 Numerische Funktionen

Normalerweise werden die Ergebnisse aller numerischen Funktionen in einfacher Genauigkeit geliefert.

So wird z.B. für **EXP(1)** = 2.71828 ausgegeben. Der Wert von e mit doppelter Genauigkeit ist E#=2.71828182845905#.

Ergebnisse nicht-trigonometrischer Funktionen sind dann von doppelter Genauigkeit, wenn das Argument von doppelter Genauigkeit ist.

Beispiel:

?VAL("123456789012") liefert 123456789012

Folgende Funktionen stehen zur Verfügung

LOG natürlicher Logarithmus zur Basis e; andere Logarithmen können durch $LOG_B(X)$=**LOG(X)/LOG(B)**
gebildet werden.

EXP Potenz der Zahl e.

SQR Quadratwurzel; nur aus positiven Zahlen

ABS Absolutbetrag

INT Gaußwert, nächster links auf der Zahlenskala liegender, ganzzahliger Wert

FIX Ganzzahlanteil nach Unterdrücken des Nachkommateils

SGN Vorzeichenfunktion

CDBL Konvertierung eines numerischen Werts in einen doppelt genauen Wert gemäß Rundungsvorschriften; vgl. Kapitel 2.5.1.5

CSNG Konvertierung eines numerischen Werts in einen einfach genauen Wert gemäß Rundungsvorschriften, vgl. Kapitel 2.5.1.5

CINT Konvertierung eines numerischen Werts in einen Integer-Wert gemäß Rundungsvorschriften; vgl. Kapitel 2.5.1.5

RND Entwicklung einer Zufallszahl zwischen Ø und 1. Um die Zufallszahlentwicklung besser zu beeinflussen, kann dabei noch die Anweisung **RANDOMIZE** verwendet werden.

Trigonometrische Funktionen liefern das Ergebnis immer in einfacher Genauigkeit. Das Argument ist im Bogenmaß anzugeben. Zur Umrechnung gilt folgende Tabelle:

von \ nach	Altgrad	Bogenmaß	Neugrad
Altgrad	-	*PI#/180	/.9
Bogenmaß	*180/PI#	-	*200/PI#
Neugrad	*.9	*PI#/200	-

Der Variablen PI# muß vorab der Wert 3.14159265358979# zugewiesen werden.

Zur Verfügung stehen die Bibliotheksfunktionen **COS, SIN, TAN** und **ATN.**

Zur direkten Berechnung anderer trigonometrischer Funktionen in einfacher Genauigkeit können die folgenden Ausdrücke benutzt werden:

```
TAN(X)= SIN(X)/COS(X)
SEC(X)=1/COS(X)
CSC(X)=1/SIN(X)
COT(X)=1/TAN(X)
ARCSIN(C)=ATN(X/SQR(-X*X+1))
ARCCOS(X)=-ATN(X/SQR(-X*X+1))+1.5708
ARSEC(X)=ATN(X/SQR(X*X-1))+SGN(SGN(X)-1)*1.5708
```

```
ARCCSC(X)=ATN(X/SQR(X*X-1))+(SGN(X)-1)*1.5708
ARCCOT(X)=ATN(X)+1.5708
SINH(X)=(EXP(X)-EXP(-X))/2
COSH(X)=(EXP(X)+EXP(-X))/2
TANH(X)=EXP(-X)/(EXP(X)+EXP(-X))*2+1
SECH(X)=2/(EXP(X)+EXP(-X))
CSCH(X)=2/(EXP(X)-EXP(-X))
COTH(X)=EXP(-X)/(EXP(X)-EXP(-X))*2+1
ARCSINH(X)=LOG(X+SQR(X*X+1))
ARCCOSH(X)=LOG(X+SQR(X*X-1))
ARCTANH(X)=LOG((1+X)/(1-X))/2
ARCSECH(X)=LOG((SQR(-X*X+1)+1)/X)
ARCSCH(X)=LOG((SGN(X)*SQR(X*X+1)+1)/X)
ARCOTH(X)=LOG((X+1)/(X-1))/2
```

Wenn mit höheren Genauigkeiten gerechnet werden soll, stehen folgende Approximationsformeln zur Verfügung

$$SIN(X)=\sum_{N=1}^{\infty} (-1)^N*(X^(2*N+1))/(2*N+1)!$$ (!=Fakultät)

$$COS(X)=\sum_{N=1}^{\infty} (-1)^N*(X^(2*N))/(2*N)!$$

Zur Berechnung der Fakultät läßt sich folgendes Unterprogramm einsetzen:

```
1000 REM Fakultät von N%;Ergebnis F# = Fakultät von N%
1010 REM Eingangsparameter: N%; Ausgangsparameter: F#
1005 F# = 1
1010 FOR J%=N% TO 2 STEP -1
1020 F# =VAL(STR$(F#*J%))
1030 NEXT:RETURN
```

Um Rundungs- bzw. Konvertierungsdifferenzen möglichst klein zu halten, empfiehlt sich beim Rechnen mit doppelt genauen Werten bzw. mit Werten unterschiedlicher Genauigkeit die Rechnung auf dezimaler Ebene. Auf die Zwischenergebnisse ist **VAL(STR$)...))** anzuwenden (vgl. Anweisung 1020 im vorherigen Beispiel und Beispiel in Kapitel 2.6.1). Dies verlängert allerdings die Rechenzeit.

Eine andere Möglichkeit ergibt sich durch Hinzufügen von "Schutzstellen"; z.B. ist 3.4≈3.4000000000000001.

4.3.17 Deutscher Zeichensatz (ISO-Code-Tabelle)

Adressierung: erst Spalte
dann Zeile

(in () = Sonderfunktionstasten)

hexadezimal (&H) / Dual	0	1	2	3	4	5	6	7
	0000	0001	0010	0011	0100	0101	0110	0111
0	0000	0 NUL	16 DLE	32	48 0	64 §	80 P	96	112 p
1	0001	1 SOH	17 DC1	33 !	49 1	65 A	81 Q	97 a	113 q
2	0010	2 STX	18 DC2	34 "	50 2	66 B	82 R	98 b	114 r
3	0011	3 ETX	19 DC3	35 #	51 3	67 C	83 S	99 c	115 s
4	0100	4 EOT	20 DC4	36 $	52 4	68 D	84 T	100 d	116 t
5	0101	5 ENQ	21 NAK	37 %	53 5	69 E	85 U	101 e	117 u
6	0110	6 ACK	22 SYN	38 &	54 6	70 F	86 V	102 f	118 v
7	0111	7 BEL	23 ETB	39 '	55 7	71 G	87 W	103 g	119 w
8	1000	(⌫) 8 BS	24 CAN	40 (	56 8	72 H	88 X	104 h	120 x
9	1001	9 HT	25 EM	41)	57 9	73 I	89 Y	105 i	121 y
A	1010	10 LF	26 SUB	42 *	58 :	74 J	90 Z	106 j	122 z
B	1011	11 VT	27 ESC	43 +	59 ;	75 K	91 Ä	107 k	123 ä
C	1100	12 FF	28 FS	44 ,	60 <	76 L	92 Ö	108 l	124 ö
D	1101	13 CR	29 CS	45 -	61 =	77 M	93 Ü	109 m	125 ü
E	1110	14 SO	30 RS	46 .	62 >	78 N	94 ^	110 n	126 ß
F	1111	15 SI	31 US	47 /	63 ?	79 O	95 _	111 o	127 DEL

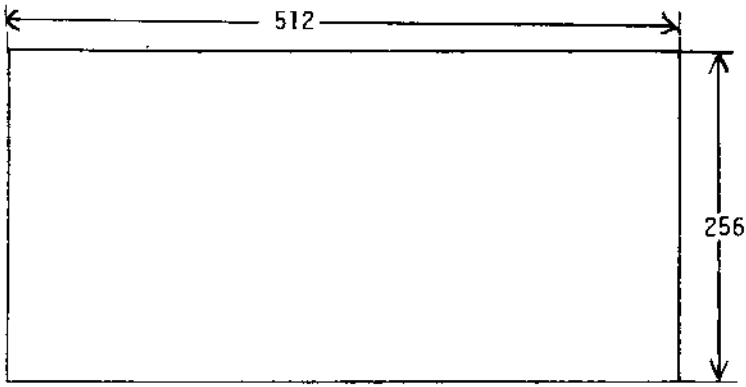
← dezimal

dez. Code:

↓ = 154
← = 155
HOME = 156
→ = 157
↑ = 158

4.3.18 Bildschirmorganisation und Windows

Der M20-Bildschirm ist physisch in ein Raster von 512 x 256 Elementarpunkten (= 131Ø72) eingeteilt.



Darstellungsfläche
auf M20-Bildschirm

Sowohl die graphische als auch die alphanumerische Darstellung basiert auf dem Leuchten oder Nichtleuchten von einzelnen Elementarpunkten innerhalb der 512x256-Punkte-Matrix. Für die Darstellung des Bildschirminhalts steht beim Schwarz/Weiß-Schirm ein Speicherbereich von 16 KB im Systemteil des Speichers zur Verfügung. Für den Vier-Farbschirm werden 32 KB benötigt; 16 KB davon werden im Anwender-Speicher belegt. Für den Acht-Farbschirm werden weitere 16 KB benötigt.

Dieser Speicherbereich wird als Bitmap bezeichnet.

Beim Schwarz/Weiß-Schirm ist ein Bit innerhalb der Bitmap einem Elementarpunkt auf dem Bildschirm zugeordnet:

Bit	Darstellung auf dem Schirm
Ø	Hintergrundfarbe (leuchtet nicht)
1	Vordergrundfarbe (leuchtet)

Beim Vier-Farb-Schirm werden für jeden dargestellten Elementarpunkt zwei Bits in der Bitmap reserviert:

Bits	Darstellung auf dem Schirm
ØØ	Hintergrundfarbe
Ø1	Farbe 1
1Ø	Farbe 2
11	Farbe 3

Es können somit 4 verschiedene Farben (zur Auswahl aus insgesamt 8 Farben) gleichzeitig dargestellt werden.

Beim Acht-Farb-Schirm werden für jeden Elementarpunkt drei Bits in der Bitmap reserviert.

Die Zeichenkapazität auf dem Bildschirm ist variabel und kann per BASIC-Programm und/oder über PCOS festgelegt werden (PCOS-Befehl ss bzw. BASIC-Funktion **WINDOW**).

Zeichenkapazität	Anz. Zeichen pro Zeile	Anzahl Zeilen	Anz. Zeichen auf Schirm
minimal	64	16	1024
maximal	80	25	2000

Im folgenden soll die Zeichendarstellung, die Bestimmung der Anzahl Zeichen pro Zeile und die Festlegung der Anzahl Zeilen des Schirms erläuter werden.

4.3.18.1 Zeichendarstellung und Bildschirmorganisation

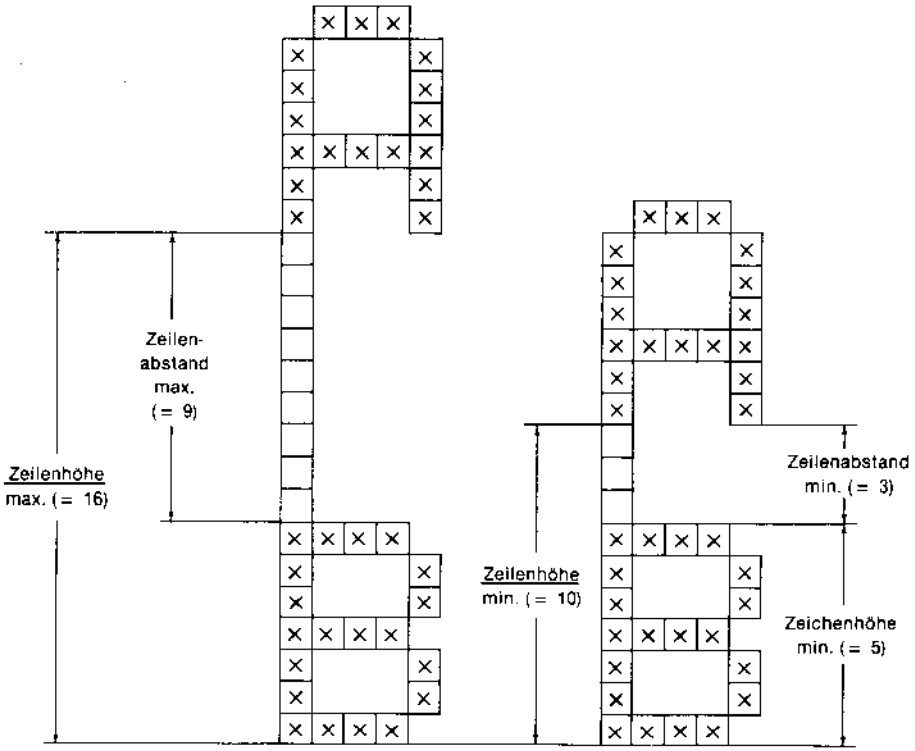
Die Zeichen sind in einer 8 x 10-Punktmatrix definiert, von der im allgemeinen nur 5 x 7 Punkte belegt werden.

Sonderfälle bilden z.B. Buchstaben mit Unterlänge wie g, q und j oder mit Oberlänge wie die Umlaute Ä, Ü und Ö.

Die den ISO-Codes zugeordneten Punktmatrizen können (nur für den Bildschirm) mit Hilfe des PCOS-Befehls **rf** in ein Datenfile eingelesen werden, mit Hilfe des Full-Screen-Editors (PCOS-Befehl **ed**) geändert bzw. in ihrer Anzahl erweitert werden (z.B. zur eigenen Definition von neuen Graphik-Zeichen). Mit dem PCOS-Befehl **wf** kann ein solcher neu erstellter Bildschirmzeichensatz aktiviert werden. (Solche neuen Zeichensätze gelten nicht für die Drucker.)

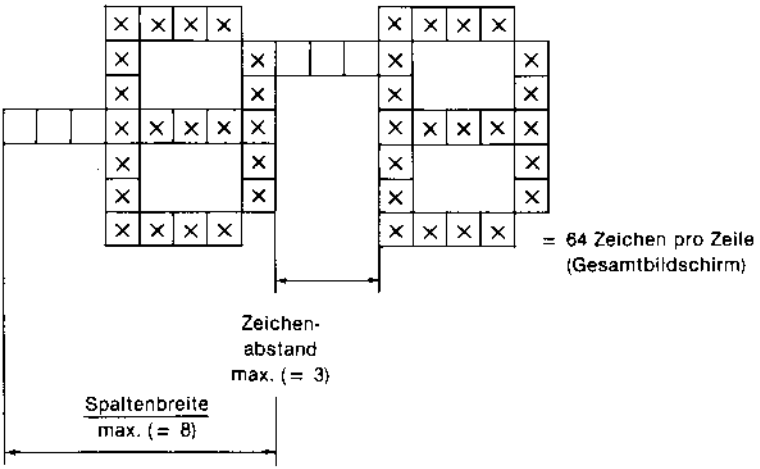
Bestimmung der Anzahl Zeilen im Bildschirm

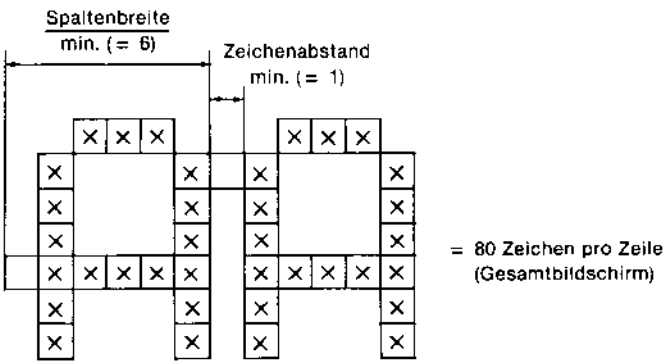
Die Anzahl Textzeilen im Gesamt-Bildschirm kann zwischen 16 und 25 variiert werden. Die Zeilenhöhe setzt sich zusammen aus der Höhe eines Zeichens (zu kalkulieren sind 10 Elementarpunkte) plus einem zusätzlich möglichen Zeilenabstand (Ø - 6 Elementarpunkte).



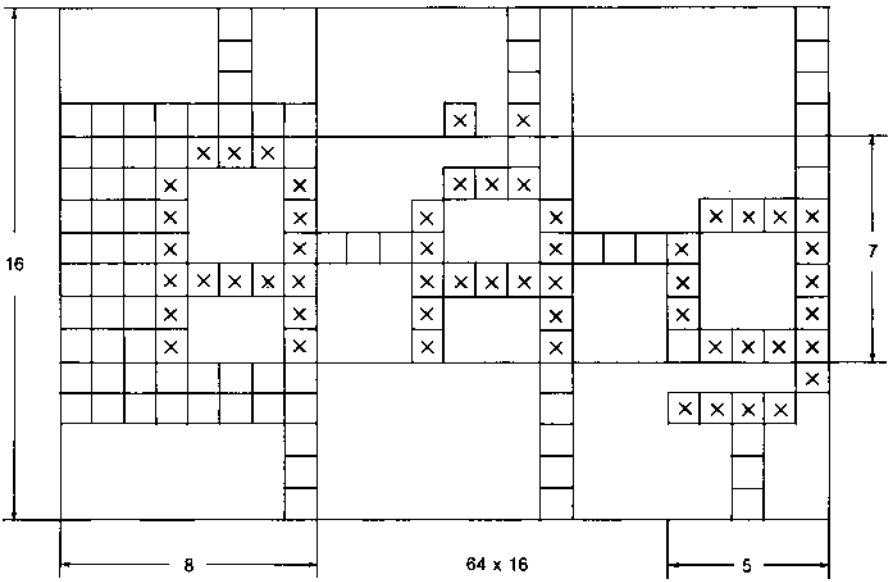
Bestimmung der Anzahl Zeichen pro Zeile

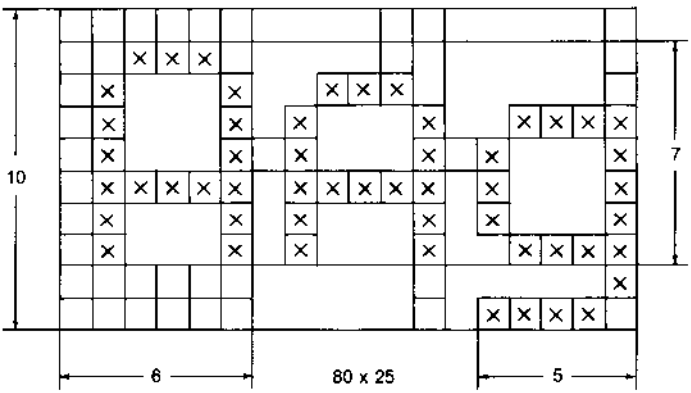
Die Anzahl Textzeichen pro Zeile ist festgelegt durch die Anzahl Elementarpunkte einer Textspalte. Dabei kann die Spaltenbreite entweder 6 Elementarpunkte (80-Zeichen-Window) oder 8 Elementarpunkte (64-Zeichen-Window) umfassen. Von diesen werden normalerweise nur die 5 am weitesten rechts liegenden zur Punktdarstellung verwendet; beim 80-Zeichen-Window werden die zwei Punktspalten ganz links außen nicht abgebildet.





Zusammenfassung der Zeichendarstellung bei 64 x 16- und 80 x 25-Bildschirmformat





4.3.18.2 Arbeit mit Windows

Per Definition ist der M20-Bildschirm als ein Window zu betrachten, aus dem bis zu 16 unabhängige Windows durch Teilung gebildet werden können. Diese Windows können als "eigene Bildschirme" betrachtet werden, die unabhängig voneinander angesprochen werden können. Das Anlegen von WIndows geschieht mit der Funktion **WINDOW**. Innerhalb der Windows kann all das gemacht werden, was auf dem Gesamtbildschirm möglich ist; d.h., es existiert in jedem Window ein Text- und ein Graphik-Cursor; ein Window kann einzeln aufgelöst werden (Anweisung **CLOSE WINDOW**). Es ist möglich, eine Hardcopy für ein einzelnes Window zu machen (nur bei graphikfähigem Drucker; PCOS-Befehl sp), den Text in einem Window auszugeben (PCOS-Befehl ls), ein eigenes Scrolling durchzuführen oder die Anzahl Zeilen bzw. Anzahl Zeichen pro Zeile oder eine eigene Skalierung für jedes Window festzulegen.

Anlegen von Windows

Nach dem Laden des Systems ist der gesamte Bildschirm als ein Window zu betrachten. Er hat dann die Window-Nr. 1.

Das Anlegen ("Eröffnen") eines neuen Windows erfolgt durch Teilung eines bestehenden Windows. Dabei wird festgelegt,in welcher Form die Teilung erfolgt und wie groß das neue Window sein soll.

Das ursprüngliche Window bleibt mit seiner Nummer erhalten und wird in seiner Größe so reduziert, daß es den nach der Abspaltung des neuen Windows verbleibenden Rest umfaßt.

Auflösen von Windows

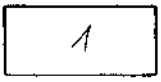
Windows können einzeln wieder aufgelöst werden. Dabei wird der freiwerdende Bereich wieder dem Window zugeordnet, aus dem das Window hervorgegangen ist.

Beispiel:

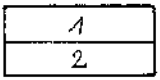
Es werden vier Windows vereinbart und der daraus entstehende Bildschirmaufbau dargestellt:

Bildschirm:

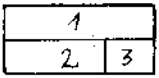
1)



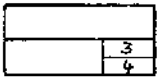
2)



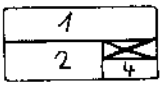
3)



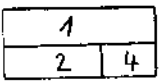
4)



Wird jetzt Window Nr. 3 gelöscht, ergibt sich:



und daher:



Aktivieren eines Windows

Jedes Window stellt einen logisch selbständigen Bildschirm dar.

Da immer nur mit einem Bildschirm gearbeitet werden kann, muß jeweils eines der geöffneten Windows als aktives Window vereinbart werden (Anweisung **WINDOW%**).

Alle Anweisungen und Meldungen, die sich auf den Bildschirm beziehen und nicht einen eigenen Parameter 'Window-Nr.' haben, gelten immer nur für das aktive Window. Die dabei geltenden Ausgabeformate sind durch die Grenzen des aktiven Windows bestimmt.

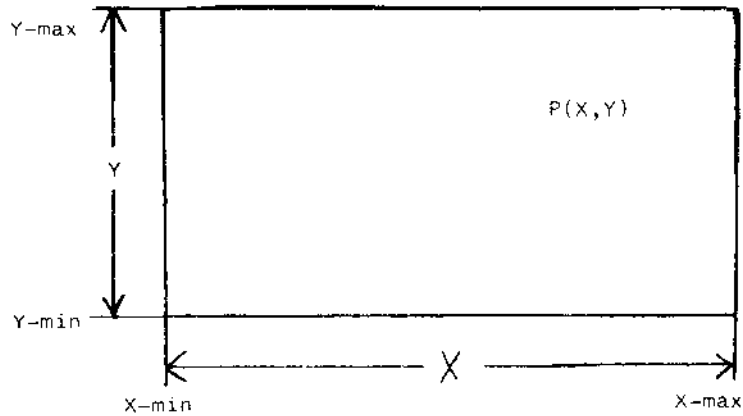
Die Inhalte der nicht aktiven Windows bleiben unverändert bestehen. Das System merkt sich auch alle Informationen über die Grösse, den Wertebereich und die Position der Cursor in diesen Windows. Wird ein solches Window erneut aktiviert, können die alten Werte direkt weiter verwendet werden.

Die genaue Beschreibung der Regeln über das Anlegen, Auflösen und Aktivieren von Windows findet sich bei den Anweisungen **WINDOW %**, **CLOSE WINDOW** und **CLEAR** sowie der Funktion **WINDOW**.

Arbeitsweise innerhalb von Windows

Im Window kann gleichzeitig alphanumerisch (Text) und graphisch gearbeitet werden. Beim alphanumerischen Arbeiten (Alpha-Mode) kann unabhängig von anderen Windows die Anzahl Zeichen pro Zeile (80 oder 64, bezogen auf den Gesamt-Bildschirm) und die Anzahl Zeilen (16 bis 25, bezogen auf den Gesamt-Bildschirm) bestimmt werden. Dies wird mit Hilfe der **WINDOW**-Funktion festgelegt. Der Text-Cursor kann durch die Anweisung **CURSOR** unter Angabe von Spalte und Zeile an beliebiger Stelle positioniert und dargestellt werden. Die Position (1,1) (d.h. Spalte 1, Zeile 1) befindet sich in der linken oberen Ecke des Windows. Näheres zum alphanumerischen Arbeiten siehe Kapitel 4.3.9.1.

Für jedes Window kann ein anwenderbezogener Koordinatenbereich definiert werden. Dazu werden über die Anweisung **SCALE** die minimalen und maximalen X- und Y-Koordinaten angegeben.



Die Default-Werte für die Koordinaten innerhalb eines Windows ohne **SCALE**-Anweisung sind:

X-min	Ø
Y-min	Ø
X-max	Anzahl horizontaler Elementarpunkte - 1
Y-max	Anzahl vertikaler Elementarpunkte - 1

Die Default-Werte von **SCALE** für den gesamten Schirm sind:

bei 64 Zeichen pro Zeile:

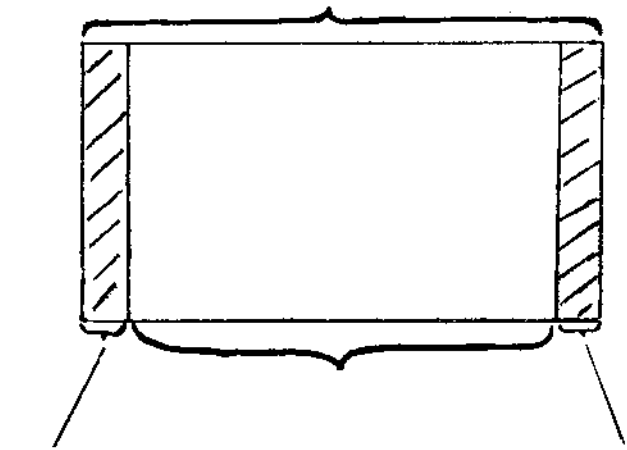
X-min	Ø
Y-min	Ø
X-max	511
Y-max	255

bei 80 Zeichen pro Zeile:

X-min	Ø
Y-min	Ø
X-max	479
Y-max	255

(64 Zeichen-Schirm)

64x8=512 Punkte



16 Punkte 80x6=480 Punkte 16 Punkte

(80-Zeichen-Schirm)

Skizze:

Matrix der Elementarpunkte für beide Bildschirm-Formate

Der graphische Cursor kann auf einen beliebigen Elementarpunkt innerhalb eines Windows positioniert werden (Anweisung **CURSOR** mit Parameter **POINT**). Die Position **(Ø,Ø)** befindet sich in dem Elementarpunkt in der linken unteren Ecke, falls mit **SCALE** nicht eine andere Koordinateneinteilung definiert wurde. Näheres über Graphik-Anweisungen siehe Kapitel 4.3.14.

4.3.18.3 Verwendung von Farbe

Beim Schwarz/Weiß-Bildschirm stehen die Farben schwarz und weiß zur Darstellung der Elementarpunkte zur Verfügung. Dabei unterscheidet man in Hintergrund- und Vordergrund-Farbe. Die Hintergrund-Farbe ist die Farbe, welche der Bildschirm nach Ausführung der Anweisung **CLS** annimmt. Der Default-Wert für die Hintergrundfarbe beim Schwarz/Weiß-Schirm ist schwarz.

Schwarz/Weiß-Schirm

Farbcode	Farbe
Ø	schwarz
1	weiß

Beim mehrfarbigen Schirm stehen für die Darstellung der Elementarpunkte acht Farben zur Verfügung. Aus diesen acht Farben können beim Vier-Farb-Schirm mit der Anweisung **COLOR=** (globale **COLOR**-Anweisung) vier Farben ausgewählt werden, mit denen gleichzeitig gearbeitet werden kann. Den Farben ist ein Farbcode (Ø-7) zugeordnet, mit dem diese Farben angesprochen werden. Damit wird gleichzeitig für den Vier-Farb-Schirm eine Hintergrund- und eine Vordergrundfarbe bestimmt. Der Defaultwert für die Hintergrundfarbe ist schwarz (Ø) und für die Vordergrundfarbe grün (1).

Am Schwarz/Weiß- und am Acht-Farb-Schirm kann die Bestimmung von Hinter- bzw. Vordergrundfarbe nur durch die Anweisung **COLOR** erfolgen (andere Anweisung als **COLOR=**).

Mehrfarbiger Bildschirm

Farbcode	Farbe
Ø	schwarz
1	grün
2	blau
3	cyan (türkis)
4	rot
5	gelb
6	magenta (purpur)
7	weiß

Beim Vier-Farb-Schirm ist in der Anweisung **COLOR** nicht der Farbcode anzugeben, sondern die Position (Ø-3) der Farben in der Anweisung **COLOR=** (von links abzählend). Diese Position wird in den Format-Beschreibungen mit 'Farbindex' bezeichnet. Für Schwarz/Weiß- und Acht-Farb-Schirm decken sich 'Farbcode' und 'Farbindex'. Mit dem 'Farbindex' können bei den Graphik-Anweisungen **LINE, CIRCLE, DRAW, PAINT** und **PSET** bestimmte Farben spezifiziert werden.

Mit der Funktion **POINT** läßt sich der Farbindex in einem bestimmten Punkt abfragen.

5. PROGRAMMIERUNG PERIPHERER EINHEITEN

5.1 Parallel-Schnittstelle (Centronics)

In der Grundausstattung des M20 ist eine Parallel-Schnittstelle vorhanden. An diese können Geräte angeschlossen werden, die eine Centronics-ähnliche Parallelschnittstelle haben.

Standardmäßig sind dort die Olivetti-Drucker, wie z.B. PR1471, PR2400, PR1450, PR1481 oder PR2300 angeschlossen. (Davon sind vorläufig nur die Drucker PR1450 und PR2400 graphikfähig, d.h., über sie können auch Hardcopies des gesamten Bildschirms oder auch nur eines Windows ausgegeben werden (PCOS-Befehl **sp**)). Die Hardcopy muß per Programm vorgesehen werden und kann nicht zu jedem beliebigen Zeitpunkt durch Eingriff von außen abgerufen werden. Von den Texten im Bildschirm kann über den PCOS-Befehl **ls** an allen Druckern eine Abbildung erreicht werden. Auch **ls** kann nur im Programm selbst aufgerufen werden.

Die Ausgabe über derart angeschlossene Drucker erfolgt mit **LPRINT**-Anweisungen. Die Druckerparameter sind mit dem PCOS-Befehl **sf** zu setzen. Es ist auch möglich, die Drucker seriell anzuschließen. Dazu müssen die Drucker mit serieller Schnittstelle ausgestattet sein und über die PCOS-Befehle **rs**, **sc** und **sf** entsprechend konfiguriert werden. Ggf. kann über Wechsel des Ein-/Ausgabemediums auch das am Drucker ausgegeben werden, was normalerweise nur am Bildschirm erscheint (**PRINT**- bzw. **WRITE**-Anweisungen, **INPUT**- bzw. **LINE INPUT**-Anweisungen und Eingaben im Command-Mode bzw. in der PCOS-Ebene). Vgl. hierzu das Hanbuch "PCOS Betriebssystem".

Durch Änderung der Stellung von Schaltern ("dip-switches") innerhalb der Drucker können z.B. nationale Zeichensätze eingestellt werden (am PR1450 und am PR2300 ist dies sogar über BASIC-Programm möglich), grundsätzliche Zeilenschaltungen (LF) vorgeschrieben werden u.ä.. Sie finden darüber Informationen im jeweiligen Druckerhandbuch. Normalerweise werden solche Änderungen durch den Technischen Kundendienst von OLIVETTI vorgenommen.

Zahlreiche Druckerparameter (Zeilen/Seite, Fettdruck etc) können mit Hilfe von **LPRINT** über BASIC gesetzt werden. Die dabei möglichen Funktionen für die einzelnen Drucker sowie zugeordneten Steuerzeichen(folgen) sind dem Kapitel 7, ANHANG II sowie -detaillierter- den jeweiligen Druckerhandbüchern zu entnehmen.

Beispiele

LPRINT CHR$(7);
erzeugt am PR1471 ein akustisches Signal (ohne Zeilenschaltung).

LPRINT CHR$(12)
bewirkt am PR1471 und PR1450 einen Seitenvorschub. Dieser ist abhängig von der gesetzten Anzahl Zeilen/Seite (nicht über **sf**, sondern über **LPRINT**!)

LPRINT CHR$(27)+CHR$(51)
legt am PR1471 Fettdruck fest; dieser muß mit

LPRINT CHR$(27)+CHR$(52)
aufgehoben werden.

LPRINT CHR$(27)+CHR$(81)+"Ø72"+CHR$(27)+CHR$(9Ø)
legt am PR1471 fest, daß bei Seitenvorschüben von 72 Zeilen/Blatt ausgegangen wird (Parameter 'Anz.Zeilen/Seite' bei **sf** beachten!!)

LPRINT CHR$(27)+"Q12Ø;Ø15;Ø5Ø"+CHR$(27)+"Z"

legt am PR1471 fest, daß mit 120 Zeichen/Zeile gearbeitet wird und an den Spalten 15 und 50 Tabulationsstops gesetzt sind.

LPRINT CHR$(27)+"G13"+CHR$(2^Ø+2^1+2^3+2^6)

aktiviert die Plotfunktion am PR1450 und liefert die folgende Punktspalte

1. Punkt gesetzt (2^Ø)
2. Punkt gesetzt (2^1)
3. Punkt nicht gesetzt
4. Punkt gesetzt (2^3)
5. Punkt nicht gesetzt
6. Punkt nicht gesetzt

Da das 7. Bit (2^ 6) gesetzt ist, bleibt die Plotfunktion aktiv.

5.2 V.24 - (RS232C-)Schnittstelle

Als serielle Schnittstelle ist standardmäßig eine V.24-(RS232C-)-Schnittstelle vorhanden. Über eine Erweiterungsplatine können entweder zwei weitere V.24-Schnittstellen oder eine weitere V.24-Schnittstelle und eine 20mA-Schnittstelle (Current-Loop) oder zwei 20mA-Schnittstellen vorgesehen werden. An alle V.24-Schnittstellen können auch Drucker angeschlossen werden; notfalls auch solche, die normalerweise über die Parallel-Schnittstelle bedient werden. Diese Drucker müssen dann mit dem PCOS-Befehl **sf** als serielle Drucker definiert werden.

Für die Programmierung solcher V.24-Einheiten sind die folgenden PCOS-Befehle nötig:
rs, sc, ci (wobei **ci** über **CALL** aus BASIC aufzurufen ist) und ggf. **sf** und **sd.**

Vergleiche Handbücher "PCOS Betriebssystem" und "V.24-Peripherie"

5.3 IEEE 488 (IEC-Bus)

Mit einer Erweiterungsplatine wird eine IEEE 488-Schnittstelle (ebenfalls paralleler Art) möglich.
Die Programmierung solcher IEEE-Einheiten geschieht - nach Verwendung des PCOS-Befehls **ie** - über spezielle IEEE-BASIC-Anweisungen:

ISET, IRESET, ON SRQ GOSUB, POLL, WBYTE, PRINT§, RBYTE, INPUT§ und **LINE INPUT§**

Die Beschreibung dieser Anweisung und der Voraussetzungen bezüglich der Systemumgebung entnehmen Sie bitte dem Handbuch "IEC-Schnittstelle".

6. MELDUNGEN DES INTERPRETERS: FEHLER-CODES

6.1 Liste der Fehler-Codes (BASIC-Fehler)

Die folgende Liste ist eine Zusammenstellung der Meldungen des BASIC-Interpreters (BASIC-Fehler)

ERROR-Nr.	MELDUNG
1	NEXT without FOR
2	Syntax error
3	RETURN without GOSUB
4	Out of DATA
5	Illegal function call
6	Overflow
7*	Out of memory
8	Undefinded line number
9	Subscript out of range
10	Duplicate Definition
11	Division by zero
12	Illegal direct
13*	Type dismatch
14	Out of string space
15	String too long
16	String formula too complex
17	Can't continue
18	Undefinded user function
19	No RESUME
20	RESUME without error
22	Missing operand
23	Line buffer overflow
26	FOR without NEXT

ERROR-Nr.	MELDUNG
29	WHILE without WEND
30	WEND without WHILE
31	IEEE: Invalid talker listener adress
32	IEEE: talker = listener adress
33	IEEE: unprintable error
34	IEEE: Board not present
35	Window not open
36	Unable to create window
37	Invalid action-verb
38	Parameter out of range
39	Too many dimensions
40	x)
50	FIELD overflow
51	Internal error
52	Bad file number
53*	File not found
54*	Bad file mode
55*	File already open
57*	Disk I/O error
58*	File already exists
61*	Disk full
62*	Input past end
63*	Bad record number
64*	Bad file name
66	Direct statement in file
67*	Too many files
68	Internal error
69	Volume name not found
70	Rename error
71*	Volume number error

ERROR-Nr.	MELDUNG
72	Volume not enabled
73*	Invalid Password
74*	Illegal disk change
75*	Write Protected
76*	Error in Parameter
77*	Too many parameters
78*	File not OPEN
79*	Printer error

Die mit * versehenen Felder können auch von PCOS gemeldet werden, haben dort aber evtl. andere Bedeutung (vgl. PCOS-Handbuch).

X) Es wird nur "in 'Zeilennr.'" gemeldet. Es ist ein PCOS-Fehler aufgetreten. Der Fehler 4Ø kann auf **ERR** nicht mit Hilfe der **ERROR**-Anweisung zugewiesen werden. Nicht bei allen PCOS-Fehlern erhält **ERR** den Wert 4Ø.

Die nicht angeführten Error-Nummern bewirken die Meldung "Unprintable error". Sie können im Anwenderprogramm beliebig besetzt werden (**ERROR**-Anweisung und Fehlerbehandlungsroutine).

Beinhaltet die reservierte Variable **ERR** den Inhalt Ø, ist entweder seit dem Laden des Systems über die PCOS-Ebene kein Fehler aufgetreten oder zuletzt eine Fehlerbehandlungsroutine mit **RESUME** verlassen worden.

6.2 Erläuterung der Fehler-Codes (BASIC-Fehler)

ERROR-Nr.	BEDEUTUNG
1	Es wurde ein **NEXT** erreicht, dem kein **FOR** zugeordnet ist (fehlendes Stack).
2	Es wurde ein Syntaxfehler erkannt. Das System verzweigt in den Edit-Mode und zeigt sofort die entsprechende Zeile (Zeilennummer) an. Der Syntaxfehler kann mit den Möglichkeiten des Edit-Modes behoben werden. Sollen vor der Behebung noch Variablenwerte geprüft werden, muß der Edit-Mode mit Q verlassen werden, da andernfalls die Variableninhalte gelöscht werden. Wurde das Programm mit **SAVE** und Parameter **P** gespeichert, erfolgt zusätzlich die Meldung "Illegal function call".
3	Es wurde ein **RETURN** erreicht, ohne daß der Rücksprung aus dem Unterprogramm möglich ist (fehlendes Stack wegen fehlendem **GOSUB**).
4	Es wird versucht, mehr Daten aus **DATA**-Anweisungen zu lesen als verfügbar sind.
5	Unzulässiger Aufruf einer Funktion. Dies kann durch falsche Angabe der Parameter oder durch Ausführung von Anweisungen oder Befehlen, die für ein geschütztes Programm unzulässig sind, verursacht sein. a) negativer oder zu großer Array-Index b) **LOG**-Argument ≤ 0 c) **SQR**-Argument < 0 d) Nicht-Integer-Exponent bei negativer Basis

ERROR-Nr.	BEDEUTUNG
	e) falsches Argument in: **MID$, LEFT$, RIGHT$, TAB, SPC, SRING$, SPACE$, INSTR, ASC, CHR$, ON...GOTO** oder **ON...GOSUB**
6	Erzeugung einer zu großen Zahl. Es wird die größtmögliche Zahl gesetzt und mit dieser Zahl unmittelbar fortgesetzt.
7	Überlauf im Speicher. Das kann durch zu große Datenmengen, zu viele offene Stacks oder ein zu großes Programm verursacht werden.
8	Es wird versucht, eine nicht vorhandene Programmzeile anzusprechen.
9	Es wird versucht, ein Arrayelement mit einem unzulässigen Index anzusprechen. Entweder liegt der Index über 10 und ist nicht über eine **DIM**-Anweisung als zulässiger Index festgelegt oder der Array ist mit **ERASE** aufgehoben worden, oder es liegt die Anweisung **OPTION BASE** 1 zugrunde, obwohl ein Index 0 angesprochen wird. Der Fehler kann auch auftreten, ohne daß einer dieser Fälle vorliegt. Nach Einfügen einer Anweisungszeile (z.B. **REM** xxx) direkt vor der den Fehler verursachenden Zeile wird der Fehler nicht mehr auftreten.

ERROR-NR.	BEDEUTUNG
10	Ein bereits dimensionierter Array soll zum zweiten Mal dimensioniert werden. Entweder soll eine zweite **DIM**-Anweisung für einen Array ausgeführt werden oder die **DIM**-Anweisung steht erst nach der ersten Verwendung des Arrays. Der Fehler kann auch auftreten, ohne daß einer dieser Fälle vorliegt. Nach Einfügen einer Anweisungszeile (z.B. **REM** xxx) direkt vor der den Fehler verursachenden Zeile wird der Fehler nicht mehr auftreten.
11	Es soll durch 0 dividiert werden. Es wird die größte darstellbare Zahl angenommen (abhängig vom Typ der Operanden) und unmittelbar damit weitergerechnet.
12	Es wird versucht, eine Anweisung im Direkt-Mode auszuführen, die nur in Programmen verwendet werden kann (z.B. **DEF**).
13	Es wird versucht, einem String einen numerischen Wert oder einer numerischen Variablen einen String zuzuweisen oder unterschiedliche Typen zu vergleichen oder falsche Typen als Parameter einer Anweisung zu verwenden.
14	Die Summe der Längen der verwendeten Strings übersteigt den dafür benötigten Speicherbereich. Mit der Funktion **FRE**(Stringausdruck) kann der Inhalt des Arbeitsspeichers neu geordnet werden.
15	Durch Ausführen der Operation würde der String mehr als 255 Zeichen lang werden.

ERROR-Nr.	BEDEUTUNG
16	Bei Ausführung der Stringoperationen wird eine zu tiefe Schachtelung des Stacks erreicht.
17	Es wurde **CONT** eingegeben, es ist jedoch die Information über das Programm verloren (z.B., wenn nach einem Syntaxfehler durch Edit-Mode das Stack gelöscht wurde oder eine neue Programmzeile aufgenommen wurde) oder keine Programmzeile mehr zur Ausführung vorhanden.
18	Aufruf einer selbstdefinierten Funktion, die nicht zuvor mit **DEF FN** vereinbart wurde.
19	Es wird aufgrund eines Fehlers in eine Fehlerbehandlungsroutine gesprungen, die kein **RESUME** enthält.
20	Es soll eine **RESUME**-Anweisung ausgeführt werden, ohne daß zuvor aufgrund eines Fehlers in die Fehlerbehandlungsroutine gesprungen wurde.
22	Es fehlt ein Operand nach einem Operator.
23	Es wurde versucht, eine Anweisungszeile mit mehr als 255 Zeichen einzugeben oder ein Random-File mit **LOAD** in den Arbeitsspeicher zu laden.
26	Es soll ein **FOR** ausgeführt werden, ohne daß ein zuzuordnendes **NEXT** gefunden wird. Die **NEXT**-Anweisung wird unmittelbar bei der Ausführung von **FOR** gesucht.
29	Es soll ein **WHILE** ausgeführt werden, ohne daß ein zuzuordnendes **WEND** vorhanden ist. Die **WEND**-Anweisung wird unmittelbar bei der Ausführung von **WHILE** gesucht.

ERROR	BEDEUTUNG
30	Es soll ein **WEND** ausgeführt werden, ohne daß zuvor ein **WHILE** ausgeführt wurde.
31	Verwendung einer unzulässigen Sender/Empfänger-Adresse (IEEE 488).
32	Es wurde versucht, eine Sender-Adresse mit einem Empfangsbefehl oder eine Empfänger-Adresse mit einem Sendebefehl anzusprechen (IEEE 488).
33	Es wurde ein Fehler erzeugt, dem keine Fehlermeldung entspricht und ohne daß eine Fehlerbehandlungsroutine auszuführen ist (IEEE 488).
34	Es wird versucht, eine IEEE-Schnittstelle anzusprechen, obwohl keine vorhanden ist (IEEE 488).
35	Es wird ein Window angesprochen, das nicht eröffnet wurde.
36	Das Eröffnen eines Windows ist nicht möglich weil: - entweder in **sb** eine geringere Anzahl vereinbart wurde oder - dieses Window bereits existiert oder - die Parameter fehlerhaft sind (z.B. die Teilung außerhalb des aktiven Windows erfolgen müßte).
37	In einer **LINE-, CIRCLE**- oder **PUT%**-Anweisung ist für 'Operand' ein zulässiger Operand angegeben.
38	Ein Parameter überschreitet die zulässigen Grenzen. Diese Meldung tritt auch auf, wenn der in **GET%** angegebene Array nicht genügend Platz für die Speicherung des Bildes bietet (kein ausreichend hoher Index bei **DIM**).

ERROR-Nr.	BEDEUTUNG
39	Es wird ein Array mit mehr Dimensionen angegeben, als für die Anweisung zulässig ist (z.B. bei **GET%**).
40	Während der Auführung von **EXEC** oder **CALL** ist ein PCOS-Fehler entstanden.
50	Für ein Random-File wird versucht, den Random-File-(**FIELD**)-Puffer mit mehr Zeichen zu definieren, als beim Öffnen des Files mit **OPEN** unter 'Recordlänge' vereinbart wurde.
51	Nicht-spezifizierte Meldung (interner Fehler). Bitte berichten Sie die Fehler-Konditionen an die Olivetti-Zentrale (Zentrale Software) in Frankfurt.
52	Es wird versucht, ein File anzusprechen, das nicht geöffnet ist, oder der Wert für 'Filenummer' liegt außerhalb des gültigen Bereichs (größer als 15 oder größer als in **sb** gesetzt).
53	Es wird kein File mit dem verlangten Namen gefunden (falsche Schreibweise des Filenamens oder falsche Diskette angesprochen, z.B. weil die Diskettenspezifikation nicht angegeben ist und zuletzt eine andere Diskette angesprochen wurde).
54	Es wird versucht, eine Anweisung für ein File eines anderen Typs auszuführen (z.B. **GET** für sequentielles File) oder es ist in **OPEN** eine falsche 'Zugriffsart' angeführt.

ERROR-Nr.	BEDEUTUNG
55	Es wird versucht, eine bereits vergebene 'Filenummer' erneut zu vergeben oder ein offenes File mit **KILL** zu löschen. Der Fehler kann auch auftreten, weil ein anderes File offen ist. Er ist dann durch Eingabe von **CLOSE** im Direkt-Mode zu beheben.
57	Beim Zugriff auf Diskette tritt ein Übertragungsfehler auf. Dieser Fehler weist auf beschädigte Disketten oder eine schlecht justierte Diskettenstation hin. Möglicherweise liegt in der angesprochenen Station keine oder eine nicht formatierte Diskette oder eine solche mit Schreibschutz ein. Es ist Datenverlust anzunehmen!
58	Es wird versucht, einem File einen Namen zu geben, der auf der Diskette bereits existiert.
61	Die Diskette ist voll. Es sind weniger Sektoren verfügbar als aufgrund des gültigen Parameters 'Anzahl Sektoren' in **ss** angelegt werden sollen. Es ist anzunehmen, daß nicht alle Daten abgespeichert wurden.
62	Es wird versucht, aus einem sequentiellen File mehr Daten zu lesen als im File verfügbar sind. Die **EOF**-Funktion kann zur Prüfung eingesetzt werden.
63	Die Angabe der Record-Nummer für ein Random-File liegt außerhalb des gültigen Bereichs (von 1 bis 32767).
64	Fehlerhafter Aufbau des Filenamens, z.B. zu lang oder mit unzulässigen Zeichen.

ERROR-NR.	BEDEUTUNG
66	Die Meldung tritt auf, wenn im Speicher Programmzeilen ohne Zeilennummer auftraten (z.B. nach Laden einer Assembler-Routine oder eines sequentiellen Datenfiles in den Arbeitsspeicher mit **LOAD**).
67	Es wird versucht, ein weiteres File anzulegen, obwohl im Inhaltsverzeichnis der Diskette kein Platz mehr ist (max. 196 File-Namen bei 320-KB-Disketten, bei HD erheblich mehr).
68	Nicht-spezifizierte Meldung (interner Fehler).
69	Bei der Diskettenspezifikation eines Filenamens wurde die Diskettenbezeichnung einer Diskette eingegeben, die nicht einliegt.
70	Unzulässige neue Diskettenbezeichnung beim Umbenennen einer Diskette.
71	Bei einer Diskettenspezifikation wurde eine nicht zulässige Stationsbezeichnung angegeben.
72	Bei einer Diskettenspezifikation wurde das Disketten-Password nicht oder unrichtig angegeben.
73	Bei einer Filespezifikation wurde das File-Password nicht oder unrichtig angegeben.
74	Während auf einer Diskette ein File offen ist, wurde die betreffende Diskette ausgetauscht. Nachdem die richtige Diskette wieder eingelegt ist, kann weitergearbeitet werden.

ERROR-NR.	BEDEUTUNG
75	Das File, das zum Schreiben angesprochen wurde, ist durch PCOS-Befehl **fw** mit Schreibschutz versehen worden. Der Fehler kann auch auftreten, wenn einer Systemdiskette ein Disketten-Password zugeordnet wurde. Sind Disketten niemals mit **vf** formatiert worden, wird beim Kopierversuch ebenfalls Error 75 gemeldet. Diese Disketten können nicht mehr kopiert werden.
76	Ein Parameter wurde nicht korrekt angegeben.
77	Es wurden zuviele Parameter angegeben.
78	Für die File-Nummer, unter der eine Lese- oder Schreiboperation durchgeführt werden soll, ist keine **OPEN**-Anweisung mehr gültig.
79	Der Drucker steht auf LOCAL oder ist nicht aktiv.

6.3 Überblick über die PCOS-Fehler

CODE	BEDEUTUNG	ERLÄUTERUNG
7	out of memory	es wurde ein Programm aufgerufen, das für den zur Verfügung stehenden Speicherplatz zu groß ist; kann auftreten, weil ein PCOS-Befehl die dafür vorgesehene Segmentgrenze des Arbeitsspeichers überschreitet
13	bad data type	der Typ eines Parameters ist nicht zulässig, z.B. keine Integer-Variable bei **CALL "lt"**
35	non-existent window	ein nicht eröffnetes Window wurde angesprochen
53	file not found	das angegebene File wird nicht gefunden (z.B., weil zuletzt eine andere Station angesprochen wurde und die Diskettenspezifikation beim Filenamen fehlt)
54	bad file open mode	das angesprochene File ist vom falschen Typ (z.B., weil ein sequentielles File benötigt wird, aber ein Random-File vorliegt)
55	file already open	das angesprochene File ist noch offen (wegen Ein- bzw. Ausgabe an dieses File mit Hilfe von "Wechsel des Ein/Ausgabe-Mediums"); das File kann nur durch Anwendung des PCOS-Befehls -sFilename bzw. -d[Filename] geschlossen werden

57	disk i/o error	Schreib- oder Lese-Fehler auf der angesprochenen Diskette; evtl. ist logischer Reset nötig; dann ist Datenverlust zu befürchten! Evtl. liegt auf der angesprochenen Station keine Diskette ein oder die einliegende Diskette ist noch nicht korrekt mit **vf** formatiert oder ist mit Schreibschutz versehen.
58	file already exists	der angeführte Filename existiert bereits auf der angesprochenen Diskette
60	disk not initialized	die angesprochene Diskette ist noch nicht mit **vf** formatiert worden
61	disk filled	die Diskette ist voll; die abzuspeichernden Daten bzw. Programme sind wahrscheinlich nicht vollständig abgespeichert worden
62	end of file	das Ende des Files ist erreicht, ohne daß dies erwartet wurde
63	bad record number	die verwendete Record-Nr. ist $\leq$ Ø oder $>$ 32767
64	bad filename	der verwendete Filename ist zu lang oder enthält unzulässige Zeichen (z.B. Blank zwischen Diskettenspezifikation: und File-Identifier)

71	volume name not found	eine Diskette mit der angegebenen Diskettenbezeichnung wird nicht gefunden
73	invalid volume number	die bei der Diskettenspezifikation angegebene Stationsbezeichnung ist unzulässig
74	volume sizes don't match	bei einer Operation, die gleiche Diskettentypen erfordert, liegen verschiedene Diskettentypen vor
75	volume not enabled	das anzugebende Disketten-Password ist gar nicht oder falsch angegeben worden. Evtl. wurde versucht, eine Diskette zu kopieren, die niemals mit **vf** formatiert wurde. Diese Diskette kann nicht mehr kopiert werden
76	password not valid	das angegebene File-Password ist gar nicht oder falsch angegeben owrden
77	illegal disk change	während der Verwendung eines Files wurde die betreffende Diskette ausgetauscht
78	write protected file	es wurde versucht, ein File mit File-Schreibschutz zu beschreiben
79	copy protected file	es wurde versucht, ein File zu kopieren, das mit Kopierschutz versehen ist
90	error in parameter	einer oder mehrere der angeführten Parameter enthalten unzulässige Werte

91	too many parameters	zu viele Parameter
92	command not found	unzulässiges Schlüsselwort oder Befehl nicht gefunden (z.B. weil benötigte Systemdiskette nicht einliegt)
96	file not open	File ist nicht offen
99	bad load file	der aufgerufene PCOS-Befehl ist nicht mit dem geladenen Betriebssystem kompatibel (die PCOS-Befehle von Release 1.3 sind nicht kompatibel mit denen von Rel. 2.Ø ff.) oder das aufgerufene File vom falschen Typ
101	time or date	falscher Parameter 'Datum' oder 'Uhrzeit'
106	function key already exists	es wurde versucht, eine Taste mit **pk** zu belegen, deren Original-Belegung bereits geändert wurde
108	call-user error	Fehler beim Aufruf einer Anwenderroutine in Assembler
110	time-out	die angesprochene Einheit ist nicht aktiviert (z.B. der Drucker auf **LOCAL** oder nicht angeschlossen)
111	invalid device	der angegebene Name der Einheit ist unkorrekt (z.B. weil unzulässige Zeichen enthalten sind)

7. ANHANG

1. Kontrollierter Input mit Modul T20500

MODUL: T2Ø5ØØ

FUNKTION: Das Modul ermöglicht eine Eingabe über die Tastatur und weist die Eingabe auf eine Variable zu (ähnlich wie die BASIC-Anweisungen **INPUT** bzw. **LINE INPUT**). Maximale Stellenzahl und Typ der Eingabe können festgelegt werden.
Die Cursorsteuerung wird aktiviert. Eine Vorgabe im Eingabefeld kann übernommen bzw. korrigiert werden. Die verwendete Abschlußtaste wird ermittelt.

EINGANGS-PARAMETER: OH1, OH2, OH3, OH12, OI1, QH4, QH5, OH$
zusätzlich möglich: OH6, OH8 (s. BEMERKUNGEN)

AUSGANGS-PARAMETER: OHØ, OH$, OH bzw. PH bzw. QH sowie (nur bei DES-Version) OT.

AUFRUF: **GOSUB** 2Ø5ØØ

WIRKUNG: Ist das Modul in das BASIC-Programm eingebunden, kann es - nach jeweiliger Spezifizierung der Eingangsparameter - aufgerufen werden, um die Annahme eines kontrollierten Inputs zu ermöglichen. Unter Verwendung der Ausgangsparameter kann der Input im Programm weiterverarbeitet werden.

1. Einbinden des Moduls

Sie befinden sich in der BASIC-Ebene. Im Arbeitsspeicher befindet sich das BASIC-Programm, zu dem das Modul zugeladen werden soll. Legen Sie die Diskette mit dem Modul T2Ø5ØØ (BASIC-File im

ASCII-Format) in Station Ø ein und geben Sie den BASIC-Befehl **MERGE**"Ø:T2Ø5ØØ". Danach ist das Modul zu Ihrem bisherigen Programm im Arbeitsspeicher zugeladen und das gesamte Programm kann mit **SAVE**... auf der gewünschten Diskette abgespeichert werden.

2. Voraussetzungen im BASIC-Programm:

- Die Zeilennrn. 2Ø5ØØ bis 2Ø599 incl. müssen vor dem **MERGE**-Befehl frei sein.
- Am Anfang des BASIC-Programms (Vereinbarungsteil) müssen folgende Anweisungen (je 1 mal) erfolgt sein:

  DEFINT O
  DEFSNG P
  DEFDBL Q
  OHØ = Ø
  und (nur bei DES-Version): OT = Ø

  Diese Anweisungen müssen vor allen **DIM**-Anweisungen stehen.

  Die **DEF**...-Anweisungen legen fest, daß Variablen ohne Typkennzeichen am Ende automatisch von folgendem Typ sind:

Anweisung	Anfangsbuchstabe der Variablen	festgelegter Typ	wie mit Typkennzeichen
DEFINT O	O	Integer	%
DEFSNG P	P	einfache Genauigkeit	!
DEFDBL Q	Q	doppelte Genauigkeit	#
	sonstige	einfache Genauigkeit	!

3. Aufruf des Moduls (Eingangsparameter)

Vor jedem Input-Aufruf mit **GOSUB** 2Ø5ØØ sind folgende Eingangsparameter mit folgender Bedeutung zu setzen:

OH1: Inputart (s. folgende Tabelle)

OH2: Spalte im aktiven Window, in der die Eingabe beginnen soll

OH3: Zeile im aktiven Window, in der die Eingabe beginnen soll

OH12: Typ, von dem das Input-Ergebnis sein soll
- OH12=1 Integer-Input
- OH12=2 Input in einfacher Genauigkeit
- OH12=3 Input in doppelter Genauigkeit
- OH12=4 String-Input

OI1: maximale Anzahl Zeichen des Inputs (bei numerischen Eingaben sind Vorzeichen und - falls zulässig - Dezimalpunkt mitzukalkulieren)

QH4: Minimum bzw. Anzahl Zeichen (nur bei OH1=3 bzw. OH1=4 bzw. OH1=5 nötig)

QH5: Maximum
(nur bei OH1=3 bzw. OH1=4 nötig)

OH$: Wert der Vorgabe im Eingabefeld (bei numerischer Vorgabe muß diese per **STR$**-Funktion in einen String konvertiert werden). Soll das Eingabefeld leer sein (bzw. vor dem Input gelöscht werden), ist OH$="" (Leerstring, nicht " "!) zu setzen.

Tabelle der Inputarten (OH1, QH4 und QH5)

OH1	Art des Inputs	QH4	QH5
1	alphanumerischer Input ohne Prüfung	-	-
2	num. Input ohne genaue Prüfung	-	-
3	ganzzahliger Input mit Intervallprüfung	Minimum	Maximum
4	num. Input mit Kontrolle der Vor- und Nachkommastellen (Anzahl)	Mindestanzahl Vor- und Nachkommastellen	Maximalanzahl Vor- und Nachkommastellen
5	ganzzahliger, positiver Input mit festgelegter Stellenanzahl	Anzahl Stellen	-

Beispiel zu OH1=3:

Alle ganzzahligen Eingaben zwischen Ø und 9999 (incl.) sollen zulässig sein:

OH1=3 : QH4=Ø : QH5=9999 : OI1=5

(Punkt nicht möglich; Vorzeichen kann bei Ausgabe mit Maske nötig werden)

QH4: linke Intervallgrenze

QH5: rechte Intervallgrenze

Beispiel zu OH1=4:

Alle Eingaben mit mindestens 1 Vorkomma- und 2 Nachkommastellen, höchstens aber 4 Vorkomma- und 3 Nachkommastellen sollen akzeptiert werden:

OH1=4 : QH4=Ø1Ø2 : QH5=Ø4Ø3 : OI1=9 (incl. Punkt und Vorzeichen)

Ø1: Mindestanzahl Vorkommastellen

Ø2: Mindestanzahl Nachkommastellen

Ø3: Höchstanzahl Nachkommastellen

Ø4: Höchstanzahl Vorkommastellen

Beispiel zu OH1=5:

Alle Eingaben müssen mit genau 6 Stellen eingegeben werden; nur Ganzzahlen sind zulässig:

OH1=5 : QH4=6 : OI1=7

(Punkt nicht möglich; Vorzeichen kann bei Ausgabe mit Maske nötig werden)

4. Ausgangsparameter:

Sofern der Input zulässig war, steht er immer nach Rückkehr aus der Input-Subroutine in der Variablen OH$ als String zur Verfügung. OH$ ist dabei stets rechts zur Länge von OI1 mit Blanks aufgefüllt.

Darüber hinaus steht er, je nach Eingangsparameter OH12, typgerecht in der Variablen OH, PH oder QH. Es gilt

OH12	Input steht in den Variablen
1	OH$, OH (Integer)
2	OH$, PH (einfach genau)
3	OH$, QH (doppelt genau)
4	OH$

In der Variablen OHØ steht entweder 1, 2 oder 3, je nachdem, welche Abschlußtaste (↵, S1 oder S2) gedrückt wurde.

OHØ	bei Abschluß mit
1	↵
2	S1
3	S2

Bei der DES-Version des M20 (mit zusätzlichen Abschlußtasten S3, S5 und S6) stehen darüber hinaus in der Variablen OT folgende Werte:

OT	bei Abschluß mit
1	↵
2	S1
3	S2
4	S3
5	S5
6	S6

BEMERKUNGEN:

- Der Input darf sich nur über eine Bildschirmzeile erstrecken.
- Statt **H** ist die Cursorsteuerung aktiv, wobei die Cursor-Steuertasten folgendermaßen wirken:

SHIFT + HOME	(Löschen des gesamten Eingabefeldes)
SHIFT + →	(ein Zeichen nach rechts)
SHIFT + ←	(ein Zeichen nach links)
SHIFT + ↓	(das Zeichen an der aktuellen Position löschen)
SHIFT + ↑	(Einfügen an aktueller Position, Aufhebung durch Betätigen einer anderen Cursor-Steuertaste)

- Die Eingabe muß stets mit einer der Abschlußtasten (↵ , S1 oder S2) beendet werden.
- Zur genaueren Steuerung der Inputs empfiehlt es sich, eine laufende Nr. des Inputfeldes zu vergeben. Diese kann als Eingangsparameter OH6 optional gesetzt werden.
- Zeile 2Ø585 ist normalerweise zu desaktivieren (Einfügen eines **REM** vor dem eigentlichen Inhalt der Zeile:

  Vorgehensweise:

  EDIT 2Ø585
  I-Taste drücken
  rem und anschließend Blank eintippen
  CR-Taste drücken)

  Wird dieses **REM** nicht gesetzt bzw. wieder entfernt, gilt folgendes:

Bei Eingabe des Zeichens ? an erster Position des Inputfeldes wird automatisch mit **GOTO** zur Zeile 21ØØ verzweigt, sofern der zusätzliche Eingangsparameter OH8 beim Aufruf des Moduls den Wert Ø hat. Ab Zeile 21ØØ kann die Ausgabe einer Help-Zeile (zur Bedienerführung) für das betreffende Inputfeld programmiert werden (**ON** OH6 **GOTO** ..., ..., ...). Der Rücksprung aus dieser Routine muß per **GOTO** 2Ø5ØØ erfolgen. Nach Annahme des Inputs sollte(n) die vorher generierte(n) Help-Zeile(n) wieder gelöscht werden. Ist die Zeile 2Ø585 aktiv (kein **REM**) und OH8 hat den Wert 1, wird von T2Ø5ØØ nicht zur Zeile 21ØØ verzweigt.

- Weitergehende Prüfungen als durch OH1 ermöglicht müssen nach Annahme des Inputs in OH$ bzw. OH, PH oder QH durchgeführt werden (**IF**-Anweisung (en)).
- Soll die DES-Version eingesetzt werden, ist die richtige Tastatur zu aktivieren (per **sl** 16 in der PCOS-Ebene bzw. **CALL "sl** 16" in der BASIC-Ebene (z.B. im automatischen Startprogramm INIT.BAS)). Das so konfigurierte Betriebssystem muß beim Arbeiten mit T2Ø5ØØ geladen sein, andernfalls sind die Tasten S3, S5 und S6 völlig ohne Wirkung.

BEISPIELE:

```
Ok
new
Ok
rem es wird neu erfaßt
Ok
0 CLEAR
20 DEFINT O:DEFSNG P:DEFDBL Q:OHO=0
30 BR#=9D+99 ' unplausibler Anfangswert --> noch nicht erfaßt
100 GOSUB 1010
110 IF OHO=2 OR OHO=3 THEN 100 'erneut bei S1 oder S2
120 GOSUB 2010
130 IF OK$="n" OR OK$="N" THEN 100
135 'hier:  Abspeichern Brutto auf Disk
150 IF OHO=2 THEN END 'Ende bei S1
160 GOTO 30
1000 REM kontrollierter Input
1010 REM Eingabe: Bruttobetrag (#), maximal 6 VK-Stellen,
         max. 2 NK-Stellen, keine Mindestanzahl Stellen
1020 CURSOR(1,5):PRINT "Brutto  :           :"
1030 OH2=10:OH3=5:OH12=3
1035 OH1=4:QH4=0:QH5=602:OI1=10
1040 OH$="" 'keine Vorgabe
1042 IF BR#()9D+99 THEN OH$=MID$(STR$(BR#),2+(BR#(0))
1043 'nur Vorgabe, wenn br# noch keinen Inhalt; Vorgabe
      linksbündig
1050 GOSUB 20500
1090 BR#=QH
1099 RETURN
2000 REM Eingabe J/N
2010 CURSOR(1,11):PRINT "Ok (j/n): :"
2030 OH2=10:OH3=11:OH12=4
2035 OH1=1:OI1=1
2040 OH$="j"
2050 GOSUB 20500
2060 IF INSTR("JNjn",OH$)=0 THEN 2040
2090 OK$=OH$
2099 RETURN
```

Fortsetzung s. nächstes Blatt

```
merge "0:T20500"
Ok
rem Diskette mit Modul T20500 muß auf Station 0: einliegen
Ok
list 20500
20500 REM T20500 03033
Ok
save "0:Erf.prg"
Ok
rem Abspeichern des Programms
Ok
run
```

II Drucker-Steuerzeichen

STEUERZEICHEN FÜR DEN DRUCKER PR 1450

Code(s), dezimal	Wirkung	Bezeichnung
10	Druck + Zeilenvorschub + auf linken Rand	LF
11	Druck + auf nächsten vert. Tab-Stop	VT
12	Druck + Formularvorschub + auf linken Rand	FF
13	Druck + Wagenrücklauf	CR
27,32,X *)	Festlegung Formularlänge	FL
27,48	Löschen Puffer + Einstellungen (außer DSS)	RIS
27,51	Beginn Fettdruck	DSS
27,52	Aufheben Fettdruck	DSR
27,60	Festlegung 10 Zeichen/Zoll-Druck	HS 10
27,61	Festlegung 12 Zeichen/Zoll-Druck	HS 12
27,62	Festlegung 16,6 Zeichen/Zoll-Druck	HS 16
27,71,m,n **)	Umschalten auf Plotfunktion und Plotten	ESC G
27,73,n,n,n	Druck + Vorschub um nnn Zeilen + auf linken Rand	VPR
27,81,n,n,n ,59,n,n,n ... ,27,90	Festlegung Anzahl Zeilen/Formular; ggf. Festlegung vert. Tab-Stop(s) bezügl. Blattanfang (max. 30)	VTP
27,83,51	Hinweis zur Formulareinführung bei MFF	ESC 53
27,91,Y ***)	Festlegung nationaler Zeichensatz	IGS
28	Umschalten auf alphanumerische Arbeitsweise	FS
127	Löschen Puffer; Einstellungen bleiben erhalten	DEL

*) X = Anzahl Zeilen/Formular + 32 (min. 32, max. 126)

**) m = 48 bewirkt weiß auf schwarz (negativ); m = 49 bewirkt schwarz auf weiß;
n = Zeichenauflösung in Punkten/Zoll (50=71,4 P/Z; 51=62,5; 52=55,5; 53=50)

***) Y: "010" international "020" Deutschland "030" Portugal
"040" Spanien "050" Dänemark/Norwegen "060" Frankreich
"070" Italien "080" Schweden/Finnland "090" Schweiz
"100" Großbritannien "110" US-ASCII

BEISPIELE: Festlegung 80 Zeilen/Formular: 45 LPRINT CHR$(27)+" 112";
Plotten 3 Punktspalten: erste nichts, zweite oberster Punkt, dritte oberste drei Punkte (6 Plotpunkte/Punktspalte sind möglich):
LPRINT CHR$(27)+"G13"+CHR$(2^6)+CHR$(2^0+2^6)+CHR$(2^0+2^1+2^2+2^6)

STEUERZEICHEN FÜR DEN DRUCKER PR 1471

Code(s) dezimal	Wirkung	Bezeichnung
7	akustisches Signal (2 Sek.)	BEL
8	ein Zeichen zurück	BS
9	auf nächsten horiz. Tab-Stop	HT
10	Druck + Zeilenvorschub + auf linken Rand	LF
11	Druck + auf nächsten vert. Tab-Stop	VT
12	Druck + Formularvorschub + auf linken Rand	FF
13	Druck + Wagenrücklauf	CR
27,42,48	Beginn Unterstreichen	USS
27,43	Aufheben Unterstreichen	USR
27,48	Löschen Puffer + Einstellungen; Setzen Default	RIS
27,51	Beginn Fettdruck	DSS
27,52	Aufheben Fettdruck	DSR
27,60	Festlegung 10 Zeichen/Zoll-Druck	HS 10
27,61	Festlegung 12 Zeichen/Zoll-Druck	HS 12
27,62	Festlegung 16,6 Zeichen/Zoll-Druck	HS 16
27,69	Festlegung 6 Zeilen/Zoll-Zeilenschaltung	VS 2I
27,70	Festlegung 8 Zeilen/Zoll-Zeilenschaltung	VS 8L
27,72,n,n,n	Druck + Tabulation auf nnn. Zeichen der Zeile	HPA
27,73,n,n,n	Druck + Vorschub um nnn Zeilen	VPR
27,74,n,n,n	Festlegung linker Rand auf nnn. Zeichen	LMS
27,75,n,n,n	Bewegung um nnn Zeichen nach rechts	HPR
27,76,n,n,n	Druck + Vorschub auf nnn. Zeile	VPA
27,77,n,n,n	Festlegung nnn Leerzeilen am Formularende	BOF
27,80,n,n,n ,59,n,n,n ... ,27,90	Festlegung Anzahl Zeichen/Zeile; ggf. Festlegung hor. Tab-Stop(s) bezügl. linken Rand (max. 16)	HTP
27,81,n,n,n ,59,n,n,n ... ,27,90	Festlegung Anzahl Zeilen/Formular; ggf. Festlegung vert. Tab-Stop(s) bezügl. Blattanfang (max. 30)	VTP
127	Löschen Puffer; alle Einstellungen außer DSS und USS bleiben erhalten	DEL

BEISPIEL: Festlegung 120 Zeichen/Zeile; Tab-Stops auf Positionen 25 und 70 .

```
6011 LPRINT CHR$(27)+CHR$(80)+"120"+";025;070"+CHR$(27)+"Z";
```

STEUERZEICHEN FÜR DEN DRUCKER PR 1481

Code(s), dezimal	Wirkung	Bezeichnung
7	akustisches Signal (2 Sek.)	BEL
8	ein Zeichen zurück	BS
9	auf nächsten horiz. Tab-Stop	HT
10	Druck + Zeilenvorschub + auf linken Rand	LF
11	Druck + auf nächsten vert. Tab-Stop	VT
12	Druck + Formularvorschub + auf linken Rand	FF
13	Druck + Wagenrücklauf	CR
14	höchstes Bit auf 0 (nur 7-Bit-Datenaustausch)	SO
15	höchstes Bit auf 1 (nur 7-Bit-Datenaustausch)	SI
17	Fortsetzung Übertragung in den Puffer nach DC3	DC1
19	Setzen inaktiv (Ready aus); nur DC1 wird erkannt	DC3
20	vertikale Formulartransporte nur über SF1494	DC4
27,33,c	Farbauswahl (bei BCP 1485 bzw. FCP 1484)	COS
27,42,48	Beginn Unterstreichen	USS
27,43	Aufheben Unterstreichen	USR
27,48	Löschen Puffer + Einstellungen; Setzen Default	RIS
27,49,66,n,n,n	positiver Plot-Mode	BIM HTS
27,49,70,n,n,n	negativer Plot-Mode	BIM HTS
27,50	Ende Plot-Mode	HTR
27,51	Beginn Fettdruck	DSS
27,52	Aufheben Fettdruck	DSR
27,55	eine Zeilenschaltung rückwärts	RI
27,60	Festlegung 10 Zeichen/Zoll-Druck	HS 10
27,61	Festlegung 12 Zeichen/Zoll-Druck	HS 12
27,62	Festlegung 16,6 Zeichen/Zoll-Druck	HS 16
27,69	Festlegung 6 Zeilen/Zoll-Zeilenschaltung	VS 2I
27,70	Festlegung 8 Zeilen/Zoll-Zeilenschaltung	VS 8L
27,72,n,n,n	Druck + Tabulation auf nnn. Zeichen der Zeile	HPA
27,73,n,n,n	Druck + Vorschub um nnn Zeilen	VPR
27,74,n,n,n	Festlegung linker Rand auf nnn. Zeichen	LMS
27,75,n,n,n	Bewegung um nnn Zeichen nach rechts	HPR
27,76,n,n,n	Druck + Vorschub auf nnn. Zeile	VPA
27,77,n,n,n	Festlegung nnn Leerzeilen am Formularende	BOF
27,80,n,n,n ,59,n,n,n ... ,27,90	Festlegung Anzahl Zeichen/Zeile; ggf. Festlegung hor. Tab-Stop(s) bezügl. linkem Rand (max. 16)	HTP
27,81,n,n,n ,59,n,n,n ... ,27,90	Festlegung Anzahl Zeilen/Formular; ggf. Festlegung vert. Tab-Stop(s) bezügl. Blattanfang (max. 20)	VTP
27,88,n,n,n	Bewegung um nnn Elementarpunkte vorwärts	HPI
127	Löschen Puffer; alle Einstellungen außer DSS und USS bleiben erhalten	DEL

Erläuterungen:

c: 0 = schwarz; 1 = rot; 2 = blau; 3 = grün

nnn: Länge der Druckzeile in 0,362-mm-Elementarschritten (max. 924)

BEISPIEL: Festlegung 120 Zeichen/Zeile; Tab-Stops auf Positionen 25 und 70 :

```
6011 LPRINT CHR$(27)+CHR$(80)+"120"+";025;070"+CHR$(27)+"Z";
```

STEUERZEICHEN FÜR IFS1497 (einfache manuelle Vorsteckeinrichtung)

27,79	Auswurf der Karte	SHE
27,83,49	Ansteuerung der Druckwalze	SDC
27,83,51	Ansteuerung der IFS1497	SDC
27,85,49	akust. Signal + Aufleuchten CARD-REQ-Lampe	OCA

STEUERZEICHEN FÜR IFA1496 (automatische Fronteinzugsvorrichtung)

27,64	Druck Markierung	
27,78	Einziehen Karte bis Markierung bzw. Zeile 001	SHL
27,79	Auswurf der Karte	SHE
27,83,49	Ansteuerung der Druckwalze	SDC
27,83,53	Ansteuerung der IFA1496 + LF, RI u. RIS	SDC
27,84,n,n,n	Einziehen Karte bis Zeile nnn	LPP
27,85,u	akust. Signal + Aufleuchten CARD-REQ-Lampe	OCA

Erläuterungen:
u: 1 = Benutzer 1; 2 = Benutzer 2 (nur bei Konsole für gemeinsame Benutzung)

STEUERZEICHEN FÜR DEN DRUCKER PR 2400

Code(s), dezimal	Wirkung	Bezeichnung
10	Druck + Zeilenschaltung um 10 Elementarpunkte	LF
11	wie LF	
12	wie LF	
13	Druck + Erzeugen von 3 Elementarpunktzeilen	CR
27,48	Löschen Puffer + Einstellungen	RIS
27,71	Umschalten auf Plotfunktion für eine Punktzeile	PS
27,74,n,n,n	Festlegung linker Rand im Plotstatus	LMS
27,80,n,n,n	Festlegung Zeilenlänge in 7-Bit-Bytes	LL
27,167	Konversion alphanumerisch nach graphisch	
127	Löschen Puffer; Einstellungen bleiben erhalten	DEL

BEISPIELE: Vorbereitung Plotten:

```
LPRINT CHR$(27)+"J000"+CHR$(27)+"P070"+CHR$(27)+"G"
```

Die zu sendenden Bytes sind in Gruppen zu je 7 Bit zu organisieren, wobei das Bit mit der höchsten Signifikanz jeweils den äußersten linken Punkt repräsentiert.

STEUERZEICHEN FÜR DEN DRUCKER PR 2300

Code(s) dezimal	Wirkung	Bezeichnung
7	akustisches Signal (300 ms)	BEL
9	auf nächsten horiz. Tab-Stop	HT
10	Druck + Zeilenvorschub + auf linken Rand	LF
11	Druck + auf nächsten vert. Tab-Stop	VT
12	Druck + Formularvorschub + auf linken Rand	FF
13	Druck + Zeilenvorschub	CR
27,37	Aufheben Druck in doppelter Höhe	
27,38,p	Einstellen Breite Zeilenschaltungen	
27,39	Beginn Druck in doppelter Höhe	
27,42,t	Beginn Unterstreichen	USS
27,43	Aufheben Unterstreichen	USR
27,45	Negativdarstellung des Bilds, das in der Folge mit PLM übertragen wird	
27,47	Zoom des in der Folge mit PLM übertragenen Bilds (vert. + hor. Verdoppelung aller Punkte)	
27,48	Löschen Puffer + Einstellungen; Setzen Default	RIS
27,51	Beginn Fettdruck	DSS
27,52	Aufheben Fettdruck	DSR
27,60	Festlegung 10 Zeichen/Zoll-Druck	HS 10
27,61	Festlegung 12,2 Zeichen/Zoll-Druck	HS 12
27,62	Festlegung 18,3 Zeichen/Zoll-Druck	HS 16
27,69	Festlegung 6 Zeilen/Zoll-Zeilenschaltung	VS 2I
27,70	Festlegung 8 Zeilen/Zoll-Zeilenschaltung	VS 8L
27,71,i,i,i,59,j,j,j,59,k,k,k,59,r,27,90	Ansteuerung Plot-Mode	PLM
27,73,n,n,n	Druck + Vorschub um nnn Zeilen	VPR
27,80,n,n,n ,59,n,n,n ... ,27,90	Festlegung Anzahl Zeichen/Zeile; ggf. Festlegung hor. Tab-Stop(s) bezügl. linkem Rand (max. 8)	HTP
27,81,n,n,n ,59,n,n,n ... ,27,90	Festlegung Anzahl Zeilen/Formular; ggf. Festlegung vert. Tab-Stop(s) bezügl. Blattanfang (max. 30)	VTP
27,91,z,z,z	Festlegung nationaler Zeichensatz	IGS
127	Löschen Puffer; alle Einstellungen außer DSS und USS bleiben erhalten	DEL

Erläuterungen:

p: 1 bis 9 ; es handelt sich um Elementarpunktzeilen

t: 0 = normales Unterstreichen; 1 = doppelt; 2 = gesperrt

iii: Beginn des Drucks in Elementarpunkten vom linken Rand (001 - 255)

jjj: horizontale Abmessung der Abbildung (Bitmap) in Bytes (001 - 110)

kkk: vertikale Abmessung der Abbildung (Bitmap) in Bytes (001 - 255)

r: vertikale Auflösung (Mindestabstand zwischen zwei Punkten bei Punktbreite von 0,1175 mm); Default: 0,235 mm = 108 Punkte/Zoll (1 - 9)

zzz: 010 = international 020 = Deutschland 030 = Portugal
040 = Spanien 050 = Dänemark/Norwegen 060 = Frankreich
070 = Italien 080 = Schweden/Finnland 090 = Schweiz
100 = Großbritannien 110 = US-ASCII

BEISPIEL: Festlegung 120 Zeichen/Zeile; Tab-Stops auf Positionen 25 und 70 :

```
6011 LPRINT CHR$(27)+CHR$(80)+"120"+",025,070"+CHR$(27)+"Z";
```

STEUERZEICHEN FÜR DEN DRUCKER PR 430

Code(s) dezimal	Wirkung	Bezeichnung
7	akustisches Signal (1 Sek.)	BEL
8	ein Zeichen zurück, wenn druckbares Zeichen vorausgeht, sonst: Druckwiederholung	BS
9	auf nächsten horiz. Tab-Stop	HT
10	Druck + Zeilenvorschub + auf linken Rand	LF
11	Druck + auf nächsten vert. Tab-Stop	VT
12	Druck + Formularvorschub + auf linken Rand	FF
13	Druck + Wagenrücklauf	CR
20	vert. Formulartransport nur über SF436	DC4
27,40	Beginn Fettdruck ein Zeichen	ESC (
27,41	Aufheben Fettdruck ein Zeichen	ESC)
27,42,48	Beginn Unterstreichen	USS
27,43	Aufheben Unterstreichen	USR
27,48	Löschen Puffer + Einstellungen; Setzen Default	RIS
27,56	Zeilenschaltung um 1 Elementarzeile rückwärts	ESC 8
27,57	Zeilenschaltung um 1 Elementarzeile vorwärts	ESC 9
27,65	Zeilenschaltung um 5 Elementarzeilen	5I
27,66	Zeilenschaltung um 4 Elementarzeilen	4I
27,67	Zeilenschaltung um 3 Elementarzeilen	3I
27,68	Zeilenschaltung um 6 Elementarzeilen	6I
27,69	Zeilenschaltung um 2 Elementarzeilen	2I
27,72,n,n	Druck + Tabulation auf Position nn	HPA
27,73,n,n	Druck + Vorschub um nn Zeilen	VPR
27,74,n,n,n	Festlegung linker Rand auf nnn. Zeichen	LMS
27,75,n,n,n	ohne Wirkung	
27,76,n,n,n	ohne Wirkung	
27,77,n,n,n	Festlegung nnn Leerzeilen am Formularende	BOF
27,78	bewirkt Formulareinzug (nur bei ASF 435)	SF
27,79	Einzelblatt-Transport von Walze nach Ablagefach (nur bei ASF 435)	SR
27,80,n,n,n[,59,n,n,n ... ,27,90]	Festlegung Anzahl Zeichen/Zeile; ggf. Festlegung hor. Tab-Stop(s) bezügl. linkem Rand (max. 30)	HTP
27,81,n,n,n[,59,n,n,n ... ,27,90]	Festlegung Anzahl Zeilen/Formular; ggf. Festlegung vert. Tab-Stop(s) bezügl. Blattanfang (max. 30)	VTP
27,83,49	Festlegung der Walze als Papierführungseinheit	
27,83,65	Festlegung automatische Einzelblattzuführung (nur bei ASF 435)	
27,88,n,n,n	Bewegung um nnn Basisschritte (=0,21 mm) vorwärts	
127	Löschen Puffer; alle Einstellungen außer DSS und USS bleiben erhalten	DEL

BEISPIEL: Festlegung 120 Zeichen/Zeile; Tab-Stops auf Positionen 25 und 70 :

```
6011 LPRINT CHR$(27)+CHR$(80)+"120"+";025;070"+CHR$(27)+"Z";
```

III BINAER - HEXADEZIMAL - DEZIMAL - Umrechnungstabelle

HEX	BIN	DEZ	DEZ*256	HEX	BIN	DEZ	DEZ*256	HEX	BIN	DEZ	DEZ*256	HEX	BIN	DEZ	DEZ*256
0	00000000	0	0	40	01000000	64	16640	80	10000000	128	32768	C0	11000000	192	49152
1	00000001	1	256	41	01000001	65	16896	81	10000001	129	33024	C1	11000001	193	49408
2	00000010	2	512	42	01000010	66	17152	82	10000010	130	33280	C2	11000010	194	49664
3	00000011	3	768	43	01000011	67	17408	83	10000011	131	33536	C3	11000011	195	49920
4	00000100	4	1024	44	01000100	68	17664	84	10000100	132	33792	C4	11000100	196	50176
5	00000101	5	1280	45	01000101	69	17920	85	10000101	133	34048	C5	11000101	197	50432
6	00000110	6	1536	46	01000110	70	18176	86	10000110	134	34304	C6	11000110	198	50688
7	00000111	7	1792	47	01000111	71	18176	87	10000111	135	34560	C7	11000111	199	50944
8	00001000	8	2048	48	01001000	72	18432	88	10001000	136	34816	C8	11001000	200	51200
9	00001001	9	2304	49	01001001	73	18688	89	10001001	137	35072	C9	11001001	201	51456
A	00001010	10	2560	4A	01001010	74	18944	8A	10001010	138	35328	CA	11001010	202	51712
B	00001011	11	2816	4B	01001011	75	19200	8B	10001011	139	35584	CB	11001011	203	51968
C	00001100	12	3072	4C	01001100	76	19456	8C	10001100	140	35840	CC	11001100	204	52224
D	00001101	13	3328	4D	01001101	77	19712	8D	10001101	141	36096	CD	11001101	205	52480
E	00001110	14	3584	4E	01001110	78	19968	8E	10001110	142	36352	CE	11001110	206	52736
F	00001111	15	3840	4F	01001111	79	20224	8F	10001111	143	36608	CF	11001111	207	52992
10	00010000	16	4096	50	01010000	80	20480	90	10010000	144	36864	D0	11010000	208	53248
11	00010001	17	4352	51	01010001	81	20736	91	10010001	145	37120	D1	11010001	209	53504
12	00010010	18	4608	52	01010010	82	20992	92	10010010	146	37376	D2	11010010	210	53760
13	00010011	19	4864	53	01010011	83	21248	93	10010011	147	37632	D3	11010011	211	54016
14	00010100	20	5120	54	01010100	84	21504	94	10010100	148	37888	D4	11010100	212	54272
15	00010101	21	5376	55	01010101	85	21760	95	10010101	149	38144	D5	11010101	213	54528
16	00010110	22	5632	56	01010110	86	22016	96	10010110	150	38400	D6	11010110	214	54784
17	00010111	23	5888	57	01010111	87	22272	97	10010111	151	38656	D7	11010111	215	55040
18	00011000	24	6144	58	01011000	88	22528	98	10011000	152	38912	D8	11011000	216	55296
19	00011001	25	6400	59	01011001	89	22784	99	10011001	153	39168	D9	11011001	217	55552
1A	00011010	26	6656	5A	01011010	90	23040	9A	10011010	154	39424	DA	11011010	218	55808
1B	00011011	27	6912	5B	01011011	91	23296	9B	10011011	155	39680	DB	11011011	219	56064
1C	00011100	28	7168	5C	01011100	92	23552	9C	10011100	156	39936	DC	11011100	220	56320
1D	00011101	29	7424	5D	01011101	93	23808	9D	10011101	157	40192	DD	11011101	221	56576
1E	00011110	30	7680	5E	01011110	94	24064	9E	10011110	158	40448	DE	11011110	222	56832
1F	00011111	31	7936	5F	01011111	95	24320	9F	10011111	159	40704	DF	11011111	223	57088
20	00100000	32	8192	60	01100000	96	24576	A0	10100000	160	40960	E0	11100000	224	57344
21	00100001	33	8448	61	01100001	97	24832	A1	10100001	161	41216	E1	11100001	225	57600
22	00100010	34	8704	62	01100010	98	25088	A2	10100010	162	41472	E2	11100010	226	57856
23	00100011	35	8960	63	01100011	99	25344	A3	10100011	163	41728	E3	11100011	227	58112
24	00100100	36	9216	64	01100100	100	25600	A4	10100100	164	41984	E4	11100100	228	58368
25	00100101	37	9472	65	01100101	101	25856	A5	10100101	165	42240	E5	11100101	229	58624
26	00100110	38	9728	66	01100110	102	26112	A6	10100110	166	42496	E6	11100110	230	58880
27	00100111	39	9984	67	01100111	103	26368	A7	10100111	167	42752	E7	11100111	231	59136
28	00101000	40	10240	68	01101000	104	26624	A8	10101000	168	43008	E8	11101000	232	59392
29	00101001	41	10496	69	01101001	105	26880	A9	10101001	169	43264	E9	11101001	233	59648
2A	00101010	42	10752	6A	01101010	106	27136	AA	10101010	170	43520	EA	11101010	234	59904
2B	00101011	43	11008	6B	01101011	107	27392	AB	10101011	171	43776	EB	11101011	235	60160
2C	00101100	44	11264	6C	01101100	108	27648	AC	10101100	172	44032	EC	11101100	236	60416
2D	00101101	45	11520	6D	01101101	109	27904	AD	10101101	173	44288	ED	11101101	237	60672
2E	00101110	46	11776	6E	01101110	110	28160	AE	10101110	174	44544	EE	11101110	238	60928
2F	00101111	47	12032	6F	01101111	111	28416	AF	10101111	175	44800	EF	11101111	239	61184
30	00110000	48	12288	70	01110000	112	28672	B0	10110000	176	45056	F0	11110000	240	61440
31	00110001	49	12544	71	01110001	113	28928	B1	10110001	177	45312	F1	11110001	241	61696
32	00110010	50	12800	72	01110010	114	29184	B2	10110010	178	45568	F2	11110010	242	61952
33	00110011	51	13056	73	01110011	115	29440	B3	10110011	179	45824	F3	11110011	243	62208
34	00110100	52	13312	74	01110100	116	29696	B4	10110100	180	46080	F4	11110100	244	62464
35	00110101	53	13568	75	01110101	117	29952	B5	10110101	181	46336	F5	11110101	245	62720
36	00110110	54	13824	76	01110110	118	30208	B6	10110110	182	46592	F6	11110110	246	62976
37	00110111	55	14080	77	01110111	119	30464	B7	10110111	183	46848	F7	11110111	247	63232
38	00111000	56	14336	78	01111000	120	30720	B8	10111000	184	47104	F8	11111000	248	63488
39	00111001	57	14592	79	01111001	121	30976	B9	10111001	185	47360	F9	11111001	249	63744
3A	00111010	58	14848	7A	01111010	122	31232	BA	10111010	186	47616	FA	11111010	250	64000
3B	00111011	59	15104	7B	01111011	123	31488	BB	10111011	187	47872	FB	11111011	251	64256
3C	00111100	60	15360	7C	01111100	124	31744	BC	10111100	188	48128	FC	11111100	252	64512
3D	00111101	61	15616	7D	01111101	125	32000	BD	10111101	189	48384	FD	11111101	253	64768
3E	00111110	62	15872	7E	01111110	126	32256	BE	10111110	190	48640	FE	11111110	254	65024
3F	00111111	63	16128	7F	01111111	127	32512	BF	10111111	191	48896	FF	11111111	255	65280

8. ALPHABETHISCHES VERZEICHNIS DER BEFEHLE, ANWEISUNGEN UND FUNKTIONEN

NAME	ART	ZWECK/FUNKTION
ABS	FUNKTION	Liefert den Absolutwert (Betrag) eines numerischen Werts
ASC	FUNKTION	Liefert den ISO-Code eines Zeichens
ATN	FUNKTION	Liefert von einem numerischen Wert seinen Arcustangens (Winkel im Bogenmaß)
AUTO	BEFEHL	Automatische Erzeugung von Zeilennummern für einzugebende Anweisungen eines Programms
CALL	ANWEISUNG	Aufruf und Abarbeitung eines PCOS-Befehls oder einer selbstdefinierten Assembler-Routine
CDBL	FUNKTION	Konvertiert einen numerischen Wert in doppelte Genauigkeit
CHAIN	ANWEISUNG	Aufruf eines Programms, eventuell mit Übergabe von Variablen vom gegenwärtigen Programm an das aufgerufene sowie ggf. Laden von Overlays und/oder Löschen von Programmzeilen
CHR$	FUNKTION	Liefert dasjenige Zeichen, das durch einen ISO-Code bestimmt ist
CINT	FUNKTION	Konvertiert einen numerischen Wert in einen ganzzahligen (Integer-)Wert
CIRCLE	FUNKTION	Erlaubt das Zeichnen von Kreisen oder Ellipsen

CLEAR	ANWEISUNG	Löschen des Datenbereichs und der Bitmap für den Bildschirm im Arbeitsspeicher; setzen des gesamten Bildschirms auf Default-Hintergrundfarbe; Aufhebung aller Window-Vereinbarungen; schließen aller Fälle; ggf. Definition der logischen Größe des Arbeitsspeichers und/ oder des für Stack-Speicher zu reservierenden Platzes im Arbeitsspeicher
CLOSE	ANWEISUNG	Schließen von einem, mehreren oder allen externen Datenfiles
CLOSE WINDOW	ANWEISUNG	Auflösen von Windows
CLS	ANWEISUNG	Löscht den Inhalt des aktiven oder des angegebenen Windows
COLOR	ANWEISUNG	Auswahl oder Neufestlegung von Vorder- und Hintergrundfarbe
COLOR=	ANWEISUNG	Auswahl von vier aus acht Farben für das Arbeiten mit dem Vier-Farbschirm
COMMON	ANWEISUNG	Definition der Variablen in einem BASIC-Programm, deren Werte bei der Verkettung von Programmen erhalten bleiben
CONT	BEFEHL	Fortsetzung der Programmausführung nach einer Unterbrechung
COS	FUNKTION	Liefert den Cosinus eines im Bogenmaß angegebenen Winkels
CSNG	FUNKTION	Konvertiert einen numerischen Wert in einfache Genauigkeit

CURSOR	ANWEISUNG	Steuerung des Text- oder des Graphik-Cursors und ggf. Festlegung der Cursor-Eigenschaften
CVD	FUNKTION	Verwandlung der ersten 8 Bytes eines Strings in einen doppelt genauen numerischen Wert
CVI	FUNKTION	Verwandlung der ersten 2 Bytes eines Strings in einen Integer-Wert
CVS	FUNKTION	Verwandlung der ersten 4 Bytes eines Strings in einen einfach genauen numerischen Wert
DATA	ANWEISUNG	Erzeugung eines internen Files von Daten, die anschließend mit **READ**-Anweisungen auf Variable(n) zugewiesen werden
DATE$	SPEZIELLE VARIABLE	Von der Echtzeituhr verwaltetes Tagesdatum
DEFDBL	ANWEISUNG	Typvereinbarung aller Variablen mit bestimmten Anfangsbuchstaben und ohne Typkennzeichen als doppelt genaue numerische Variable
DEF FN	ANWEISUNG	Definition einer numerischen oder einer Stringfunktion innerhalb einer BASIC-Zeile durch den Anwender
DEFINT	ANWEISUNG	Typvereinbarung aller Variablen mit bestimmten Anfangsbuchstaben und ohne Typkennzeichen als numerische Integer-Variablen

DEFSNG	ANWEISUNG	Typvereinbarung aller Variablen mit bestimmten Anfangsbuchstaben und ohne Typkennzeichen als einfach genaue numerische Variablen
DEFSTR	ANWEISUNG	Typvereinbarung aller Variablen mit bestimmten Anfangsbuchstaben und ohne Typkennzeichen als String-Variablen
DELETE	BEFEHL	Löschung einer oder mehrerer Programmzeilen im Arbeitsspeicher
DIM	ANWEISUNG	Festlegung der Dimension und der maximalen Werte der Indices von Arrays
DRAW	ANWEISUNG	Festlegung von Zeichenvorgängen
EDIT	BEFEHL	Ändern bestehender Programmzeilen eines Programms im Arbeitsspeicher
END	ANWEISUNG	Festlegung eines logischen Programmendes
EOF	FUNKTION	Abfrage auf Ende eines sequentiellen Files beim Input von sequentiellen Files
ERASE	ANWEISUNG	Aufheben der Dimensionierung eines Arrays
ERL	RESERVIERTE VARIABLE	Enthält die Zeilennr. derjenigen Programmzeile, in der der zuletzt entdeckte BASIC-Fehler auftrat bzw. die Zeilennr., in der die letzte **ERROR**-Anweisung auftrat

ERR	RESERVIERTE VARIABLE	Enthält den Fehler-Code des zuletzt entdeckten BASIC-Fehlers bzw. nach einer **ERROR**-Anweisung den dadurch zugewiesenen Wert
ERROR	ANWEISUNG	Zuweisung eines Fehler-Codes auf die reservierte Variable **ERR** (Simulation eines BASIC-Fehlers durch den Anwender)
EXEC	ANWEISUNG	Aufruf und Abarbeitung eines PCOS-Befehls oder einer selbstdefinierten Assembler-Routine
EXP	FUNKTION	Berechnung einer Potenz der Eulerschen Zahl e
FIELD	ANWEISUNG	Vereinbarung der Feldlängen der Records eines bestimmten Random-Files, Festlegung der Namen der Feldvariablen, ihrer Position und Länge in Bytes im Random-File-Puffer sowie Bereitstellung des Random-File-Puffers für folgende **GET**- oder **PUT**-Anweisungen
FILES	BEFEHL	Ausgabe des Inhaltsverzeichnisses einer Diskette oder der Angaben über ein einzelnes File
FIX	FUNKTION	Ermittlung des ganzzahligen Teils eines numerischen Werts
FN	FUNKTION	Eine vom Anwender definierte Funktion wird aufgerufen und abgearbeitet
FOR	ANWEISUNG	Startanweisung für die mehrmalige Ausführung von Anweisungen innerhalb einer zählergesteuerten Schleife

FRE	FUNKTION	Ermittlung des im Arbeitsspeicher noch freien Platzes (in Bytes) und ggf. Bereinigung des Arbeitsspeichers ("garbage collection")
GET	**ANWEISUNG**	Übertragen eines logischen Records von einem Random-File in den Random-File-Puffer ("Lesen")
GET%	ANWEISUNG	Speichern eines Bereichs eines Windows in einem Array
GOSUB	ANWEISUNG	Sprung zu einer bestimmten Anweisung, bei der ein Unterprogramm beginnt
GOTO	ANWEISUNG	Sprung zu einer bestimmten Zeile des Programms
HEX$	FUNKTION	Liefert den hexadezimalen Wert einer dezimal dargestellten Zahl; das Ergebnis steht in Form eines Strings zur Verfügung
IF...THEN ...ELSE	ANWEISUNG	Bedingte Verzweigung(en) in einem Programm
INKEY$	FUNKTION	Überprüfung, ob ein Zeichen im Tastaturpuffer ist und wenn, welches das erste ist
INPUT	ANWEISUNG	Anforderung einer Eingabe über die Tastatur und Zuweisung der Eingabe(n) auf Variable(n)
INPUT#	ANWEISUNG	Lesen von Daten von einem sequentiellen File oder aus einem Random-File-Puffer und Zuweisung auf Variable(n)

INPUT$	FUNKTION	Liefert eine bestimmte aus dem Tastaturpuffer oder aus einem File einzulesende Anzahl Zeichen
INSTR	FUNKTION	Aufsuchen der Anfangsposition eines gesuchten Strings in einem String
INT	FUNKTION	Ermittlung derjenigen ganzen Zahl aus einem numerischen Wert, die als nächste auf der Zahlenskala links von dem angegebenen num. Wert liegt ("Gauß-Funktion")
KILL	BEFEHL	Löschen eines auf Diskette gespeicherten Files aus dem Inhaltsverzeichnis
LEFT$	FUNKTION	Einem String wird von links kommend ein Teilstring entnommen
LEN	FUNKTION	Bestimmung der Länge eines String-Ausdrucks
LET...=	ANWEISUNG	Zuweisung des Werts eines Ausdrucks an eine Variable
LINE	ANWEISUNG	Zeichnen von Linien oder Rechtecken oder mit einer Farbe gefüllten Rechtecken
LINE INPUT	ANWEISUNG	Anforderung der Eingabe einer Kette von Zeichen ohne Trennzeichen über Tastatur und Zuweisung auf eine Stringvariable
LINE INPUT#	ANWEISUNG	Lesen einer Folge von Zeichen von einem sequentiellen File oder aus einem Random-File-Puffer und Zuweisung auf eine Stringvariable

LIST	BEFEHL	Ausgabe einer oder mehrerer Zeilen eines im Arbeitsspeicher befindlichen Programms am Bildschirm
LLIST	BEFEHL	Ausgabe einer oder mehrerer Zielen eines im Arbeitsspeicher befindlichen Programms auf dem Drucker
LOAD	BEFEHL	Laden eines Programms von Diskette in den Arbeitsspeicher
LOC	FUNKTION	Gibt die Anzahl gelesener bzw. geschriebener Sektoren bei sequentiellen Files oder die letzte angesprochene Record-Nr. bei Random-Files an
LOF	FUNKTION	Liefert für sequentielle Files die aktuelle Pointer-Position; bei Random-Files kann damit ermittelt werden, in welchem Sektor des Files der höchste beschriebene Record steht
LOG	FUNKTION	Liefert den natürlichen Logarithmus zur Basis e eines numerischen Werts
LPOS	FUNKTION	Liefert die aktuelle Position des Druckpointers innerhalb des Druckpuffers
LPRINT	ANWEISUNG	Ausgabe von Zahlen und/oder Strings auf dem Drucker
LPRINT USING	ANWEISUNG	Ausgabe von formatierten Daten (numerisch und/oder String) auf dem Drucker

LSET	ANWEISUNG	Zuweisung eines linksbündigen, rechts eventuell mit Blanks aufgefüllten Strings auf die Feldvariable eines Random-File-Puffers (zur Vorbereitung eines **PUT**-Anweisung) oder auf eine Stringvariable
MERGE	BEFEHL	Kombination eines sich im Arbeitsspeicher befindlichen Programms mit einem File, das im ASCII-Format (Parameter **A** bei **SAVE**) abgespeichert ist
MID$...=	ANWEISUNG	Ersetzen eines String-Bestandteils durch einen anderen String
MID$	FUNKTION	Einem String wird ab einer bestimmten Position ein Teilstring entnommen
MKD$	FUNKTION	Konvertierung eines doppelt genauen numerischen Werts in einen 8 Zeichen (Bytes) langen String
MKI$	FUNKTION	Konvertierung eines Inter-Werts in einen 2 Zeichen (Bytes) langen String
MKS$	FUNKTIONµ	Konvertierung eines einfach genauen numerischen Werts in einen 4 Zeichen (Bytes) langen String
NAME	BEFEHL	Änderung des Namens eines Programms oder Datenfiles auf Diskette
NEW	BEFEHL	Löschen des Arbeitsspeicher-Inhalts; Vorbereitung zur Eingabe eines BASIC-Programms oder mit Zeilen-Nr. versehenen Textes

NEXT	ANWEISUNG	Endanweisung einer oder mehrerer **FOR-NEXT** Schleifen
NULL	ANWEISUNG	Verzögert die Ausführung von Anweisungen, die Ausgabeeinheiten betreffen
OCT$	FUNKTION	Liefert den oktalen Wert einer dezimal dargestellten Zahl; das Ergebnis steht in Form eines Strings zur Verfügung
ON ERROR GOTO	ANWEISUNG	Verzweigung zu einer Fehlerbehandlungsroutine bei Auftreten eines BASIC-Fehlers
ON... GOSUB	ANWEISUNG	Sprung in ein Unterprogramm; das Sprungziel ist abhängig vom Wert eines num. Ausdrucks
ON...GOTO	ANWEISUNG	Sprung in eine bestimmte Programmzeile; das Sprungziel ist abhängig vom Wert eines num. Ausdrucks
OPEN	ANWEISUNG	Öffnen eines sequentiellen oder Random-Files für Zugriffe; Festlegung der Länge des File-Puffers; evtl. auch Neuanlage eines nicht vorhandenen Files bzw. Vorbereitung zur automatischen Verlängerung eines vorhandenen Files
OPTION BASE	ANWEISUNG	Festlegung des kleinsten möglichen Index-Wertes aller Arrays
PAINT	ANWEISUNG	Ausfüllen eines ganzen oder eines Teils eines Windows
POINT	FUNKTION	Ermittlung der Farbe in einem Koordinatenpunkt

POS	FUNKTION	Liefert die aktuelle Spalten- oder Zeilen-Position des Bildschirm-Cursors
PRESET	ANWEISUNG	Darstellung eines Elementarpunkts in der Hintergrundfarbe
PRINT	ANWEISUNG	Ausgabe von Zahlen und/oder Strings im Standardformat auf dem Bildschirm
PRINT#	ANWEISUNG	Schreiben von Daten auf ein externes sequentielles File; Trennzeichen werden nicht automatisch auf das File geschrieben; am Ende wird ggf. ein CR und ein LF auf das File geschrieben
PRINT USING	ANWEISUNG	Ausgabe von formatierten Daten (numerisch und/oder String) auf dem Bildschirm
PRINT# ...USING	ANWEISUNG	Schreiben von formatierten Daten (numerisch und/oder String) auf ein sequentielles File
PSET	ANWEISUNG	Darstellung eines Elementarpunktes in vorgewählter Farbe
PUT	ANWEISUNG	Übertragen eines logischen Records von einem Random-File-Puffer in das zugehörige Random-File ("Schreiben")
PUT %	ANWEISUNG	Ein in einem Array gespeichertes Bild wird auf dem Bildschirm dargestellt
RANDOMIZE	ANWEISUNG	Festlegung bzw. Auswahl des Anfangswertes und der Folge von später aufzurufenden Zufallszahlen

READ	ANWEISUNG	Zuweisung von Werten aus dem internen Datenfile an Variablen; die Werte wurden vorher in einer internen Tabelle mit Hilfe von **DATA**-Anweisungen generiert
REM	ANWEISUNG	Einfügung von Erläuterungen in ein Programm
RENUM	BEFEHL	Umnumerierung der Zeilen eines im Arbeitsspeicher vorhandenen Programms
RESTORE	ANWEISUNG	Setzen des Pointers des internen Datenfiles auf das erste Element einer **DATA**-Anweisung
RESUME	ANWEISUNG	Fortsetzung eines Programms nach einer Verzweigung in eine Fehlerbehandlungsroutine
RETURN	ANWEISUNG	Logisches Ende eines Unterprogramms; Rücksprung zu der unmittelbar auf das aufrufende **GOSUB** oder **ON...GOSUB** folgenden Anweisung
RIGHT$	FUNKTION	Einem String wird von rechts kommend ein Teilstring entnommen
RND	FUNKTION	Liefert eine Zufallszahl zwischen Ø und 1
RSET	ANWEISUNG	Zuweisung eines rechtsbündigen, links eventuell mit Blanks aufgefüllten Strings auf die Feldvariable eines Random-File-Puffers (zur Vorbereitung einer **PUT**-Anweisung) oder auf eine Stringvariable
RUN	BEFEHL	Start der Programmausführung

SAVE	BEFEHL	Abspeicherung eines BASIC-Programms oder eines mit Zeilennummern versehenen Textes (ASCII-Files) auf Diskette
SCALE	ANWEISUNG	Festlegung des Koordinatensystems innerhalb eines Windows
SCALEX	FUNKTION	Gibt an, der wievielte Elementarpunkt des aktiven Windows durch eine Anwenderbezogene X-Koordinate angesprochen wird
SCALEY	FUNKTION	Gibt an, der wievielte Elementarpunkt des aktiven Windows durch eine Anwenderbezogene Y-Koordinate angesprochen wird
SGN	FUNKTION	gibt Aufschluß über das Vorzeichen eines numerischen Werts ("Signum-Funktion")
SIN	FUNKTION	Liefert den Sinus eines im Bogenmaß angegebenen Winkels
SPACE$	FUNKTION	Liefert einen String aus Blanks
SPC	FUNKTION	Ausgabe einer bestimmten Anzahl von Leerzeichen (Blanks) auf dem Bildschirm oder auf dem Drucker
SQR	FUNKTION	Liefert die positive Quadratwurzel eines numerischen Werts
STOP	ANWEISUNG	Unterbrechen der Programmausführung, Übergang in den Direkt-Mode
STR$	FUNKTION	Bildung eines Strings aus einem numerischen Ausdruck
STRING$	FUNKTION	Erzeugung eines Strings aus vorgegebenem Zeichen mit vorgegebener Länge

SWAP	ANWEISUNG	Vertauschen des Inhalts zweier Variablen vom gleichen Typ
SYSTEM	BEFEHL	Aufruf der PCOS-Ebene
TAB	FUNKTION	Positionieren auf eine gewünschte Spalte am Bildschirm oder Drucker
TAN	FUNKTION	Liefert den Tangens eines im Bogenmaß angegebenen Winkels
TIME$	SPEZIELLE VARIABLE	Von der Echtzeituhr verwaltete Uhrzeit
TROFF	BEFEHL	Aufhebung der Wirkung von **TRON**
TRON	BEFEHL	Ausdruck der Zeilennummern während des Programmablaufs
USING	ANWEISUNGS-BESTANDTEIL	Bestimmung des Formats, in welchem auszugebende Größen (Zahlen und/oder Strings) dargestellt werden
VAL	FUNKTION	Ermittlung des numerischen Gehalts eines Strings
VARPTR	FUNKTION	Liefert die Byte-Adresse einer Variablen im Arbeitsspeicher oder die Start-Adresse des I/O-Puffers, der der Filenr. eines sequentiellen Files zugeordnet ist bzw. die Adresse des Random-File-Puffers für ein unter der Filenr. geöffnetes Random-File
WEND	ANWEISUNG	End-Anweisung einer **WHILE-WEND**-Schleife

WHILE	ANWEISUNG	Start-Anweisung einer Schleife, die solange durchgeführt wird, wie eine Bedingung erfüllt ist
WIDTH	ANWEISUNG	Festlegung der Anzahl Zeichen pro Ausgabezeile für Drucker oder Bildschirm
WINDOW	FUNKTION	Definition der Lage und Größe eines neuen Windows ("Anlegen" bzw. "Eröffnen") und/oder Neu-Festlegung der Anzahl Zeichen pro Zeile und/oder der Anzahl Zeilen in einem Window; ggf. Ermittlung der Window-Nr. des aktiven Windows
WINDOW%	ANWEISUNG	Anwählen ("Aktivieren) eines eröffneten Windows
WRITE	ANWEISUNG	Ausgabe von Daten am Bildschirm; als Trennzeichen werden automatisch Kommas ausgegeben; Strings werden automatisch in Anführungszeichen ausgegeben
WRITE#	ANWEISUNG	Schreiben von Daten auf ein externes sequentielles File; als Trennzeichen wird automatisch ein Komma auf das File geschrieben; Strings werden automatisch in Anführungszeichen eingeschlossen auf das File gebracht; am Ende wird ein CR und ein LF geschrieben