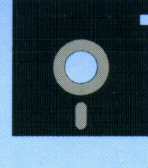
Markt&Technik

Said Baloui

C64/C128 PROFI-TOOLS

Eine vollständige Sammlung von Assemblermodulen für den professionellen Basic-Programmierer:
★ Pull-down-Menüs ★ Windowing ★ Quicksort ★ Tastaturmakros und vieles mehr



Auf Diskette (Format 1541) enthalten:
Alle Assemblermodule inkl. Quellcode und ausführlichen Demoprogrammen.

C64/C128

PROFI-TOOLS

Eine vollständige Sammlung von Assemblermodulen für den professionellen Basic-Programmierer:

Pull-down-Menüs
Windowing
Quicksort
Tastaturmakros
und vieles mehr

Said Baloui

Markt&Technik
Verlag AG

CIP-Titelaufnahme der Deutschen Bibliothek

Baloui, Said:
C64, C128 Profi-Tools . e. vollst. Sammlung von
Assemblermodulen für d. professionellen Basic-Programmierer:
Pull-down-Menüs, Windowing, Quicksort,
Tastaturmakros u. vieles mehr / Said Baloui. –
Haar bei München · Markt-u.-Technik-Verl., 1988.
ISBN 3-89090-617-6

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
91 90 89 88

ISBN 3-89090-617-6

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Ebner Ulm
Printed in Germany

Inhaltsverzeichnis

Vorwort

Sehr geehrter Leser,

verwechseln Sie die Tools auf der beiliegenden Diskette bitte keinesfalls mit einer der unzähligen Basic-Erweiterungen für den C64.

Diese Befehlserweiterungen bieten »Spielereien« wie COLOR-, INSTR- oder LOCATE-Anweisungen, die zwar oft nützlich sind, Ihnen aber kaum dabei helfen, wirklich professionelle Programme in Basic zu erstellen.

Im Gegensatz dazu wenden sich die vorliegenden Tools und Utilities an den fortgeschrittenen (und entsprechend anspruchsvolleren) Programmierer.

»Tool« oder »Utility« heißt soviel wie »Werkzeug« oder »Hilfsmittel«. Und genau das finden Sie auf der Diskette – Hilfsmittel zur Erstellung wirklich professioneller Basic-Programme. Ein Beleg dafür ist die »Vorgeschichte« der Programme.

Alle Tools entstanden bei der Erstellung des Programms MasterBase, einer Dateiverwaltung für den C64 und den Plus4, die von Markt und Technik vertrieben wird. Diese Dateiverwaltung gehört zu den leistungsfähigsten und benutzerfreundlichsten Programmen, die für diese Rechner angeboten werden.

Die Profi-Tools enthalten alle Routinen, die ich zur Erstellung von MasterBase benötigte!

Was Ihnen hier angeboten wird, ist somit nicht weniger als die Möglichkeit, jene Hilfsmittel in Ihren eigenen Programmen einzusetzen, die so oder ähnlich nahezu jedes professionelle Programm enthält.

Assembler-Programmierer können zusätzlich durch das Studium der kommentierten Quelltexte profitieren.

Mit dieser »Schatzkiste« stehen Ihnen nahezu unbegrenzte Möglichkeiten für Ihre eigenen Programme zur Verfügung. Bei geschicktem Einsatz der Utilities können Sie

Programme schreiben, die ebenso schnell und komfortabel sind wie das genannte MasterBase.

Zum Beispiel wird niemand erkennen können, ob Sie nun eigentlich ein Basic- oder ein Maschinensprachcprogramm geschrieben haben. Die Utilities stellen Ihnen für alle in der Praxis benötigten zeitkritischen Programmteile reine Maschinenprogramme zur Verfügung. Sogar die gefürchtete Garbage Collection wird durch die Utilities zur Bedeutungslosigkeit verurteilt.

Diese Tools bieten Ihnen weitaus mächtigere Hilfsmittel als einige zusätzliche Anweisungen wie LOCATE (Cursor auf eine angegebene Position setzen) oder INSTR (prüfen, ob ein String in einem anderen enthalten ist).

Basic-Erweiterungen enthalten relativ einfache Anweisungen, die im Grunde bereits »serienmäßig« eingebaut sein sollten (was bei den meisten Rechnern auch der Fall ist, nur nicht beim C64). Derartige einfache Anweisungen enthalten teilweise auch die vorliegenden Tools – allerdings mehr als »Abfallprodukte«!

Vor allem enthalten die Tools sehr komplexe Routinen, die weit über den Rahmen jeder Basic-Erweiterung hinausgehen. Den Unterschied zwischen einer Befehlserweiterung und den vorliegenden Tools und Utilities zeigen am besten einige Beispiele:

Pull-down-Menüs

Heutzutage sind professionelle Programme ohne Pull-down-Menüs wie auf dem Atari ST oder dem Amiga nicht mehr vorstellbar. Im Gegensatz zu Rechnern wie dem ATARI ST oder dem AMIGA ist der C64 jedoch nicht bereits von Haus aus auf die Verwaltung von Pull-down-Menüs eingerichtet.

Diese komplexe Aufgabe sprengt den Rahmen einer Befehlserweiterung bei weitem. Sie ist nicht vergleichbar mit zusätzlichen Befehl(chen) wie LOCATE oder INSTR, sondern erfordert ein hochkomplexes eigenes Programm, das vollständig in Maschinensprache geschrieben werden muß.

Sortierroutinen

In Basic ist es unmöglich, größere Arrays – die zum Beispiel Adressen oder Vokabeln enthalten – in halbwegs zufriedenstellender Zeit zu sortieren. Auch nicht mit schnellen Sortieralgorithmen wie zum Beispiel QUICKSORT.

Die Tools enthalten unter anderem auch eine in Maschinensprache (wie alle Routinen) geschriebene QUICKSORT-Routine, die selbst 1000 Strings in weniger als einer Sekunde sortiert.

Hohe Sortiergeschwindigkeiten allein sind jedoch in der Praxis bei weitem nicht ausreichend. Sortierroutinen müssen flexibel sein. Nehmen Sie einen »Vokabeltrainer«. In einem Stringarray befinden sich die englischen Vokabeln, in einem zweiten die dazugehörigen deutschen Ausdrücke:

```
E$(1) = "LIBRARY"          D$(1) = "BIBLIOTHEK"
E$(2) = "SOURCE"           D$(2) = "QUELLE"
E$(3) = "DESTINATION"      D$(3) = "BESTIMMUNG"
E$(4) = "FAME"             D$(4) = "SCHICKSAL"
.....
.....
.....
```

Angenommen, Sie wollen die englischen Vokabeln alphabetisch sortieren. Dann darf keinesfalls die Beziehung zwischen den englischen und den deutschen Vokabeln verlorengehen, wie es beim einfachen Sortieren des Arrays *E$(..)* der Fall wäre. Jede Änderung im Array *E$(..)* muß im Array *D$(..)* nachvollzogen werden. Man sagt, *D$(..)* wird »mitsortiert«.

```
E$(1) = "DESTINATION"      D$(1) = "BESTIMMUNG"
E$(2) = "FAME"             D$(2) = "SCHICKSAL"
E$(3) = "LIBRARY"          D$(3) = "BIBLIOTHEK"
E$(4) = "SOURCE"           D$(4) = "QUELLE"
.....
.....
.....
```

Die QUICKSORT-Routine ist unter anderem in der Lage, ein Array beliebigen Typs zu sortieren (String, Integer, Real) und ein zweites Array mitzusortieren, dessen Typ ebenfalls beliebig ist.

Eingaberoutinen

Ein weiteres Beispiel für den Einsatz der Tools ist eine Eingaberoutine. Jedes professionelle Programm benötigt eine eigene Eingaberoutine. Denken Sie an eine Dateiverwaltung, die mit Masken zur Eingabe von Datensätzen arbeitet.

```
Name: <.........>      Vorname: <............>
Strasse: <..............>
Plz: <....>    Ort: <................>
Telefon: <..............>
```

Jedes Feld besitzt eine bestimmte Länge. Dem Benutzer darf es nicht möglich sein, während der Eingabe den Cursor aus dem Feld herauszubewegen oder gar (siehe INPUT) den Bildschirm zu löschen.

Zusätzlich muß der Programmierer vorgeben können, welche Zeichen in einem bestimmten Feld eingegeben werden dürfen. Zum Beispiel dürfen im Feld »PLZ« nur Zahlen eingegeben werden. Die Eingabe ist somit auf die Zeichen »1234567890« beschränkt.

Vor allem in Masken genügt es keineswegs, daß eine Eingabe bei INPUT nur mit der Taste ↵ zu beenden ist. In einer Eingabemaske soll zum Beispiel auch die Taste ↓ die aktuelle Eingabe beenden (und den Cursor zum nächsten Feld der Maske bewegen).

Eine Eingaberoutine zum Einsatz in professionellen Programmen muß somit mindestens die folgenden Forderungen erfüllen:

- Eingabe nur in einer definierten Eingabezone
- Definition der zulässigen Zeichen
- Definition der Zeichen, die die Eingabe abschließen

Kein Basic-Interpreter und auch keine Befehlserweiterung enthält Eingaberoutinen, die diesen Anforderungen voll genügen – im Gegensatz zu den hier vorgestellten Tools!

Sie haben nun eine Vorstellung darüber, was die beiliegende Diskette enthält und kennen den Unterschied zwischen einer Befehlserweiterung und komplexen Werkzeugen.

Ich wünsche Ihnen nun viel Erfolg beim Einsatz Ihrer neuen Hilfsmittel. Alle dazu nötigen Hinweise finden Sie auf den folgenden Seiten.

Said Baloui

C64/C128

PROFI-TOOLS

Teil 1

Benutzung der Utilities

Inhalt der Diskette
Das Initialisierungsprogramm TOOLS.INT
Aufruf der Utilities
Ständig aktive Utilities

1

Inhalt der Diskette

Die beiliegende Diskette enthält drei Dateigruppen: die »Kernprogramme« MC1.OBJ und MC2.OBJ enthalten die eigentlichen Tools und Utilities. TOOLS.INIT ist ein Basic-Programm, das Ihnen die Anwendung der Utilities erleichtert. COMMODORE.COD und ASCII.COD sind zwei Dateien, die dafür sorgen, daß Ihr Drucker auch die auf dem Bildschirm angezeigten Umlaute korrekt wiedergibt. Die Funktion von VIZA.COD wird im Abschnitt »Sonstiges« besprochen.

Diese Kernprogramme sollten Sie (bis auf VIZA.COD, das normalerweise nicht benötigt wird) auf alle Disketten kopieren, die Sie zur Speicherung Ihrer Basic-Programme verwenden. Ihr Vorhandensein ist eine grundlegende Voraussetzung zum Einsatz der Utilities!

Die zweite Gruppe (Sourcetexte) ist nur für Assembler-Programmierer interessant. Die einzelnen Dateien dieser Gruppe enthalten den Quellcode der Routinen. Der Quellcode wurde mit dem Hypra-Ass und – im Falle von QUICK.SRC und CHAR.SRC – dem MAE erstellt.

Die dritte Gruppe (Demoprogramme) enthält Demoprogramme, die die Anwendung der einzelnen Utilities in einem Basic-Programm demonstrieren.

Kernprogramme

TOOLS.INIT	Initialisierungsprogramm
MC1.OBJ	Utilities Teil 1
MC2.OBJ	Utilities Teil 2
COMMODORE.COD	Codetabelle für Commodore-Drucker
ASCII.COD	Codetabelle für ASCII-Drucker
VIZA.COD	VIZAWRITE-Codetabelle

Sourcetexte

DISKIO.SRC	Disk-Utilities etc. (Hypra-Ass)
WINDOW.SRC	Windowing-Routinen (Hypra-Ass)
INPUT.SRC	Eingaberoutinc (Hypra-Ass)
QUICK.SRC	QUICKSORT (MAE)
CHAR.SRC	Deutscher Zeichensatz (MAE)
TREIBERETC.SRC	Drucker-Utilities ctc.(Hypra-Ass)
GARBAGE.SRC	Garbage Collection (Hypra-Ass)

Demoprogramme

INPUT1	Eingaberoutine Demo 1
INPUT2	Eingaberoutine Demo 2
INPUT3	Eingaberoutine Demo 3
SATZINFO	Demo von SATZINFO
BLAETTERN	Demo von BLÄTTERN
STROUT	Demo der Stringausgabe
SEARCH	Demo der Suchroutine
QUICK1	QUICKSORT Demo 1
QUICK2	QUICKSORT Demo 2
QUICK3	QUICKSORT Demo 3
MEMORYSAVE/MEMORYLOAD	Bereiche speichern/laden
FASTSAVE/FASTLOAD	Arrays speichern/laden
DIRECTORY	Inhaltsverzeichnis lesen
PAINTWINDOW	Window + Inhalt ausgeben
WINDOWING 1	Window retten/holen
WINDOWING 2	Überlappende Windows
INVERT	Invertier-Routine
PULL-DOWN-MENUE	Pull-down-Menüs
FASTGARBAGE	Schnelle Garbage Collection
ZAHL	Anwendung von TOOLS.INIT

2

Das Initialisierungsprogramm TOOLS.INIT

Das für Sie wichtigste Programm ist TOOLS.INIT, das Initialisierungsprogramm. Initialisierung bedeutet Vorbereitung, und genau das ist die Aufgabe dieses Basic-Programms. Dieses Programm trifft nach dem Start einige Vorbereitungen zum Einsatz der Utilities. Vor allem lädt es die restlichen Kernprogramme nach.

Das heißt, um die Utilities einzusetzen, müssen sich alle Kernprogramme gemeinsam auf einer Diskette befinden (TOOLS.INIT, MC1.OBJ, MC2.OBJ, COMMODORE.OBJ und ASCII.OBJ).

Sie werden TOOLS.INIT von nun an ständig verwenden. Mit diesem kleinen Basic-Programm müssen Ihre eigenen Programme beginnen, wenn Sie darin die Utilities einsetzen wollen.

```
10 REM ** MC-ROUTINEN NACHLADEN **
12 IF PEEK(56)=149 THEN 26
14 F=PEEK(187)+256*PEEK(188):REM AKTUELLER FILENAME
.....
.....
.....
200 REM ****************************
210 REM *       HAUPTPROGRAMM        *
220 REM ****************************
```

Inhalt von TOOLS.INIT

Das heißt natürlich nicht, daß Sie alle Zeilen aus TOOLS.INIT abtippen müssen. Wenn Sie ein Programm schreiben, in dem Sie die Utilities einsetzen wollen, gehen Sie so vor:

Laden Sie TOOLS.INIT. Die letzte Anweisung dieses Basic-Programms besitzt die Zeilennummer 220. Ihr eigenes Programm schreiben Sie einfach ab den folgenden Zeilennummern; das heißt, Sie beginnen Ihr Programm mit der Zeilennummer 230 und kümmern sich überhaupt nicht um die vorhergehenden Zeilen des Initialisierungsprogramms.

Das folgende einfache Beispiel gibt die Zahlen 1 bis 100 aus. Zuerst wurde TOOLS.INIT geladen und anschließend ab 230 das eigentliche Hauptprogramm eingegeben.

```
10 REM ** MC-ROUTINEN NACHLADEN **
12 IF PEEK(56)=149 THEN 26
14 F=PEEK(187)+256*PEEK(188):REM AKTUELLER FILENAME
.....
.....
.....
200 REM ****************************
210 REM *      HAUPTPROGRAMM       *
220 REM ****************************
```

Inhalt von TOOLS.INIT

```
230 :
240 REM ZAHLEN 1-100 AUSGEBEN
250 FOR I=1 TO 100
260 : PRINT I
270 NEXT I
```

Hauptprogramm

Dieses kleine Demoprogramm befindet sich unter dem Namen ZAHL auf der Diskette. Bitte laden und starten Sie das Programm. Sie werden feststellen, daß nach dem Start mit RUN die Floppy anläuft und es einen kleinen Moment dauert, bis wie gewünscht die Zahlen 1 bis 100 ausgegeben werden.

Viel interessanter ist jedoch, wie sich die Tastatur nach dem Start dieses Programms verhält. Zuerst einmal stellen Sie fest, daß automatisch der KLEIN-/GROSSCHRIFT-Modus eingeschaltet wird. Dieser Modus ist nicht nur sinnvoll (was soll eine Textverarbeitung, bei der keine Eingabe von Kleinbuchstaben möglich ist). Er ist für die Benutzung der Utilities zwingend notwendig!

Aber vor allem verfügt Ihr C64 auf einmal über einen deutschen Zeichensatz, das heißt über die Umlaute »öäüßÖÄÜ«.

Die Utilities enthalten unter anderem eine Routine, die die Tastenbelegung des C64 ändert. Genau diese Routine wird bei der Initialisierung aufgerufen. Die Tastatur besitzt nun folgende Belegung:

Alte Belegung	Neue Belegung	Alte Belegung	Neue Belegung
:	ö	SHIFT + :	Ö
;	ä	SHIFT + ;	Ä
@	ü	SHIFT + @	Ü
£	ß	SHIFT + .	:
		SHIFT + ,	;

Übrigens: wer VIZAWRITE kennt, hat keinerlei Umstellungsprobleme. Die Tastaturbelegung ist die gleiche wie in diesem weitverbreiteten Textverarbeitungs-Programm.

Sie haben soeben einen ersten Eindruck von den Möglichkeiten der TOOLS erhalten. Bei der Initialisierung wird automatisch ein deutscher Zeichensatz eingeschaltet. Diese Maßnahme ist sinnvoll, da Sie die TOOLS erwarben, um professionelle Programme zu erstellen. Professionelle Programme sind nun einmal ohne Umlaute nicht denkbar – stellen Sie sich eine Textverarbeitung ohne Umlaute vor!

Utilities, die Umlaute auf den Bildschirm zaubern, finden Sie in Massen in entsprechenden Fachzeitschriften. In eigenen Programmen können Sie diese Utilities meist nicht verwenden, da es oft unmöglich ist, diese Umlaute auch korrekt auszudrucken. Was haben Sie von Umlauten, die beim Ausdruck als » . « , » (« oder gar als Grafikzeichen ausgegeben werden?

Bei den TOOLS sind entsprechende Befürchtungen jedoch völlig unbegründet. Ihnen stehen unter anderem auch Utilities zur Druckeranpassung zur Verfügung. Mit diesen Utilities ist es kein Problem, praktisch jeden Drucker zur einwandfreien Wiedergabe der Umlaute zu bringen. Und zwar unabhängig davon, ob Ihr Drucker (vielleicht über ein Interface) an die normale serielle Schnittstelle des C64 oder mit einem entsprechenden Kabel direkt an den Userport angeschlossen ist!

Die einzige Voraussetzung ist, daß Ihr Drucker Umlaute kennt, was bei fast allen Druckern der Fall ist (außer bei den Commodore-Druckern MPS 801 und MPS 803). Wie Sie vorgehen müssen, um mit Ihrem Drucker die auf dem Bildschirm sichtbaren Umlaute korrekt zu drucken, wird im Abschnitt »Drucker« detailliert beschrieben.

Bitte blättern Sie nun nicht einfach zu diesem Abschnitt vor. Warten Sie noch ein wenig mit dem Drucken. Sie sollten sich die Utilities in der Reihenfolge »zu Gemüte führen«, die in diesem Handbuch benutzt wird. Diese Reihenfolge hat ihren Sinn. Die Utilities sind in Gruppen unterteilt. Zuerst werden jene Gruppen beschrieben, die die am einfachsten zu benutzenden Utilities enthalten, dann die komplexeren Tools. Wenn Sie sich an die vorgegebene Reihenfolge halten, werden Sie daher schrittweise in die Benutzung Ihrer neuen Werkzeuge eingeführt.

3

Aufruf der Utilities

Der Aufruf der verschiedenen Utilities ist standardisiert. Alle Routinen werden mit dem SYS-Befehl unter Angabe der Startadresse der betreffenden Routine aufgerufen. Eventuell nötige Zusatzinformationen (Parameter) werden durch Kommata getrennt hinter der Startadresse angegeben. Das allgemeine Format einer Routine lautet:

```
SYS ADRESSE, PARAMETER 1, PARAMETER 2, PARAMETER 3,...
```

Welche Parameter anzugeben sind, ist sehr unterschiedlich. Es können Zahlen, Zeichenketten oder Variablen sein. Der jeweilige Parametertyp wird auf folgende Weise gekennzeichnet:

1. PARAMETER$: Stringvariable (*A$*)
2. PARAMETER%: Integervariable (*X%*)
3. PARAMETER: Beliebige Zahlenkonstante (3), Zahlenvariable (*X*) oder beliebig komplexer Ausdruck (3*X+5)

Achtung: Immer dann, wenn ein Parameter mit dem Zeichen »*$*« oder »*%*« gekennzeichnet ist, muß eine Variable angegeben werden!

Da es sehr fehlerträchtig ist, Maschinenroutinen durch Angabe Ihrer Startadresse aufzurufen, weist das Initialisierungsprogramm TOOLS.INIT jede Startadresse einer eigenen Variablen zu.

```
24 REM ** STARTADRESSEN **
26 IO=38912:        REM MODUL 1 (DISKIO)
28 BLAETTERN=IO:    REM BLAETTERN
30 COVERT=IO+3:     REM CONVERT
32 CRS=IO+6:        REM CURSOR SETZEN
34 SI=IO+9:         REM SATZINFO
36 DAUSGABE=IO+12: REM DISK-AUSGABE
38 DEINGABE=IO+15: REM DISK-EINGABE
40 DIR=IO+18:       REM DIRECTORY
42 SPEICHERN=IO+21:REM BLOCK SPEICHERN
```

```
44 VLADEN=IO+24:    REM BLOCK LADEN
46 :
48 TR=39888:        REM MODUL 2(TREIBER)
50 MAKRO=TR:        REM MAKROS INIT.
52 FSCREEN=TR+3:    REM FASTSCREEN INIT.
54 WAHL=TR+6:       REM KOMMANDO-AUSWAHL
56 SEARCH=TR+9:     REM STRING SUCHEN
58 AUSGABE=TR+12:   REM STRING-AUSGABE
60 SRIELL=TR+15:    REM SERIELL INIT.
62 PARALLEL=TR+18:  REM PARALLEL INIT.
64 PRESENT=TR+21:   REM DEVICE PRESENT
66 :
68 PINIT=50960:     REM PUFFER INIT.
70 WINDOW=PINIT+3:  REM WINDOWS MALEN
72 PUFFER=PINIT+6:  REM WINDOWS PUFFERN
74 INVERT=PINIT+9:  REM INVERTIEREN
76 CNTROL=PINIT+12:REM MENUE-KONTROLLE
78 :
80 EINGABE=50176:   REM EINGABE-ROUTINE
82 QUICK=52240:     REM QUICKSORT
84 CHAR=53039:      REM ZEICHENSATZ INIT.
86 GC=38144:        REM GARBAGE COLLECT.
88 :
90 :
92 REM ** STARTADRESSEN D.TABELLEN **
94 MT=1536:         REM MAKRO-TABELLE
96 CT=1792:         REM CODE-TABELLE
98 :
100 :
```

Die Variablennamen wurden sehr sorfältig ausgesucht, um Überschneidungen zu vermeiden. Änderungen sollten Sie unbedingt vermeiden! Ein Beispiel: Angenommen, Ihnen paßt der Name *DEINGABE* für »Disk-Eingabe« nicht (Zeile 38) und Sie ändern den Namen in *EINGABE*. Dann sind Schwierigkeiten unvermeidlich, da in Zeile 80 die Startadresse einer weiteren Routine ebenfalls der Variablen *EINGABE* zugewiesen wird.

Denken Sie daran: Für den Basic-Interpreter sind nur die beiden ersten Zeichen eines Variablennamens signifikant. Wenn Sie in Ihrem Hauptprogramm eine Variable namens *EIN* verwenden, ändern Sie dadurch die Startadresse der Routine *EINGABE*!

Wie die Utilities zu benutzen sind, lernen Sie am besten durch das Studium der folgenden Beispiele.

Deutschen Zeichensatz einschalten

Die Routine zum Einschalten des deutschen Zeichensatzes beginnt ab Adresse 53039. 53039 ist die Startadresse der Routine. Außer dem Aufruf selbst sind keine weiteren Informationen notwendig. Daher genügt zum Einschalten die Anweisung:

```
SYS 53039
```

Da TOOLS.INIT diese Startadresse in der Variablen *CHAR* speichert (siehe Zeile 84 von TOOLS.INIT), können Sie ebenso folgenden Aufruf verwenden:

```
SYS CHAR
```

Im folgenden werde ich bei allen Beispielen für Startadressen die Variablen verwenden, in der TOOLS.INIT diese Adressen speichert. Denn der Aufruf `SYS CHAR` ist doch erheblich freundlicher als das unverständliche `SYS 53039`.

Cursor setzen

Eines der erwähnten »Abfallprodukte« ist die Routine SETCURSOR zum Setzen des Cursors auf eine bestimmte Position. Diese Routine beginnt ab der Adresse 38918, die TOOLS.INIT in der Variablen *CRS* speichert (Zeile 32 von TOOLS.INIT). An Parametern benötigt sie die gewünschte Position, also Spalte und Zeile. Das Format des Aufrufs:

```
SYS CRS, SPALTE, ZEILE
```

Der Parameter *SPALTE* ist ein beliebiger ganzzahliger Wert zwischen 0 und 39. *ZEILE* kann im Bereich zwischen 0 und 24 frei gewählt werden. Zum Beispiel setzt der Aufruf

```
SYS CRS,0,0
```

den Cursor auf die linke obere Ecke des Bildschirms. Mit folgender Befehlsfolge könnten Sie die Zeichenfolge »HALLO« ab dieser Position ausgeben:

```
SYS CRS,0,0: PRINT "HALLO"
```

Oder mit

```
SYS CRS,0,24: PRINT "HALLO"
```

die gleiche Zeichenfolge ab dem Beginn der untersten Bildschirmzeile ausgeben. Äquivalente Aufrufe sind:

```
SYS CRS,3+5,10       => Cursor auf Spalte 8 von Zeile 10
X=3:Y=7:SYS CRS,X,Y  => Cursor auf Spalte 3 von Zeile 7
A%=5:SYS CRS,A%,A%+5 => Cursor auf Spalte 5 von Zeile 10
```

Stringarray durchsuchen

SEARCHSTRING durchsucht ein Stringarray. Beim Aufruf geben Sie an, wonach gesucht wird, welcher Bereich des Arrays zu durchsuchen ist, und in welcher Variablen SEARCHSTRING Ihnen das Resultat der Suche übergibt. Die Adresse von SEARCHSTRING (39897) speichert das Initialisierungsprogramm in der Variablen *SEARCH*.

Der Aufruf lautet (beachten Sie den Zusatz »V/« vor jedem Parameter!):

```
SYS SEARCH, SUCH$, ARRAY$(START), ARRAY$(ENDE), INDEX%
```

SEARCHSTRING durchsucht ein Stringarray ab *ARRAY$(START)* bis *ARRAY$(ENDE)* nach *SUCH$*. *SUCH$* ist das »Suchkriterium«, zum Beispiel ein gesuchter Name. *ARRAY$(START)* und *ARRAY$(ENDE)* geben das zu durchsuchende Stringarray an. *INDEX%* ist eine Integervariable, in der STRINGSEARCH den Index des gefundenen Arraystrings übergibt. STRINGSEARCH besitzt die Adresse 39897. Ein Beispiel für einen korrekten Aufruf:

```
S$="MAIER"
SYS SEARCH, S$, A$(1), A$(100), I%
```

Nach diesem Aufruf wird STRINGSEARCH der Reihe nach die Variablen *A$(1)*, *A$(2)*, *A$(3)*, ... , *A$(100)* mit dem Inhalt von *S$*, also »MAIER«, vergleichen. Angenommen, *A$(20)* enthält den gesuchten Namen. Dann ist die Suchroutine beendet und in *P%* wird der gefundene Index 20 gespeichert.

Alle Parameter müssen in Form von Variablen angegeben werden! Folgende Beispiele sind falsch:

```
SYS SEARCH, "MAIER", A$(1), A$(100), I%
S$="MAIER": SYS SEARCH, S$, A$(1), A$(100), 5
```

Die Unzulässigkeit im zweiten Beispiel ist offensichtlich: Wie sollte die Suchroutine in der Konstanten 5 eine Zahl speichern können?

Merken Sie sich: Endet ein Parameter mit »$« oder »%«, muß für diesen Parameter eine Variable verwendet werden!

Directory einlesen

Ein weiteres Beispiel: Die Routine DIRECTORY (die Adresse 38930 speichert TOOLS.INIT in der Variablen *DIR*) liest das Inhaltsverzeichnis einer Diskette ein. Die Syntax des Aufrufs:

```
SYS DIR, LFN, ARRAY$(START), ANZAHL%
```

Das heißt, für LFN können Sie wahlweise eine Konstante (2), Variable (A) oder einen Ausdruck (3 + A − 1) einsetzen. Für *ARRAY$(START)* und *ANZAHL%* müssen dagegen Variablen (Stringarrayvariable beziehungsweise Integervariable) verwendet werden!

4

Ständig aktive Utilities

Die Utilities können grob in zwei Gruppen unterteilt werden:

1. Utilities, die nach einem Aufruf nur kurzzeitig aktiv sind.
2. Utilities, die nach einer einmaligen Initialisierung bis zu einem RESET oder Ausschalten des Rechners aktiv sind.

Zur ersten Gruppe gehören nahezu alle Utilities. Zum Beispiel ist SETCURSOR nur einen kurzen Moment aktiv. Nach dem Aufruf wird der Cursor auf die angegebene Position gesetzt und SETCURSOR ist bis zum nächsten Aufruf inaktiv.

Bei der zweiten Gruppe wird das betreffende Utility nur einmal aufgerufen (initialisiert) und ist von diesem Zeitpunkt an ständig aktiv. Zu dieser Gruppe gehören die Utilities CHARACTERSINIT, MAKROSINIT, FASTSCREENINIT, PARALLELINIT, und SERIELLINIT.

Einige Beispiele hierzu:

1. CHARACTERSINIT schaltet den deutschen Zeichensatz ein. Dieser Zeichensatz bleibt bis zum Ausschalten erhalten.
2. MAKROSINIT ermöglicht »Tastatur-Makros«, das heißt die Belegung einer Taste mit einer kompletten Zeichenfolge. Nach der einmaligen Initialisierung kann das Utility jederzeit verwendet werden, um neue Tastatur-Makros zu definieren oder abzurufen. Und zwar sowohl im Direktmodus als auch während eines Programmlaufs.
3. FASTSCREENINIT beschleunigt alle Bildschirmausgaben Ihres Programms, die die PRINT-Anweisung benutzen. Die Geschwindigkeitssteigerung bleibt nach der Initialisierung erhalten. Das heißt, Sie können anschließend ein anderes Ihrer

Programme laden und starten, und auch dessen Bildschirmausgaben werden weiterhin beschleunigt.

4. PARALLELINIT/SERIELLINIT: die Utilities enthalten eine Centronics-Schnittstelle, mit der ein Drucker – mit Hilfe eines geeigneten Kabels – direkt an den Userport angeschlossen werden kann. Nach der Initialisierung dieser Schnittstelle mit PARALLELINIT werden alle Druckausgaben (egal, ob im Direkt- oder im Programm-Modus) über den Userport geleitet, bis der Rechner ausgeschaltet oder mit SERIELLINIT wieder die normale serielle Ausgabe initialisiert wird.

Man könnte sagen, die ständig aktiven Utilities »lauern unsichtbar im Hintergrund« und erfüllen zuverlässig Ihre Aufgaben, ohne daß ein wiederholter Aufruf notwendig wäre. Dies gilt, wie gesagt, bis zu einem RESET oder Ausschalten des Rechners (oder bis zum Laden eines Maschinenprogramms, das den gleichen Speicherbereich wie die Utilities verwendet und diese somit beim Laden überschreibt).

C64/C128

PROFI-TOOLS

Teil 2 Referenzteil

Pull-down-Menüs
Floppy
Variablen
Bildschirm/Tastatur
Drucker
Sonstiges

Die Beschreibung der Utilities erfolgt nach Gruppen geordnet. Innerhalb jeder Gruppe werden die einzelnen Utilities in alphabetischer Ordnung vorgestellt. Der Aufbau der Beschreibungen ist standardisiert:

1. Name und Funktionskurzbeschreibung
2. Adresse und zugehöriger Variablenname
3. Syntax (Format des Aufrufs)
4. Parameter (Beschreibung der anzugebenden Informationen)
5. Funktion (Beschreibung der genauen Wirkungsweise)
6. Beispiel(e)
7. Besondere Hinweise (nur, wo angebracht)

Beispiel: SetCursor

Cursor positionieren

Adresse `38918`

Variablenname `CRS`

Syntax `SYS CRS, SPALTE, ZEILE`

Parameter SPALTE: Spaltenposition (0–39)

ZEILE: Zeilenposition (0–24)

Funktion SETCURSOR setzt den Cursor auf eine in *SPALTE* und *ZEILE* angegebene Position. *SPALTE* und *ZEILE* sind ganzzahlige Werte zwischen 0 und 39 (Cursorspalte) beziehungsweise 0 und 24 (Cursorzeile).

Beispiel

```
SYS CRS,0,0: PRINT "HALLO"
```

setzt den Cursor auf Spalte 0 von Zeile 0 (linke obere Bildschirmecke) und gibt ab dieser Position die Zeichenkette »HALLO« aus.

Einige Tips aus der Praxis

Im letzten Beispiel ist 38918 die absolute Adresse der Routine SETCURSOR. Statt die Routine mit dieser absoluten Adresse aufzurufen (SYS 38918,0,0) ist es vorzuziehen, die Variable *CRS* zu verwenden, in der die absolute Adresse bei der Initialisierung gespeichert wurde (SYS CRS,0,0).

Bei der Angabe falscher Datentypen verhalten sich die Routinen recht »tolerant«. Eine falsche Syntax wird wie vom Basic-Interpreter mit der Meldung »SYNTAX ERROR IN ...« quittiert (die Zeile »500 SYS CRS, A$, 5« führt zur Meldung »SYNTAX ERROR IN 500«).

Weniger tolerant sind die Routinen jedoch, wenn die Syntax beim Aufruf stimmt, Sie jedoch »unmögliche« Werte übergeben! Wenn Sie zum Beispiel den Cursor auf Zeile 27 setzen wollen. Das Schlimmste, was Ihnen passieren kann, ist der Aufruf einer Routine mit einer falschen Adresse. Nehmen wir an, *CRS* enthält nach dem Start des Initialisierungsprogramms die korrekte Adresse der Routine zum Setzen des Cursors, nämlich 38918.

In Ihrem Hauptprogramm verwenden Sie jedoch eine Variable namens *CR* und weisen ihr den Wert 2 zu (CR=2). Da für den Interpreter nur die beiden ersten Stellen eines Variablennamens signifikant sind, kennzeichnen *CRS* und *CR* die gleiche Variable. Beim Aufruf mit

```
SYS CRS, 10,12
```

wird sich Ihr C64 mit ziemlicher Sicherheit »aufhängen«, da beim Aufruf der Routine mit der falschen Adresse 2 gewissermaßen ins Leere gesprungen wird.

Der falsche Einsatz der Utilities ist also nicht ganz ungefährlich. Während der Eingewöhnungsphase empfehle ich Ihnen, bei der Arbeit an einem größeren Projekt Ihr Programm in regelmäßigen Abständen zu speichern. Irrtümmer aufgrund des falschen Einsatzes einer Routine sind weniger ärgerlich, wenn nicht das Resultat mehrerer Arbeitsstunden verloren geht.

Noch ein Hinweis zur Verwendung des deutschen Zeichensatzes: Bei Auslösen von RUN/STOP + RESTORE könnte man zunächst meinen, der C64 sei abgestürzt, doch bei »blinder« Eingabe von SYS CH ↵ und anschließendem SHIFT + C= stellt man schnell den Normalstand her. Voraussetzung: Die Variable *CH* muß mit der richtigen Adresse belegt sein.

1

Pull-down-Menüs

»Pull-down-Menüs« ist ein sehr umfangreiches Programm. Dieses Programm ist zur Verwaltung der von Ihnen definierten Pull-down-Menüs gedacht. Das Gesamtprogramm enthält allerdings einige Utilities, die auch einzeln verwendbar sind, mit denen Sie einander überlagernde Windows verwalten oder Ausschnitte des Bildschirms invertieren können und so weiter. Im Einzelnen stehen Ihnen folgende Möglichkeiten zur Verfügung:

- Die Ausgabe einer Anzahl von Strings in einem rechteckigen Kästchen aus Grafikzeichen, einem sogenannten »Window« (PAINTWINDOW)
- Beliebige rechteckige Bildschirmausschnitte (Windows) können Sie vor einem Überschreiben in einen freien Speicherbereich kopieren (retten) und jederzeit aus diesem Bereich wieder auf den Bildschirm zurückkopieren (holen), um den Ausgangszustand wiederherzustellen (WINDOWING)
- Sie können beliebige rechteckige Bildschirmausschnitte schlagartig invertieren oder wieder normalisieren (INVERT)
- Und nicht zuletzt können Sie das Gesamtpaket zur Verwaltung kompletter Pull-down-Menüs einsetzen (CONTROLMENUE).

1.1 BuffersInit

Initialisierung der Window-Tabelle

Adresse `50960`

Variablenname `PINIT`

Syntax `SYS PINIT`

Parameter keine

Funktion BUFFERSINIT ist kein Utility, sondern eine notwendige Voraussetzung zum Einsatz der Utilities WINDOWING und PULLDOWN-MENUE. Beide Utilities verwenden Tabellen, in denen festgehalten wird, wieviele Windows bereits kopiert wurden und wo sich diese im Speicher befinden.

Diese Tabelle muß gelöscht werden, bevor eines der beiden Utilities zum ersten Mal verwendet wird. Diese Aufgabe übernimmt BUFFERSINIT. BUFFERSINIT ist einmalig nach dem Programmstart aufgerufen (übernimmt unser »Standard-Programmvorspann« TOOLS.INIT). Anschließend sind alle Utilities der Pulldown-Menüs ohne Einschränkungen einsatzbereit.

1.2 PaintWindow

Window mit Inhalt ausgeben

Adresse 50963

Variablenname WINDOW

Syntax SYS WINDOW, SPALTE, ZEILE, BREITE, LÄNGE, ARRAY$(START)

Parameter

SPALTE: Spalte der linken oberen Windowecke

ZEILE: Zeile der linken oberen Windowecke

BREITE: Windowbreite in Spalten

LÄNGE: Windowlänge in Zeilen

ARRAY$: (START) Index des ersten Elementes eines Stringarrays, das die im definierten Window auszugebenden Strings enthält

Funktion PAINTWINDOW zeichnet auf dem Bildschirm ein durch seine linke obere Ecke (*SPALTE* und *ZEILE*), seine Breite (*BREITE*) und Länge (*LÄNGE*) definiertes »Window«, ein Rechteck aus Grafikzeichen. Ein Window mit einer definierten Länge von 10 Zeilen besitzt genau 8 »Innenzeilen« (10 Zeilen minus der oberen beziehungsweise unteren Begrenzungslinie).

Diese Innenzeilen werden mit den angegebenen Strings gefüllt, die oberste Zeile mit dem Inhalt von *ARRAY$(START)*, die nächste Zeile mit *ARRAY$(START+1)*, die letzte Innenzeile mit dem Inhalt der Variablen *ARRAY$(START+LÄNGE-2)*.

Beispiel PAINTWINDOW

Nehmen wir an, Sie wollen das Window von Seite 34 auf den Bildschirm ausgeben. Der Inhalt dieses Windows könnte das Hauptmenü einer einfachen Adreßverwaltung sein. In der Abbildung sind die Window-Parameter schlecht zu erkennen. Dieses Window ist durch folgende Parameter definiert:

```
(1) = Adressen eintragen
(2) = Adressen suchen
(3) = Adressen ausdrucken
(4) = Adressen sortieren
(5) = Neue Adressdatei
(6) = Directory anzeigen
(7) = Disk-Kommando
(8) = Programm beenden
```

SPALTE: 2

ZEILE: 3

BREITE: 29 Spalten

LÄNGE: 10 Zeilen

SPALTE/ZEILE: kennzeichnen hierbei die Koordinaten der linken oberen Ecke (┌).

PAINTWINDOW malt dieses Window komplett mit Inhalt an der angegebenen Position, wenn zuvor die Innenzeilen definiert wurden, zum Beispiel im Array *A$(..)*:

```
A$(1) = "(1) = Adressen eintragen"
A$(2) = "(2) = Adressen suchen"
A$(3) = "(3) = Adressen ausdrucken"
A$(4) = "(4) = Adressen sortieren"
A$(5) = "(5) = Neue Adressdatei"
A$(6) = "(6) = Directory anzeigen"
A$(7) = "(7) = Disk-Kommando"
A$(8) = "(8) = Programm beenden"
```

Genau diese Aufgabe (Definition der Innenzeilen und Aufruf von PAINTWINDOW) besitzt das Demoprogramm (Listing 1.1).

In den Zeilen 250–320 wird der Inhalt der Innenzeilen definiert. Anschließend genügt ein Aufruf von PAINTWINDOW unter Angabe der Window-Parameter und des Index des ersten Stringarray-Elementes, um das komplette Window auszugeben (Zeile 360).

Zuvor wird mit PRINT CHR$(18); der Invers-Modus eingeschaltet (Zeile 355), um das Window komplett invers darzustellen. Nach der Ausgabe wird der Invers-Modus wieder ausgeschaltet (Zeile 370).

```
200 rem ******************************
210 rem *        hauptprogramm         *
220 rem ******************************
230 :
240 rem *** innenzeilen definieren **
250 a$(1)=" (1) = Adressen eintragen"
260 a$(2)=" (2) = Adressen suchen"
270 a$(3)=" (3) = Adressen ausdrucken"
280 a$(4)=" (4) = Adressen sortieren"
290 a$(5)=" (5) = Neue Adressdatei"
300 a$(6)=" (6) = Directory anzeigen"
310 a$(7)=" (7) = Disk-Kommando"
320 a$(8)=" (8) = Programm beenden"
330 :
340 :
350 rem *** window malen ***
355 print chr$(18);:rem revers on
360 sys window, 2, 3, 29, 10, a$(1)
370 print chr$(146):rem revers off
ready.
```

Listing 1.1: *Hauptprogramm von PAINTWINDOW*

Hinweis PAINTWINDOW gibt ein Window immer in der Farbe WEISS aus, um es gegenüber dem normalerweise dunkleren Untergrund hervorzuheben. Sollten Sie gegen diese Darstellung etwas einzuwenden haben, können Sie das Programm »patchen«, das heißt individuell verändern. In der Speicherstelle 51110 befindet sich der Farbcode jener Farbe, in der das Window gezeichnet wird. Wenn gewünscht, kann mit einem entsprechenden POKE-Befehl die Voreinstellung 1 (Farbe WEISS) geändert werden.

Beispiel Soll im Demoprogramm das Window in der Farbe GELB (Farbcode 7) statt WEISS ausgegeben werden, »poken« Sie vor dem Aufruf von PAINTWINDOW diesen Farbcode in die Speicherstelle 51110, das heißt, Sie ändern zum Beispiel Zeile 360 in:

```
POKE 51110,7 : SYS WINDOW, 2, 6, 29, 10, A$(1)
```

Hinweis Unmittelbar vor der Ausgabe eines Windows sollten Sie mit PRINT CHR$(18); (Semikolon nicht vergessen!) den Invers-Modus einschalten. Durch die inverse Ausgabe wirkt das Window weit »ansprechender« als in der Normaldarstellung.

1.3 Windowing

Retten/holen eines Bildschirmausschnitts

Adresse 50966

Variablenname PUFFER

Syntax SYS PUFFER, SPALTE, ZEILE, BREITE, LÄNGE, FLAG

Parameter

SPALTE:	Spalte der linken oberen Windowecke
ZEILE:	Zeile der linken oberen Windowecke
BREITE:	Windowbreite in Spalten
LÄNGE:	Windowlänge in Zeilen
FLAG:	0 = Window retten / 1 = Window holen

Funktion WINDOWING erlaubt Ihnen professionelles »Windowing«. Das heißt, Sie können einen beliebigen rechteckigen Ausschnitt des Bildschirms (oder auch den ganzen Bildschirm) mit einem Aufruf an WINDOWING in einen freien Speicherbereich (einen »Window-Puffer«) kopieren.

WINDOWING kümmert sich selbständig darum, wohin der Ausschnitt kopiert wird und verwaltet entsprechende Tabellen mit der Anzahl und den Orten der geretteten Ausschnitte.

Nach dem Retten kann der betreffende Bildschirmausschnitt überschrieben werden. Mit einem weiteren Aufruf von WINDOWING können Sie den kopierten Inhalt jederzeit zurückkopieren lassen – das heißt, der ursprüngliche Bildschirminhalt vor dem Überschreiben des Ausschnittes wird wiederhergestellt.

WINDOWING gestattet Ihnen die Überlagerung beliebig vieler Windows. In diesem Fall wird zuerst Ausschnitt Nummer 1 gerettet, dann Ausschnitt Nummer 2 und so weiter. Bei der Wiederherstellung müssen die geretteten Ausschnitte in umgekehrter Reihenfolge zurückkopiert werden, also zuerst Ausschnitt Nummer 2 und dann Ausschnitt Nummer 1 (LIFO-Struktur »last in, first out«, analog der Stack-Verwaltung).

```
200 rem ******************************
210 rem *       hauptprogramm          *
220 rem ******************************
230 :
240 rem *** innenzeilen definieren **
250 a$(1)=" (1) = Adressen eintragen"
260 a$(2)=" (2) = Adressen suchen"
270 a$(3)=" (3) = Adressen ausdrucken"
280 a$(4)=" (4) = Adressen sortieren"
290 a$(5)=" (5) = Neue Adressdatei"
300 a$(6)=" (6) = Directory anzeigen"
310 a$(7)=" (7) = Disk-Kommando"
320 a$(8)=" (8) = Programm beenden"
330 :
332 rem *** pseudo-untergrund ***
334 print chr$(147);:rem clear screen
336 for i=1 to 24
338 : print "Dies ist ein Test der WINDOWING-Routine"
340 next i
342 :
350 rem *** window ein-/ausblenden **
355 sys puffer, 2, 3, 29, 10, 0
356 get a$:if a$="" then goto 356
358 print chr$(18);:rem revers on
360 sys window, 2, 3, 29, 10, a$(1)
370 get a$:if a$="" then goto 370
380 sys puffer, 2, 3, 29, 10, 1
ready.
```

Listing 1.2: *Hauptprogramm von WINDOWING1*

Dieses Demoprogramm verwendet das gleiche Window wie das zuletzt vorgestellte Programm PAINTWINDOW.

```
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Di┌──────────────────────────┐-Routine
Di│ (1) = Adressen eintragen │-Routine
Di│ (2) = Adressen suchen    │-Routine
Di│ (3) = Adressen ausdrucken│-Routine
Di│ (4) = Adressen sortieren │-Routine
Di│ (5) = Neue Adressdatei   │-Routine
Di│ (6) = Directory anzeigen │-Routine
Di│ (7) = Disk-Kommando      │-Routine
Di│ (8) = Programm beenden   │-Routine
Di└──────────────────────────┘-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
```

Diesmal wird jedoch vor der Ausgabe des Windows eine Art »Pseudo-Hintergrund« erzeugt (Zeile 334–340). Das Window soll nur kurz eingeblendet werden, bis Sie eine beliebige Taste drücken. Anschließend wird das Window ausgeblendet und der Originaluntergrund soll wieder erscheinen.

Daher muß der Inhalt jenes Ausschnitts, den das Window überschreiben wird, zuvor gerettet werden (Zeile 355). Die WINDOWING-Routine wird aufgerufen, um einen Ausschnitt mit den Eck-Koordinaten 2/3 (*SPALTE/ZEILE*), einer Breite von 29 Spalten (*BREITE*) und einer Länge von 10 Zeilen (*LÄNGE*) zu retten.

»Retten« heißt, für den Parameter *FLAG* ist der Wert 0 anzugeben.

Anschließend wartet das Programm darauf, daß Sie eine beliebige Taste drücken (Zeile 356).

Nach dem Retten des aktuellen Inhaltes des Ausschnitts wird er durch den Aufruf von PAINTWINDOW mit dem Window überschrieben (Zeile 360). Vor der Ausgabe wird zur Verbesserung der Optik der Invers-Modus eingeschaltet (Zeile 358).

Nach der Windowausgabe wartet das Programm erneut, bis Sie eine beliebige Taste drücken (Zeile 370).

Nun wird der ursprüngliche Inhalt des geretteten Ausschnitts durch einen zweiten Aufruf von WINDOWING wiederhergestellt (Zeile 380). Für *FLAG* wird der Wert 1 verwendet, da der Ausschnitt diesmal nicht wie zuvor in einen Puffer zu kopieren, sondern gerade umgekehrt der Inhalt des Pufferspeichers auf den Bildschirm zurückzukopieren ist. Die einzelnen Schritte im Überblick:

1. Vor dem Einblenden des Windows

```
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
```

2. Nach dem Einblenden

```
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Di┌──────────────────────────┐-Routine
Di│ (1) = Adressen eintragen │-Routine
Di│ (2) = Adressen suchen    │-Routine
Di│ (3) = Adressen ausdrucken│-Routine
Di│ (4) = Adressen sortieren │-Routine
Di│ (5) = Neue Adressdatei   │-Routine
Di│ (6) = Directory anzeigen │-Routine
Di│ (7) = Disk-Kommando      │-Routine
Di│ (8) = Programm beenden   │-Routine
Di└──────────────────────────┘-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
```

3. Nach dem Zurückholen des geretteten Ausschnitts

```
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
```

Übrigens: Vielleicht wundern Sie sich, daß in Zeile 358 der Invers-Modus ein-, anschließend aber nicht wieder ausgeschaltet wird. Vielleicht wissen Sie, daß jede PRINT-Anweisung (Ausgabe des RETURN-Codes) den Invers-Modus immer ausschaltet. Genau das macht zufällig auch die WINDOWING-Routine, so daß ein explizites Ausschalten überflüssig ist!

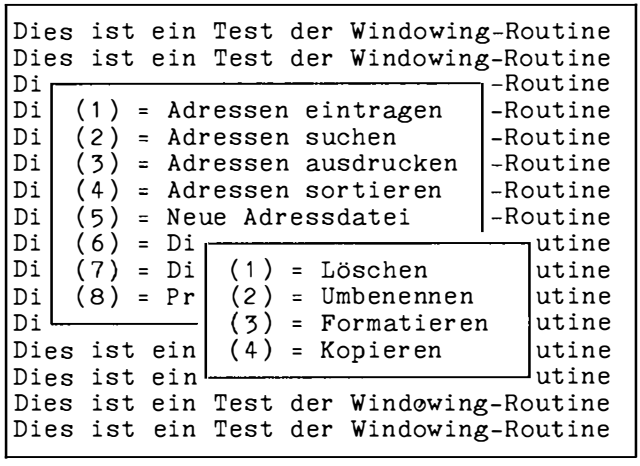
Beispiel WINDOWING2

Stellen Sie sich vor, im letzten Demoprogramm suchen Sie sich aus dem Menü den Befehl aus: »DISK-KOMMANDO«. Anschließend erscheint ein weiteres Menü, aus dem Sie sich das gewünschte Disk-Kommando aussuchen können.

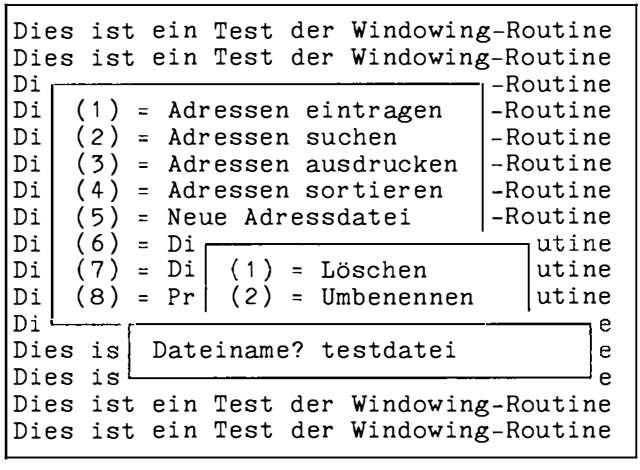
Optisch ansprechend sieht das Ganze aus, wenn es in Form überlappender Windows realisiert wird, wie auf Seite 41.

Nach dem Einblenden des Hauptmenüs und Auswahl einer Funktion wird daher ein weiteres Menü eingeblendet – allerdings erst, nachdem auch der von diesem Menü verdeckte Untergrund zuvor mit WINDOWING gerettet wurde.

Denn nach der Anwahl und Ausführung eines Kommandos soll dieses zweite Menü ausgeblendet und das – inzwischen teilweise verdeckte – Hauptmenü wieder vollständig sichtbar sein. Und anschließend kann auch das Hauptmenü wieder ausgeblendet und der Ausgangszustand wiederhergestellt werden.



Dieses Spiel können Sie beliebig fortsetzen und den Benutzer zum Beispiel nach Anwahl des Kommandos »LÖSCHEN« in einem dritten Window nach dem Namen der zu löschenden Datei fragen.



Das Demoprogramm begnügt sich mit den beiden »Menüwindows« und verzichtet auf ein drittes »Eingabewindow«.

In den Zeilen 240–311 wird der Inhalt des ersten Windows definiert, in den Zeilen 313–320 der des zweiten Windows.

Der aus dem vorigen Demoprogramm bekannte Pseudohintergrund wird ausgegeben (Zeile 332–340), und nun beginnt das

»Fensterln«. Nach jedem Schritt wartet das Programm darauf, daß Sie eine beliebige Taste drücken, bevor es fortfährt (Zeile 356, 390, 410 und 440). Zweimal wird der Invers-Modus eingeschaltet, und zwar jeweils vor der Ausgabe eines der beiden Windows (Zeile 358 und 395).

Noch ist kein Window zu sehen.

```
200 rem ******************************
210 rem *       hauptprogramm          *
220 rem ******************************
230 :
240 rem *** window 1 definieren **
250 a$(1)=" (1) = Adressen eintragen"
260 a$(2)=" (2) = Adressen suchen"
270 a$(3)=" (3) = Adressen ausdrucken"
280 a$(4)=" (4) = Adressen sortieren"
290 a$(5)=" (5) = Neue Adressdatei"
300 a$(6)=" (6) = Directory anzeigen"
310 a$(7)=" (7) = Disk-Kommando"
311 a$(8)=" (8) = Programm beenden"
312 :
313 rem ** window 2 definieren **
314 b$(1)=" (1) =  L:schen"
316 b$(2)=" (2) =  Umbenennen"
318 b$(3)=" (3) =  Formatieren"
320 b$(4)=" (4) =  Kopieren"
330 :
332 rem *** pseudo-untergrund ***
334 print chr$(147);:rem clear screen
336 for i=1 to 24
338 : print "Dies ist ein Test der WINDOWING-Routine"
340 next i
342 :
350 rem *** windows ein-/ausblenden **
355 sys puffer, 2, 3, 29, 10, 0
356 get a$:if a$="" then goto 356
358 print chr$(18);:rem revers on
360 sys window, 2, 3, 29, 10, a$(1)
375 :
380 sys puffer, 12, 9, 22, 6, 0
390 get a$:if a$="" then goto 390
395 print chr$(18);:rem revers on
400 sys window, 12, 9, 22, 6, b$(1)
410 get a$:if a$="" then goto 410
420 sys puffer, 12, 9, 22, 6, 1
430 :
440 get a$:if a$="" then goto 440
450 sys puffer, 2, 3, 29, 10, 1
ready.
```

Listing 1.3: *Hauptprogramm von WINDOWING2*

1. Ausgangszustand

```
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
```

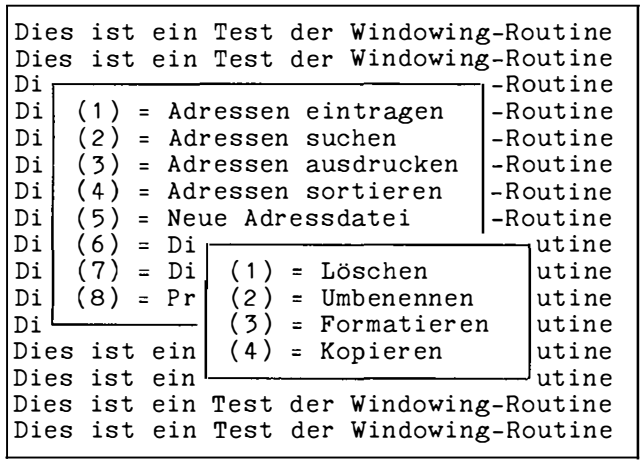
Zeile 355 sorgt für das Retten des Hintergrunds unter dem auszugebenden Hauptmenü. Nachdem Sie eine Taste gedrückt haben, wird das Hauptmenü ausgegeben (Zeile 360).

2. Nach dem Puffern und der Ausgabe des Hauptmenüs

```
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Di                                -Routine
Di  (1) = Adressen eintragen      -Routine
Di  (2) = Adressen suchen         -Routine
Di  (3) = Adressen ausdrucken     -Routine
Di  (4) = Adressen sortieren      -Routine
Di  (5) = Neue Adressdatei        -Routine
Di  (6) = Directory anzeigen      -Routine
Di  (7) = Disk-Kommando           -Routine
Di  (8) = Programm beenden        -Routine
Di                                -Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
```

Der Hintergrund, den das Nebenmenü verdecken wird, rettet das Programm ebenfalls (Zeile 380) und gibt dieses zweite Menü aus (Zeile 400).

3. Nach dem Puffern und der Ausgabe des Nebenmenüs



Nun werden nach jeder Tastenbetätigung die beiden Menüs in umgekehrter Reihenfolge zugeklappt. Zuerst wird der vor der Ausgabe des Nebenmenüs gerettete Hintergrund auf den Bildschirm zurückkopiert (Zeile 420).

4. Nach dem Holen des zweiten geretteten Windowhintergrundes

Das Hauptmenü ist wieder vollständig sichtbar (Seite 45 oben). Die durch die Ausgabe des Nebenmenüs vorgenommenen Veränderungen wurden rückgängig gemacht.

Im letzten Schritt (wieder nach Betätigung einer Taste) stellt das Demoprogramm auch den Originalbildschirminhalt vor Erscheinen des Hauptmenüs wieder her (Zeile 450).

```
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Di                                    -Routine
Di  (1) = Adressen eintragen          -Routine
Di  (2) = Adressen suchen             -Routine
Di  (3) = Adressen ausdrucken         -Routine
Di  (4) = Adressen sortieren          -Routine
Di  (5) = Neue Adressdatei            -Routine
Di  (6) = Directory anzeigen          -Routine
Di  (7) = Disk-Kommando               -Routine
Di  (8) = Programm beenden            -Routine
Di                                    -Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
```

5. Nach dem Holen des ersten geretteten Windowhintergrundes

```
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
Dies ist ein Test der Windowing-Routine
```

Halten Sie beim eigenen »Windowing« bitte folgenden Ablauf ein:

```
- Hintergrund von Window 1 retten ┐
- Window 1 ausgeben               │
- Hintergrund von Window 2 retten │
- Window 2 ausgeben               │
- Hintergrund von Window 3 retten │  Windows
- Window 3 ausgeben               │  1-N
.....                             │  einblenden
.....                             │
- Hintergrund von Window N retten │
- Window N ausgeben               ┘

- Hintergrund von Window N holen  ┐
.....                             │
.....                             │  Windows
- Hintergrund von Window 3 holen  │  1-N
- Hintergrund von Window 2 holen  │  ausblenden
- Hintergrund von Window 1 holen  ┘
```

Hinweis Voraussetzung für die Anwendung von WINDOWING ist, daß BUFFERSINIT durch einmaligen Aufruf initialisiert wurde (geschieht automatisch bei Verwendung des Standard-Initialisierungsprogramms).

Reihenfolge Windows werden in einer LIFO-Struktur (»last in, first out«) verwaltet. Das heißt, bei gleichzeitiger Verwendung von mehr als einem Window müssen die Windows nach dem Einblenden in umgekehrter Reihenfolge wieder ausgeblendet werden. Assembler-Programmierer sollten zum besseren Verständnis das Retten/Holen eines Windows mit einer PHA/PLA-Anweisung vergleichen.

Window-Anzahl Ein Programm kann beliebig viele Windows nacheinander verwalten. Begrenzt ist nur die Größe gleichzeitig zu pufferndor Windows (überlappende Windows). Die Maximalanzahl gleichzeitig zu pufferndor Windows ist prinzipiell unbegrenzt. Praktisch wird sie von der Größe des zur Verfügung stehenden Pufferspeichers und der Größe der gepufferten Windows begrenzt. Der Window-Pufferspeicher umfaßt 2 Kbyte. Da für jedes zu pufferndo Zeichen 2 Byte benötigt werden (Zeichen- und Farbinformation), können maximal 1024 Zeichen gepuffert werden. Den benötigten Pufferplatz können Sie selbst errechnen:

PLATZBEDARF = 2 * BREITE * LÄNGE

Für die zwei überlappenden Windows des Demoprogramms errechnet sich ein Platzbedarf von 844 Byte. Es wäre also noch ausreichend Pufferspeicher für ein drittes überlappendes Window vorhanden.

In der Praxis heißt das: Auch bei vielen überlappenden Windows müssen Sie sich kaum Gedanken über den benötigten Pufferplatz machen. Die Grenze von 1024 Zeichen ist eher theoretischer Natur und wird in der Praxis nicht erreicht.

1.4 Invert

Ausschnitt invertieren/normalisieren

Adresse 50969

Variablenname INVERT

Syntax SYS INVERT, SPALTE, ZEILE, BREITE, LÄNGE, FLAG

Parameter

SPALTE:	Spalte der linken oberen Windowecke
ZEILE:	Zeile der linken oberen Windowecke
BREITE:	Windowbreite in Spalten
LÄNGE:	Windowlänge in Zeilen
FLAG:	1 = invertieren / 0 = normalisieren

Funktion INVERT invertiert/normalisiert einen rechteckigen Bildschirmausschnitt (ein »Window«). Ob invertiert oder normalisiert wird, bestimmt der Parameter *FLAG* (1=invertieren / 0= normalisieren).

Beispiel

```
SYS INVERT, 0, 0, 40, 25, 1
```

invertiert den gesamten Bildschirm.

Beispiel

```
SYS INVERT, 0, 0, 20, 25, 0
```

normalisiert die linke Bildschirmhälfte wieder.

Dieses Demoprogramm verwendet INVERT, um ein mit PAINT-WINDOW erzeugtes Window blinken zu lassen. Zuerst wird der Windowinhalt definiert (Zeile 250–320) und das Window ausgegeben (Zeile 360).

Nun tritt INVERT in Aktion. In einer Endlosschleife erhält *F* abwechselnd den Wert 0 beziehungsweise 1 (Zeile 390–400). Entsprechend wird mit dem folgenden Aufruf von INVERT das Window abwechselnd invertiert und wieder normalisiert (Zeile 410). Resultat: Das Window »blinkt«.

```
200 rem ******************************
210 rem *       hauptprogramm          *
220 rem ******************************
230 :
240 rem *** innenzeilen definieren **
250 a$(1)=" (1) = Adressen eintragen"
260 a$(2)=" (2) = Adressen suchen"
270 a$(3)=" (3) = Adressen ausdrucken"
280 a$(4)=" (4) = Adressen sortieren"
290 a$(5)=" (5) = Neue Adressdatei"
300 a$(6)=" (6) = Directory anzeigen"
310 a$(7)=" (7) = Disk-Kommando"
320 a$(8)=" (8) = Programm beenden"
330 :
340 :
350 rem *** window malen ***
360 sys window, 2, 3, 29, 10, a$(1)
370 :
380 rem *** blinkendes window ***
390 if f=0 then f=1:goto 410
400 f=0
410 sys invert, 2, 3, 29, 10, f
420 for i=1 to 100:next i
430 goto 390
ready.
```

Listing 1.4: *Hauptprogramm von INVERT*

1.5 ControlMenü

Pull-down-Menüs verwalten

Adresse 50972

Variablenname CNTROL

Syntax `SYS CNTROL,HEADER$,MENUE$(START),MENUE%,KOMMANDO%`

Parameter

HEADER$:	String mit »Header-« oder »Kopfzeile«
MENUE$(START):	Stringarray mit Menüdefinitionen
MENUE%:	Integervariable, in der die Nummer des zuletzt selektierten Menüs übergeben wird
KOMMANDO%:	Integervariable, in der die Nummer des gewählten Menükommandos übergeben wird

Funktion CONTROLMENUE ist ein Steuerungsprogramm, das die Routinen PAINTWINDOW, WINDOWING und INVERT zur »intelligenten« Steuerung von Pull-down-Menüs im Folgenden einsetzt:

Bild 1: *Pull-down-Menü von »Datei«*

Bild 2: *Pull-down-Menü von »File«*

Ein Pull-down-Menü besteht aus einer »Headerzeile«

Datei File Ausgabe Sonstiges

mit den Namen der einzelnen Menüs und natürlich den Menüs selbst. Zur Auswahl eines Kommandos besitzen Sie folgende Möglichkeiten:

1. Steuerung mit den Cursortasten

[Cursor →] und [Cursor ←] inaktivieren das momentan aufgeklappte Menü und aktivieren das Menü rechts beziehungsweise links davon.

[Cursor ↑] und [Cursor ↓] bewegen den inversen Balken – den Cursor – innerhalb eines Menüs von Kommando zu Kommando.

Sie können also ein gewünschtes Kommando anwählen, indem Sie sich mit [Cursor →]/[Cursor ←] von Menü zu Menü und – wenn das betreffende Menü aktiviert ist – mit [Cursor ↑]/[Cursor ↓] von Kommando zu Kommando vortasten.

Wenn Sie das gewünschte Kommando selektiert haben, drücken Sie einfach [↵], um die Auswahl dieses Kommandos zu bestätigen. PULL-DOWN-MENUE wird verlassen. Das Utility übergibt Ihr Basic-Programm in jenen Integervariablen, die Sie für die Parameter *MENUE%* und *KOMMANDO%* angaben, die Nummer des zuletzt aktiven Menüs und die Nummer des darin selektierten Kommandos.

Beide Werte beginnen ab 0. War das erste Pull-down-Menü (ganz links) aktiv, erhalten Sie als Menünummer eine 0. Selektierten Sie in diesem Menü das oberste Kommando, erhalten Sie als Kommandonummer ebenfalls eine 0.

[Cursor ↓]: Markierung nach unten bewegen

[Cursor ↑]: Markierung nach oben bewegen

[Cursor →]: Nächstes Menü aktivieren

[Cursor ←]: Voriges Menü aktivieren

2. Direktanwahl

Als schnellere Alternative zur Cursorsteuerung bietet Ihnen PULL-DOWN-MENUE die Direktanwahl von Menüs und Kommandos. Sie funktioniert so: wenn Sie den Anfangsbuchstaben eines Menünamens eingeben, wird das betreffende Menü direkt aktiviert.

Achtung: Der Anfangsbuchstabe muß in GROSS-SCHRIFT eingegeben werden, zusammen mit der [SHIFT]-Taste! Probieren Sie es aus, um das Menü mit dem Namen »File« zu aktivieren. Drücken Sie gleichzeitig [SHIFT] + [F] wie »File«. Das »File«-Menü wird aktiviert.

Ähnlich können Sie auch ein beliebiges Kommando innerhalb eines Menüs direkt über dessen Anfangsbuchstaben anwählen. Die [SHIFT]-Taste wird nicht benötigt. Drücken Sie die Taste [D] (ohne [SHIFT]), um das Kommando »Directory« auszuwählen.

Menüanwahl: [SHIFT] + Anfangsbuchstabe des Menünamens

Kommandoanwahl: Anfangsbuchstabe des Kommandonamens

Sie sehen, Sie haben die Wahl: Zum Eingewöhnen ist die Steuerung über die Cursortasten sicher geeigneter. Wenn Sie irgendwann mit den Menü- und Kommandonamen vertraut sind, werden Sie diese Steuerungsmethode womöglich als langsam und umständlich empfinden. Dann steht es Ihnen frei, auf die Direktanwahl umzusteigen.

Vor dem Aufruf von PULL-DOWN-MENUE müssen Sie die Headerzeile und die Menüs definieren. Für die Headerzeile verwenden Sie eine Stringvariable, zum Beispiel *HK$*:

```
hk$ = "  Datei   File   Ausgabe   Sonstiges"
```

Der String muß wie in diesem Beispiel mit mindestens zwei Leerzeichen beginnen, da die Pull-down-Menüs zwei Spalten links der zugehörigen Menünamen beginnen (siehe Abbildungen).

Die Menüs definieren Sie in einem Stringarray. Jedes Menü wird nach folgendem Schema definiert:

- String mit Leerzeichen, dessen Länge der gewünschten Menübreite entspricht

- Für jede der Innenzeilen ein String mit dem gewünschten Inhalt

Die Definition des in den Abbildungen gezeigten Pull-down-Menüs sieht unter Verwendung des Arrays *HM$(..)* wie nachstehend aus.

Sehr wichtig ist der Abschluß dieser Definitionen. Als Endekennzeichen muß der Definition des letzten Menüs unbedingt ein String folgen, der genau ein Leerzeichen enthält!

Beim Aufruf geben Sie den String mit der Headerzeile (*HK$*), den ersten String der Menüdefinitionen (*HM$(1)*) und zwei Integervariablen an.

```
dim hm$(24):rem menue-array
hk$ = "  Datei   File   Ausgabe   Sonstiges"

hm$(1) = "                    "  ┐
hm$(2) = " Eintragen"            │
hm$(3) = " Suchen/Edit"          │  Definition
hm$(4) = " Ausgabefolge"         │  des ersten
hm$(5) = " Reorganisation"       │  Menüs
hm$(6) = " Bearbeiten"           │
hm$(7) = " Importieren"          ┘
hm$(8) = "                  "    ┐
hm$(9) = " Directory"            │
.....                            │  Definition
.....                            │  des folgenden
.....                            │  Menüs
.....                            ┘
.....                            ┐
hm$(24) = " "                    ┘  Endekennzeichen!!!
```

Sie können nun in den Menüs umherblättern und sich ein Kommando aussuchen. Nach der Anwahl eines Kommandos wird PULL-DOWN-MENUE Ihrem Basic-Programm, wie erläutert, in der ersten der beiden Integervariablen die Menü- und in der zweiten die Kommandonummer mitteilen.

Dieses Demoprogramm erzeugt ein Pull-down-Menü, das exakt den beiden Abbildungen entspricht. In den Zeilen 240 bis 490 wird das Menü definiert.

Um das Ganze optisch zu verschönern, wird als Farbe des Bildschirmrahmens Dunkelblau und als Zeichenfarbe Gelb eingestellt (Zeile 591–592). In der untersten Bildschirmzeile fordert Sie das Programm auf, ein Kommando auszuwählen (Zeile 593).

Vor dem Aufruf von PULL-DOWN-MENUE schaltet `PRINT CHR$(18);` die inverse Darstellung ein (invers ausgegeben wirken die Pull-down-Menüs erst richtig attraktiv).

Nach dem Aufruf (Zeile 600) befinden sich in *M%* und *B%* die – PULL-DOWN-MENUE zählt ab 0! – Kommando- und Menünummer, die auf dem Bildschirm ausgegeben werden (Zeile 610).

```
200 rem ******************************
210 rem *       hauptprogramm          *
220 rem ******************************
230 :
240 rem *** menue - definitionen ***
245 dim hm$(24):rem menue-array
250 hk$="  Datei    File    Ausgabe  Sonstiges     "
260 hm$(1)= "                    "
270 hm$(2)= " Eintragen"
280 hm$(3)= " Suchen/Edit"
290 hm$(4)= " Ausgabefolge"
300 hm$(5)= " Reorganisation"
310 hm$(6)= " Bearbeiten"
320 hm$(7)= " Importieren"
330 hm$(8)= "                  "
340 hm$(9)= " Directory"
350 hm$(10)=" Umbenennen"
360 hm$(11)=" L:schen"
370 hm$(12)=" Formatieren"
380 hm$(13)="                  "
390 hm$(14)=" Satz"
400 hm$(15)=" Teilmenge"
410 hm$(16)=" Datei"
420 hm$(17)=" Parameter"
430 hm$(18)=" Codetabelle"
440 hm$(19)=" Ger;t"
450 hm$(20)="                  "
460 hm$(21)=" Makros laden"
470 hm$(22)=" Editor"
480 hm$(23)=" Beenden"
490 hm$(24)=" "
500 :
510 :
590 rem *** aufruf ***
591 poke 53280,6:rem rahmen dunkelblau
592 print chr$(158):rem zeichenfarbe gelb / revers on
593 sys crs,0,24:print chr$(18) " Bitte w;hlen Sie ein Kommando aus     ";
596 :
600 print chr$(18);:sys cntrol, hk$, hm$(1), m%, b%
605 :
610 print:print "Men$>" m%,"Kommando>" b%
ready.
```

Listing 1.5: *Hauptprogramm von PULL-DOWN-MENUE*

2

Floppy

Die folgenden Routinen erleichtern und beschleunigen den Umgang mit der Floppy:

- Schnelles Speichern einer Variablen, eines Ausschnitts oder gar kompletten Arrays beliebigen Typs (FASTSAVE)
- Schnelles Laden der mit FASTSAVE gespeicherten Daten (FASTLOAD)
- Speichern eines beliebigen Speicherbereichs (MEMORYSAVE)
- Laden des Speicherbereichs (MEMORYLOAD)
- Directory in Stringarray einlesen (DIRECTORY)

2.1 FastSave

Variable/Array speichern

Adresse 38924

Variablenname DAUSGABE

Syntax `SYS DAUSGABE, LFN, ANZAHL, VARIABLE`

Parameter

LFN:	logische Filenummer der geöffneten Datei
ANZAHL:	Anzahl der zu speichernden Variablen
VARIABLE:	Einfache oder indizierte Variable beliebigen Typs (beim Speichern eines Arrays Index der ersten zu speichernden Variablen des betreffenden Arrays)

Funktion FASTSAVE speichert eine einfache Variable beliebigen Typs (*A*, *A%*, *A$*), einen Teil oder ein komplettes Array beliebigen Typs (*A(..)*, *A%(..)*, *A$(..)*.

Ebenso wie bei der PRINT#-Anweisung muß vor Aufruf von FASTSAVE mit der OPEN-Anweisung eine logische Datei geöffnet sein.

Beispiel Speichern einer einfachen Variablen *A$* in der sequentiellen Datei »TEST«.

```
OPEN 2,8,2,"TEST,S,W"
SYS DAUSGABE, 2, 1, A$
CLOSE 2
```

Beispiel Speichern des Integerarrays *A%(1)* bis *A%(55)* in der sequentiellen Datei »NUMTEST«.

```
OPEN 2,8,2,"NUMTEST,S,W"
SYS DAUSGABE, 2, 55, A%(1)
CLOSE 2
```

Beispiel Speichern der einfachen Variablen *X$* und der drei Arrays *A$(1)* bis *A$(50)*, *B%(20)* bis *B%(40)* und *C(130)* bis *C(157)* in der sequentiellen Datei »GLOBALTEST«.

```
OPEN 2,8,2,"GLOBALTEST,S,W"
SYS DAUSGABE, 2, 50, A$(1): REM 50 ARRAYVARIABLEN
SYS DAUSGABE, 2, 21, B%(20):REM 21 ARRAYVARIABLEN
SYS DAUSGABE, 2, 28, C(130):REM 28 ARRAYVARIABLEN
SYS DAUSGABE, 2, 1, X$:REM EINE EINFACHE VARIABLE
CLOSE 2
```

```
200 rem ******************************
210 rem *       hauptprogramm          *
220 rem ******************************
230 :
235 print "Bitte warten> ich erzeuge die Arrays"
240 dim a$(50), b%(40), c(157)
250 for i=1 to 50
260 : a$(i)="String"+str$(i)
270 next i
280 :
290 for i=20 to 40
300 : b%(i)=int(rnd(0)*100)
310 next i
320 :
330 for i=130 to 157
340 : c(i)=rnd(0)
350 next i
360 :
365 x$=chr$(34)+",><,.?/!#$%&'()+-1234567890"
370 x$=x$+",><:;§öabcdefghijklmnopqrstuvwxyz"
380 x$=x$+"äÜ:ABCDEFGHIJKLMNOPQRSTUVWXYZ"
400 :
410 rem *** speichern ***
415 open 1,8,15,"s:globaltest":close 1
420 open 2,8,2,"globaltest,s,w"
430 sys dausgabe,2,50,a$(1)
440 sys dausgabe,2,21,b%(20)
450 sys dausgabe,2,28,c(130)
460 sys dausgabe,2,1,x$
470 close 2
480 :
490 for i=1 to 50:a$(i)="":next i
492 for i=20 to 40:b%(i)=0:next i
494 for i=130 to 157:c(i)=0:next i
496 x$=""
500 :
510 rem *** laden ***
520 open 2,8,2,"globaltest,s,r"
530 sys deingabe,2,50,a$(1)
540 sys deingabe,2,21,b%(20)
550 sys deingabe,2,28,c(130)
560 sys deingabe,2,1,x$
570 close 2
580 :
585 print:print "Stringarray>"
590 for i=1 to 50:print a$(i):next i
595 print:print "Integerarray>"
600 for i=20 to 40:print b%(i);:next i
605 print:print:print "Realarray>"
610 for i=130 to 157:print c(i);:next i
615 print:print:print "String>"
620 print x$
ready.
```

Listing 2.1: *Hauptprogramm von FASTSAVE/LOAD*

Dieses Programm erzeugt die im letzten Beispiel verwendete einfache Variable *X$*, die drei Arrays *A$(..)*, *B%(..)* und *C(..)* (Zeile 235–90). Die numerischen Arrays enthalten Zufallszahlen, das Stringarray die Zeichenketten »String 1«, »String 2« etc.

Nach dem Löschen einer eventuell bereits vorhandenen Datei »GLOBALTEST« (Zeile 415) wird unter diesem Namen eine Datei geöffnet und alle Variablen mit FASTSAVE werden in ihr gespeichert (Zeile 420–470). Die Variablen im Speicher werden gelöscht (Zeile 490–496), die komplette Datei mit FASTLOAD eingelesen (Zeile 520–570) und die Inhalte der Variablen zur Kontrolle auf den Bildschirm ausgegeben (Zeile 585–620).

2.2 FastLoad

Variable/Array laden

Adresse 38927

Variablenname DEINGABE

Syntax `SYS DEINGABE, LFN, ANZAHL, VARIABLE`

Parameter

LFN:	logische Filenummer der geöffneten Datei
ANZAHL:	Anzahl der zu ladenden Variablen
VARIABLE:	Einfache oder indizierte Variable beliebigen Typs (beim Laden eines Arrays Index der ersten zu ladenden Variablen des betreffenden Arrays)

Funktion FASTLOAD lädt eine mit FASTSAVE gespeicherte einfache Variable beliebigen Typs (*A*, *A%*, *A$*), einen Teil oder ein komplettes Array beliebigen Typs *(A(..), A%(..), A$(..)*.

Ebenso wie bei der INPUT#-Anweisung muß vor Aufruf von FASTLOAD mit der OPEN-Anweisung eine logische Datei geöffnet sein.

Hinweis Im Gegensatz zur INPUT#-Anweisung klappt mit FASTLOAD auch das Lesen von Strings, die länger als 88 Zeichen sind (maximal 255 Zeichen).

Außerdem dürfen zu lesende Strings alle (!) Zeichen enthalten, auch die mit INPUT# nicht verträglichen Zeichen » " «, » , «, » ; « und » : «.

Als dritte Verbesserung liest FASTLOAD auch problemlos »leere Zeichenketten« ein, was mit INPUT# nicht möglich ist und oft zu folgenden Konstruktionen führt:

Schreiben:

```
IF A$="" THEN PRINT#2,"*"
```

Lesen:

```
INPUT#2,A$ : IF A$="*" THEN A$=""
```

Derartige Konstruktionen sind bei Verwendung von FASTSAVE und FASTLOAD nicht mehr nötig!

Das heißt, diese Routinen bieten Ihnen alle Möglichkeiten der GET#-Anweisung, um die Schwächen von INPUT# zu umgehen, obwohl die Geschwindigkeit sogar weit höher als bei PRINT# und INPUT# ist!

Beispiel Laden einer einfachen Variablen *A$* aus der sequentiellen Datei »TEST«.

```
OPEN 2,8,2,"TEST,S,R"
SYS DEINGABE, 2, 1, A$
CLOSE 2
```

Beispiel Laden des Integerarrays *A%(1)* bis *A%(55)* aus der sequentiellen Datei »NUMTEST«.

```
OPEN 2,8,2,"NUMTEST,S,R"
SYS DEINGABE, 2, 55, A%(1)
CLOSE 2
```

Beispiel Laden der einfachen Variablen *X$* und der drei Arrays *A$(1)* bis *A$(50)*, *B%(20)* bis *B%(40)* und *C(130)* bis *C(157)* aus der sequentiellen Datei »GLOBALTEST«.

```
OPEN 2,8,2,"GLOBALTEST,S,R"
SYS DEINGABE, 2, 50, A$(1) :REM 50 ARRAYVARIABLEN
SYS DEINGABE, 2, 21, B%(20):REM 21 ARRAYVARIABLEN
SYS DEINGABE, 2, 28, C(130):REM 28 ARRAYVARIABLEN
SYS DEINGABE, 2, 1, X$     :REM EINFACHE VARIABLE
CLOSE 2
```

Dieses Programm wurde bereits im Abschnitt über FASTSAVE besprochen. Diesmal möchte ich speziell auf die Möglichkeiten beim Lesen von Strings eingehen.

Das Programm erzeugt die im letzten Beispiel verwendete einfache Variable *X$*, die drei Arrays *A$(..)*, *B%(..)* und *C(..)* (Zeile 235–390). Die numerischen Arrays enthalten Zufallszahlen, das Stringarray die Zeichenketten »String 1«, »String 2« etc.

Nach dem Löschen einer eventuell bereits vorhandenen Datei GLOBALTEST (Zeile 415) wird unter diesem Namen eine Datei geöffnet und alle Variablen mit FASTSAVE in ihr gespeichert (Zeile 420–470). Die Variablen im Speicher werden gelöscht (Zeile 490–496), die komplette Datei mit FASTLOAD eingelesen (Zeile 520–570) und die Inhalte der Variablen zur Kontrolle auf dem Bildschirm ausgegeben (Zeile 585–620).

Beachten Sie nun bitte das völlig korrekte Einlesen von *X$*. Und das, obwohl diese Zeichenkette unter anderem die für INPUT# verbotenen Zeichen » " , : ; « enthält und länger als 88 Zeichen ist!

Das heißt, diese könnten Sie zwar speichern, keinesfalls jedoch mit INPUT# wieder einlesen – aber völlig problemlos mit FAST-LOAD, und zwar mit weit höherer Geschwindigkeit, als dies in einer GET#-Schleife möglich wäre!

```
200 rem ******************************
210 rem *       hauptprogramm          *
220 rem ******************************
230 :
235 print "Bitte warten> ich erzeuge die Arrays"
240 dim a$(50), b%(40), c(157)
250 for i=1 to 50
260 : a$(i)="String"+str$(i)
270 next i
280 :
290 for i=20 to 40
300 : b%(i)=int(rnd(0)*100)
310 next i
320 :
330 for i=130 to 157
340 : c(i)=rnd(0)
350 next i
360 :
365 x$=chr$(34)+",><,.?/!#$%&'()+-1234567890"
370 x$=x$+",><:;§öabcdefghijklmnopqrstuvwxyz"
380 x$=x$+"äÜ:ABCDEFGHIJKLMNOPQRSTUVWXYZ"
400 :
410 rem *** speichern ***
415 open 1,8,15,"s:globaltest":close 1
420 open 2,8,2,"globaltest,s,w"
430 sys dausgabe,2,50,a$(1)
440 sys dausgabe,2,21,b%(20)
450 sys dausgabe,2,28,c(130)
460 sys dausgabe,2,1,x$
470 close 2
480 :
490 for i=1 to 50:a$(i)="":next i
492 for i=20 to 40:b%(i)=0:next i
494 for i=130 to 157:c(i)=0:next i
496 x$=""
500 :
510 rem *** laden ***
520 open 2,8,2,"globaltest,s,r"
530 sys deingabe,2,50,a$(1)
540 sys deingabe,2,21,b%(20)
550 sys deingabe,2,28,c(130)
560 sys deingabe,2,1,x$
570 close 2
580 :
585 print:print "Stringarray>"
590 for i=1 to 50:print a$(i):next i
595 print:print "Integerarray>"
600 for i=20 to 40:print b%(i);:next i
605 print:print:print "Realarray>"
610 for i=130 to 157:print c(i);:next i
615 print:print:print "String>"
620 print x$
ready.
```

Listing 2.2: *Hauptprogramm von FASTSAVE/LOAD*

2.3 MemorySave

Speicherbereich speichern

Adresse 38933

Variablenname SPEICHERN

Syntax `SYS SPEICHERN, FILENAME$, START, ENDE+1`

Parameter

FILENAME$:	Vollständiger (!) Dateiname
START:	Startadresse des zu speichernden Bereichs
ENDE+1:	Endadresse(+1) des zu speichernden Bereichs

Funktion MEMORYSAVE speichert einen beliebigen Speicherbereich in einer Datei. Als Dateiname ist der vollständige (!) Name anzugeben. Das heißt, wenn Speicherbereiche wie üblich in einer Programmdatei gespeichert werden, auch der Zusatz »,P,W« (für »Programmdatei« und »Write«).

Beispiel Ab Adresse 1536 beginnt die Tabelle mit den Tastatur-Makros (siehe MASKROSINIT). Die Tabelle umfaßt 255 Zeichen, das heißt, sie endet bei Adresse 1536+255. Das Speichern dieser Tabelle ist gleichbedeutend mit dem Speichern aller momentan definierten Tastaturmakros. Gespeichert wird sie unter Angabe eines – vollständigen! – Dateinamens, der Startadresse 1536 und der Endadresse 1536+255 plus 1, also 1536+256:

```
N$="MAKRODEFS,P,W":SYS SPEICHERN,N$,1536,1536+256
```

Nach diesem Aufruf (vorher Schreibschutz entfernen!) erzeugt MEMORYSAVE auf der eingelegten Diskette eine Programmdatei mit dem Namen »MAKRODEFS«, die die Tabelle mit den aktuellen Makrodefinitionen enthält. Diese Tabelle kann jederzeit wieder mit MEMORYLOAD geladen werden.

Dieses Demoprogramm füllt den Bildschirm mit der Zeichenkette »Dies ist ein Test von MemorySave/Load« (Zeile 240–270). Der Bildschirminhalt soll in einer Datei mit dem Namen SCREEN gespeichert werden. Zuvor wird eine eventuell bereits unter diesem Namen existierende Datei gelöscht (Zeile 287).

Der vollständige (inklusive »,P,W«) Dateiname wird in *N$* gespeichert und MEMORYSAVE unter Angabe dieses Dateinamens und der Start- beziehungsweise Endadresse des Bildschirmspeichers aufgerufen (Zeile 300).

```
200 rem ********************************
210 rem *        hauptprogramm         *
220 rem ********************************
230 :
240 print chr$(147):rem clear screen
250 for i=1 to 20
260 : print "Dies ist ein Test von MemorySave/Load"
270 next i
280 :
285 rem *** speichern ***
287 open 1,8,15,"s:screen":close 1
290 n$="screen,p,w"
300 sys speichern,n$,49152,49152+1001
310 :
320 print chr$(147):rem clear screen
330 :
340 rem *** laden ***
350 n$="screen,p,r"
360 sys vladen,n$,49152,49152+1001
ready.
```

Listing 2.3: *Hauptprogramm von MEMORYSAVE/LOAD*

Die Startadresse ist nicht wie gewohnt $0400 (dezimal 1024)! Das Initialisierungsprogramm ruft unter anderem CHARACTERSINIT auf, um den deutschen Zeichensatz einzuschalten. CHARACTERSINIT verschiebt den Bildschirmspeicher nach $C000 (dezimal 49152)!

Das heißt, als Startadresse wird 49152 und als Endadresse(+1) 49152+1001 angegeben (1000 Zeichen sind zu speichern).

Das Demoprogramm löscht nach dem Speichern den Bildschirm und lädt anschließend mit MEMORYLOAD den Inhalt der Programmdatei in den Bildschirmspeicher – der Bildschirmzustand vor dem Löschen ist wiederhergestellt!

Hinweis Vollständiger Dateiname heißt: inklusive des Zusatzes »,P,W«, wenn wie üblich beim Speichern eine Programmdatei benutzt wird.

2.4 MemoryLoad

Speicherbereich laden

Adresse 38937

Variablenname VLADEN

Syntax `SYS VLADEN, FILENAME$, START, ENDE+1`

Parameter

FILENAME$:	Vollständiger Dateiname
START:	Beim Speichern angegebene Startadresse
ENDE+1:	Beim Speichern angegebene Endadresse(+1)

Funktion MEMORYLOAD lädt einen mit MEMORYSAVE gespeicherten Speicherbereich. Als Dateiname ist der vollständige (!) Name anzugeben. Wurde der Speicherbereich wie üblich in einer Programmdatei gespeichert, muß der Zusatz »,P,R« (für »Programmdatei« und »Read«) angegeben werden.

Beispiel Ab Adresse 1536 beginnt die Tabelle mit den Tastatur-Makros (siehe MASKROSINIT). Die Tabelle umfaßt 255 Zeichen, das heißt, sie endet bei Adresse 1536+255. Wurde diese Tabelle (und damit die aktuellen Makrodefinitionen) mit MEMORYSAVE in der Programmdatei MAKRODEFS gespeichert, wird sie folgendermaßen mit MEMORYLOAD wieder geladen:

```
N$="MAKRODEFS,P,R" : SYS VLADEN, N$,1536,1536+256
```

Die Endadresse(+1) ist 1536+256! Nach diesem Aufruf sind alle in dieser Datei enthaltenen Makrodefinitionen sofort wieder verwendbar. Das heißt, jene Makros sind wieder funktionsfähig, die beim Zeitpunkt des Speicherns der Tabelle Gültigkeit besaßen.

Dieses bereits bekannte Demoprogramm (siehe MEMORYSAVE) füllt den Bildschirm mit der Zeichenkette »Dies ist ein Test von MemorySave/Load« (Zeile 240–270). Der Bildschirminhalt soll in einer Datei mit dem Namen SCREEN gespeichert werden. Zuvor wird eine eventuell bereits unter diesem Namen existierende Datei gelöscht (Zeile 287).

Der vollständige (inklusive »,P,R«) Dateiname wird in *N$* gespeichert und MEMORYSAVE unter Angabe dieses Dateinamens

und der Start- beziehungsweise Endadresse des Bildschirmspeichers aufgerufen (Zeile 300).

```
200 rem ********************************
210 rem *        hauptprogramm          *
220 rem ********************************
230 :
240 print chr$(147):rem clear screen
250 for i=1 to 20
260 : print "Dies ist ein Test von MemorySave/Load"
270 next i
280 :
285 rem *** speichern ***
287 open 1,8,15,"s:screen":close 1
290 n$="screen,p,w"
300 sys speichern,n$,49152,49152+1001
310 :
320 print chr$(147):rem clear screen
330 :
340 rem *** laden ***
350 n$="screen,p,r"
360 sys vladen,n$,49152,49152+1001
ready.
```

Listing 2.4: *Hauptprogramm von MEMORYSAVE/LOAD*

Die Startadresse ist nicht wie gewohnt $0400 (dezimal 1024)! Das Initialisierungsprogramm ruft unter anderem CHARACTERSINIT auf, um den deutschen Zeichensatz einzuschalten. CHARACTERSINIT verschiebt den Bildschirmspeicher nach $C000 (dezimal 49152)!

Das heißt, als Startadresse wird 49152 und als Endadresse(+1) 49152+1001 angegeben (1000 Zeichen sind zu speichern).

Das Demoprogramm löscht nach dem Speichern den Bildschirm und lädt anschließend mit MEMORYLOAD den Inhalt der Programmdatei in den Bildschirmspeicher – der Bildschirmzustand vor dem Löschen ist wiederhergestellt!

Hinweis Vollständiger Dateiname heißt: inklusive des Zusatzes »,P,R«, wenn wie üblich beim Speichern eine Programmdatei benutzt wurde und deren Inhalt nun wieder geladen werden soll.

Hinweis Bei MEMORYLOAD wird immer absolut geladen. Das heißt, der Inhalt der Datei wird an jene Adresse geladen, an der er sich beim Speichern befand. Daher sind die Parameter *START* und *ENDE+1* »Dummywerte«, für die – bei MEMORYLOAD nicht weiter interessierende – Werte eingesetzt werden können.

2.5 Directory

Directory in Stringarray einlesen

Adresse 38940

Variablenname DIR

Syntax SYS DIR, LFN, EXT$, ARRAY$(START), ANZAHL%

Parameter

LFN:	Logische Filenummer, unter der die Directory-Datei (»$0«) geöffnet wurde
EXT$:	Extension der Dateinamen
ARRAY$: (START)	Index des ersten Elementes des Stringarrays, in das die Directory eingelesen wird
ANZAHL%:	DIRECTORY übergibt in dieser Variablen die Anzahl der gelesenen Directory-Einträge.

Funktion DIRECTORY liest das Inhaltsverzeichnis der momentan eingelegten Diskette. Dieses Verzeichnis wird jedoch nicht einfach auf dem Bildschirm ausgegeben, sondern jeder Eintrag zur freien Weiterverwendung durch den Benutzer in einem Element des angegebenen Stringarrays gespeichert.

Vor dem Aufruf muß eine logische Datei unter dem Namen »$0« mit der Sekundäradresse 0 geöffnet werden. Der erste Eintrag wird in *ARRAY$(START)* gespeichert, der zweite in *ARRAY$(START+1)* und so weiter. In der als Parameter *ANZAHL%* angegebenen Integervariablen übergibt DIRECTORY die Anzahl der gelesenen Einträge. Das heißt, auf den letzten Eintrag können Sie mit *ARRAY$(START+ANZAHL%-1)* zugreifen.

EXT$ ist nützlich, wenn Sie verschiedene Dateiarten durch unterschiedliche Namenserweiterungen oder Extensions kennzeichnen. Zum Beispiel ist es in Maschinensprache oft üblich, für Sourcetexte die Endung .SRC und Objektdateien durch die Endung .OBJ zu verwenden. Oder auch für Basic-Programme die Endung .BAS.

DIRECTORY ermöglicht es, selektiv nur solche Directory-Einträge zu lesen, die die gewünschte Endung besitzen, also zum Beispiel alle Einträge, die mit ».OBJ«, »DEMO«, »WILLI« etc. enden. Beim selektiven Lesen werden die Einträge ohne die spezi-

fische Endung im angegebenen Stringarray gespeichert. Sollen nicht-selektiv alle Einträge gelesen werden, geben Sie als Parameter *EXT$* einen leeren String an.

Bei den folgenden Beispielen wird davon ausgegangen, daß auf der eingelegten Diskette folgende fünf Dateien vorhanden sind:

```
TOOLS.INIT
MC1.OBJ
MC2.OBJ
----------------
INPUT1
```

Beispiel Nicht-selektives Lesen aller Einträge:

```
OPEN 1,8,0,"$0"
EXT$="" : SYS DIR, 1, EXT$, DIR$(1), NR%
CLOSE 1
```

NR% enthält die Anzahl der gelesenen Einträge, im Beispiel fünf. Der Inhalt des Arrays *DIR$(..)*:

```
DIR$(1) = "TOOLS.INIT"
DIR$(2) = "MC1.OBJ"
DIR$(3) = "MC2.OBJ"
DIR$(4) = "----------------"
DIR$(5) = "INPUT1"
```

Beispiel Selektives Lesen aller Einträge mit der Endung ».OBJ«:

```
OPEN 1,8,0,"$0"
EXT$=".OBJ" : SYS DIR, 1, EXT$, DIR$(1), NR%
CLOSE 1
```

NR% enthält die Anzahl der gelesenen Einträge, im Beispiel zwei (zwei Dateien mit der angegebenen Endung ».OBJ«). Der Inhalt des Arrays *DIR$(..)*:

```
DIR$(1) = "MC1"
DIR$(2) = "MC2"
```

Beachten Sie, daß die angegebene Endung beim Speichern entfernt wird. Sie ist überflüssig, denn: wäre sie nicht bekannt, könnten Sie sie wohl kaum angeben.

```
200 rem ********************************
210 rem *        hauptprogramm         *
220 rem ********************************
230 :
240 dim dir$(144):rem dir-array
250 :
260 ext$="":print:input"erweiterung";ext$
265 :
270 open 1,8,0,"$0"
280 sys dir, 1, ext$, dir$(1), nr%
285 close 1
290 :
292 print "Eintr;ge>" nr%:print
300 for i=1 to nr%
310 : print dir$(i)
320 next i
330 goto 260
ready.
```

Listing 2.5: *Hauptprogramm von DIRECTORY*

Dieses Demoprogramm legt zuerst ein Stringarray *DIR$(..)* mit 145 Variablen an (Zeile 240). Bedenken Sie, daß eine Diskette bis zu 144 Einträge enthalten kann!

Sie können nun eine beliebige Namensendung eingeben, die in *EXT$* gespeichert wird (Zeile 260). Sollen alle Einträge gelesen werden, drücken Sie einfach – ohne Eingabe – ↵.

In Zeile 270 wird die Directory-Datei geöffnet. Die Sekundäradresse 0 und der Name »$0« sind fest vorgeschrieben. Verändern Sie bitte keinesfalls einen dieser Parameter!

Der Aufruf von DIRECTORY liest alle Einträge in *DIR$(1)*, *DIR$(2)* etc. ein, die die angegebene Endung besitzen (Zeile 280).

Die Datei wird geschlossen und das Demoprogramm gibt die Zahl der gelesenen Einträge (in *NR%* enthalten) und die Dateinamen selbst auf dem Bildschirm aus.

Hinweis Da sich auf einer Diskette bis zu 144 Dateien (und somit Einträge im Inhaltsverzeichnis) befinden können, sollte das verwendete Array immer mit DIM NAME$(144) dimensioniert werden, um eine ausreichende Anzahl von Stringvariablen anzulegen!

Hinweis Verwenden Sie immer beim Öffnen der Datei die Sekundäradresse 0 und den Dateinamen »$0«!

3

Variablen

Die folgenden Utilities sind vor allem bei der Behandlung großer Variablenmengen sehr nützlich.

- Komfortable »Blättermöglichkeiten« (BLAETTERN)
- Kommando selektieren (SEARCHCOMAND)
- Schnelle und flexible Suchroutine für Stringarrays (SEARCH)
- Extrem schnelle und vielseitige Sortierroutine (QUICKSORT)

BLAETTERN und SEARCHCOMAND gehören zu den erwähnten »Abfallprodukten«. Weitaus wichtiger sind SEARCH und QUICKSORT. Beide Routinen sind in Basic – zumindest so extrem vielseitig – aus Geschwindigkeitsgründen nicht realisierbar. Und für beide Routinen gilt: Trotz des getriebenen »Riesenaufwands« wird nicht ein einziger Stringeintrag angelegt. Auch bei QUICKSORT nicht – die Garbage Collection findet nicht statt (QUICKSORT vertauscht nur Zeiger, keine Variableninhalte)!

3.1 Blättern

Flexibles Blättern in großen Datenmengen

Adresse 38912

Variablenname BLAETTERN

Syntax SYS BLAETTERN, MAX, KOMMANDO, NR%

Parameter

MAX: Gibt die Größe der Datenmengen an, zum Beispiel 100 in einem Array *A$(1)* bis *A$(100)*.

KOMMANDO: Zahl zwischen eins und sechs, die angibt, um wieviel die aktuelle Datennummer erhöht / vermindert werden soll.

1 → NR% = NR% + 1 / 2 → NR% = NR% − 1
3 → NR% = 1 / 4 → NR% = MAX%
5 → NR% = NR% + 10 / 6 → NR% = NR% − 10

NR%: Aktuelle Nummer, die von BLAETTERN je nach Kommando verändert wird

Funktion BLAETTERN ist ein »Mini-Utility«, das den Umgang mit größeren Datenmengen erleichtert. Wie der Name sagt, wird BLAETTERN zum »Umherblättern« in größeren Datenmengen verwendet, in einer großen Tabelle, in umfangreichen Adreßdateien und so weiter.

In all diesen Fällen besitzt der erste Wert (zum Beispiel die erste Adresse) normalerweise die Nummer 1, *MAX* ist die Nummer des letzten Wertes (letzte Adresse). Von großen Datenmengen ist nur ein Teil auf dem Bildschirm sichtbar, zum Beispiel eine einzige Adresse. BLAETTERN erleichtert das schnelle Durchblättern der Daten. *NR%* gibt die Nummer der momentan auf dem Bildschirm sichtbaren Daten an (zwischen 1 und MAX).

Je nach Kommando *KOMMANDO*, einer Zahl zwischen eins und sechs, wird diese Nummer erhöht beziehungsweise verringert. Das Erhöhen/Verringern ist gleichbedeutend mit dem »Vor-/Zurückblättern« in der Tabelle. BLAETTERN kennt folgende Kommandos:

Funktion	KOMMANDO%	Resultat V/NR%
Blättern um 1 weiter	1	NR%=NR%+1
Blättern um 1 zurück	2	NR%=NR%-1
Blättern zum Anfang	3	NR%=1
Blättern zum Ende	4	NR%=MAX%
Blättern um 10 weiter	5	NR%=NR%+10
Blättern um 10 zurück	6	NR%=NR%-10

BLAETTERN ist eine Funktion, die sehr gut mit einem Basic-Programm zu vergleichen ist. Der Aufruf:

```
SYS BLAETTERN, M, K, NR%
```

verändert je nach Inhalt der Variablen *M* (Maximalanzahl) und *I* (Kommando) den Inhalt von *NR%* auf die gleiche Weise wie folgendes Basic-Programm (das allerdings sehr langsam ist, wenn die »Blätterkommandos« schnell aufeinander folgen):

```
100 IF I=1 THEN NR%=NR%+1
110 IF I=2 THEN NR%=NR%-1
120 IF I=3 THEN NR%=1
130 IF I=4 THEN NR%=M
140 IF I=5 THEN NR%=NR%+10
150 IF I=6 THEN NR%=NR%-10
160 IF I<0 THEN I=1
170 IF I>M THEN I=M
```

Beispiel Gehen wir von einer Dateiverwaltung aus, die 100 Adressen enthält (MAX=100). Auf dem Bildschirm sichtbar ist momentan die Adresse Nummer 17 (NR%=17). Der Benutzer soll nun gefragt werden, welche Adresse er nun sehen will:

```
230 PRINT "1 => NAECHSTE ADRESSE"
240 PRINT "2 => VORHERGEHENDE ADRESSE"
250 PRINT "3 => ERSTE ADRESSE"
260 PRINT "4 => LETZTE ADRESSE"
270 PRINT "5 => 10 ADRESSEN WEITERBLAETTERN"
280 PRINT "6 => 10 ADRESSEN ZURUECKBLAETTERN"
290 :
300 INPUT K  :  REM KOMMANDO?
310 SYS BLAETTERN, 100, K, NR%
```

Je nach Kommando *K* (1-6) wird der Inhalt der Integervariablen *NR%*, also die Nummer der aktuellen Adresse, erhöht oder ver-

ringert. Der Benutzer besitzt somit recht variable Möglichkeiten, die Adreßdatei in mehr oder weniger großen Sprüngen zu durchblättern.

Wirklich interessant wird BLAETTERN erst zusammen mit dem Utility SEARCHCOMAND, das einen String nach einem Zeichen durchsucht, zum Beispiel einen String mit allen zulässigen »Blättertasten« nach der gedrückten Taste.

Das in der Abbildung gezeigte Programm erzeugt eine Tabelle mit 100 Zufallszahlen (Zeile 240–280), die Sie mit den Funktionstasten F1, F2, F3, F4, F5 und F6 in kleineren oder größeren Schritten durchblättern können.

Angezeigt werden jeweils 10 Tabellenwerte. Mit einem weiteren Utility SATZINFO wird Ihnen ständig angezeigt, wo innerhalb der Tabelle Sie sich befinden.

```
200 rem ******************************
210 rem *       blaettern-demo         *
220 rem ******************************
230 :
240 rem ** pseudo-tabelle erzeugen **
250 dim x(100)
260 for i=1 to 100
270 : x(i)=rnd(0)
280 next i
290 :
300 rem ** tabelle durchblaettern **
310 nr%=1:rem start mit wert nr.1
315 :
317 bl$=chr$(134)+chr$(138)+chr$(133)+chr$(137)+chr$(135)+chr$(139)
318 :
320 print chr$(147):rem clear screen
330 for i=nr% to nr%+9
340 : print x(i)
350 next i
360 :
370 sys si,12,0,nr%,100
380 :
390 get a$:if a$="" then goto 390
400 i%=1:sys wahl,bl$,asc(a$),i%
410 if i%<>0 then sys blaettern,90,i%,nr%
440 goto 320
ready.
```

Listing 3.1: *Hauptprogramm von BLAETTERN*

NR% gibt an die Nummer des ersten der momentan angezeigten Werte an. *BL$* enthält die sechs »Blättertasten« F1 – F6.

Nach der Ausgabe der zehn ersten Werte wartet das Programm, bis Sie eine der Funktionstasten drücken. Das Utility SEARCH-COMAND prüft, ob die gedrückte Taste eine der in *BL$* enthaltenen Funktionstasten ist. Wenn ja, übergibt SEARCHCOMAND in *I%* die Nummer der Taste innerhalb von *BL$*. Sonst erhält *I%* den Wert 0.

BLAETTERN erhöht oder vermindert nun die aktuelle Wertnummer *NR%*, ausgehend vom Kommando *I%* (das der Position der gedrückten Funktionstaste im Kommandostring *BL$* entspricht). Als Maximalwert für *NR%* wird 90 angegeben, da bei NR%=90 aufgrund der Ausgabe von jeweils 10 Werten die letzten Werte mit den Nummern 90 bis 100 erscheinen.

3.2 FastGarbage

Schnelle Garbage Collection

Adresse 38144

Variablenname GC

Syntax SYS GC

Parameter keine

Funktion FASTGARBAGE »entschärft« den Alptraum jedes Basic-Programmierers – die Garbage Collection! Der Aufruf mit SYS GC führt eine superschnelle Garbage Collection durch (unter einer Sekunde, eher eine Zehntelsekunde). Dieser Aufruf sollte sich überall dort in Ihrem Programm befinden, wo viel »Stringmüll« erzeugt wird und Sie befürchten müssen, daß die (unglaublich langsame) serienmäßige Garbage Collection erbarmungslos zuschlägt.

Falls Sie nicht wissen, was eine Garbage Collection ist, so geben Sie einfach folgendes Programm ein:

```
235 DIM A$(1000)
240 FOR I=1 TO 1000
242 : PRINT I
250 : FOR J=1 TO 30
260 :   A$(I)=A$(I)+"X"
270 : NEXT J
280 NEXT I
```

Die DIM-Anweisung in Zeile 235 reserviert Platz für ein umfangreiches Stringarray. An jedes Element dieses Arrays wird in einer inneren Schleife 30mal das Zeichen »X« angehängt. Bei jedem neuen Durchgang der äußeren Schleife wird die Nummer des aktuellen Durchgangs auf dem Bildschirm ausgegeben (Zeile 242).

Durch die Zuweisungen in der inneren Schleife wird eine Unmenge an »Stringmüll« produziert. Resultat: In immer kürzeren Abständen stoppt die Bildschirmausgabe – und zwar für immer größere Zeiträume. An diesen Unterbrechungen ist die Garbage Collection schuld. Immer dann, wenn für einen anzulegenden String kein Platz vorhanden ist, tritt die Garbage Collection des Basic-Interpreters in Aktion, die »Müllabfuhr«, die alle nicht mehr benötigten Zeichenketten aus dem Speicher entfernt.

Und diese Müllabfuhr ist extrem langsam. Wenn Sie das Programm lange genug laufen lassen, so daß immer Strings angelegt werden, wird die Garbage Collection möglicherweise irgendwann einmal Stunden (!) benötigen.

Die für die Garbage Collection benötigte Zeit wächst etwa quadratisch mit der Anzahl der vorhandenen Strings!

Um diese ins Unglaubliche anwachsenden Wartezeiten, während denen Ihr Programm »blockiert« ist, zu vermeiden, verwenden Sie FASTGARBAGE. In allen kritischen Programmteilen, die sehr häufig (vor allem in Schleifen!) Stringzuweisungen enthalten, fügen Sie den Aufruf SYS GC ein.

FASTGARBAGE wird aufgerufen und beseitigt den momentan vorhandenen Stringmüll in weniger als einer Sekunde. Wird dieser Aufruf in allen kritischen Programmteilen verwendet, kommt es niemals zu einer »normalen« Garbage Collection.

```
200 rem ******************************
210 rem *       hauptprogramm          *
220 rem ******************************
230 :
235 dim a$(8000)
240 for i=1 to 1000
242 : print i
244 : sys gc
250 : for j=1 to 30
260 :   a$(i)=a$(i)+"x"
270 : next j
280 next i
ready.
```

Listing 3.2: *Hauptprogramm von FASTGARBAGE*

Dieses Demoprogramm entspricht fast exakt jenem, mit dem Sie die normale Garbage Collection provozieren sollten. Mit einem entscheidenden Unterschied: In Zeile 244 wird FASTGARBAGE aufgerufen, bevor die innere Schleife beginnt und wieder 30 Strings angelegt werden. Das heißt, vor diesen 30 Zuweisungen wird aufgeräumt, um dafür zu sorgen, daß im Speicher mit Sicherheit ausreichend Platz für diese 30 Zeichenketten vorhanden ist. Daher wird während des Ablaufs der inneren Schleife im Gegensatz zur

vorigen Version der Basic-Interpreter keine Veranlassung haben, seine eigene Garbage Collection zu benutzen.

Resultat: Das Programm läuft insgesamt einen »Hauch« langsamer ab als zuvor, da bei jedem neuen Durchgang der äußeren Schleife für circa eine Zehntelsekunde FASTGARBAGE in Aktion tritt. Aber dafür wird niemals eine echte »Blockade« erfolgen.

Hinweis Es ist zweckmäßig, sich erst nach Fertigstellung des eigentlichen Basic-Programms um die Garbage Collection zu kümmern. Wenn Ihr Programm fertig ist, fügen Sie vor kritischen Teilen den Aufruf von FASTGARBAGE ein, vor allem vor Schleifen, in denen wiederholt Zeichenketten angelegt werden. Wenn Ihr Programm mit einem Hauptmenü arbeitet, ist es sinnvoll, auf alle Fälle vor dem Verzweigen zu den einzelnen Unterprogrammen (ON ... GOSUB ..., ..., ...) den Aufruf von FASTGARBAGE einzufügen (SYS GC : ON ... GOSUB ..., ..., ...).

Sie können auch empirisch vorgehen: Tritt beim Testen des fertigen Programms in einem bestimmten Programmteil ab und zu eine »Blockade« auf, wissen Sie, daß dort die normale Garbage Collection zugeschlagen hat. Sie fügen einfach den Aufruf SYS GC an der betreffenden Stelle ein.

Hinweis »FastGarbage« kann Stringarrays mit einer Gesamtgröße von bis zu 12 KB perfekt bearbeiten. Bei noch größeren Stringmengen (kommt in der Praxis eigentlich nie vor) funktioniert FastGarbage meist (!) ebenfalls einwandfrei. Sollte Ihr Programm unglücklicherweise mehr als 12 KB Strings in einer solchen Art und Weise anlegen, daß FastGarbage »ins Schleudern« kommt, verzichten Sie einfach auf den Aufruf der Routine.

3.3 SearchComand

Benutzerkommando überprüfen

Adresse 39894

Variablenname WAHL

Syntax SYS WAHL,BEFEHLE$,CODE,NR%

Parameter

BEFEHLE$:	String mit den »Kommandotasten«
CODE:	Zeichencode der zu prüfenden Taste
NR%:	Zeichennummer, ab der *BEFEHLE$* durchsucht werden soll

Funktion SEARCHCOMAND prüft, ob ab einer angegebenen Position *NR%* das Zeichen mit dem Code *CODE* im String *BEFEHLE$* enthalten ist. Wenn ja, wird in *NR%* die Position übergeben, an der der Zeichencode enthalten ist. Wenn nein, enthält *NR%* nach der Rückkehr den Wert 0.

SEARCHCOMAND eignet sich vor allem zur Kommandoauswahl. Alle Tasten, die ein bestimmtes Kommando einleiten sollen, werden in einem String gespeichert. Mit SEARCHCOMAND können Sie jederzeit prüfen, ob eine vom Benutzer gedrückte Taste eine der Kommandotasten ist (und welche dieser Tasten), um anschließend zum Beispiel zu einem entsprechenden Unterprogramm zu verzweigen.

Hinweis SEARCHCOMAND erwartet als Parameter *CODE* nicht die gedrückte Taste selbst, sondern deren ASCII-Code (ermittelt die ASC-Funktion)!

Beim Aufruf von SEARCHCOMAND geben Sie mit dem Parameter *V/NR%* an, ab welchem Zeichen der String *BEFEHLE$* durchsucht werden soll. Angenommen, *BEFEHLE$* enhält 10 Kommandotasten und zu einem bestimmten Zeitpunkt darf der Benutzer nur eines der drei letzten Kommandos eingeben. Dann weisen Sie vor dem Aufruf dem Parameter *NR%* den Wert 7 zu, damit der Vergleich ab dem siebten Zeichen in *BEFEHLE$* beginnt. Im Normalfall wird *NR%* mit 1 initialisiert (Vergleich ab dem ersten Zeichen von *BEFEHLE$*) .

Beispiel Nehmen wir an, Ihr Programm kennt die Befehlstasten »L« (wie LOAD), »S« (SAVE), »D« (DIRECTORY), und die Kommandos »X«, »Y« und »Z«. Zuerst speichern Sie diese Zeichen in einem Kommandostring.

```
230 K$="LSDXYZ"
```

Wird später der Benutzer nach einem Kommando gefragt, vergleichen Sie mit SEARCHCOMAND die gedrückte Taste mit den in *K$* enthaltenen Zeichen.

```
240 GET A$:IF A$="" THEN 240
250 I%=1:SYS WAHL,K$,ASC(A$),I%
```

I% erhält vor dem Aufruf den Wert 1, da *K$* ab dem ersten Zeichen durchsucht werden soll. Drückte der Benutzer eine in *K$* enthaltene Taste, gibt *I%* nach der Rückkehr die Zeichenposition im Befehlsstring an (0 = *A$* ist nicht enthalten). Diese Position kann nun als Index zum Verzweigen zu den sechs verschiedenen Unterprogrammen verwendet werden (entsprechend den sechs in *K$* enthaltenen Kommandotasten).

```
260 ON I% GOSUB 500,600,700,800,900,1000
```

3.4 SearchString

Stringarray durchsuchen

Adresse 39897

Variablenname SEARCH

Syntax `SYS SEARCH,SUCHKRIT$,ARRAY$(START),ARRAY$ (ENDE), NR%`

Parameter

SUCHKRIT$:	String, der das Suchkriterium enthält (Maier, M??er, M* etc.)
ARRAY$: (START)	Erstes Element des zu durchsuchenden Stringarrays
ARRAY$: (ENDE)	Letztes Element des zu durchsuchenden Stringarrays
NR%:	Integervariable, in der das Suchergebnis mitgeteilt wird (0 = nicht gefunden, sonst Index des gefundenen Elements)

Funktion SEARCHSTRING durchsucht den angegebenen Bereich eines Stringarrays nach einer Zeichenkette, dem Suchkriterium. Nach dem Motto lieber zuviel als zuwenig finden, wird nicht zwischen Groß-/Kleinschreibung unterschieden. Das heißt, mit dem Suchkriterium »Maier« finden Sie auch einen versehentlich klein geschriebenen »maier«.

Das von der Floppy bekannte »Abkürzen« und »Maskieren« mit den Sonderzeichen »*« und »?« ist auch bei SEARCHSTRING möglich. Beispiele für den Einsatz dieser Suchmethoden:

Beispiel
```
s$="Maier":sys search,a$(1),a$(100),nr%
```

vergleicht die Elemente *A$(1)* bis *A$(100* mit der Zeichenkette »Maier«. *NR%* enthält anschließend den Index des ersten Strings, der »Maier« enthält oder 0 (»Maier« ist in keinem der 100 Strings enthalten).

Beispiel
```
s$="M*":sys search,a$(30),a$(100),nr%
```

vergleicht die Elemente *A$(30)* bis *A$(100* mit der Zeichenkette »M*«. Für einen positiven Vergleich genügt es, wenn das erste Zeichen eines der 70 Strings ein »M« ist, egal welche Zeichen folgen.

Beispiel

```
s$="M??er":sys search,a$(1),a$(100),nr%
```

vergleicht die Elemente *A$(1)* bis *A$(100)* mit der Zeichenkette »M??er«. SEARCHSTRING vergleicht nur das erste, vierte und fünfte Zeichen. Die Zeichen 2 und 3 sind »ausmaskiert«, das heißt, ihr Inhalt ist für das Resultat des Vergleichs uninteressant.

Beispiel

```
s$="M??e*":sys search,a$(1),a$(100),nr%
```

vergleicht die Elemente *A$(1)* bis *A$(100)* mit der Zeichenkette »M??e*«. Die Zeichen 2 und 3 sind »ausmaskiert«, das heißt, ihr Inhalt ist für das Resultat des Vergleichs uninteressant. Ebenso uninteressant ist, welche Zeichen die Strings ab der Position 5 enthalten (Kombination von Maskieren und Abkürzen).

Beispiel

```
s$="M??e*":sys search,a$(1),a$(100),nr%
```

vergleicht die Elemente *A$(1)* bis *A$(100)* mit der Zeichenkette »M??e*«. Die Zeichen 2 und 3 sind »ausmaskiert«, das heißt, ihr Inhalt ist für das Resultat des Vergleichs uninteressant. Ebenso uninteressant ist, welche Zeichen die Strings ab der Position 5 enthalten (Kombination von Maskieren und Abkürzen).

Beispiel Da SEARCHSTRING ein beliebiger Suchbereich angegeben wird, kann ein Array bis zum Ende durchsucht und alle gefundenen Strings ausgegeben werden.

```
230 s$="M*"
235 nr%=1:rem beginn der suche
240 sys search,a$(nr%),a$(1000),nr%
250 if nr%<>0 then print a$(nr%):nr%=nr%+1:goto
240
```

SEARCHSTRING sucht ab *A$(NR%)*, also ab *A$(1)* (siehe Zeile 240), nach dem ersten String, der mit einem »M« beginnt. Wurde ein solcher String gefunden, enthält *NR%* den Index des Arrayelements. Der zugehörige String wird ausgegeben, *NR%* um 1 erhöht und die Suche ab dem folgenden String fortgesetzt.

Daher gibt diese Routine nicht nur den ersten, sondern alle Strings im Array *A$(..)* aus, die mit einem »M« beginnen!

Beispiel STRINGSEARCH

```
200 rem ******************************
210 rem *      searchstring            *
220 rem ******************************
225 :
230 a=1000
240 dim a$(a)
250 for i=1 to a-1
260 : a$(i)="Maier"
270 next i
280 a$(a-1)="Mayer":a$(a)="Meier"
290 print "es sind";a-1;"'Maier', ein 'Mayer' u.ein 'Meier' vorhaden"
300 input "suchkriterium";s$
310 :
320 x=ti
330 sys search,s$,a$(1),a$(a),p%
340 t=(ti-x)/60
350 print "gefunden> element nummer";p%
360 print "suchzeit in sec>";t:print
370 goto 280
ready.
```

Listing 3.3: *Hauptprogramm von SEARCH*

Dieses Programm erzeugt ein Stringarray mit 998 »Maier« (*A$(1)* bis *A$(998)*, einem »Mayer« (*A$(999)*) und einem »Meier« (*A$(1000)*).

Sie werden aufgefordert, ein Suchkriterium einzugeben. SEARCH-STRING vergleicht jeden String des Arrays mit Ihrer Eingabe und meldet Ihnen den Index eines gefundenen Elements (oder 0). Außerdem wird Ihnen die Suchdauer in Sekunden angezeigt, die selbst beim Durchsuchen des kompletten Arrays (1000 Strings!) mit komplexen Methoden wie beim Suchkriterium »Me?er« weit unter einer Sekunde bleibt.

3.5 QuickSort

Sortierrroutine

Adresse `52240`

Variablenname `QUICK`

Syntax 1 `SYS QUICK, CODE, NR, ARRAY1(START), ARRAY2(ENDE)`

Syntax 2 `SYS QUICK,CODE,NR,ARRAY1(START),ARRAY2(ENDE), ARRAY2(START), ARRAY2(ENDE)`

Parameter

CODE:	Code des Trennzeichens
NR:	Nummer des Sortier-Teilfeldes
ARRAY1: (START)	Erstes Element des zu sortierenden Arrays
ARRAY1: (ENDE)	Letztes Element des zu sortierenden Arrays
ARRAY2: (START)	Erstes Element des mitzusortierenden Arrays
ARRAY2: (ENDE)	Letztes Element des mitzusortierenden Arrays

Funktion QUICKSORT sortiert ein Array beliebigen Typs (String, Integer, Real) innerhalb der angegebenen Grenzen. Es kann – muß jedoch nicht – ein zweites Array ebenfalls beliebigen Typs angegeben werden, das »mitsortiert« wird. In diesem Fall ist Syntax2 gültig, das heißt, zwei zusätzliche Parameter sind anzugeben.

Bitte vergessen Sie vorläufig die Parameter *CODE* und *NR*. Diese Parameter benötigen Sie nur bei sehr komplexen Sortierproblemen. Im Normalfall geben Sie beiden Parametern den Wert 0. Im folgenden bespreche ich der Reihe nach die verschiedenen Einsatzmöglichkeiten von QUICKSORT.

Einfaches Sortieren eines Arrays

Für die Parameter *CODE* und *NR* geben Sie den Wert 0 an. *ARRAY1(START)* und *ARRAY1(ENDE)* ist das erste beziehungsweise letzte Element des zu sortierenden Arrays.

Beispiel `SYS QUICK, 0, 0, A$(1), A$(1000)`

Sortieren des Stringarrays *A$(1)* bis *A$(1000)*

Beispiel `SYS QUICK, 0, 0, X%(1), X%(1000)`

Sortieren des Integerarrays *X%(1)* bis *X%(1000)*

Beispiel `SYS QUICK, 0, 0, W(1), W(1000)`

Sortieren des Realzahlenarrays *W(1)* bis *W(1000)*

Mitsortieren eines zweiten Arrays

»Mitsortieren« heißt, daß zwei Arrays existieren, zum Beispiel das Stringarray *A$(..)* und das Integerarray *X%(..)* und zwischen beiden Arrays Verbindungen bestehen. Zum Beispiel könnte *A$(..)* Adressen enthalten und *X%(..)* die zugehörigen Recordnummern dieser Adressen in einer relativen Datei.

Oder ein Array *V$(..)* könnte Vokabeln enthalten. Während des Vokabeltests wird festgehalten, wie oft jede Vokabel bereits richtig übersetzt wurde. Diese Häufigkeit richtiger Übersetzungen einer Vokabel wird im zugehörigen Array *H$(..)* gespeichert. *H$(1)* enthält die Häufigkeit richtiger Übersetzungen der in *V$(1)* gespeicherten Vokabel, *H$(2)* die Häufigkeit für die Vokabel in *V$(2)* und so weiter.

Sowohl bei den Adressen als auch den Vokabeln darf nicht nur ein einfaches Sortieren des Stringarrays stattfinden. Anschließend wären die Beziehungen zwischen den beiden Arrays zerstört. Im Fall der Vokabeln muß beim Sortieren von *V$(..)* das »abhängige« Array *H$(..)* »mitsortiert« werden. Bei jedem Vertauschen zweier Strings des Vokabelarrays müssen auch die beiden zugehörigen Variablen des Häufigkeitsarrays miteinander vertauscht werden. Genau das leistet die Option »Array mitsortieren« von QUICKSORT.

Beispiel `SYS QUICK, 0, 0, A$(1), A$(1000), X%(1), X%(1000)`

Sortieren des Stringarrays *A$(1)* bis *A$(1000)* bei gleichzeitigem Mitsortieren des Integerarrays *X%(1)* bis *X%(1000)*.

Beispiel `SYS QUICK, 0, 0, X(1), X(1000), Q%(1), Q%(1000)`

Sortieren des Realzahlenarrays *X(1)* bis *X(1000)* bei gleichzeitigem Mitsortieren des Integerarrays *Q%(1)* bis *Q%(1000)*.

```
200 rem ******************************
210 rem *       hauptprogramm          *
220 rem ******************************
230 :
240 a=9:dima$(a),p%(a)
250 print"bitte warten. sortstrings werden erzeugt"
260 fori=0toa:a$(i)="":forii=1to20:a$(i)=a$(i)+chr$(int(rnd(ti)*25)+65):nextii
270 p%(i)=i:nexti:printchr$(147)
280 fori=0toa:printa$(i),p%(i):next
282 z1=ti
285 :
290 sys quick,0,0,a$(0),a$(a),p%(0),p%(a)
292 :
294 z2=ti
300 print:fori=0toa:printa$(i),p%(i):next
310 print(z2-z1)/60"sec"
320 print "taste druecken"
330 get a$:if a$="" then 330
340 goto 250
ready.
```

Listing 3.4: *Hauptprogramm von QUICK1*

Dieses Demoprogramm erzeugt zwei Arrays mit jeweils zehn Elementen. *A$(0)* bis *A$(9)* enthält zufällig erzeugte Zeichenketten, *P%(0)* bis *P%(9)* zugehörige – ebenfalls zufällig erzeugte – Integerzahlen. Diese Integerzahlen stehen stellvertretend für die erwähnten Recordnummern oder Häufigkeiten.

Die noch unsortieren Strings und zugehörigen Zahlen werden auf dem Bildschirm ausgegeben (Zeile 280). Danach wird das Array *A$(..)* sortiert und das Array *P%(..)* mitsortiert, jeweils im Indexbereich 0 bis *A* (*A* = höchster Index):

```
290 SYS QUICK, 0, 0, A$(0), A$(A), P%(0), P%(A)
```

Auf dem Bildschirm erscheint das sortierte und das mitsortierte Array und die für den Sortierlauf benötigte Zeit.

Wenn Sie eine beliebige Taste drücken, wiederholt sich der gesamte Vorgang.

Der in Zeile 240 an *A* zugewiesene Wert bestimmt die Größe der beiden Arrays. Wenn Sie die Geschwindigkeit beim Sortieren größerer Arrays testen wollen, ändern Sie einfach diese Zeile. Weisen Sie *A* statt 9 den Wert 100 oder gar 1000 zu. Allerdings müssen Sie nun sehr lange warten, bis das zufällige Erzeugen der Arrayinhalte beendet ist.

Dennoch werden Sie feststellen, daß QUICKSORT selbst zum Sortieren und Mitsortieren eines jeweils 1000 Elemente großen Arrays weniger als eine Sekunde benötigt!

```
200 rem ******************************
210 rem *        hauptprogramm         *
220 rem ******************************
230 :
240 a=9:dima(a),x$(a)
250 fori=0toa:a(i)=rnd(0)*10000:x$(i)=mid$(str$(i),2)+".testsstring":next
260 fori=0toa:printa(i),x$(i):next
262 z1=ti
264 :
270 sys quick,0,0,a(0),a(a),x$(0),x$(a)
272 :
274 z2=ti
280 print:print:fori=0toa:printa(i),x$(i):next
290 print(z2-z1)/60"sec"
300 print"taste druecken"
310 get a$:if a$="" then 310
320 goto 250
ready.
```

Listing 3.5: *Hauptprogramm von QUICK2*

Dieses Demoprogramm sortiert ein Realzahlenarray mit 10 Elementen *A(0)* bis *A$(9)*, das Zufallszahlen enthält. Ein Stringarray *X$(1)* bis *X$(9)* wird mitsortiert. Es enthält die Zeichenketten:

```
X$(0) = "0.ter Teststring"
X$(1) = "1.ter Teststring"
.....
.....
X$(9) = "9.ter Teststring"
```

Beide Arrays werden im Indexbereich 0 bis *A* (*A* = höchster Index) sortiert:

```
270 SYS QUICK, 0, 0, A(0), A(A), X$(0), X$(A)
```

Als Ausgabe erscheint nach dem Sortierlauf außer den neuen Arrayinhalten zusätzlich die für das Sortieren benötigte Zeit in Sekunden.

Dieses Programm eignet sich sehr gut, die Geschwindigkeit von QUICKSORT zu testen. Ersetzen Sie A=9 in Zeile 240 zum Beispiel durch A=999, um Arrays mit je 1000 Elementen zu erzeugen (*A(0)* bis *A(999)* und *X$(0)* bis *X$(999)*).

Da das Erzeugen von Zufallszahlen weit weniger aufwendig ist als das Erzeugen von Strings mit zufälligen Inhalten, müssen Sie nur etwa 20 bis 30 Sekunden warten, bis das unsortierte Array erscheint.

Und nun wird es interessant. Nach der unsortierten Ausgabe der zweimal 1000 Arrayelemente Ausgabe »zögert« das Programm nur einen kurzen Moment (circa 5 Sek), bevor das Ergebnis des Sortierlaufs erscheint. Das heißt, zum Sortieren von 1000 Elementen (und Mitsortieren weiterer 1000 Elemente!) wurden knapp 5 Sekunden benötigt!

Diese 5 Sekunden zum Sortieren von 1000 Elementen sind bereits außergewöhnlich lang! QUICKSORT ist beim Sortieren von Realzahlen weit langsamer als beim Sortieren von Integers oder Strings.

Sie dürfen davon ausgehen, daß das Sortieren von 1000 Strings etwa 0,5 und von 1000 Integerzahlen etwa 2 Sekunden dauert.

»Teilfeld-Sortieren« von Strings

Kommen wir zur komplexesten Anwendung. QUICKSORT ist nicht nur schnell, sondern so flexibel, daß Sie mit dieser Routine jedes (!) Sortierproblem lösen können. Sogar das »Teilfeld-Sortieren« von Strings.

Strings enthalten manchmal nicht nur eine, sondern mehrere Informationen. So ist es möglich, in einem String komplette Adressen zu speichern, die aus mehreren Teilen bestehen.

```
A$(0) = "Baloui/Said/6700/Breite Str.100a"
A$(1) = "Krause/Maria/3000/Breite Str.100a"
A$(2) = "Beckenbach/Andi/6000/Aalweg 2"
.....
.....
```

In diesem Beispiel enthält jeder String des Arrays *A$(..)* eine komplette Adresse, die aus den Teilen »Name«, »Vorname«, »Plz« und »Strasse« besteht.

Die einzelnen »Teilfelder« werden durch ein Sonderzeichen voneinander unterschieden – im Beispiel durch »/«.

QUICKSORT ist in der Lage, Strings aus mehreren Feldern nach jedem beliebigen Teilfeld zu sortieren. Voraussetzung ist, daß die einzelnen Felder durch ein immer gleiches Sonderzeichen voneinander zu unterscheiden sind. Das muß keineswegs das Zeichen »/« sein. Sie können sich das Trennzeichen frei aussuchen, zum Beispiel einen Stern » * «:

```
A$(0) = "Baloui*Said*6700*Breite Str.100a"
A$(1) = "Krause*Maria*3000*Breite Str.100a"
A$(2) = "Beckenbach*Andi*6000*Aalweg 2"
.....
.....
```

Allgemein ausgedrückt: Sie verwenden als Trennzeichen irgendein Sonderzeichen, das mit Sicherheit nicht in den Feldinformationen selbst vorkommt (zum Beispiel auch ein Grafikzeichen).

Beim Aufruf geben Sie QUICKSORT an, welches Trennzeichen Sie verwenden. Und zwar geben Sie den ASCII-Code an! Das heißt, bei einem Stern geben Sie als Parameter *CODE* den Wert 42 an, bei Verwendung des Zeichens »/« den Wert 47.

Außerdem müssen Sie die Nummer des Feldes angeben, das QUICKSORT als »Sortierkriterium« verwenden soll. Diese Angabe entspricht dem Parameter *NR*.

Beispiel A$(0) = "Baloui*Said*6700*Breite Str.100a"

A$(1) = "Krause*Maria*3000*Breite Str.100a"

A$(2) = "Beckenbach*Andi*6000*Aalweg 2"

Um diese drei Strings (Felder mit »*« (Code 42) getrennt) nach dem ersten Feld »Name« zu sortieren, rufen Sie QUICKSORT so auf:

```
SYS QUICK, 42, 1, A$(0), A$(2)
```

Resultat A$(0) = "Baloui*Said*6700*Breite Str.100a"

A$(1) = "Beckenbach*Andi*6000*Aalweg 2"

A$(2) = "Krause*Maria*3000*Breite Str.100a"

Mit dem Aufruf

```
SYS QUICK, 42, 3, A$(0), A$(2)
```

werden die drei Strings nicht nach dem ersten, sondern nach dem dritten Feld »PLZ« (3000, 6000 und 6700) sortiert:

Resultat A$(0) = "Krause*Maria*3000*Breite Str.100a"

A$(1) = "Beckenbach*Andi*6000*Aalweg 2"

A$(2) = "Baloui*Said*6700*Breite Str.100a"

Beispiel Selbstverständlich können Sie auch weiterhin ein zweites Array mitsortieren lassen. Der Aufruf:

```
SYS QUICK, 42, 3, A$(0), A$(1000), P%(0),
P%(1000)
```

sortiert 1000 Elemente des Arrays *A$(..)* nach dem dritten Feld der durch das Zeichen mit dem Code 42 getrennten Felder. Bei jedem Vertauschen zweier Strings werden zusätzlich im Array *P%(..)* die Variablen mit den gleichen Indizes vertauscht – *P%(..)* wird mitsortiert.

```
200 rem ******************************
210 rem *       hauptprogramm          *
220 rem ******************************
230 :
240 a=9:dima$(a):rem a=letztes element
250 print"Stringaufbau>  Name^Vorname^Strasse^Ort"
260 input"sortfeld (nr)";sf:print chr$(147);
270 a$(0)="Baloui^Said^6700^Breite Str.100a"
280 a$(1)="Krause^Maria^3000^Breite Str.100a"
290 a$(2)="Beckenbach^Andi^6000^Aalweg 2"
300 a$(3)="Amendt^Otto^1000^Wienerwaldstr.1"
310 a$(4)="Maier^Gerd^2000^Irisweg5"
320 a$(5)="Demel^Werner^7000^Efeuweg5"
330 a$(6)="Willi^Otto^5500^Gartenstr.5"
340 a$(7)="Mueller^Gerhard^8000^Steinweg 7"
350 a$(8)="Wagner^Frieda^1500^Stussenstr.11a"
360 a$(9)="Gans^Gustav^3000^Gaenterichweg 2"
370 if a>9 then for i=10 to a:a$(i)="string^ist^leer":next
380 :
390 rem *** originalreihenfolge ***
400 fori=0toa:printa$(i):next
410 z1=ti:rem zeit stoppen
420 :
430 rem *** aufruf quicksort ***
440 sys quick,94,sf,a$(0),a$(a)
450 :
460 rem *** sortierte reihenfolge ***
470 z2=ti:rem zeit stoppen
480 print:fori=0toa:printa$(i):next
490 print:print(z2-z1)/60"sec",
500 goto260
ready.
```

Listing 3.6: *Hauptprogramm von QUICK3*

Dieses Programm erzeugt 10 Strings *A$(0)* bis *A$(9)*. Jeder String enthält eine vollständige Adresse, deren vier Felder »Name«, »Vorname«, »PLZ« und »Ort« mit dem Zeichen »Pfeil nach oben« (neben der RESTORE-Taste, ASCII-Code 92) voneinander getrennt sind.

Es fordert Sie auf, die Nummer des Sortierfeldes (1–4) einzugeben. Sie sehen die unsortierten Strings, der Sortierlauf findet statt, anschließend wird Ihnen das sortierte Array gezeigt und die benötigte Zeit mitgeteilt.

Sortierzeiten:

1000 Strings:	0,5 sec
1000 Integers:	2 sec
1000 Reals:	4 sec

Die Sortierzeiten werden durch das Mitsortieren eines zweiten Arrays praktisch nicht verlängert!

Sortierreihenfolge: Die Sortierreihenfolge entspricht den ASCII-Codes im Commodore-Handbuch, abgesehen von den Umlauten (kleine Umlaute werden vor allen anderen Kleinbuchstaben, große Umlaute vor allen anderen Großbuchstaben einsortiert).

Es ist zu beachten, daß dieses Sortierprinzip zwar sehr zweckmäßig ist, aber nicht der »Duden«-Norm in allen Punkten entspricht. (siehe Umlaute).

4

Bildschirm/Tastatur

Die folgenden Utilities dienen zur komfortableren (und schnelleren Behandlung) von Bildschirm und Tastatur. Sie zeichnen sich durch folgende Fähigkeiten aus:

- deutscher Zeichensatz (CHARACTERSINIT)
- beschleunigte Bildschirmausgabe (FASTSCREENINIT)
- Eingaberoutine für Einzeleingaben, Eingabemasken und Bildschirmeditoren (INPUT)
- Tastatur-Makros (MAKROSINIT)
- Hilfsanzeige beim Blättern in großen Datenmengen (SATZINFO)
- Direktes Setzen des Cursors (SETCURSOR)
- Schnelle und komfortable Maskenausgabe (STROUT)

Einige dieser Utilities sind nach dem einmaligen Aufruf ständig im Hintergrund aktiv (CHARACTERSINIT, FASTSCREENINIT und MAKROSINIT). Diesen einmaligen Aufruf übernimmt das Initialisierungsprogramm TOOLS.INIT.

4.1 CharactersInit

Deutscher Zeichensatz

Adresse 53039

Variablenname CHAR

Syntax SYS CHAR

Parameter keine

Funktion Richtet durch Änderung der Tastaturbelegung einen deutschen Zeichensatz mit Umlauten ein ("öäüßÖÄÜ"). Dabei wird nur der Groß-/Kleinschrift-Modus berücksichtigt, nicht der Groß-/Grafik-Modus. Das heißt, um eine korrekte Bildschirmdarstellung zu erhalten, muß dieser Modus mit PRINT CHR$(14) eingeschaltet und mit PRINT CHR$(8) »verriegelt« werden!

Beispiel
```
SYS CHAR: PRINT CHR$(14) CHR$(8)
```

Neue Tastaturbelegung:

Code	Alte Belegung	Neue Belegung
58	:	ö
59	;	ä
64	@	ü
92	£	ß
91	SHIFT + :	Ö
93	SHIFT + ;	Ä
186	SHIFT + @	Ü
62	SHIFT + .	:
60	SHIFT + ,	;

Zeichencodierung und Ausdruck: Die Kenntnis der Codierung ist für den Druck der Umlaute wichtig. Mit den Drucker-Utilities können Sie bestimmen, welchen Code die Utilities für ein bestimmtes Zeichen an den Drucker senden. Laut Tabelle besitzt »ö« den Code 58. ASCII-Drucker erwarten für ein »ö« jedoch meist den Code 215. Sie geben mit den Drucker-Utilities daher an, daß beim Ausdruck der Code 58 durch den Code 215 ersetzt

werden soll, jenen Code, der Ihren Drucker zum Druck des Zeichens »ö« veranlaßt.

Bildschirmspeicher: Die letzten 2 Kbyte Inhalt des Charakter-ROM's werden ins RAM nach $F800-$FFFF kopiert und manipuliert. Zusätzlich wird die VIC-Basisadresse geändert (nun $C000). Die Startadresse des Bildschirmspeichers ist nicht mehr wie gewohnt $0400 (dezimal 1024), sondern $C000 (dezimal 49152). Befehle, die direkt auf den Bildschirmspeicher zugreifen, sind entsprechend zu ändern.

Beispiel

```
Alt:  FOR I=Ø TO 999        Neu:  FOR I=Ø TO 999
      POKE 1Ø24+I,1               POKE 49152+I,1
      NEXT I                      NEXT I
```

4.2 FastScreenInit

Beschleunigte Bildschirmausgabe

Adresse 39891

Variablenname FSCREEN

Syntax SYS FSCREEN

Parameter keine

Funktion Beschleunigt alle Bildschirmausgaben eines Basic-Programms mit der PRINT-Anweisung um die Hälfte (und Ausgaben in Assembler-Routinen mit BSOUT). Nach der Initialisierung mit SYS FSCREEN bleibt die beschleunigte Ausgabe bis zum einem Reset oder Ausschalten des Rechners erhalten.

Beispiel SYS FSCREEN

Funktionsweise Bei der Initialisierung wird der BSOUT-Vektor auf die FSCREEN-Routine »verbogen«. Anschließend wird bei Ausgaben auf den Bildschirm ein Großteil der auszugebenden Zeichen nicht mehr an BSOUT weitergeleitet, sondern direkt in den Bildschirmspeicher geschrieben (unabhängig davon, wo sich das Video-RAM befindet).

4.3 Input

Eingaberoutine

Adresse 50176

Variablenname EINGABE

Syntax SYS EINGABE, SPALTE, ZEILE, LAENGE, VORGABE$, CHAR$, ENDE$, LAST$, NR%, FLAG%, EDITF%

Parameter Hin

SPALTE:	Startspalte der Eingabezone
ZEILE:	Startzeile der Eingabezone
LAENGE:	Länge der Eingabezone
VORGABE$:	Vorgegebener Inhalt der Eingabezone
CHAR$:	String mit allen zulässigen Eingabezeichen
ENDE$:	String mit allen Zeichen, die die Eingabe beenden
NR%:	Nummer des Zeichens, auf das der Cursor nach dem Aufruf gesetzt werden soll (0 oder 1-LAENGE). Normalerweise verwenden Sie den Wert 0 für normale einzelne Eingabe. Ein Wert ungleich 0 ist nur dann zu verwenden, wenn die Eingaberoutine zum Editieren des gesamten Bildschirms eingesetzt wird (Textverarbeitung etc.).
FLAG:	0 = Eingabezone wird normal dargestellt 1 = Eingabezone wird invers dargestellt 2 = Malmodus

Parameter Zurück

VORGABE$:	Enthält den Eingabestring
LAST$:	Enthält nach Rückkehr das Zeichen, mit dem die Eingabe beendet wurde
NR%:	Enthält die letzte Position des Cursors (1-LAENGE) in der Eingabezone (nur, wenn beim Aufruf ein Wert ungleich 0 angegeben wurde)
EDITF%:	0 = Vorgabe wurde nicht verändert <> 0 = Vorgabe wurde verändert

Funktion Eingabe in einer beim Aufruf anzugebenden Zone (maximal 255 Zeichen Länge). Cursorbewegungen sind auf die beim Aufruf durch *SPALTE*, *ZEILE* und *LAENGE* definierte Zone begrenzt. Alle Editiertasten funktionieren wie gewohnt (die Cursortasten, *DEL*, *INST*). Zusätzlich bestehen folgende Editiermöglichkeiten in Kombination mit der C=-Taste (Commodore-Taste):

C= + Cursor → → Sprung zum nächsten Wort

C= + Cursor ↑ → Sprung zum vorigen Wort

Beim Aufruf wird die Zone mit dem Inhalt des Strings *VORGABE$* gefüllt, der auch leer sein kann (keine Vorgabe). Als gültige Eingabezeichen werden nur jene Zeichen akzeptiert, die im String *CHAR$* enthalten sind. Ob die Eingabe mit ↵, einer anderen oder gar mehreren Tasten beendet wird, bestimmen Sie: Als »Endzeichen« gelten alle in *ENDE$* enthaltenen Zeichen. Das Zeichen, mit dem eine Eingabe beendet wurde (eines der Zeichen in *ENDE$*), speichert die Eingaberoutine vor der Rückkehr nach Basic im String *LAST$*.

Die Eingaberoutine kann für Einzeleingaben, für Eingabemasken oder zum Editieren des gesamten Bildschirms eingesetzt werden, zum Beispiel für die Erstellung einer Eingabemaske. Die jeweilige Anwendung bestimmt *NR%*.

Der Parameter *LAST$*: INPUT arbeitet fehlerhaft, wenn die Variable, die als Parameter *LAST$* angegeben wird, beim Aufruf noch gar nicht existiert! Welche Stringvariable Sie verwenden, ist zwar Ihnen überlassen; Sie müssen jedoch dafür sorgen, daß vor dem Aufruf bereits irgendeine (egal welche) Zuweisung an die betreffende Variable vorgenommen wurde, zum Beispiel mit *LAST$*="".

Einzeleingaben (NR% = 0)

Sie weisen *NR%* vor dem Aufruf den Wert 0 zu. Die Routine verhält sich wie beschrieben: Der Versuch, den Cursor aus der Zone herauszubewegen, wird ignoriert. *FLAG%* bestimmt den Darstellungsmodus: Weisen Sie *FLAG%* den Wert 0 zu, wird die Zone normal dargestellt, mit 1 erreichen Sie, daß die komplette Zone invers dargestellt wird.

```
200 rem ******************************
210 rem *     eingaberoutine demo1     *
220 rem ******************************
230 :
235 rem * eingabezeichen *
237 num$="1234567890.- "
240 alpha$="qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM"
250 alpha$=alpha$+":;$öäÜ:,./<>?=*^+-1234567890!#$%&'() "+chr$(34)
260 :
270 rem * endezeichen *
280 e$=chr$(13):rem return-taste
290 :
300 rem *** aufruf 1 ***
305 last$="":rem stringvar.initial.!!!
310 nr%=0
320 v$="Dies ist die Vorgabe!"
330 sys eingabe,5,3,30,v$,alpha$,e$,last$,nr%,1,e%
340 print:print v$
350 :
360 rem *** aufruf 2 ***
370 v$="":rem keine vorgabe
380 sys eingabe,10,15,10,v$,num$,e$,last$,nr%,0,e%
390 print:print v$
ready.
```

Listing 4.1: *Hauptprogramm von INPUT1*

Üblicherweise sind Eingaben meist rein numerischer (Ziffern und die Zeichen » . - " «) oder alphanumerischer Art (Buchstaben, Ziffern und Sonderzeichen)). In den Zeilen 237 und 240–250 werden die entsprechenden Eingabezeichen den Strings *NUM$* und *ALPHA$* zugewiesen.

Da Einzeleingaben im Normalfall mit ↵ beendet werden, wird in Zeile 280 ein Endzeichen-String definiert, der nur dieses eine Zeichen enthält (E$ = CHR$(13)), also den Code der ↵-Taste.

Nun folgen zwei Eingaben. Zuvor wird die Stringvariable *LAST$* mit LAST$="" initialisiert. Diese Initialisierung ist zwingend vorgeschrieben. Vor dem Aufruf von INPUT muß die als Parameter *LAST$* verwendete Variable bereits existieren!

Um zu kennzeichnen, daß es sich um zwei »normale« Einzeleingaben handelt, erhält *NR%* den Wert 0 (Zeile 310). Beim ersten Aufruf der Eingaberoutine wird die Eingabezone mit der Vorgabe »Dies ist die Vorgabe!« gefüllt.

Die Eingabezone beginnt ab Spalte 5 von Zeile 3 und ist 30 Zeichen lang. Die Vorgabe enthält *V$*, in *ALPHA$* sind die zulässigen

Eingabezeichen und in *E$* die Endezeichen (hier: nur ↵) enthalten. *LAST$* wird das Zeichen aufnehmen, mit dem Sie die Eingabe beenden (hier: ↵), *NR%* angeben, auf welchem Zeichen innerhalb der Zone sich der Cursor zuletzt befand.

Der Wert 1 für den Parameter *FLAG%* gibt der Routine an, daß die Eingabezone invers darzustellen ist.

Nach der Rückkehr aus der Eingaberoutine enthält *VORGABE$* den neuen Inhalt der Eingabezone – also Ihre Eingabe. Diese Eingabe zeigt Zeile 340 zur Kontrolle auf dem Bildschirm an, ebenso wie den Inhalt von *E%*.

E% wird entweder eine 0 oder einen beliebigen Wert ungleich 0 enthalten. 0 bedeutet, daß Sie die Vorgabe nicht verändert haben. Ein Wert ungleich 0 teilt dem Basic-Programm mit, daß der Inhalt der Eingabezone nicht mit der Vorgabe identisch ist.

Die Eingaberoutine wird zur Demonstration ein zweites Mal aufgerufen (Zeile 370–390). Diesmal ist der Vorgabestring *V$* leer, das heißt, es erscheint eine leere Eingabezone. Außerdem wird als String mit den zulässigen Eingabezeichen *NUM$* angegeben. Es wird Ihnen daher nicht mehr möglich sein, Buchstaben oder Sonderzeichen einzugeben. Da für den Parameter *FLAG%* eine 0 angegeben wird, erscheint die Eingabezone nicht wie zuvor invers, sondern wird normal dargestellt.

Eingabemasken (NR% = 0)

Eingabemasken kennen Sie von professionellen Programmen, zum Beispiel Dateiverwaltungen. Eine Eingabemaske besteht aus einzelnen Feldern. In jedem Feld können Sie eine Eingabe vornehmen. Mit den Tasten Cursor ↑ und Cursor ↓ gelangen Sie zum jeweils nächsten beziehungsweise vorhergehenden Feld. Mit ↵ oder vielleicht auch einer Funktionstaste wird die gesamte Eingabe beendet.

```
Name:.............  Vorname:............

Strasse:........................

PLZ:.... Ort:...................

Telefon:...............
```

Die Eingaberoutine können Sie problemlos zur Verwaltung derartiger Masken einsetzen. Allerdings bildet die Routine in diesem Fall nur den Kern eines größeren Programms. Die Eingaberoutine ist für eine einzelne Eingabe zuständig, die »Steuerung« übernimmt ein Basic-Programm.

Das heißt vor allem, daß für diesen Einsatzzweck mehrere Endetasten zu deklarieren sind. Zum Beispiel soll auch die Taste [Cursor ↓] die Eingabe in ein Feld beenden. Allerdings wird bei Betätigen von [Cursor ↓] nicht die gesamte Eingabe beendet. Stattdessen ruft das Basic-Programm die Eingaberoutine mit veränderten Werten erneut auf, da nun ([Cursor ↓]) eine Eingabe in das nächste Feld vorzunehmen ist. Einigen wir uns auf folgende »Steuertasten«:

[Cursor ↓]: Eingabe in das nächste Feld
[Cursor ↑]: Eingabe in das vorhergehende Feld
[↵]: Beenden der Gesamteingabe

Der Beginn des Hauptprogramms ändert sich nicht. Wir benötigen zwei Typen von Eingaben: numerische Eingaben (Feld »PLZ«) und alphanumerische Eingaben in allen anderen Feldern.

Außer [↵] sollen diesmal auch die Tasten [Cursor ↓] und [Cursor ↑] die Eingabe in ein Feld beenden (Zeile 280).

Jedes Feld besitzt eine eigene Eingabezone, einen bestimmten Typ und eine Vorgabe. Diese Feldbeschreibung speichert man üblicherweise in Arrays. Ab Zeile 300 werden die entsprechenden Zuweisungen vorgenommen.

Die Maske besteht aus insgesamt 6 Feldern. Diese Feldanzahl wird in *FA* gespeichert (Zeile 305), anschließend finden die benötigten Array-Dimensionierungen statt (Zeile 306).

Im Array *N$(..)* sollen die Feldnamen, in *V$(..)* die Vorgaben für die Eingabezonen, in *X%(..)* und *Y%(..)* die Position dieser Feldnamen und in *L%(..)* die Länge jeder Eingabezone gespeichert werden. *T$(..)* wird für jedes Feld einen String mit den Zeichen enthalten, die in der betreffenden Zone eingegeben werden dürfen.

Die folgenden Deklarationen erläutere ich stellvertretend am Beispiel des ersten Feldes:

```
310 n$(1)="Name:":v$(1)="":x%(1)=1:y%(1)=2:
l%(1);=12 :t$(1) =alpha$
```

```
200 rem ******************************
210 rem *     eingaberoutine demo2     *
220 rem ******************************
230 :
235 rem * eingabezeichen *
237 num$="1234567890.- "
240 alpha$="qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM"
250 alpha$=alpha$+":;$öäÜ:,./<>?=*^+-12345678901#$%&'() "+chr$(34)
260 :
270 rem * endezeichen *
280 e$=chr$(13)+chr$(17)+chr$(145):rem return/crs.down/crs.up
290 :
300 rem * feld-deklarationen *
305 fa=6:rem anzahl der felder
306 dim n$(fa),v$(fa),x%(fa),y%(fa),l%(fa),t$(fa)
307 :
310 n$(1)="Name>":v$(1)="":x%(1)=1:y%(1)=2:l%(1)=12:t$(1)=alpha$
320 n$(2)="Vorname>":v$(2)="":x%(2)=20:y%(2)=2:l%(2)=10:t$(2)=alpha$
330 n$(3)="Strasse>":v$(3)="":x%(3)=1:y%(3)=4:l%(3)=25:t$(3)=alpha$
340 n$(4)="Plz>":v$(4)="":x%(4)=1:y%(4)=6:l%(4)=4:t$(4)=num$
350 n$(5)="Ort>":v$(5)="":x%(5)=12:y%(5)=6:l%(5)=20:t$(5)=alpha$
360 n$(6)="Telefon>":v$(6)="":x%(6)=1:y%(6)=8:l%(6)=15:t$(6)=alpha$
370 :
380 rem * maske ausgeben *
385 leer$="                                   "
390 print chr$(147):rem clear screen
400 for i=1 to fa
410 : sys crs,x%(i),y%(i)
420 : print n$(i) chr$(18) left$(leer$,l%(i))
430 next i
440 :
450 rem *** maskensteuerung ***
455 last$="":rem initialis.noetig !!!
460 i=1:rem aktuelle feldnummer !!!
470 :
475 nr%=0:rem immer einzeleingabe !!!
480 sys eingabe,x%(i)+len(n$(i)),y%(i),l%(i),v$(i),t$(i),e$,last$,nr%,1,e%
490 :
500 if last$=chr$(17) and i<fa then i=i+1
510 if last$=chr$(145) and i>1 then i=i-1
515 if last$<>chr$(13) then goto 475
520 :
530 rem * kontrolle *
540 sys crs,0,15
550 for i=1 to fa
560 : print v$(i)
570 next i
ready.
```

Listing 4.2: *Hauptprogramm von INPUT2*

Das erste Feld erhält die Bezeichnung »Name:« (n$(1)="Name:"). Diese Bezeichnung soll ab Spalte 1 (x%(1)=1) von Zeile 2 (y%(1)=2) ausgegeben werden. Die Länge der zugehörigen Zone beträgt 12 Zeichen (l%(1)=12). Beim Aufruf wird kein Zoneninhalt vorgegeben, die Zone ist leer (v$(1)=""). Der Typ der Eingabezone ist alphanumerisch (t$(1)=alpha$), das heißt, in die Zone können später alle in *ALPHA$* enthaltenen Zeichen eingegeben werden (Buchstaben, Sonderzeichen und Ziffern).

Analog verlaufen die Deklarationen der restlichen Felder. Jeweils ein Feldname, die Position dieses Namens (nicht der Zone!), die Länge der zugehörigen Zone, ihr Typ und eine Zonenvorgabe (immer leer) wird gespeichert.

Der Arrayindex gibt die Nummer des betreffenden Feldes an. Das heißt, die Variablen *N$(4)*, *V$(4)*, *X%(4)*, *Y%(4)*, *L%(4)* und *T$(4)* enthalten die Deklarationen des vierten Feldes mit der Bezeichnung »PLZ«.

In Zeile 380–420 wird die Maske auf dem Bildschirm ausgegeben. *FA* enthält die Feldanzahl (6 Felder). In einer Schleife wird der Cursor mit der Routine SETCURSOR (Syntax: SYS CRS, SPALTE, ZEILE) auf die in *X%(I)* und *Y%(I)* gespeicherte Position des Feldnamens Nummer *I* gesetzt. An dieser Position wird der betreffende Feldname *N$(I)* und ein String aus Leerzeichen in der Länge der Eingabezone *L%(I)* ausgegeben. Da vor der Ausgabe dieser Leerzeichen mit PRINT CHR$(18) der Invers-Modus eingeschaltet wird, erscheint neben dem Namen ein inverser Balken, der die Eingabezone darstellt.

Das folgende Steuerungsprogramm selbst ist dank der Flexibilität der INPUT-Routine sehr kurz (Zeile 450–515). Zuerst wird *LAST$* initialisiert (Zeile 455). Erinnern Sie sich: Für die einwandfreie Funktion von INPUT ist es notwendig, daß jene Stringvariable, die für den Parameter *V/LAST$* verwendet wird, vor dem Aufruf bereits existiert. In diesem Programm wird die Variable *LAST$* verwendet, die mit der Zuweisung LAST$="" angelegt wird (auch wenn ihr nicht ein Zeichen zugewiesen wird).

I soll während der Eingabe in die Maske die aktuelle Feldnummer enthalten, das heißt, die Nummer jenes Feldes, in dessen Eingabezone sich gerade der Cursor befindet. Sinnvoll ist es, den Cursor

zuerst auf die erste Eingabezone zu setzen. Daher wird *I* mit 1 initialisiert (Zeile 460).

Nun folgt die eigentliche Eingabe in das Feld Nummer *I*. Vor dem Aufruf von INPUT wird *NR%* immer der Wert 0 zugewiesen (Zeile 475). Für INPUT bedeutet dieser Wert, daß es sich um eine »normale« Eingabe handelt, bei der der Versuch, die Eingabezone mit [Cursor →] oder [Cursor ←] zu verlassen, einfach ignoriert werden soll.

Die Eingabezone soll unmittelbar rechts neben dem zugehörigen Feldnamen beginnen. Die beim Aufruf von INPUT anzugebende Startspalte ist daher X%(I)+LEN(N$(I)), die Spalte des Feldnamens plus der Länge dieses Namens. Die Zeile ist *Y%(I)*, also identisch mit der Zeile, in der sich der Feldname befindet. *L%(I)* enthält die Länge der Eingabezone Nummer *I*, *V$(I)* die Vorgabe, *T$(I)* die zulässigen Eingabezeichen.

Die Angabe des Strings mit den Endezeichen ist unabhängig vom gerade »editierten« Feld. *E$* enthält die drei Endezeichen [↵], [Cursor ↓] und [Cursor ↑]. *LAST$* wird das Zeichen aufnehmen, mit dem der Benutzer die Eingabe beendet, also eines der drei Endezeichen. Für den Parameter *FLAG%* wird eine 1 angegeben, um die Eingabezonen invers darzustellen. *E%* wird nach dem Beenden der Eingabe einen Wert gleich oder ungleich 0 enthalten, der uns mitteilt, ob die Vorgabe in der aktuellen Zone geändert wurde oder nicht (was in diesem Demoprogramm allerdings nicht weiter interessiert).

Nach dem Beenden der Eingabe speichert INPUT den neuen Zoneninhalt des Feldes Nummer *I* in der Stringvariablen *V$(I)*. Danach wird (Zeile 500–515) der Inhalt von *LAST$* untersucht und geprüft, mit welchem der drei Endezeichen der Benutzer die Eingabe beendete. War es die Taste [Cursor ↓], wird die aktuelle Feldnummer *I* erhöht, wenn nicht bereits das letzte Feld Nummer *FA* (im Beispiel 6) erreicht ist.

Im Gegensatz dazu wird *FA* um 1 vermindert, wenn der Benutzer [Cursor ↑] drückte und sich der Cursor nicht bereits in der ersten Eingabezone befindet, wenn also die Feldnummer *I* größer als 1 ist.

Nach dieser Korrektur der aktuellen Feldnummer wird die Eingaberoutine erneut aufgerufen (Zeile 515), außer wenn der Benutzer ↵ drückte, um die Gesamteingabe zu beenden. Die Eingaberoutine übernimmt nun die Eingabe im neuen Feld Nummer *I*, das je nach Endetaste entweder das vorhergehende oder das nachfolgende Feld ist.

Dieses »Spiel« setzt sich fort, bis der Benutzer ↵ drückt. Mit ↵ wird die Gesamteingabe beendet und INPUT nicht erneut aufgerufen. Stattdessen gibt das Demoprogramm zur Kontrolle die in *V$(1)* bis *V$(FA)* gespeicherten Zoneninhalte auf dem Bildschirm aus (Zeile 530-570).

Einsatz als Bildschirmeditor (NR% < > 0)

Die Eingaberoutine kann eingesetzt werden, um den gesamten Bildschirm zu editieren. In der letzten Anwendung wurde eine Eingabemaske fest im Programm definiert. Die Praxis sieht anders aus: Mit einem Bildschirmeditor »malt« der Benutzer die Maske, wobei ihm der gesamte Bildschirm zur Verfügung steht. Genauer gesagt: fast der gesamte Bildschirm, da in der Praxis die untersten Zeilen meist für Informationen über die Programmbedienung genutzt werden. Gehen wir daher davon aus, daß zum Malen der Maske die ersten 20 Zeilen zur Verfügung gestellt werden.

Nach dem Gestalten der Maske analysiert das Programm die erstellte Maske und findet selbständig heraus, welche Eingabezonen sie enthält, wie lang die Zonen sind und so weiter.

Derartiger Komfort ist sehr leicht mit der INPUT-Routine zu verwirklichen, die problemlos als Bildschirmeditor einzusetzen ist. Sie kann sogar dazu verwendet werden, eine Maske mit Linien zu verzieren.

```
Name:.............  Vorname:............
Strasse:........................
PLZ .... Ort:...................
Telefon:................
Geb:........   Ausbildg:..............
Firma:.........................
```

In diesem Beispiel malten Sie (oder ein anderer Benutzer Ihres Programms) eine Maske für eine Adressverwaltung. Zonen wurden durch Punktreihen gekennzeichnet. Ein entsprechendes Auswertungsprogramm könnte anhand dieser Punktreihen die Positionen und Längen der Eingabezonen ermitteln.

Wie Sie ein solches Analyseprogramm gestalten, ist Ihre Sache. Zuvor müssen Sie wissen, wie die Eingaberoutine für derartige Anwendungen aufzurufen ist.

Im Grunde handelt es sich bei der Maskeneditierung um eine Sonderform einer Eingabemaske, in der zum Beispiel 20 Eingabezonen (die oberen 20 Bildschirmzeilen) mit einer Länge von je 40 Zeichen existieren. Eine Zone ist identisch mit einer Bildschirmzeile. Die Positionen der Zonen entsprechen den Startpositionen der einzelnen Bildschirmzeilen:

Zone Nr.	**Spalte**	**Zeile**
1	0	0
2	0	1
3	0	2
.....		
.....		
20	0	19

Da in jeder Zone alle Zeichen eingegeben werden dürfen, ist *NUM$* überflüssig. Wir benötigen nur *ALPHA$*, der alle üblichen Zeichen enthält (bis auf die Zeichen » + « und » − «, die in diesem Programm eine spezielle Bedeutung besitzen). Als Endezeichen sind diesmal außer [↵], [Cursor ↓] und [Cursor ↑] auch die Zeichen » + « und » − « in *E$* enthalten (Zeile 280).

Eine ausführliche Deklaration aller Zonen ist überflüssig, da Länge, Typ etc. aller Zonen identisch sind. Daher wird in Zeile 306 nur das Stringarray *V$(...)* dimensioniert, das die 20 Eingaben speichern wird. Nach dem Programmstart soll ein leerer Bildschirm ohne Vorgaben erscheinen. Entsprechend werden *V$(1)* bis *V$(20)* Leerstrings zugewiesen (Zeile 400).

LAST$ wird wie in den anderen Beispielen initialisiert (Zeile 455) und *NR%* diesmal nicht 0, sondern 1 zugewiesen! Mit dem Wert 0 für *NR%* fungiert INPUT als gewöhnliche Eingaberoutine. Der

Versuch, den Cursor aus der Eingabezone herauszubewegen, wird einfach ignoriert.

```
200 rem ******************************
210 rem *     eingaberoutine demo3     *
220 rem ******************************
230 :
235 rem * eingabezeichen *
240 alpha$="qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM"
250 alpha$=alpha$+":;$öäÜ:,,./<>?=*^1234567890!#$%&'() "+chr$(34)
260 :
270 rem * endezeichen *
280 e$=chr$(13)+chr$(17)+chr$(145)+"+"+"-":rem return/crs.down/crs.up/+/-
290 :
300 rem * vor-/eingabearray *
306 fa=20:dim v$(fa):rem eingabestrings
400 for i=1 to fa:v$(i)="":next i
440 :
450 rem *** maskensteuerung ***
455 last$="":rem initialis.noetig !!!
470 nr%=1:rem maskeneditor !!!
472 mode=0:rem schreibmodus!!!
474 print chr$(147):rem clear screen
475 i=1:rem aktuelle feldnummer !!!
476 :
480 sys eingabe,0,i-1,40,v$(i),alpha$,e$,last$,nr%,mode,e%
490 :
500 if last$=chr$(17) and i<fa then i=i+1
502 if last$=chr$(145) and i>1 then i=i-1
504 if last$=chr$(29) and i<fa then i=i+1:nr%=1
506 if last$=chr$(157) and i>1 then i=i-1:nr%=40
508 if last$="-" then mode=2:rem malen
509 if last$="+" then mode=0:rem schreiben
515 if last$<>chr$(13) then goto 480
520 :
530 rem * kontrolle *
535 print chr$(147):rem clear screen
540 sys crs,0,15
550 for i=1 to fa
560 : print v$(i)
570 next i
ready.
```

Listing 4.3: *Hauptprogramm von INPUT3*

Hat *NR%* beim Aufruf einen beliebigen anderen Wert als 0, ändert sich die Funktionsweise von INPUT. Der Cursor wird nicht unbedingt auf den Anfang der Eingabezone gesetzt, sondern auf Zeichen Nummer *NR%*. Mit dem Wert 5 für *NR%* können Sie zum Beispiel erreichen, daß sich der Cursor nach dem Aufruf sofort auf dem fünften Zeichen innerhalb der Zone befindet.

Außerdem übergibt INPUT nun in *NR%* die Nummer des Zeichens, auf dem sich der Cursor zuletzt befand. Diese Nummer müssen wir kennen, wenn eine Maske editiert wird.

Ein Beispiel: Der Benutzer befindet sich in der Mitte der obersten Zeile, auf Zeichen Nummer 20. Er drückt [Cursor ↓], um zur nächsten Zone zu gelangen. Beim Editieren des Bildschirms soll der Cursor nun nicht auf den Anfang der nächsten Zone gesetzt werden, also auf Zeichen Nummer 1, sondern auf das Zeichen Nummer 20 dieser folgenden Zone. Nur durch NR% ist es möglich, daß sich der Cursor beim Editieren des Bildschirms nach oben oder unten bewegt und die aktuelle Spaltenposition beibehält.

Hat *NR%* beim Aufruf einen Wert ungleich 0, wird außerdem nun der Versuch, die Eingabezone mit [Cursor →] oder [Cursor ←] zu verlassen, nicht einfach ignoriert. Ein entsprechender Versuch beendet die Eingabe! Das heißt, wenn sich der Cursor zum Beispiel auf dem letzten Zeichen innerhalb der Zone befindet (in unserem Fall also auf Zeichen Nummer 40, dem letzten Zeichen einer Zeile) und der Benutzer erneut [Cursor →] drückt, wird die Eingabe beendet.

Dieses Verhalten ist sinnvoll, da zum Beispiel bei »Dauerfeuer« auf [Cursor →] der Cursor am Ende der aktuellen Zeile nicht einfach stehen bleiben, sondern zum Anfang der darunterliegenden Zeile bewegt werden soll.

Sie sehen, die Zuweisung des Wertes 1 an *NR%* ändert das gesamte Verhalten von INPUT. Die Routine ist nun optimal für die Erstellung von Masken geeignet.

Wie erläutert, können sogar Linien gezogen werden, und zwar ganz einfach mit den Cursortasten. Im »Malmodus« ziehen Sie mit Cursorbewegungen waagrechte ([Cursor →] [Cursor ←]) Linien hinter sich her. Der Malmodus wird eingeschaltet, indem für den Parameter *FLAG%* der Wert 2 angegeben wird.

Um zwischen dem Schreib- und dem Malmodus umschalten zu können, verwendet das Demoprogramm für diesen Parameter im Gegensatz zu den vorhergehenden Beispielen keine Konstante, sondern die Variable *MODE*. *MODE* wird mit 0 initialisiert (Zeile 472), da zu Beginn des Editierens der normale (nicht inverse) Schreibmodus eingeschaltet sein sollte.

Nun wird der Bildschirm gelöscht (Zeile 474), die aktuelle Zonennummer *I* mit 1 initialisiert (Zeile 475) und die Eingaberoutine aufgerufen. Der Aufruf lautet:

```
480 SYS EINGABE, 0, I-1, V$(I), ALPHA$, E$,
LAST$, NR%, MODE, E%
```

Die Spalte ist immer 0. Jede Zone beginnt ganz links. Die Zeile der Zone Nummer *I* errechnet sich mit dem Ausdruck I-1, wenn sich die erste Zone Nummer 1 in der obersten Bildschirmzeile (Zeile 0) befinden soll.

V$(I) ist der Vor-/Eingabestring, der den Inhalt der Zone Nummer *I* speichert. *ALPHA$* gibt an, daß praktisch alle Zeichen eingegeben werden dürfen. *E$* enthält unsere drei »Standard-Endetasten« [↵], [Cursor ↓], [Cursor ↑]. Da *NR%* mit 1 und *MODE* mit 0 initialisiert wurden, befindet sich der Cursor nach diesem Aufruf auf dem ersten Zeichen der ersten Zone (auf der [HOME]-Position), und Sie sind im gewohnten »Schreibmodus« der INPUT-Routine.

Der folgende Programmteil (Zeile 500–515) analysiert die Taste, mit der die Eingabe beendet wurde. War es eine der Tasten [Cursor ↑] oder [Cursor ↓], wird wie gewohnt die aktuelle Zonennummer *I* erhöht, beziehungsweise verringert. Der letzte Wert von *NR%* bleibt erhalten, so daß der Cursor beim nächsten Aufruf von INPUT in der neuen Zone auf die gleiche Spalte wie zuvor gesetzt wird.

In ihrer Eigenschaft als Maskeneditor (*NR%* ungleich 0) behandelt INPUT die Tasten [Cursor →] und [Cursor ↓] wie erläutert als zusätzliche Endezeichen, wenn dadurch der Versuch unternommen wird, den Cursor nach rechts oder links aus der definierten Zone herauszubewegen. In diesem Fall wird die Zonennummer *I* und die Position innerhalb der Zone *NR%* so manipuliert, daß der Cursor beim nächsten Aufruf am entgegengesetzten Ende der nächsten beziehungsweise vorhergehenden Zone erscheint (Zeile 504–506).

Die Tasten [+] und [–] sind ebenfalls Endezeichen (siehe Definition von *E$*). Mit diesen Tasten wird zwischen dem Schreib- und dem Malmodus umgeschaltet (»+«=Schreibmodus / »–«=Malmodus). Wie erläutert ziehen Sie im Malmodus mit [Cursor →] [Cursor ←] waagrechte Linien.

Mit [↵] beenden Sie wie üblich die Gesamteingabe. Anschließend wird zur Kontrolle die gesamte in *V$(1)* bis *V$(20)* enthaltene Maske noch einmal ausgegeben (Zeile 530 – 570).

Steuerung:

[+] Schreibmodus einschalten
[-] Malmodus einschalten
[↵] Gesamteingabe beenden

Sie sehen, INPUT ist weit mehr als eine einfache Eingaberoutine. Bei geeignetem Aufruf und mit einem entsprechenden Basic-Steuerungsprogramm können Sie diese Routine für normale Einzeleingaben, für Maskeneingaben oder gar als Bildschirmeditor einsetzen.

Als Tribut an diese Vielseitigkeit ist allerdings auch der Aufruf entsprechend komplex. Vergessen Sie bei Ihren Versuchen bitte niemals die unbedingt notwendige Initialisierung jener Stringvariablen, die Sie als Parameter *V/LAST$* verwenden!

4.4 MakrosInit

Tastatur-Makros initialisieren

Adresse 39888

Variablenname MAKRO

Syntax SYS MAKRO

Parameter keine

Hinweis Makro-Tabelle : $0600-$06FF (dezimal: 1536 – 1791)

Funktion Der Aufruf von MAKRO aktiviert ein Programm, das es Ihnen ermöglicht, Tastatur-Makros zu definieren. Ein Tastatur-Makro ist die Belegung einer Tastenkombination mit einer ganzen Folge von Tasten. Tastenkombination, weil zur Aktivierung eines Makros immer gleichzeitig die CTRL-Taste zusammen mit einer anderen Taste gedrückt wird.

Das heißt, Sie können mit einem Makro zum Beispiel bewirken, daß dem Rechner beim Druck auf CTRL + e vorgetäuscht wird, Sie würden der Reihe nach die Tasten »H«, »a«, »l«, »l« und »0« drücken. Man sagt, CTRL + e wird mit dieser Tastenfolge »belegt«. Wenn Sie später die Tastenkombination CTRL + e betätigen, hat dies die gleiche Wirkung, als würden Sie der Reihe nach jene Tasten betätigen, mit denen CTRL + e belegt wurde – auf dem Bildschirm erscheint die Zeichenfolge »Hallo«.

Die Einsatzmöglichkeiten von Tastatur-Makros sind vielfältig und letztlich nur durch Ihre Phantasie begrenzt. Einige Beispiele:

1. Funktionstasten zum Abruf häufiger Basic-Kommandos: Tastaturmakros können die Eingabe häufig vorkommender Basic-Kommandos wie LIST, LOAD, SAVE etc. vereinfachen. Zum Beispiel können Sie die Tastenkombination CTRL + l mit der Zeichenfolge »LOAD"« und CTRL + s mit »SAVE"« belegen. Nach der Betätigung von CTRL + s wird bereits »LOAD"« auf dem Bildschirm stehen. Sie müssen nur noch den Rest der vollständigen LOAD-Anweisung eingeben (»NAME",8«).

2. Individuell wählbare Programmbedienung: Nehmen wir an, Sie schreiben eine Textverarbeitung. Möglicherweise funktioniert das Speichern des aktuellen Textes so: Sie drücken zuerst [F1], um zum Hauptmenü zu gelangen. Anschließend [d], um ein weiteres Menü »Disk-Operationen« zu aktivieren. Dann folgt die Taste »s«, um das Kommando »SAVE« anzuwählen. Und nun fragt Sie das Programm noch »Sind Sie sicher (j/n) ?«. Auf diese Frage dürfen Sie nun »j« eingeben und noch [↵] drücken.

Das heißt, Sie benötigen die Tastenfolge [F1][d][s][j][↵]. Wenn Ihnen diese Tastenfolge zu langwierig ist oder ganz einfach nicht gefällt, erstellen Sie ein Tastatur-Makro und legen die Tastenfolge auf eine beliebige [CTRL]-Tastenkombination wie [CTRL]+[s]. Ab diesem Zeitpunkt genügt es, [CTRL]+[s] zu drücken, um den Text zu speichern.

Beachten Sie bitte, daß das Makro im zweiten Beispiel nicht nur Buchstaben, sondern auch Sondertasten wie [F1] und [↵] enthält! Tatsächlich können Sie bei der »Makrodefinition« jede beliebige Taste verwenden, zum Beispiel auch die Cursortasten!

MAKRO besteht aus zwei Teilen, einem Makro-Recorder, der die gewünschte Tastenfolge aufzeichnet, und einem Wiedergabeteil, der prüft, ob eine [CTRL]-Tastenkombination gedrückt wurde und die zugehörige Tastenfolge simuliert. Beide Programmteile können nach der (einmaligen) Initialisierung mit SYS MAKRO jederzeit verwendet werden, sowohl im Direkt-Modus als auch in einem laufenden Programm.

Makro-Aufzeichnung einschalten/ausschalten ([C=] + [↵])

Um ein Makro zu definieren, müssen Sie zuerst die Makro-Aufzeichnung einschalten, und zwar mit der Tastenkombination [C=] + [↵]. Der Bildschirm-Rahmen wird heller als zuvor. Das Utility MAKRO teilt Ihnen auf diese Weise mit, daß es zur Aufzeichnung eines Makros bereit ist.

Sie geben nun an, welche Tastenkombination Sie belegen wollen.

Nehmen wir an, Sie wollen [CTRL]+[d] mit der Tastenfolge zum Laden des Inhaltsverzeichnisses belegen: »LOAD"$",8«

Drücken Sie bitte [CTRL]+[d]. Scheinbar geschieht nichts. Tatsächlich hat sich MAKRO jedoch gemerkt, daß die Tastenkombination [CTRL]+[d] nun mit einer Zeichenfolge belegt werden soll.

Geben Sie anschließend die gewünschte Tastenfolge ein, drücken Sie also der Reihe nach die Tasten [L] [O] [A] [D] ["] [$] ["] [,] und [8].

Wie Sie sehen, speichert MAKRO nicht nur die Tastenfolge, sondern leitet sie weiter. Das heißt, jede Taste übt die gewohnte Wirkung aus. Daher sehen Sie sofort, ob Sie auch wirklich die richtige Taste gedrückt haben oder das Makro fehlerhaft ist und Sie es neu definieren müssen.

Ich gehe davon aus, daß Sie [CTRL]+[d] korrekt mit der Zeichenfolge »LOAD"$",8« belegt haben. Die »Aufzeichnung« muß nun beendet werden. Mit [C=] + [↵] können Sie die Makro-Aufzeichnung nicht nur ein-, sondern auch wieder ausschalten.

Drücken Sie daher bitte [C=] + [↵]. Der Untergrund nimmt wieder seine gewohnte Farbe an – der »Aufzeichnungsmodus« ist ausgeschaltet. Ab jetzt können Sie das Makro jederzeit mit [CTRL]+[d] aufrufen, und es erscheint die Zeichenfolge »LOAD"$",8«. Sie müssen nur noch [↵] drücken, um das Inhaltsverzeichnis zu laden.

Übrigens: Theoretisch wäre es möglich gewesen, auch noch die Taste [↵] zu speichern. Diese Ergänzung ist jedoch nicht empfehlenswert. Stellen Sie sich vor, Sie drücken bei der Erstellung eines Programms versehentlich [CTRL]+[d]. Dann müssen Sie hilflos zuschauen, wie das Inhaltsverzeichnis geladen und Ihr aktuelles Programm dadurch überschrieben wird!

Hier noch einmal zusammengefaßt die Vorgehensweise zur Definition eigener Makros (Voraussetzung: MAKRO wurde mit SYS MAKRO initialisiert):

1. Mit [C=] + [↵] die Aufzeichnung einschalten.
2. Die zu belegende [CTRL]-Kombination drücken, zum Beispiel [CTRL]+[d] oder [CTRL]+[s].
3. Die Tastenfolge eingeben, die Sie speichern wollen.
4. Mit [C=] + [↵] die Aufzeichnung wieder ausschalten.

Auch Makros besitzen ihre technischen Grenzen. Diese Grenzen können Sie sehr leicht selbst feststellen. Versuchen Sie, die Zeichenfolge »Dies ist ein Makro« auf die Tastenkombination [CTRL] + [x] zu legen.

Schalten Sie bitte mit [C=] + [↵] die Aufzeichnung ein. Betätigen Sie anschließend [CTRL] + [x] und tippen Sie diese Tastenfolge ein. Nach dem zehnten Zeichen »c« geschieht etwas Merkwürdiges: Der Bildschirm-Hintergrund wird von selbst wieder dunkel, obwohl Sie die Aufzeichnung noch nicht mit [C=] + [↵] beendeten!

Sie lernten soeben die Grenzen eines Makros kennen. Die Länge einer Makrodefinition ist auf genau zehn Zeichen beschränkt. Nach dem zehnten Zeichen wird die Aufzeichnung automatisch abgeschaltet!

Auch die Gesamtanzahl aller definierten Makros ist beschränkt. Sie können maximal 20 Makros definieren, was in der Praxis sicher ausreicht.

Löschen einzelner Makros / aller Makros

Zum Löschen von Makros gibt es zwei verschiedene Methoden. Die einfachste: Sie drücken gleichzeitig die Tasten [C=] (Commodore-Taste) und die Leertaste, wodurch alle definierten Makros mit einem Schlag gelöscht werden.

Probieren Sie es bitte aus. Für einen kurzen Moment wird der normalerweise dunkle Hintergrund hell. Diese optische Anzeige sagt Ihnen, daß nun alle Makros gelöscht wurden. Wenn Sie nun [CTRL] + [d], [CTRL] + [x] oder ein anderes Makro »aufrufen«, geschieht nichts.

Mit [C=] + [Leertaste] können Sie alle Makro-Definitionen auf einmal löschen. Wollen Sie jedoch nur ein einzelnes Makro löschen, gehen Sie so vor:

- Mit [C=] + [↵] die Aufzeichnung einschalten
- Die Tastenkombination des zu löschenden Makros betätigen, zum Beispiel [CTRL] + [d]
- Mit [C=] + [↵] die Aufzeichnung ausschalten

Sie haben soeben CTRL + d mit »nichts« belegt, was mit dem Löschen der bisherigen Belegung identisch ist!

Makros speichern/laden: Tastatur-Makros nützen Ihnen wenig, wenn alle definierten Makros beim Ausschalten des Rechners verloren gehen. Das muß jedoch nicht sein. Alle Makro-Definitionen werden im Speicherbereich $0600–$06FF (dezimal: 1536–1791) abgelegt. Mit den Utilities MEMORYSAVE und MEMORYLOAD können Sie beliebige Speicherbereiche auf Diskette speichern und wieder laden, also auch jenen Bereich, in dem sich die Makro-Definitionen befinden. Wurde das bereits initialisiert, sind nach dem Laden alle gespeicherten Makros wieder aktiv und können sofort wieder benutzt werden.

4.5 SatzInfo

Indexanzeige

Adresse 38921

Variablenname SI

Syntax SYS SI, SPALTE, ZEILE, ZAHL1, ZAHL2

Parameter

SPALTE: Spaltenposition der Anzeige

ZEILE: Zeilenposition der Anzeige

ZAHL1: Erste auszugebende Zahl

ZAHL2: Zweite auszugebende Zahl

Funktion SATZINFO ist eine sehr spezielle Funktion und bei weitem nicht so vielseitig einsetzbar wie die restlichen Utilities. Sie dürfen es als eines der erwähnten »Abfallprodukte« ansehen, das im ein oder anderen Fall hilfreich sein kann.

```
200 rem ******************************
210 rem *        satzinfo-demo         *
220 rem ******************************
230 :
240 rem ** pseudo-tabelle erzeugen **
250 dim x(100)
260 for i=1 to 100
270 : x(i)=rnd(0)
280 next i
290 :
300 rem ** tabelle durchblaettern **
310 nr=1:rem start mit wert nr.1
315 :
320 print chr$(147):rem clear screen
330 for i=nr to nr+9
340 : print x(i)
350 next i
360 :
370 sys si,12,0,nr,100
380 :
390 get a$:if a$<>"+" and a$<>"-" then goto 390
400 if a$="+" and nr<90 then nr=nr+10
410 if a$="-" and nr>10 then nr=nr-10
420 goto 320
ready.
```

Listing 4.4: *Hauptprogramm von SATZINFO*

SATZINFO gibt an einer frei wählbaren Bildschirmposition (*SPALTE* und *ZEILE*) die Information »*ZAHL1* von *ZAHL2*« aus, zum Beispiel »23 von 150«, wenn als Parameter *ZAHL1* eine 23 und als Parameter *ZAHL2* eine 150 angegeben wurde.

Nützlich ist dieses Utility in Programmen mit großen Datenmengen, um die Übersicht nicht zu verlieren. Zum Beispiel in einer Adreßverwaltung, in der Adressen wie in einem Karteikasten durchblättert werden können. In einem solchen Programm könnte mit SATZINFO ständig die Nummer der aktuellen Adresse und die Gesamtzahl aller vorhandenen Adressen angezeigt werden.

In den Zeilen 240–280 wird ein Array *X(1)* bis *X(100)* angelegt, das Zufallszahlen enthält. Diese 100 Zufallszahlen stehen stellvertretend für eine »echte« Tabelle.

Diese Tabelle können Sie nun durchblättern, und zwar mit den Tasten [+] und [-]. Jeweils 10 Werte werden auf dem Bildschirm angezeigt. Mit [+] blättern Sie zehn Werte weiter, mit [-] 10 Werte zurück. Mit SATZINFO wird ständig angezeigt, welche Nummer der momentan angezeigte Wert besitzt (in *I* enthalten), und wieviele Werte die Tabelle insgesamt enthält (im Beispiel: 100 Werte).

4.6 SetCursor

Cursor positionieren

Adresse 38918

Variablenname CRS

Syntax `SYS CRS, SPALTE, ZEILE`

Parameter SPALTE: Spaltenposition (0–39)

ZEILE: Zeilenposition (0–24)

Funktion SETCURSOR setzt den Cursor auf eine in *SPALTE* und *ZEILE* angegebene Position. *SPALTE* und *ZEILE* sind ganzzahlige Werte zwischen 0 und 39 (Cursorspalte) beziehungsweise 0 und 24 (Cursorzeile).

Beispiel

```
SYS CRS,0,0: PRINT "HALLO"
```

setzt den Cursor auf Spalte 0 von Zeile 0 (linke obere Bildschirmecke) und gibt ab dieser Position die Zeichenkette »HALLO« aus.

4.7 Strout

Schnelle Stringausgabe für Masken

Adresse 38918

Variablenname AUSGABE

Syntax `SYS AUSGABE, SPALTE%(START), ZEILE%(START), LAENGE% (START), STRING$(START), ANZAHL, MODE`

Parameter

SPALTE%: (START)	Erstes Element des Arrays, das die Spaltenpositionen der Ausgabezonen enthält.
ZEILE%: (START)	Erstes Element des Arrays, das die Zeilenpositionen der Ausgabezonen enthält.
LAENGE%: (START)	Erstes Element des Arrays, das die Längen der Ausgabezonen enthält.
STRING$: (START)	Erstes Element des Arrays, das die in den einzelnen Zonen auszugebenden Strings enthält.
ANZAHL:	Anzahl der auszugebenden Strings
MODE:	0 = normale Ausgabe 1 = inverse Ausgabe

Funktion STROUT gibt eine beliebige Anzahl an Strings in vorher definierten Ausgabezonen aus. Diese Funktion ist vor allem für Masken sehr nützlich. Jede Zone einer Maske, in der Informationen ausgegeben werden (Ausgabezone), befindet sich an einer definierten Position. Diese Positionen (Spalte und Zeile) und die Längen der Ausgabezone sind in drei Integerarrays zu speichern.

In diesen Zonen auszugebende Informationen (Zeichenketten) sind in einem Stringarray zu speichern. Jeder Arrayindex (1, 2, 3, ...) kennzeichnet einen String und die Definition seiner zugehörigen Ausgabezone. Unter Angabe der Anzahl auszugebender Strings und des Darstellungsmodus (normal oder invers) gibt STROUT alle Strings an den definierten Positionen aus. Strings, die länger sind als die definierte Zonenlänge, werden abgeschnitten. Werden Strings ausgegeben, die kürzer als die zugehörige Zonenlänge sind (Normalfall), so wird der Zonenrest mit Leerzeichen aufgefüllt.

```
200 rem ******************************
210 rem *       stringausgabe          *
220 rem ******************************
230 :
240 fa=6:dim x%(fa),y%(fa),l%(fa),s$(fa)
250 :
255 rem ** ausgabezonen **
260 x%(1)=5:y%(1)=1:l%(1)=14
270 x%(2)=29:y%(2)=1:l%(2)=10
280 x%(3)=8:y%(3)=3:l%(3)=28
290 x%(4)=4:y%(4)=5:l%(4)=4
300 x%(5)=13:y%(5)=5:l%(5)=25
310 x%(6)=8:y%(6)=7:l%(6)=15
320 :
330 rem ** ausgabe von adresse 1 **
340 s$(1)="Maier":s$(2)="Hans":s$(3)="Aalweg 10":s$(4)="8000"
350 s$(5)="M$nchen":s$(6)="089/123456"
360 :
370 sys ausgabe,x%(1),y%(1),l%(1),s$(1),fa,1
380 :
390 print:print:print "Taste dr$cken"
400 get a$:if a$="" then goto 400
410 :
420 rem ** ausgabe von adresse 2 **
430 s$(1)="M$ller":s$(2)="Wilhelm":s$(3)="Ottostr.1a":s$(4)="6800"
440 s$(5)="Mannheim":s$(6)="0621/52756"
450 :
460 sys ausgabe,x%(1),y%(1),l%(1),s$(1),fa,1
ready.
```

Listing 4.5: *Hauptprogramm von STROUT*

Das Demoprogramm geht von folgender Maske aus, die typisch ist für eine Adreßverwaltung. Die Maske enthält sechs Zonen, wobei in jeder Zone ein Teil einer Adresse erscheinen soll.

```
Name:.............. Vorname:..........

Strasse:............................

PLZ:.... Ort:.........................

Telefon:...............
```

Die durch Punktreihen markierten Ausgabezonen dieser Maske sind wie folgt definiert:

Zone	Spalte	Zeile	Länge
1	5	1	14
2	29	1	10
3	8	3	28
4	4	5	4
5	13	5	25
6	8	7	15

Dies Zonenparameter speichert das Demoprogramm (Zeile 240–310) in den drei Integerarrays *X%(..)* (Spalten), *Y%(..)* (Zeilen) und *L%(..)* (Längen).

Der folgende Programmteil (Zeile 340–350) speichert in dem Stringarray *S$(1)* bis *S$(6)* eine Adresse. In der Praxis entspricht dies einer von der Diskette in dieses Array eingelesenen Adresse.

Zeile 370 gibt diese Adresse mit STROUT aus. Angegeben wird außer dem Startindex (dem Index des ersten auszugebenden Strings und der zugehörigen Zonenvariablen) der Parameter *ANZAHL*, also die Anzahl auszugebender Strings *FA*, und der Wert 1 für den Parameter *MODE*, da die Ausgabe invers erfolgen soll.

Danach wartet das Programm darauf, daß Sie eine beliebige Taste drücken.

Anschließend wird eine andere Adresse in *S$(1)* bis *S$(6)* gespeichert (Zeile 430–440) und mit dem gleichen Aufruf von STROUT ausgegeben (Zeile 460).

In der Praxis entspricht dieses Zuweisen einer Adresse, wie erläutert, zum Beispiel dem Einlesen einer Adresse aus einer Adreßdatei. Nach jedem Einlesen einer neuen Adresse genügt der immer gleiche Aufruf von STROUT, um die komplette Adresse an vor+-definierten Positionen in der Maske auszugeben.

Allgemein ausgedrückt: STROUT wird eingesetzt, wenn wiederholt Strings an fest vorgegebenen Positionen möglichst schnell und ohne Aufwand auszugeben sind, wie es unter anderem bei der Ausgabe in den festgelegten Zonen einer Maske der Fall ist.

5

Drucker

Die Utilities bieten Ihnen die Möglichkeit der Druckeranpassung. Das heißt, mit Hilfe verschiedener Tools können nicht nur die »serienmäßigen« Zeichen Ihres C64, sondern auch die neu hinzugekommenen Umlaute korrekt gedruckt werden (sofern Ihr Drucker Umlaute überhaupt kennt).

Zusätzlich enthalten die Utilities eine integrierte Centronics-Schnittstelle. Das heißt, Sie können einen Fremddrucker, der nicht von der Firma Commodore stammt, mit einem geeigneten Kabel (in nahezu jedem Fachgeschäft erhältlich) ohne ein zwischengeschaltetes Interface direkt an den Userport Ihres C64 anschließen. Mit den Utilities PARALLELINIT und SERIELLINIT können Sie jederzeit zwischen der »parallelen« Ausgabe (über den Userport) und der normalen »seriellen« Ausgabe umschalten.

Centronics-Schnittstelle und Codetabellen

Was Sie zu tun haben, um Ihren Drucker an den neuen Zeichensatz anzupassen, ist leider je nach Druckertyp höchst unterschiedlich. Hier nun eine Kurzanleitung, die Ihnen dann ausreicht, wenn Sie sich bereits recht gut mit Ihrem Drucker (und vor allem, falls verwendet, Ihrem Interface!) auskennen. Prinzipiell sind drei Druckertypen zu unterscheiden:

1. Die Commodore-Drucker MPS 801, MPS 803 und dazu völlig kompatible Drucker.
2. Nicht-kompatible Drucker, bei denen die Kompatibilität mit einem Interface hergestellt wird, z. B. einem Interface von Wiesemann, Data Becker oder Görlitz.
3. ASCII-Drucker ohne Interface, die direkt am Userport Ihres C64/C128 angeschlossen werden.

Die Utilities kennen zwei verschiedene Methoden der Datenübertragung zwischen Rechner und Drucker:

1. Serielle Ausgabe

   Seriell heißt, daß der Drucker ebenso wie die Floppy seine Daten über die serielle Schnittstelle des C64 empfängt (rundes, fünfpoliges Kabel). Die serielle Schnittstelle ist die Standard-Schnittstelle des C64 und wird verwendet, wenn Sie einen Commodore-Drucker oder aber einen Drucker mit einem für den C64 geeigneten Interface verwenden, egal ob dieses Interface im Drucker eingebaut ist oder sich zwischen Drucker und Rechner befindet.

2. Parallele Ausgabe

   Die Utilities besitzen eine integrierte Centronics-Schnittstelle, die eine »parallele« Ausgabe ermöglicht. Das heißt, Sie können einen Drucker, der nicht zu den Typen MPS 801 und MPS 803 kompatibel ist, auch ohne Interface direkt am Userport des C64 anschließen! Allerdings benötigen Sie hierzu ein spezielles Kabel, das auf der einen Seite mit dem Drucker und auf der anderen mit dem Userport Ihres Rechners verbunden wird (recht billig im Fachhandel erhältlich).

Voreingestellt ist die Ausgabe über die serielle Schnittstelle. Die Art der Datenübertragung interessiert Sie daher nicht weiter, wenn Sie einen Commodore-Drucker (MPS 801, MPS 803 oder dazu voll Kompatiblen) oder einen Drucker mit Interface benutzen.

Wenn Sie wollen, können Sie Ihren Drucker jedoch mit einem geeigneten Kabel auch direkt (ohne Interface) an den Userport anschließen wollen und mit dem Utility PARALLELINIT die Druckausgaben auf den Userport »umlenken«. Ein einmaliger Aufruf von PARALLELINIT ohne weitere Parameterangaben genügt. Es ist möglich, mit dem Utilities SERIELLINIT jederzeit wieder auf die serielle Ausgabe zurückzuschalten.

Dieser Aufruf von PARALLELINIT ist sogar schon für Sie vorbereitet, und zwar in Zeile 117 im Initialisierungsprogramm TOOLS.INIT:

```
117 REM SYS PARALLEL:REM USERPORT ???
```

Wenn Sie Ihren Drucker am Userport anschließen, entfernen Sie einfach die Anweisung REM, damit PARALLELINIT aufgerufen wird.

Kommen wir nun zum eigentlichen Problem, der Anpassung von Rechner- und Druckercodes. Ohne Anpassung ist es nicht möglich, den auf dem Bildschirm sichtbaren geänderten Zeichensatz korrekt auszudrucken (übrigens auch nicht die Zeichen »;« und »:«).

Die benötigte Anpassung ist im Initialisierungsprogramm TOOLS.INIT bereits für Sie vorbereitet, und zwar in den Zeilen 120 und 121.

```
120 REM N$="COMMODORE.COD,P,R":REM COMMODORE-CODES
121 N$="ASCII.COD,P,R":REM ASCII-CODES ???
```

Wenn Sie einen Commodore-Drucker besitzen, der nicht über einen echten ASCII-Zeichensatz verfügt, lassen Sie diese Zeilen bitte unverändert.

Besitzt Ihr Drucker jedoch einen ASCII-Zeichensatz (und damit auch Umlaute), dann fügen Sie am Anfang von Zeile 120 bitte die Anweisung REM ein und entfernen das REM am Anfang von Zeile 121.

```
120 N$="COMMODORE.COD,P,R":REM COMMODORE-CODES
121 REM N$="ASCII.COD,P,R":REM ASCII-CODES ???
```

Ist Ihr Drucker über die parallele Schnittstelle direkt am Userport angeschlossen, klappt nun alles einwandfrei. Der Druck normaler Zeichen und der Ausdruck von Umlauten.

Besitzt Ihr Drucker jedoch ein Interface (egal, ob eingebaut oder zwischen Drucker und Rechner angeschlossen), wird er nun völlig wirre Ausdrucke liefern. Sie müssen die Codewandlung Ihres Interfaces abschalten, den sogenannten Linearkanal einschalten. Wenn Sie wissen, wie das geht: Tun Sie es und auch Sie erhalten einwandfreie Ausdrucke. Wenn Sie es nicht wissen, lesen Sie bitte den folgenden Abschnitt, der Ihnen alles Wissenswerte über die Hintergründe der Codewandlung Ihres Interfaces und der Utilities vermittelt.

Optimale Druckeranpassung durch modifizierbare Codetabellen

Jeder Rechner ordnet den verfügbaren Zeichen Zahlen, sogenannte interne Zeichencodes zu. Diese Codes finden Sie in Ihrem C64-Handbuch im Anhang – in der sogenannten ASCII-Tabelle. In dieser Tabelle ist jedem Zeichen ein ganz bestimmter Code zugeordnet.

Über eine solche Tabelle verfügt auch Ihr Drucker. In der Praxis passiert zum Beispiel beim Ausdruck eines »A« folgendes:

- Ihr C64 schaut in seiner Tabelle nach und sendet dem Drucker jenen Code, der laut Tabelle dem Zeichen »A« zugeordnet ist.
- Der Drucker empfängt diesen Code und durchsucht ebenfalls seine Tabelle, um herauszubekommen, welches Zeichen er nun drucken soll.

Dieser Vorgang klappt verständlicherweise nur dann reibungslos, wenn die Tabellen von Rechner und Drucker identisch sind. Leider verwendet Commodore seit jeher eine ASCII-Tabelle, die stark vom genormten Standard abweicht. Zum Beispiel ordnet Ihr C64 dem Zeichen »a« den Code 65 zu. 65 ist in der »echten« ASCII-Tabelle jedoch der Code des großgeschriebenen Zeichens »A«!

Die Firma Commodore hat in ihrer Tabelle Groß- und Kleinbuchstaben einfach vertauscht! Außerdem fehlen in der Commodore-Tabelle Umlaute – einfach, weil der C64 überhaupt keine Umlaute kennt!

Wenn Sie einen echten Commodore-Drucker besitzen, ist das allerdings egal. Denn Commodore-Drucker besitzen die gleiche »fehlerhafte« ASCII-Tabelle wie der C64. Das heißt, der MPS 801 und der MPS 803 verstehen unter dem Code 65 ebenfalls ein kleines »a«.

Jeder »normale« Drucker wird jedoch statt Großbuchstaben Kleinbuchstaben drucken und umgekehrt. Wenn Sie keinen Commodore-Drucker besitzen, wundern Sie sich nun sicher, daß das bei Ihnen nicht der Fall ist.

Der Grund dafür ist das von Ihnen verwendete Interface. Vielleicht wissen Sie gar nicht, daß Sie ein Interface benutzen. Oft sind Interfaces nicht nachträglich zwischen Drucker und Rechner angeschlossen, sondern im Drucker eingebaut, zum Beispiel bei dem weitverbreiteten STAR NL-10.

Ihr Interface besitzt vor allem die Aufgabe der Codewandlung. Das heißt, es übersetzt die von Ihrem C64 gesendeten »fehlerhaften« Codes in die korrekten ASCII-Codes, die Ihr Drucker erwartet. Drucken Sie zum Beispiel das Zeichen »a«, sendet Ihr C64 den Code 65. Das Interface wird nun diesen Code abfangen und Ihrem Drucker stattdessen den korrekten ASCII-Code 97 übermitteln.

Bei den Umlauten ergibt sich nun folgendes Problem: Umlaute sind beim C64 nicht vorgesehen, sie besitzen daher keine standardisierten Codes in der »Commodore-ASCII-Tabelle«. Sollten Sie das nicht glauben, schauen Sie einfach in der ASCII-Tabelle Ihres C64-Handbuches nach – Sie werden keine Umlaute entdecken!

Jeder Programmierer, der für den C64 einen neuen Zeichensatz konstruiert, der die fehlenden Umlaute enthält, steht nun vor dem gleichen Problem: Welche Codes soll er diesen neuen Zeichen zuordnen?

Theoretisch wäre die beste Lösung, den Umlauten einfach die zugehörigen Codes der »echten« ASCII-Tabelle zuzuordnen – denn die »echte« ASCII-Tabelle enthält sehr wohl Umlaute!

Diese Lösung scheidet leider aus technischen Gründen aus. Daher verwendet zwangsläufig jedes Programm, das Umlaute ermöglicht (VIZAWRITE, DATAMAT etc. – und auch die Utilities!), mangels Standardisierung seine individuellen Umlaut-Codes. Im Falle der Utilities ergeben sich durch den neuen Zeichensatz folgende Änderungen gegenüber der Commodore-Tabelle:

Zeichen	Code	Zeichen	Code	Zeichen	Code
ö	58	:	62	ß	92
ä	59	ü	64	Ä	93
;	60	Ö	91	Ü	186

Bis auf diese Zeichen (außer den Umlauten auch die Zeichen » ; « und » : «) entspricht die von den Utilities verwendete Codetabelle exakt der originalen Commodore-Codetabelle.

Das heißt auch, daß Ihr Commodore- oder Fremddrucker alle Zeichen bis auf diese völlig korrekt druckt. Bis auf diese Zeichen, da zum Beispiel das Zeichen » ; « den Code 60 besitzt. 60 ist aber normalerweise der Code des Zeichens » < «!

Angenommen, Sie wollen das Zeichen » ; « ausdrucken. An den Drucker wird der neue Code dieses Zeichens gesendet, also 60. Statt einem » ; « erhalten Sie auf dem Papier das Zeichen » < «!

Um dieses Problem zu lösen, müßten Sie eine Codewandlung vornehmen können, ähnlich wie ein Interface. Es müßte möglich sein, anzugeben: »sende für das Zeichen » ; « nicht den Code 60, den es neuerdings besitzt, sondern stattdessen den Code 59«.

Mit den Utilities können Sie eine Codetabelle anlegen, die für jedes Zeichen den zu sendenden Code enthält. Auf der Diskette befinden sich drei fertig vorbereitete Codetabellen. Nach dem Start lädt das Initialisierungsprogramm automatisch die Tabelle COMMODORE.COD.

»N.D.« bedeutet »nicht darstellbar«. Diese Zeichen sind keine Zeichen im eigentlichen Sinn. Sie können nicht auf dem Bildschirm dargestellt werden und interessieren uns nicht weiter. Die darstellbaren Zeichen beginnen ab 32 (Leerzeichen).

Zur Interpretation der Tabelle: Die linke Spalte enthält die Nummer eines Zeichens (den internen Code im geänderten Zeichensatz). Die rechte Spalte enthält jenen Code, den die Utilities für dieses Zeichen zum Drucker senden.

Wie Sie sehen, wird nur selten eine Codewandlung vorgenommen. Für fast jedes Zeichen wird der Originalcode unverändert zum Drucker gesendet. Ausnahmen sind natürlich die (im Original-Zeichensatz überhaupt nicht enthaltenen) Umlaute und zusätzlich die Zeichen » ; « und » : «.

interner Code	**Zeichen**	**gesendeter Code**
1	n.d.	1
2	n.d.	2
3	n.d.	3
.....		
.....		
32	Leerzeichen	32
33	!	33
34	"	34
35	#	35
.....		
.....		

interner Code	Zeichen	gesendeter Code
56	8	56
57	9	57
58	ö	58
59	ä	59
60	;	59
61	=	61
62	:	58
63	?	63
64	ü	64
65	a	65
66	b	66
67	c	67
68	d	68
.....		
.....		
.....		

Tabelle COMMODORE.COD

Das Zeichen » ; « besitzt normalerweise den Code 59. Durch die Änderung des Zeichensatzes besitzt nun jedoch das Zeichen »ä« diesen Code – und » ; « den neuen Code 60, den der Drucker falsch interpretieren wird!

Daher steht in der rechten Tabellenspalte (gesendeter Code) 59, jener Wert, den der Drucker als » ; « auffaßt. Dieser Tabelleneintrag entspricht somit folgender Anweisung:

»Sende dem Drucker für das Zeichen mit dem Code 60 (» ; «) nicht diesen Code, sondern stattdessen den Code 59«.

Das Gleiche gilt für das Zeichen » : «, das durch die Änderung des Zeichensatzes nun statt dem Code 58 den neuen Code 62 besitzt. Als zu sendender Code ist in der Tabelle weiterhin der alte Code 58 eingetragen.

Diese Tabelle ist optimal für Sie geeignet, wenn Sie einen MPS 801, MPS 803 oder dazu voll kompatiblen Drucker besitzen. Sie sorgt dafür, daß die Zeichen » ; « und » : « korrekt gedruckt werden. Um die Codes der Umlaute müssen Sie sich nicht weiter kümmern. Da Ihr Drucker Umlaute überhaupt nicht kennt, dürfen Sie jede Hoffnung aufgeben, jemals Umlaute auf dem Papier zu sehen.

Wenn Sie einen Drucker mit eingebautem oder zwischengeschaltetem Interface verwenden, wird ebenfalls alles korrekt gedruckt (außer den Umlauten) – auch die Zeichen » ; « und » : «.

Dank der Tabelle erhält Ihr Interface auch für » ; « und » : « die Original-Commodore-Codes. Und diese Codes wird Ihr Interface wie beschrieben in die »echten« ASCII-Codes umwandeln, die Ihr Drucker erwartet.

Komplizierter wird es, wenn Sie auch Umlaute drucken wollen, wozu Ihr Drucker ja in der Lage ist. Theoretisch sollte man annehmen, es genügt, in der Codetabelle einfach die korrekten ASCII-Codes der Umlaute einzutragen. Zum Beispiel den Code 215 für das Zeichen »ö«.

Dabei spielt allerdings das Interface nicht mit! Bedenken Sie: Das Interface ist darauf eingerichtet, für die Commodore-Codes die korrekten ASCII-Codes an den Drucker zu senden. Wenn Sie zum Interface für ein »ö« den korrekten ASCII-Code 215 senden, wird auch dieser gewandelt. Und genau das darf auf keinen Fall passieren!

Das heißt, um Umlaute auszudrucken, bleibt Ihnen keine Wahl: Sie müssen die Codewandlung des Interfaces abschalten und allein mit der Codewandlung der Utilities arbeiten.

Interfaces gibt es wie am Sand am Meer – und ebenso viele Möglichkeiten, ihre Codewandlung abzuschalten!

Jedes Interface hat einen Linearkanal, in dem es Daten unverändert so zum Drucker weiterleitet, wie sie vom Rechner kommen. Dieser Linearkanal wird leider je nach Interface auf sehr unterschiedliche Art und Weise eingeschaltet. Wie es bei Ihrem speziellen Interface geht, kann ich Ihnen nicht sagen. Diese Information müssen Sie dem Handbuch zum Interface beziehungsweise zum Drucker (wenn das Interface im Drucker eingebaut ist) entnehmen. Prinzipiell gibt es folgende Möglichkeiten:

1. Mit einem »DIP-Schalter«, einem winzigen Schalter am Drucker oder Interface.
2. Durch Verwendung einer speziellen »Sekundäradresse«.
3. Mit einer speziellen »ESC-Sequenz«.

1. ist die einfachste Lösung. In diesem Fall bringen Sie den Schalter in die entsprechende Stellung und die Daten werden ohne Wandlung durch das Interface übertragen.

Die Verwendung einer speziellen Sekundäradresse ist ebenfalls harmlos. Angenommen, Sie müssen die Sekundäradresse 100 verwenden. Dann öffnen Sie in Ihren Basic-Programmen eine Druckerdatei eben nicht wie üblich mit OPEN 1,4, sondern mit OPEN 1,4,100.

Bei der ESC-Sequenz-Methode senden Sie diese Sequenz sofort nach dem Öffnen der logischen Datei an den Drucker. Wird zum Beispiel laut Handbuch die Sequenz »27 5« benötigt, schreiben Sie:

```
OPEN 1,4
PRINT#1,CHR$(27) CHR$(5)
PRINT#1,"HALLO"
CLOSE 1
```

Nachdem das geschafft ist, dürfen Sie nun die Tabelle COMMODORE.COD komplett ändern. Da das Interface »ausgeblendet« ist, müssen nun die Utilities jeden einzelnen Commodore-Code in den zugehörigen echten ASCII-Code umwandeln.

Das Gleiche gilt auch, wenn Sie Ihren Drucker direkt – ohne Interface – am Userport anschließen.

Keine Angst, diesen Aufwand habe ich Ihnen abgenommen. Auf der Diskette befindet sich die Codetabelle ASCII.COD. Diese Tabelle enthält für jeden Code des neuen Zeichensatzes – auch für die Umlaute – den zugehörigen ASCII-Code. Das Initialisierungsprogramm TOOLS.INIT übernimmt die Aufgabe, mit Hilfe einer dieser beiden Tabellen – ASCII.COD oder COMMODORE.COD – die benötigte Codewandlung vorzunehmen.

```
116 REM ** CODEWANDLUNG **
117 REM SYS PARALLEL:REM USERPORT ???
118 :
119 POKE 2,0:REM WANDLUNG INITIAL. !!!
120 N$="COMMODORE.COD,P,R":REM COMMODORE-CODES
121 REM N$="ASCII.COD,P,R":REM ASCII-CODES ???
122 SYS VLADEN, N$, 1792, 1792+256:REM CODETABELLE LADEN
```

Nach dem Start nimmt dieser Programmteil die Druckeranpassung vor. Wenn Sie Ihren Drucker am Userport anschließen, entfernen Sie das REM in Zeile 117, damit die parallele Schnittstelle initialisiert wird.

Zeile 119 schaltet mit POKE 2,0 die Codewandlung durch die Utilities ein. Ohne diese POKE-Anweisung würden die Codes völlig unverändert zum Drucker/Interface gesendet.

In Zeile 120 wird COMMODORE.COD,P,R als vollständiger (!) Dateiname der Tabelle COMMODORE.COD der Variablen *N$* zugewiesen. Diese Datei wird anschließend in Zeile 122 durch den Aufruf des Utilities MEMORYLOAD geladen (in den Speicherbereich 1792 bis 1792+255).

Sollten Sie die ASCII-Tabelle benötigen, fügen Sie am Anfang von Zeile 120 einfach ein REM ein und entfernen die REM-Anweisung in Zeile 121.

```
120 REM N$="COMMODORE.COD,P,R":REM COMMODORE-CODES
121 N$="ASCII.COD,P,R":REM ASCII-CODES ???
```

interner Code	Zeichen	gesendeter Code
1	n.d.	1
2	n.d.	2
3	n.d.	3
.....		
.....		
32	Leerzeichen	32
33	!	33
34	"	34
35	#	35
.....		
.....		
56	8	56
57	9	57
58	ö	215
59	ä	214
60	;	59
61	=	61
62	:	58
63	?	63
64	ü	216
65	a	97
66	b	98
67	c	99
68	d	100
.....		

Tabelle COMMODORE.COD

N$ enthält nun den vollständigen Dateinamen der Codetabelle ASCII.COD und bei der Initialisierung wird diese Codetabelle geladen.

COMMODORE.COD enthält zu jedem Zeichen den zugehörigen standardisierten ASCII-Code. Das heißt, für ein »a« oder »b« wird nicht mehr der Commodore-interne Code 65 beziehungsweise 66 gesendet, sondern der »echte« ASCII-Code 97 und 98. Die Utilities übernehmen genau jene Aufgabe, die zuvor Ihr Interface besaß.

Sie kennen nun die Hintergründe der Codewandlung und wissen, welche prinzipiellen Möglichkeiten es gibt, die Codewandlung eines Interfaces auszuschalten. Inzwischen sollte auch Ihr Drucker alle am Bildschirm sichtbaren Zeichen korrekt ausdrucken. Testen Sie das bitte mit einem kleinen Probeausdruck. Laden Sie TOOLS.INIT, passen Sie die Initialisierung wie erläutert auf Ihren Drucker an und speichern Sie es unter

anderen Namen wieder ab. Sie haben nun Ihr persönliches Initialisierungsprogramm. Öffnen Sie nach der Initialisierung (ab Zeile 230) eine Druckerdatei und drucken Sie mal ein wenig.

Schauen Sie sich den Ausdruck bitte genau an, vor allem die Umlaute (in »Düsseldorf« oder »München«). Möglicherweise entdecken Sie, daß einige Zeichen nicht korrekt gedruckt werden. Vielleicht erscheint statt »ü« ein »;« oder gar ein Grafikzeichen.

Dann gehört Ihr Drucker zu den Exoten, deren ASCII-Zeichensatz leider von der Norm abweicht. Die Utilities senden zwar für ein »ü« den korrekten ASCII-Code, Ihr Exote erwartet jedoch einen »individuellen« Code.

Sogar solche Sonderwünsche Ihres Druckers können Sie mit den Utilities erfüllen – indem Sie die Standard-ASCII-Tabelle einfach individuell ändern.

Dazu müssen Sie natürlich wissen, wo sich diese Tabelle befindet. Die Codewandlung übernimmt ein Utility, das davon ausgeht, daß sich im Speicherbereich 1792 bis 1792+256 eine Codetabelle befindet. Genau dorthin lädt das Initialisierungsprogramm nach dem Start eine der beiden vorbereiteten Tabellen COMMODORE.COD beziehungsweise ASCII.COD.

Und zwar befindet sich am Anfang der Tabelle, also in Speicherstelle 1792, der zu sendende Code für das Zeichen mit dem internen ASCII-Code 1. In 1793 befindet sich der zu sendende Code für das zweite, in 1794 der Sendecode des dritten Zeichens und so weiter. Allgemein ausgedrückt: In Speicherstelle

```
1792 + (X-1)
```

befindet sich der zu sendende Code des Zeichens mit dem internen ASCII-Code X.

Ein Beispiel: Laut Commodore-Tabelle hat das Zeichen »?« den Code 63. Das heißt, in der Tabelle befindet es sich in der Speicherstelle 1792 + (63–1), also in Speicherstelle Nummer 1854.

Diese Speicherstelle enthält nach dem Laden einer der beiden Codetabellen den Wert 63 (können Sie mit PRINT PEEK(1854) selbst nachprüfen). Das heißt, für das Zeichen mit dem internen Code 63, das Fragezeichen, wird auch an den Drucker der Code 63 gesendet.

Angenommen, Ihr Exote erwartet für ein Fragezeichen jedoch den Code 200. Dann ändern Sie einfach den entsprechenden Tabelleneintrag mit POKE 1854,200.

Noch einige Beispiele:

1. »ö« besitzt im geänderten Zeichensatz den Code 58. Der zugehörige Sendecode (normalerweise 215) befindet sich in Speicherstelle 1792 + (58–1).

2. »ü« besitzt den internen Code 64. Der für »ü« gesendete Code befindet sich somit in Speicherstelle 1792 + (64–1) und kann mit POKE 1792+(64–1),CODE jederzeit geändert werden.

Diese Änderungen sind natürlich nur dann dauerhaft, wenn Sie die modifizierte Tabelle speichern. Und zwar mit dem Utility MEMORYSAVE:

```
N$="TESTTABELLE,P,W" : SYS SPEICHERN, N$, 1792, 1792+256
```

Dieser Aufruf speichert den Speicherbereich, in dem sich die Tabelle befindet, unter dem Namen TESTTABELLE als Programmdatei. Diese Tabelle können Sie mit

```
N$="TESTTABELLE,P,R" : SYS VLADEN, N$, 1792, 1792+256
```

jederzeit wieder laden. Die einfachste Möglichkeit: Sie ändern in Zeile 120 des Initialisierungsprogramms den Namen der Standard-Codetabellen, die automatisch nach dem Programmstart geladen wird, in TESTTABELLE,P,R.

Zusammenfassung

1. MPS 801, MPS 802 und dazu kompatible Drucker ohne ASCII-Zeichensatz: serielle Datenübertragung; Commodore-Codetabelle (COMMODORE.COD).
2. Drucker mit ASCII-Zeichensatz, die ohne Interface mit Hilfe der im Programm integrierten Centronics-Schnittstelle direkt an den Userport angeschlossen werden: parallele Datenübertragung (PARALLELINIT); ASCII-Codetabelle (ASCII.COD).
3. Drucker mit ASCII-Zeichensatz, die mit Hilfe eines (eingebauten oder externen) Interfaces an die serielle Schnittstelle angeschlossen werden: serielle Datenübertragung; ASCII-Codetabelle (ASCII.COD)

   Zusätzlich muß die Codewandlung des Interfaces abgeschaltet werden. Je nach Interface entweder mit einem DIP-Schalter oder aber durch Veränderung der Sekundäradresse beziehungsweise mit einer ESC-Sequenz.
4. Modifizierung der Codetabelle mit POKE 1792+(X-1),CODE. X ist hierbei der interne Code des betreffenden Zeichens. Dieser Code entspricht bis auf wenige Änderungen durch den neuen Zeichensatz exakt der ASCII-Tabelle in Ihrem C64-Handbuch. Die Ausnahmen:

Zeichen	Code	Zeichen	Code	Zeichen	Code
ö	58	:	62	ß	92
ä	59	ü	64	Ä	93
;	60	Ö	91	Ü	186

5.1 ParallelInit

Centronics-Schnittstelle initialisieren

Adresse 39906

Variablenname PARALLEL

Syntax SYS PARALLEL

Parameter keine

Funktion Initialisiert die integrierte Centronics-Schnittstelle. Nach dem Aufruf von PARALLEL wird für alle Druckausgaben (Gerätenummer zwischen 4 und 7) der Userport verwendet.

5.2 SeriellInit

Serielle Schnittstelle initialisieren

Adresse 39903

Variablenname SRIELL

Syntax SYS SRIELL

Parameter keine

Funktion Initialisiert die serielle Schnittstelle. Nach dem Aufruf von PARALLEL wird für alle Druckausgaben (Gerätenummer zwischen 4 und 7) die serielle Schnittstelle verwendet.

Hinweis Die serielle Ausgabe ist »voreingestellt«. Im Normalfall ist daher keine Initialisierung notwendig, wenn Sie einen Commodore-Drucker (oder einen Fremddrucker mit eingebautem oder dazwischengeschaltetem Interface) an die serielle Schnittstelle anschließen.

Der Aufruf von SERIELLINIT ist daher nur nötig, wenn die Ausgabe über die serielle Schnittstelle erfolgen soll, aber zwischendurch mit PARALLELINIT die parallele Ausgabe eingeschaltet wurde, zum Beispiel beim Betrieb von zwei Druckern (einer an der seriellen Schnittstelle, ein zweiter am Userport angeschlossen).

6

Sonstiges

Unter dieser Rubrik erläutere ich zwei Utilities, die nicht eindeutig einer bestimmten Gruppe zuzuordnen sind.

- DEVICEPRESENT macht genau das, was der Name aussagt: Die Routine prüft, ob ein bestimmtes Gerät betriebsbereit ist. Diese Routine läßt sich gleichermaßen zur Prüfung von Floppy und Drucker einsetzen, wobei beim Drucker egal ist, ob er über die serielle oder über die integrierte Centronics-Schnittstelle angesprochen wird.
- CONVERT ist ein Utility, das Sie zum Datenaustausch mit anderen Programmen benötigen, zum Beispiel mit VIZAWRITE oder DATAMAT. Professionelle Programme besitzen immer Funktionen zum Ex- und Import von Daten fremder Programme. Voraussetzung für diesen Datenaustausch ist allerdings eine Aufbereitung der Zeichencodierung, die das betreffende Programm verwendet. Diese Aufbereitung übernimmt CONVERT.

6.1 DevicePresent

Prüfen, ob Gerät betriebsbereit ist

Adresse 39909

Variablenname PRESENT

Syntax `POKE 144,NR : SYS PRESENT : PRINT PEEK(144)`

Parameter NR: Gerätenummer

Funktion DEVICEPRESENT prüft, ob das Gerät Nummer *NR* betriebsbereit ist. Vor dem Aufruf muß in Speicherstelle 144 die Gerätenummer »gepokt« werden (normalerweise Floppy : 8 und Drucker : 4).

Nach dem Aufruf enthält die Speicherstelle 144 entweder den Wert 0 (Gerät ist betriebsbereit) oder einen Wert ungleich 0 (Gerät ist nicht betriebsbereit).

Beispiel DEVICEPRESENT

Das Demoprogramm fordert Sie auf, eine Taste zu drücken. Anschließend prüft es, ob die Floppy (Gerätenummer 8) betriebsbereit ist.

Zeile 280 »pokt« die Gerätenummer 8 in die Speicherstelle 144 und ruft danach DEVICEPRESENT auf.

Nun wird entweder die Meldung »Floppy ist betriebsbereit« ausgegeben (wenn Speicherstelle 144 nach dem Aufruf den Wert 0 enthält) oder die Meldung »Floppy ist nicht betriebsbereit« (wenn sie einen Wert ungleich 0 enthält).

Danach beginnt das Spielchen von vorn. Dieses Programm können Sie testen, indem Sie die Floppy abwechselnd an- und ausschalten. Sie erhalten immer die jeweils zutreffende Meldung.

6.2 Convert

Dateien konvertieren

Adresse 38915

Variablenname COVERT

Syntax `POKE 2,FLAG : SYS COVERT, LFN1, LFN2`

Parameter

FLAG: 0 = Export / 1 = Import

LFN1: Logische Filenummer der Eingabedatei

LFN2: Logische Filenummer der Ausgabedatei

Funktion CONVERT kopiert eine Datei. Jedes Zeichen der Originaldatei (= Eingabedatei) durchläuft eine Codewandlung, bevor es in die Ausgabedatei kopiert wird.

Für die Codewandlung verantwortlich ist die momentan geladene Codetabelle (siehe Kapitel »Drucker«).

Mit CONVERT kann die Zeichencodierung eines Programms an die von einem anderen Programm verwendete Codierung angepaßt werden.

Mit *FLAG* wird die gewünschte Wandlungsrichtung angegeben. 0 (Export) heißt, daß für jedes Zeichen der Eingabedatei beim Schreiben in die Ausgabedatei der in der Tabelle eingetragene zugehörige Code verwendet wird. Diese Richtung funktioniert analog der unter »Drucker« beschriebenen Codewandlung bei der Datenübertragung zum Drucker. Diesmal werden die ermittelten Zeichencodes jedoch nicht zum Drucker gesendet, sondern in die Ausgabedatei geschrieben.

Hat *FLAG* den Wert 0 (Import), wird gerade umgekehrt in der Tabelle nach dem Code des gelesenen Zeichens gesucht und der zugehörige interne Zeichencode eingesetzt, den die Utilities verwenden.

Beispiel Auf der Diskette befindet sich eine bereits fertig vorbereitete Codetabelle VIZA.COD für den Datenaustausch zwischen einem mit den Utilities geschriebenen Programm und VIZAWRITE.

Angenommen, Sie schreiben ein Programm, das in einer sequentiellen Datei Adressen speichert, zum Beispiel den Namen »Müller«. VIZAWRITE verwendet für Umlaute eine andere Zeichencodierung als die Utilities, zum Beispiel statt 64 den Code 184 für das Zeichen »ü«.

Wenn Sie Ihre Adreßdatei in VIZAWRITE einlesen, wird daher aus »Müller« ein »M@ller«, da VIZWARITE den Code 64 als »@« interpretiert.

Die vorbereitete Tabelle VIZA.COD enthält an Position Nummer 64 (entspricht laut interner Codierung der Utilities dem Zeichen »ü«) den Wert 184, also jenen Code, den VIZAWRITE für dieses Zeichen erwartet.

Wird Ihre Adreßdatei als Eingabedatei verwendet, um – unter Verwendung dieser Codetabelle – eine Ausgabedatei zu erzeugen, schreibt CONVERT für ein aus Ihrer Adreßdatei gelesenes Zeichen mit dem Code 64 den Code 184 in die Ausgabedatei.

Mit folgendem Programm könnte eine sequentielle Datei namens ADRESSEN für ein Einlesen in VIZAWRITE konvertiert werden:

```
230 N$="VIZA.COD" : SYS VLADEN, N$, 1792,
1792+256
240 OPEN 1,8,2,"ADRESSEN,S,R" : REM EINGABEDATEI
250 OPEN 2,8,2,"KOPIE,S,W"    : REM AUSGABEDATEI
260 POKE 2,0 : SYS COVERT, 1, 2
270 CLOSE 2: CLOSE 1
```

In Zeile 230 wird mit MEMORYLOAD die benötigte Codetabelle VIZA.COD geladen. Anschließend wird die bereits vorhandene sequentielle Adreßdatei ADRESSEN geöffnet (Zeile 240).

Danach wird die zu erzeugende Ausgabedatei geöffnet, die den Namen KOPIE erhält (Zeile 250).

POKE 2,0 gibt CONVERT an, daß die Codetabelle benutzt werden soll, um Daten zu exportieren (die Utility-Codes werden in die zugehörigen VIZAWRITE-Codes gewandelt).

Nach dem Aufruf werden beide Dateien wieder geschlossen (Zeile 270) – die Konvertierung ist beendet.

Auf der Diskette befindet sich nun eine zweite sequentielle Datei KOPIE, die bis auf die konvertierten Zeichencodes exakt den Inhalt Ihrer Adreßdatei enthält und problemlos in VIZAWRITE eingelesen werden kann.

Auch die umgekehrte Richtung ist möglich: Der Import von Daten, also das Konvertieren der Codes, die ein Fremdprogramm verwendet, in die von den Utilities benutzten Codes.

Nehmen wir wieder das Beispiel VIZAWRITE und den Umlaut »ü«. Wie erläutert, verwendet VIZAWRITE für ein »ü« den Code 184, die Utilities jedoch den Code 64.

Beim Datenimport (FLAG = 1) verläuft die Wandlung in umgekehrter Richtung. Die Eingabedatei ist nun eine von VIZAWRITE erzeugte Datei, die Sie in Ihr Basic-Programm einlesen wollen. Gehen wir davon aus, daß die von VIZAWRITE erzeugte Datei ebenfalls ADRESSEN heißt und die konvertierte Datei wieder den Namen KOPIE erhält.

```
230 N$="VIZA.COD" : SYS VLADEN, N$, 1792,
1792+256
240 OPEN 1,8,2,"ADRESSEN,S,R" : REM EINGABEDATEI
250 OPEN 2,8,2,"KOPIE,S,W"    : REM AUSGABEDATEI
260 POKE 2,1 : SYS COVERT, 1, 2
270 CLOSE 2: CLOSE 1
```

Das Programm ist mit der vorigen Version identisch – bis auf Zeile 260. Als Parameter *FLAG* wird nun in die Speicherstelle 2 statt einer 0 der Wert 1 »gepokt«.

Wenn CONVERT aus der Eingabedatei den Code 184 liest (den VIZAWRITE-Code für »ü«), sucht es diesen Code in der Tabelle und schreibt den zugehörigen internen Utility-Code 64 in die Ausgabedatei KOPIE.

Hinweis Die vorbereitete Konvertierungstabelle VIZA.COD entspricht bis auf wenige Zeichen exakt der Codetabelle COMMODORE.COD. Die Unterschiede betreffen ausschließlich die von VIZAWRITE anders codierten Zeichen:

Zeichen	Code	Zeichen	Code
"	98	Ä	185
ä	165	Ö	186
ö	182	Ü	187
ü	184	ß	188

Hinweis Vergessen Sie nicht, nach beendeter Konvertierung wieder die für Ihren Drucker benötigte Codetabelle zu laden, die durch die Konvertierungstabelle überschrieben wurde (sonst erhalten Sie wahrscheinlich recht »lustige« Ausdrucke).

Anhang A

TOOLS.INIT im Detail

Es ist empfehlenswert, immer das vorgegebene Initialisierungsprogramm TOOLS.INIT zu verwenden. Wenn Sie unbedingt wollen, können Sie jedoch auch Ihr eigenes Initialisierungsprogramm schreiben. Dann sollten Sie jedoch die genaue Vorgehensweise von TOOLS.INIT kennen.

```
10 REM ** MC-ROUTINEN NACHLADEN **
11 IF PEEK(56)=149 THEN 26
12 F=PEEK(187)+256*PEEK(188):REM ADRESSE DES AKTUELLEN
FILENAMENS
14 N$="":FOR I=0 TO PEEK(183)-1:N$=N$+CHR$(PEEK(F+I)):NEXT: REM
N$ = FILENAME
15 IF N$="MC1.OBJ" THEN POKE 55,0:POKE 56,149:CLR:LOAD"MC2.
OBJ" ,8,1
16 LOAD"MC1.OBJ",8,1
```

Dieser erste Teil hat die Aufgabe, die Maschinenprogramme MC1.OBJ und MC2.OBJ nach dem ersten Programmstart nachzuladen und das Ende des für Basic verfügbaren Speicherbereichs herabzusetzen. Das Herabsetzen ist nötig, da sich MC1.OBJ am Ende des Basic-Speichers befindet und ansonsten beim Anlegen von Strings durch die dort gespeicherten Zeichenketten überschrieben würde.

Daher wird nach dem Nachladen in 55 (Low-Byte des Basic-Endes) der Wert 0 und in 56 (High-Byte des Basic-Endes) der 149 gepokt (Zeile 15). Mit CLR wird der Interpreter veranlaßt, verschiedene Zeiger entsprechend diesem neuen Wert zu korrigieren. Statt bei $A000 endet der für Basic verfügbare Speicher nun bei $9500. Die Utilities verkleinern den Basic-Speicher somit um 11 »Pages« oder knapp 3 Kbyte.

Kommen wir zum Nachladen. Nach jeder LOAD-Anweisung innerhalb eines Programms führt der Basic-Interpreter einen Neustart des Programms mit RUN aus. Dies

ist beim Nachladen eines Basic-Programms sicher sinnvoll, damit das neue Programm automatisch gestartet wird. Jedoch nicht beim Nachladen eines Maschinenprogramms, da nach dem Neustart wieder die LOAD-Anweisung ausgeführt und das Maschinenprogramm ein zweites Mal geladen wird und so weiter. Das heißt, fatalerweise bilden wir mit

```
100 LOAD"MC1.OBJ",8,1
110 LOAD"MC2.OBJ",8,1
```

eine Endlosschleife! Die übliche richtige Methode zum Nachladen geht so:

```
IF A=0 THEN A=1 : LOAD"MC1.OBJ"
IF A=1 THEN A=2 : LOAD"MC1.OBJ"
```

Diese Standardtechnik hat leider einen gewaltigen Nachteil: sie funktioniert nicht mehr, wenn Sie Ihr Programm später einmal kompilieren!

Daher meine weitaus umständlichere Methode: In Speicherstelle 56 ist das High-Byte des Basic-Endes enthalten, normalerweise also 160 (Basic-Ende: $A000). Nach Ausführung von Zeile 18 (Laden von MC2.OBJ und Herabsetzen des Basic-Endes) enthält 56 stattdessen wie erläutert, den Wert 149.

Dieser Wert wird am Programmanfang überprüft. Enthält 56 den Wert 149, wurde das Nachladen und Herabsetzen bereits durchgeführt – das Programm übergeht diesen Teil und verzweigt sofort zu Zeile 26, dem zweiten Teil der Initialisierung.

Die Zeilen 12 bis 16 sind ein wenig komplex. Nach dem Laden eines Programms enthalten die Speicherstellen 187 und 188 einen Zeiger auf den aktuellen Filenamen, also den Namen der zuletzt geladenen Datei. Zeile 12 ermittelt den dezimalen Wert dieses Zeigers.

In Zeile 14 wird der ab der Adresse <F> im ASCII-Code vorhandene Filename ermittelt. In Speicherstelle 183 befindet sich die Länge des Filenamens. Anhand der Länge und der Adresse wird Zeichen für Zeichen gelesen und in *N$* gespeichert. *N$* enthält nach Verlassen der Schleife den Namen der zuletzt geladenen Datei.

Wurde zuletzt das erste Maschinenprogramm MC1.OBJ geladen, wird der Basic-Speicher herabgesetzt und MC2.OBJ geladen (Zeile 15). Ansonsten wird der erste Teil geladen, MC1.OBJ.

Um diesen Programmteil zu verstehen, müssen Sie unbedingt beachten, daß nach jeder LOAD-Anweisung automatisch RUN ausgeführt wird, ein Programm-Neustart!

Die Zeilen 26 bis 96 kennen Sie. Dieser Programmteil bringt ein wenig Komfort in die Benutzung der Utilities. Alle wichtigen Adressen werden in Variablen gespeichert.

Interessanter wird es im folgenden Abschnitt. SYS FSCREEN initialisiert die schnelle Bildschirmausgabe, SYS MAKRO die Aufzeichnung und Verwendung von Tastatur-

Makros, SYS CHAR den deutschen Zeichensatz und SYS PINIT die Puffertabelle der Windowing-Routinen.

Übrigens: Ohne SYS FSCREEN ist nicht nur die Bildschirmausgabe langsamer. Zusätzlich wird keine Codewandlung vorgenommen!

Die Utilities können Sie nur nach Initialisierung des deutschen Zeichensatzes verwenden. Da nur der Groß-/Klein-Zeichensatz kopiert wird, ist auch die Umschaltung in diesen Modus mit PRINT CHR$(14) und die Verriegelung mit CHR$(8) zwingend.

Das Entfernen von SYS FSCREEN, SYS MAKRO oder SYS PINIT führt zu Teileinschränkungen bei der Benutzung der Utilities. Anschließend können Sie einzelne Tools nicht mehr verwenden!

SYS CHAR und PRINT CHR$(14) CHR$(8) dagegen sind die zwingendsten Initialisierungen überhaupt! Ohne diese Initialisierungen können Sie die Utilities überhaupt nicht verwenden!!!

Der letzte Teil wurde im Abschnitt »Drucker« eingehend besprochen. Beachten Sie bitte Zeile 119.

```
119 POKE 2,0 : REM WANDLUNG INITIAL. !!!
```

Eine Codewandlung anhand der Codetabelle findet nur statt, wenn Speicherstelle 2 einen Wert zwischen 0 und 127 enthält. Sie können also jederzeit mit POKE 2,128 die Codewandlung ab- und mit POKE 2,0 wieder einschalten (obwohl ich mir keinen Grund vorstellen kann, die Codewandlung abzuschalten).

Anhang B

Geänderter Zeichensatz und ASCII-Codes

Die folgende Tabelle enthält alle Änderungen der Tastaturbelegung und der Zeichencodes, die durch den deutschen Zeichensatz hervorgerufen werden. Alle übrigen Zeichencodes entsprechen unverändert der ASCII-Tabelle im Handbuch zum C64.

Alte Belegung	Neue Belegung	Alte Belegung	Neue Belegung
:	ö	SHIFT + :	Ö
;	ä	SHIFT + ;	Ä
@	ü	SHIFT + @	Ü
£	ß	SHIFT + .	:
		SHIFT + ,	;

Anhang C

Speicherbelegung und die Einbindung von Assembler-Routinen

(Erweiterte) Zeropage:	nahezu alles belegt
$0400-$05FF:	QuickSort-Stacks (temporär)
$0600-$06FF:	Makro-Speicher
$0700-$07FF:	Codetabelle
$0800-$94FF:	Basic-Programm, -Variablen
$9500-$9FFF:	Maschinenprogramme
$A000-$BFFF:	Schnelle Garbage Collection (temporär)
$C000-$C3FF:	Bildschirmspeicher
$C400-$CFFF:	Maschinenprogramme
$E000-$EFFF:	Schnelle Garbage Collection (temporär)
$F000-$F7FF:	Window-Stacks (temporär)
$F800-$FFFF:	Zeichensatz

Es gibt eine einfache Möglichkeit, Platz für Ihre eigenen Assembler-Routinen zu schaffen: setzen Sie ebenso wie die Utilities selbst den Basic-Speicher herunter.

Die Utilities setzen das Basic-Ende auf $9500, und zwar in Zeile 15 des Initialisierungsprogramms. Ersetzen Sie zum Beispiel die Anweisung POKE 56,149 durch POKE 45,144 (Basic-Ende $9000), wenn Ihnen fünf zusätzliche Pages genügen.

Wichtig: Wenn Sie den Zeiger auf das Basic-Ende manipulieren, müssen Sie natürlich auch die Abfrage dieses Zeigers in Zeile 11 korrigieren! Im Beispiel ersetzen Sie IF PEEK(56)=149 ... durch IF PEEK(56)=144.

Schwieriger ist es, die erweiterte Zeropage zur Datenablage oder gar die Zeropage für Pointer zu verwenden.

Die Routinen benötigen sehr viele Bereiche in der (erweiterten) Zeropage. Und zwar bei weitem nicht nur die »offiziellen« Bereiche wie $FB-$FE oder $02A7-$02FF. Die Bereiche werden teilweise nur temporär genutzt (zum Beispiel FAC und ARG während des Ablaufs von QUICKSORT), teilweise auch permanent.

Das macht die Erweiterung eines Basic-Programms um eigene Assembler-Routinen natürlich nicht ganz unproblematisch. Uneingeschränkt verwenden können Sie eigentlich nur den Stack. Für »Amateure«: Das ist kein Witz!; der Stack ist bei geeigneter Verwendung zur Datenablage tatsächlich ebenso gut geeignet wie jeder andere Speicherbereich.

Und vor allem: Der Speicher von FAC und ARG ($61-$70) wird nur temporär von QUICKSORT benutzt. Das heißt, Ihnen steht ein bei weitem ausreichender Bereich zur Ablage von Pointern in der Zeropage zur Verfügung.

Wenn Sie unbedingt weitere Bereiche der Zeropage benötigen, beginnen Sie Ihr Programm so:

```
      LDX ANZAHL
LOOP  LDA START,X
      PHA
      DEX
      BPL LOOP
```

und beenden Sie es mit:

```
      LDX ANZAHL
LOOP  PLA
      STA START,X
      DEX
      BPL LOOP
```

Das heißt, kopieren Sie den Inhalt des gewünschten Bereichs einfach in den Stack und stellen Sie vor dem Beenden Ihrer Routine den Ausgangszustand wieder her. Diese Methode ist natürlich mit Vorsicht zu genießen, wenn Sie »riesige« Bereiche benötigen (Stack-Überlauf).

Dann verwenden Sie statt dem Stack einfach einen beliebigen Speicherbereich (STA ADRESSE,X statt PHA und LDA ADRESSE,X statt PLA).

Anhang D

Manipulierte Vektoren

Einige nette Überraschungen können Sie erleben, wenn Sie tiefergehende Assembler-Routinen schreiben, die zum Beispiel den Interrupt- ($0314/$0315) oder den BSOUT-Vektor ($0326/$0327) manipulieren. Diese Vektoren sind nämlich bereits verbogen!

Wenn Sie beispielsweise mit

```
LDA #<(ADRESSE)
STA $Ø314
LDA #>(ADRESSE)
STA $Ø315
```

den Interrupt-Vektor verbiegen, ist die MAKRO-Routine lahmgelegt! Oder Sie verbiegen rücksichtslos den BSOUT-Vektor. Dann haben Sie gleich mehrere Routinen ausgeschaltet. FASTSCREEN, auf die dieser Vektor nach dem Aufruf SYS FSCREEN zeigt, und gleichzeitig die Codewandlung, da FSCREEN (sehr unschön, ich weiß) direkt zur Codewandlungs-Routine »jumpt«.

Also programmieren Sie im Gegensatz zu mir »Schlamper« bitte anständig. Das heißt, retten Sie den aktuellen Vektor und beenden Sie Ihre Routine mit einem indirekten Sprung über den geretteten Vektor.

```
Vektor retten
LDA $Ø314
STA KOPIE
LDA $Ø315
STA KOPIE+1
```

```
Vektor verbiegen
LDA #<(ADRESSE)
STA $0314
LDA #>(ADRESSE)
STA $0315
Hauptprogramm
.....
.....
.....
Sprung über Originalvektor
JMP (KOPIE)
```

Und noch eine Kleinigkeit, die nur in sehr außergewöhnlichen Spezialprogrammen Bedeutung erlangen kann: Die MAKRO-Routine klinkt sich über einen verbogenen IRQ-Vektor in den System-Interrupt ein.

Diese Routine filtert – um Tastatur-Makros zu ermöglichen – gewisse Zeichen aus. Die Tastaturabfrage befindet sich am Ende der Interrupt-Routine (JSR $EA87). Nehmen wir an, Sie binden eine eigene Routine, wie beschrieben, »sauber« in den System-Interrupt ein. Sie verbiegen also den IRQ-Vektor auf Ihre eigene Routine und springen am Ende Ihrer Routine weiter zur MAKRO-Routine.

Da Ihre Routine zuerst kommt, erwarten Sie sicher, daß Sie eine am Ende des letzten Interrupts ermittelte gedrückte Taste »ungefiltert« erhalten, noch bevor die MAKRO-Routine in Aktion tritt.

Allerdings ist diese Annahme falsch! Mit »Hängen und Würgen« (Stackmanipulationen, die die RTI-Rücksprungadresse manipulieren) schafft es die MAKRO-Routine, sich nicht vor, sondern hinter den IRQ-Service-Routinen einzuklinken!

Das heißt, unmittelbar nach dem Aufruf der Tastaturabfrage am Ende der IRQ-Routine ist MAKRO dran – noch bevor Sie zum Beginn des nächsten Interrupts kommen.

Ihre Routine wird daher an die Resultate der Tastaturabfrage im Interrupt ohne vorhergehende Filterung durch MAKRO herankommen. In der Praxis heißt das: [CTRL]-Kombinationen bekommt Ihr Programm niemals »zu sehen«. Also versuchen Sie bitte in Ihrer Interrupt-Routine erst gar nicht, irgendwelche [CTRL]-Kombinationen abzufragen.

Das Gleiche gilt für die Tastenkombinationen [C=] + [↵] und [C=] + [Leertaste], an denen MAKRO bekanntlich ebenfalls interessiert ist.

Ein schwerwiegender Nachteil ist das sicher nicht. Wenn Sie anderer Meinung sind, steht es Ihnen frei, MAKRO (den Source-Text haben Sie ja) zu ändern, in Ihrer Routine selbst die Tastaturabfrage aufzurufen oder gar ebenfalls mit Stackmanipulationen herumzuspielen, um Ihre Routine am Interrupt-Ende einzuklinken.

Ich wünsche Ihnen dabei viel Spaß und zahlreiche Abstürze (ein geeigneter Gruß für Assembler-Programmierer, finden Sie nicht?).

Bücher zum Commodore 64/128

S. Vilsmeier
3D-Konstruktion mit GIGA-CAD Plus auf dem C64/C128
1986, 370 Seiten, inkl. 2 Disk.
Mit GIGA-CAD können Computergrafiken von besonderer Räumlichkeit und Faszination geschaffen werden. GIGA-CAD Plus ist schneller und einfacher zu bedienen, die Benutzeroberfläche wurde verbessert und der Befehlssatz erweitert. Die Eingabe erfolgt in erster Linie über den Joystick. Hardware-Anforderung: C64 mit Floppy 1541 oder C128 (im 64'er-Modus), Fernseher oder Monitor, Joystick und Commodore- oder Epson-kompatibler Drucker.
● Das verbesserte GIGA-CAD-Programm mit neuen Features wie erweitertem Befehlssatz und bis zu 10mal schneller liegt dem Buch im Floppy-1541-Format bei.
Best.-Nr. 90409
ISBN 3-89090-409-2
DM 49,–
(sFr 45,10/öS 382,20)

H. Haberl
Mini-CAD mit Hi-Eddi plus auf dem C64/C128
1986, 230 Seiten, inkl. Diskette
Auf der beiliegenden Diskette findet der Leser das vollständige Zeichenprogramm »Hi-Eddi«, mit dem das komfortable Erstellen von technischen Zeichnungen, Plänen oder Diagrammen ebenso möglich ist wie das Malen von farbigen Bildern, Entwurf und Ausdruck von Glückwunschkarten, Schildern, ja sogar von bewegten Sequenzen (kleine Trickfilme, Schaufenster-Werbung).
● Wer sagt, daß CAD auf dem C64 nicht möglich ist?!
Best.-Nr. 90136
ISBN 3-89090-136-0
DM 48,–
(sFr 44,20/öS 374,40)

B. Bornemann-Jeske
Vizawrite-Buch für den C64/C128
1987, 228 Seiten
Mit dem »Vizawrite-Buch« liegt erstmals ein vollständiges und detailliertes Arbeitsbuch für den Anfänger und den professionellen Anwender zur Textverarbeitung auf dem C64/C128 vor. Die Grundlagenkapitel führen Sie anhand kurzer Übungsaufgaben in die elementaren Funktionen des Systems ein. Das Kapitel für Fortgeschrittene zeigt Ihnen jede Programmfunktion im Detail. Zahlreiche praktische Tips aus verschiedenen Anwendungsbereichen ermöglichen Ihnen die optimale Nutzung Ihres Textverarbeitungssystems.
Best.-Nr. 90231
ISBN 3-89090-231-6
DM 49,–
(sFr 45,10/öS 382,20)

O. Hartwig
Experimente zur Künstlichen Intelligenz mit C64/C128
1987, 248 Seiten
Sind Maschinen intelligent? Können Computer denken? Erschließen Sie sich eines der interessantesten Gebiete der modernen Computerforschung! Anhand zahlreicher Programme erfahren Sie hier die Möglichkeiten der Künstlichen Intelligenz, speziell auf dem C64 und dem C128. Der Schwerpunkt des Buches liegt auf der Praxis. Alle KI-Techniken werden durch anschauliche Programme vorgestellt, die sofort nachvollziehbar sind. Zusätzlich erhalten Sie jede Menge Anregungen zu eigenen Experimenten. Die KI-Programme können ohne weiteres in eigene Programme integriert werden.
Best.-Nr. 90472
ISBN 3-89090-472-6
DM 49,–
(sFr 45,10/öS 382,20)

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

706358

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

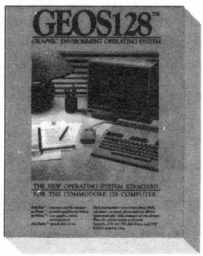
Eine neue Welt für C64/128:

GEOS

GEOS für den C128 (deutsch)

Der neue Betriebssystemstandard – in der deutschen Originalversion für den C128. GEOS 64 wurde an den 128er-Modus des C128 angepaßt und kann sowohl die doppelte Auflösung als auch den größeren Speicher nutzen. Unterstützt werden am RGB-Eingang angeschlossene Monitore (80 Zeichen), sowie die üblichen PAL-Monitore und Fernsehapparate. Ansonsten gelten die Leistungsmerkmale von GEOS 64.

Hardware-Anforderung:
C128, Floppy 1541, 1570 oder 1571, Joystick oder Maus 1351.
5¼-Zoll-Diskette
Bestell-Nr. 50327 **DM 119,–***



Deskpack 1/GeoDex für den C64/C128 (deutsch)

Deskpack 1/GeoDex: die nützlichen Zusatzprogramme für GEOS Graphics-Grabber! Überträgt Grafiken von Print Shop, Print Master und Newsroom zur Anwendung mit GeoPaint und GeoWrite.
Leistungsumfang: Icon Editor – erstellt und verändert Icons nach Ihren Vorstellungen. GeoDex – Adreß- und Notizbuch mit Modemunterstützung. GeoMerge – Suchen nach Adreßgruppen aus GeoDex sowie Erstellen von Formbriefen und Listen. Blackjack – das klassische Glücksspiel. Kalender.
Hardware-Anforderungen:
C64 oder C128, Floppy 1541, 1570 oder 1571, Joystick.
Software-Anforderung: GEOS 64.
Bestell-Nr. 50322 **DM 69,–***

GeoWrite Workshop für den C64/C128

Bestell-Nr. 50323 **DM 89,–***

GeoFile für den C64/C128

Bestell-Nr. 50324 **DM 89,–***

GeoCalc für den C64/C128

Bestell-Nr. 50325 **DM 89,–***

* Unverbindliche Preisempfehlung

In Vorbereitung:

GeoWrite Workshop 128

Bestell-Nr. 50329 ca. **DM 119,–***

GeoFile 128

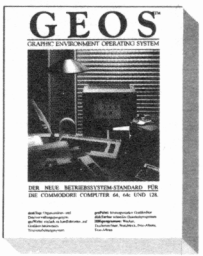
Bestell-Nr. 50330 ca. **DM 119,–***

GeoCalc 128

Bestell-Nr. 50331 ca. **DM 119,–***

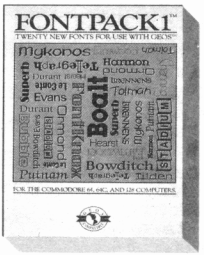
GEOS, Version 1.3, für den C64/C128 (deutsch)

Der neue Betriebssystemstandard für Commodore 64. Leistungsumfang: Desk-Top – das Grafikinterface zum GEOS-Betriebssystem. Schauen Sie sich die Dateien als Icons oder im Textmodus an. Automatisches Sortieren von Dateien nach Alphabet, Größe, Typ oder Datum der letzten Änderung ist kein Problem. Dateien kopieren, löschen und Disketten formatieren ist natürlich enthalten.



GeoPaint: ein umfangreiches Zeichenprogramm in Farbe mit 14 verschiedenen Grafiktools, 32 Pinselstärken, 32 verschiedenen Mustern. GeoWrite: ein einfaches, leichtbedienbares Textprogramm. Desk-Accessories: Wecker, Notizblock, Taschenrechner.
Hardware-Anforderungen:
C64 oder C128 (64er-Modus), Floppy 1541, 1570 oder 1571, Joystick.
Bestell-Nr. 50320 **DM 59,–***

Update von älteren englischen Versionen auf die neue deutsche Version 1.3. Erhältlich direkt beim Markt&Technik-Buchverlag gegen Einsendung des Originalprodukts und gegen Vorauskasse.
Bestell-Nr. 50320U **DM 39,–***



Ergänzende Literatur:

Alles über GEOS 1.3

1987, 576 Seiten
»Alles über GEOS V1.3« informiert umfassend über diese deutschsprachige, grafische Benutzeroberfläche für den Commodore 64/128. Vom Einstieg bis zur Programmierung können Sie auf dieses ausführliche Nachschlagewerk zurückgreifen.
Bestell-Nr. 90570
ISBN 3-89090-570-6
(sFr 54,30/öS 460,20) **DM 59,–**

Fontpack 1 für den C64/C128 (deutsch)

Die unentbehrliche Utility für GEOS-Benutzer! Fontpack 1 wurde für die GEOS-Applikationen GeoPaint und GeoWrite entwickelt und enthält 20 neue, außergewöhnliche Schriftarten, die jeden Anwender begeistern werden.
Hardware-Anforderungen:
C64 oder C128, Floppy 1541, 1570 oder 1571, Joystick.
Software-Anforderungen: GEOS 64
Bestell-Nr. 50321 **DM 49,–***

Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

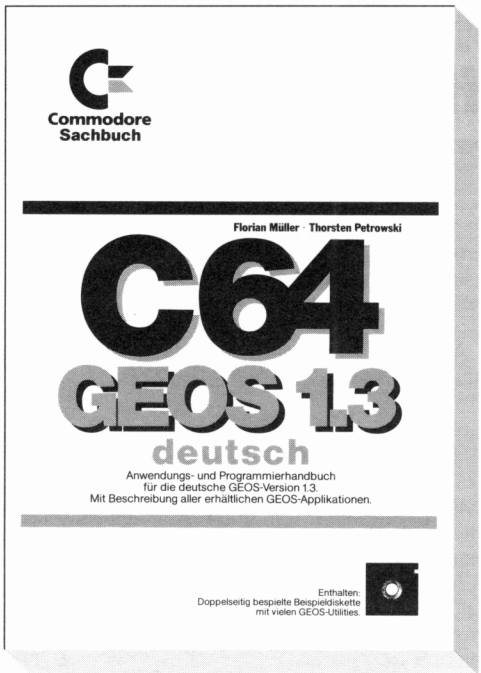
802241

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

EXKLUSIV bei Markt&Technik

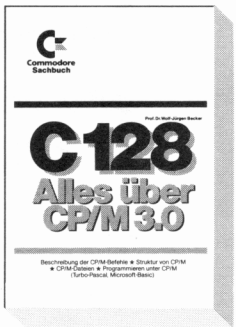
Commodore-Sachbücher

Commodore Sachbuchreihe
Alles über den C64
2. Auflage 1986, 514 Seiten
Dieses umfangreiche Grundlagenbuch zum C64 enthält neben einem Basic-Lexikon alle Informationen und Tips, die der Spezialist zur Grafik- und Musikprogrammierung benötigt. Ein Kapitel beschäftigt sich mit der Programmierung in Maschinensprache und der Einbindung von Maschinensprache-Routinen in Basic-Programme. In diesem Zusammenhang erfahren Sie auch alles über einen wichtigen Bestandteil des Betriebssystems aller Commodore-Computer, das »Kernal«.
Bestell-Nr. 90379
ISBN 3-89090-379-7
DM 59,–
(sFr 54,30/öS 460,20)



F. Müller/T. Petrowski
Alles über GEOS Version 1.3 Anwendungs-, Programmier- und Systemhandbuch
1987, 532 Seiten, inklusive Diskette

Das umfassende Buch über Anwendung und Programmierung der grafischen Benutzeroberfläche GEOS.
Bestell-Nr. 90570,
ISBN 3-89090-570-6
DM 49,–
(sFr 45,10/öS 382,20)



Prof. Dr. W.-J. Becker
C128 – Alles über CP/M 3.0
1986, 299 Seiten
Eine fundierte Einführung in die Anwendung des Betriebssystems CP/M 3.0 bzw. CP/M Plus auf dem Commodore 128.
Bestell-Nr. 90370
ISBN 3-89090-370-3
DM 52,–
(sFr 47,80/öS 405,60)

Dr. Ruprecht
C128-ROM-Listing
1986, 456 Seiten
Dieses kommentierte ROM-Listing umfaßt das Betriebssystem des C128, den Monitor des C128 sowie das Basic 7.0 von Microsoft.
Bestell-Nr. 90212
ISBN 3-89090-212-X
DM 58,–
(sFr 53,40/öS 452,40)

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

711356

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

C64/C128 Profi-Tools

Basic ist die Standardsprache Ihres C64 bzw. C128, leicht zu erlernen und zu verstehen, aber wer hat sich nicht schon über mangelnde Geschwindigkeit bei dieser Sprache beklagt.
Die Profi-Tools setzen den Hebel genau an dieser Stelle an. Mit der umfangreichen Sammlung von professionellen Assembler-Routinen eröffnet sich für Sie die Möglichkeit, aus Basic-Programmen oder mit Basic-Kenntnissen schnelle Programme zu schreiben. Sie binden einfach die fertigen Routinen in Ihre Werke ein und müssen sich nicht extra mit Assembler beschäftigen. Zusätzlich erfahren Sie aus der ausführlichen Beschreibung die Funktion, das Einsatzgebiet und die Stärken der einzelnen Routinen.
Zum Beispiel führt der Aufruf **SYS GC** eine Garbage Collection unter einer Sekunde durch. **SYS QUICK,A$(1),A$(1000)** sortiert ein Array mit 1000 Strings in etwa einer halben Sekunde!
Mit **SYS CHAR** schalten Sie den deutschen Zeichensatz ein, der sich auch problemlos ausdrucken läßt, und über **SYS PARALLEL** aktivieren Sie die integrierte Centronics-Schnittstelle. Die beiliegende Diskette enthält 26 Werkzeuge dieser Leistungsklasse.
Hier noch einige Beispiele:

- CONTROLMENUE: Verwaltung von Pull-down-Menüs
- WINDOWING: Einzelne oder überlappende Windows
- DIRECTORY: Directory in Stringarray einlesen
- QUICKSORT: Extrem schnelle und flexible Sortierroutine
- FASTSAVE: Blitzschnelles Speichern kompletter Arrays
- FASTLOAD: Ebenso schnelles Laden kompletter Arrays
- MAKROSINIT: Definition von Tastatur-Makros.

Die Begleitdiskette:
Sie enthält alle 26 beschriebenen Profi-Tools mit ausführlich kommentierten Source-Texten.
Hardware-Anforderungen:
C64 bzw. C128 im 64'er-Modus, Floppy 1541/1570/1571.
Software-Anforderungen:
gegebenenfalls Assemblersystem (z.B. Hypra-Ass) zum Ändern der Routinen.

ISBN 3-89090-617-6

DM 49,-
sFr 45,10,
öS 382,20,
Unverbindliche Preisempfehlung

Markt&Technik