dietz 600

C-BASIC

dietz 600

C-BASIC

Heinrich Dietz
433 Mülheim-Ruhr
Solinger Straße 9
Tel. 0208/485024
Telex 856770

DIETZ Computer SYSTEME

2–7504–00–007

Diese Dokumentation beschreibt die Programmiersprache des kommerziellen Systems DIETZ 600.

Sie ist nicht als Programmierhandbuch gedacht, sondern soll einen Überblick über die Möglichkeiten der Sprache C-BASIC geben. Dabei werden besonders Sprachelemente betont, die über die Dialogsprache BASIC hinausgehen. Vor allem ist der Komfort zur Fehlerbehandlung, zur Ein/Ausgabe und die Dateizugriffssprache zu erwähnen.

Nähere Informationen zum Timesharing-Betriebssystem sowie zur Programmierung in BASIC sind der entsprechenden Zusatzdokumentation zu entnehmen.

Inhalts-Verzeichnis

1. Allgemeines

1.1 Eigenschaften von C-BASIC

C-BASIC ist eine auf Dartmouth-BASIC aufbauende Sprache zur Formulierung der Aufgaben in der kommerziellen Datenverarbeitung.

C-BASIC unterscheidet sich von BASIC vor allem durch:

* spezielle kommerzielle Arithmetik, Integer-Arithmetik
* komfortable Zeichenketten-Verarbeitung
* format- und maskenspezifizierte Ausgabebefehle
* benutzergesteuerte Fehlerbehandlung
* problemangepaßte Strukturierung von Dateien in Sätze und Felder
* sequentieller oder direkter Zugriff über Schlüsselverzeichnisse

1.2 Time-sharing-Betriebssystem

C-BASIC läuft auf dem System DIETZ 600 und besteht aus 2 Hauptteilen:

* Sprach-Verarbeitung
* Time-sharing-Operating-System (TSOS)

TSOS erlaubt in der Grundausbaustufe bis zu 12 Teilnehmern den gleichzeitigen Zugang zum Rechner. Jeder Teilnehmer arbeitet an einer Konsole praktisch unabhängig von anderen Benutzern des Systems. Normalerweise ist für Systemeingriffe, Systemabfragen und Protokollierung eine zusätzliche System-Konsole vorzusehen. Bei kleineren Systemen können deren Funktionen jedoch von einer bestimmten Teilnehmerkonsole (Masterkonsole) übernommen werden.

TSOS hat 3 Aufgaben:

* JOB-Scheduling (JOBS):

  Allen rechenwilligen Teilnehmern wird nach einem Zeitscheibenverfahren (Time-Slicing) der Reihe nach Rechenzeit zugeordnet.

* Resource-Management (RESMA):

  Verwaltung der allen Teilnehmern gemeinsamen Systemhilfsmittel (EA-Geräte, Dateien) nach der Methode First in - First out.

* Input-Output Control-System (IOCS):

  Steuerung aller EA-Vorgänge, zum Teil unter Verwendung der SPOOLING-Technik.

TSOS ist modular aufgebaut. Nur diejenigen Teile, die in einer speziellen Konfiguration erforderlich sind, werden in das System eingebunden.

Der sprachverarbeitende Systemteil ist in 3 Hauptteile gegliedert:

* Command-Interpreter C-INT
* Statement-Interpreter S-INT
* Statement-Compiler S-COM

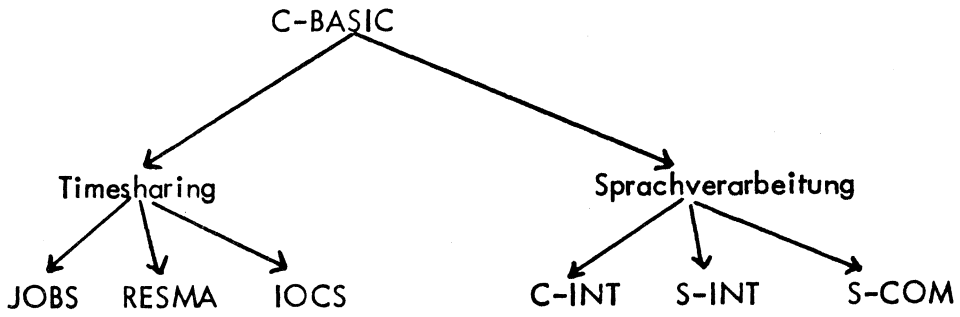

Bild 1: Hauptbestandteile des Systems

1.3 Speicher-Belegung

Das vollständige C-BASIC-System residiert auf der Platte. Neben dem speicherresidenten Kern TSOS werden die sprachverarbeitenden Programmteile nach Bedarf in einen Überlagerungs (Overlay) - Bereich geladen. Auf diese Weise sind durch das System nur etwa 18Kbyte belegt. Der Rest des Speichers wird auf die einzelnen Teilnehmer aufgeteilt. Sie werden vom System-Manager an der System - bzw. Masterkonsole bei der Anmeldung des Teilnehmers vergeben.
C-BASIC enthält eine Reihe von Dienstprogrammen zur Dateiverwaltung Datensicherung, Sortierung. Diese laufen in der Partition des aufrufenden Teilnehmers ab.

Teilnehmer-Programme, die nicht vollständig in den zugewiesenen Partitionen Platz finden, können in Segmente zerlegt und programmgesteuert in den Speicher geladen werden.

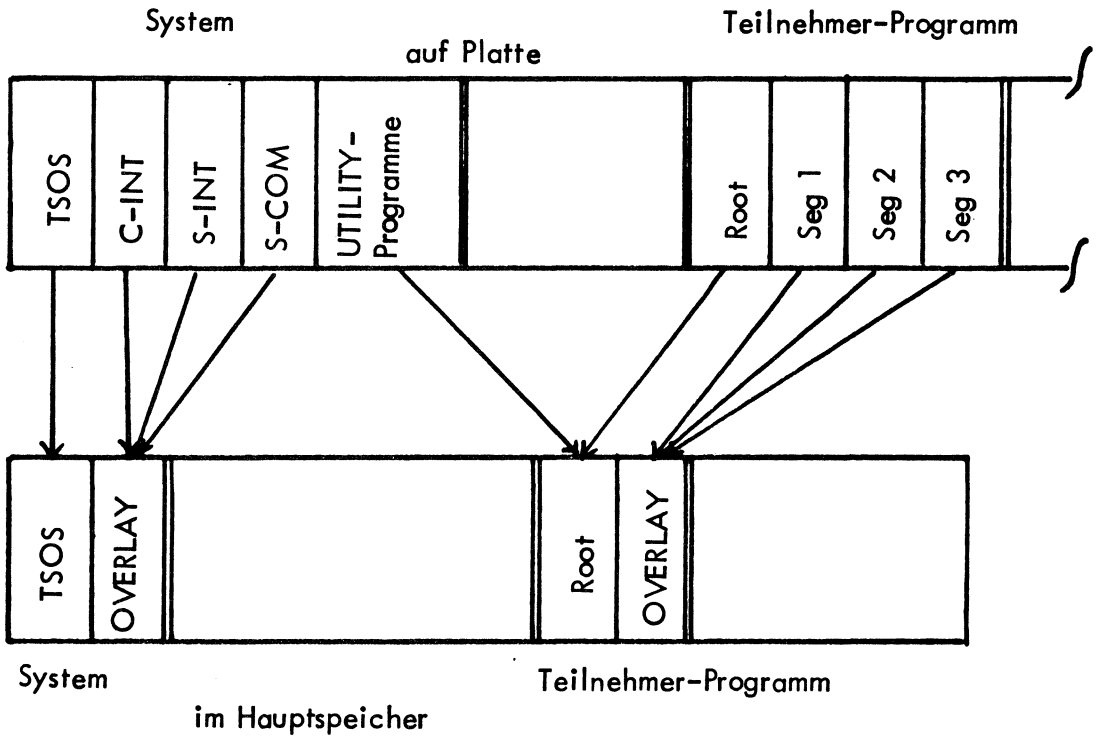

Bild 2: Speicher-Belegung

2. C-BASIC-Kommandos

Nach Inbetriebnahme des Rechners meldet sich C-BASIC auf der System/Master-Konsole mit "READY". Der System-Manager kann jetzt Teilnehmer an Konsolgeräten anmelden und für diese Speicherbedarf anfordern. Nach der Registrierung meldet sich C-BASIC auf der zugelassenen Konsole mit "READY" und reagiert nun auf Eingaben des Teilnehmers. Diese Eingaben zerfallen in zwei Gruppen, Kommandos und Statements.

Kommandos beginnen mit einem Kommandowort und kommen unmittelbar zur Durchführung. Statements beginnen mit einer Statementnummer und werden im Teilnehmerspeicher nach den Nummern geordnet. Insgesamt bilden sie ein Programm zur Lösung einer speziellen Aufgabe. Kommandos, die ein Teilnehmer eingeben kann, beziehen sich hauptsächlich auf dieses Programm. Er hat die Möglichkeit:

* Programme zu editieren
* Programme ein/auszugeben
* Programme durchzuführen
* Programme zu archivieren

Der System-Manager ist durch Organisationskommandos in der Lage, Eingriffe in den Timesharing-Ablauf des Systems vorzunehmen. Für den Teilnehmer sind diese Kommandos gesperrt. Es kann außerdem vorgesehen werden, daß die nachstehenden Kommandos ausschließlich von der Systemkonsole aus erlaubt sind, während alle übrigen Konsolen nur im Rahmen bereits erstellter Programme benutzt werden.

2.1 Time sharing-Organisation

`ACTIVATE k , s`

Damit meldet der System-Manager ein Konsolgerät k dem System an und beauftragt es, dem Teilnehmer, der an diesem Gerät arbeitet, s Sektoren (s * 128 byte) Speicherbereich zuzuweisen.

`TERMINATE k`

Der Teilnehmer am Konsolgerät k hat seine Arbeit beendet und wird abgemeldet. Der von ihm belegte Speicherbereich wird freigegeben.

`SUSPEND k`

Ein Programm oder Kommando-Auftrag am Konsolgerät k soll vorläufig zurückgestellt werden. Diese Maßnahme ermöglicht es dem System-Manager, beispielsweise ein EA-Gerät betriebsfähig zu machen (z.B. Papierwechsel). Falls die Angabe k entfällt, werden alle Aufträge suspendiert.

```
CONTINUE k
```

Die durch SUSPEND zurückgestellte Arbeit an der Konsole k wird wieder fortgesetzt. Die Angabe k kann wieder entfallen, wenn alle suspendierten Aufträge wieder zugelassen werden sollen.

```
CHANGE k1, k2
```

Der Auftrag an der Konsole k1 soll an der Konsole k2 fortgesetzt werden. Diese Maßnahme ist beispielsweise nötig, wenn ein Konsolgerät ausfällt.

```
ARM d
DISARM d
```

Mit diesen beiden Kommandos kann ein peripheres Gerät d zugelassen und wieder gesperrt werden.

```
STATUS t
```

Mit diesem Kommando ist der System-Manager in der Lage, sich über den System-Zustand (Auslastung) zu informieren und die Zeitscheibe zu verändern. "t" gibt an, welcher Teilauftrag durchzuführen ist.

t = P Ausgabe einer Liste der Speicher-Partitionen

t = D Ausgabe der Warteliste auf Pool-Geräte

t = K Ausgabe des Katalogs aller Datei-Namen, ihrer Zugriffsart und ihrer Plattenbelegung

t = T Angabe des Arbeitszustandes aller Teilnehmer

t = ddd Ändern der Zeitscheibe auf "ddd" msec.

2.2 Programm-Edition

```
SCRATCH
```

Mit diesem Kommando löscht ein Teilnehmer Programm- und Datenbereich seiner Partition und setzt das System in den Ausgangszustand für neue Programmeingabe.

LIST n1, n2

Die Statements von Nummer n1 bis n2 sollen auf dem Konsolgerät aufgelistet werden. Statt einer vollständigen Zahlenangabe sind auch Kurzschreibweisen erlaubt:

LIST	alle Statements ausgeben
LIST ,n2	alle Statements bis zur Nummer n2 ausgeben
LIST n1,	alle Statements ab Nummer n1 ausgeben
LIST n1	Statement n1 ausgeben

DELETE n1, n2

Die Statements von n1 bis n2 werden gelöscht(inklusiv). Als Abkürzungen gelten die bei LIST aufgezählten Formen.

RENUMBER n1, n2

Das gesamte Programm im Speicher erhält durch dieses Kommando neue Statementnummern, das erste Statement die Nummer n1, die folgenden n1 +n2, n1 + 2*n2, ... Speziell in der Programmerstellungsphase, wo Statements an eine Stelle eingefügt werden müssen, oder bei Übernahme fremder Programmteile, hat der Anwender damit ein nützliches Hilfsmittel. Bei der Umbenennung sind auch alle Sprungstatements mit berücksichtigt.

2.3 Ein/Ausgabe auf Poolgeräte

Poolgeräte sind allen Teilnehmern am Timesharing-Betrieb zugängliche zentrale Ein/Ausgabe-Geräte wie Lochstreifenleser, -stanzer, Schnelldrucker. Jedes Gerät ist dem System unter einer Gerätenummer "d" bekannt.

2.3.1 Spooling-Methode

Die Ein/Ausgabe bei solchen Geräten kann je nach Systemkonfigurierung direkt zwischen Gerät und Hauptspeicher oder nach der SPOOLING-Methode erfolgen.

Die Spooling-Methode erhöht den Durchsatz bei Systemen mit mehreren Teilnehmern beträchtlich. Dies betrifft jedoch weniger die Ein/Ausgabevorgänge bei Kommandos, als vielmehr solche im Programmablauf. Sollen dort Dateien auf Drucker ausgegeben werden, so ist es erforderlich, den Drucker für andere Teilnehmer zu sperren. Da jedoch Ausgaben zum Teil langdauernde Aufbereitungsvorgänge erfordern und darüberhinaus auch eine Umschaltung von Teilnehmer zu Teilnehmer stattfindet, ist der Drucker schlecht ausgenutzt.

Nach der Spooling-Methode werden deshalb alle Daten, so wie sie anfallen, auf einen speziellen Bereich der Systemplatte (SPOOL-Datei) geschrieben. Anschließend wird das Gerät vom System reserviert, falls es frei ist, und der Inhalt der SPOOL-Datei komplett ausgegeben.

Bei der Eingabe wird zunächst, falls das Gerät frei ist, die SPOOL-Datei mit den Eingabedaten belegt, ehe sie zur Weiterverarbeitung bei Bedarf der Reihe nach ausgelesen werden.

2.3.2 Anforderung/Freigabe von Poolgeräten

Die Interpretation der Lese/Schreib-Kommandos auf Poolgeräten nimmt eine automatische Reservierung der Geräte vor. Dasselbe geschieht auch bei der Durchführung von Lese/Schreib-Statements in der Spooling-Methode. Ist jedoch kein Spooling-Modul vorhanden, so muß die Anforderung und Freigabe von Geräten durch explizite Kommandos erfolgen, wenn solche in Programm-Statements angesprochen werden.

REQUEST (d)

Nach Eingabe dieses Kommandos prüft das System, ob das verlangte Gerät "d" frei ist. Falls ja, wird das Gerät belegt, falls nein, erfolgt die Meldung:

"NOT READY"

RELEASE (d)

Mit diesem Kommando wird das Gerät "d" wieder freigegeben.

2.3.3 Ein/Ausgabe

READ (d)

Wie beschrieben, enthält das Kommando READ die Funktionen von REQUEST und RELEASE. Zunächst wird geprüft, ob das Gerät frei ist. Falls nein, erfolgt die Fehlermeldung "NOT READY". Falls ja, erfolgt die Eingabe entweder direkt in den Hauptspeicher oder indirekt über die SPOOL-Datei.

Bei der Belegung der Teilnehmer-Partition wird ein bereits vorhandenes Programm jedoch nicht gelöscht, sondern das neue Programm angefügt. Soll dies vermieden werden, muß vorher SCRATCH erteilt werden.

LIST (d)

Das im Speicher stehende Programm soll auf das Gerät "d" ausgegeben werden. Die Ausgabe geschieht direkt oder über das Spooling-Modul.

Die Reservierung des Gerätes und dessen Freigabe wird implizit vorgenommen.

2.4 Programm-Durchführung

C-BASIC kennt 2 Methoden der Programm-Durchführung:

* interpretierend
* compilierend

Die "interpretierende"Methode ist adäquat in der Erstellungsphase eines Programmes. Sie unterstützt die Phase durch eine Anzahl von Testmöglichkeiten (TRACE) und gewährleistet volle Interaktivität.

Nach Erstellung eines Programmes und nach einem vollen funktionellen Test, erreicht man durch Compilieren eine optimale Durchführungsgeschwindigkeit.

Das folgende Bild soll die Arbeitsweise verdeutlichen. Man sieht daran, daß Compilieren gegenüber dem Interpretieren einen zusätzlichen Arbeitsgang erfordert.

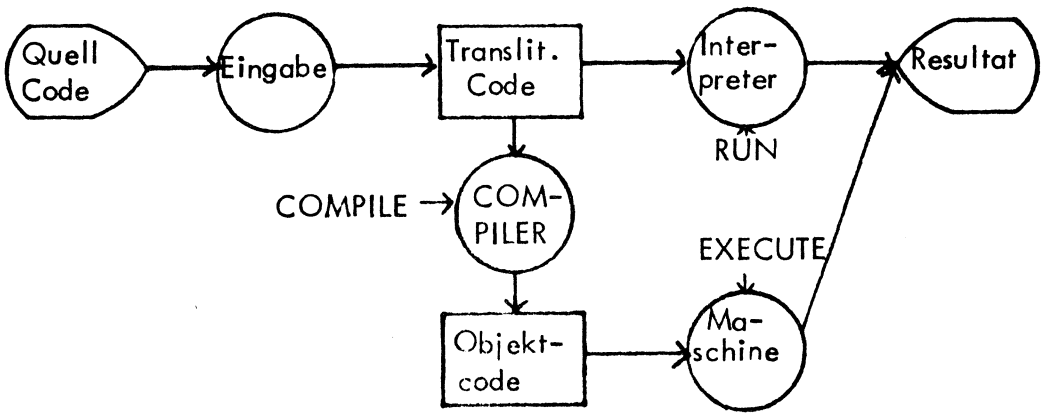

Bild 4: Interpreter-Compiler

RUN u, Name oder RUN Name

Dieses Kommando lädt von der Plattenein heit "u" (bzw. u = o) das Programm mit dem angegebenen Namen und führt es interpretierend durch.

Steht das Programm bereits im Speicher, so kann es mit dem Kommando

RUN n1, n2

direkt an Statementnummer n1 gestartet werden. Es läuft dann bis Statementnummer n2 einschließlich . Als Abkürzungen sind zulässig:

RUN	Gesamtprogramm im Speicher interpretieren
RUN n1,	Ab Statement n1 interpretieren
RUN, n2	Bis Statement n2 interpretieren
RUN n1	Statement n1 interpretieren

TRACE n1, n2 (Variable)

Mit diesem Kommando wird das Programm w ie bei RUN interpretierend abgearbeitet. Bei jeder Wertzuweisung zu der in Klammern angegebenen Variablen erfolgt zusätzlich die Ausgabe:

n : Variablenname = Wert, wobei n die Statementnummer ist, wo die Zuweisung des ausgedruckten Wertes zu der Variablen (LET-Statement, FOR-Statement, READ-Statement) erfolgte.

Falls die Variablenangabe entfällt, gibt dasSystem bei berechneten und bedingten Verzweigungen aus:

FROM n1 TO n2, wobei n1 die Statementnummer angibt, bei welcher eine Verzweigung nach Statement n2 erfolgte.

COMPILE * u1, Name 1, u2, Name 2

Das Programm mit dem Namen "Name 1" auf der Platteneinheit "u1" wird compiliert und unter dem Namen "Name 2" auf Einheit "u2" abgelegt. Falls die Angaben u1, u2 nicht vorhanden sind, ist Einheit o angesprochen.

Die Angabe " *" ist eine Option. I t sie vorhanden, so entfallen im Objekt-code alle Fehlerprüfungen (wie auf Bereichsüberschreitungen, Schleifen-Inkrement = o usw.).

In einem voll ausgetesteten Programm kann man dadurch noch höhere Effizienz bezüglich Laufzeit und Speicherbelegung erreichen.

EXECUTE u, Name

Das angegebene compilierte Programm wird von Einheit "u" geladen und gestartet. Nach Durchführung des Programms ist das System wieder im Kommando-Betrieb. Die Angabe "u" kann wieder entfallen.

2.5 Kommandos zur Datei-Verwaltung

Dateien sind Bereiche auf dem Plattenspeicher, in denen die Teilnehmer Programme oder Daten archiviert haben. Im Teilnehmer-Betrieb ergibt sich das Problem des Datei-Schutzes. Der die Datei eröffnende Teilnehmer kann ihr folgenden Status geben:

- privat : Zugriff nur dem Autor erlaubt.
- öffentlich : Zugriff allen Teilnehmern erlaubt.
- halb-öffentlich: Autor darf lesen und ändern, alle anderen Teilnehmer dürfen nur lesen.

Um Dateien vor nicht autorisiertem Zugriff schützen zu können, müssen ihre Namen im Datei-Verzeichnis mit einem Code versehen sein.

CODE ccc

Mit diesem Kommando teilt ein Teilnehmer dem System ein Codewort ccc mit.

Alle vom Teilnehmer ab jetzt eröffneten Dateien (Programme- oder Datenbereiche) werden im Dateiverzeichnis mit diesem Wort versehen und als privat erklärt. Will der Teilnehmer auf private oder bei Änderungen auf halb-öffentliche Dateien zugreifen, vergleicht das System deren Codewort mit dem des Teilnehmers.

Ein spezielles Code-Wort erlaubt dem System-Manager den Zugriff zu allen Dateien.

SAVE u, Name

Ein im Speicher stehendes Programm wird auf Einheit "u" unter dem angegebenen Namen archiviert und als privat erklärt.

LOAD u, Name

Ein Programm wird aus dem Archiv in den Speicher geladen. Ein dort bereits vorhandenes Programm wird jedoch nicht gelöscht, sondern durch das neue ergänzt. Es tritt dabei ein Mischeffekt auf. Soll dies vermieden werden, ist vor diesem Kommando mit SCRATCH der Speicher zu löschen.

KILL u, Name

Ein Programm soll gelöscht werden. Nur durch Code-Wort autorisierten Teilnehmern ist dies erlaubt.

ALTER u, Name 1, s, Name 2

Dieses Kommando erlaubt dem autorisierten Teilnehmer eine Änderung im Dateienverzeichnis. Die Datei mit dem Namen "Name 1" soll den neuen Status "s" und den neuen Namen "Name 2" bekommen. Eine von beiden Angaben kann auch entfallen.

Damit hat ein Teilnehmer die Möglichkeit, eine private Datei der Öffentlichkeit voll oder nur zum Zweck des Lesens zur Verfügung zu stellen. Er kann aber auch ein von ihm abgelegtes Programm später wieder privatisieren.

SERVICE Typ

Der praktische Umgang mit Dateien verlangt eine Reihe von System-Dienstprogrammen, die über dieses Kommando angesprochen werden können. Die Angabe "Typ" spezifiziert dabei ein spezielles Programm, das von der Platte in den Speicherbereich des rufenden Teilnehmers geladen und durchgeführt wird. Die Parameter zur Durchführung eines Programmes (Dateiname, Ausgabegerät usw.) werden im Dialog spezifiziert.

L = Liste : Ausgabe des Verzeichnisses der privaten und/oder öffentlichen Dateien auf Konsole oder Drucker.

K = Key : Ausgabe der Schlüssel in den Schlüsselverzeichnissen, die zu einer Datei gehören (siehe 5)

R = Rollin/Rollout: Übertragung von Datei-Inhalten von einem Gerät auf ein anderes (Platteneinheiten, Bänder, Drucker, Lochstreifen).

O = Organisation : Umorganisation von Dateien, beispielsweise zum Zweck der "Garbage collection".

Das Kommando SERVICE erlaubt auch auf einfache Weise den Einbau neuer System-Diensprogramme, beispielsweise für das Sortieren von Dateien nach bestimmten Kriterien.

"Name"

Anstelle der Kommandofolge "SCRATCH ; LOAD, o, Name" kann ein Programm von der Platteneinheit o auch über seinen Namen direkt aufgerufen werden.

3. C-BASIC-Sprachelemente

3.1 Datentypen

C-BASIC kennt 3 Datentypen:

*	Integer	(Ganzzahl)
*	Real	(= kommerzielle Zahl)
*	String	(Zeichenkette)

Die Integerzahl wird im System durch 2 byte binär dargestellt. Der zulässige Zahlenbereich ist somit [- 32768, 32767] . Integers werden vor allem zur Indizierung sowie für Darstellung von logischen Größen verwendet.

Die kommerzielle Zahl wird im System durch 8 byte binär-codiert -dezimal (BCD) dargestellt.
Die Genauigkeit der Darstellung ist 10^{-14}, der erreichbare Zahlenbereich $10^{\pm 64}$.
Ein spezielles Statement (siehe 4. 3) erlaubt die Angabe eines Rundungsverfahrens.

Die Zeichenkette kann eine beliebige, nur durch den Speicher begrenzte Anzahl von Zeichen enthalten.

Zu allen Datentypen gibt es Konstanten, Variablen und Strukturen.
Zahlenkonstanten werden in der bekannten wissenschaftlichen Notation geschrieben. Das System erzeugt bei entsprechender Größe entweder den platzsparenden Integertyp oder die kommerzielle Zahl.

Beispiele: 1, - 1.2, 1.2 E + 3, 1.23 E - 4 usw.
(Die Angabe E bedeutet "10 hoch")

Die Stringkonstante besteht aus einer Folge im ASCII-Alphabet zugelassener Zeichen oder aus einer Folge von Hexaziffern. Erstere haben als Pre- und Suffix das Zeichen ", letztere das Zeichen %.

Beispiel: "AB - DE"
%1A2F%

Die Länge von Zeichenketten-Konstanten ist durch die Eingabe-Zeile beschränkt.

Variable werden mit einem Namen von 1 bis 4 Zeichen Länge angegeben. Zur Unterscheidung des Typs erhalten Real-Variable das Suffix @, Stringvariable das Suffix $.
Beispiele: KOM@ : Name für Real-Rariable
I : Name für Integer-Variable
KET$: Name für String-Variable

Mehrere Variablenwerte können zu einer Struktur zusammengefaßt werden. C-BASIC kennt den ein- und zweidimensionalen Zahlenbereich für "Integers" und "Reals" und einen eindimensionalen Stringbereich. Strukturen müssen im allgemeinen deklariert werden. Fehlt eine solche Deklaration, so reserviert das System automatisch Platz für 10 Zahlen bzw. 10 Stringkonstanten, wobei hier jede 2 Zeichen umfaßt. Über Indexangaben kann auf Einzelelemente eines Bereiches zugegriffen werden.

Beispiele: ARR ∂ (5, 10) : 2-dim. Bereich ARR für "Reals"
I (3) : 1-dim. Bereich I für "Integers"
KET $ (5) : 1-dim. Bereich KET $ für Strings

Die Abspeicherung eines 2-dimensionalen Bereiches geschieht zeilenweise.

3.2 Operatoren

C-BASIC kennt folgende Operatoren:

arithmetisch	: +, -, *, /, ↑, −	unit. für Integers und Reals
Vergleich	: >, >=, <, <=, ≠	für Integers, Reals, Strings
logisch	: AND, OR, NOT	für Integers
Verkettung	: &	für Strings

Operanden und Operatoren dürfen zu beliebigen Ausdrücken nach den bekannten Regeln der Mathematik in Verbindung mit Klammerpaaren zusammengesetzt werden. Eine Mischung von Integers- und Reals ist zulässig, jedoch keine Mischung von Zahlen mit String-Operanden.

Die Typen-Mischung zwischen Integers und Reals verlangt Konventionen bezüglich der Konvertierung.

- Sind beide Operanden Integer, so ist das Resultat der Operation Integer (z.B.: 10/3 = 3)
- Sind beide Operanden Real, so ist das Resultat vom selben Typ (z.B.: 10/3 = 3.3333333333333)
- Bei gemischten Operanden wird der Integer-Typ für die Berechnung in eine Real-Zahl konvertiert. Das Resultat ist vom Typ Real.
- Bei der Zuweisung eines Wertes zu einer Variablen (LET, READ, INPUT-Statement) wird, falls nötig, der zuzuweisende Wert entsprechend dem Typ der Variablen konvertiert.

Die Konversion zwischen Zahlen und Zeichenketten ist ebenfalls möglich, wird aber explizit durch Funktionen angegeben.

3.3 Funktionen

In C-BASIC sind folgende Standardfunktionen benutzbar:

* numerische Funktionen

Y = ABS (X)	Absolut-Betrag. Das Argument der Funktion kann ein beliebiger numerischer Ausdruck vom Typ "Integer" oder "Real" sein.
Y = SGN (X)	Signum. Für das Argument gilt das eben Gesagte. Resultat der Funktion ist: 1, 0, - 1 für X >0, =0, <0.

Die folgenden Funktionen sind bei Integers nicht sinnvoll.

Y = INT (X)	Ganzzahliger Teil von X : [X]
Y = FPT (X)	"Fractional Part" : X - [X]

* Stringfunktionen

Y = LEN (X$)	Längenbestimmung des Strings X$
Y = POS (X1$, X2$)	Bestimmung der Position des Strings X2$ im String X1$. Resultat ist eine Integer-Zahl.
Y$ = SEL (X$, A, L)	Selektion der Zeichenfolge im String X$ ab Zeichen A bis A+L-1

* Konversionen

Y = ASC (X$)	Resultat ist der ASCII-Wert des Zeichen X$
$ = CHR (X)	Resultat ist das ASCII-Zeichen, welches der Zahl X entspricht.
Y = NUM (X$)	Die Zeichen von X$ (zulässig nur +-.E und Ziffer) werden in die interne Zahlendarstellung umgerechnet.
Y$ = STR (X)	Die Zahl X wird in die externe Zahlendarstellung umgerechnet.

3.4 System-Variablen

C-BASIC kennt einige Systemvariablen, die wie gewöhnliche Variable abgefragt, jedoch nicht verändert werden können.

- Zeit-Variablen

MIN : Gibt die aktuelle Minute an.

HOUR : Gibt die aktuelle Stunde an.

DATE$: Liefert als Zeichenkette von 10 Zeichen das aktuelle Datum (z.B.: 10.12.1974)

Die System-Variablen erhalten bei der System-Initialisierung durch den System-Manager Anfangswerte und werden dann vom System geführt.

- Fehler-Variablen

Zur Behandlung von semantischen Fehlern, die bei der Durchführung eines Programmes auftreten können, (z.B. Bereichsüberschreitung, fehlerhafte Eingabe) stellt C-BASIC folgende Variablen zur Verfügung:

ERN : Fehler-Nummer

ERL : Fehler-Label

ERN gibt den aufgetretenen Fehlertyp, ERL (Error Label) die Nummer desjenigen Statements an, bei dem der Fehler aufgetreten ist.

Die genaueren Angaben zur Fehlerbehandlung sind unter dem Kapitel "Statements" (4.7) zu finden.

4. C-BASIC-Statements

Statements bestehen aus einer Statementnummer, gefolgt von einem Statement-Codewort und einer Reihe weiterer spezifischer Angaben. Die Statementnummer (1 bis 32767) gibt dem System an, wie die Statements geordnet werden sollen, dient als Unterscheidungsmerkmal von Kommandos und vor allem auch als Statementmarke für Sprunganweisungen.

C-BASIC erlaubt die Eingabe mehrerer durch ":" getrennter Statements pro Zeile (≤ 72 Zeichen). Eine Zeile wird durch das Zeichen "cr" (Wagenrücklauf) abgeschlossen. Mit dem Zeichen "Del" kann eine Eingabe gelöscht werden, die Sonderzeichen folge "←←← ..." löscht die zuletzt eingegebenen Zeichen.

4.1 Kommentare

```
REM Kommentar
```

Dieses Statement dient dem Einfügen von Kommentaren in ein Programm. Es wird bei Durchführung eines Programmes und bei Übersetzung (COMPILE) übergangen.

Das Zeichen ! hinter einem Statement erlaubt die Zufügung von Kommentar in einer Zeile.

Kommentare kosten zwar Speicherplatz, sind aber für die Lesbarkeit und Strukturierung eines Programmes unbedingt erforderlich. Über die Möglichkeit des Mischeffektes (LOAD, READ) lassen sich Kommentare separat führen und für Dokumationszwecke in ein Programm einbauen. Auf diese Weise ist der Speicherplatz zur Laufzeit optimal ausgenutzt.

4.2 Deklarationen

4.2.1 DIM-Statement

```
DIM v1 (i), v2 (i,k)
DIM v1@ (i), v2@ (i,k)
DIM v1$ (i) !), v2$ (i)
```

Das DIM-Statement reserviert Speicherplatz für einen oder mehrere Bereiche. Hinter dem Schlüsselwort DIM sind die Bereichsangaben spezifiziert. Jede von ihnen enthält den Bereichsnamen von Typ "Real", "Integer", "String" und die Bereichsgröße. Bei Real-Bereichen werden (i + 1) * 8 bzw. (i + 1) * (k + 1) * 8 Speicherbytes reserviert, bei Integer-Bereichen (i + 1) * 2 bzw. (i + 1) *(k + 1)* 2. Bei String-Bereichen sind es (i + 1)* ! Bytes bzw. (i + 1)* 2 byte, falls die Angabe ! wie im dritten Statement entfällt.

Bereichsangaben für Real-, Integer- und Stringbereiche können auch gemischt stehen.

Beispiel: 10 DIM AB (15), B ∅ (20,2), CD$ (10) 5)

Defekt-Modus: Eindimensionale Bereiche brauchen nicht explizit deklariert zu werden.

C-BASIC nimmt in diesem Fall als Defekt einen Bereich von 10 Zahlen oder 10 Strings zu je 2 Zeichen an.

4.2.2 CHAR - Statement

```
CHAR  v1$ (I1), v2$ (I2),....
```

Dieses Statement reserviert für eine einfache Stringvariable einen Speicherplatz von I1 bzw. I2 byte. Entfällt eine solche Reservierung, so werden als Defektmodus 2 Zeichen angenommen.

Beispiel: 15 CHAR ARB$ (10), BETA$ (100)

4.2.3 DEF-Statement

```
DEF  FNf (v1, v2, ....) = a
```

C - BASIC kennt die Möglichkeit zur Deklaration einer einzeiligen Benutzerfunktion mit bis zu 4 Parametern.

Die Funktion mit dem Namen FNf, wobei f für eine beliebige Zeichenfolge

(≤ 4 Zeichen) steht, kann in einem beliebigen Ausdruck anstelle eines Operanden in der Form FNf (a1, a2, ...) verwendet werden. a1, a2 bedeuten hierbei Ausdrücke, deren Wert errechnet und vor Aufruf der Funktion den Leer-Variablen v1, v2 zugewiesen werden. Danach wird der Funktionsausdruck a ausgewertet.

```
Beispiel: 100 DEF FNPOLY∅(X,A,B,C) = A*X↑3 + B*X↑2 + C*X
          110 DEF FNEXOR  (X1,X2) = (X1 AND NOT X2) OR (NOT X1 AND X2)
```

Beispiel 1 errechnet ein Polynom, Beispiel 2 bildet die Funktion "Exklusiv Oder".

4.2.4 DATA-Statement

```
DATA c1, c2, c3 ....
```

Dieses Statement spezifiziert eine Liste von Konstanten ci. Diese können vom Typ "Zahl (Real, Integer)" oder "String" sein. Die Konstanten werden durch das Statement READ (4.3.1) gelesen und Variablen zugewiesen.

Beispiel: 100 DATA 5, 6.23, - 7E5, "ENDE"

4.2.5 Weitere Deklarationen

Zur Deklaration von Dateien sind weitere Statements vorgesehen, die im Kapitel 5 behandelt werden.

4.3 Zuordnungen

4.3.1 READ-Statement

```
READ v1, v2, v3, ....
```

Das Statement leistet die Zuordnung von Konstanten, die in DATA-Statements aufgezählt sind, zu Variablen. Im System ist ein Zeiger vorhanden, der bei Programmbeginn auf die erste Konstante des ersten DATA-Statements zeigt. Diese Konstante wird gelesen und der ersten in READ angegebenen Variablen zugewiesen. Anschließend zeigt der Zeiger auf die nächste Konstante, die der Wert der nächsten Variablen wird. Ist ein DATA-Statement erschöpft, setzt das System den Zeiger auf die erste Konstante des nächstfolgenden DATA-Statements oder meldet Fehler, falls ein solcher nicht existiert. Die gelesenen Konstanten müssen im Typ mit den Variablen übereinstimmen.

C-BASIC bietet die Möglichkeit, mit einer speziellen Angabe einen Bereich mit Daten zu füllen. Wird im Argument einer Bereichsvariablen statt einer Zahl oder eines Ausdruckes das Zeichen " * " angegeben, so werden aus dem DATA-Feld so viele Konstanten gelesen, bis der Bereich belegt ist. Die Abspeicherung geschieht bei zweidimensionalen Bereichen zeilenweise.

Beispiel: 10 DATA 5,6,7,8, 9E4, "AB"

```
20 READ A, B, C, D, E@, D$
30 DIM BE@(3)
40 DATA 6.23, 7.5, 6.8, 9.4, "ABC", "DEF"
50 READ BE@(*), CHAR$ (*)
```

Im Statement 20 werden Werte zu Einzelvariablen übertragen, in Statement 50 an Bereiche.

4.3.2 RESTORE-Statement

```
RES  n
```

Der DATA-Zeiger wird mit Hilfe des Statements neu gesetzt. Nach Durchführung von RESTORE zeigt er auf die erste Konstante im auf Statement n folgenden DATA-Statement. Die Nummernangabe kann auch entfallen, wenn der Zeiger auf das erste DATA-Statement gesetzt werden soll.

4.3.3 INPUT-Statement

```
INPUT  DEV (d), t, u1, u2, u3,....
```

Das Statement erlaubt die Eingabe von Konstanten von einem Gerät d während des Programmablaufes. Das Gerät kann die Konsole oder auch ein beliebiges anderes Eingabegerät sein. Entfällt eine Geräteangabe, so erfolgt die Eingabe vom zuletzt in einem INPUT- oder PRINT-Statement spezifizierten. Bei Programmanfang ist das Konsolgerät (d = o) angewählt. Der Teilnehmer wird in diesem Fall vom System durch "?" aufgefordert, Daten einzugeben. Vor dem "?" wird der Text "t", falls er vorhanden ist, noch ausgegeben.

Danach werden Daten in der üblichen Weise (mit Korrekturmöglichkeit "Del" und "← ") eingegeben. Nach Abschluß durch das Zeichen "cr" ordnet das System die eingegebenen Werte den aufgeführten Variablen zu. Hier gelten genau dieselben Vereinbarungen wie bei READ; hat der Teilnehmer zu wenig Daten eingegeben, so meldet sich das System mit "??".

Fehler, die bei der Interpretation der Benutzereingabe registriert werden, führen nicht zum Programm-Abbruch, wenn im Programm vor dem INPUT-Statement ein ON ERR-Statement (siehe 4.7) steht.

Beispiele: 10 INPUT DEV (1) , "GIB EIN", A$, B$, C@
20 INPUT "WIEVIEL IST", DE (5,6)

4.3.4 LET-Statement

```
LET v = a
LET v1 = a1, v2 = a2, ....
```

Dieses Statement ist die häufigste Zuweisungsform eines Wertes zu einer Variablen.

Der Ausdruck a wird zunächst ausgewertet und dann in den durch die Variable v angegebenen Platz eingeschrieben. Bei Typenmischung rechts und links vom Zuweisungszeichen erfolgt eine automatische Konversion entsprechend dem Variablentyp. (Bei Zahlen-String-Mischung Fehlermeldung!).

Die Syntax des LET-Statements erlaubt neben der einfachen auch die Mehrfachzuweisung.

Beispiele: 10 LET A ∋ = B ∋ + 5
20 LET XA (5,3) = XB + 6 E3, XB = 5 ↑ (A + 1)
30 LET Z$ (2) = "ABCD" & X$

4.3.5 ROUND - Statement

```
ROUND r
```

Dieses Statement dient dazu, dem System anzugeben, wie reelle Zahlenwerte vor Zuweisung zu einer reellen Variablen im LET-Statement oder vor Ausgabe in einen PRINT-Statement gerundet werden sollen. Die Angabe "r" ist entweder eine Konstante oder Variable. Der Wert von r wird ab diesem ROUND-Statement zu dem errechneten Ausdruckswert hinzuaddiert. Anschließend werden alle Stellen ab der durch "r" angegebenen ersten signifikanten Ziffer auf 0 gesetzt.

Beispiel 1: Rundung vor dem Komma.
10 ROUND 50
20 LET A = 501 * 10, B = 507 * 10

Wert von A: 5010, B:5070. Nach Rundung A:5000, B:5100

Beispiel 2: Rundung nach dem Komma
10 ROUND 0.05
20 LET = 5.3 * 6.2, D = 0.1 * 0.8

Wert von C:32.86, D: 0.08. Nach Rundung C: 32.90, D:0.10

Bei ROUND 0 erfolgt keine Rundung mehr.

4.4 Ausgabe

4.4.1 Ausgabe von Zahlen, Texten, Steuerzeichen

```
PRINT DEV (d), a
PRINT DEV (d), a1, a2, ....
```

Das Statement dient der Ausgabe von Zahlenwerten und/oder Texten auf ein peripheres Gerät "d", das in DEV (d) spezifiziert ist. Eine explizite Spezifikation ist dabei nicht nötig, wenn die Ausgabe auf das zuletzt angewählte

Gerät erfolgen soll. Die Ausdrücke a können beliebige numerische Ausdrücke Stringkonstanten bzw. Stringvariablen sein. Als Trennzeichen zwischen den einzelnen Angaben fungiert das Komma oder der Strichpunkt. Ein Komma bedeutet, daß nach der Ausdrucksausgabe solange Leerzeichen folgen, bis die nächste Tabulatorposition (Spalte 16,32,48,..) erreicht ist.

Die Ausgabe von Zahlen geschieht, wenn nicht anders spezifiziert (siehe 4.4.2), in einem automatischen Format, das entsprechend der Zahlengröße arbeitet.

Zeichenketten können aus einer Folge direkt auszugebender Zeichen (z.B. "ABCD..") bestehen, aus einer Folge von Hexakonstanten (z.B. % A1FF B005 %), aber auch aus einer Folge von Funktionssymbolen (z.B. 'CRLF'), die gewisse Steuerfunktionen am Bildschirm, Drucker, Lochstreifengerät auslösen.
Prinzipiell können solche Funktionen auch im Hexacode angegeben werden, aus Gründen der Mnemotechnik ist es jedoch sinnvoll, hierfür eine spezielle Symbolik zu verwenden.

CR Carriage Return	SF Set Foreground
LF Line Feed	SB Set Background
BL Bell	
HT Horizontal Tab.	FF Form Feed
VT Vertical Tab.	PL Print Line
CP Clear Page	
CH Cursor Home	

Diese Funktionsliste ist erweiterungsfähig.

Beispiel: 10 PRINT 'CRLFLFLFBL'; "RESULTAT"; 3 A + B
15 PRINT DEV (3); 'FF'; "UEBERSCHRIFT"; 'CRLFLF'

4.4.2 Formatspezifikationen

Anstelle eines Ausdruckes in der Liste des PRINT-Statement kann eine Formatspezifikation stehen. Wird eine solche bei der Programmdurchführung angetroffen, ist die bisherige Spezifikation aufgehoben und durch die neue ersetzt. Alle Zahlen werden ab jetzt gemäß dieser Spezifikation formatisiert. C-BASIC kennt eine FORTRAN und eine COBOL-ähnliche Spezifikation.

```
a = FMT (f)
a = USING (s)
```

Bei der Angabe FMT steht f für folgende Möglichkeiten:

A : Umschalten zum automatischen Format

Fw.d: Ausgabe im Dezimal (oder bei d = 0: Integer) Format mit insgesamt w Stellen (inklusiv Vorzeichen und Dezimalpunkt) , wobei "d" Stellen hinter dem Komma stehen.

Ew.d: Die Ausgabe soll im Exponentialformat erfolgen . Hinter den Ziffern der Mantisse wird zusätzlich ausgegeben E $\pm$ xx.

Beispiele: 10 PRINT FMT (F 9.3) 12345E-2 : 123.450
20 PRINT FMT (E12.4) 12345E-2 : 1.2345 E-02

Bei der Angabe USING ist "s" eine Folge von Zeichen, die bei der Verwendung zur Zahlenausgabe folgendermaßen interpretiert werden.

9 Stelle für Ziffer

Z Stelle für Ziffer, jedoch statt führender Null Leerstellenausgabe

* Stelle für Ziffer, jedoch statt führender Null * Ausgabe (Schecksicherung)

- Stelle für Vorzeichen (- bei neg. Zahlen, ␣ bei pos. Zahlen)

+ Stelle für Vorzeichen (- bei neg. Zahlen, + bei pos. Zahlen)

Die Vorzeichenstelle kann sowohl vor, als auch hinter der Zahl stehen.

, Stelle für Dezimalkomma. Die Zahl wird in das Format so eingepaßt, daß hier das Dezimalkomma steht.

␣ Leerstelle (vor, hinter oder zwischen Zahlen möglich).

E Stelle für Exponentialzeichen

Beispiel: 100 PRINT USING (+ZZ 9.999) 123.45 : +123.450
110 PRINT USING (-ZZZZ E+ZZ) 123.45 : 1234E-01

4.4.3 Positionierungsfunktion

Neben den Zeichen "; " und "," sowie den Steuerfunktionen ist in C-BASIC noch folgende Funktion zur Positionierung erlaubt:

TAB (X)	: x- Positionierung
TAB (x,y)	: xy - Positionierung

Diese Funktion darf anstelle eines Ausdruckes in der PRINT-Liste stehen. Die Integer-Werte von x, y geben die absolute Positionierung für den Schreibkopf (Cursor) an.

4.5 Ablauf-Befehle

4.5.1 Sprung-Statement

```
GOTO   n
GOTO   a  OF  n1, n2,....
```

Mit GOTO wird der Programmablauf bei dem Statement mit der Anweisungsnummer "n" fortgesetzt. In der Form GOTO..OF wird das Sprungziel aus dem Wert des Ausdruckes a bestimmt. Liegt dieser zwischen den positiven ganzen Zahlen i (> o) und i+1, so wird zu dem Statement mit der Nummer ni verzweigt, falls diese in der Aufzählung noch vorhanden ist. Entspricht der Wert von a keiner der angegebenen Nummern, wird das auf GOTO folgende Statement ausgeführt.

Beispiel:
```
10 INPUT A
20 GOTO A OF 100,200,300
30 PRINT "A < 1  ODER  A > 3"
```

4.5.2 Unterprogramm-Sprung

```
GOSUB n (a1,a2...)
GOSUB a OF n1, n2,.... (a1, a2,...)
```

Die Anweisung GOSUB bewirkt einen Sprung in ein Unterprogramm, das mit der Anweisungsnummer n beginnt. Beim Aufruf des Unterprogrammes kann eine Liste von Ausdrücken angegeben werden. Die Werte dieser Ausdrücke werden errechnet und den Leervariablen im Statement SUB zugewiesen. Es ist auch ein Unterprogrammaufruf ohne Parameterübergabe möglich.

Das Sprungziel beim berechneten GOSUB wird wie bei GOTO errechnet.

Am Ende einer Unterprogramm-Durchführung (RETURN) erfolgt Rücksprung auf das hinter GOSUB stehende Statement.

4.5.3 Unterprogramm-Statements

```
SUB (v1, v2,...)
RETURN
```

Bei Unterprogrammen mit Parametern muß das erste Statement im Unterprogramm das Statement SUB sein. In Klammern stehen eine Reihe von Leervariablen, denen vor Durchführung des Unterprogrammes die in GOSUB errechneten Werte zugewiesen werden. Die Übergabe-Technik entspricht der bei DEF. Die Rückkehr in das rufende Programm geschieht nach Interpretation von RETURN.

Eine Schachtelung von Unter-Programmaufrufen ist bis zu einer Tiefe von 8 möglich.

4.5.4 IF-Statement

```
IF a THEN n
IF a Statement
```

Dieses Statement erlaubt die bedingte Verzweigung zu einer Statementnummer "n" oder die bedingte Durchführung eines aktiven (nicht deklarativen) Statements. Ist der Wert des Ausdruckes $\neq 0$, so gilt die Bedingung als erfüllt.

Beispiel:
```
10 IF A@ = B@ THEN 200
20 IF A AND B     PRINT "RICHTIG"
30 IF A$ = "ABCD" GOTO 200
```

4.5.5 Schleifen-Statement

```
FOR v = a1 TO a2 STEP a3
NEXT v
```

Die Statements FOR und NEXT eröffnen und schließen eine Programmschleife. Bei Beginn wird die Schleifenvariable v auf den Anfangswert a1 gesetzt und die Schleife durchlaufen. Bei NEXT wird der Wert a3, der positiv oder negativ sein kann, zum Schleifenindex hinzuaddiert.
Anschließend erfolgt die Prüfung, ob er den Schleifenendwert a2 schon überschritten (bei positiver Schrittweite) oder unterschritten (bei negativer Schrittweite) hat. Ist dies der Fall, wird die Schleife beendet.

Jedem FOR muß genau ein NEXT entsprechen. Schleifen dürfen verschachtelt werden (bis zur Stufe 8). Überkreuzte Schachtelung führt jedoch zu einer Fehlermeldung.

Beispiel:
```
10 FOR I = 0.1 TO END STEP 0.1 * S
100 NEXT I
```

4.5.6 Programmende

```
STOP
END
```

Das Statement END ist das letzte Statement eines Programmes. Statt eines Sprungstatements auf END, kann innerhalb eines Programmes STOP verwendet werden.

4.6 Programm-Verkettung

```
LINK   seg
ENDS
```

Der Speicherbereich eines Teilnehmers am Timesharingbetrieb wird häufig für Programm- und Datenspeicherung nicht voll ausreichen. In diesem Fall kann das Programm "segmentiert", d.h. in Teile zerlegt werden. Segmente liegen unter einem Namen "seg" auf der Platte und werden durch das Statement LINK bei Durchführung in den Speicher geladen.

Segmentierte Programme bestehen aus einer Wurzel (Root) und bis zu 255 Segmenten. Die Wurzel ist dadurch ausgezeichnet, daß sie permanent im Speicher steht. Sie braucht im Prinzip nur aus dem Statement LINK zu bestehen. Im allgemeinen ist es jedoch sinnvoll, Programmteile und Daten, die in vielen Segmenten gebraucht werden, in der Wurzel unterzubringen.

Ein Segment kann mit ENDS in die Wurzel zurückkehren oder mit "LINK seg" ein weiteres Segment aufrufen. Dabei wird es aber überschrieben. Durch ein anderes Segment kann es jedoch wieder gerufen werden.

Beispiel:

```
 5 DIM DAT (100)
10 LINK ABFR                                  Wurzel
20 END
                                              ----------
100 REM SEGMENT ABFRAGE
110 INPUT "WELCHE FUNKTION", FUN
120 GOTO FUN OF 150, 160, 170                 Segment 1
150 LINK EING
160 LINK RECH
170 LINK AUSG
180 STOP                                      ----------
100 REM SEGMENT EINGABE
100 REM SEGMENT RECHNUNG                      Weitere Segmente
100 REM SEGMENT AUSGABE
```

In diesem Beispiel wird in der Wurzel (5-20) ein Datenfeld zur Aufnahme von Daten eröffnet. Das Segment ABF R dient der Abfrage einer Funktion, die der Benutzer anwählen kann. Im Beispiel soll dies die Dateneingabe im Segment EING, eine Datenauswertung im Segment RECH und eine Ausgabe in AUSG sein.

Es ist zu beachten, daß verschiedene Segmente dieselben Statementnummern haben können, da sie nichtgleichzeitig im Speicher stehen. Alle Variablennamen, die nicht in der Wurzel vorkommen, sind außerdem lokal !

4.7 Fehlerbehandlung

Bei Fehlern, die bei Programmeingabe auftreten, gibt C-BASIC auf der Teilnehmerkonsole aus:

ERR Fehlernummer

Aus einer Fehlertabelle entnimmt der Teilnehmer die entsprechende Fehlerspezifikation und eventuell Korrektur-Hinweise. Zur Laufzeit können syntaktische Fehler auftreten, die bei Programmeingabe vom System noch nicht erkannt werden konnten (z.B. zu FOR fehlt ein NEXT). Außerdem ist eine Reihe semantischer Fehler erkennbar (z.B. Feldbereichs-Überschreitung). Normalerweise bricht das System bei Fehlern den Programmablauf ab und gibt die Meldung aus:

ERR e IN LINE n SEG seg

Es kann jedoch häufig wünschenswert sein, solche Fehlerabbrüche zu vermeiden und eine benutzerspezifische Maßnahme zu ergreifen. Dazu stellt C-BASIC folgende Statements zur Verfügung:

```
ON ERR GOTO n
RESUME n
RESET
```

Mit ON ERR wird dem System mitgeteilt, daß ab jetzt die Fehlerangabe unterbleiben und statt dessen zu Statement n verzweigt werden soll, wenn ein Fehler auftritt. Ab diesem Statement wird ein Reaktionsprogramm zur Fehleranalyse und zum Ergreifen spezieller Maßnahmen stehen. Beispielsweise kann es wünschenswert sein, das Statement, bei dem der Fehler aufgetreten ist, zu wiederholen (z.B. INPUT). In diesem Fall wird das Statement RESUME ohne Nummeranangabe verwendet. Das System löscht die Fehlerzelle und wiederholt das Statement. Oft ist es jedoch nötig, einen speziellen Vorgang vor dem Fehlerstatement zu wiederholen. Dazu wird RESUME n benutzt. Nach Löschen der Fehlerzelle geht das System zu Statement n. Soll der nächste auf das Fehlerstatement folgende Befehl durchgeführt werden, wird RESUME o geschrieben.

Häufig kann es notwendig sein, zum Zweck der Fortsetzung des Programms nach Auftreten des Fehlers interne Zustandsmerker (Zeiger für Schleifen- und Unterprogramm-Keller, Format- und Rundungsspezifikation) zurückzusetzen. Dies leistet das Statement RESET.

Zur Abfrage der Fehlerart und der Nummer des Fehlerstatements hat C-BASIC die Systemvariablen ERN, ERL parat.

Beispiel:

```
10 ON ERR  GOTO 1000
20 INPUT   A,B,C
30 PRINT   A+B+C
40 GOTO    20

1000 REM FEHLERBEHANDLUNG

1010 IF ERL = 20 RESUME
1020 IF ERL = 30 PRINT "OVERFLOW"
1030 RESUME 20
```

5. C-BASIC-Datei-Verwaltung

Ein Hauptmerkmal der kommerziellen Datenverarbeitung ist die Menge der anfallenden Daten. Sie müssen in problemangepaßten Strukturen auf einen Externspeicher untergebracht und über rationelle Zugriffsmethoden abgerufen werden können.

C-BASIC liefert hierzu Sprachelemente, die dem allgemeinen Niveau der BASIC-Sprache angepaßt sind und auf dem Datei-Verwaltungsmodul DFMS (Disk File Management System) beruhen.

5.1 Daten-Struktur

Bild 5 zeigt den prinzipiellen Aufbau der Datenstruktur auf der Platte.

Alle logisch zusammengehörigen Daten sind physikalisch zu einer Datei zusammengefaßt. Jede Datei besteht aus Sätzen, die nach einem Klassifikationsmerkmal unterschieden werden. Sätze selbst sind in Felder strukturiert. Felder können Einzeldaten oder Datengruppen enthalten.

Beispiel:
Die Lagerbestands-Verwaltung benutzt in der allgemeinen Firmendatenbank eine Datei, welche alle Daten über integrierte Schaltkreise enthält.

Die Sätze der Datei werden beispielsweise nach der logischen Funktion eines IC klassifiziert.

Satz 1: NAND-4 Satz 2: NOR-4 usw.

Die Satzfelder können folgende Daten enthalten:

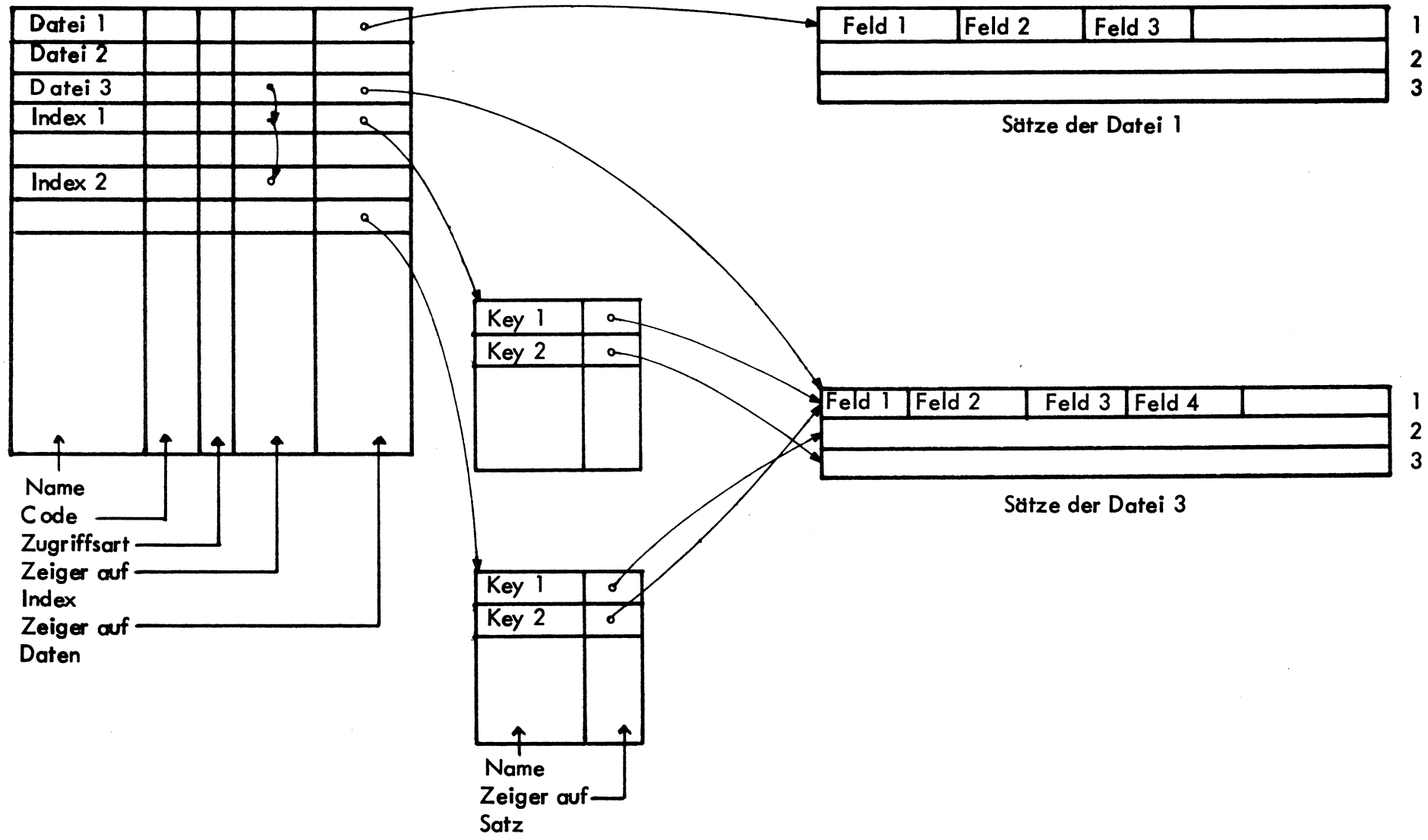

Bild 5: Struktur der Datenbank

Lagerbestand - Benötigte Stückzahl pro Rechner

Lieferfirma 1

Lieferzeit - Kosten pro Stück - Rabatte

Leistungsverbrauch, Gehäuseform - Toleranzen - Zuverlässigkeit

Ausstehende Lieferungen - Mengenverträge

Lieferfirma 2

usw.

Die Namen aller Dateien sind in einem Datei-Verzeichnis mit dem Schutzcode des Autors, der erlaubten Zugriffsart (privat, öffentlich, halböffentlich) und mit 2 Zeigern versehen. Der erste Zeiger weist auf den ersten Satz der Datei, der zweite ist entweder leer oder weist auf ein Schlüsselverzeichnis (Index), das Namen zu den Sätzen der Datei enthält. Im obigen Beispiel sind die Satzschlüssel die IC-Typen (NAND-4 usw). Eine C-BASIC-Datei kann mehrere Schlüsselverzeichnisse haben. Jedes von ihnen hat einen Namen. Im obigen Beispiel ist sicher ein Verzeichnis der zum Bau eines Rechnertyps erlaubten IC sinnvoll, weiter kann ein Verzeichnis der IC in MIL-Norm unter Umständen für einen Verarbeitungsvorgang notwendig sein.

Der Sinn mehrere Schlüsselverzeichnisse zu einer Datei ist der, daß mehrere Personen mit unterschiedlichen Verarbeitungsaufgaben mit Namen, die ihren praktischen Bedürfnissen entsprechen, und vor allem schnell auf Daten zugreifen können.

Auf alle Sätze einer Datei mit Schlüsselverzeichnissen kann direkt über die Satznummer oder indirekt über den Schlüssel zugegriffen werden. Direkt wird man immer dann zugreifen, wenn der Inhalt des angesprochenen Satzes bekannt ist. Meist ist dies jedoch nicht der Fall, da der Aufbau einer Datei von mehreren Personen her erfolgen kann.

Da die Schlüssel in einem Schlüsselverzeichnis immer sortiert sind, erfolgt der Zugriff zu einem bestimmten Satz relativ schnell.

Alle Sätze einer Datei haben eine maximale in einer Deklaration anzugebende Länge. Ebenso ist die Maximalzahl der Sätze in der Deklaration zu vereinbaren. Bevor in eine Datei Daten eingetragen werden können, ist diese zu öffnen. Die Datei-Verwaltung führt zu jeder eröffneten Datei einen Satz-Zeiger, der auf den gerade in Bearbeitung befindlichen Satz zeigt, außerdem einen Satz - offset - Zeiger, der das Feld angibt, auf das innerhalb des Satzes als nächstes zugegriffen wird.

C-BASIC hat mehrere Statements zum Eröffnen von Schlüsselverzeichnissen zum Eintragen, Löschen, Ändern und Verketten von Schlüsseln.

5.2 Datei-Deklaration

```
CREATE u, file (s, l)
```

u = Geräte-Einheit (Platte)
file = Dateiname (1. Zeichen: Buchstabe, 2. bis 6. Zeichen beliebig)
s = Anzahl der Sätze
l = Länge der Sätze (in Sektoren zu 128 byte)

Mit CREATE wird eine Datei eröffnet.

Die Dateiverwaltung prüft, ob der Name bereits auf der Einheit "u" vorliegt. Ist dies der Fall, erfolgt eine Fehlermeldung. Sonst wird der Name mit dem Code des eröffnenden Teilnehmers in das Dateien-Verzeichnis aufgenommen und als "privat" erklärt.

Gleichzeitig wird der benötigte Platz auf der Platte reserviert und initialisiert.

Beispiel:
```
100 CREATE  0, ICTYPE  (200, 15)
150 CREATE  5, ANGEST (150, 32)
```

```
CRIND u, file; index (si, li)
CRIND u, file; index 1 (si1, li1), index 2 (si2, li2), ....
```

index = Namen des Schlüsselverzeichnisses (1-6 Zeichen)
si = Zahl der Schlüssel
li = Länge der Schlüssen (in Byte)

Die Datei-Verwaltung eröffnet zu der angegebenen Datei "file" auf Einheit "u" Schlüsselverzeichnisse unter den angegebenen Namen und reserviert für sie entsprechend den Informationen über Zahl und Länge der Schlüssel Platz. Durch dieses Statement ist jedoch noch keine Zuordnung zu den Sätzen der Datei vorgenommen. Die Eröffnung von Schlüsselverzeichnissen kann auch dann erfolgen, wenn bereits Sätze der Datei beschrieben sind.

Beispiel:
```
100 CREATE  5, ANGEST (150, 32)
110 CRIND   5, ANGEST, NAME (150,20),WEIBL (30,20 ),
               MAEN (120,20)
```

In dem Beispiel wird eine Datei ANGEST(ELLTE) mit 150 Sätzen zu je 4K Datenspeicher eröffnet. Danach bekommt die Datei 3 Schlüsselverzeichnisse, eines für alle Personen, ein zweites für die weiblichen, ein drittes für die männlichen Angestellten.

5.3 Eröffnen und Schließen von Dateien

```
OPEN u, file [, index] AS w
EXOP u, file [, index] AS w
CLOSE w1 , w2 , ...
```

u = Platteneinheit
file = Datei-Name
index= Schlüsselverzeichnis
w = Integervariable zur Aufnahme der Arbeitsnummer

Bevor zu den Daten oder zu dem Inhalt von Schlüsselverzeichnisse zugegriffen werden kann, muß durch OPEN eine Anmeldung bei der Datei-Verwaltung erfolgen. Diese bestimmt daraufhin die physikalische Adresse des ersten Satzes der Datei oder des Indexverzeichnisses, falls "index" angegeben ist und speichert diese in einer Tabelle im Hauptspeicher ab. Gleichzeitig werden in dieser Tabelle die Satzzeiger initialisiert.
Die Nummer der Eintragung wird sodann der Integer-Variablen w zugewiesen. Diese Nummer dient ab jetzt als Arbeitsnummer für sämtliche Lese- und Schreiboperationen. Ihre Verwendung spart Schreibarbeit und sorgt vor allem für schnellen Zugriff.

Dateien können im Teilnehmerbetrieb von mehreren Benutzern gleichzeitig zum Zwecke des Lesens und Schreibens eröffnet werden, wenn sie nicht privat sind. Erfordert es ein Vorgang (z.B. Sortieren), daß, während der gesamten Zeit, in der ein Teilnehmer auf eine Datei zugreift, kein anderer die Datei benutzen darf, so wird sie mit EXOP (= EXclusiv OPEN) eröffnet. Dabei prüft das System zunächst, ob die Datei bereits belegt wurde. Ist dies der Fall, so wird das neu eröffnende Programm solange in den Wartezustand versetzt, bis die Datei nicht mehr belegt ist. Alle Teilnehmer, die ab jetzt auf diese Datei mit OPEN oder EXOP zugreifen wollen, müssen warten.

Eine Datei wird geschlossen durch CLOSE. In einem CLOSE können gleichzeitig mehrere Arbeitsnummern angegeben werden.

5.4 Beschreiben/Löschen von Sätzen

```
WRITE (w    [, key] ) a1, a2, a3, ....
WRITE (w    [, key] )
```

Mit WRITE wird ein Satz der eröffneten Datei w mit den Werten der Ausdrücke a1, a2, a3 ... beschrieben. Die Angabe "key" kann dabei ein numerischer Integerausdruck (Konstante oder Variable) oder auch ein String (Konstante oder Variable) sein. Im ersten Fall wird über die Satznummer, im zweiten über den Satzschlüssel zugegriffen. Die Angabe muß mit der in OPEN angegebenen Eröffnungsart übereinstimmen. Ist der angegebene Satzschlüssel noch nicht vorhanden, wird er in das Verzeichnis aufgenommen.

Das zu beschreibende Feld ist durch die Satzangabe und durch den Satz-offset-Zeiger bestimmt. Falls jedoch die Satzangabe "key" entfällt, wird der Satz von dem internen Satzzeiger ausgewählt. Diese Möglichkeit erlaubt sequentiellen Satzzugriff, da der Satzzeiger automatisch inkrementiert wird, wenn ein Satz bearbeitet ist.

Der Reihe nach werden nun die Werte der Ausdrücke "ai" errechnet und insgesamt in den Satz übertragen. Die Werte können vom Typ numerisch oder Strings sein. Ihre Größe bestimmt die Länge der Satzfelder. Stringfelder werden durch eine spezielle Stringendmarke abgeschlossen, damit beim Lesen der richtige Zugriff erfolgen kann.

Wie bereits bei der Beschreibung des READ-Statement angegeben wurde, können durch Angabe von "Name (*)" oder "Name (*,*)" komplette Bereiche übertragen werden.

Am Ende eines Schreibvorganges zeigt der Satzoffset-Zeiger auf die nächste unbelegte Feldposition, so daß im nächsten Schreibbefehl an dieser Stelle weitergeschrieben werden kann.

Ein Schreibbefehl ohne eine Ausdrucksliste dient zum Löschen eines Satzes. Der Satzoffset-Zeiger wird auf den Anfang des Satzes gestellt. Schlüssel, die in eventuellen Verzeichnissen auf diesen Satz weisen, sind davon unberührt.

5.5. Lesen/Ändern von Sätzen

```
READ   (w    [, key]  ) v1, v2, v3, ....
UPDATE (w    [, key]  ) v1, v2, v3, ....
```

Der Befehl READ liest aus dem durch "key" adressierten Satz der Datei w Zahlen und Stringkonstante und weist sie den Variablen "v" der Variablenliste zu. Die Variablentypen müssen dabei den Datentypen bei dem vorangegangenen Schreibbefehl entsprechen.

Die Arbeitsweise des READ-Statement entspricht dem Statementpaar READ-DATA. Wie dort, lassen sich auch hier komplette Bereiche übertragen.

Der Satzschlüssel "key" kann ein Integerwert oder ein String sein. Wird er nicht angegeben, ist die Stellung des zur Nummer w gehörenden Satzzeigers für die Adressierung maßgebend.

Das Statement UPDATE hat, was das Lesen eines Satzes betrifft, dieselbe Semantik wie READ. Jedoch dient hier das Lesen zum Zwecke der Änderung (UPDATE) eines Satzes. Der Zugriff zu diesem Satz wird deshalb solange gesperrt, bis derselbe Teilnehmer erneut einen READ/UPDATE oder WRITE-Befehl anmeldet.

Außerdem wird der Satzzeiger nach dem Lesen auf den vor dem Lesen bestehenden Ausgangszustand gebracht. Der nach UPDATE übliche Befehl ist im allgemeinen der Befehl WRITE, wobei die zu schreibenden Werte gegenüber den Originalwerten ganz oder in Teilen modifiziert sind.

5.6 Ändern eines Schlüsselverzeichnisses

Wie bereits bei WRITE erwähnt, werden dort "neue" Schlüssel in das eröffnete Schlüsselverzeichnis aufgenommen und Satznummern zugeordnet. Folgende Befehle dienen dem Löschen, Umbenennen und Verketten von Schlüsseln:

```
UNKEY  w    , Key1, Key2,....
REKEY  w    , Key1 TO Key2, Key3 TO Key4,....
COKEY  w    , Key TO w1,   Key1 TO w2,   Key2, ....
```

UNKEY (= Unsave Key) löscht alle in der Liste aufgeführten Schlüssel im eröffneten Verzeichnis w . Die Inhalte der Sätze bleiben jedoch unberührt.

REKEY (= Rename Key) erlaubt die Umbenennung von Schlüsseln in dem eröffneten Verzeichnis.

COKEY (= Connect Key) dient der Verknüpfung von Schlüsseln aus den Schlüsselverzeichnissen w1 , w2 , ... mit einem Schlüssel "key" aus w , der bereits einem Satz zugeordnet ist. Sind die Namen key1, key2 noch nicht vorhanden, so werden sie in das Verzeichnis aufgenommen.

Ist die Schlüsselangabe "Key" vom Typ Integer, so werden Sätze, die über Satznummern eingetragen wurden, nachträglich mit Namen belegt.

5.7 Datei-Kommandos

Die in 2.5 beschriebenen Kommandos KILL und ALTER gelten nicht nur für Dateien, die Programme enthalten und durch SAVE eröffnet werden, sondern auch für Dateien, die mit CREATE eröffnet wurden.

Es muß darüberhinaus möglich sein, Schlüsselverzeichnisse einer Datei löschen und verändern zu können.

KILL u, file, index

Dieses Kommando löscht das Schlüsselverzeichnis "index" in der Datei "file".

ALTER u, file, index 1, index 2

Das Schlüsselverzeichnis "index 1" wird in "index 2" umbenannt.

Heinrich Dietz
Industrie-Elektronik
433 Mülheim-Ruhr
Solinger Straße 9
Tel. (0 21 33) 48 50 24
Telex 8 56 770

DIETZ Computer SYSTEME