Der Einsatz von datakit in eigenen Basic-Programmen

Vorbemerkung: Dieser Teil der Anleitung wendet sich an Nutzer, die über grundlegende Kenntnisse in Basic verfügen und das MC-Programm datakit in eigene Programme integrieren möchten. In den beiden nachfolgenden Teilen dieses Handbuchs wird erklärt, wie die beiden noch auf Ihrer Kassette befindlichen Nutzerprogramme verwendet werden.

Übersicht:
- wozu dient datakit, was kann es leisten
- der Aufruf von datakit
- die 4 Arten von Datenfeldern, die datakit verwaltet
- die datakit-Befehle im einzelnen:
  a) Einrichten, Ergänzen und Löschen von Feldern: DIM, DEF FN, TAB, +, ERASE
  b) Ein- und Ausgabebefehle: TO und READ
  c) Manipulation binärer Felder: BIN, =, INVERSE, AND, OR, X (=XOR), DATA
  d) Manipulation des Basic-Teils: LINE, FN, s0 und s1 als SAVE/LOAD-Hilfe
  e) Suchroutinen: ?
  f) Sortierroutinen: CAT
  g) Hilfsroutinen: LEN, c (Clear Program), f (free) und VAL$

Um die angegebenen Beispiele durchzuführen, muß datakit (der Einfachheit halber wird von nun an das Kürzel dk verwendet) geladen werden: Bitte legen Sie die Kassette ein und laden mit LOAD "". Das Menü des Ladeprogramms bietet mit der Eingabe "c" die Voraussetzung für die Nutzung von dk in eigenen Programmen und damit auch für die folgenden Beispiele. Vor dem Aufruf von dk in den Beispielen müssen Sie einige Variablen einrichten:

LET dk=62200: LET k$="": LET k=0: LET z$="". Diese Eingaben sind nur einmal nötig.

Wozu dient dk - was kann es leisten?

dk ist eine Erweiterung des Spectrum-Betriebssystems und als MC-Programm im obersten Bereich des Speichers über RAMTOP abgelegt. Es dient der Einrichtung und Manipulation von Datenfeldern mit zusätzlichen Befehlen.

Das Programm kann Datenfelder definieren, kontinuierlich ergänzen oder vollständig löschen. Zeichenfelder können sortiert und mit allen Spectrum-Vergleichsoperanden durchsucht werden; das Suchen kann mit verschiedenen Kriterien gestaffelt durchgeführt werden.

Neben den bekannten BASIC-Feldern bietet dk zwei zusätzliche Arten von Feldern: komprimierte Zeichenfelder und binäre Felder, die sehr platzsparend arbeiten.

Aufruf von dk

Wie jedes MC-Programm wird dk über einen USR-Befehl aufgerufen; die Adresse des Aufrufs ist immer 62200. Daher bietet sich an, diese Adresse einer Variablen zuzuordnen, die man anschließend benutzt:

```
        LET dk=62200
dann:   RANDOMIZE USR dk
oder:   LET wert=USR dk
oder:   PRINT USR dk
etc.
```

Vor dem Aufruf muß natürlich noch der gewünschte Befehl angegeben werden (siehe unten).

Der eigentliche dk-Befehl kann auf zwei Arten angegeben werden:

a) entweder unmittelbar hinter dem Aufruf in einem REM-Befehl:
   RANDOMIZE USR dk: REM dk-Befehl
b) oder in der Zeichenkette z$.

Findet dk hinter einem Aufruf ein REM-Statement, erwartet es in diesem den auszuführenden Befehl. In jedem anderen Fall wird der Inhalt von z$ als dk-Befehl interpretiert.

Beispiel 1:

Der dk-Befehl ' LEN Name' ergibt die aktuelle Anzahl von Elementen eines Feldes.

(Alle dk-Befehle, die wie BASIC-Kürzel aussehen, werden auch wie diese eingegeben: LEN also mit CAPS SHIFT + SYMBOL SHIFT und dann K.)

Bitte geben Sie nun ein: DIM a$(1,10)
PRINT USR dk:REM LEN a$

Dieser Aufruf ergibt den Wert 1; ergänzen Sie nun das Feld a$ um ein Element:

```
LET k$="a$ zwei"
RANDOMIZE USR dk: REM +a$
```

Dieser Befehl, (er wird noch ausführlich erklärt), ergänzt a$ um ein neues Element, das aus k$ kopiert wird:

```
PRINT a$(2)
```

ergibt wieder "a$ zwei". a$ hat nun die neue Dimension (2,10). Nun zur Kontrolle wieder der LEN-Befehl:

```
PRINT USR dk: REM LEN a$
```

Es ergibt sich die neue Anzahl 2.

```
LET z$="LEN a$":PRINT USR dk
```

führt zum gleichen Ergebnis; der Befehl wird aus z$ gelesen, wenn dem Aufruf kein REM-Statement folgt.

Der Befehl "ERASE a$" löscht a$:

```
RANDOMIZE USR dk: REM ERASE a$
PRINT a$(1)
```

führt zu einer Fehlermeldung, da a$ nicht mehr existiert.

Welche Felder kann dk verwalten?

a) BASIC-Felder

dk verwaltet zweidimensionale Zeichenkettenfelder, die mit DIM Name (Anzahl, Länge) definiert sind.

Zahlenfelder müssen mit DIM Name (Anzahl) eingerichtet sein. Beim Ansprechen von Zahlenfeldern durch dk wird synonym zum $-Zeichen das &-Zeichen an den Namen angefügt. Die bei BASIC üblichen Klammern - etwa bei a$(1) - entfallen bei dk: a&1 entspricht a(1).

b) dk-eigene Felder

1) Komprimierte Zeichenkettenfelder
   sind den $-Feldern ähnlich, bieten aber den Vorteil, daß die Länge der einzelnen Elemente unterschiedlich sein kann. Die platzraubende Speicherung abschließender Leerzeichen in $-Feldern und lästige Abkürzungen wegen zu kurzer Längen entfallen. Dem Namen dieser Felder ist anstelle des $-Zeichens ein %-Zeichen angefügt.

2) Binäre Felder
   speichern 0/1-Werte und haben zur Unterscheidung das #-Zeichen vor (!) ihren Namen. Mit diesen Feldern können extrem platzsparend alle Ja-/Nein-Daten gespeichert werden, wie etwa "Mitglied (j/n)" oder "männlich/weiblich" etc. Die #-Felder werden von dk selber zur Markierung von Elementen benutzt, die eine Suchbedingung erfüllen. Mit #-Feldern können logische Verknüpfungen durcngeführt werden, die die schnelle Verbindung verschiedener Suchbedingungen erlauben.

Die im folgenden vorgestellten Felder werden nur noch als Listen bezeichnet, was ihren eindimensionalen Charakter unterstreicht; dabei soll auch deutlicher werden, wie die Daten gespeichert sind und daß (etwa in einer Datei) Elemente verschiedener Listen und gleicher Nummer jeweils einander zugehörig sind.

Die dk-Befehle im einzelnen

Die angegebenen Befehle sind immer als Ergänzung zum dk-Aufruf zu verstehen und stehen bei der Anwendung z.B. in einem REM-Statement.

Beispiel: RANDOMIZE USR dk: REM LEN a$

Der Befehl ist hier nur 'LEN a$'.

a) Einrichten, Ergänzen und Löschen von Listen

1) BASIC-Listen werden wie gewohnt mit dem DIM-Befehl eingerichtet. Es ist wie im Bsp. 1 angedeutet jeweils der Dimensionsparameter Anzahl gleich 1 ausreichend.

2) %-Listen (komprimierte Listen) können mit dem Befehl "DEF FN" eingerichtet werden:

Format	: DEF FN Name% Anzahl
Bsp.	: DEF FN a%1
Name kann sein	: Jedes mit der Tastatur erzeugbare Zeichen außer #. Groß- und Kleinbuchstaben werden unterschieden! Auch '% oder STEP% etc. sind erlaubt.
Anzahl kann sein	: Entweder eine positive Ganzzahl, jede Zahlenvariable, deren Namen aus einem (!) Buchstaben besteht und die kein Zahlenfeld benennt. Die Zählvariable einer FOR/TO-Schleife ist erlaubt.
Einschränkung	: Intern werden die Elemente einer %-Liste durch das gesetzte Bit 7 im letzten Byte getrennt. Deshalb können in %-Listen keine Zeichen mit einem Code größer als 127 gespeichert werden. Dies betrifft insbesondere die BASIC-Befehle wie "LET" und die User Defined Graphics. Sollen auch solche Zeichen gespeichert werden, muß eine $-Liste eingerichtet werden.

%-Listen können auch aus bestehenden $-Listen erzeugt werden. Dabei werden alle abschließenden Leerzeichen entfernt.

Format: OPEN# a$

Dieser Befehl erzeugt aus dem BASIC-Feld a$ das komprimierte Feld a%. Der neue Name besteht immer aus dem Kleinbuchstaben des $-Feldes. Der Name a$ stent anschließend wieder zur Verfügung, denn dieses Feld existiert für BASIC nach dem OPEN#-Befehl nicht mehr.

Dieser Befehl wird nur akzeptiert, wenn noch keine Liste mit dem neuen Namen besteht.

3) #-Listen

Binäre Listen werden ebenfalls mit dem "DEF FN"-Befehl eingerichtet:

Bsp.: DEF FN #a16 richtet die #-Liste #a mit 16 Elementen ein.

Für Name und Anzahl gelten die gleichen Angaben wie bei %-Listen.

Zusätzlich können die #-Listen mit Namen #0 und #1 für die Suchroutine vereinfacht erstellt werden:

FORMAT: TAB Name (Name steht für den Namen einer beliebigen Liste)

Dieser Befehl stellt zunächst die Anzahl der Elemente in der Liste 'Name' fest und richtet dann die beiden Listen #0 und #1 mit genau dieser Anzahl von Elementen ein. Dabei sind alle Elemente auf 1 gesetzt und können als Basis für Suchroutinen in der Liste 'Name' benutzt werden (wird beim Suchbefehl näher erläutert).

4) Ergänzen von Listen

Mit dem Befehl '+' können alle Listen um neue Elemente ergänzt werden.

FORMAT	: +Name
Name kann sein	: Der Name jeder existierenden Liste.
Bsp.	: +a%

Der Inhalt des neuen Elements wird bei den Zeichenlisten (%- und $-Listen) aus dem String k$ kopiert, bei den Zahlenlisten (&-Listen) aus der Variablen k.

#-Listen erhalten als neues Element eine '1', wenn die Variable k beim Aufruf ungleich Null ist; ist k gleich Null, wird das neue Element ebenfalls '0'.

5) Löschen von Listen

Alle vier Arten von Listen können mit ERASE gelöscht werden; sie werden tatsächlich gelöscht und nicht etwa mit Leerzeichen oder Nullen überschrieben!

FORMAT : ERASE Name

Name kann sein: der Name jeder existierenden Liste. Ein Beispiel wurde schon im Abschnitt "Aufruf" gegeben.

b) Ein- und Ausgabebefehle

Die %- und #-Listen sind für BASIC-Befehle nicht zugänglich. Allerdings werden sie bei 'LOAD' und 'SAVE' mitgespeichert und von 'CLEAR' und 'RUN' gelöscht.

Um Elemente dieser Listen ein- und auszugeben, werden besondere dk-Befehle benutzt.

Obwohl die daneben verwalteten reinen BASIC-Felder mit LET oder PRINT (BASIC-Befehle) erreichbar sind, können die in diesem Abschnitt vorgestellten Befehle auch auf diese BASIC-Listen angewendet werden. Das vereinfacht die Handhabung, wenn in Schleifen mehrere verschiedene Listenarten bedient werden sollen.

Alle Ein- und Ausgabebefehle bedienen sich der Variablen k und k$. Aus diesem Grund dürfen diese Variablen nicht gleichzeitig für andere Zwecke benutzt werden!

1) Der Befehl TO ordnet einem Listenelement einen Wert zu, der je nach Listenart aus k oder k$ kopiert wird:

FORMAT	: TO Name Nummer
Name kann sein	: Jeder Name einer existierenden Liste
Nummer kann sein	: Die Nummer eines existierenden Elements der angesprochenen Liste. Es gelten die gleichen Möglichkeiten wie für 'Anzahl' bei der Einrichtung von Listen.

Bsp. 2:

Dem Element Nr. 1 der Liste a% soll der Inhalt "a%1" zugeordnet werden. Zunächst muß allerdings die Liste a% definiert werden:

```
RANDOMIZE USR dk: REM DEF FN a%1
```

Die Liste a% besteht nun aus einem Element (einem Leerzeichen).

```
LET k$="a%1"     (k$ wird vorbereitet)
RANDOMIZE USR dk:REM TO a%1
```

Der Inhalt von k$ wird in das Element a%1 kopiert.

Nun können Sie mit einer kleinen BASIC-Schleife weitere Elemente anfügen oder bisherige ändern:

```
10 PRINT "Anzahl der Elemente in a%: ";USR dk:REM LEN a%
20 INPUT "Neues Element (n) /altes Element ändern (a) ";k$: IF k$="n" THEN LET
z$="+a%": GOTO 50
30  IF k$<>"a" THEN PRINT "?": GOTO 20
40  INPUT "Welche Nr? ";x: LET Z$="TO a%x"
50  INPUT "Text: ";k$
60  RANDOMIZE USR dk: GOTO 10
```

Einige Erklärungen:
- Dieses Programm mit GOTO 10 starten und mit BREAK oder STOP im Input beenden.
- In der Zeile 10 wird dk durch den PRINT-Befehl aufgerufen.
- In Zeile 20 und 40 wird je nach Eingabe z$ mit dem gewünschten dk-Befehl 'geladen', der letztlich in Zeile 60 ausgeführt wird.
- Der Inhalt von k$ wird in die neuen/alten Elemente kopiert.
- Die Fehlermeldung 'Out of Data' erhalten Sie, wenn Sie in Zeile 40 eine Nummer eingeben, die noch nicht existiert; GOTO 40 hilft weiter. Über den LEN-Befehl könnte man in diesem Fall vor Aufruf des Befehls 'TO a%x' testen, ob die gewünschte Nummer erlaubt ist.

Dieses Beispiel kann für andere Listenarten im gleichen Sinne verwendet werden. Beachten Sie aber, daß bei #- und &-Listen (letzteres sind BASIC-Zahlenfelder) der Inhalt der Elemente aus der Variablen k kopiert wird.

2) Der Befehl READ ist die Umkehrung des TO-Befehls:

```
FORMAT                  : READ Name Nummer
                          (Name und Nummer wie bei TO)
```

Das angesprochene Element wird je nach Listenart nach k oder k$ kopiert; bei #-Listen kann der Inhalt (0/1) direkt mit 'PRINT USR dk: REM READ #Name Nummer' ausgegeben werden.

Ergänzen Sie das Bsp. 2 um folgende Zeilen:

```
45  RANDOMIZE USR dk: REM READ a%x
46  PRINT "alt: ";k$
```

Nun wird vor der Änderung eines Elements der alte Inhalt ausgegeben.

Achtung:

dk bietet zwei Hilfen, um die Ein-/Ausgabe von Elementen in Schleifen für verschiedene Listenarten zu erleichtern: Sie können mit dem dk-Befehl VAL$ (dieselbe Verfahrensweise wie bei LEN) testen, ob sich in k$ eine (positive) Zahl befindet und mit dem BASIC-Befehl PEEK 23728, ob READ eine Zeichenkette oder eine Zahl gelesen hat. Bei VAL$ ist das Ergebnis 1, wenn in k$ eine Zahl ist (sonst 0) und bei PEEK 23728 ist das Ergebnis 0, wenn durch READ eine Zahl gelesen wurde (sonst 1).

Bsp. 3:

Ein kleines Dateiprogramm, das später ergänzt wird.

Löschen Sie bitte zunächst folgendermaßen das letzte Beispiel und geben Sie ein:

```
RANDOMIZE USR dk: REM c (ergibt Fehlermeldung c und löscht das BASIC-Programm)
CLEAR: LET dk=62200: LET k=0: LET k$="": LET z$=""
(NEW würde auch die Programmzeile 0 löschen.)
```

Im folgenden Programm werden drei Listen benutzt, um Name, Geburtstag und Geschlecht zu speichern. Diese Listen werden zunächst eingerichtet und mit einem ersten Element geladen:

```
10  RANDOMIZE USR dk: REM DEF FN a%1: REM DEF FN #a1
20  DIM a$(1,6)
30  LET k$="Schneider Hans": RANDOMIZE USR dk: REM TO a%1
40  LET a$(1)="121061"
50  LET k=0: RANDOMIZE USR dk: REM TO #a1
```

Die Namen werden also in a%, die Geburtstage in a$ und das Geschlecht mit 0/1 für m/w in der binären Liste #a gespeichert.

Die internen Listennamen (a% usw.) werden nun in einer eigenen Liste x$ gespeichert; dies ermöglicht später einen automatischen Aufruf:

```
60  DIM x$(3,2): LET x$(1)="a%": LET x$(2)="a$": LET x$(3)="#a"
```

Die Label (Name usw.) werden ebenfalls gespeichert; hierzu dient die Liste n%:

```
 70  RANDOMIZE USR dk: REM DEF FN n%3
 80  DATA "Name: ","Geburtstag: ","m/w (0/1): "
 90  RESTORE: FOR x=1 TO 3: READ k$
100 RANDOMIZE USR dk: REM TO n%x
110 NEXT x
```

Das bisherige Programm dient nur dem Aufbau der später benötigten Listen; nach 'GOTO 10' könnten die Zeilen 10 bis 110 wieder gelöscht werden.

Es folgt ein Menuprogramm, mit dem die Optionen

- Neuer Eintrag
- Einzelausgabe
- Ausgabe aller Einträge

aufgerufen werden können. Die Korrektur von Einträgen soll nach jeder Ausgabe möglich sein. Das Menü beginnt mit der Anzeige der bisherigen Anzahl von Einträgen:

```
200 LET ins=USR dk: REM LEN a%
205 PRINT "insgesamt ";ins;" Eintraege"
210 INPUT "Neu (n)/Einzeln (e)/Alle (a)? ";k$
220 IF k$="n" THEN GOTO 300
230 IF k$="e" THEN GOTO 400
240 IF k$="a" THEN GOTO 500
250 IF k$<>" STOP " THEN GOTO 210
260 STOP: GOTO 210
```

Es folgt der Teil "Neueintrag".

```
300 LET k$="_": LET k=0: REM Mit diesen Daten werden alle Listen zunaechst um ein
Element ergänzt
310 FOR x=1 TO USR dk: REM LEN n%
315 LET z$="+"+x$(x): RANDOMIZE USR dk: NEXT x
316 REM z$ erhaelt hier den '+'-Befehl nacheinander für alle vorhandenen Listen
320 LET ins=ins+1: LET ausg=ins: GOTO 450
```

Der Neueintrag besteht bislang nur aus "_" in den $- und %-Listen und "0" in der #-Liste. Um die neuen Elemente mit den gewünschten Einträgen zu versehen, erfolgt ein Sprung in den Teil "Ausgabe": dort wird der noch leere Neueintrag angezeigt und zur Korrektur angeboten.

Hier der Abschnitt "Einzelausgabe".

```
400 INPUT "Welche Nr.? ";k: LET k= INT k: IF k>ins OR k<1 THEN GOTO 200: REM Fal-
sche Nummern werden abgeblockt
410 LET ausg=k
440 REM Ausgabe aller Eintraege mit den Nummern von ausg bis ins:
450 FOR x=ausg TO ins
460 FOR y=1 TO USR dk:REM LEN x$
465 REM Dies ergibt die Anzahl der Listen!
470 RANDOMIZE USR dk: REM READ n%
475 PRINT k$;" ";: REM Schreibt ein Label
480 LET z$="READ "+x$(y)+"x"
482 RANDOMIZE USR dk: IF NOT PEEK 23728 THEN LET k$=STR$ k
484 REM Wenn PEEK 23728=0 ist, wurde vom READ-Befehl eine Zahl nach k ausgegeben
485 PRINT k$: REM Ein Element wird ausgegeben
490 NEXT y: REM Das nächste Label+Element
495 PRINT "       ": INPUT "Korrektur? (j/Enter)";k$
497 IF k$="j" THEN GOSUB 600: GOTO 495
498 IF k$=" STOP " THEN GOTO 200: REM Abbruch
499 NEXT x: GOTO 200: REM Der naechste Fall; nach letztem Fall zurueck zum Menu
500 LET ausg=1: GOTO 450: REM Ausgabe aller Fälle
600 REM Korrekturteil: x ist immer die Fall-Nummer
610 FOR y=1 TO USR dk: REM LEN n%
620 RANDOMIZE USR dk: REM READ n%y
625 REM Die Label werden nacheinander gelesen und im folgenden INPUT vorgegeben
630 INPUT (k$);k$: IF k$=" STOP " THEN RETURN
635 IF k$="" THEN GOTO 660:  REM Leereingabe belaesst den Eintrag unveraendert
640 IF x$(y,2)<>"&" AND x$(y,1)<>"#" THEN GOTO 650
642 REM Wenn eine Zahl als Eingabe verlangt wird, prueft die naechste Zeile k$
645 LET k=USR dk: REM VAL$
646 IF NOT k THEN PRINT "ZAHL!": GOTO 630
647 LET k=VAL k$
650 LET z$="TO "+x$(y)+"x":RANDOMIZE USR dk
660 NEXT y: RETURN
```

Dieses Programm kann mit GOTO 200 gestartet werden. Die Eingabe von STOP bricht Unterprogramme ab oder beendet im Menu das gesamte Programm.

Sie können die Datei natürlich um weitere Listen wie 'Straße', 'Ort' etc. ergänzen:

Definieren Sie zunächst mit DIM oder DEF FN und freien Namen diese Listen; beachten Sie aber, daß die neuen Listen mit der aktuellen Anzahl der bisherigen Einträge dimensioniert werden. Bei 12 Einträgen also z.B. "RANDOMIZE USR dk: REM DEF FN b%12".

Soll in b% z.B. der Straßenname gespeichert werden, muß mit LET k$="Str.:": RANDOMIZE USR dk: REM +n% das neue Label in n% eingetragen werden.

c) Die Suchroutinen

Für die Erklärung der nächsten Befehle soll ein neues Beispiel dienen. Speichern Sie deshalb das letzte Programm auf Kassette (SAVE "Bsp. 3") und bereiten Sie das neue Beispiel vor:

```
RANDOMIZE USR dk: REM c    (Fehlermeldung C)
CLEAR: LET dk=62200: LET k$="": LET k=0: LET z$=""
```

Bsp. 4: Suchen + Manupulation binärer Listen

Zunächst wird eine kleine Datei aufgebaut, in der 10 Namen und Geburtsjahre stehen:

```
10  DIM a$(10,10): RESTORE: FOR x=1 TO 10: READ a$(x): NEXT x
20  DATA "Hans","Erika","Ute","Bernd","Dieter","Heinrich","Otto","Anton","Antonia"
,"Michael"
30  RANDOMIZE USR dk: REM OPEN# a$
40  REM Aus a$ wird a%
50  DIM a$(10,2): FOR x=1 TO 10: READ k: LET a$(x)=STR$ k: NEXT x
60  DATA 60,52,64,59,68,57,64,66,66,71
70  RANDOMIZE USR dk: REM TAB a%
```

In Zeile 70 werden die binären Listen #0 und #1 eingerichtet; sie enthalten ebensoviele Elemente wie a% und dienen als 'Notizblatt' beim Suchen.

Der Rest des Programms dient der Demonstration der Suchroutinen; zunächst werden die Inhalte der verschiedenen Listen angezeigt: Die 'Notizlisten' #0 und #1 und die beiden eigentlichen Datenlisten a% (Namen) und a$ (Geburtsjahre). Danach soll dann ein Suchkommando eingegeben werden, dessen Ergebnis wieder angezeigt wird:

```
100 LET s0=USR dk: REM LEN #0
105 LET s1=USR dk: REM LEN #1
110 PRINT "#0: ";s0;"  #1: ";s1
112 REM Der LEN-Befehl ergibt bei #-Listen immer die Zanl der Elemente, die '1'
enthalten
120 FOR x=1 TO 10: REM Soviele Elemente sind vorhanden
130 RANDOMIZE USR dk: REM READ a%x
132 REM In k$ steht nun jeweils ein Vorname aus a%
140 LET null=USR dk: REM READ #0x
145 LET eins=USR dk: REM READ #1x
147 REM In null und eins steht jeweils der Wert aus #0 und #1
150 PRINT TAB 3;null;" ";eins;"  ";k$; TAB 20;a$(x)
```

```
160 NEXT x:Print
200 INPUT "dk-Befehl; ";z$: PRINT "Befehl: ";z$
210 IF z$="" STOP " THEN STOP: GOTO 100
220 IF z$="" THEN GOTO 200
230 IF z$(1)="?" THEN INPUT "k$=";k$: IF k$="" THEN LET k$=" "
240  REM Wenn ein Suchbefehl eingegeben wird,  muß in k$ die Zeichenfolge  gespei-
chert werden, nach der gesucht oder mit der verglichen werden soll
250 RANDOMIZE USR dk: GOTO 100
252 REM Der Befehl in z$ wird ausgeführt
```

Nach GOTO 10 wird die 'Datei' angezeigt. Geben Sie nun einen ersten Suchbefehl ein:

"?a%=#0" und bei k$: "Otto"

Das könnte man lesen: Suche in der Liste a% die Werte, die gleich k$ sind und notiere das Ergebnis in #0.

Bei Fehlermeldungen haben Sie wahrscheinlich einen unkorrekten Befehl eingegeben. Nach "GOTO 10" sollte ein neuer Versuch möglich sein.

In der Liste #0 ist das Ergebnis vermerkt: vor 'Otto' steht eine '1' als Notiz für 'Gefunden'.

Geben Sie nun als Suchbefehl

"?a$=#1" und bei k$: "5" ein.

Das Ergebnis steht diesmal in der Liste #1.

Wie setzt sich der Suchbefehl zusammen?

FORMAT: ? Name Vergleich #Liste

'Name' steht also für die Liste, in der gesucht wird, '#Liste' für die Ergebnis- oder Notizliste.

Die #-Liste speichert aber nicht nur das Ergebnis der Suche, sondern ist zugleich auch das Verzeichnis der Elemente, die beim Suchen überhaupt berücksichtigt werden; geben Sie ein:

"?a%=#0" und für k$: "Dieter"

Weil das Element 'Dieter' vor der Suche in #0 nicht mit einer '1' markiert war, wird es nicht gefunden.

Um alle Elemente wieder mit '1' zu besetzen, geben Sie als dk-Befehl

"TAB a%" ein.

Als nächstes könnte man aus unseren 10 Kandidaten die heraussuchen, die in den 60er Jahren geboren sind:

"?a$=#0" und bei k$: "6"

Nun alle Namen, die mit Buchstaben kleiner als "G" beginnen:

"?a%<#1" und bei k$: "G"

Das Ergebnis ist unbefriedigend: Es wurde niemand gefunden! Obwohl der Befehl dem letzten Beispiel fast gleich ist, besteht ein wichtiger Unterschied:

In beiden Fällen wurde nach dem ersten Zeichen eines Strings gesucht, allerdings einmal in einer $-Liste, das andere Mal in einer %-Liste.

Bei $-Listen werden soviele Zeichen verglichen, wie k$ enthält; danach wird der Vergleich abgebrochen. So konnte dk im ersten Fall wie erwartet arbeiten.

Bei %-Listen besteht aber das Problem, daß die Elemente unterschiedlich lang sein können; würde dk hier ebenfalls bei k$="G" nach dem ersten Zeichen den Vergleich abbrechen, könnte ein Eintrag, der nur (!) aus einem "G" besteht nicht mehr alleine gefunden werden.

Deshalb gibt es für die %-Listen eine Ergänzung im Suchbefehl, der anzeigt, daß über die Länge von k$ hinaus weitere Zeichen unberücksichtigt bleiben sollen: hierzu dient der senkrechte Strich unter der Taste 'S', der an den Befehl angehängt wird.

Aber wie findet man in einer $-Liste den Eintrag "G" ?

Nun, wenn die Länge der Elemente der $-Liste größer als 1 ist, steht ja dort z.B. "G ". Das findet man mit k$="G ". Meist reicht natürlich zur Unterscheidung schon k$="G ". Ist k$ übrigens länger als die Elemente einer $-Liste, wird k$ entsprechend gekürzt.

Zurück zur Suche nach Namen, die mit Buchstaben <G beginnen: zunächst wieder "TAB a%" zum Zurücksetzen der #-Listen.

Dann "?a%<#0!" (kein ! sondern CAPS=SYMB SHIFT und noch einmal SYMB SHIFT + S)
und "G" bei k$

Dies führt zu dem gewünschten Ergebnis.

Sie können bei der Suche auch führende Zeichen ausblenden:

"?a$/1>#1" und "5" bei k$

Durch "/1" bleibt das erste Zeichen der Listenelemente unberücksichtigt; von den Geburtsjahren werden nur die Einerstellen verglichen. Statt 1 können Sie auch andere Zahlen oder Variable verwenden; es gelten die gleichen Vorschriften wie für "Zahl" im Befehl DEF FN.

Experimentieren Sie nun mit den verschiedenen Vergleichsoperationen gleich, größer gleich, kleiner etc.

b) Manipulation von binären Listen

Ergänzen Sie das letzte Beispiel um folgende Zeilen:

```
215 IF z$="b" THEN GOTO 300
300 REM Andere Anzeige von #0 und #1 zur Demonstration
310 PRINT ,"#0: ";
320 FOR x=1 TO 10
330 RANDOMIZE USR dk: REM READ #0x
340 PRINT k;: NEXT x
350 PRINT ,"#1: ";
360 FOR x=1 TO 10
370 RANDOMIZE USR dk: REM READ #1x
380 PRINT k;: NEXT x
400 INPUT "BIN-Befehl: ";z$: IF z$=" STOP " THEN GOTO 100
410 IF z$="" THEN GOTO 400
420 PRINT z$: RANDOMIZE USR dk: GOTO 310
```

Wenn Sie nun für "dk-Befehl" ein "b" eingeben, wird dieser neue Programmteil aufgerufen.

Starten Sie mit "GOTO 10" und suchen zunächst noch einmal alle 60er Jahrgänge:

"?a$=#1" und bei k$: "6"

Nun geben Sie "b" ein: die #-Listen werden beide angezeigt.

Der "BIN"-Befehl hat die Form

FORMAT : BIN #Liste1 Operand #Liste2
#Liste1 und #Liste2 müssen dabei die gleiche Anzahl von Elementen haben
Operand kann sein : =, INVERSE, AND, OR, x (für XOR)

Geben Sie als ersten BIN-Befehl ein:

"BIN #0=#1" (im BASIC-Sprachgebrauch könnte man hierzu sagen: LET #0=#1)

Die Liste #0 erhält den Inhalt von #1. Der Operand = kopiert also #-Listen.

Nun: "BIN #1 INVERSE #0"

Hier wird ebenfalls kopiert: von #0 nach #1, allerdings wird der Wert jeweils invertiert.
Sie können natürlich auch eine Liste direkt invertieren:

"BIN #1 INVERSE #1"

Die Operanden AND, OR und x für XOR verknüpfen die angesprochenen Listen mit diesen logischen Operanden; das Ergebnis steht immer in der erstgenannten Liste.

AND ergibt dann eine '1', wenn in beiden Listen eine '1' steht
OR ergibt '1', wenn mindestens in einer Liste eine '1'steht
XOR ausschließlich dann, wenn nur in einer der beiden Listen eine '1' steht.

Überlegen Sie, was "BIN #1x#1" ergibt? Überprüfen Sie Ihre Antwort.

Mit diesen Verknüpfungen sind verschachtelte Suchroutinen sehr leicht durchzuführen; allerdings ist dies an unserem kleinen Datei-Beispiel nicht so ergiebig. Besser geeignet hierfür ist das mitgelieferte BASIC-Programm "Datenpflege" in Verbindung mit der Beispieldatei. Im entsprechenden Abschnitt der Anleitung finden sich Anregungen.

Dennoch ein kleines Beispiel: Suchen Sie die Kandidaten, die entweder nach 1960 geboren sind oder aber im Alphabet nach "I" stehen! (Wozu auch immer das gut sein soll ...)

Geben Sie aber zunächst " STOP " ein, damit Sie wieder in den anderen Programmteil kommen. Die BIN-Befehle können Sie auch dort eingeben.

Lösungsvorschlag: 1. "TAB a%" zum Zurücksetzen
2. "?a$>#1" und k$="60"
3. "?a%>#0!" und k$="I" (kein !)
4. "BIN #0 OR #1"
Das Ergebnis steht in Liste #0.

Es gibt einen speziellen Befehl, mit dem alle Fälle einer Datei schnell ausgegeben werden können, die in einer #-Liste mit '1' eingetragen sind:

FORMAT : DATA &Liste #Liste
(&-Listen sind Zahlenfelder)

Dieser Befehl schreibt in die &-Liste die Nummern der Elemente, die in der #-Liste den Wert '1' haben. Daher muß die &-Liste (mindestens) soviele Elemente haben, wie Einsen in der #-Liste vorhanden sind.

Bsp.:

Ergänzen Sie das letzte Programm mit

```
216 IF z$="a" THEN GOTO 600:  REM "a" ruft den neuen Programmteil auf
600 PRINT "Ausgabe nach #-Listen"
610 INPUT "Name der #-Liste : ";k$: IF k$=" STOP " THEN GOTO 100
620 IF k$>"#1" OR k$<"#0" THEN GOTO 610:REM Blockt flasche Eingaben ab
630 LET z$="LEN"+k$:LET k=USR dk: IF NOT k THEN PRINT "keine Einträge": GOTO 610
640 DIM z(k): REM Die &-Liste wird eingerichtet
650 LET z$="DATA z&"+k$: REM k$ ist noch immer der Name der #-Liste
660 RANDOMIZE USR dk
670 REM In z stehen nun die gewuenschten Nummern
680 LET z$="READ a%k"
700 FOR x=1 TO k: REM k ist die Anzahl der Elemente
710 PRINT z(x);" ";: LET k=z(x): RANDOMIZE USR dk:Print k$
720 NEXT x: GOTO 600
```

Anmerkung:
In Zeile 630 wird in z$ der LEN-Befehl zusammengestellt, der die Anzahl der vorhandenen Einseinträge feststellt. Ist diese Anzahl Null, würde der DIM-Befehl in Zeile 640 zu einer Fehlermeldung führen. Zeile 660 führt den DATA-Befehl aus, der wieder in z$ steht; eine direkte Benennung in einem REM-Statement ist hier nicht möglich, der Name der angesprochenen #-Liste ändert sich ja und steht in k$. Die Schleife ab Zeile 700 gibt dann die Namen aus, deren Element-Nummern in der Liste z& stehen.

Rufen Sie den Programmteil mit "a" auf und lassen Sie sich nach verschiedenen #-Listen die Datei ausgeben.

e) Sortierroutinen

dk kann $- und %-Listen sortieren. Die Geschwindigkeit ist recht beachtlich: 100 Namen werden in deutlich weniger als einer Sekunde sortiert.

```
FORMAT          : CAT NAME Liste&
                  'Name' ist der Name einer $- oder %-Liste
                  'Liste&' ist der Name einer &-Liste, die als Indexliste
                  dient und die gleiche Elementzahl wie 'Name' haben muß.
```

Saven Sie auch das letzte Beispiel und testen Sie die Sortierleistung von dk:

```
RANDOMIZE USR dk: REM c   (Fehlermeldung C)
CLEAR: LET dk=62200: LET k$="": LET k=0

 10 DIM a$(1,10):LET a$(1)="test"
 20 LET k$=""
 30 FOR x=1 TO 4+RND*6: LET k$=k$+CHR$ INT (26*RND+97):NEXT x
 40 PRINT k$, USR dk: REM +a$
 50 IF INKEY$ <>" STOP " THEN GOTO 20:REM Bis Sie STOP eingeben (oder der Speicher
voll ist) erzeugt diese Schleife Zeichenketten und speichert sie in a$: die Zahlen
rechts sind immer um 1 größer als die aktuelle Anzahl von Elementen in a$
 70 LET k=USR dk:REM f
 80 LET x=USR dk: REM LEN a$
 90 IF x*5>k THEN PRINT "Speichermangel?"
100 DIM z(x)
110 PRINT "Nach TASTE startet das Sortierprogramm!"
120 PAUSE 0
130 RANDOMIZE USR dk: REM CAT a$z&
140 BEEP .2,.2: CLS :PRINT "FERTIG"
150 PRINT "alte Liste", "sortierte Liste"
160 FOR x=1 TO x
170 PRINT a$(x),a$(z(x))
180 NEXT x
200 INPUT "Mehr? (j)";k$: IF k$="j" THEN GOTO 20
```

Zeile 70 fragt den restlichen Speicherplatz ab, um die Zeile 90 zu testen, ob noch genügend Platz für das Feld z vorhanden ist.

Die Suchroutinen verändern nichts an den Listen, die 'sortiert' werden. Es wird lediglich eine Zahlenliste erstellt, die im ersten Element die Nummer des kleinsten Elements und in ihrem letzten Element die Nummer des größten Elements angibt.

Warum eine Indexliste?

Das Sortieren mit einer Indexliste hat verschiedene Vorteile; der wesentlichste besteht darin, daß die ursprüngliche Reihenfolge der Liste erhalten bleibt. Stellen Sie sich vor, in einer Datei mit Namen, Anschrift etc. würde tatsächlich die Liste "Name" sortiert. Das wäre nur dann vertretbar, wenn parallel alle anderen Listen ebenfalls in die neue Reihenfolge gebracht würden, ein erheblicher Aufwand. Daneben ginge die Reihenfolge der Eingabe verloren, die Fall-Nummern müßten zusätzlich in einer Liste gespeichert werden oder würden sich laufend ändern.

Das Prinzip der Indexliste in einem kleinen Beispiel:

Liste a$, die sortiert wird	Indexliste i
Hans	4
Otto	1
Klemens	3
Alfred	2

In i (1) steht die Adresse des kleinsten Elements von a$: Alfred. Um a$ sortiert auszugeben, wendet man folgende Schleife an:

```
FOR x=1 TO 4
PRINT a$(i(x))
NEXT x
```

f) Befehle, die BASIC-Programmteile manipulieren

1) "c": Löschen des Programms:

FORMAT : c
Funktion : Dieser Befehl löscht alle BASIC-Programmzeilen. Lediglich die 0-Zeilen bleiben erhalten, denn sie sind für die hier folgenden Befehle nötig. Sinn ist tatsächlich weniger das Löschen als die Bereitstellung der 0-Zeilen.

2) "LINE": Kopie einer BASIC-Zeile von z$ in eine 0-Zeile

FORMAT : LINE
Funktion : Der Inhalt von z$ wird ohne Prüfung auf korrekte Syntax als BASIC-Zeile in die Zeile 0 kopiert.

Bsp.:

Versuchen Sie einmal, ein BASIC-Programm zu schreiben, das folgende Aufgabe erfüllt:

Mit einem INPUT-Befehl soll der Benutzer angeben, welche Variable z.B. ausgeschrieben werden soll; etwa in dieser Art:

```
INPUT "Name der Variablen?";k$
PRINT  ???(Name steht in k$)
```

Sie werden feststellen, daß eine solche Aufgabe nur sehr umständlich lösbar ist, ein passender BASIC-Befehl existiert nicht.

Lösung mit dk:

```
INPUT "Name der Variablen?;k$
LET z$="PRINT"+k$
RANDOMIZE USR dk: REM LINE
GOSUB 0
```

Schauen Sie sich die erste 0-Zeile mit LIST an; sie bietet Platz für 20 Zeichen, der RETURN- Befehl ist schon eingebaut und wird nicht verändert.

ACHTUNG:

Vermeiden Sie unbedingt Zahlen in z$: diese werden in Programmzeilen anders gespeichert, als in einer Zeichenkette.

Geben Sie also statt

```
LET z$="DIM a$(10,10)"   (falsch!)
```

besser

```
LET x=10: LET z$="DIM a$(x,x)"
```

ein. Nach dem LINE-Befehl und GOSUB 0 wird der kopierte Befehl korrekt ausgeführt.

3) FN a und FN b: automatische Änderung der DEF FN-Statements in den 0-Zeilen.

FORMAT : FN a oder FN b
Funktion : In die DEF FN-Statements der 0-Zeilen wird der Inhalt von z$ kopiert. Dies muß ein dk-Befehl sein, der danach mit dem BASIC-Befehl
RANDOMIZE FN a(1) oder FN b(1)
aufgerufen werden kann.
Bsp. : LET z$="LEN a%"
RANDOMIZE USR dk: REM FN a

Danach wird jeder Aufruf von FN a(1) (BASIC!) die Länge von a% ergeben. Die Zahl 1 ist beliebig, platzsparender ist FN a(PI).

Wenn Sie sich die 0-Zeilen mit LIST anschauen, sehen Sie, daß das REM-Statement des erwareten dk-Befehls schon vorhanden ist.

g) Hilfsroutinen

1) LEN: Anzahl der Elemente in einer Liste

FORMAT : LEN Name
Name muß der Name einer existierenden Liste sein.
Der LEN-Befehl schreibt das Ergebnis auch nach k:
PRINT USR dk: REM LEN a%
ist gleich
RANDOMIZE USR dk: REM LEN a%
PRINT k

ACHTUNG:

Bei binären Listen (#-Listen) ergibt LEN nicht die Gesamtzahl, sondern die Anzahl der mit '1' notierten Fälle.

2) VAL$: prüft k$ auf Zahl

FORMAT : VAL$
Der Aufruf ergibt den Wert '1', wenn in k$ eine positive Ziffernfolge steht, in der max. ein Dezimalpunkt vorkommt. Ist dies nicht der Fall, hat VAL$ den Wert Null.

Bsp.:

Um die Eingabe eines STOP durch den Benutzer zu verhindern, soll statt

```
10 INPUT k          der Befehl
10 INPUT LINE k$    verwendet werden, um danach mit
20 LET k=VAL k$     weiterzuarbeiten.
```

Vor dem letzten Befehl wird nun zur Vermeidung der Fehlermeldung VAL$ aufgerufen:

```
12 LET k=USR dk: REM VAL$
13 IF NOT k THEN GOTO 10
```

Übersicht: Alle dk-Befehle

Die dk-Befehle stehen entweder im String z$ oder in einem REM-Statement unmittelbar hinter dem Aufruf des MC-Programms.

```
(LET dk=62200)
LET z$="dk-BEFEHL": RANDOMIZE USR dk
```

oder

```
RANDOMIZE USR dk: REM dk-Befehl
```

Statt

```
'RANDOMIZE' kann auch 'LET wert = USR dk'
                  oder 'PRINT USR dk'
```

verwendet werden.

Zusätzlich kann ein dk-Befehl mit

```
RANDOMIZE FN a(1) oder FN b(1)
```

aufgerufen werden, wenn zuvor die FN-Funktion mit einem dk-Befehl 'geladen' wurde:

```
LET z$="dk-Befehl": RANDOMIZE USR dk: REM FN a
                    RANDOMIZE USR dk: REM FN b
```

Der dk-Befehl 'FN a' (bzw. FN b) kopiert den Inhalt von z$ in die REM-Statements der DEF FN - Befehle in den 0-Zeilen des BASIC-Programms.

Befehl	Funktion
DEF FN Name Anzahl	Richtet eine Liste mit 'Anzahl' Elementen ein.
Bsp.	: DEF FN v%12 DEF FN #w120
Vor.	: Eine Liste mit dem genannten Namen darf noch nicht vorhanden sein.
OPEN# Name$	Wandelt eine $-Liste in eine %-Liste um.
Bsp.	: OPEN# a$ Neue %-Liste: a%
Vor.	: a% darf noch nicht existieren; Zeichen in a$ mit einem Code>127 werden durch ein ? ersetzt.
ERASE Name	Löscht die Liste 'Name' vollständig.
Bsp.	: ERASE #3
+NAME	Ergänzt die Liste 'Name' um ein neues Element, das bei Textlisten aus k$, bei Zahlenlisten aus k kopiert wird.
Bsp.	: LET k$="neu": RANDOMIZE USR dk: REM +a% Die Dimension 'Anzahl' des angesprochenen Feldes wird automatisch um eins erhöht.
TO Name Nummer	Kopiert den Inhalt von k$ (oder k) in die Liste 'Name' an die Stelle 'Nummer'
Bsp.	: LET k$="a%17":RANDOMIZE USR dk:REM TO a%17

READ Name Nummer — Der Inhalt des Elements 'Nummer' in der Liste 'Name' wird nach k$ (oder k) kopiert

Bsp. : READ a%17

TAB Name — Löscht die binären Listen #0 und #1 (falls vorhanden) und richtet sie erneut mit der Anzahl ein, die der Anzahl der Elemente in der Liste 'Name' entspricht. Dabei werden alle Elemente von #0 und #1 auf den Wert eins gesetzt; diese Listen können somit schnell als Grundlage einer Suchroutine in 'Name' eingesetzt werden.

? Name OP #Liste — Suchbefehl: Vergleiche alle Elemente der Liste 'Name' mit k$ entsprechend dem eingesetzten Vergleichsoperanden Op. Berücksichtigt werden nur Einträge, die zuvor in der Liste '#Liste' mit '1' eingetragen sind. Alle Elemente in 'Name', die die Bedingung erfüllen, werden in '#Liste' mit '1' notiert.

Op kann sein : gleich, größer, kleiner, größer gleich, kleiner gleich, ungleich

Bsp. : LET k$="profisoft"
RANDOMIZE USR dk: REM TAB a%
LET gefunden= USR dk: REM ?a%=#0
'gefunden' ist die Anzahl der Einträge 'Profisoft' in der Liste a%. Alle diese Einträge haben in der Liste #0 anschließend den Wert eins.

? Name /n Op #Liste — Wie oben, allerdings werden die ersten n Zeichen der Elemente beim Vergleich mit k$ nicht berücksichtigt.

? Name% Op #Liste ! — (! steht für CAPS + SYMB SHIFT und SYM SHIFT + S)
Bei %-Listen kann durch den senkrechten Strich angegeben werden, daß Zeichen in der Suchliste, die über die Länge von k$ hinausreichen, nicht berücksichtigt werden sollen.

CAT Name Name& — Sortieren mit Indexliste
Legt in der Zahlenliste Name& die alphabetische Reihenfolge der Elemente der Liste Name an.

Vor. : Name& muß (mindestens) ebensoviele Elemente haben wie 'Name'. 'Name' muß eine $- oder eine %-Liste sein.

Bsp. : DIM z (Anzahl in a$)
RANDOMIZE USR dk: REM CAT a$z&
Ausgabe der 'sortierten' Liste:
FOR x=1 TO (Anzahl der Elemente in a$)
PRINT a$(z(x))
NEXT x

DATA Name& #Liste — Notiert in der Zahlenliste Name& die Nummern der Elemente in #Liste, die den Wert 'eins' haben.

Bsp. : DATA a&#0

Vor. : Die Liste a& muß ebensoviele Elemente haben, wie in #0 Einsen vorhanden sind.
Dieser Befehl dient der schnellen Ausgabe von Listen, die in einer #-Liste mit 0 oder 1 markiert sind.

BIN #List1 Op #List2 — Logische Verknüpfung von binären Listen.
Op kann sein : =, INVERSE, AND, OR, x (für XOR)
Vor. : In #List1 und #List2 müssen die gleiche Anzahl von Elementen gespeichert sein.
Die BIN-Operationen können auch mit nur einer Liste durchgeführt werden, die dann zweimal benannt wird. Das Ergebnis steht immer in der erstgenannten Liste.

LINE — Kopiert den Inhalt von z$ (max. 20 Zeichen) ohne Prüfung der Syntax als BASIC-Befehl in die erste 0-Zeile. Der Befehl kann anschließend mit "GOSUB 0" ausgefünrt werden.
Bsp. :
```
LET z$="DIM a$(k)":LET k=10
RANDOMIZE USR dk: REM LINE
GOSUB 0
```

FN a und FN b — Kopiert den Inhalt von z$ (max. 20 Zeichen) in die REM-Statements der DEF FN -Befehle in den 0-Zeilen.
Bsp. : LET z$="LEN a%": RANDOMIZE USR dk: REM FN a
Jeder BASIC-Aufruf von FN a ergibt als Wert nun den Wert des dk-Befehls 'LEN a%'.

s0 und s1 — s1 richtet eine Zeichenkette a$ ein, in der alle z.Zt. vorhandenen Variablen gespeichert werden.
Nach 'RANDOMIZE USR dk: REM s1' kann mit "SAVE "Name" DATA a$()" der komplette Inhalt der Variablen gespeichert werden.

ACHTUNG! — Nach s1 existiert nur noch die Zeichenkette a$. Auch die Variable dk ist für BASIC nun nicht vorhanden.
Um die Variablen wieder aus a$ zu befreien muß der Befehl s0 mit
RANDOMIZE USR 62200: REM s0
eingegeben werden.
Dies gilt auch, wenn a$ geladen wurde! Vor dem Laden der 'Sammelvariablen' sollte mit CLEAR der Speicher geräumt werden, um nach LOAD "" DATA a$() und USR 62200:REM s0 Doppelungen zu vermeiden.

c — Löscht das BASIC-Programm bis auf die 0-Zeilen; die Variablen können mit CLEAR gelöscht werden.
Bsp. :
```
RANDOMIZE USR dk: REM c
```
Nach 'c' erscheint die Fehlermeldung C.

LEN Name — Ergibt die Anzahl der Elemente in der Liste 'Name', bei #-Listen abweichend die Anzahl der Elemente mit dem Wert eins.
Bsp. :
```
PRINT USR dk: REM LEN a$
```

f — Ergibt die Anzahl der freien Bytes zwischen 'E LINE' und 'RAMTOP', also den verfügbaren Raum für weitere Einträge.
Es wird (wie vom Spectrum-Betriebssystem) ein zusätzlicher Bedarf von 50 Byte als Reserve abgezogen.
Bsp. :
```
PRINT USR dk: REM f
```

VAL$ Ergibt dann den Wert eins, wenn in k$ eine Zeichenkette existiert, die nur aus Ziffern und max. einem Dezimalpunkt besteht. In jedem anderen Fall ergibt sich der Wert null. VAL$ kann verwendet werden, um vor dem BASIC-Befehl "LET k=VAL k$" sicherzustellen, daß keine Fehlermeldung auftritt, wenn in k$ z.B. "abc" steht.

Bsp. :
```
10 INPUT LINE k$
20 LET k= USR dk: REM VAL$
30 IF NOT k THEN PRINT:"Zahl eingeben!": GOTO 10
40 LET k=VAL k$
```

Die vier Listenarten, die dk verwalten kann:

1) BASIC-Zeichenfelder der Art a$(Anzahl, Länge): $-Listen

Diese Listen werden in dk-Befehlen mit a$, b$ usw. bezeichnet; die Klammern - z.B. bei a$(14) - entfallen in dk-Befehlen.

2) BASIC-Zahlenfelder der Art a(Anzahl): &-Listen

Diese Listen werden in dk-Befehlen mit a&, b& usw. bezeichnet; Klammern werden ebenfalls weggelassen.

3) Komprimierte Zeichenfelder : %-Listen

Diese Listen sind den $-Listen ähnlich, jedoch wird keine einheitliche Länge der Elemente festgelegt.
In dk-Befehlen wird das $-Zeichen durch das %-Symbol ersetzt. In %-Listen können keine Zeichen mit dem CODE > 127 gespeichert werden.
Jedes mit der Tastatur erzeugbare Zeichen (außer #) kann Name einer %-Liste sein, Klein- und Großbuchstaben werden unterschieden.
%-Listen sind mit BASIC-Befehlen nicht erreichbar.

4) Binäre Listen: #-Listen

In binären Listen werden nur die Werte null oder eins als Elemente gespeichert. Ihren Namen wird das #-Zeichen vorangestellt, jedes mit der Tastatur erzeugbare Zeichen kann Name einer #-Liste sein.

"dk - NEW" und "dk - DATA" : Beispiel der Anwendung von dk

dk ist ursprünglich als reines Maschinencode-Programm und als Ergänzung des Spectrum-Betriebssystems konzipiert. Das mitgelieferte Dateisystem (BASIC-Rahmenprogramme) versteht sich daher eher als praktische Ergänzung und kann dem Benutzer sicherlich wertvolle Hinweise für eigene BASIC-Rahmenprogramme bieten.

Für die eigene Nutzung des MC-Programms gab der vorherige Teil der Anleitung die notwendigen Informationen.

Im vorliegenden Teil wird die Benutzung der Rahmenprogramme erklärt.

Laden Sie das Dateisystem mit LOAD ""

Nachdem das Ladeprogramm auch den MC-Teil geladen hat, können Sie zwischen drei Optionen wählen, die jeweils erklärt werden:

a) dk - NEW
b) dk - DATA
c) Nutzung nur des MC-Programms

Wählen Sie für diese Anleitung 'b' und starten Sie das Band erneut: Es wird das übernächste Programm der Kassette geladen.

Das Programm 'dk - DATA' meldet sich mit einer Auflistung der Befehle, die zur Verfügung stehen.

Die Befehle im Programm 'dk - DATA' werden mit ihren Namen eingegeben. Sie können die Befehle abkürzen; ist eine Abkürzung nicht eindeutig, verlangt das System weitere Zeichen.

Befehle verlangt das System mit der Aufforderung

'Gib Befehl:'

Das Programm wird zusammen mit dem Beispiel einer Datei geliefert, die Sie sich nun anschauen können.

Befehl: 'zalle' und 'ENTER' (= zeige alle)
Sie können auch abkürzen: za

Hat das System einen Befehl erkannt, bittet es um Bestätigung:

zalle OK? ´=Ja

Drücken Sie einfach ENTER, denn dies soll der Pfeil jeweils andeuten. Mit '´' wird immer angegeben, wie das System eine leere Eingabe interpretiert. Sie können auch 'j' für ja eingeben; jede andere Eingabe wird als 'nein' interpretiert.

Im Folgenden steht '´' immer für die Taste 'ENTER'.

Nach der Bestätigung von 'zalle OK?' durch 'j' oder ´ wird die erste Karteikarte der Datei aufgelistet.

Sie sollen nun angeben, ob in der angezeigten Karteikarte eine Korrektur gewünscht wird:

'Korrektur? ´=nein'

Geben Sie ´ ein und die zweite Karteikarte wird ausgegeben. Insgesamt enthält die Datei 15 Karteikarten. Wenn Sie die Auflistung abbrechen möchten, geben Sie

'STOP' ein (SYMBOL SHIFT und A)

Mit dieser Eingabe können Sie praktisch aus jeder Routine zum 'Gib Befehl' zurückkehren.

Falls Sie mit STOP zum 'Gib Befehl' zurückgekehrt sind, geben Sie nochmals 'zalle' ein.

Nun soll nämlich eine Korrektur durchgeführt werden. Antworten Sie also auf

'Korrektur?' mit 'j'

Als erstes wird 'Name' angeboten. Sie können wählen:

´ (=ENTER) läßt den eingetragenen Namen unverändert,
'Testmann Anton' als Eingabe ändert die Eintragung.

Nach ´ wird das nächste Label angezeigt. Label heißt nichts anderes als 'Überschrift' und ist der Name, mit dem im Programm die gespeicherten Daten unterschieden werden. Beispiel für die Label im vorliegenden Fall sind 'Name', 'Str.:' und '*'. Letzteres soll für 'Geburtstag' stehen.

Wenn Sie möchten, ändern Sie die Eintragungen der angezeigten Karteikarte. Sind alle Label abgefragt, erscheint

'OK?'.

Hier müssen Sie für 'ja' mit 'j' antworten, jede andere Taste wird als 'nein' interpretiert. ´ wird nicht akzeptiert: Dies dient als 'Bremse', wenn Sie im Eildurchgang die Label anschauen. Kehren Sie danach mit 'STOP' zum 'Gib Befehl' zurück. Mit der Eingabe '?' oder 'i' werden wieder die Befehle aufgelistet.

Was heißt 'z(eige)listen'?

Sie erinnern sich an den Begriff 'Label', mit dem die verschiedenen Einträge benannt wurden. Alle Einträge, die zu einem Label gehören, bilden im Speicher eine Liste. Also alle Namen, Straßen, usw.

Geben Sie 'zl' und ´´ ein: Das System fragt nun, welche Liste gezeigt werden soll. Sie können das Label, also den Namen der Liste, im Klartext eingeben. Abkürzungen sind wieder zulässig, bei Unklarheiten wird um Ergänzung gebeten.

Als erstes soll die Liste 'Name' ausgegeben werden: Beantworten Sie 'welche Liste' mit 'N' oder auch 'Name'.

Das Programm gibt nun - vorausgesetzt, es erkennt den Namen - die komplette Liste aus. In unserem Beispiel ist es wahrscheinlich unnötig, aber bei Listen mit sehr vielen Eintragungen können Sie mit 'STOP' die Auflistung abbrechen; allerdings sollten Sie STOP nicht als Antwort auf 'scroll?' eingeben, da dies zu einem unerwünschten BREAK führen würde. Hier hilft dann CONTINUE weiter.

Ist die Liste vollständig ausgegeben, kehrt das Programm zum 'Gib Befehl' zurück. Lassen Sie sich weitere Listen anzeigen!

Ähnlich wie 'zalle' funktioniert 'zeinzeln'. Geben Sie auf die Frage 'Nr?' die gewünschte Nummer der Karteikarte (im Programm Fall genannt) an, die Sie ausgeben möchten. Nach der Anzeige der gewählten Karte werden die nachfolgenden Karteikarten automatisch weiter angeboten; Sie können wie gewohnt mit 'STOP' bei 'Korrektur?' aus der Routine aussteigen.

'zsortiert' und 'zsuchliste' ist nur dann sinnvoll, wenn Sie zuvor die entsprechende Routine 'sortiere' oder 'suche' aufgerufen haben:

'sortiere' als Befehl ergibt die Nachfrage, anhand welcher Liste die Datei sortiert werden soll. Es ist wichtig zu wissen, daß sich das Sortieren nur auf die Ausgabereihenfolge bei 'zsortiert' auswirkt: die Listen selber bleiben immer in der alten Reihenfolge.

Geben Sie nach 'sortiere' und '' auf die Frage 'welche Liste?' wieder 'N' für 'Name' ein. Hat das Programm den Listennamen erkannt, wird unmittelbar sortiert. Dies geht recht schnell; an-schließend wird zum 'Gib Befehl' zurückgekehrt.

Nun führt 'zsortiert' '' zur Ausgabe der Datei entsprechend der alphabetischen Reihenfolge. Sie können auch nach 'PLZ' oder anderen Listen sortieren lassen.

Sortiert werden nur Listen, die Texte enthalten: Im vorliegenden Beispiel sind dies alle Listen außer 'm=0/w=1' und 'aktiv'. In diesen Listen ist immer nur eine Null oder eine Eins gespeichert. Die Listen 'Tel' und 'PLZ' sind entgegen dem ersten Anschein Textlisten und können sortiert werden.

dk kann vier Arten von Listen verwalten, die im Teil 'Einrichten von Dateien' erklärt werden.

Nun zum Befehl 'suche':

Die Suchroutinen sind sehr komfortabel, allerdings müssen Sie sich mit einigen Begriffen vertraut machen.

Zur Unterstützung der Suchroutinen werden sogenannte 'Suchlisten' eingesetzt: sie heißen

#3, #4 und #5.

Diese Suchlisten können Sie als Notizblock betrachten, auf dem das Ergebnis der Suche protokolliert wird.

Doch Theorie beiseite:

Rufen Sie das Suchprogramm mit 'su' '' auf. Sie sehen nun zunächst die insgesamte Zahl von Fällen in der Datei und hinter den Namen der Suchlisten jeweils verzeichnet, wieviele Fälle dort notiert sind: es sollten jeweils 15 sein.

Das System bietet nun zwei Wege an: 'Suchlisten verändern' oder 'weitersuchen'. Falls in den Suchlisten nicht die gleiche Zahl wie bei Insg steht, geben Sie bitte 's' und ´ und dann 1 und ´ ein: dies bewirkt das Zurücksetzen der Suchlisten und wird später noch erklärt.

Wählen Sie nun 'weitersuchen', indem Sie einfach ENTER eingeben.
Beantworten Sie die folgenden

Fragen ...	mit Eingabe von:	
welche Liste?	N	für Name
ab welchem Zeichen?	ENTER	für 1. Zeichen
Vergleichsart?	>	für größer
... wonach suchen?	N	für Suche nach 'N'
welche Suchliste?	#3	für Liste #3
... nachfolgende Zeichen	n	für nein
OK?	j	für ja

Nun wird in der Liste 'Name' nach Einträgen gesucht, deren erstes Zeichen größer als 'N' ist. Das Ergebnis dieser Suche ist in der Liste #3 mit der Anzahl der gefundenen Fälle verzeichnet.

Nach ´´ sind Sie wieder bei 'Gib Befehl' und können mit 'zsu' (zeige suchliste) die gefundenen Fälle ausgeben lassen. 'Welche Suchliste?' ist mit '#3 ' zu beantworten, denn dort sind die ausgesuchten Fälle notiert.

Es werden nun nacheinander alle Einträge angezeigt, für die die Suchbedingung erfüllt war.

Unterbrechen Sie mit 'STOP' bei 'Korrektur' und rufen Sie erneut das Suchprogramm mit 'su' auf.

In der Liste #3 sind immer noch die zuvor gefundenen Fälle notiert. Unter diesen sollen die gesucht werden, die nach 1960 geboren sind:

```
Liste                : * (so heißt die 'Geburtstagsliste')
ab welchem Zeichen?  : Geben Sie '5' ein, denn das Geburtsjahr steht hinter Tag
                       und Monat: 310752
Vergleichsart?       : >
womit vergleichen?   : 60
Suchliste:           : #3
OK?                  : j
```

Durch die erneute Wahl der Suchliste #3 werden nun nur die Fälle getestet, die hier eingetragen waren. Sie können sich das Ergebnis wieder mit ´´ und 'zsuchliste' '#3' ansehen. Kehren Sie aber wieder zum Suchprogramm zurück.

Wählen Sie nun den anderen Weg: s für 'Suchlisten ändern'.

Sie werden aufgefordert, einen Befehl einzugeben. Diese Befehle dienen dazu, die Suchlisten miteinander zu verknüpfen.

Alle diese Befehle haben die Form:

```
#Liste1   Operand   #Liste2        (Ergebnis: in Liste1)
```

'Operand' kann sein ... und bewirkt:

=	Kopie von #Liste2 nach #Liste1
INVERSE (Keyword)	wie =, aber Umtausch von notiert (1) und nicht notiert (0).
AND (Keyword)	notiert in #Liste1 alle Karteikarten, die in beiden Listen gemeinsam eingetragen sind.
OR (Keyword)	notiert in #Liste1 die Karteikarten, die mindestens in einer der beiden Listen stehen.
x (für XOR)	beläßt die Karteikarten in #Liste1, die in der einen oder der anderen Liste stehen, aber ohne die, die in beiden gleichzeitig stehen.

Daneben ist auch die Eingabe von '1' möglich: dies setzt alle Suchlisten auf die insgesamte Anzahl.

*** Das System überprüft hier nicht, ob der Verknüpfungsbefehl korrekt ist: bei unzulässigen Befehlen wird eine Fehlermeldung angezeigt. Sie werden aber darüber informiert, wie Sie ohne Schaden die Arbeit fortsetzen können: GOTO 200

Verwenden Sie aber als erste Liste immer eine der drei Suchlisten, denn hier wird das Ergebnis notiert.

Geben Sie nun (evtl. nach 'suche' und 's') als Befehle ein:

a) #3x#3	mit XOR wird jede Liste auf Null gesetzt
s	wieder zu 'Suchlisten verändern'
#3 INVERSE #3	nun sind alle 15 Elemente mit '1' gesetzt
s und #4=#3	kopiert Liste 3 nach Liste 4

b) Nun eine neue Suche:

Name der Liste	*	für Geburtstag
ab Zeichen ...	3	Zeichen 3 und 4 bilden den Monat
Vergleichsart	>=	
womit vergleich.?	11	sucht alle, die im Nov. oder Dez. geboren sind.
welche #-Liste	#3	
OK?	j	

Daneben sollen aber noch die im Januar Geborenen gesucht werden: hierzu wird die Suchliste 4 verwendet, um danach alle Geburtstagskinder von Nov. bis Jan. in einer Liste zusammen zu haben:

Name der Liste	*	
ab Zeichen ...	3	
Vergleichsart	=	für '='
womit vergleich.?	01	
welche #-Liste	#4	
OK?	j	

Nun die Verknüpfung mit 'oder':

s	ruft 'Suchlisten verändern' auf
#3 OR #4	kopiert nach Liste 3 alle Fälle, die in einer der beiden Listen verzeichnet sind.

'' und zsu(chliste) mit '#3' zeigt die gesuchten Fälle!

Experimentieren Sie mit den Suchroutinen und den Suchlisten! Bei unerwarteten Ergebnissen können Sie auf dem Bildprotokoll wahrscheinlich die Ursache ermitteln.

Wenn Sie in den Listen 'm=0/w=1' und 'aktiv?' suchen wollen, erhalten Sie einen Hinweis auf den Einsatz dieser Listen: sie sind genauso aufgebaut wie die Suchlisten und können mit diesen direkt verknüpft werden.

Die Liste m/w hat die interne Bezeichnung #6. Kopieren Sie also einfach #6 nach #3:

(evtl. 'su' und 's')
'#3=#6'

Danach sind in #3 alle Fälle mit dem Merkmal 'w' notiert. Eine Suche ist nicht notwendig. Achten Sie aber darauf, daß Sie immer als erstes den Namen einer Suchliste angeben, um die Veränderung der festen Listen zu vermeiden!

Einige Hinweise zu den Suchroutinen

Vielleicht haben Sie schon bemerkt, daß die Frage 'sollen nachfolgende Zeichen berücksichtigt werden?' nicht bei allen Listen auftaucht: z.B. bei der Suche nach 'PLZ'.

Woran liegt das?

Beim Einrichten einer Datei kann man für die Speicherung von Texten zwischen $-Listen für eine feste Länge aller Einträge (z.B. Länge 4 für PLZ) und %-Listen für Einträge mit unterschiedlichen Längen (z.B. für Name) wählen.

Die Suchroutinen können beide Arten von Listen bearbeiten. Dabei ergibt sich allerdings folgender Unterschied.

Grundsätzlicn werden zunächst jeweils soviele Zeichen der Elemente verglichen, wie Sie eingeben:

'vergleiche mit "G" ' testet also genau ein Zeichen.

Dabei werden in $-Listen folgende Einträge gefunden:

"Gerda ","Gustav " und auch "G ".

Suchen Sie in einer #-Liste nur nach dem Eintrag "G", müssen Sie mindestens "G " oder "G " eingeben.

In den %-Listen sind normalerweise keine abschließende Leerzeichen vorhanden: "Gerda","Gustav","G".

Deshalb müssen Sie angeben, ob tatsächlich nach "G" oder nach Einträgen, die mit "G" beginnen, gesucht werden soll!

Nun zu den weiteren Befehlen:

Mit 'druck ein' und 'druck aus' können Sie die Ausgabe auf dem (angeschlossenen) Drucker ein- und ausschalten.

Der Befehl 'manipuliere' bricht das BASIC-Programm ab, gibt aber zuvor noch ein Verzeichnis der vorhandenen Listen und ihrer internen Namen aus. Wenn Sie sich mit dem dk-Betriebssystem vertraut gemacht haben, können Sie diese Listen direkt manipulieren.

Verändern Sie nicht das DEF FN a Statement in den 0-Zeilen!
Es wird vom Programm benötigt.

Auch mit 'b(asic)' können Sie das Programm abbrechen.

Starten Sie nie mit 'Run', denn alle Daten sind in Variablen gespeichert, die mit 'RUN' oder 'CLEAR' gelöscht würden!

Zum 'Gib Befehl:' kommen Sie mit "GOTO 2000"!

Mit 'sa(ve Datei)' können Sie alle Daten auf Kassette speichern. Auf die Angabe eines Namens können Sie verzichten. Geben Sie aber einen Namen ein, werden nur die ersten 9 Zeichen berücksichtigt.

Nach dem Speichern wird automatisch 'VERIFY' durchgeführt. Spulen Sie hierzu das Band zurück und starten wieder.

Beachten Sie unbedingt die Hinweise für den Fall eines Ladefehlers! Möchten Sie auf 'VERIFY' verzichten, drücken Sie 'BREAK' und starten entsprechend der Anweisung.

Der Befehl 'l(ade Datei)' dient dem Wiedereinladen einer (beliebigen) dk-Datei. Beachten Sie auch hier die angezeigten Hinweise.

Sie können nach 'b' (für BASIC) auch das gesamte Programm mit der Datei speichern. Als automatische Startzeile geben Sie 'LINE 9000' ein. Die Kopie funktioniert natürlich nur dann, wenn zuvor das Betriebssystem ab Adresse 62200 geladen wird.

Ändern der Bildschirmausgabe und Einrichten zusätzlicher Listen in bereits bestehenden Dateien

Sie können mit dem Programm "dk - NEW" (auf der Kassette vor dem hier besprochenen Programm) nicht nur neue Dateien einrichten, sondern auch die Struktur vorhandener Dateien ändern und ergänzen.

Dazu müssen Sie zunächst die Datei (mit 'sa(ve Datei)') speichern und dann 'dk - NEW' laden.

In diesem Programm wählen Sie dann die Option 'eigene Datei laden'.

Nun können Sie die Ausgabepositionen für Label und Texte verändern oder auch neue Listen einrichten.

In den neuen Listen bekommt die erste Karteikarte das von Ihnen gewählte Beispiel; alle anderen Fälle der Datei werden mit Leerzeichen, einem "_" oder Null ergänzt.

Die geänderte Datei wird wieder gesavet, um danach mit dem Datenpflegeprogramm "dk - DATA" weiterverwendet zu werden.

Wenn Sie grundsätzlichere Änderungen durchführen wollen, beachten Sie die am Ende der Anleitung gegebenen Hinweise über den internen Aufbau des Systems.

Das Programm dk - NEW

Spulen Sie die Programmkassette an den Anfang und laden Sie mit LOAD "". Stoppen Sie das Band nach Aufforderung und wählen Sie die Option 'a' für 'dk - NEW'.

Das Programm 'dk - NEW' verwendet das geladene dk-Betriebssystem zur Einrichtung neuer Dateien, die mit 'dk - DATA' verwaltet werden können.

Übersicht

- Aufbau einer Datei: Listen, Label, Printpositionen
- Unterscheidung von vier Listenarten
- das Einrichten einer neuen Datei:
  - neue Listen einrichten
  - Printpositionen/Label ändern
  - Datei laden/speichern

Im Programm selber werden viele der folgenden Hinweise ebenfalls an den entsprechenden Stellen gegeben.

a) Aufbau einer Datei

In einer Datei werden für eine Anzahl von K a r t e i k a r t e n oder von F ä l l e n jeweils gleichartige M e r km a l e gespeichert:

'Fälle' sind dabei die Mitglieder eines Vereins oder alle Schallplatten einer Sammlung.
'Merkmale' sind der Familienname, das Alter, das Eintrittsdatum oder
der Interpret, die Musikart, der Name des Lieblingstitels.
'Anzahl' der Fälle gibt an, wieviele Mitglieder oder Schallplatten zur Zeit in der Datei eingetragen sind.

Im dk-Dateisystem werden die Merkmale in Listen gespeichert. So bilden etwa

alle Familiennamen eine Liste: Meier
Schmidt
Müller
Berger
(... usw.)

Daneben gibt es je nach Wahl noch andere Listen, etwa für Geburtstage, Straßenname, Geschlecht usw.

Das Merkmal selber bildet den Namen der Liste: Familiennamen werden also in der Liste mit dem Namen "Familienname" gespeichert.

Listennamen werden üblicherweise L a b e l genannt.

Alle Fälle einer Datei sind in allen Listen mit einem Eintrag vertreten: dadurch wird eine Zuordnung möglich:

Label:		Familienname	Geburtstag	Vorname
Fall-Nr.	1	Meier	121061	Hans
	2	Schmidt	020359	Ursula
	3	Schulz	230964	Peter
	4	Berger	170160	Cäsar
	.	.	.	.
	.	.	.	.
	312	Letztmann	311270	Ultimo

Die Listen (3) sind hier senkrecht angeordnet.

Die einzelnen Einträge in den Listen werden E l e m e n t e genannt.

Die zehnten (oder 127. etc.) Elemente aller Listen bilden zusammengenommen die gespeicherte Information des zehnten Falls.

Diese gesamten Informationen eines einzelnen Falles werden bei dk gemeinsam auf dem Bildschirm angezeigt. Neben den Listenelementen werden auch jeweils die Label angezeigt.

Bsp:

```
++++++++++ Bildschirm ++++++++++
+                              +
+Name: Schmidt                 +
+Vorn: Anton    Geb.Tag: 131051+
+                              +
+Str.: Hinterer Waldweg 17     +
+Tel.: 433445                  +
+                              +
+Text: Dies ist ein kleines Bei-+
+     spiel für Label- und Text-+
+     anordnung.                +
+                              +
++++++++++++++++++++++++++++++++
```

Das System speichert neben den oben genannten Listen auch, wo Sie die Label und die zugehörigen Einträge auf dem Bildschirm angezeigt haben möchten. Die notwendigen Angaben werden im Teil 'Printpositionen ändern' eingegeben.

Die vier Arten von Listen

Das dk-Betriebssystem kann vier Arten von Listen verwalten:

$-Listen
%-Listen
&-Listen
#-Listen

Diese Bezeichnungen wurden gewählt, weil sie neben dem auch in BASIC üblichen $-Zeichen auf der Tastatur des Spectrum liegen.

Es werden vier Arten von Listen angeboten, um je nach Art des zu speichernden Merkmals eine optimale Speichernutzung zu gewährleisten. Dies setzt aber voraus, daß Sie sich mit den jeweiligen Besonderheiten der Listen vertraut machen:

$-Listen sind normale BASIC-Zeichenfelder, wie Sie sie wahrscheinlich selber schon verwenden.
dk verwaltet nur zweidimensionale $-Listen, die mit dem Befehl DIM a$(Anzahl, Länge) eingerichtet werden können.
Diese $-Listen haben für jedes Element eine feste Anzahl von Zeichen, sollten also nur dann verwendet werden, wenn alle Elemente die gleiche Länge haben; dies bietet sich bei vielen Zahlenwerten in Dateien an, etwa bei Postleitzahlen, Kennwerten, Körpergröße und ähnlichem.

%-Listen sind den $-Listen ähnlich, bieten aber für jedes Element eine variable Länge.
%-Listen speicnern deshalb platzsparender alle Eintr(ge mit wechselnden Längen. Dies trifft auf die meisten Merkmale zu: Namen, Straßen, Anmerkungen usw.. Dabei wird jeweils nur soviel Speicherplatz belegt, wie im konkreten Fall notwendig.
In %-Listen können keine Zeichen gespeichert werden, deren CODE größer als 127 ist. Dies stellt aber praktisch keine Einschränkung dar, denn es betrifft Graphikzeichen und die BASIC-Kommandos wie STOP, POKE etc. Sollen auch solche Zeichen gespeichert werden, müssen für diese Merkmale $-Listen verwendet werden.

&-Listen sind normale eindimensionale Zahlenfelder der Art DIM a(Anzahl). Diese Listen können von dk zwar eingerichtet und verwaltet werden, allerdings stehen hierfür keine Such- und Sortierroutinen zur Verfügung. Derartige Zahlenfelder sollten wegen ihres recht großen Platzbedarfs in Textdateien vermieden werden; sie bieten zwar eine große Genauigkeit (ca. 10 Stellen), die aber meist gar nicht benötigt wird. $- oder %-Listen können oft mehr als nur Ersatz hierfür sein.

#-Listen sind binäre Listen, die sehr platzsparend alle Ja-/Nein-Merkmale speichern können. Jedes Element hat entweder den Wert Eins oder Null. In einer binären Liste kann man z.B. das Merkmal "Mitglied?" speichern: für ja würde '1', für nein '0' eingegeben. Gegenüber einer $-Liste, in der 'j' und 'n' eingetragen werden, wird nur ein Achtel des Platzes benötigt. Ebenso können andere Merkmale wie "Geschlecht", "gelesen" oder "bestellt" in #-Listen gespeichert werden.

Sehr vorteilhaft sind auch Kombinationen von Merkmalen, etwa 'besondere Kenntnis in ...'

BASIC
ASSEMBLER
Spiele
Graphik

Mit diesen vier #-Listen können auch Mehrfacheinträge realisiert werden; pro Fall wird dennoch nur insgesamt ein halbes Byte Speicherplatz benötigt!

Das Einrichten einer neuen Datei mit dk - NEW

Bevor Sie nun eine eigene Datei einrichten, sollten Sie sich schon darüber im Klaren sein,

- was Sie speichern möchten, also welche Listen sie einrichten wollen,
- in welcher Reihenfolge Sie später die einzelnen Merkmale eingeben wollen
- welche Listenart jeweils die geeignetste ist.

Eine sorgfältige Vorbereitung erleichtert später die Arbeit mit der Datei. Auch sollten Sie Ihren Entwurf zunächst mit wenigen Fällen und dem Datenpflegeprogramm dk - DATA testen, um im praktischen Versuch festzustellen, ob die gewählte Anordnung und Auswahl der Merkmale leicht zu handhaben ist.

Die Art der Anzeige läßt sich zwar später jederzeit ändern, die Reihenfolge der Eingabe der Merkmale nicht (außer mit direkten Eingriffen in die Systemlisten, s. Anhang).

Ebenso können jederzeit zwar Listen ergänzt werden, das Entfernen ist aber nur indirekt möglich.

Neue Listen einrichten (b)

Mit dieser Option können Sie neue Listennamen (Label), die gewünschte Listenart ($/%/#/&) und ein Beispiel für einen Eintrag in die neue Liste eingeben.

Das Programm zeigt zunächst alle schon vorhandenen Listen an; für eine neue Datei ist eine Liste bereits eingerichtet: sie trägt den Namen "0-Label" und ist eine %-Liste. Den Namen dieser Liste können Sie im Teil 'Printpositionen/Label ändern' durch einen anderen ersetzen; planen Sie diese Liste als erste in Ihr Konzept ein.

Beantworten Sie die Fragen des Programms sinngemäß, vor der entgültigen Einrichtung der Liste bittet dk mit 'OK?' um Bestätigung. Wenn Sie nicht mit "j" antworten, wird die Liste nicht eingerichtet.

Printpositionen / Label verändern (c)

Bei Aufruf dieser Option erhalten Sie vom Programm alle nötigen Informationen.

Die Label und die Beispiele werden jeweils nacheinander zur Korrektur angeboten: die Fragen beziehen sich immer nur auf das Feld, das gerade blinkend dargestellt wird.

Für die Label können Sie zunächst wählen, ob Sie den Text oder die Position ändern wollen.

Beim Andern der Position wird die alte Zeilen- und Spalten-Nummer immer angegeben; so können Sie mit 'Versuch und Irrtum' die beste Position ausprobieren.

Achten Sie darauf, daß eventuell längere Texte als Ihr Beispiel nicht andere Einträge überschreiben können!

Die übrigen Optionen zeigen ebenfalls die notwendigen Instruktionen selbst an!

Speichern Sie die neue Datei für die Verarbeitung mit dk-DATA!

ANHANG 1

Aufbau des BASIC-Dateisystems dk-DATA

Dieser Angang wendet sich an Benutzer, die sich mit dem Betriebsystem dk vertraut gemacht haben und die mitgelieferten BASIC-Programme ergänzen oder verbessern möchten.

Die Systemlisten und Variablen in dk-DATA:

Zur Steuerung des Programms werden verschiedene Kontrollisten eingesetzt:

Name	Funktion
(%	In dieser Liste sind die 16 Befehle gespeichert.
)%	Hier werden die Startadressen der zugehörigen Unterprogramme gespeichert; dies wäre für das vorliegende Programm entbehrlich, denn die Startadressen haben bis auf den Befehl '?' den Wert (Nr. des Befehls * 100). Allerdings wird durch diese zusätzliche Liste das Ergänzen des Programms sehr erleichtert: Wenn Sie etwa ab Zeile 6000 das neue Unterprogramm 'mittelwert' selber programmieren wollen, ergänzen Sie die Liste (% um "mittelwert" und die Liste)% um "6000" (dk-Befehl +(% und +)%). Nun wird Ihr neues Programm mit 'mi(ttelwert)' auf "Gib Befehl" automatisch aufgerufen.
#0 #1	werden für die Suche nach gültigen Befehlen eingesetzt.
#3 - #5	Benutzer-Suchlisten
x$	speichert die Printpositionen der Elemente; x$ hat die Dimension (Anzahl der Label,3). Jedes Element von x$ besteht aus CHR$22 + CHR$ (x-Position) + CHR$ (y-Position).
y$	enthält die Liste der internen Namen aller Benutzerlisten: 0%,a$,b& usw. y$ hat die Dimension (Anzahl der Label,2). Die Reihenfolge der Listen in y$ bestimmt ihre Ausgabereihenfolge.
/%	speichert die Printpositionen der Label und die Label selber in der Form CHR$ 22 + CHR$ (x) + CHR$ (y) + "Labelname". Liest man ein solches Label mit dem dk-Befehl READ und printet k$, wird es durch den Kontroll-Charakter 22 automatisch an der Stelle x,y ausgegeben.
z&	verwaltet die Namen der verschiedenen Listen in der Reihenfolge # $ % &. In z(1) steht also z.B. der CODE des letzten für #-Listen vergebenen Namen. z& hat die Dimension (5), z(5) wird aber nur im Programm dk-NEW als Flag benutzt.
w&	wird vom Menu benutzt, um mit dem dk-Befehl DATA die Nummer des gefundenen Befehls aus der internen Suchliste #0 zu ermitteln.
u&	wird immer eingesetzt, wenn aus binären Listen die entsprechenden Fallzahlen erzeugt werden müssen, also bei 'zsort' und 'zsuchliste'.

Name	Funktion
v&	wird ebenso verwendet.

Verwendete Variablen

dk	62200 zum Aufruf des dk-Systems
s1 - s3	Inhalt der Suchlisten #3 bis #5
ins	insgesamte Fallzahl
ein	2000; Adresse des Menus
zahl	23728; byte, das angibt, ob READ eine Zahl gelesen hat.
dw	wird gelegentlich zur Speicherung des Werts des dk-Aufrufs verwendet.
v$ w$	dienen der Zwischenspeicherung von Listennamen, INPUTs u.ä.
DEF FN a	wird von dk-DATA benutzt, aber nicht eingerichtet! Bei einer Änderung mit dem dk-Befehl FN a ist dk-DATA nicht mehr lauffähig.

Studieren Sie die Unterprogramme! Eine individuelle Anpassung an Ihre Bedürfnisse wäre einen Versuch wert, nur Mut!

Reizvoll wäre z.B. ein Unterprogramm, daß mit PLOT und DRAW bei jeder Ausgabe graphische Ergänzungen (Einrahmungen) erzeugt oder auch zusätzliche Farben einbringt (Der Autor konnte sich bislang mit Farben in Dateien noch nicht anfreunden). Sie sollten versuchen, diese Optionen so einzusetzen, daß die zugehörigen Daten schon in dk-NEW eingegeben werden können und somit für jede Datei zur Verfügung stehen.

ANHANG 2

Aufbau der %- und #-Listen

Die %- und #-Listen sind prinzipiell genauso gespeichert wie ein normales BASIC-Feld.

Studiert man die Art der Datenspeicherung im Spectrum, kann man feststellen, daß zwei als Namen mögliche Zeichen nicht verwendet werden: CHR$ 64 (40h) und CHR$ 192 (C0h).

Diese Namen werden für %-Listen (40h) und #-Listen (C0h) eingesetzt. Untereinander werden %-und #-Listen durch das vierte Byte unterschieden: in ihm wird ein beliebiger Charakter als Name der Liste geführt.

Byte:	1	2	3	4	5	6	7	8	9	10
#-Listen:	C0h	(Länge)		Name	Anzahl von'1'		Anzahl insg.		Maske (last)	1. Wert
%-Listen:	40h	(Länge)		Name	Anzahl		alter $-Name		alte DIM1	80h

Byte:	11
%-Listen:	1.Element / 2.Element /./ Letztes Element/ 80h
	(das letzte Byte jedes Elements hat das BIT 7 gesetzt zur Trennung)