# EUMEL

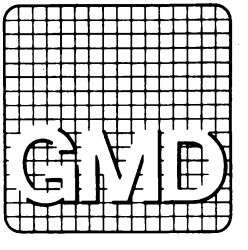
Extendable multi User Microprocessor ELAN-system

**Anwendersoftwareklasse 2**

# Compilerhandbuch
# BASIC

Hochschul-
Rechen-
Zentrum

Universität Bielefeld



# Inhaltsverzeichnis

# Installation des BASIC – Compiler

Zur Installation des BASIC - Compilers muß vom Archiv die Datei *BASIC* geladen und mit *check off* übersetzt werden. Daraufhin stehen folgende Prozeduren zur Verfügung :

PROC basic
PROC basic (TEXT CONST name)
Übersetzen eines BASIC - Programms, das in der Datei *name* stehen muss.

PROC printer (INT CONST channel)
Angabe eines Druckerkanals für die Anweisung PRINT. Bei Angabe von 0 geht die Ausgabe auf das Terminal *(Voreinstellung)*.

Ein Microsoft - BASIC - Compiler ist in Vorbereitung und steht Ende 1984 zur Verfügung.

# 1. Struktur eines BASIC-Programms

Ein BASIC - Programm besteht aus einer Folge von Anweisungen, die mit einer vierstelligen Zeilennummer beginnen. Es wird zwischen ausführbaren und nicht ausführbaren Anweisungen unterschieden:

- Die ausführbaren Anweisungen bewirken eine Aktion während des Programmlaufs.

- Nicht ausführbare Anweisungen beschreiben die für das Programm notwendigen Informationen, bewirken jedoch während des Programmlaufs keine sichtbare Aktion. Sie sollten vor Beginn des eigentlichen Programms stehen.

Die Anzahl der Anweisungen eines Programms ist beschränkt durch die Zeilennummern *(eine ganze Zahl zwischen 1 und 9999)* und die Größe einer Datei, die maximal 4000 Sätze beinhalten kann. Jede Anweisung eines BASIC - Programms entspricht einer Zeile. Durch die Zeilennummern wird die Reihenfolge der Verarbeitung im Programm festgelegt (Diese müssen aufsteigend angeordnet sein). Durch Sprünge kann die Reihenfolge der Verarbeitung geändert werden.

## 1.1 Funktionelle Zusammenhänge von BASIC-Anweisungen

Vereinbarungen für Speicherplatzzuweisungen
(nicht ausführbare Anweisungen)

| | |
|---|---|
| DCL | legt fest, welche Variablen mit einfacher Genauigkeit verarbeitet werden sollen. |
| DIM | legt die Anzahl der Elemente für numerische und alphanumerische Felder fest. |
| COMMON | bestimmt den COMMON - Bereich. |

Zuweisung

| | |
|---|---|
| LET | weist das Ergebniss eines Ausdrucks einer *Variablen zu.* |

Programmverzweigungen

| | |
|---|---|
| FOR/NEXT | dient zur Bildung von Schleifen in Abhängigkeit von einer Laufvariablen. |
| GOTO | unbedingter Sprung zu einer Zeilennummer. |
| IF...THEN | bedingter Sprung zu einer Zeilennummer. |
| ON...GOTO | bedingter Sprung zu einer Zeilennummer in Abhängigkeit vom Wert eines Ausdrucks. |

Unterprogramme und definierte Funktionen

| | |
|---|---|
| CALL | Einschieben eines ELAN - Programms. |
| DEF/FNEND | dient zur Vereinbarung mehrzeiliger Funktionen. |
| GOSUB | Aufruf eines Unterprogramms. |
| ON...GOSUB | Aufruf eines Unterprogramms in Abhängigkeit vom Wert eines Ausdrucks. |
| RETURN | Rücksprung aus einem Unterprogramm. |

Anweisungen zur Ein - /Ausgabe und für interne Files

| | |
|---|---|
| DATA | baut ein internes Datenfile auf. |
| DISP | Ausgabe von Daten im Standardformat über das Display. |
| DISP USING | Ausgabe von Daten im definierten Format über das Display. |
| : (IMAGE) | Festlegung eines Formates in einer Zeile, das dann mit "...USING Zeilennummer" aufgerufen wird. |
| ERASE | Löschen des Bildschirms. |
| INPUT | ermöglicht die Eingabe von Daten über die Tastatur. |

| | |
|---|---|
| PRINT | Ausgabe auf dem Drucker. |
| PRINT USING | Ausgabe in definiertem Format auf dem Drucker |
| READ | Lesen von Daten aus einem internen File. |
| RESTORE | Zurücksetzen des internen Datenfiles auf das erste Element. |
| RKB | Eingabe einer beliebigen Zeichenfolge über die Tastatur. |

Anweisungen für Strings

| | |
|---|---|
| ASSIGN | Die in einem String durch einen Delimiter begrenzten Teilstrings werden einer Liste von Variablen zugeordnet. |
| BUILD | Die Werte einer Liste von Ausdrücken werden einer Stringvariablen im Standardformat zugewiesen, wobei ein Trennzeichen eingefügt werden kann. |
| BUILD USING | Die Werte einer Liste von Ausdrücken werden einer Stringvariablen in definiertem Format zugewiesen. |
| CONVERT | 1. Die ASCII – Codes der Zeichen eines Stringausdrucks werden der Reihe nach den Elementen eines Vektors zugewiesen.<br>2. Die Werte eines Vektors werden als ASCII – Zeichen interpretiert und der Reihe nach einer Stringvariablen zugewiesen. |
| DEPAD | entfernt die Füllzeichen aus einer Stringvariablen. |
| PAD | füllt eine Stringvariable bis zur deklarierten Länge mit ASCII – Zeichen auf. |

Anweisungen für externe Files

| | |
|---|---|
| APPEND: | erlaubt das Anfügen von Daten an einen bestehenden Datenfile. |
| CHAIN | beendet die Programmausführung und startet ein anderes Programm. |
| FILES | legt die maximale Anzahl Files fest, die bei der Ausführung des Programms gleichzeitig geöffnet sein können und öffnet die, deren Namen in der Anweisung aufgeführt sind. |
| FILE: | schließt das dem Filedesignator zugeordnete File und erlaubt das Öffnen eines anderen Files unter Beibehaltung des Filedesignators. |
| READ: | erlaubt das Lesen von Daten aus einem File. |
| RESTORE: | positioniert auf das erste Element eines Files und erlaubt das Lesen. |
| SCRATCH: | positioniert auf das erste Element eines Files und erlaubt das Schreiben. |
| SETW: | positioniert auf eine bestimmte Zeile eines Files. |
| WHERE: | Abfrage der aktuellen Zeile eines Files. |
| WRITE: | schreibt Daten in das angegebene File. |

Spezielle Anweisungen

| | |
|---|---|
| BEEP | akustisches Signal. |
| DELAY | bewirkt die Unterbrechung der Programmausführung während einer bestimmten Zeit. |
| END | bezeichnet das physische Ende eines Programms. |
| RANDOMIZE | Bei Aufruf der Funktion RND wird eine Standardfolge von Zufallszahlen erzeugt. |

| | |
|---|---|
| REM | ermöglicht das Einfügen von Kommentaren in ein Programm. |
| STOP | beendet die Programmausführung |
| TRACE ON | bewirkt den Ausdruck der Zeilennummern in der Reihenfolge der Verarbeitung. |
| TRACE OFF | hebt die Wirkung der Anweisung TRACE ON auf. |

## 1.2 BASIC – Zeichen

Die Sprache BASIC hat einen bestimmten Zeichen – oder Wortvorrat und unterliegt gewissen Syntaxregeln. Unter diesem Aspekt sind alle verwendeten Anweisungen, Daten und Variablen zu sehen.
Die in BASIC vorkommenden Zeichen sind unterteilt in:
- alphabetische,
- numerische und
- Sonderzeichen.

Alphabetische Zeichen:
Dazu gehören die Zeichen A – Z und a – z.

Numerische Zeichen:
sind die Ziffern 0 – 9

Sonderzeichen:
sind in folgender Tabelle zusammengefasst:

| | Name | | Name |
|---|---|---|---|
| | Leerzeichen (blank) | ; | Semikolon |
| = | Gleichheitszeichen | . | Punkt |
| | oder Zuweisung | : | Doppelpunkt |
| + | Additionszeichen | % | Prozentzeichen |
| – | Subtraktionszeichen | ? | Fragezeichen |
| * | Multiplikationszeichen | < | kleiner als |
| / | Divisionszeichen | > | grösser als |
| ↑ | Potenzierung | " | Anführungszeichen |
| ( | öffnende Klammer | , | Komma |
| ) | schliessende Klammer | # | Nummernzeichen |
| | | $ | Dollarzeichen |

Bemerkung
Leerzeichen (blanks) müssen zwischen zwei Namen gesetzt werden. Sie sind nur innerhalb von Strings signifikant.
Nicht zulässig sind sie an folgenden Stellen:
- innerhalb einer Zeilennummer,
- innerhalb von BASIC Wörtern,
- innerhalb von Variablennamen und Funktionen,
- innerhalb von numerischen Konstanten.

## 1.3 Zahlendarstellung

Zahlenbereich
Der Zahlenbereich von doppelt genauen Werten umfasst alle Zahlen von +/- 1e-12 bis +/- 9.999999999999e126. Der Zahlenbereich von einfach genauen Werten umfasst alle Zahlen von -32767 bis 32767.

Genauigkeit
Die Genauigkeit einer Zahl bedeutet die maximale Anzahl signifikanter Ziffern, aus denen die Zahl bestehen kann. Das System kann mit einfach oder doppelt genauen Werten arbeiten. Es gilt:
- für einfache Genauigkeit sind nur ganze Zahlen ohne Exponentialdarstellung möglich,
- für doppelte Genauigkeit sind 13 Ziffern signifikant.

Wird nichts anderes vereinbart, arbeitet das System mit doppeltgenauen Werten.

Externe Darstellung von Zahlen
Zahlen können als
- ganze Zahlen,
- Dezimalzahlen in Festkommadarstellung oder
- Dezimalzahlen in Gleitkommadarstellung

ein- oder ausgegeben werden. Die Wahl der Darstellung ist abhängig von der Größe der Zahlen und der erforderlichen Genauigkeit. Die Zahlen können positiv oder negativ sein. Negative Zahlen werden durch ein Minuszeichen vor der Zahl dargestellt, positiven Zahlen kann ein Pluszeichen vorangestellt werden.

- Ganze Zahlen:
  Sie müssen im Bereich -32767 bis 32767 liegen.

- Dezimalzahlen in Festkommadarstellung:
  Sie können bis zu 14 Stellen aufweisen und mit einem Vorzeichen versehen sein. Dem ganzzahligen Teil folgen ein Dezimalpunkt und bis zu 13 Nachkommastellen.

- Dezimalzahlen in Gleitkommadarstellung:
  Sie bestehen aus einem optionalen Vorzeichen, gefolgt von einer ganzen oder Dezimalzahl (Mantisse), welcher der Buchstabe E ( Exponent) angehängt wird. Die Zahl hinter dem Buchstaben E gibt die Zehnerpotenz an, mit der die Mantisse multipliziert wird, und besteht aus maximal drei Ziffern, die mit einem Vorzeichen versehen sein können. Die Mantisse kann maximal 13 signifikante Ziffern enthalten.

## 1.4 Konstanten und Variablen

### 1.4.1 Konstanten

- Numerische Konstanten
  Eine numerische Konstante ist eine ganze oder eine Dezimalzahl in Fest – oder Gleitkommadarstellung, deren Wert während der Programmausführung unverändert bleibt.

- Konstante pi Die Zahl pi = 3.14562654590 ist als interne Konstante in doppelter Genauigkeit vorhanden. Sie kann mit dem Namen *pi* aufgerufen werden.

- Stringkonstanten
  Eine Stringkonstante besteht aus einer Folge von ASCII – Zeichen, die in Anführungszeichen eingeschlossen sind. Das Anführungszeichen selbst ist nicht Bestandteil der Konstanten. Zugelassen sind alle Zeichen der ASCII – Code – Tabelle mit Ausnahme des Anführungszeichens. Unter der Länge einer Stringkonstanten versteht man die Anzahl der Zeichen innerhalb der Anführungszeichen. Die maximale Länge beträgt 255 Zeichen.

### 1.4.2 Variablen

- Namen
  werden durch einen Buchstaben, dem eine beliebige Folge von Buchstaben und Ziffern folgen kann, dargestellt.

- Numerische Variablen
  sind mit Namen bezeichnete Größen, deren Wert während der Programmausführung verändert werden kann. Numerische Variable werden durch einen Namen dargestellt.
  Eine Variable, der noch kein Wert zugewiesen wurde, hat einen *nicht definierten* Wert. Wird eine solche Variable in einem Ausdruck verwendet, kann eine Fehlermeldung erfolgen.

- Stringvariablen
  enthalten eine Folge von Zeichen und können während des Programmablaufs verändert werden. Sie werden durch einen Namen, dem ein *$ - Zeichen* folgen muß, dargestellt.
  Eine Variable, der noch kein Wert zugewiesen wurde, hat einen *nicht definierten* Wert. Wird eine solche Variable in einem Ausdruck verwendet, kann eine Fehlermeldung erfolgen.
  Stringvariable können bis zu 32000 Zeichen enthalten.

### 1.4.3 Felder (indizierte Variable)

Indizierte Variable bezeichnen ein Feld von Variablen. Ein Feld kann ein - oder zweidimensional sein.

- Ein eindimensionales Feld (Vektor) kann als eine natürliche Folge von Elementen betrachtet werden.

- Ein zweidimensionales Feld ist eine Matrix, bestehend aus Zeilen und Spalten.

Ein Feldelement wird bestimmt durch den Namen des Feldes, zusammen mit einem Index bei einem Vektor, bzw. mit zwei Indizes bei einer Matrix. Die Indizes geben die Position des Elementes im Feld an. Sie können beliebige numerische Ausdrücke sein, die einen ganzzahligen Wert zwischen 1 und der oberen Feldgrenze ergeben, wobei bei einem nicht ganzzahligen Ergebnis gerundet wird.

Die Dimension eines Feldes (Anzahl Elemente) wird durch den Befehl DIM festgelegt. Ein Feld darf nicht den gleichen Namen wie eine Variable haben.

Feldvereinbarung

Unter Feldvereinbarung ist die Angabe der Felddimension (ein – oder zweidimensional) und der Anzahl Elemente zu verstehen. Eine Feldvereinbarung wird mit Hilfe der BASIC – Anweisung DIM getroffen. Fehlt die Vereinbarung für bestimmte Felder im Programm, so werden sie vom System auf 10 bzw. 10*10 gesetzt.

Der Speicherplatz der Daten während der Programmausführung ist in der nachfolgenden Tabelle dargestellt:

| Datentyp | Platzbedarf |
|---|---|
| einfachgenaue num. Konstante | 2 Bytes |
| doppeltgenaue num. Konstante | 8 Bytes |
| Stringkonstante | n Bytes<br>n = Anzahl der Zeichen |
| einfachgenaue num. Variable | 2 Bytes |
| doppeltgenaue num. Variable | 8 Bytes |
| Stringvariable | 16 Bytes Stack<br>n > 16 dann +<br>n Bytes Heap<br>n = Anzahl der Zeichen |
| Felder | Summe der Elementlängen |

# 1.5 Ausdrücke und Vergleichsoperatoren

## 1.5.1 Ausdrücke

Ein Ausdruck besteht aus einer Verknüpfung von Konstanten, (indizierten) Variablen und/oder Funktionen.
Für numerische Ausdrücke sind sowohl die Standardfunktionen als auch ein- oder mehrzeilige, vom Anwender definierte Funktionen zulässig. Die Verknüpfungsoperatoren für numerische Ausdrücke sind:

| Symbol | Bedeutung |
|---|---|
| ^ | Potenzierung |
| * | Multiplikation |
| / | Division |
| + | Addition |
| - | Subtraktion |

Für Stringausdrücke sind sowohl die Standardstringfunktionen als auch einfache oder mehrzeilige, vom Anwender definierte Funktionen zulässig. Der Verknüpfungsoperator für Stringausdrücke ist ausschliesslich die Stringaddition (Symbol + ).
Für Boolesche Ausdrücke sind folgende Operatoren zulässig:

| | |
|---|---|
| NOT | Negation |
| AND | logisches Und |
| OR | logisches Oder |
| = | Vergleich auf Gleicheit |
| < > | Vergleich auf Verschiedenheit |
| > | Vergleich auf größer |
| < | Vergleich auf kleiner |
| < = | Vergleich auf kleinergleich |
| > = | Vergleich auf größergleich |

## 1.5.2 Regeln für Operatoren

- Negation

| | |
|---|---|
| NOT a | Der logische Wert *a* wird negiert. |

- Potenzierung

| | |
|---|---|
| b^e | Die Basis *b* wird zur Potenz e erhoben. |

- Multiplikation und Addition

| | |
|---|---|
| a*b | *a* wird mit *b* multipliziert. |
| a + b | *b* wird zu *a* addiert. |

– Division und Subtraktion

| | |
|---|---|
| a/b | *a* wird durch *b* dividiert. |
| a – b | *b* wird von *a* subtrahiert. |

– Vergleiche

| | |
|---|---|
| a = b | *a* gleich *b*. |
| a < > b | *a* ungleich *b*. |
| a > b | *a* größer als *b*. |
| a < b | *a* kleiner als *b*. |
| a > = b | *a* größergleich *b*. |
| a < = b | *a* kleinergleich *b*. |

– Prioritätsregeln

Die Operatoren haben in nachstehender Reihenfolge folgende Priorität:

| | |
|---|---|
| NOT | höchste Priorität |
| ^ | |
| *, / | |
| +, – | |
| =, < >, <, < =, >, > = | |
| AND | |
| OR | |

Operatoren gleicher Priorität werden von links nach rechts verarbeitet, es sei denn es wurde durch das Setzen von Klammern die Prioritätsfolge geändert.

## 1.6 Common Bereich

Der Sinn des COMMON – Bereichs liegt im Datenaustausch zwischen Programmen, ohne daß Daten in einem externen Datenfile zwischengespeichert werden müssen. Dazu wird ein *geschützter Bereich* reserviert, dessen Inhalt bei der Beendigung eines Programmlaufs erhalten bleibt.

Bei der Definition des COMMON – Bereichs über die BASIC – Anweisung COMMON sind die gewünschten Variablen zu spezifizieren. Diese müssen in verschiedenen Programmen bezüglich Typ und Deklaration übereinstimmen, was durch die Anweisung DCL und DIM erreicht wird (Stringvariablen können unterschiedliche Längen haben).

## 1.7 Verbindung mit ELAN

Mit Hilfe der Anweisung CALL können Programmteile in ELAN geschrieben werden. Dabei können alle Variablen des BASIC - Teils mittels ihres Namens angesprochen werden, bei Stringvariablen muss das *$ - Zeichen* durch *dollar* ersetzt werden. Variablen, die zum COMMON - Bereich gehören, müssen mit *common.name* angesprochen werden. Die geöffneten Files können mit *file (file-designator)* angesprochen werden.

## 1.8 Standardformat

Bei Anweisungen des Typs
- BUILD
- DISP
- PRINT

werden die Ergebnisse von numerischen oder alphanumerischen Ausdrücken im Standardformat dargestellt bzw. übergeben.

### 1.8.1 Zahlendarstellung

Darstellung ganzer Zahlen

Ganze Zahlen werden linksbündig entsprechend ihrer Stellenzahl ausgegeben. Bei negativen Zahlen wird zuerst das Vorzeichen '-' ausgegeben.
Ist die Zahl in einfacher Genauigkeit dargestellt, so erfolgt die Ausgabe mit maximal 5 Ziffern. Ist sie intern in doppelter Genauigkeit dargestellt, so wird nach der Zahl ein Punkt ausgegeben, oder, wenn sie mehr als 8 Stellen hat, im Gleitkommaformat dargestellt.

Dezimalzahlen im Festkommaformat

Die Darstellung von Dezimalzahlen erfolgt mit maximal 8 Ziffern. Zusätzlich werden eine Stelle für ein negatives Vorzeichen und eine Stelle für den Dezimalpunkt benötigt.
Führende Nullen werden unterdrückt. Nullen am Ende der Zahl werden im Dezimalteil nicht dargestellt. Die Anzahl der Nachkommastellen in der Darstellung richtet sich nach der Anzahl der Vorkommastellen, wobei die Summe aus Vor- und Nachkommastellen maximal 7 beträgt. Ist die Summe aus signifikanten Vor- und Nachkommastellen kleiner als 7, so erfolgt

die Darstellung mit entsprechend weniger Stellen. Hat die Zahl 8 Vorkommastellen, so wird zwar der Dezimalpunkt, jedoch keine Nachkommastellen ausgegeben. Hat die darzustellende Zahl einen Absolutbetrag, der kleiner als 1 ist, so wird keine 0 vor dem Dezimalpunkt ausgegeben.

Zahlendarstellung im Gleitkommaformat

Zahlen haben im Gleitkommaformat folgendes Format:

- evtl. Vorzeichen Mantisse
- Mantisse
  - 1 Vorkommastelle (ungleich Null)
  - Dezimalpunkt
  - max 7 Nachkommastellen
- e (Kennzeichen für den Exponenten zur Basis 10)
- evtl. Vorzeichen des Exponenten
- Exponent (max 3 Stellen)

Insgesamt erfordert die Gleitkommadarstellung also max. 14 Zeichen. Der Übergang zur Gleitkommadarstellung erfolgt dann, wenn mehr signifikante Stellen vorhanden sind, als in der Festkommadarstellung ausgegeben werden können.

### 1.8.2 Darstellung von Strings

Strings werden zeichenweise linksbündig dargestellt. Die Anzahl der ausgegebenen Zeichen entspricht der aktuellen Länge des Strings.

### 1.8.3 Stellenkontrolle bei den Anweisungen DISP und PRINT

Das Trennzeichen Komma ','

Enthält die Liste der Ausgabeelemente das Trennzeichen Komma, so wird die Zeile in fünf Zonen zu je 16 Zeichen unterteilt. Diese Zonen beginnen bei den Positionen 1, 17, 33, 49 und 65. Ist das dem zuletzt dargestellten Ausgabeelement folgende Trennzeichen das Komma, so wird auf die nächste Zone positioniert.

Trennzeichen Semikolon ';'

Das Trennzeichen bewirkt keine Änderung der Stellung des Pointers. Durch getrennte Ausgabeleemente werden also unmittelbar aneinander anschliessend ausgegeben.

Funktion TAB (num. Ausdruck)

Die Funktion TAB erlaubt es, eine beliebige Position direkt anzulaufen. Ist diese Position bereits belegt, so wird auf die

nächste Zeile positioniert.
Ist der Wert des numerischen Ausdrucks kleiner als 1, so erfolgt eine Fehlermeldung. Ist der Wert > 80, so wird auf die nächste Zeile positioniert auf die Stelle Wert MODULO 80. Um durch das Trennzeichen keine weitere Tabulation zu bewirken, muß nach Aufruf der Funktion TAB das Trennzeichen (;) gesetzt werden.

Trennzeichen am Ende der Anweisung DISP oder PRINT
Wird hinter das letzte Element einer Ausgabeliste ein Trennzeichen (';' oder ',') gesetzt, so werden die Werte der nachfolgen Ausgabeanweisungen direkt angefügt. Dadurch ist es möglich, die Elemente mehrerer PRINT - oder DISP - Anweisungen in einer Zeile auszugeben.
Fehlt am Ende der Ausgabeliste das Trennzeichen, so wird auf die nächste Zeile positioniert.
Ist die Ausgabeliste leer, so wird auf nächste Zeile positioniert.

## 2.1 APPEND

Anweisung: APPEND:

Funktion: Anfügen von Daten an ein sequentielles File.

Format: `APPEND: filedesignator`
*filedesignator* ist ein numerischer Ausdruck.

Wirkung: In dem durch *filedesignator* angegebenen File wird der Pointer auf die erste leere Zeile gesetzt. Bei einer nachfolgenden WRITE: - Anweisung mit dem gleichen *filedesignator* werden die Daten an die bereits bestehenden angefügt. Ergibt die Berechnung des Ausdrucks für den *filedesignator* keinen ganzzahligen Wert, wird dieser gerundet.

Bemerkung: Der Wert des *filedesignators* muß größer als Null und kleiner oder gleich der Anzahl Filenamen in der Anweisung FILES sein.

Beispiel :

```
00010 FILES SD4T
00020 REM Dieser Teil beschreibt das File von Beginn an
00030 SCRATCH: 1
00040 FOR i=1 TO 10
00050   WRITE: 1, i
00060 NEXT i
00070 REM Es werden die ersten 5 Daten gelesen
00080 RESTORE: 1
00090 FOR i=1 TO 5
00100   READ: 1, a
00110   DISP a,
00120 NEXT i
00130 DISP
00140 REM Neue Daten werden hinzugefügt
00150
00160
00170 APPEND: 1
00180 FOR i=11 TO 15
00190   WRITE: 1, i
00200 NEXT i
00210 REM Lesen des gesamten Files
00220 RESTORE: 1
00230 READ: 1, a EOF 260
00240 DISP a; " ";
00250 GOTO 230
00260 DISP
00270 DISP "File-Ende"
00280 END
```

## 2.2 ASSIGN

Anweisung: ASSIGN

Funktion: Zuweisung von Werten aus einem Stringausdruck an eine oder mehrere numerische oder alphanumerische Variable. Als Datentrennzeichen dient ein fixer Delimiter.

Format: `ASSIGN Stringausdr., Variable [,...]; delimiter d`

*delimiter d:* numerischer Wert eines beliebigen ASCII – Zeichens, 0 < delimiter d < 255

Wirkung: Das System berechnet den Stringausdruck. Entsprechend der Zahl *delimiter d* wird die entstandene Zeichenkette in verschiedene Strings aufgeteilt, die mit ihrem numerischen oder alphanumerischem Wert den Variablen der Variablenliste zugewiesen werden.

Bemerkung: Einer numerischen Variablen in der Variablenliste muß ein numerischer Wert als Ergebnis zugewiesen werden. Die Anzahl der Strings, in die die Zeichenkette aufgeteilt wird und die durch den Delimiter d getrennt sind, muß größer oder gleich der Anzahl der Variablen in der Variablenliste sein.

Beispiel:

```
00010 DISP "Bitte Vor und Zuname eingeben";
00020 RKB a$
00030 IF length (a$) = 0 THEN 90
00040 REM Trennung des Namens in Vor und Zuname
00050 REM ------------------------------------
00060 ASSIGN a$, v$, z$; 32
00070 DISP "Vorname: "; v$, "Zuname: ";z$
00080 GOTO 20
00090 END
```

## 2.3 BEEP

Anweisung: BEEP

Funktion: Ausgabe eines akustischen Signals.

Format: BEEP

Wirkung: Während ca. 0.2 Sekunden ertönt ein Signal.

Beispiel:

```
00010 DISP "Bitte eine Zahl zwischen 1 und 10 eingeben:";
00020 INPUT i
00030 IF i >= 1 AND i <= 10 THEN 60
00040 BEEP
00050 GOTO 10
00060 END
```

## 2.4 BUILD

Anweisung: BUILD

Funktion: Übertragung des Ergebnisses einer Liste mit numerischen und/oder Stringausdrükken an eine Stringvariable.

Format: `BUILD Stringvar., Ausdruck [,...] [;delimiter d]`

Wirkung: Die Ausdrücke werden berechnet und die Ergebnisse im Standardformat an die Stringvariable übertragen. Die Elemente werden durch das dem angegebenen Delimiter entsprechenden Zeichen getrennt. Fehlt die Delimiter – Angabe, weist die Stringvariable kein Trennzeichen zwischen den Variablen aus.

Bemerkung:
- Der Wert eines numerischen Ausdrucks wird im Standardformat an die Stringvariable übergeben.
- Ist das Resultat eines Ausdrucks eine Zeichenkette, werden die einzelnen Zeichen ohne Veränderung an die Stringvariable übertragen.

Beispiel:

```
00010 REM  Beispiel 'BUILD'
00020 REM
00040 DISP "a und b eingeben ";
00050 INPUT a,b
00060 LET c=a**b
00070 BUILD a$, a, "HOCH", b, "=", c; 32
00080 DISP a$
00090 GOTO 40
00100 END
```

## 2.5 BUILD USING

Anweisung: BUILD USING

Funktion: Übertragung des Ergebnisses einer Liste mit numerischen und/oder Stringausdrücken an eine Stringvariable.

Format: `BUILD USING Format, Stringvar, Ausdruck[,...]`
*Format:* Zeilennr einer Imageanweisung oder Stringausdruck

Die Zeilennummer entspricht der Nummer derjenigen Programmzeile, die das Format der Ergebnisstringvariablen festlegt. Anstelle der Zeilennummer kann der Name einer das Format definierenden Stringvariablen stehen.

Wirkung: Die Ausdrücke werden berechnet und die Resultate an die Stringvariable übertragen. Das Format ist jenes, das in der Programmzeile mit der in der Anweisung angegebenen Nummer steht.
Steht in einer Anweisung anstelle der Zeilennummer der Name einer Stringvariablen, werden die Ergebnisse in dem darin angegebenen Format übertragen.

Bemerkung:
- Die für die Formatspezifikation von Zeichenstrings verwendbaren Zeichen sind in der Beschreibung *Formatspezifikationen* aufgeführt.
- Die bei Formatspezifikation angegebene Zeilennummer muß kleiner als die aktuelle Zeilennummer sein.

Beispiel :
```
00010 REM Beispiel für 'BUILD USING'
00020 REM
00040 DISP "a und b eingeben";
00050 INPUT a,b
00060 LET c=a**b
00070 LET d$= "#####.### HOCH #####.### = #######.###"
00080 BUILD USING d$, e$, a, b, c
00090 DISP e$
00100 GOTO 40
00110 END
```

## 2.6 CALL

Anweisung: CALL

Funktion: Einbinden von ELAN – Programmen in ein Basic Programm.

Format: CALL

Wirkung: Das ELAN – Programm wird in das BASIC – Programm eingefügt. Die Variablen des BASIC – Programms können mit ihrem Namen angesprochen werden, statt des *$ – Zeichens* muß *dollar* geschrieben werden. Bei Variablen des COMMON – Bereichs muß vor dem Namen ein *common.* stehen.
Variable mit einfacher Genauigkeit sind INT's, mit doppelter Genauigkeit REAL's und Strings sind TEXTe.

Bemerkung: Wenn der ELAN – Teil über mehrere Zeilen gehen soll, müssen die ersten fünf Spalten freibleiben.

Beispiel:

```
00010 COMMON a, b$
00020 DIM c (10, 10)
00030 CALL
      INT VAR i;
      FOR i FROM 1 UPTO 10
      REP c (i) (i) := common.a PER;
      put line ("Der Text ist : " + common.b dollar);
00040 GOTO 20
00050 END
```

## 2.7 CHAIN

Anweisung: CHAIN

Funktion: Unterbrechung der Verarbeitung eines Programms und Ausführung eines anderen Programms.

Format: CHAIN Stringausdruck

Wirkung: Die Verarbeitung des laufenden Programms wird beendet und alle in diesem Programm verwendeten Files geschlossen. Das Programm mit dem unter *Stringausdruck* angegebenen Namen wird aufgerufen und mit seiner Übersetzung begonnen.

Bemerkung: Alle im durch CHAIN aufgerufenen Programm verwendeten Files müssen mit der Anweisung FILES geöffnet werden, auch wenn im aufrufenden Programm mit den gleichen Files gearbeitet wurde.

```
00010 FILES SFILE
00020 DISP "Anfang Hauptprogramm 'Haupt'"
00030 SCRATCH: 1
00040 WRITE: 1, "Haupt"
00050 CHAIN "UP2"
00060 END
```

```
00010 FILES SFILE
00020 DISP "Anfang Unterprogramm 'UP1'"
00030 DISP "Welches Programm soll Rechnen ";
00040 RKB a$
00050 APPEND: 1
00060 WRITE: 1, "UP1"
00070 CHAIN a$
00080 END
```

```
00010 FILES SFILE
00020 DISP "Anfang Unterprogramm 'UP2'"
00030 DISP "Folgende Programme haben gerechnet:"
00040 READ: 1, a$ EOF 70
00050 DISP a$
00060 GOTO 40
00070 DISP "UP2"
00080 END
```

## 2.8 COMMON

Anweisung: COMMON

Funktion: Definition der Variablen in einem BASIC - Programm, deren Werte in verschiedenen Programmen Verwendung finden sollen.

Format: `COMMON Name [()] [, Name [()]...]`
*Name:* einer Variablen oder eines Feldes, das gemeinsam benutzt werden soll.

Wirkung: Die COMMON - Anweisung legt für das laufende Programm die Variablen fest, die ihre Werte an ein anderes Programm übergeben oder von einem anderen Programm eine Wertzuweisung erhalten sollen.
Der Speicherplatz für die Variablen der COMMON - Anweisung wird in der dort festgelegten Reihenfolge mit den gültigen Deklarationen und Dimensionen im COMMON - Bereich reserviert.

Bemerkung:
- Um den gemeinsamen Zugriff zu ermöglichen, reserviert das System einen COMMON - Bereich, der durch das Ende der Programmausführung nicht gelöscht wird. Die Grösse beträgt max. 1 MByte.
- In einem BASIC - Programm darf nur eine COMMON - Anweisung vorhanden sein.
- Die Variablen, die Werte an den COMMON - Bereich übergeben und jene, die diese Werte dem COMMON - Bereich entnehmen, müssen nicht den gleichen Namen tragen, jedoch in Typ und Deklaration übereinstimmen.
- Werte von Elementen eines Feldes können in einem Folgeprogramm auf mehrere Felder aufgeteilt werden, wobei jedoch die Summe der Elemente gleich sein muß. Bei numerischen Feldern muß außerdem die Genauigkeit (doppelt oder einfach) übereinstimmen.
- Der Inhalt einer Stringvariablen kann nicht in Teilstrings zerlegt werden.
- Der für den COMMON - Bereich reservierte Platz kann in verschiedenen Programmen unterschiedlich groß sein.

- Wenn den Variablen eines COMMON - Bereichs noch kein Wert zugewiesen wurde, haben einfach-genaue Variablen den Wert - 1, doppelt genaue sind undefiniert und Strings haben die Länge - 1.

Beispiel:

```
00010 COMMON x (), a$ (), b$
00020 DIM x (2,2), a$ (4)
00030 DCL S x (), 6 a$ (), 20 b$
00040 LET x (1,1) = 1
00050 LET x (1,2) = 2
00060 LET x (2,1) = 3
00070 LET x (2,2) = 4
00080 LET a$(1) = "Eins"
00090 LET a$(2) = "Zwei"
00100 LET a$(3) = "Drei"
00110 LET a$(4) = "Vier"
00120 LET b$ = "in Worten lautet "
00130 REM
00140 REM
00150 CHAIN "PROG2"
00160 END
```

```
00010 COMMON z(), a$(), b$
00020 DIM a$(4), z(4)
00030 DCL S z(), 6 a$(), 20 b$
00040 FOR i = 1 TO 4
00050 DISP z (i); b$, a$ (i)
00060 NEXT i
00070 END
```

## 2.9 CONVERT

Anweisung: CONVERT

Funktion: Umwandlung einer Zeichenkette in die entspechenden numerischen Codes der ASCII - Tabelle oder umgekehrt.

Format: 1) `CONVERT Stringausdr. TO num.Vektor LENGTH num.Var.`

2) `CONVERT num. Vektor TO Stringvar. LENGTH num. Ausdr.`

Wirkung: 1. Format:
Der *Stringausdruck* wird berechnet. Jedes Zeichen des sich daraus ergebenden Strings erhält den entsprechenden numerischen Code aus der ASCII - Tabelle und wird der Reihe nach den Elementen des *Vektors* aus der Anweisung zugewiesen. Die *Längen* - Variable erhält die Anzahl der umgewandelten Zeichen.

2. Format:
Der *numerische Ausdruck* wird berechnet und das Ergebnis auf die nächste ganze Zahl n gerundet. Die Werte der ersten n Elemente des *Vektors* aus der Anweisung werden ebenfalls auf die nächste ganze Zahl gerundet und dann in die entsprechenden Zeichen der ASCII - Tabelle umgewandelt. Der daraus entstehende Zeichenstring wird der entsprechenden Stringvariablen zugewiesen.

Beispiel:

```
00010 REM  *Beispiel fuer die Anweisung CONVERT
00020 REM
00030 DCL S (b, i, l, n)
00040 DIM a (32)
00050 DISP
00060 DISP
00070 DISP "String eingeben";
00080 RKB a$
00090 CONVERT a$ TO b LENGTH l
00100 DISP "a$= '";a$;"'"
00110 DISP "Laenge";l
00120 DISP "ASCII CODE";
00130 FOR i% = 1 TO l
00140 DISP b(i);
00150 NEXT i;
00160 DISP
00170 DISP
00180 DISP "Laenge des Vektors";
00190 INPUT n
00200 DISP "Laenge";n
00210 DISP "ASCII-CODE";
00220 FOR i = 1 TO n
00230 DISP "Element";i
00240 INPUT b(i)
00250 DISP b(i)
00260 NEXT i
```

```
00270 DISP
00280 CONVERT b TO a$ LENGTH n
00290 DISP "a$=  '" ;a$;"'"
00300 GOTO 50
00310 END
```

## 2.10 DATA

Anweisung: DATA

Funktion: Erzeugen eines internen Files von Daten, die anschliessend den Variablen aus der READ - Anweisung zugewiesen werden.

Format: `DATA Konstante [,Konstante...]`

Wirkung: Es wird ein File erzeugt, der alle Konstanten aller DATA - Anweisungen des Programms enthält. Ein Pointer zeigt auf das erste Element der Tabelle. Die Konstanten der Tabelle werden den Variablen aus der READ - Anweisung zugewiesen. Nach jeder Zuweisung zeigt der Pointer auf das jeweils nächste Element der Tabelle. (Mit der Anweisung RESTORE kann der Pointer wieder auf das erste Element zurückgesetzt werden.)

Bemerkung:
- Stringkonstanten müssen in Anführungszeichen stehen.
- Die DATA - Anweisungen können an jeder Stelle im Programm stehen. Jeder Stringvariablen muß eine Stringkonstante und jeder numerischen Variablen muß eine numerische Konstante zugewiesen werden.

Beispiel:

```
00010 DISP
00020 DISP "ihr Typ:"
00030 DISP
00040 RESTORE
00050 FOR i= 1 TO 5
00060 READ a$
00070 DISP a$;
```

```
00080 RKB b$
00090 DISP a$:":".b$
00100 NEXT i
00110 GOTO 10
00120 DATA "Alter","Groesse","Gewicht","Haarfarbe",
00130 DATA "Bes.Kennz."
00140 END
```

## 2.11 DCL

Anweisung: DCL (declare)

Funktion: Festlegung der mit einfacher Genauigkeit zu verarbeitenden Variablen und der maximal vorgesehenen Längen für Strings.

Format:
```
DCL S (num. Var. [()] [, num. Var. [()]...])
    n (Stringvar.[()] [, Stringvar.[()]...])
    SINGLE
```
*n* ist eine ganze Zahl zwischen 1 und 32000. Nach einem Feldnamen kann ein leeres Klammerpaar folgen. Beziehen sich *S* und *n* auf eine einzige Variable, können die Klammern entfallen.

Wirkung: Die Werte der numerischen Variablen zwischen den Klammern hinter dem *S* werden in einfacher Genauigkeit dargestellt. Wird bei einer DCL – Anweisung der Parameter *SINGLE* angegeben, werden alle Werte der numerischen Variablen in einfacher Genauigkeit gerechnet. Die Stringvariablen, auf die sich der Parameter *n* bezieht, werden mit *n*Blanks initialisiert.

Bemerkung:
- Bezieht man sich in einem Programm mehrmals, durch verschiedene DCL – Anweisungen, auf dieselbe Variable, hat die letzte Anweisung Gültigkeit.

## 2.12 DEF

Anweisung: DEF

Funktion: Nicht ausführbare Anweisung, die innerhalb einer Zeile eine numerische oder eine Stringfunktion definiert (Definition einer einzeiligen Funktion).

Format:
```
DEF FN@  (Parameter) = num. Ausdruck
DEF FN@$ (Parameter) = Stringausdruck
```
*@:* Folge von Buchstaben oder Ziffern.

*Parameter:* einfache alphanumerische Variablen.

*FN@* Name einer numerischen Funktion, die als Ergebniss einen numerischen Wert liefert. Parameter können numerische oder Stringvariablen sein.

*FN@$* Name einer Stringfunktion. Als Ergebnis wird ein String geliefert. Parameter können numerische oder Stringvariable sein.

Die in einer DEF - Anweisung innerhalb der Klammern stehenden Parameter sind Pseudovariable, die keinen Bezug zu gleichnamigen Variablen außerhalb der Funktion haben. Beim Funktionsaufruf sind diese Parameter durch entspechende Variable zu ersetzen, wodurch der Pseudovariablen der aktuelle Wert der an entsprechender Stelle im Aufruf stehenden Variablen zugewiesen wird. Die Pseudovariablen werden *formale Parameter* genannt.
Auf der rechten Seite der Anweisung können ausser den Parameternamen auch andere Variable, sogenannte globale Variable, stehen. Diese müssen aber schon vor dem Funktionsaufruf einen Wert besitzen.

Wirkung: Die Funktion FN@ oder FN@$ wird durch den Ausdruck auf der rechten Seite definiert.

Bemerkung:

- Eine Funktion kann an jeder Stelle im Programm stehen, darf jedoch nur ein einziges mal definiert werden und in dieser Form nur eine Zeile umfassen.

- Direkte rekursive Aufruffolgen
  ```
  10 DEF FNA=FNA
  ```
  oder indirekte rekursive Aufruffolgen
  ```
  10 DEF FNA=x+FNB
  20 DEF FNB=FNA+y
  ```
  sind erlaubt.

- Maximal 16 Parameter können verwendet werden. Formale Parameter haben keinerlei Beziehung zu Variablen mit gleichem Namen.

- Tritt während eines Programmlaufs die Anweisung DEF auf, wird die Verarbeitung in der folgenden Programmzeile fortgesetzt.

- Die aktuellen Parameter müssen in Typ und Anzahl mit den Formalen übereinstimmen.

Beispiel:

```
00010 DEF FNB (x) = pi/180.0 * x
00020 DEF FNC (x) = COS (FNB (x))
00030 DEF FNS (x) = SIN (FNB (x))
00040 DEF FNT (x) = TAN (FNB (x))
00050 DEF FNA (x) = 180.0*ATN(x)/pi
00060 DISP "   x         sin         cos         tan atn"
00070 DISP
00080 DISP "von Grad?, bis Grad?, Schrittweite";
00090 INPUT a, b, c
00100 FOR i=a TO b STEP c
00110 DISP i, FNS (i), FNC (i), FNT (i), FNA (i)
00120 NEXT i
00130 DISP
00140 GOTO 80
00150 END
```

## 2.13 DEF/FNEND

Anweisung: DEF/FNEND (define/function end)

Funktion: Nicht ausführbare Anweisung zur Definition einer mehrzeiligen numerischen oder Stringfunktion.

Format:

```
DEF FN@[$] [(Parameter)] [lokale Variablen]
     .
     .
     .
    FN* = num. Ausdruck
    FN*$= Stringausdruck
    mind. eine Zuweisung
     .
     .
     .
FNEND
```

| | |
|---|---|
| *FN@* | Name einer numerischen Funktion; dem Funktionsnamen wird der errechnete Wert zugewiesen. |
| *FN@$* | Name einer Stringfunktion; dem Funktionsnamen wird als Ergebnis ein Zeichenstring zugewiesen. |
| *Parameter* | Einfache numerische oder alphanumerische Variable. |
| *lokale Variable* | Im Funktionsprogramm verwendete numerische oder alphanumerische Variable, die zu gleichnamigen Variablen im Hauptprogramm keine Beziehung haben. |
| *FN*/FN*$* | Pseudovariable, welcher vor dem Rücksprung aus der Funktion FN@ oder FN@$ der jeweilige Funktionswert zugewiesen wird. |

In numerischen oder Stringausdrücken können außer Parametern und lokalen Variablen auch globale Variable auftreten.
Zwischen den Programmzeilen DEF und FNEND können neben den Anweisungen für die Berechnung der Werte der Pseudovariablen (*FN** und *FN*$*) beliebige BASIC - Anweisungen stehen.

Wirkung: Die Funktion *FN@* oder *FN@$* wird durch alle Anweisungen zwischen den Programmzeilen DEF *FN@$* oder DEF *FN@$*
und der Zeile FNEND definiert.

Bemerkung:

- Eine sich über mehrere Zeilen erstreckende Funktion kann an jeder Stelle des Programms definiert werden.

- Funktionsdefinitionen sind physisch in sich abgeschlossene Programmteile. Es darf deshalb in eine Funktionsdefinition weder verzweigt noch diese durch eine Sprunganweisung verlassen werden (z.B. IF...THEN, READ...EOF).

- Verzweigungen innerhalb einer Funktion sind zulässig.

- Der Wert einer globalen Größe wird innerhalb einer Funktion verändert, wenn die Funktion eine Zuweisung an die globale Variable enthält. Im inneren einer Funktion sind Wertzuweisungen an Parameter nicht gestattet. Die aktuellen Parameter müssen in Typ und Anzahl den formalen entspechen. Tritt während der Verarbeitung eines Programms die Anweisung DEF auf, wird die Verarbeitung in der dem FNEND folgenden Programmzeile fortgesetzt.

- Parameter haben keinerlei Beziehung zu im Programm vorkommenden Variablen mit gleichem Namen.

- Namen von lokalen Variablen haben keinerlei Beziehung zu etwaigen, gleichlautenden Namen im Hauptprogramm, d. h., die Variablen im Hauptprogramm werden durch eine Wertzuweisung an eine gleichnamige lokale Variable nicht verändert.

Beispiel:

```
00010 DEF FNA (i, j)l
00020 LET l=i-int(i/j)*j
00030 IF l=0 THEN 70
00040 LET i=j
00050 LET j=l
00060 GOTO 20
00070 LET FN*=j
00080 FNEND
00090 DISP "Eingabe a,b";
00100 INPUT a,b EOF 140
00110 DISP "Der groesste gemeinsame Teiler von";a,;"und";
00120 DISP b;"ist";FNA(a,b)
00130 GOTO 90
00140 END
```

## 2.14 DELAY

Anweisung: DELAY

Funktion: Pause vor der Ausführung des nächsten Programmschrittes.

Format: DELAY n

Wirkung: Der folgende Programmschritt wird erst nach n Zehntelsekunden ausgeführt.

Bemerkung: Die Wirkung von DELAY kann durch Drücken irgendeiner Taste aufgehoben werden.

Beispiel:

```
00100 DISP "Eingabe der Matrix A"
00110 DELAY 20
00120 DISP "Anzahl Zeilen, Anzahl Spalten";
00130 INPUT n,m
```

## 2.15 DEPAD

Anweisung: Eliminieren von Füllzeichen am Ende einer Stringvariablen.

Format: `DEPAD Stringvariable, n`
*n*: ganze Zahl zwischen 0 und 255.

Wirkung: Aus Stringvariablen werden die Zeichen entfernt, die in der ASCII – Tabelle der Zahl n entsprechen. Sie werden rechts beginnend eliminiert, und zwar so lange, bis das erste, nicht dem ASCII – Code n entsprechende Zeichen auftritt.

Beispiel:

```
00010 REM Beispiel für 'PAD' und 'DEPAD'
00020
00030 DCL 32 a$
00040 DISP
00050 DISP
00060 DISP "String eingeben ";
00070 RKB a$ EOF 140
00080 DISP "Eingabe: '";a$;"'"
00090 PAD a$, 42
00100 DISP "A$ nach 'PAD': '";a$;"'"
00110 DEPAD a$, 42
00120 DISP "A$ nach 'DEPAD': '";a$;"'"
00130 GOTO 40
00140 END
```

## 2.16 DIM

Anweisung: DIM (dimension)

Funktion: Festlegung der Dimension eines oder mehrerer Felder.

Format: `DIM Feldname (Zeilen [,Spalten]) [, ...]`

Wirkung: Folgt einem Feldnamen nur eine Zahl r zwischen Klammern, so handelt es sich um einen Vektor mit r Elementen. Alle im Programm verwendeten Feldindizes müssen kleiner oder gleich r sein. Folgt einem

Feldnamen ein Zahlenpaar zwischen Klammern (r,c), so handelt es sich um eine Matrix mit r Zeilen und c Spalten. Die Indizes dürfen die jeweiligen Feldgrenzen nicht überschreiten ( < = r und > = c). Die Indizes einer Feldvereinbarung müssen größer als 0 sein.

Bemerkung:

- Wird ein eindimensionales Feld nicht durch DIM deklariert, erhält es vom System 10 Elemente zugewiesen.
- Wird ein zweidimensionales Feld nicht durch DIM deklariert, erhält es vom System 10 Zeilen und 10 Spalten zugewiesen.
- Der vom Compiler zugelassenen Höchstwert beträgt 32767 Elemente pro Dimension.
- Die Anweisung DIM kann an jeder Stelle in einem Programm stehen.
- Ein eindimensionales Feld darf nicht den gleichen Namen haben wie eine einfache Variable oder ein zweidimensionales Feld.

Beispiel:

```
00010 REM Beispiel 'DIM'
00020 REM
00030 DCL S (i,j)
00040 DIM a (2,3)
00050 FOR i=1 TO 2 STEP 1
00060 FOR j=1 TO 3
00070 LET a (i,j) = rnd*100.0
00080 NEXT j
00090 NEXT i
00100 DISP "Welches Element";
00110 INPUT i,j EOF 150
00120 IF i < 1 OR i > 1 OR j < 1 OR j > 3 THEN 100
00130 DISP "A (";i;", ";j;") = ";a(i,j)
00140 GOTO 100
00150 END
```

## 2.17 DISP

Anweisung: DISP (display)

Funktion: Ausgabe von numerischen und/oder alphanumerischen Daten im Display

Format: `DISP (Ausdruck/TAB (num. Ausdruck)) [(,/;)...]`
,/; Trennzeichen mit einer bestimmten Bedeutung für die Ausgabe (siehe Kapitel 8.5).

Wirkung: Die Ergebnisse der Ausdrücke werden im Standardformat dargestellt und im Display sichtbar gemacht.
Die Position der Ausdrücke in einer Zeile hängt sowohl von der erforderlichen Länge der Darstellung als auch von den verwendeten Trennzeichen (Komma oder Semikolon) und der Funktion TAB ab.

Stellenkontrolle der Zeichen im Display
Die Ergebnisse eines Ausdrucks werden in Form einer Folge von Zeichen auf dem Display dargestellt.
Ergibt die Darstellung der Liste der Ausdrücke mehr Zeichen als in eine Zeile passen, werden die überzähligen in der nächsten Zeile gezeigt oder verschluckt.

Bemerkung:
- Die Anweisung DISP ohne Parameter bewirkt einen Zeilenvorschub
- Das Standardformat ist in Abschnitt 8.5 beschrieben

Beispiel:
```
00010 DISP "Werte von X und Y";
00020 INPUT x, y
00030 DISP x, y, x**y
00040 GOTO 10
00050 END
```

## 2.18 DISP USING

Anweisung: DISP USING

Funktion: Darstellung von Zahlen und Strings auf dem Display. Die Formatspezifikation ist vom Benutzer frei wählbar.

Format: `DISP USING Format, Ausdruck [,Ausdruck]`
*Format* Zeilennr einer IMAGE - Anweisung oder Stringausdrucks.

Wirkung: Die Ergebnisse der Ausdrücke werden der Reihe nach in dem durch die Formatanweisung spezifizierten oder durch die Stringvariable festgelegten Format auf dem Display ausgegeben.
Jede auf dem Display dargestellte Größe wird von links nach rechts, gemäß dem jeweiligen Abbildungszeichen des Formatfeldes, ausgegeben.

Bemerkung:
- Sollen mehr Größen ausgegeben werden als durch die Formatspezifikation angegeben wird, wird wieder am Anfang der Spezifikation angefangen. Sind es weniger, so haben die restlichen Formatelemente keine Wirkung.
- Die Ausgabeelemente der DISP USING - Anweisung müssen dem spezifizierten Format entsprechen.
- Die Stringvariable muß einen Wert haben, bzw. die angegebene Zeilennummer muß vor der Anweisung liegen.

Beispiel:
```
00010 REM Beispiel 'DISP USING'
00020 REM
00030 DCL S (i,j)
00040 LET a$="i=####.###  j=####.###"
00050 DISP "i,j";
00060 INPUT i,j EOF 100
00070 DISP USING a$, i, j
00080 DELAY 20
```

```
00090 GOTO 50
00100 END
```

## 2.19 END

Anweisung: END

Funktion: Die Anweisung gibt das Ende eines Programms an.

Format: `END`

Wirkung: Die Programmausführung wird beendet. Die Werte von Variablen sind nicht mehr definiert und alle Files werden geschlossen.

Bemerkung: Die END - Anweisung muß am Ende jedes Programms stehen.

Beispiel:

```
00500 REM ***  Die letzte Anweisung     ***
00510 REM ***  in einem BASIC-Programm  ***
00520 REM ***     ist immer:            ***
00530
00540 END
```

## 2.20 FILES

Anweisung: FILES

Funktion: Festlegung der Anzahl und gegebenenfalls der Namen der Files, auf die ein Programm zugreift.

Format: `FILES filename [;filename]`

Wirkung: Alle in der Anweisung mit Namen angegebenen Files werden geöffnet.
Jedem Filenamen wird eine Zahl (*Filedesignator*) zugewiesen, die der Reihenfolge in der Liste entspricht. Der erste Filename erhält die Zahl 1, der zweite die Zahl 2 usw. Wird anstelle eines Filenamens * gesetzt, werden Platz und Filedesignator für ein File reserviert, das in einer nachfolgenden FILE: - Anweisung geöffnet wird.

Bemerkung:
- Bei jeder sich auf ein File beziehenden Operation muß als erster Operand der Filedesignator angegeben werden.
- Jedes File kann mit einer FILE: – Anweisung durch ein anderes File ersetzt werden (siehe Anweisung FILE:).

Beispiel:

```
00010 DCL SINGLE
00020 REM Beispiel fuer 'FILES' und 'FILE:'
00030 REM
00040 REM *** Oeffnen von 5 Files ***
00050 REM
00060 FILES "SDAT"; "TDAT"; "UDAT"; "VDAT"; "WDAT"
00070 REM
00080 REM *** Oeffnen fuer Schreiben ***
00090 REM
00100 FOR i=1 TO 5
00110 SCRATCH: i
00120 READ a$ (i)
00130 REM *** Beschreiben der Files ***
00140 REM
00150 WRITE: i, "Das ist File "+a$(i)
00160 REM
00170 REM *** Schliessen der Files ***
00180 FILE: i,*
00190 NEXT i
00200 DATA "SDAT", "TDAT", "UDAT", "VDAT", "WDAT"
00210 DISP "Welches File";
00220 INPUT i
00230 IF i < 0 OR i > 5 THEN 320
00240 REM *** Oeffnen des angegebenen Files ***
00250 REM
00260 FILE: i,a$(i)
00270 REM
00280 REM *** Lesen vom File ***
00290 REM
00300 READ: i,b$
00310 DISP "File";i;": ";b$
00320 GOTO 210
00330 END
```

## 2.21 FILE:

Anweisung: FILE:

Funktion: Zugriff auf ein File, dessen Name nicht in einer FILES – Anweisung spezifiziert ist und Schliessen von Files vor Programmende.

Format: `FILE: arithm. Ausdruck, Stringausdr.`

Der numerische Ausdruck wird berechnet und das Ergebnis gerundet. Die so erhaltenen ganze Zahl n ist der Filedesignator des Filenamens in der Anweisung.

Wirkung: Der Ergebnisstring des Stringausdrucks muß ein Filename sein. Das so durch seinen Namen angegebene File ersetzt das File, das bisher durch den Filedesignator n bezeichnet wurde.
Das letztere File wird geschlossen und an seiner Stelle das File mit dem angegebenen Filenamen unter dem gleichen Filedesignator geöffnet.
Wird anstelle des Filenamens (*) angegeben, so wird das File mit dem Filedesignator n geschlossen, ohne daß ein anderes File geöffnet wird.

Bemerkung:
- Der Filedesignator n muß größer sein als 0 und darf höchstens gleich der Anzahl Files in der FILES – Anweisung sein.

Beispiel: Siehe Anweisung FILES.

## 2.22 FNEND

Anweisung: FNEND (function end)

Funktion: Kennzeichnung des Endes einer mehrzeiligen Funktionsdefinition.

Format: `FNEND`

Wirkung: siehe DEF/FNEND

Bemerkung: Jede mehrzeilige Funktionsdefinition muß mit der Anweisung FNEND enden.

## 2.23 FOR

Anweisung: FOR

Funktion: Kennzeichnung des Beginnes einer Schleife.

Format:

```
FOR Laufvar. = begin TO end [STEP step]
          .
          .
          .
     beliebige BASIC-Instruktionen
          .
          .
          .
NEXT  Laufvariable
```

*Laufvar.* einfache numerische Variable, die bei jedem Durchlauf um die Schrittweite erhöht wird.

*begin* numerischer Ausdruck, der den Anfangswert der Schleife bildet.

*end* numerischer Ausdruck, der den Endwert der Schleife bezeichnet.

*step* numerischer Ausdruck, der die Schrittweite bezeichnet.

Wirkung: Die Folge von Anweisungen zwischen FOR und NEXT wird solange ausgeführt, bis der Wert der Laufvariablen den angegebenen Endwert übersteigt. Ist der Anfangswert der Laufvariablen größer (kleiner bei Angabe einer negativen Schrittweite) als der Endwert, so wird die Schleife nicht ausgeführt; der Wert der Laufvariablen bleibt unverändert und das Programm wird mit dem ersten Befehl nach NEXT fortgesetzt. Ist der Anfangswert kleiner (bei negativer Schrittweite größer) als der Endwert, so wird die Schleife durchlaufen. Bei jeder NEXT – Anweisung wird der Wert der Laufvariablen um die Schrittweite erhöht. Ist der neue Wert der Laufvariablen kleiner oder gleich (bei negativer Schrittweite größer oder

gleich) dem Endwert, so wird die Schleife von neuem durchlaufen, und zwar solange, wie der Wert der Laufvariablen den Endwert nicht überschreitet (bzw. unterschreitet bei negativer Schrittweite). Nach Beendigung der Schleife fährt das Programm mit dem auf NEXT folgenden Programmschritt fort.

Bemerkung:

- Fehlt die Angabe der Schrittweite, wird diese implizit als 1 angenommen. Zwei oder mehrere Schleifen können geschachtelt werden. Sie dürfen sich jedoch nicht überschneiden.

| Richtig: | Falsch: |
|---|---|
| FOR a = 1 TO 10 | FOR a = 1 TO 10 |
| FOR b = 1 TO 5 | FOR b = 1 TO 5 |
| NEXT b | NEXT a |
| NEXT a | NEXT b |

- Durch Schrittweite Null wird eine Endlosschleife gebildet, wenn der Wert der Laufvariablen in der Schleife nicht verändert wird.

- Eine FOR/NEXT – Schleife kann durch Sprunganweisungen (GOTO, ON...GOTO, IF...THEN) vorzeitig beendet werden, wenn das Sprungziel außerhalb der Schleife liegt.

- Bei einem Sprung aus der Schleife behält die Laufvariable ihren letzten Wert bei. Von außerhalb darf jedoch nicht in die Schleife gesprungen werden.

- Jeder Anweisung FOR muß ein NEXT entsprechen. Sind mehrere Schleifen geschachtelt, müssen sie verschiedene Laufvariablen aufweisen.

Beispiel:

```
00010 REM Beispiel 'FOR/NEXT' *
00020 REM .
00030 DISP "1. Schleife: Anfangswert, Endwert, ";
00035 DISP "Schrittweite";
00040 INPUT a1, a2, a3 EOF 250
00050 DISP "2. Schleife: Anfangswert, Endwert, ";
```

```
00055 DISP "Schrittweite";
00060 INPUT b1, b2, b3 EOF 250
00070 DISP "Anfangswert", "Endwert", "Schrittweite"
00080 DISP
00090 DISP "i-Schleife", a1, a2, a3
00100 DISP "j-Schleife", b1, b2, b3
00110 DISP
00120 FOR i=a1 TO a2 STEP a3
00130 DISP "i-Schleife: i= ";i, "j-Schleife: ";
00140 FOR j=b1 TO b2 STEP b3
00150 DISP " j=";j;
00160 NEXT j
00170 DISP
00180 NEXT i
00190 DISP
00200 DISP "i=";i;TAB(15);"(letzter Wert (";i-a3;") + Schrittweite (";a3; "))"
00210 DISP "j=";j;TAB(15);"(letzter Wert (";j-b3;") + Schrittweite (";b3; "))"
00220 DISP
00230 DISP
00240 GOTO 30
00250 END
```

## 2.24 GOSUB

Anweisung: GOSUB

Funktion: Bewirkt den Sprung zu einer bestimmten Anweisung, bei der ein Unterprogramm beginnt.

Format: `GOSUB Zeilennummer`

Wirkung: Das Programm fährt bei der durch die Zeilennummer definierten Anweisung fort. Die zuletzt ausgeführte Anweisung des Unterprogramms muß RETURN sein, damit das Programm in die Zeile nach GOSUB (Zeile mit der nächsthöheren Zeilennummer) zurückspringt.

Bemerkung:
- In einem Unterprogramm können auch mehrere RETURN – Anweisungen vorkommen. Ein Unterprogramm kann auch GOSUB – Anweisungen enthalten. Durch RETURN wird jedesmal in die Zeile nach dem letzten GOSUB (Zeile mit der nächsthöheren Zeilennummer) gesprungen.

- Der rekursive Aufruf von Unterprogrammen ist möglich. Ein Unterprogramm darf nur mit RETURN verlassen werden.

Beispiel:

```
00010 REM *Beispiel für rekursiven Unterprogrammaufruf
00020 DCL SINGLE
00030 LET a=0
00040 DISP "Das Unterprogramm ruft sich selbst auf:"
00050 DISP
00060 GOSUB 100
00070 DISP
00080 DISP "Ende der Unterprogrammaufrufe"
00090 GOTO 170
00100 LET a=a+1
00110 IF a>5 THEN 160
00120 DISP "Ausführung ";a;" des Unterprogramms"
00130 GOSUB 100
00140 LET a=a-1
00150 DISP "Rücksprung von RETURN zu GOSUB ";a
00160 RETURN
00170 END
```

## 2.25 GOTO

Anweisung: GOTO

Funktion: Sprung zu einer bestimmten Zeile des Programms.

Format: `GOTO zeilennummer`

Wirkung: Die Verarbeitung des Programms wird mit der in GOTO bezeichneten Zeile fortgesetzt.

Bemerkung:
- Sprünge in mehrzeilige Funktionen und FOR/NEXT – Schleifen sind nicht erlaubt.
- Bei GOTO innerhalb einer Schleife muß das Sprungziel immer zwischen FOR und NEXT liegen, falls die Schleife nicht vorzeitig beendet werden soll.

Beispiel:

```
00010 REM  Programmbeispiel 'GOTO'
00020 REM
00030 REM  Beispiel fuer eine Endlosschleife
00040 REM
00050 DISP "Das ist eine Endlosschleife ..........."
00060 DISP "Das Programm kann nur mit der Taste 'SV'";
00070 DISP "abgebrochen werden"
00080 DISP
00090 GOTO 50
00100 END
```

## 2.26 IF ... THEN

Anweisung: IF ... THEN

Funktion: Bedingte Verzweigung in einem Programm.

Format: `IF (Vergleich) THEN zeilennummer`

Wirkung: Die Ausdrücke werden berechnet, die Ergebnisse miteinander verglichen und ihr Wahrheitswert gebildet. Liefert der gesamte Ausdruck den Wahrheitswert ***wahr*** , wird zur angegebenen Zeilennummer gesprungen. Liefert der Ausdruck dagegen den Wahrheitswert *falsch*, wird das Programm in der nächsten Zeile fortgesetzt.

Bemerkung: Befindet sich die Anweisung IF...THEN in einer mehrzeiligen Funktion, muß das Sprungziel der Verzweigung innerhalb der Funktion liegen. Dasselbe gilt auch, wenn die Anweisung in einer Schleife vorkommt und diese nicht aufgrund der Bedingung vorzeitig verlassen werden soll.

## 2.27 IMAGE

Anweisung: : (image)

Formatspezifikationen

Funktion: Diese Spezifikationen bestimmen das Format, in welchem die Ausgabegrößen (Zahlen und Strings) dargestellt werden (siehe PRINT USING, DISP USING und BUILD USING).

Format: : "(Text/Imagefield) [(text/imagefield)...]"
Kennzeichen für Formatspezifikation.

Wirkung: Beziehen sich die Operanden der Anweisung PRINT USING, MAT PRINT USING, DISP USING oder BUILD USING auf eine Zeile mit einer Formatspezifikation, werden die Ausgabegrößen für den Drukker, das Display bzw. die Stringvariable dem Format entsprechend generiert. Die Ausgabe der Werte erfolgt gemäß der Beschreibung im Abschnitt *Ausgabe von Werten bei PRINT USING, DISP USING, BUILD USING.*

Formatfelder (image fields)
Die Formatfelder einer Formatspezifikation müssen für numerische Werte numerisch und für Stringausdrücke Stringformatfelder sein.
Die Formatfelder für die Zahlenausgabe können sein:
- Felder für ganzzahlige Größen
- Felder für Dezimalzahlen
- Felder für Zahlen in Exponentialdarstellung
- Felder mit dem Zeichen $ als Symbol vor der Zahl

Formatfeld für ganzzahlige Größen: [#]...[#]
Es besteht aus einer Folge von für Ziffern stehenden Symbolen.
Grösse: Minimum 2, Maximum 20 Zeichen.

Formatfeld für Dezimalzahlen: ##.#...[#]
Es besteht aus einer Folge von für Dezimalzahlen stehenden Symbolen.
Grösse: Minimum 3, Maximum 26 Zeichen inkl. Dezimalpunkt.

Formatfeld für Exponentialdarstellung:
Für die Mantisse gelten die Regeln für Dezimalzahlen, zusätzlich ist für das Exponentenfeld ^^^^ einzugeben.

Grösse: Minimum 7 (3# - Zeichen,^^^^),
Maximum 30 Zeichen
Bemerkung: Das Symbol ^^^^ ist immer das letzte Zeichen des Formatfeldes.

Formatfeld mit $ als Symbol:
$[$] ... [$] Formatfeld für Ganzzahlgrößen
Formatfeld für Dezimalgrößen
Es besteht aus einer Folge von $ - Zeichen.
Grösse: Minimum 2, Maximum 20 Zeichen.

Formatfeld für Stringausdrücke:
'L L ... R
'R R ... R
'C C ... C
bestehen aus einem Hochkomma und eventuell einem oder mehreren Buchstaben L, R oder C.
Grösse: Minimum 1, Maximum 32 000 Zeichen.

Beispiel:

```
00010 REM *** Formatfelder ***
00020 REM
00030 DISP "Zur Kontrolle der Druckpositionen:"
00040 DISP
00050 DISP "123456789 123456789 123456789 123456789 123456789 123456789 123"
00060 DISP
00070 :"    #########.#^^^^,          $$$$$$$,      ####.## SFr,    ###.#%"
00080 LET a=123.4567890123
00090 LET b=1000
00100 LET c=256.12
00110 LET d=10.5
00120 DISP USING 70, a, b, c, d
00130 END
```

Ausgabe von Werten mit den Anweisungen
PRINT USING, DISP USING, BUILD USING

Die Ausgabe der Daten erfolgt nach folgenden Regeln:

1. Bei numerischen Größen in einem Formatfeld für ganzzahlige Werte entspricht jeder Ziffer ein Symbol #. Für die Ausgabe des Vorzeichen ist zusätzlich zur maximalen Stellenzahl ein weiteres Symbol vorzusehen. Die Zahl wird rechstsbündig ins Formatfeld übertragen. Ist die Zahl nicht ganzzahlig, werden die Nachkommastellen abgeschnitten. Bei einer positiven Zahl steht vor der er-

sten Ziffer eine Leerstelle, bei einer negativen Zahl ein Minuszeichen. Bei Formatüberschreitung werden anstelle der Ziffern (*) (Sternchen) ausgegeben.

2. Bei numerischen Größen in einem Formatfeld für Dezimalzahlen entspricht jeder Ziffer ein Symbol #. Sind für Nachkommastellen der Zahl nicht genügend Symbole vorgesehen, wird die Zahl gerundet und der letzte Teil rechts abgeschnitten. Bei positiven Zahlen ist das erste Zeichen eine Leerstelle, bei negativen Zahlen ein Minuszeichen. Sind für den ganzzahligen Teil der Zahl nicht genügend # – Symbole vorgesehen, wird für jedes Zeichen des Formatfeldes ein (*) ausgegeben.

3. Bei numerischen Größen in einem Formatfeld für Zahlen in Exponentialdarstellung entspricht jeder Ziffer ein # – Symbol, analog zu 2. Anstelle der Zeichen ^^^^ wird der Buchstabe E, ein Minus – oder Leerzeichen und zwei darauffolgende Ziffern gesetzt. Diese Ziffern bilden den Exponenten zur Basis 10. Das Format muß einen Dezimalpunkt enthalten.

4. Der numerische Wert wird auf ein Formatfeld mit einem $ – Zeichen als Symbol abgebildet, wobei nur ein einziges $ – Zeichen links im Formatfeld stehen muss.

Bemerkung:
- In allen Formatfeldern für die Ausgabe numerischer Werte muß eine Stelle für das Vorzeichen vorgesehen sein. Bei der Ausgabe einer positiven Zahl wird die erste Stelle des Formatfeldes (# oder $) durch ein Leerzeichen ersetzt, bei der Ausgabe einer negativen Zahl wird die erste Stelle des Formatfeldes für die Ausgabe des Minus – Zeichen verwendet.

- Zahlen, deren Absolutbetrag kleiner ist als 1, werden mit einer 0 vor dem Dezimalpunkt dargestellt.

## 2.28 INPUT

Anweisung: INPUT

Funktion: Eingabe von Werten während des Programmablaufs.

Format: `INPUT Variable [,...] [EOF Zeilennr.]`

Wirkung: Der Programmablauf wird angehalten und auf dem Display erscheint ein Fragezeichen. Der Operator kann nun die Werte, getrennt durch Kommata, eingeben. Sie werden den Variablen der INPUT - Anweisung der Reihe nach zugewiesen. Die Zuweisung erfolgt erst, wenn der Programmablauf nach dem Eintasten aller Werte durch Drücken der Taste RETURN fortgesetzt wird.

Bemerkung: Zwei Fragezeichen (??) auf dem Display geben an, daß noch nicht allen Variablen Werte zugewiesen wurden. Das System wartet auf die Eingabe der restlichen Daten. Die Eingabewerte müssen mit dem Typ der entsprechenden Variablen übereinstimmen. Die Anzahl der eingegebenen Werte darf nicht grösser sein als die Anzahl der vorhandenen Variablen. Ein String muß in Anführungszeichen (") gesetzt werden, falls er das Komma - oder das Leerzeichen enthält.
Wenn *EOF* angegeben wurde, so wartet das System nicht auf die Eingabe von fehlenden Daten, sondern springt zur angegebenen Zeilennummer.

Meldungen: ?
Das Fragezeichen gibt an, daß eine Eingabe verlangt wird. Es kann aber auch Bestandteil einer vorangehenden Display - Anweisung sein, falls diese durch das Trennzeichen ";" abgeschlossen wurde.

??
Zwei Fragezeichen zeigen an, dass noch nicht allen Variablen der Variablenliste der Anweisung INPUT ein Wert zugewiesen wurde. Die Eingabe ist fortzusetzen, bis alle Variablen einen Wert aufweisen.

`Too much input-excess ignored`
Diese Meldung gibt an, daß die Anzahl eingegebener Werte die Anzahl der Elemente der Variablenliste übersteigt. Die überzähligen Eingaben werden ignoriert.

```
Incorrect format - retype line
```

Diese Meldung erscheint, wenn numerischen Variablen alphanumerische Daten zugewiesen werden. Zahlen werden als String interpretiert und ohne Fehlermeldung übernommen, falls sie an alphanumerischen Variablen übergeben werden.

Beispiel:

```
00010 DISP "Zahl,String,Zahl";
00020 INPUT a, s$, b EOF 50
00030 DISP s$, a, b
00040 GOTO 10
00050 LET k=9
00060 LET i=9
00070 INPUT i, b(i) EOF 100
00080 DISP "Altes I=";k;" ,Neues I=";i;" ,b(";k;")=";b(k)
00090 GOTO 50
00100 END
```

## 2.29 LET

Anweisung: LET

Funktion: Zuweisung der Werte von Ausdrücken an eine Variable eines Programms.

Format: `LET Variable = Ausdruck`

Wirkung: Der Ausdruck wird berechnet und das Ergebnis der Variablen zugewiesen.

Bemerkung:

- Das Schlüsselwort LET muß nicht eingegeben werden.
- Die aktuelle Länge der Stringvariablen in einer Zuweisung ist gleich der Anzahl der Zeichen, die sich als Resultat des Stringausdrucks ergeben.

Beispiel:

```
00010 REM  Beispiel fuer LET
00020 DISP
00030 LET a$="Die Kapitel"
```

```
00040 LET b$=" 4 "
00050 LET c$=" 5"
00060 LET d$=" 6"
00070 LET e$=" 7 bis 9 "
00080 LET f$="beschreiben das System."
00090 LET g$="beschreiben die Sprache."
00100 LET h$="  "+a$+b$+","+c$+"und"+d$+f$
00110 DISP "h$=";h$
00120 LET h$=a$+e$+g$
00130 DISP "h$=";h$
00140 LET s$=""
00150 FOR i=32 TO 126
00160 LET s$=s$+chr$(i)
00170 NEXT i
00180 DISP "s$=";s$
00190 END
```

## 2.30 NEXT

Anweisung: NEXT

Funktion: Ende einer Schleife.

Format: `NEXT Laufvariable`

Wirkung: Siehe Anweisung FOR/NEXT.

## 2.31 ON ... GOSUB

Anweisung: ON ... GOSUB

Funktion: Sprung in ein Unterprogramm. Das Sprungziel ist abhängig vom Wert eines Ausdrucks.

Format: `ON num. Ausdruck GOSUB Zeilennr. [,Zeilennr....]`
*Zeilennr.* gibt das Sprungziel an.

Wirkung: Der numerische Ausdruck wird berechnet und das Ergebnis gerundet. Diese ganze Zahl gibt an, zu welcher der Zeilennummern rechts von GOSUB gesprungen werden soll. Jede Zeilennummer in der Anweisung gibt die erste Zeile des jeweiligen Unter-

programms an. Die letzte Zeile jedes Unterprogramms enthält die Anweisung RETURN, die den Rücksprung in die Zeile nach ON ... GOSUB bewirkt.

Bemerkung:

- In einem Unterprogramm können auch mehrere RETURN – Anweisungen vorkommen.

- Ergibt der numerische Ausdruck nach der Rundung eine Zahl, die kleiner als 1 oder grösser als die Anzahl der Zeilennummern in der Anweisung ON ... GOSUB ist, wird das Programm mit der nächsten Anweisung nach ON ... GOSUB fortgesetzt.

Beispiel:

```
00010 REM   GOSUB Test
00020 DCL SINGLE
00030 DISP "Bitte das Gewuenschte Unterprogramm angeben (1-5):";
00040 INPUT a EOF 600
00050 DISP "Es wurde Unterprogramm No."; a; " gewaehlt."
00060 ON a GOSUB 100, 200, 300, 400, 500
00070 GOTO 30
00080
00100 DISP "Dies ist das Unterprogramm No. 1"
00120 RETURN
00130
00200 DISP "Dies ist das Unterprogramm No. 2"
00210 RETURN
00220
00300 DISP "Dies ist das Unterprogramm No. 3"
00310 RETURN
00320
00400 DISP "Dies ist das Unterprogramm No. 4"
00410 RETURN
00420 00500 DISP "Dies ist das Unterprogramm No. 5"
00510 RETURN
00600 DISP "Ende des Programms.";
00610 END
```

## 2.32 ON ... GOTO

Anweisung: ON ... GOTO

Funktion: Sprung in eine bestimmte Programmzeile in Abhängigkeit vom Wert eines numerischen Ausdruckes.

Format: `ON num. Ausdruck GOTO Zeilennr. [, Zeilennr....]`
*Zeilennr.* gibt das Sprungziel an.

Wirkung: Der numerische Ausdruck wird berechnet und das Ergebnis auf die nächste ganze Zahl gerundet. Diese Zahl gibt an, zu welcher Zeilennummer rechts von GOTO gesprungen werden soll.

Bemerkung: Ergibt der Ausdruck nach der Rundung eine Zahl, die kleiner als 1 oder grösser als die Anzahl der Zeilennummern in der Anweisung ON ... GOTO ist, wird kein Sprung ausgeführt, sondern das Programm mit der nächsten Anweisung fortgesetzt.

Beispiel:

```
00010 REM  ON...GOTO
00020 REM
00030 FOR i=-1 TO 5
00040 ON i GOTO 70, 90, 110, 130
00050 DISP "kein gueltiges i"
00060 GOTO 140
00070 DISP "i=1"
00080 GOTO 140
00090 DISP "i=2"
00100 GOTO 140
00110 DISP "i=3"
00120 GOTO 140
00130 DISP "i=4"
00140 NEXT i
00150 END
```

## 2.33 PAD

Anweisung: PAD

Funktion: Auffüllen einer Stringvariablen mit Füllzeichen.

Format: `PAD Stringvariable, delimiter d`

Wirkung: Die *Stringvariable* wird mit dem Zeichen *delimiter d* bis zur deklarierten Länge aufgefüllt.

## 2.34 PRINT

Anweisung: PRINT

Funktion: Ausgabe von Zahlen und Strings im Standardformat.

Format: `PRINT Ausdruck/TAB (num. Ausdruck) [,/; ...]`
, / ; : Trennzeichen mit einer bestimmten Bedeutung für die Ausgabe (siehe Kapitel 8.5).

Wirkung: Die Ergebnisse der Ausdrücke werden berechnet und im Standardformat von links nach rechts der Reihe nach gedruckt. Die Position der Zeichen in der Druckzeile kann mit den Anweisungselementen
- TAB (num. Ausdruck)
- "," Komma
- ";" Semikolon

beliebig festgelegt werden.
PRINT ohne Angabe von Ausgabeelementen bewirkt die Ausgabe einer Leerzeile.

Bemerkung:
- Durch Verwendung des Kommas (,) als Trennzeichen wird die Zeile in 5 Druckzonen zu je 16 Stellen aufgeteilt.
- Die Funktion TAB bewirkt die Ausgabe des nächsten Elementes ab einer bestimmten Position.
- Das Standardformat ist in Abschnitt 2.5 beschrieben
- Mit der Prozedur *printer (INT CONST channel)* kann der Ausgabekanal gesetzt werden. Kanal 0 bedeutet Ausgabe auf Terminal.

Beispiel:
```
00010 PRINT "OLIVETTI"
00020 PRINT
00030 PRINT tab (10); "M20"
00040 PRINT 1,2,3,4,5
00050 PRINT 1;2;3;4;5
00060 PRINT 1,2,3,4,"Dieser String ist zu lang"
00070 END
```

## 2.35 PRINT USING

Anweisung: PRINT USING

Funktion: Ausdruck von Daten in definiertem Format.

Format: `PRINT USING Format, Ausdr. [,Ausdr. ...]`
*Format*: Zeilennr einer IMAGE – Anweisung oder Stringausdruck

Wirkung: Die Werte der Ausdrücke werden in dem Format gedruckt, welches in der Formatspezifikation angegeben ist. Jeder Wert entspricht einem Formatfeld, von links beginnend (siehe IMAGE – Anweisung).

Bemerkung:
- Jede Anweisung PRINT USING bewirkt, daß die Werte in einer neuen Zeile gedruckt werden, auch wenn die vorhergehende Anweisung PRINT mit ',' oder ';' endet. Ist die zu druckende Anzahl von Werten grösser als die im Formatstring vorgesehene, werden die restlichen Werte in der nächsten Zeile im gleichen Format ausgegeben. Sind weniger Ausgabeelemente als Formatelemente vorhanden, werden anstelle der restlichen Formatelemente Leerzeichen gesetzt. Die Ausgabeelemente und Formatfelder müssen in ihrem Typ übereinstimmen.

- Mit der Prozedur *printer (INT CONST channel)* kann der Ausgabekanal gesetzt werden. Kanal 0 bedeutet Ausgabe auf Terminal.

Beispiel:
```
00010 REM *** PRINT USING ***
00020 REM
00030 REM *** Numerische Felder ***
00040 :"Mit/ohne Nachkommastellen:  #####, #####.# und #####.####"
00050 :"Exponentialdarstellung:              ##.##^^^^"
00060 :"Dollar Felder:                    $$$$$.# und $####.####"
00070 DISP "Numerischer Wert";
00080 INPUT a EOF 150
00090 PRINT USING 40,a,a,a
00100 PRINT USING 50,a
```

```
00110 PRINT USING 60,a,a
00120 PRINT
00130 GOTO 70
00140 REM
00150 REM *** Alphanumerische Felder ***
00160 : "Rechtsbuendige Ausgabe: 'RRRRRRRRRRRRRRRRRR"
00170 : "Linksbuendige Ausgabe:  'LLLLLLLLLLLLLLLLLL"
00180 : "Zentrierte Ausgabe:     'CCCCCCCCCCCCCCCCCC"
00190 REM
00200 DISP "String";
00210 RKB b$ EOF 300
00220 PRINT "String: ";b$
00230 PRINT
00240 PRINT USING 160, b$
00250 PRINT USING 170, b$
00260 PRINT USING 180, b$
00270 PRINT
00280 PRINT
00290 GOTO 200
00300 END
```

## 2.36 RANDOMIZE

Anweisung: RANDOMIZE

Funktion: Bei Aufruf der Anweisung RANDOMIZE ohne Parameter werden unterschiedliche Zufallszahlenfolgen erzeugt. Bei Aufruf mit jeweils demselben Parameter werden gleiche Standardfolgen erzeugt.

Format: `RANDOMIZE [num. Ausdruck]`

Bemerkung: Nach dem Beginn einer Task wird beim Erzeugen von Zufallszahlen jeweils mit der gleichen Basiszahl begonnen.

Beispiel:

```
00010 REM  Erzeugung von Zufallszahlen
00020 REM
00030 REM RANDOMIZE mit Parameter
00040 RANDOMIZE 1
00050 FOR i = 1 TO 20
00060 DISP rnd,
00070 NEXT i
```

```
00080 DISP
00090 REM RANDOMIZE ohne Parameter
00100 RANDOMIZE
00110 FOR i = 1 TO 20
00120 DISP rnd,
00130 NEXT i
00140 END
```

## 2.37 READ

Anweisung: READ

Funktion: Zuweisung von Werten aus dem internen File an Variable. Die Werte werden in einer internen Tabelle mit Hilfe von DATA - Anweisungen generiert (siehe Anweisung DATA).

Format: `READ Variable [, Variable ...]`

Wirkung: Die Werte der internen Tabelle werden der Reihe nach den Variablen in der READ - Anweisung zugewiesen. Mit dem Lesen der Tabelle wird an der durch den Pointer bezeichneten Stelle begonnen (siehe Anweisung DATA).
Der Pointer kann mit der Anweisung RESTORE auf das erste Element der Tabelle zurückgesetzt werde.

Bemerkung:
- Die Werte werden den Indizes von Feldern erst beim Aufruf innerhalb einer READ - Anweisung zugewiesen; somit kann eine Variable der READ - Anweisung als Index eines Feldelementes der gleichen READ - Anweisung verwendet werden. Die Werte müssen im Typ mit der Variablen übereinstimmen.

- READ kann nur dann verwendet werden, wenn im Programm mindestens eine DATA - Anweisung vorkommt. In einer READ - Anweisung dürfen nur so viele Werte verlangt werden, wie in der internen Tabelle noch vorhanden sind.

Beispiel:

```
00010 REM  READ
00020 REM
00030 DATA 1, 2.54
00040 READ i, a(i)
00050 DISP "i="; i; "  a(i)="; a(i)
00060 END
```

## 2.38 READ

Anweisung: READ:

Funktion: Zuweisung von Daten aus einem externen File an die Variablen des Programms.

Format: `READ: filedesignator, Variable [,...] [EOF Zeilennr.]`
*filedesignator* numerischer Ausdruck

Wirkung: Der Wert des numerischen Ausdrucks wird berechnet und auf die nächste ganze Zahl n gerundet. Diese Zahl n bezeichnet das File in der FILES – Anweisung, dessen Elemente gelesen werden sollen. Der Lesevorgang beginnt bei dem durch den Pointer des Files bezeichneten Element. Nach dem Lesen weist der Pointer auf das nächste Element des Datenfiles. Wird beim Lesen das Ende des Files überschritten, erfolgt eine Fehlermeldung. Ist jedoch der Parameter *EOF Zeilennr.* angegeben, wird das Programm bei Erreichen des File – Endes bei der Zeile *Zeilennr.* fortgesetzt, ohne daß eine Fehlermeldung erfolgt.

Bemerkung:

- Textfiles können mit READ: wie sequentielle Datenfiles gelesen werden. Jede Textzeile entspricht einem String, in dem die *Zeilennr.* nicht enthalten ist.

- Der Filedesignator muß grösser sein als Null und kleiner oder gleich der Anzahl der Filenamen in der FILES – Anweisung.

- In einer READ: - Anweisung kann eine Variable als Index eines nachfolgenden Feldelementes derselben READ:-Anweisung verwendet werden.

Beispiel:

```
00010 REM *** READ: mit EOF-Anweisung ***
00020 REM
00030 FILES "SFILE"
00040 DCL S i
00050 RESTORE: 1
00060 FOR i=1 TO 100
00070 READ: 1,a,a$ EOF 100
00080 DISP a$;a,
00090 NEXT i
00100 DISP "File-Ende nach"; i-1; " Wertepaaren erreicht"
00110 END
```

## 2.39 REM

Anweisung: REM (remark)

Funktion: Einschub von erläuternden Texten in ein Programm.

Format: `REM Kommentar`
*Kommentar:* beliebiger Text.

Wirkung: REM ist eine nicht ausführbare Anweisung. Der Kommentar erscheint im Ausdruck (Listing), ist jedoch für die Programmausführung bedeutungslos.

Beispiel:

```
00010 REM*** Dies ist ein Beispiel ***
00020 REM***   fuer die Anweisung  ***
00030 REM
00040 REM          REMARK
00050 REM
00060 REM
00070 END
```

## 2.40 RESTORE

Anweisung: RESTORE

Funktion: Setzen des Pointers des internen Files auf das erste Element des Files. (d.h. auf das erste Element der DATA – Anweisung mit der niedrigsten Zeilennummer).

Format: RESTORE

Wirkung: Der Pointer des internen Files wird auf die erste Stelle gesetzt.

Bemerkung: Siehe Anweisung DATA und READ.

Beispiel:

```
00010 REM   RESTORE
00020 DCL SINGLE
00030 DATA 1,2,3,4,5,6,7,8,9
00040 FOR i=1 TO 5
00050 READ a
00060 DISP a,
00070 NEXT i
00080 READ a,b,c,d
00090 DISP d, c, b, a
00100 RESTORE
00110 FOR i=1 TO 9
00120 READ x
00130 DISP x; "  ";
00140 NEXT i
00150 END
```

## 2.41 RESTORE:

Anweisung: RESTORE:

Funktion: Positionieren des Pointers eines Text – oder sequentiellen Datenfiles auf das erste Element im File und Setzen des Files in den Lese – Modus.

Format: RESTORE: filedesignator
*filedesignator:* numerischer Ausdruck

Wirkung: Der Wert des numerischen Ausdrucks wird berechnet und auf die nächste ganze Zahl n gerundet.
Der Pointer des durch den Filedesignator n bestimmten Files wird auf die erste Stelle gesetzt.

Bemerkung: Der Filedesignator muß eine Zahl sein, die grösser als Null und kleiner oder gleich der Anzahl der Filenamen in der FILES – Anweisung ist.

## 2.42 RETURN

Anweisung: RETURN

Funktion: *RETURN* bildet das logische Ende eines Unterprogramms und bewirkt den Rücksprung in die unmittelbar auf das aufrufende GOSUB oder ON...GOSUB folgende Programmzeile.

Format: RETURN

Wirkung: Siehe Anweisungen GOSUB und ON...GOSUB

## 2.43 RKB

Anweisung: RKB (read keybord)

Funktion: Eingabe beliebiger Zeichen *(einschliesslich aller Sonderzeichen, wie Komma (,) und Anführungszeichen (")*) über die Tastatur und ihre Zuweisung an eine Stringvariable.

Format: RKB Stringvariable
*Stringvariable* : einfache oder indizierte Variable.

Wirkung: Der Programmablauf wird unterbrochen. Im Display erscheint ein Fragezeichen (?). Die angegebene Zeichenfolge wird der Stringvariablen zugewiesen.

Beispiel:

```
00010 REM ** Beispiel RKB **
00020 REM
00030 INPUT a$, b$
00040 INPUT c$
00050 DISP
00060 DISP a$, b$, c$
00070 RKB a$
00080 RKB b$
00090 RKB c$
00100 DISP a$, b$, c$
00110 END
```

## 2.44 SCRATCH:

Anweisung: SCRATCH:

Funktion: Vorbereiten für das Schreiben auf ein Datenfile ab dem ersten Element.

Format: `SCRATCH: filedesignator`

Wirkung: Der Wert des numerischen Ausdrucks wird berechnet und auf die nächste ganze Zahl n gerundet.
Der Pointer des Files mit dem Filedesignator n wird an den Anfang des Files gesetzt. Mit WRITE: können nun die Daten sequentiell auf das externe File geschrieben werden.

Bemerkung: Der Filedesignator muss grösser als Null und kleiner oder gleich der Anzahl der Filenamen in der FILES – Anweisung sein.

Beispiel:

```
00010 FILES "SFILE"
00020 SCRATCH: 1
00030 INPUT a
00040 WRITE: 1, a
    .
```

## 2.45 SETW:

Anweisung: SETW: (set word)

Funktion: Setzen des Pointers eines Files auf eine beliebige Zeile im File.

Format: `SETW: filedesignator TO zeilennr`
*zeilennr:* numerischen Ausdruck.

Wirkung: Die Ergebnisse der numerischen Ausdrücke werden auf die nächste ganze Zahl gerundet.
Der Pointer des Files mit dem Filedesignator n wird auf die Zeile *zeilennr* im File positioniert.

Bemerkung: Nach der Anweisung *SETW:* können Daten in das File geschrieben *(WRITE:)* oder Daten vom File gelesen *(READ:)* werden.

Beispiel:

```
00010 REM *** Worte 1-25 mit Zahlen 1-25 beschreiben ***
00020 REM
00030 DCL SINGLE
00040 REM
00050 FILES "RANDOM"
00060 SETW: 1 TO 1
00070 FOR i=1 TO 25
00080 WRITE: 1, i
00090 NEXT i
00100 REM
00110 REM *** Suchen durch manuelle Pointerwahl ***
00120 REM
00130 DISP "Bitte die Satznr. (1-25) eingeben: ";
00140 INPUT p
00150 SETW: 1 TO p
00160 READ: 1, a
00170 DISP "Pointer auf ";p;TAB (20);"Wert=";a
00180 GOTO 140
00190 END
```

## 2.46 STOP

Anweisung: STOP

Funktion: Unterbrechung der Programmausführung

Format: STOP

Wirkung: Der Programmablauf wird unterbrochen. Auf dem Display erscheint die Meldung *ERROR : stop*.

Beispiel:

```
00010 REM ** Beispiel fuer STOP **
00020 DISP "1"
00030 DISP "2"
00040 STOP
00050 DISP "3"
00060 END
```

## 2.47 TRACE ON/OFF

Anweisung: TRACE ON/OFF

Funktion: Ausdruck der Zeilennummern während des Programmablaufs.

Format: TRACE ON/OFF

Wirkung: Die Zeilennummer der jeweils ausgeführten Anweisung wird ausgedruckt.
Ausnahmen: Nicht ausführbare Anweisungen (REM, DCL, DIM FN usw.).
Der Ausdruck der Zeilennummern erfolgt zwischen den programmgesteuerten Ausgaben.

Beispiel:

```
00010 TRACE ON
00020 REM
00030 DCL SINGLE
00040 DIM A (25)
00050 REM
00060 DEF FNA (x)
```

```
00070 LET fn*=x*x
00080 FNEND
00090 PRINT TAB (10);
00100 FOR i = 1 TO 3
00110 DISP FNA (i); " ";
00120 NEXT i
00130 DISP
00140 GOSUB 160
00150 GOTO 240
00160 DISP TAB (10);
00170 FOR k=10 TO 30 STEP 10
00180 DISP fna (k);" ";
00190 NEXT k
00200 TRACE OFF
00210 DISP
00220 DISP
00230 RETURN
00240 END
```

## 2.48 WHERE:

Anweisung: WHERE:

Funktion: Abfrage der Zeilennummer eines Files.

Format: `WHERE: filedesignator, n1`
*n1:* numerische Variable.

Wirkung: Der Variablen *n1* wird die Nummer der aktuellen Zeile des Files zugewiesen.

## 2.49 WRITE:

Anweisung: WRITE:

Funktion: Schreiben von Daten auf ein externes File.

Format: `WRITE: filedesignator, Ausdruck [,...] [EOF Zeilennr.]`
*filedesignator* numerischer Ausdruck

Wirkung: Der Wert des numerischen Ausdrucks wird berechnet und auf die nächste ganze Zahl n gerundet. Diese Zahl n bezeichnet das für die Aufzeichnung vorgesehene File in der FILES – Anweisung. Der Schreibvorgang beginnt bei dem durch den Pointer des Files bezeichneten Element. Nach dem Schreiben weist der Pointer auf das nächste Element des Datenfiles. Wird das Ende eines Files überschritten, erscheint eine Fehlermeldung. Ist jedoch der Parameter *EOF Zeilennummer* angegeben, wird das Programm beim Erreichen des File – Endes ohne Fehlermeldung bei der Zeile *Zeilennummer* fortgesetzt.

# 3. Standardfunktionen

Die Standardfunktionen liefern als Ergebnis einen numerischen oder alphanumerischen Wert. Ist das Ergebnis der Funktion numerisch, so wird von einer numerischen Funktion gesprochen. Entsprechend wird von einer alphanumerischen Funktion gesprochen, wenn das Ergebnis alphanumerisch ist.
Numerischen Funktionen können auch alphanumerische Parameter (z.B. LEN) und alphanumerische Funktionen auch numerische Parameter (z.B. CHR$) aufweisen.
Der Aufruf numerischer Standardfunktionen ist innerhalb numerischer Ausdrücke (bzw. anstelle von numerischen Ausdrücken) möglich; analog dazu ist der Aufruf alphanumerischer Funktionen innerhalb von Stringausdrücken erlaubt.
Das allgemeine Format für Funktionen lautet :

FKT (Arg1 [,Arg2]...)

*FKT:* Name der Funktion, der bei numerischen Funktionen einer Folge von 3 Buchstaben entspricht, welche die Bedeutung der Funktion beschreibt. Alphanumerische Funktionen haben Namen, deren viertes Zeichen ein *Dollarzeichen ($)*ist.

*Arg:* Die Argumente (Parameter) der Funktion können numerische und/oder alphanumerische Ausdrücke sein.

## 3.1 Trigonometrische Funktionen

Die trigonometrischen Funktionen
`SIN (num. Ausdruck)`
`COS (num. Ausdruck)`
`TAN (num. Ausdruck)`
`COT (num. Ausdruck)`
`ASN (num. Ausdruck)`
`ACS (num. Ausdruck)`
`ATN (num. Ausdruck)` liefern die Werte der entsprechenden Sinus-, Cosinus-, Tangens-, Cotangens, Arcussinus-, Arcuscosinus- und Arcustangensfunktion. Das Argument muß im Bogenmaß vorliegen; bei den Arcusfunktionen ist das Ergebnis ein Winkel im Bogenmaß.
Für die Umwandlung eines Argumentes von Altgrad in Bogenmaß bzw. von Bogenmaß in Altgrad dienen die Funktionen

```
RAD (num. Ausdruck)
DEG (num. Ausdruck)
```

## 3.2 Mathematische Standardfunktionen

```
EXP (x) Exponentialfunktion
LOG (x) natürlicher Logarithmus von x
LGT (x) Zehnerlogarithmus von x
SQR (x) Quadratwurzel von x
ABS (x) Absolutbetrag von x
ENT (x) Nächste ganze Zahl, die kleiner oder gleich dem Wert
        von x ist
SGN (x) Vorzeichen von x
```

*x* Numerischer Ausdruck

## 3.3 Numerische Funktionen ohne Argument

PI

Die Funktion *PI* liefert als Ergebnis die Zahl 3.141592... in doppelter Genauigkeit.

RND

Die Funktion *RND* (random number) liefert eine im Intervall (0,1) liegende Zufallszahl. Bei wiederholtem Aufruf wird jeweils eine neue Folge geliefert.

## 3.4 Spezielle numerische Funktionen

LEN (Stringausdruck)

Die Funktion *LEN* liefert die aktuelle Länge (Anzahl Zeichen) des als Argument angeführten Stringausdrucks.

SCN (Stringausdr.1, Stringausdr.2, num.Ausdruck1, num.Ausdruck2)

Die Funktion *SCN* ermöglicht das Aufsuchen der Position eines Teilstrings in einem String.

*Stringausdr. 2* ist ein Teilstring von *Stringausdr.1*.

*num.Ausdruck 1* gibt an, das wievielte Auftreten von *Stringausdr.2* in *Stringausdr.1* ermittelt werden soll.

*num.Ausdruck 2* gibt die Stelle in *Stringausdr.1* an, ab der mit dem Suchen begonnen werden soll.

Als Ergebnis wird die Stelle in *Stringausdr.1* geliefert, an der *Stringausdr.2* (gesucht ab der *num.Ausdruck 2* – ten Stelle) das *num.Ausdruck 1* – te Mal in *Stringausdr.1* auftritt.
Kommt *Stringausdr.2* im angegebenen Teil von *Stringausdr.1* nicht oder weniger als *num.Ausdruck 1* mal vor, so liefert die Funktion SCN als Ergebnis den Wert 0.

**TAB** (x)
Die Funktion *TAB* ermöglicht bei der Ausgabe von Werten im Standardformat mit DISP oder PRINT die Tabulation an eine bestimmte Position.

## 3.5 Alphanumerische Funktionen

**CHR$** (num.Ausdruck)
Die Funktion *CHR$* liefert als Ergebnis das dem Wert des Arguments entsprechende Zeichen aus der ASCII – Code – Tabelle.
Liegt der Wert des *num. Ausdruck* nicht zwischen 0 und 255, so wird ein Fehler gemeldet.

**EXT$** (Stringausdruck, num.Ausdruck 1, num.Ausdruck 2)
Als Ergebnis des Funktionsaufrufes wird der Teilstring vom *num.Ausdruck 1* – ten Zeichen bis zum *num.Ausdruck 2* – ten Zeichen des *Stringausdrucks* geliefert.
- Ist *num.Ausdruck 1* gleich *num.Ausdruck 2*, so wird als Ergebnis ein Zeichen geliefert.
- *num.Ausdruck 1* muss grösser als 0 und kleiner gleich *num. Ausdruck 2* sein.
- *num.Ausdruck 2* muss kleiner oder gleich der aktuellen Länge des *Stringausdrucks* sein.

**REP$**
In dem *Stringausdr.1* wird *Stringausdr.2* durch *Stringausdr.3* ersetzt.
*num.Ausdruck 1* gibt dabei an, wie oft ersetzt wird.
*num.Ausdruck 2* gibt die Position des Zeichen in *Stringausdr.1* an, ab welcher mit der Suche nach *Stringausdr.2* begonnen werden soll.
- Ist *num.Ausdruck 1* gleich 0, hat die Funktion keine Wirkung.
- Ist *num.Ausdruck 1* kleiner als 0, wrird *Stringausdr.2* bei jedem Auftreten durch *Stringausdr.3* ersetzt.

- Ist *num.Ausdruck 1* größer als 0 und *Stringausdr.2* ein Nullstring, so wird *Stringausdr.3* vor dem Zeichen *num.Ausdruck 2* in *Stringausdr.1 num.Ausdruck 1* – mal eingefügt.
- Ist *num.Ausdruck 1* kleiner Null und *Stringausdr.2* ein Nullstring, so wird ein Fehler gemeldet.

# Indexverzeichnis

*Unterschiede zwischen EUMEL - Basic und P6066 - Basic*

1. Die Zeilennummern müssen im EUMEL - Basic nicht vierstellig sein. Sie können von 1 bis 32767 reichen.

2. Programmzeilen müssen in der Datei streng aufteigend angeordnet sein.

3. Eine Programmzeile kann maximal 32000 Zeichen enthalten.

4. Bei einfacher Genauigkeit werden keine Nachkommastellen berücksichtigt. Der Bereich der einfachen Genauigkeit ist [ - 32767,32767]
   Der Bereich der doppelten Genauigkeit ist ( - 1.e127 - 1.e127) mit einer Genauigkeit von 14 Nachkommastellen

5. Der Aufruf von Assemblerroutinen ist nicht möglich, dafür können mit CALL aber ELAN - Programmtexte eingebunden werden.

6. Das Standardformat ist anders.

7. Bei der IMAGE - Anweisung muss das Format in " eingeschlossen werden.

8. Bei INPUT und RKB kann ein EOF angegeben werden.

9. PRINT geht nicht direkt auf einen Drucker, sondern auf einen beliebigen Kanal.

10. BASSIGN, BBUILD und BPAD sind gestrichen.

11. Die EUMEL - Filestruktur ist anders. Files bestehen generell aus bis zu 4000 Zeilen, in denen Texte oder die entsprechenden numerischen Werte stehen. Sie können in beliebiger Reihenfolge gelesen und beschrieben werden. Mit SETW: kann auf eine bestimmte Zeile positioniert werden. WHERE gibt die augenblickliche Zeilennummer an.

14. Die Anweisung FKEY und INTERRUPT ENABLE sind gestrichen.

13. Beim Aufruf der Funktion RND wird immer eine neue Folge generiert, mit RANDOMIZE kann aber eine Standardfolge erzeugt werden.

14. Die OPTIONEN sind nicht vorhanden.

15. Variablennamen können belibig lang sein.

16. Es können beliebig viele Namen verwendet werden.

19. Stringvariable können immer bis zu 32 000 Zeichen lang sein. Mit der Anweisung DCL werden sie lediglich auf eine bestimmte Länge vorinitialisiert, es erfolgt jedoch keine Fehlermeldung bei Überschreiten der DCL – Grenze.

20. Felder und Variablen dürfen nicht den gleichen Namen tragen.

21. Vergleiche haben keinen numerischen sondern eine boolschen Wert.
    (a > b) * 3 ist also nicht erlaubt.

Softwareklasse 2 (Anwendersoftware):

Regelmäßige Wartung

Autoren:

Heiko Indenbirken

Kontaktadresse:

HRZ Bielefeld
Postfach

4800 Bielefeld 1

Umschlaggestaltung:

Hannelotte Wecken

Druck:

Zentrale Vervielfältigungsstelle der Universität Bielefeld
im Juli 1984

## Hinweis:

*Diese Dokumentation wurde mit größtmöglicher Sorgfalt erstellt. Dennoch wird für die Korrektheit und Vollständigkeit der gemachten Angaben keine Gewähr übernommen. Bei vermuteten Fehlern der Software oder der Dokumentation bitten wir um baldige Meldung, damit eine Korrektur möglichst rasch erfolgen kann. Anregungen und Kritik sind jederzeit willkommen.*