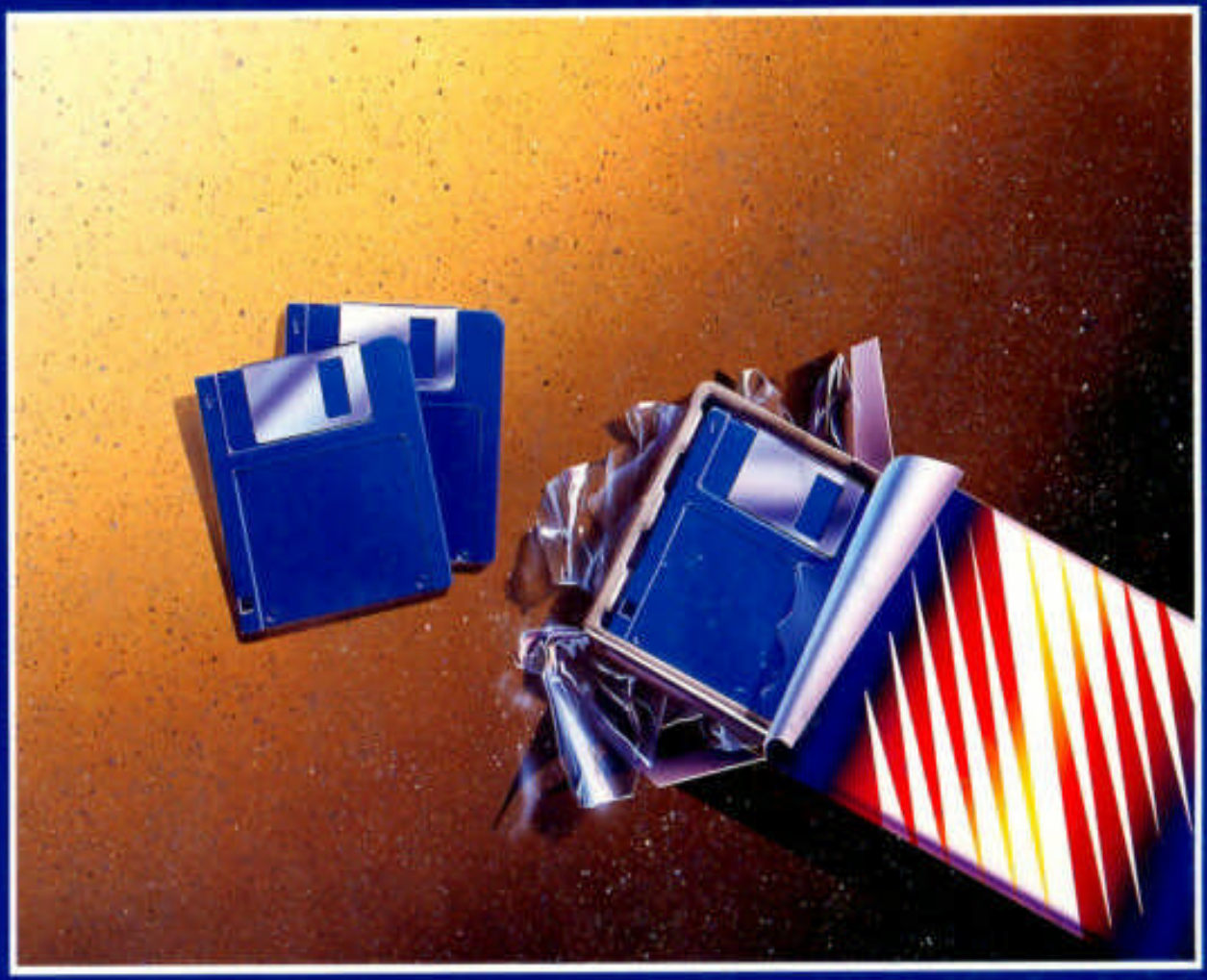
Günter Born

PowerBASIC

Programmiertechniken



kirschbaum
soft ware

Günter Born

PowerBasic-Programmierhandbuch

Günter Born

Satz: Günter Born

Herstellung: Günter Born

Zum Buchkonzept

Eine *saustarke* Toolbox

Sie besitzen Power BASIC und suchen Tools zur Unterstützung dieser Sprache? Sie programmieren in Power BASIC und benötigen Hilfestellung bei verschiedenen Problemen. Dann halten Sie genau das richtige in Ihren Händen, eine *saustarke* Sache. Hier wird Ihnen nicht zu x-ten mal der Sprachumfang von Power BASIC erklärt, sondern praxisnahe Utilities an die Hand gegeben.

Werkzeuge zur Unterstützung der Programmentwicklung (wie z.B: Cross-Referenz-Generatoren, Programmformatierer, Druckerspooler, etc.) bilden unverzichtbare Hilfsmittel für jeden ernsthaften Power BASIC-Anwender. Sie tragen zur Reduzierung der Entwicklungszeit bei und unterstützen einen transparenten Programmierstil. Weitere Beispiel widmen sich der Implementierung von Textbearbeitungtools wie die UNIX-Utilities WC, CUT und PASTE. Treiber für die PostScript-Ausgabe unter DOS, oder die Sie Befehlserweiterungen für Batchprogramme (ASK, ESC, FKEY, GET, WAIT, XCALC) ermöglichen gänzliche neue Möglichkeiten für das Betriebssystem. Bibliotheken für die Maus oder Bildschirmsteuerung sind weitere behandelte Themen.

Alle Beispiele werden schrittweise von den Anforderungen bis hin zur lauffähigen Lösung entwickelt. Die Programme sind ausführlich kommentiert und liegen im Quellcode auf Diskette bei. Die Toolbox stellt damit ein unverzichtbares Hilfsmittel für jeden ernsthaften Power BASIC-Anwender dar.

Sie gehören noch zu den Einsteigern und trauen sich noch nicht an die Programmierung heran? Dann sollten Sie zumindest die fertigen Programme nutzen. Was halten Sie von einem fertigen PostScript-Treiber oder dem erweiterten DOS-Befehlssatz? Anschließend steigen Sie mit Hilfe des Buches schrittweise in die Programmierung ein.

Sie sind bereits fortgeschritten und suchen gebrauchsfertige Bibliotheken und Beispielprogramme? Hier sind Sie genau richtig: die Bibliotheken zur Maussteuerung oder zur Erstellung von Pop-Up- und Pull-Down-Menüs verleihen Ihren Programmen ein professionelles Outfit und erleichtern den Umgang mit Power BASIC.

Inhaltsverzeichnis

Vorwort

Softwareentwicklung ist die Kunst, auf systematische Weise Algorithmen zu formulieren und zu Programmen zu kombinieren. Dabei kommt einer strukturierten und planvollen Vorgehensweise eine erhebliche Bedeutung zu. Ausgehend von einer Anforderung ist Schritt für Schritt der Weg zur fertigen Lösung zu beschreiten.

Durch den breiten Einsatz der Personalcomputer befassen sich immer mehr Computeranwender mit der Entwicklung von Programmen. Nicht selten gelangt dabei PowerBASIC als Programmiersprache zum Einsatz. Dieser Compiler bietet eine gute Basis, um auch anspruchsvollere Lösungen zu erstellen. Leider findet der Einsteiger und fortgeschrittene Anwender nur wenig Literatur über die Programmentwicklung in Power BASIC. Sobald tiefergehende Problemstellungen auftreten, findet sich überhaupt nichts.

Im Herbst 1991 kam bei Diskussionen mit Georg Weiherer die Idee auf, ein Buch zu Power BASIC zu schreiben. Material war aus früheren Projekten genügend vorhanden und so entstand recht schnell ein Konzept. Ziel war es, abgeschlossene Lösungen - schrittweise von der Anfangskonzeption über den Entwurf bis hin zur fertigen Implementierung - in Power Basic vorzustellen.

Allerdings sollte es noch eine ganze Weile dauern, genau bis Sommer 92, bis das Projekt im Endstadium angelangt war. Das Warten hat, so glaube ich, sich gelohnt. Das Buch spricht einerseits den Einsteiger an, der systematisch in die Programmentwicklung eingeführt wird. Daß Basic nicht zwangsläufig zu Spaghetticode führen muß, sondern übersichtlich strukturierte Programme erlaubt, dies wird auf den folgenden Seiten gezeigt. Aber auch der Aufsteiger und fortgeschrittene Profi wird genügend Anregungen und Tips für die tägliche Praxis finden. Außerdem erhält jeder PowerBASIC-Anwender eine Sammlung praktischer Werkzeuge für den Umgang mit dem PC. Dies reicht von Tools zur Unterstützung der Programmentwicklung in PowerBASIC, über PostScript-Treiber und Textbearbeitungsfunktionen, bis hin zu Erweiterungen des MS-DOS Befehlssatzes. Die Bibliotheken zur Konstruktion von Menüsteuerungen, zur Maussteuerung oder zum Zugriff auf dBase-Daten eröffnen gänzlich neue Möglichkeiten. Nicht immer ließen sich alle Wünsche und Vorstellungen realisieren, da PowerBASIC schon einige Einschränkungen erzwingt. Es kann auch nicht ausgeschlossen werden, daß alle Fehler erkannt und behoben wurden. Dies beeinträchtigt aber nicht die Verwendbarkeit des vorliegenden Buches als Hilfsmittel für die Arbeit mit PowerBASIC. Für Hinweise und Verbesserungsvorschläge sind Verlag und Autor dankbar.

An dieser Stelle sei allen Personen gedankt, die die Entstehung dieses Buches überhaupt erst ermöglichten. Hier sei meine Familie und insbesondere meine Frau Martha erwähnt, die mit viel Geduld und Verständnis die Arbeit begleitete. Georg Weiherer betreute die Arbeit und stand mit Rat und Tat zur Seite. Allen Lesern wünsche ich viel Spaß und Erfolg bei der Arbeit mit diesem Buch.

Günter Born

1 Einführung

Mit PowerBASIC steht eine mächtige Sprache für die Programmierung unter DOS zur Verfügung. Das Konzept vereint die Vorteile der interpretativen Sprache »Basic**!Syntaxfehler, DER**en, anspruchsvollere Programmlösungen zu erstellen. Dadurch läßt sich PowerBASIC von einer breiten Benutzerschicht einsetzen. Die Ergebnisse müssen im Hinblick auf Laufzeitverhalten und Codelänge professionellen Lösungen in nichts nachstehen.

Das vorliegende Buch greift diesen Gedanken auf, und bietet Anregungen zur Entwicklung eigener Programme. Ein Schwerpunkt liegt in Programmergänzungen zu PowerBASIC und MS-DOS. Dadurch gelangt der Anwender Schritt für Schritt zu einer Sammlung von Werkzeugen, die offene Wünsche befriedigen. Programme zur formatierten Druckerausgabe, Generatoren für Querverweislisten, Ausgabe beliebiger Dateien als Hexdump, Treiber für PostScript: dies sind nur einige Stichwörter zu den behandelten Themen. Auch wer sich mehr mit den alltäglichen Problemen beschäftigt, findet abgeschlossene Lösungen.

Durch PowerBASIC lassen sich die Programme zusätzlich in schnellen Maschinencode übersetzen, so daß sie in ihrem Laufzeitverhalten kaum professionellen Produkten nachstehen. Daß diese Lösungen auch noch transparent und verständlich realisiert werden können, wird nachfolgend gezeigt. Dabei beschränkt sich das Buchkonzept nicht auf den Abdruck reiner Listings. Vielmehr wird - beginnend bei der konzeptionellen Festlegung über den Entwurf bis hin zur Implementierung - jede Lösung stufenweise besprochen. Das Listing ist deshalb nur ein untergeordneter Teil des jeweiligen Abschnitts.

Dieses Vorgehen spiegelt den Ablauf der Software-Entwicklung wieder, so daß der Weg bis zur Lösung leichter nachvollziehbar ist. Zusätzliche Hinweise und Anregungen sollen schließlich zu eigenen Erweiterungen anregen.

Die Programmentwicklung direkt am Rechner ist zwar bei vielen Programmierern verbreitet. Die vermeintlich gewonnene Zeit geht aber spätestens bei der Fehlersuche verloren, ganz zu schweigen von der Qualität dieser Programme. Transparenz und Wartbarkeit sind kaum gegeben, wodurch insbesondere Lösungen in Basic negativ hervortreten. Aber dies muß nicht unbedingt so sein.

Aus umfangreicheren Entwicklungsvorhaben resultiert die Erfahrung, daß die Erstellung von Software in mehreren Phasen erfolgen sollte. Die nachfolgenden Kapitel berücksichtigen diesen Ansatz, indem sie neben dem Programmlisting insbesondere die Überlegungen in den vorhergehenden Phasen beschreiben. Da nicht jedem Leser die

verwendeten Begriffe vertraut sind, werden diese nachfolgend kurz erläutert.

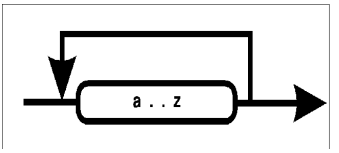


Bild 1.1: Grafische Darstellung eines Syntaxdiagramms

Die Entwicklung einer Software beginnt mit der Analyse der Anforderungen. Innerhalb dieser Spezifikationsphase dienen teilweise Syntaxdiagramme zur Beschreibung einzelner Sachverhalte. Bild 1.1 zeigt ein solches Diagramm, mit dem die Erzeugung von Worten aus Kleinbuchstaben des Alphabets beschrieben wird. Ausgehend von einem Startpunkt in der linken Ecke läßt sich das Diagramm beliebig oft durchlaufen, wobei jeweils ein Buchstabe (a .. z) an das bereits bestehende Teilwort angehängt wird. Ist das Wort komplett, wird das Syntaxdiagramm verlassen. Mit diesem Verfahren lassen sich viele Zusammenhänge einfach aber präzise darstellen.

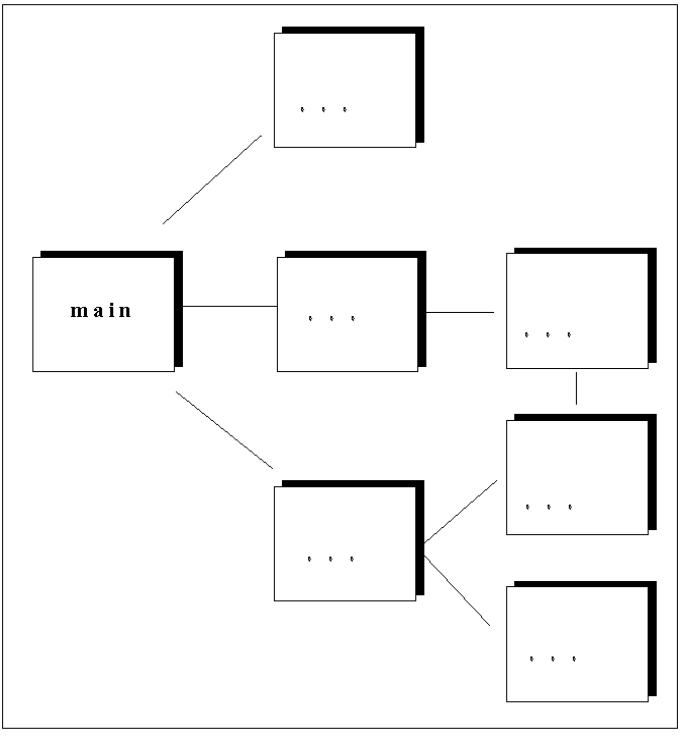


Bild 1.2: Modul- oder Hierarchiediagramm

Zusätzlich ist im Rahmen der Spezifikation die Bedieneroberfläche des Programmes festzulegen. In jedem Kapitel werden deshalb die Vorüberlegungen zur Erstellung der Software diskutiert. Erst dann erfolgt der Entwurf des Programmes. Um auch komplexere Aufgabenstellungen zu beherrschen, bietet es sich an, durch Modularisierung kleinere Teile zu schaffen, die leichter zu bearbeiten und zu handhaben sind. Alle Teilfunktionen werden dann schrittweise zur Gesamtlösung zusammengebaut. Um den Zusammenhang der einzelnen Module zu verdeutlichen, werden in diesem Buch Hierarchie- oder Moduldiagramme (Bild 1.2) benutzt.

Diese Diagramme zeigen den abstrahierten funktionalen und/oder datenmäßigen Zusammenhang, ohne durch Details in der Realisierung der Module zu verwirren.

Hinweis: Hierarchiediagramme werden zwar üblicherweise in vertikaler Richtung aufgebaut. Um die Darstellung zu vereinfachen, wurde aber obige Anordnung gewählt.

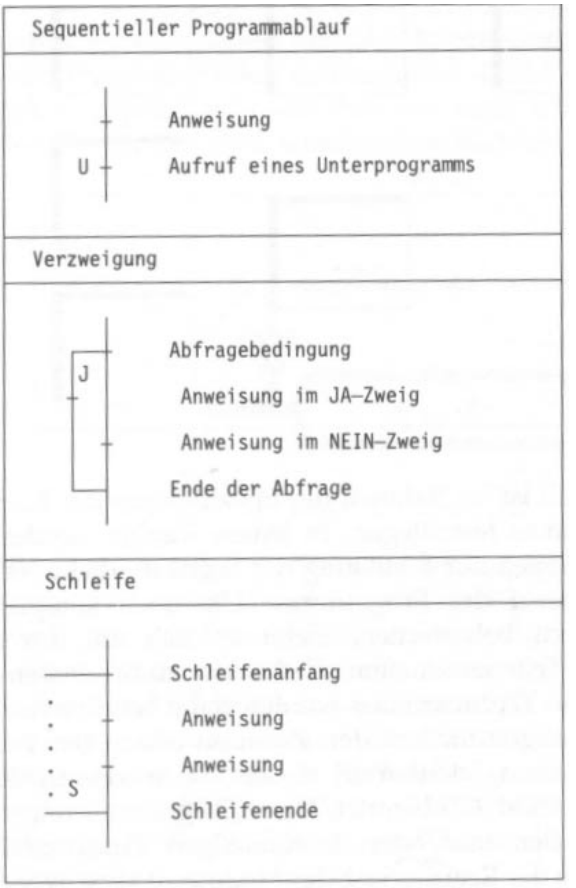


Bild 1.3: Darstellung des Programmablaufs mit Strichdiagrammen

Nach Zerlegung der Aufgabe in kleinere Pakete lassen sich die jeweiligen Funktionen durch Basic-Programme realisieren. Es hat sich in der Praxis als vorteilhaft erwiesen, vorher den Grobablauf des Programmes zu entwerfen. Bekannte Hilfsmittel hierfür sind Flußdiagramme und Strukturdiagramme (Struktogramme). Beiden Methoden haftet der Nachteil an, daß die grafische Darstellung aufwendig ist. Insbesondere bei Änderungen muß viel umgezeichnet werden. Ich benutze deshalb bereits seit einigen Jahren erfolgreich eine Abstraktion der Struktogramme in Form von Strichdiagrammen. Diese lassen sich sehr gut mit Texteditoren erstellen und pflegen. Sie zeigen den Programmfluß in Form von Linien, die die Darstellungen in Bild 1.3 verwenden:

Wichtig ist, daß diese Strichdiagramme so abstrahiert werden, daß sich der Programmablauf auf einem Blatt darstellen läßt. Notfalls können einzelne Punkte auf weiteren Blättern verfeinert werden. Mit etwas Übung lassen sich so recht schnell die Ablauf- und Kontrollstrukturen eines Programmes entwerfen, was sich in der Qualität des Programmentwurfs deutlich niederschlägt.

Nachdem die Struktur in dieser Form vorliegt, beginnt die eigentliche Programmierung. Es wird auf die Eingabe der üblichen Basic-Zeilennummern gänzlich verzichtet. Weiterhin sind die Programme ausgiebig kommentiert, um später die Einarbeitung zu erleichtern. Kommentare lassen sich in PowerBASIC nach wie vor mit dem Schlüsselwort REM eingeben. Um das Listing besser strukturieren und optisch gestalten zu können, wurde jedoch für Kommentare die Zeichenkombination '! gewählt. Das Ausrufezeichen dient nur zur optischen Signalisierung und gehört eigentlich nicht mehr zum Kommentar. Weitere Einzelheiten lernen Sie in den folgenden Kapiteln kennen.

2 Werkzeuge für PowerBASIC

In diesem Kapitel möchte ich einige Werkzeuge zur Unterstützung der Softwareentwicklung in PowerBASIC vorstellen. Es handelt sich dabei um Programme, die ich in den letzten Jahren für eigene Zwecke geschrieben und mittlerweile an PowerBASIC angepaßt habe. Neben der Aufbereitung von Programmtexten für die Druckerausgabe gehört ein Cross-Referenz-Generator für PowerBASIC dazu. Damit lassen sich auch umfangreichere Programme entwickeln und pflegen.

LISTER: Formatierte Druckerausgabe von Quellprogrammen

Die Programmerstellung erfolgt bei vielen Programmierern direkt am Bildschirm. Ich persönlich bevorzuge dagegen die Programmentwicklung auf einem Blatt Papier, d.h. der Programmentwurf entsteht auf dem Papier und wird erst dann in den Rechner eingegeben. Nach dem Editieren wird sofort ein Listing ausgegeben, mit dem dann die Fehlersuche und Weiterentwicklung des Programmes erfolgt.

Allerdings besteht unter PowerBASIC ein Problem: Programmlistings lassen sich nur mit den DOS-Befehlen PRINT oder COPY auf dem Drucker ausgeben. Bei intensiver Programmiertätigkeit treten aber früher oder später Schwierigkeiten auf. Einmal sind die Ausgaben von PRINT oder COPY nicht an den Drucker angepaßt. Bei mehrseitigen Programmen wird dann meist über die Perforation zwischen den Seiten gedruckt. Dies ist nicht nur optisch störend, sondern auch die Ablage in Ordnern wird so verhindert, da beim Auftrennen meist eine Zeile verloren geht. Bei manchen Druckern läßt sich zwar softwaremäßig die Zahl der Druckzeilen pro Seite einstellen. Aber dies ist keine überzeugende Lösung und erfordert meist ein entsprechendes Steuerprogramm.

Weiterhin läßt sich mit diesem Trick folgendes Handicap nicht beheben: Bei umfangreicher Programmentwicklung liegt nach einiger Zeit ein Stapel Listings vor. Dann beginnt die Raterei: wie heißt doch noch gleich die Quelldatei zu diesem Listing? Wann wurde das Listing ausgedruckt? Entspricht das Listing auch der neuesten Programmversion? Sicher, mit entsprechender Disziplin lassen sich einige dieser Probleme beheben. Aber wer hält in der Hitze des Gefechts schon eisern Disziplin? Und überhaupt, warum kann nicht der Computer die Aufgabe übernehmen, ein sauber formatiertes Listing zu erzeugen. Was fehlt, ist ein Programm zur Ausgabe von Listings auf dem Drucker, welches allen Ansprüchen genügt. Da der Hersteller von PowerBASIC dies nicht standardmäßig liefert, bleibt nur die Eigenentwicklung. PowerBASIC bietet hier sicherlich die Möglichkeit zur

Realisierung eines geeigneten Programmes. Das nachfolgend vorgestellt Programm ist daher der erste Ansatz für eine Lösung. Im nächsten Abschnitt werden dann einige Verfeinerungen gezeigt.

Bevor wir uns in die Entwicklung stürzen, sollten vorher die Anforderungen geklärt und spezifiziert werden. Welche Wünsche stehen eigentlich an?

Die Spezifikation

Als erstes soll das Programm nach einer vordefinierten Zeilenzahl einen Seitenvorschub im Listing einsetzen. Damit ist endlich das Problem gelöst, daß einzelne Zeilen genau auf dem Perforationsrand ausgedruckt werden. Wegen der verschiedenen Papierformate sollte die Zahl der Zeilen pro Druckseite variabel definierbar sein. Um anhand des Listings auch nach einiger Zeit auf den Dateinamen schließen zu können, muß auf jeder Druckseite der entsprechende Dateiname erscheinen. Zusätzlich ist eine fortlaufende Seitennumerierung durchzuführen. Schön wäre es weiterhin, wenn zumindest auf der ersten Seite das Datum des Ausdrucks enthalten ist. Ein weiteres Ärgernis betrifft die Einstellung des linken und rechten Druckrandes. Falls das Papier falsch eingespannt ist, oder der Text nicht in eine Zeile paßt, wird häufig über den Papierrand gedruckt. Steht der Text zu weit links, gehen beim Lochen der Blätter einzelne Buchstaben verloren. Um dies zu beheben, sollte eine variable Einstellung des linken und rechten Druckrandes möglich sein. Diese Anforderungen bringen bereits einigen Komfort in das Programm. Weitere Wünsche kommen aber noch hinzu. Vielfach ist es so, daß ein Programmlisting keine Zeilennummern enthält. Auch PowerBASIC setzt diese nicht mehr voraus. Oft ist aber eine Numerierung der Ausgabezeilen erwünscht. Deshalb muß diese Option wahlweise zu- oder abschaltbar sein. Dies bringt aber sofort ein Problem mit sich: Es dürfen nur die Zeilen im Originalprogramm numeriert werden. Falls die Druckbreite eine Aufteilung auf mehrere Zeilen erfordert, ist immer nur die erste Zeile zu numerieren. Optisch schön ist es weiterhin, wenn die dann zu umbrechenden Folgezeilen in der gleichen Spalte wie die erste Zeile beginnen (Bild 2.1).

```
21  IF a% = 10 THEN
22     z% = INSTR(20,textvariable$,
       "dies ist ein Test"
23  ENDIF
24  len = len + 1
```

Bild 2.1: Einrückung von Folgezeilen

Nachdem nun die Basisfunktionen geklärt sind, gilt der nächste Schritt der Gestaltung der Bedienoberfläche. Hier ergeben sich zwei Wege zur Benutzerführung. Naheliegend ist ein interaktiver Dialog, der die

benötigten Parameter abruft. Nachfolgend wird die zu realisierende Benutzeroberfläche beschrieben.

Wird das Programm mit der Eingabe:

```
LISTER
```

aufgerufen, ist der Bildschirm zu löschen und die Kopfmeldung erscheint:

```
L I S T E R                                (c) Born Version 1.0

Optionen  [  /L=00 linker Rand          /R=75   rechter Rand  ]
          [  /Z=60 Zeilen pro Seite    /N  Zeilennumerierung ]

File      :
Optionen  :
```

Bild 2.2: Kopfmeldung des Programms LISTER

Als Dateiname darf jede gültige MS-DOS-Bezeichnung einschließlich Laufwerks- und Pfadbezeichnung verwendet werden. Die Abfrage der »Optionen« soll allerdings erst nach Eingabe des Dateinames erfolgen. Die im Kopftext eingetragenen Werte zeigen die Standardeinstellung. Falls die Frage nach den Optionen durch Betätigung der Eingabetaste übergangen wird, übernimmt das Programm die Standardeingaben. In der vorliegenden Implementierung gilt:

```
Linker Rand bei Spalte     0
Rechter Rand bei Spalte    75
Zeilen pro Seite           60
Zeilennumerierung          Aus
```

Tabelle 2.1: Standardparameter für LISTER

Dadurch wird in vielen Fällen eine brauchbare Ausgabe ermöglicht. Die Optionen dürfen zwar in beliebiger Reihenfolge eingegeben werden, das Format ist jedoch gemäß obigen Angaben aufzubauen. Jede Option beginnt mit dem Zeichen »/«, gefolgt von einem Großbuchstaben für die Option.

```
/L=xx  linke Randeinstellung
/R=xx  rechte Randeinstellung
/Z=xx  Zeilen pro Druckseite
/N     Zeilennumerierung ein
```

Tabelle 2.2: Optionen beim Aufruf von LISTER

Bei numerischen Werten ist zwischen der Zahl und dem Buchstaben das Gleichheitszeichen erforderlich. Die Optionen sind durch ein Leerzeichen zu trennen. Nachfolgend sind einige gültige Optionen angegeben:

```
/N
/L=10 /Z=55 /R=60 /N
/L=5
/N /R=60
```

Fehlerhafte Eingaben (z.B. linker Rand größer als rechter Rand etc.) sind durch das Programm abzufangen. In diesem Fall erscheint die Fehlermeldung:

```
Bitte Randeinstellung neu setzen
```

Wird kein Dateiname eingegeben, bricht das Programm mit folgender Meldung ab:

```
Der Dateiname fehlt
```

Existiert die angegebene Datei nicht, endet das Programm ebenfalls mit der Meldung:

```
Die Datei  <Name>  existiert nicht
```

Wird eine Datei gefunden, beginnt die Ausgabe mit dem Hinweis:

```
Die Datei <Name>  wird auf dem Drucker ausgegeben
```

Name steht dabei für den eingegebenen Dateinamen. Gleichzeitig erfolgt auf dem Drucker die Ausgabe des Inhalts der Textdatei mit Dateiname, Datum und Seitennummer zu Beginn jeder Seite:

```
LISTER  /L=5 /R=74 /Z=55 /N       (C) Born Version 1.0
Datei:  <filename>         Datum:  MM/TT/JJ        Seite : 1

  1 '************************************************
  2 ' File       : LISTER.BAS
  3 ' Vers.      : 1.0
  4 ' Last Edit  : 20.4.92
  5 ' Autor      : G. Born
  6 ' File I/O   : INPUT, OUTPUT, FILE, PRINTER
  7 ' Progr. Spr.: POWERBASIC
  8 ' Betr. Sys. : DOS 2.1 - 5.0
  9 ' Funktion: Das Programm dient zur Ausgabe von Listings mit
 10 '           Seitennummern, Datum, Dateinamen und einer wähl-
 11 '           baren Zeilennumerierung. Weiterhin wird nach n
 12 '           Zeilen ein Papiervorschub auf dem Drucker ausge-
 13 '           löst. Es lassen sich beliebige Textdateien mit
 14 '           diesem Programm ausgeben.
 15 '
 16 ' Aufruf:   LISTER Filename /Optionen
 17 '           Optionen:  /N   Zeilennumerierung ein  [Aus]
 18 '                      /Lxx linker Rand            [ 0 ]
 19 '                      /Rxx rechter Rand           [75 ]
 20 '                      /Zxx Zeilen pro Seite       [60 ]
 21 '
 22 '           Die Werte in [] geben die Standardeinstellung
 23 '           wieder. Wird das Programm ohne Parameter aufge-
 24 '           rufen, sind Dateiname und Optionen explizit ab-
 25 '           zufragen. Mit dem Aufruf:
 26 '
 27 '              LISTER /?
 28 '
 29 '           wird ein Hilfsbildschirm ausgegeben.
```

```
30 ' ********************************************************
31 ' Variable definieren
32 %on = 1: %off = 0
33 nummer% = %off                          '! keine Zeilennummern
34 zeile& = 0                              '! Zeilennummer Listing
35 seite% = 1                              '! Seitennummer Listing
36 maxzeile% = 60                          '! Zeilen pro Seite
37 rechts% = 75                            '! rechter Rand
38 links% = 0                              '! linker Rand
39 spalte% = 0                             '! Einrückung
.  .
```

Bild 2.3: Ausgabeformat auf dem Drucker

Die Kopfzeile mit den Information über die Aufrufparameter:

```
LISTER  /L=10 /R=70 /N                (c) Born Version 1.0
```

soll allerdings nur auf der ersten Druckseite erscheinen. Nachdem die Ausgabe beendet ist, soll sich das Programm mit folgender Meldung verabschieden:

```
Ausgabe beendet
```

Anschließend erscheint die DOS-Systemmeldung wieder auf dem Bildschirm.

Damit scheint die Bedieneroberfläche auf den ersten Blick hinreichend beschrieben. Oft benutzt der Softwareentwickler jedoch Batchdateien zur Bearbeitung der Programme. Auch PowerBASIC bietet die Kommandozeilenversion. Hier lassen sich alle Kommandos und Eingaben von der DOS-Kommandoebene oder aus der Stapelverarbeitungsdatei übernehmen. Dann ist es natürlich äußerst störend, wenn zum Beispiel der Ablauf der Stapeldatei durch Benutzerabfragen unterbrochen wird. Hier muß ein Aufruf mit Parameterübergabe aus der Kommandoebene möglich sein. Diese Aufrufform ist von den MS-DOS-Kommandos (*DIR *.* /W*) hinreichend bekannt. Unser Ausgabeprogramm soll sich gemäß folgender Notation von der DOS-Oberfläche starten lassen:

```
LISTER Dateiname /Optionen
```

Sobald im Aufruf der Dateiname mit angegeben wird, schaltet das Programm in den Kommandomodus. Der oben beschriebene Dialog mit Kopfmeldung und Benutzerabfragen darf nicht mehr erscheinen. Lediglich Fehlermeldungen sind noch auf dem Bildschirm zulässig. Werden keine Optionen eingegeben, dann übernimmt das Programm die voreingestellten Standardwerte. Als Randbedingung gilt, daß der Dateiname immer zuerst einzugeben ist. Die Optionen müssen durch mindestens ein Leerzeichen vom Dateinamen getrennt werden. Die folgenden Eingaben stellen gültige Aufrufe des Programms dar:

```
LISTER Datei
LISTER Datei /N /L=2 /R=15 /Z=69
lister datei /N
```

Um es nochmals zusammenzufassen: Durch Eingabe des reinen Programmnamens (LISTER) wird die interaktive Benutzeroberfläche selektiert. Sobald in der Kommandozeile der Name der Ausgabedatei erscheint, befindet sich das Programm im sogenannten Kommandomodus. Dann sind alle Eingabeparameter in der Kommandozeile einzutragen und durch das Programm zu lesen.

Als letzter Punkt soll das Programm noch eine Online-Hilfe bieten. In Anlehnung an DOS 5.0 wird diese Online-Hilfe mit folgender Option abgerufen:

```
LISTER /?
```

Dann muß auf dem Bildschirm folgender Hilfstext erscheinen:

```
L I S T E R                          (c) Born Version 1.0

Aufruf: Lister <Filename> <Optionen>

Optionen:

  /L=00 setzt den linken Rand
  /R=75 setzt den rechten Rand
  /Z=60 setzt die Zeilenzahl pro Seite
  /N    schaltet die Zeilennumerierung ein

Das Programm gibt ein Listing der Datei aus, wobei sich
die Ränder und die Zahl der Zeilen einstellen läßt.
```

Danach bricht das Programm und der DOS-Prompt erscheint wieder.

Der Entwurf

Das Programm ist zweckmäßigerweise in mehrere Module zu unterteilen, deren Zusammenführung in Bild 2.4 gezeigt wird.

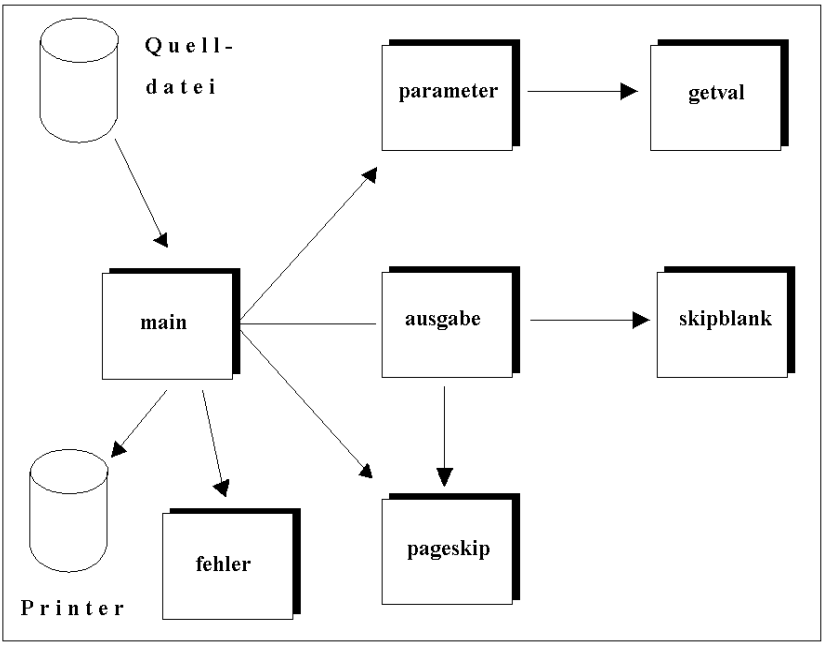


Bild 2.4: Modulhierarchie des Programmes LISTER

Die Steuerung der Benutzereingaben sowie der Hauptablauf wird in das Modul *main* verlegt. Der Ablauf innerhalb dieses Moduls wird an folgendem Strichdiagramm (Bild 2.5) deutlich.

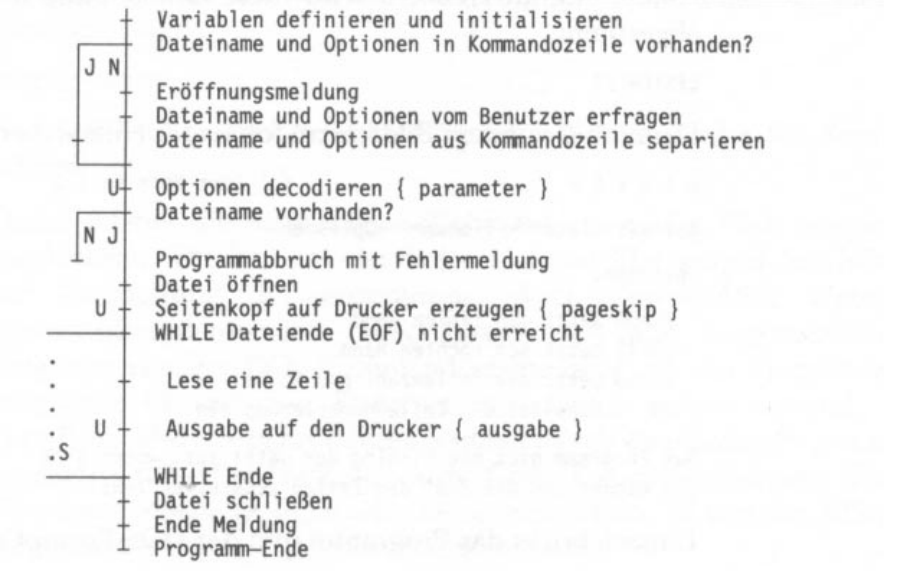


Bild 2.5: Programmablauf in LISTER.BAS

Der Ablauf innerhalb der Hilfsmodule ist recht einfach, so daß auf eine Darstellung in Form von Strichdiagrammen verzichtet wird.

Die Implementierung

Das Programm besitzt die oben gezeigte Modulstruktur. Aufbau und Funktion dieser Module sollen nun kurz beschrieben werden.

Hauptprogramm

In diesem Modul sind die Variablen zu definieren und zu initialisieren. Hier lassen sich auch die Standardwerte für die Randeinstellung vorgeben. Die Zahl der auszugebenden Zeilen pro Seite wird in der Variablen *maxzeile%* geführt. Erreicht der aktuelle Zeilenzähler (*szeile%*) diesen Wert, erfolgt ein Papiervorschub und eine neue Seite (*seite%*) wird ausgegeben. Falls der Dateiname in der Kommandozeile enthalten ist, ist dieser mit den Optionen in die Variablen *filename$* und *options$* zu separieren. In diesem Fall kann auf die Ausgabe der Kopfmeldung verzichtet werden. Andernfalls erscheint die Kopfmeldung und der Name des auszugebenden Files sowie die Optionen werden abgefragt. Durch die Angabe:

```
ON ERROR GOSUB fehler
```

werden mögliche Fehler abgefangen. Hierzu zählen auch nicht vorhandene Dateien. Existiert diese nicht, liefert das Unterprogramm eine Fehlermeldung und bricht den Ablauf ab. Andernfalls wird der erste Seitenkopf mittels *pageskip* auf dem Drucker ausgegeben. Dies ist möglich, da der Wert von *szeile%* auf den Wert von *maxzeile%* gesetzt wurde. Mit der Schleife

```
WHILE NOT (EOF(datei))
 LINE INPUT #datei, linie$
 GOSUB ausgabe
WEND
```

wird die Datei zeilenweise gelesen und über das Modul »ausgabe« verarbeitet. Ist das Dateiende erreicht, wird die Schleife verlassen, die Daten geschlossen und das Programm beendet.

Die Lister-Hilfsmodule

Um den Aufbau des Programmes transparent und änderungsfreundlich zu gestalten, gelangen mehrere Unterprogramme zum Einsatz. Dadurch konnte insbesondere das Hauptprogramm sehr kompakt gehalten werden.

pageskip

Zur Ausgabe des Programmkopfes auf den Drucker wurde bereits der Aufruf `GOSUB pageskip` erwähnt. Dieses Modul prüft bei jedem Aufruf, ob die aktuell ausgegebene Zeilenzahl (*szeile%*) den Grenzwert (*maxzeile%*)

erreicht hat und veranlaßt gegebenenfalls einen Seitenvorschub, wobei die Anweisung:

```
LPRINT CHR$(12)
```

das Steuerzeichen (Form Feed = 12) an den Drucker ausgibt. Anschließend protokolliert *pageskip* den Dateinamen, das Datum und die aktuelle Seitennummer.

ausgabe

Dieses Unterprogramm bearbeitet die in der Variablen *linie$* übergebenen Texte für die Druckerausgabe. Zuerst werden so viele Leerzeichen ausgegeben, daß die linke Randeinstellung stimmt. Bei eingeschalteter Zeilennumerierung wird die aktuelle Zeilennummer (*zeile&*) ausgegeben. Dann kann der Druck des eigentlichen Textes beginnen. Übersteigt die Satzlänge den spezifizierten Druckbereich (rechts - links), dann erfolgt eine Aufteilung auf mehrere Ausgabezeilen. Dabei ist auf eine korrekte Einrückung der Folgezeilen zu achten. In der Variablen *spalte%* wird der Beginn der Druckspalte gespeichert. Bei Erreichen der maximalen Zeilenzahl pro Seite sorgt der Aufruf von *pageskip* für einen Seitenwechsel.

skipblank

Dieses kleine Hilfsprogramm ermittelt lediglich die Zahl der führenden Leerzeichen in einer eingelesenen Textzeile. Dieser Wert dient zur Bestimmung der Einrückung für die Folgezeilen.

parameter

Das Unterprogramm ist für die Decodierung der Eingabeoptionen zuständig. Normalerweise werden Eingaben einer kompletten Syntaxanalyse unterzogen. Aus Aufwandsgründen benutzt das Modul eine sehr einfache aber wirkungsvolle Strategie zur Erkennung gesetzter Optionen. Mit Hilfe der Basic-Funktion INSTR wird einfach der Inhalt der Variablen *options$* nach den Zeichenmustern für die jeweiligen Optionen abgefragt. Wird ein gültiger Schalter erkannt (/L=), dann kann der Eingabewert decodiert werden. Anschließend überschreibt das eingelesene Ergebnis die Standardeinstellung. Durch sukzessive Abfrage der verschiedenen Möglichkeiten lassen sich alle Optionen ermitteln.

getval

Hier handelt es sich um ein echtes Unterprogramm, welches als Parameter die jeweils zu setzende Variable erhält. In dem Programm wird der String *options$* ab der Position *ptr%+3* untersucht. Hier findet sich der eingegebene Wert als ASCII-Zahl. Zuerst muß das Ende der Zahl gefunden werden. Dies wird durch ein Leerzeichen oder das Textende signalisiert. Dann ist der Wert in ein numerisches Format zu codieren und zurückzugeben.

fehler

Dieses Modul bildet den Fehlerausgang des Programmes LISTER. Es wird immer dann durch Basic aktiviert, wenn Laufzeitfehler auftreten.

Einzelheiten über den Programmablauf sind dem Listing zu entnehmen. Das Programm eignet sich nicht nur zur Ausgabe von Basic-Listings. Auch bei Turbo Pascal, dBase etc. treten ähnliche Probleme (fehlende Dateinamen und Datum, Druck über die Perforation etc.) auf. Selbst normale Textdateien lassen sich so formatiert ausgeben.

Erweiterungsvorschläge

Das Programm benutzt die Anweisung LPRINT für die Ausgabe auf den Drucker. Wer die Ausgabe in beliebige Dateien umleiten möchte, kann alle LPRINT-Statements durch »PRINT #aus «

```
X R E F   /Z=55                              (c) Born Version 1.0
Datei : lister.bas      Datum : 05-12-1992          Seite :  1

 Zeile     Anweisung

      ' ********************************************************
      ' File       : LISTER.BAS
      ' Vers.      : 1.0
      ' Last Edit  : 20. 4.92
      ' Autor      : G. Born
      ' File I/O   : INPUT, OUTPUT, FILE, PRINTER
      ' Progr. Spr.: POWERBASIC
      ' Betr. Sys. : DOS 2.1 - 5.0
      ' Funktion: Das Programm dient zur Ausgabe von Listings mit
      '           Seitennummern, Datum, Dateinamen und einer wähl-
      '           baren Zeilennumerierung. Weiterhin wird nach n
      '           Zeilen ein Papiervorschub auf dem Drucker ausge-
      '           löst. Es lassen sich beliebige Textdateien mit
      '           diesem Programm ausgeben.
      '
      ' Aufruf:   LISTER Filename /Optionen
      '           Optionen:  /N   Zeilennumerierung ein  [Aus]
      '                      /Lxx linker Rand            [ 0 ]
      '                      /Rxx rechter Rand           [75 ]
      '                      /Zxx Zeilen pro Seite       [60 ]
      '
      '           Die Werte in [] geben die Standardeinstellung
      '           wieder. Wird das Programm ohne Parameter aufge-
      '           rufen, sind Dateiname und Optionen explizit ab-
      '           zufragen. Mit dem Aufruf:
      '
      '              LISTER /?
      '
      '           wird ein Hilfsbildschirm ausgegeben.
      ' ********************************************************
      ' Variable definieren
    1 %on = 1: %off = 0
```

```
    2 nummer% = %off                          '! keine Zeilennummern
    3 zeile& = 0                              '! Zeilennummer Listing
    4 seite% = 1                              '! Seitennummer Listing
    5 maxzeile% = 60                          '! Zeilen pro Seite
    6 rechts% = 75                            '! rechter Rand
    7 links% = 0                              '! linker Rand
    8 spalte% = 0                             '! Einrückung

    9 datei% = 2                              '! Dateinummer

   10 ON ERROR GOTO fehler                    '! Fehlerausgang

      '############################################################
      '#                       Hauptprogramm                      #
      '############################################################

   11 kommando$ = COMMAND$                    '! Parameter?
   12 IF LEN (kommando$) = 0 THEN             '! User Mode?
   13  CLS                                    '! clear Screen

   14  PRINT "L I S T E R                          (c) Born
Version 1.0"
   15  PRINT
   16  PRINT "Optionen [ /L=00 linker Rand         /R=75  rechter
Rand   ]
        "
   17  PRINT "          [ /Z=60 Zeilen pro Seite    /N
Zeilennumerierung ]
        "
   18  PRINT
   19  INPUT  "File     : ",filename$
   20  INPUT  "Optionen : ",options$
   21  PRINT
   22 ELSE
   23  ptr% = INSTR (kommando$,"/?")          '! Option /?
   24  IF ptr% <> 0 THEN                      '! Hilfsbildschirm
   25   PRINT "L I S T E R                    (c) Born Version 1.0"
   26   PRINT
   27   PRINT "Aufruf: LISTER <Filename> <Optionen>"
   28   PRINT
   29   PRINT "Optionen :"
   30   PRINT
   31   PRINT "  /L=00 setzt den linken Rand"
   32   PRINT "  /R=75 setzt den rechten Rand"
   33   PRINT "  /Z=60 setzt die Zeilenzahl pro Seite"
   34   PRINT "  /N    schaltet die Zeilennumerierung ein"
   35   PRINT
   36   PRINT "Das Programm gibt ein Listing der Datei aus, wobei
sich"
   37   PRINT "die Ränder und die Zahl der Zeilen einstellen läßt."
   38   PRINT
   39   SYSTEM
   40  END IF
   41       '! Kommando Mode
   42  ptr% = INSTR (kommando$,"/")           '! Optionen?
```

```
   43  IF ptr% = 0 THEN
   44   filename$ = kommando$                '! nur Filename
   45  ELSE
   46   filename$ = LEFT$(kommando$,ptr% -1)'! Filename separieren
   47   options$  = MID$(kommando$,ptr%)     '! Optionen separieren
   48  END IF
   49 END IF

   50 GOSUB parameter                       '! Optionen decodieren

   51 IF (rechts% < links%) or (maxzeile% < 10) THEN '! sinnlose
   52  PRINT                                         '! Einstellung
   53  PRINT "Bitte Randeinstellung neu setzen"      '! Fehlerexit
   54  SYSTEM
   55 END IF


   56 IF filename$ = "" THEN                '! Leereingabe?
   57  PRINT
   58  PRINT "Der Dateiname fehlt"
   59  SYSTEM
   60 END IF

      ' prüfe ob Datei vorhanden, nein -> exit

   61 OPEN filename$ FOR INPUT AS #datei%
   62 PRINT
   63 PRINT "Die Datei: ";filename$;" wird auf dem Drucker
ausgegeben "

   64 GOSUB pageskip                        '! Seitenkopf ausgeben

   65 WHILE NOT (EOF(datei%))               '! datei sequentiell
lesen
   66  LINE INPUT #datei%, linie$           '! lese Zeile
   67  GOSUB ausgabe                        '! drucke Zeile
   68 WEND

   69 LPRINT CHR$(12)                       '! Seitenvorschub
   70 CLOSE #datei%                         '! Datei schließen
   71 PRINT
   72 PRINT "Ausgabe beendet"
   73 END

      '############################################################
      '#                    Hilfsroutinen                         #
      '############################################################

   74 fehler:
      '------------------------------------------------------------
      '! Fehlerbehandlung in LISTER
      '------------------------------------------------------------

   75 IF ERR = 53 THEN
   76  PRINT "Die Datei ";filename$;" existiert nicht"
```

```
   77 ELSE
   78  IF ERR = 27 THEN
   79   INPUT "Druckerstörung, bitte eine Taste betätigen...";t$
   80  ELSE
   81   PRINT "Fehler : ";ERR;" unbekannt"
   82   PRINT "Programmabbruch"
   83  END IF
   84 END IF
   85 END                                    '! MSDOS Exit

   86 parameter:
      '-----------------------------------------------------------
      '! Decodiere die Eingabeoptionen
      '-----------------------------------------------------------

   87 ptr% = INSTR (options$,"/N")
   88 IF ptr% > 0 THEN nummer% = %on        '! Zeilennumerierung
EIN

   89 ptr% = INSTR (options$,"/Z=")
   90 IF ptr% > 0 THEN CALL getval (maxzeile%) '! Zeilen/Seite
   91 szeile% = maxzeile% + 1               '! Zeilennr Seite
wechseln

   92 ptr% = INSTR (options$,"/L=")
   93 IF ptr% > 0 THEN CALL getval (links%) '! linker Rand

   94 ptr% = INSTR (options$,"/R=")
   95 IF ptr% > 0 THEN CALL getval (rechts%) '! rechter Rand

   96 RETURN

   97 SUB getval (wert%)
      '-----------------------------------------------------------
      '! Decodiere den Eingabestring in eine Zahl
      '-----------------------------------------------------------
   98  SHARED options$, ptr%
   99  LOCAL i%

  100  ptr% = ptr% + 3                     '! ptr hinter /x=
  101  i% = 1
  102  WHILE ((ptr%+i%) =< LEN (options$)) and
(MID$(options$,ptr%+i%,1) <
       > " ")
  103   i% = i% + 1                        '! Ziffernzahl + 1
  104  WEND
  105  wert% = VAL(MID$(options$,ptr%,i%))  '! decodiere die Zahl
  106 END SUB

  107 pageskip:
      '-----------------------------------------------------------
      '! Seitenvorschub mit Kopf (Dateiname, Datum, Seite)
      '-----------------------------------------------------------
  108  IF szeile% =< maxzeile% THEN RETURN  '! kein Seitenwechsel
!!
```

```
  109  IF seite% > 1 THEN                    '! 1. Seite k. Vorschub
  110   LPRINT CHR$(12)                      '! Vorschub
  111   szeile% = 3                          '! Kopf 3 Zeilen
  112  ELSE
  113   LPRINT "LISTER "; options$; SPACE$(27);
  114   LPRINT "(c) Born Version 1.0"
  115   szeile% = 4                          '! Kopf 1. Seite = 4
Zeilen
  116  END IF
  117  LPRINT "Datei : ";filename$;"     Datum : ";DATE$;
  118  LPRINT "          Seite : "; seite%
  119  LPRINT
  120  INCR seite%                           '! Seite erhöhen


  121 RETURN


  122 ausgabe:
      '-------------------------------------------------------
      '! Ausgabe der eingelesenen Zeile auf dem Printer.
      '! rest% gibt an, wieviele Zeichen pro Zeile gedruckt
      '! werden dürfen. Ist die eingelesene Zeile länger, wird
      '! sie auf mehrere Ausgabezeilen aufgeteilt.
      '-------------------------------------------------------

  123  zeile& = zeile& + 1                   '! Zeile im Listing
  124  GOSUB pageskip                        '! Seitenvorschub?

  125  spalte% = links%                      '! linker Rand
  126  LPRINT SPACE$(spalte%);               '! auf linken Rand

  127  IF nummer% = %on THEN                 '! Zeilennumerierung?
  128   LPRINT USING "###### "; zeile&;      '! Zeilennummer drucken
  129   spalte% = spalte% + 7                '! Spalte 7 setzen
  130  END IF

  131  rest% = rechts% - spalte%             '! Restzeilenlänge
  132  GOSUB skipblank                       '! merke Blanks
  133  LPRINT LEFT$(linie$,rest%)            '! Ausgabe Teilstring
  134  linie$ = MID$(linie$, rest% + 1)      '! Reststring
  135  szeile% = szeile% + 1

  136  WHILE LEN(linie$) > rest%             '! String > Zeile
  137   GOSUB pageskip                       '! Seitenvorschub?
  138   LPRINT SPACE$(spalte%);              '! linker Rand
  139   LPRINT LEFT$(linie$,rest%)           '! Teilstring ausgeben
  140   linie$ = MID$(linie$,rest% + 1)      '! Reststring bestimmen
  141   szeile% = szeile% + 1                '! Zeile im Listing
  142  WEND

  143  IF LEN(linie$) > 0 THEN
  144   GOSUB pageskip                       '! Seitenvorschub?
  145   LPRINT SPACE$(spalte%);linie$        '! Reststring ausgeben
```

```
  146   szeile% = szeile% + 1
  147  END IF
  148 RETURN

  149 skipblank:
      '-------------------------------------------------------
      '! zähle führende Blanks
      '-------------------------------------------------------
  150 i% = 1
  151 WHILE (i% < LEN(linie$)) and (MID$(linie$,i%,1) = " ")
  152  i% = i%+1

Datei : lister.bas      Datum : 05-12-1992         Seite :  6

 Zeile     Anweisung

  153  spalte% = spalte% + 1
  154 WEND
  155 RETURN
      ' ###### Programm Ende #########
  156 END
```

Listing 2.1: LISTER.BAS

SPOOL: Ausgabe an den Drucker im Hintergrund

Das im vorhergehenden Abschnitt besprochene Programm LISTER sorgt bereits für ein sauber formatiertes Listing mit Seitennumerierung und Randeinstellung. Damit ergänzt es die DOS-Funktionen COPY und PRINT. Falls mehrere Dateien auszugeben sind, macht sich aber eine Schwäche des Programmes LISTER bemerkbar. Während PRINT die Ausgabe an den Drucker im Hintergrund abwickelt, blockiert das Programm LISTER den Rechner für die Zeit der Ausgabe.

Es stellt sich daher die Aufgabe einer Erweiterung des Programmes LISTER um eine Funktion zur Druckerausgabe im Hintergrund.

Der Entwurf

Die Benutzeroberfläche wird von LISTER übernommen, so daß auf die explizite Beschreibung verzichtet wird. Offen ist aber nach wie vor die Frage, wie sich die Druckerausgabe im Hintergrund realisieren läßt. Naheliegend ist der Gedanke, einen geeigneten Algorithmus zu implementieren, der die Ausgabe an den Drucker zwischenpuffert. Dann wird der Inhalt des Puffers im Hintergrund über die Schnittstelle ausgegeben. Dies setzt einmal eine solide Kenntnis des Betriebssystems voraus. Andererseits eignet sich PowerBASIC auch nicht optimal für diese Aufgabe. Ein weiteres Problem tritt auf, falls der Puffer im Speicher des Rechners liegt. MS-DOS kann nur einen Hauptspeicherbereich von 640 Kbyte verwalten. Wenn nun für den Spooler ein Pufferbereich von 100 bis 200 Kbyte reserviert wird, fehlt dieser Bereich den

Anwendungsprogrammen. Eine Verkleinerung des Puffers bringt auch nichts, da dieser dann schnell überläuft und damit die Ausgabe im Hintergrund blockiert ist. Also ist der Zwischenpuffer in Form einer Datei zu realisieren. Dies ist nicht weiter tragisch, da ja die Möglichkeit besteht, den formatierten Text direkt in die Ausgabedatei zu schreiben. Es bleibt damit nur die Weiterverarbeitung dieser Datei im Hintergrund. Dies ist wiederum eine Aufgabe, die erhebliches Detailwissen über den internen Aufbau von MS-DOS/PC-DOS erfordert. Da zwischen den verschiedenen Versionen von MS-DOS Unterschiede bestehen, muß die Implementierung dies berücksichtigen. Vor der Implementierung eines solchen Programmes sollte aber überlegt werden, ob es nicht eine einfachere Lösung gibt.

Das MS-DOS Programm PRINT.COM oder PRINT.EXE erfüllt im Grunde unsere Anforderungen. Da es mit DOS ausgeliefert wird, sind die versionsspezifischen Details bereits berücksichtigt. Optimaler kann es eigentlich nicht sein. Damit reduziert sich die Aufgabe auf die Aktivierung des Programmes PRINT. Hierzu existieren zwei Alternativen, die nachfolgend kurz skizziert werden.

Einmal bietet die PowerBASIC-Funktion SHELL eine recht einfache Schnittstelle zur Aktivierung von PRINT. Der Aufruf:

```
SHELL "PRINT" + filename$
```

aktiviert das Programm PRINT und übergibt die Dateibezeichnung in *filename$*. Auf dem Bildschirm erscheinen anschließend die Ausgaben von PRINT (z.B. der Inhalt der Druckerwarteschlange).

Die Benutzung von PRINT bietet noch einige andere Vorteile. So lassen sich zum Beispiel beim ersten Aufruf verschiedene optionale Parameter (z.B. Schnittstelle LPT1..n, COM1..n, Zahl der Zeittake etc.) angeben. Damit läßt sich PRINT an die individuellen Bedürfnisse anpassen. Weiterhin können jederzeit weitere Dateien in der Druckerwarteschlange gelöscht oder hinzugefügt werden. Eine Beschreibung dieser Aufrufe würde den Rahmen dieses Buches sprengen. Normalerweise ist die Bedienerschnittstelle aber in den DOS-Handbüchern beschrieben.

Für den Systemprogrammierer bietet DOS eine undokumentierte Funktion, um über den Interrupt 2FH mit PRINT zu kommunizieren. Genauer gesagt besteht PRINT aus mehreren Teilen. Beim ersten Aufruf wird ein Initialisierungsteil durchlaufen, welcher den residenten Code installiert. Das residente Modul ist dann für die Ausgabe im Hintergrund zuständig. Wird von der DOS-Kommandoebene das Programm PRINT aufgerufen, bezieht sich dies auf den transienten Teil, der lediglich die Einträge in der internen Druckerwarteschlange manipuliert. Der Interrupt 2FH bietet nun ebenfalls einige Funktionen, um mit dem residenten Teil von PRINT zu kommunizieren. Tabelle 2.3 gibt eine Übersicht über die verfügbaren Funktionen.

Funktion	Register
Check Instal-lationsstatus	AX = 0100 (CALL) AL =Statusbyte (RETURN) AL=FFH PRINT installiert
Datei in die Liste eintragen	AX = 0101 (CALL) DS:DX Zeiger auf die Submittabelle CY=1 Fehler (RETURN) AL Status
Datei aus der Liste löschen	AX = 0102 (CALL) DS:DX Zeiger auf Dateiname CY=1 Fehler (RETURN) AL Status
Liste löschen	AX = 0103 (CALL) CY=1 Fehler (RETURN) AL Status
Status abfragen	AX = 0104 (CALL) CY=1 Fehler (RETURN) AL Status CY=0 ok DS:SI Zeiger auf Ausgabeliste
Ende Statusab-frage	AX = 0105 (CALL) AL Status (RETURN) CY=1 Fehler

Tabelle 2.3: Funktionen des undokumentierten DOS-Interrupts INT 2F

Auf eine detailliert Beschreibung der Systemaufrufe wird an dieser Stelle verzichtet, da sie den Rahmen dieses Buches sprengt. Der interessierte Leser sei hier auf die Literaturangabe /1/ im Anhang dieses Buches verwiesen.

Die Implementierung

Da das Programm sich sehr stark an das Modul LISTER anlehnt, werden nur noch die für SPOOL spezifischen Teile besprochen. Bild 2.6 enthält eine Übersicht aller Module und deren Zusammenschaltung.

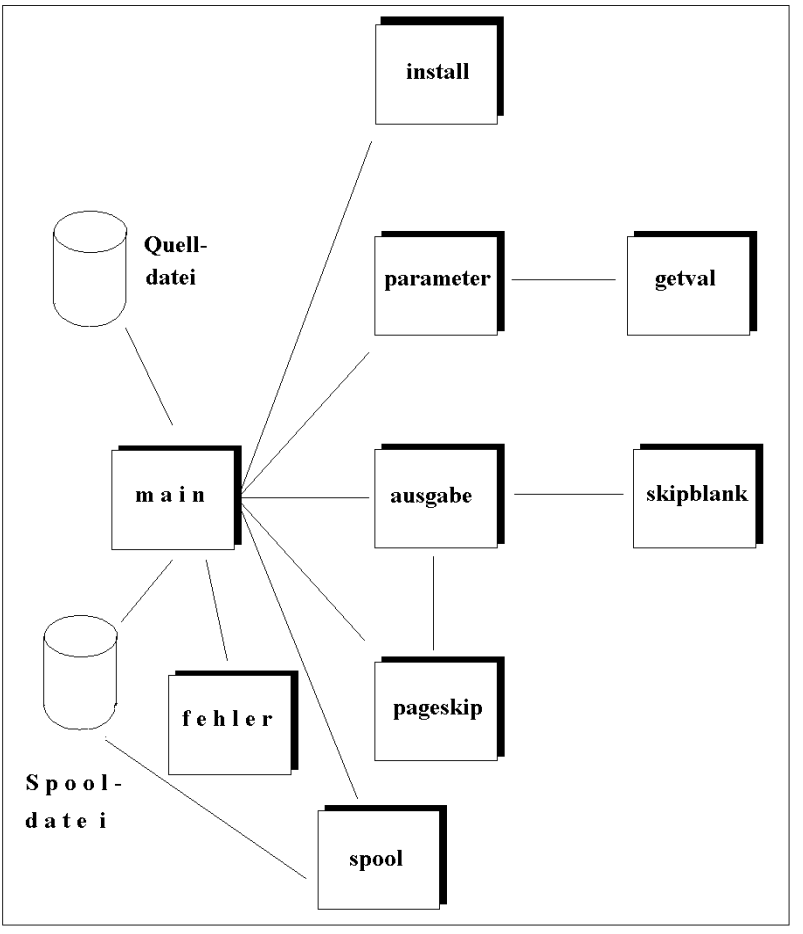


Bild 2.6: Hierarchiediagramm von SPOOL.BAS

Hauptmodul

Im Hauptprogramm wird als erstes das Unterprogramm »install« aufgerufen. Dieses prüft, ob der residente Teil von PRITN bereits installiert ist. Trifft dies zu, wird das Programm fortgesetzt, andernfalls erfolgt ein Abbruch.

Theoretisch besteht zwar die Möglichkeit, das Programm PRINT über den SHELL-Befehl zu installieren. Dies ist aber wegen der nachteiligen Folgen nicht erwünscht. Auf den unteren Speicheradressen ist DOS abgelegt, und daran schließen sich die Anwenderprogramme an. Hier befindet sich auch der Code des Programmes SPOOL. Der Befehl SHELL lädt nun eine Kopie des DOS-Kommandointerpreters COMMAND.COM in den Speicher. Bei der anschließenden Installation wird der residente Teil von PRINT oberhalb dieser Programme in den freien Speicher geladen. Nach Beendigung von SPOOL verbleibt der residente Teil von PRINT im

Speicher. Damit zerfällt der freie Speicherbereich in zwei Teile, wobei Anwendungsprogramme immer nur zusammenhängende Bereiche nutzen können. Benötigt ein Programm nun 450 Kbyte Speicher, läßt es sich nicht laden, falls kein zusammenhängendes Stück dieser Größe existiert. Die Ursache liegt in der Fraktionierung durch den residenten Teil von PRINT, die den Speicher in zwei kleinere Teile splittet. Abhilfe schafft in dieser Situation nur noch ein Neustart des Systems. Der interessierte Leser sei hier ebenfalls auf /1/ verwiesen. In diesem Buch findet sich auch eine detaillierte Beschreibung des DOS-Speichermanagements.

Für unseren Fall bleibt festzuhalten, daß PRINT vor dem Aufruf von SPOOL zu installieren ist. Damit ist sichergestellt, daß der residente Teil direkt oberhalb von DOS liegt. Damit werden die beschriebenen Probleme vermieden. Der formatierte Text wird nicht an den Drucker, sondern in eine Datei übertragen. Diese erhält den Namen der Eingabedatei, wobei aber als Extension die Bezeichnung TMP angehängt wird. Der Benutzer sollte deshalb diese Bezeichnung nicht für andere Dateien verwenden, da diese dann bei gleicher Namensgebung überschrieben werden.

Ausgabe

Dieses Modul wurde aus dem Programm LISTER übernommen. Die Anweisung:

```
PRINT #outdatei%, ...
```

ersetzt die LPRINT-Befehle und sorgt für die Testausgabe in die Spooldatei.

Erwähnt werden soll noch ein kleine Besonderheit. Der Name der Ausgabedatei wird aus dem Namen der Quelldatei bestimmt. Dabei wird lediglich die Extension entfernt und durch die Endung »TMP« ersetzt. Beid er Ermittlung de Zieldatei wird davon ausgegangen, dass keine Laufwerks- und Pfadbezeichnungen benutzt wurden. Gegebenenfalls müssen Sie dies bei Ihrer Anwendung korrigieren. Weiterhin fragt das Modul den DOS-Umgebungsbereich nach einer Variablen TEMP ab. Diese Variable gibt unter DOS 5.0 und Windows 3.x die Lage des Verzeichnisses mit den temporären Dateien an. Sofern dieses Verzeichnis existiert, wird die Ausgabedatei in diesem Verzeichnis angelegt. Fehlt die Umgebungsvariable, wird die Ausgabedatei im aktuellen Verzeichnis ausgegeben.

install

Hier wird der Installationsstatus von PRINT überprüft. Das Unterprogramm benutzt den INT 2F für diese Aufgabe. Ist der residente Teil von PRINT vorhanden, erhält das Register AL den Wert 0FFH. Bei anderen Werten bricht das Modul mit der Meldung:

```
Bitte PRINT installieren
```

ab. Anschließend erscheint die DOS-Systemmeldung auf dem Bildschirm.

spool

Sobald der Text fertig formatiert in der Ausgabedatei vorliegt, ist der Name in die Warteschlange von PRINT einzutragen. Die vorliegende Implementierung benutzt hierzu das SHELL-Kommando.

Weitere Einzelheiten sind nachfolgendem Listing zu entnehmen.

Erweiterungsvorschläge

Das Programm überschreibt die Ausgabedatei ohne Warnung. Hier könnte eine entsprechende Bedienerabfrage eingefügt werden. Eine andere Möglichkeit besteht darin, den Dateinamen über den INT 2F in die Warteschlange einzutragen. Dadurch entfällt die Meldung von PRINT auf dem Bildschirm. Weiterhin könnten die Einstellparameter per Umgebungsvariable vordefiniert werden.

```
X R E F   /Z=55                              (c) Born Version 1.0
Datei : spool.bas      Datum : 05-12-1992          Seite :  1

 Zeile     Anweisung

      ' ********************************************************
      ' File       : SPOOL.BAS
      ' Vers.      : 1.0
      ' Last Edit  : 28.4.92
      ' Autor      : G. Born
      ' File I/O   : INPUT, OUTPUT, FILE, PRINTER
      ' Progr. Spr.: POWERBASIC
      ' Betr. Sys. : DOS 3.0 - 5.0
      ' Funktion: Das Programm dient zur Ausgabe von Listings mit
      '           Seitennummern, Datum, Dateinamen und einer wähl-
      '           baren Zeilennumerierung. Weiterhin wird nach n
      '           Zeilen ein Papiervorschub auf dem Drucker ausge-
      '           löst. Es lassen sich beliebige Textdateien mit
      '           diesem Programm ausgeben.
      '
      ' Aufruf:   SPOOL Filename /Optionen
      '           Optionen:  /N   Zeilennumerierung ein  [Aus]
      '                      /Lxx linker Rand            [ 0 ]
      '                      /Rxx rechter Rand           [75 ]
      '                      /Zxx Zeilen pro Seite       [60 ]
      '
      '           Die Werte in [] geben die Standardeinstellung
      '           wieder. Wird das Programm ohne Parameter aufge-
      '           rufen, sind Dateiname und Optionen explizit ab-
      '           zufragen. Das Programm erzeugt eine Datei mit
      '           dem Namen der Eingabedatei und der Extension TMP.
      '           Anschließend wird diese Datei mit Hilfe des DOS
      '           Spoolers PRINT im Hintergrund ausgegeben.
      ' ********************************************************
      ' Variable definieren
    1 %on = 1: %off = 0
    2 nummer% = %off                          '! keine Zeilennummern
```

```
    3 zeile& = 0                              '! Zeilennummer Listing
    4 seite% = 1                              '! Seitennummer Listing
    5 maxzeile% = 60                          '! Zeilen pro Seite
    6 rechts% = 75                            '! rechter Rand
    7 links% = 0                              '! linker Rand
    8 spalte% = 0                             '! Einrückung

    9 indatei% = 1                            '! Dateinummer Eingabe
   10 outdatei% = 2                           '! Dateinummer Ausgabe

   11 ON ERROR GOTO fehler                    '! Fehlerausgang

      '############################################################
      '#                       Hauptprogramm                      #
      '############################################################

   12 GOSUB install                           '! PRINT installiert?

   13 kommando$ = COMMAND$                    '! Parameter?
   14 IF LEN (kommando$) = 0 THEN             '! User Mode?
   15  CLS                                    '! clear Screen

   16  PRINT "S P O O L                              (c) Born
Version 1.0"
   17  PRINT
   18  PRINT "Optionen [ /L=00 linker Rand         /R=75  rechter
Rand   ]
         "
   19  PRINT "          [ /Z=60 Zeilen pro Seite    /N
Zeilennumerierung ]
         "
   20  PRINT
   21  INPUT  "File     : ",filename$
   22  INPUT  "Optionen : ",options$
   23  PRINT
   24 ELSE

   25  ptr% = INSTR (kommando$,"/?")          '! Option /?
   26  IF ptr% <> 0 THEN                      '! Hilfsbildschirm
   27   PRINT "S P O O L                      (c) Born Version 1.0"
   28   PRINT
   29   PRINT "Aufruf: SPOOL <Filename> <Optionen>"
   30   PRINT
   31   PRINT "Optionen :"
   32   PRINT
   33   PRINT "  /L=00 setzt den linken Rand"
   34   PRINT "  /R=75 setzt den rechten Rand"
   35   PRINT "  /Z=60 setzt die Zeilenzahl pro Seite"
   36   PRINT "  /N    schaltet die Zeilennumerierung ein"
   37   PRINT
   38   PRINT "Das Programm gibt ein Listing der Datei aus, wobei
sich"
   39   PRINT "die Ränder und die Zahl der Zeilen einstellen läßt.
Die"
```

```
   40   PRINT "Ausgabe erfolgt in eine Datei, die per PRINT
ausgedruckt"
   41   PRINT "wird."
   42   PRINT
   43   SYSTEM
   44  END IF
   45 ||||      '! Kommandomodus
   46  ptr% = INSTR (kommando$,"/")         '! Optionen?
   47  IF ptr% = 0 THEN
   48   filename$ = kommando$               '! nur Filename
   49  ELSE
   50   filename$ = LEFT$(kommando$,ptr% -1) '! Filename separieren
   51   options$  = MID$(kommando$,ptr%)    '! Optionen separieren
   52  END IF
   53 END IF

   54 GOSUB parameter                      '! Optionen decodieren

   55 IF (rechts% < links%) or (maxzeile% < 10) THEN '! sinnlose
   56  PRINT                                        '! Einstellung
   57  PRINT "Bitte Randeinstellung neu setzen"     '! Fehlerexit
   58  END                                 '! Exit
   59 END IF

   60 IF filename$ = "" THEN               '! Leereingabe?
   61  PRINT
   62  PRINT "Der Dateiname fehlt"
   63  END                                 '! Exit
   64 END IF

   65 ptr% = INSTR(filename$,".")          '! hat Datei eine
Extension?
   66 IF ptr% > 0 THEN
   67  outfile$ = LEFT$(filename$,ptr%) + "TMP" '! Filename ohne
Extension
   68 ELSE
   69  outfile$ = filename$ + ".TMP"       '! Extension anhängen
   70 END IF

      '! falls TEMP-Verzeichnis existiert, lege Ausgabedatei dort
an.
   71 outfile$ = ENVIRON$("TEMP") + "\"+ outfile$

      ' prüfe ob Datei vorhanden, nein -> exit

   72 OPEN filename$ FOR INPUT AS #indatei%  '! Öffne Eingabedatei
   73 OPEN outfile$ FOR OUTPUT AS #outdatei% '! Öffne Ausgabedatei
   74 PRINT
   75 PRINT "Die Datei: ";filename$;" wird bearbeitet"

   76 GOSUB pageskip                       '! Seitenkopf ausgeben

   77 WHILE NOT (EOF(indatei%))            '! Datei sequentiell
lesen
   78  LINE INPUT #indatei%, linie$        '! lese Zeile
```

```
   79  GOSUB ausgabe                          '! drucke Zeile
   80 WEND

   81 CLOSE #indatei%                         '! Datei schließen
   82 CLOSE #outdatei%                        '! Datei schließen
   83 PRINT
   84 PRINT "Die Datei: ";filename$;" wird auf dem Drucker
ausgegeben "
   85 GOSUB spool                             '! aktiviere Print
   86 END


      '############################################################
      '#                    Hilfsroutinen                         #
      '############################################################

   87 fehler:
      '-----------------------------------------------------------
      '! Fehlerbehandlung in SPOOL
      '-----------------------------------------------------------

   88 IF ERR = 53 THEN
   89  PRINT "Die Datei ";filename$;" existiert nicht"
   90 ELSE
   91  PRINT "Fehler : ";ERR;" unbekannt"
   92  PRINT "Programmabbruch"
   93 END IF
   94 END                                  '! MSDOS Exit
   95 RETURN

   96 parameter:
      '-----------------------------------------------------------
      '! Decodiere die Eingabeoptionen
      '-----------------------------------------------------------
   97 ptr% = INSTR (options$,"/N")
   98 IF ptr% > 0 THEN nummer% = %on        '! Zeilennumerierung

   99 ptr% = INSTR (options$,"/Z=")
  100 IF ptr% > 0 THEN CALL getval (maxzeile%) '! Zeilen/Seite
  101 szeile% = maxzeile% + 1               '! Zeilennr Seite
wechseln

  102 ptr% = INSTR (options$,"/L=")
  103 IF ptr% > 0 THEN CALL getval (links%) '! linker Rand

  104 ptr% = INSTR (options$,"/R=")
  105 IF ptr% > 0 THEN CALL getval (rechts%) '! rechter Rand

  106 RETURN

  107 SUB getval (wert%)
      '-----------------------------------------------------------
      '! Decodiere den Eingabestring in eine Zahl
      '-----------------------------------------------------------
  108  SHARED options$, ptr%
```

```
  109  LOCAL i%

  110  ptr% = ptr% + 3                        '! ptr hinter /x=
  111  i% = 1
  112  WHILE ((ptr%+i%) =< LEN (options$)) and
(MID$(options$,ptr%+i%,1) <
       > " ")
  113   i% = i% + 1                           '! Ziffernzahl + 1
  114  WEND
  115  wert% = VAL(MID$(options$,ptr%,i%))  '! decodiere die Zahl
  116 END SUB

  117 pageskip:
      '-------------------------------------------------------
      '! Seitenvorschub mit Kopf (Dateiname, Datum, Seite)
      '-------------------------------------------------------
  118  IF szeile% < maxzeile% THEN RETURN   '! kein Seitenwechsel
!!
  119  IF seite% > 1 THEN                    '! 1. Seite k. Vorschub
  120   PRINT #outdatei%, CHR$(12)           '! Vorschub
  121   szeile% = 3                          '! 3 Kopfzeilen
  122  ELSE
  123   PRINT #outdatei%, "S P O O L "; options$; SPACE$(27);
  124   PRINT #outdatei%, "(c) Born Version 1.0"
  125   szeile% = 4                          '! 4 Kopfzeilen
  126  END IF
  127  PRINT #outdatei%, "Datei : ";filename$;"     Datum :
";DATE$;
  128  PRINT #outdatei%, "         Seite : "; seite%
  129  PRINT #outdatei%,
  130  INCR seite%

  131 RETURN


  132 ausgabe:
      '-------------------------------------------------------
      '! Ausgabe der eingelesenen Zeile auf dem Printer.
      '! rest% gibt an, wieviele Zeichen pro Zeile gedruckt
      '! werden dürfen. Ist die eingelesene Zeile länger, wird
      '! sie auf mehrere Ausgabezeilen aufgeteilt.
      '-------------------------------------------------------

  133  INCR zeile&                           '! Zeile im Listing + 1
  134  GOSUB pageskip                        '! Seitenvorschub?

  135  spalte% = links%                      '! linker Rand
  136  PRINT #outdatei%, SPACE$(spalte%);   '! auf linken Rand

  137  IF nummer% = %on THEN                 '! Zeilennumerierung?
  138   PRINT #outdatei%, USING "###### "; zeile&; '! Zeilennummer
drucken
  139   spalte% = spalte% + 7                '! Spalte 7 setzen
  140  END IF
```

```
  141  rest% = rechts% - spalte%            '! Restzeilenlänge
  142  GOSUB skipblank                      '! merke Blanks
  143  PRINT #outdatei%, LEFT$(linie$,rest%) '! Ausgabe Teilstring
  144  linie$ = MID$(linie$, rest% + 1)     '! Reststring
  145  INCR szeile%

  146  WHILE LEN(linie$) > rest%            '! String > Zeile
  147   GOSUB pageskip                      '! Seitenvorschub?
  148   PRINT #outdatei%, SPACE$(spalte%);  '! linker Rand
  149   PRINT #outdatei%, LEFT$(linie$,rest%) '! Teilstring
ausgeben
  150   linie$ = MID$(linie$,rest% + 1)     '! Reststring bestimmen
  151   INCR szeile%                        '! Zeile im Listing + 1
  152  WEND

  153  IF LEN(linie$) > 0 THEN
  154   GOSUB pageskip                      '! Seitenvorschub?
  155   PRINT #outdatei%, SPACE$(spalte%);linie$  '! Reststring
ausgeben
  156   INCR szeile%                        '! Zeile im Listing + 1
  157  END IF
  158 RETURN

  159 skipblank:
      '-----------------------------------------------------------
      '! zähle führende Blanks
      '-----------------------------------------------------------
  160 i% = 1
  161 WHILE (i% < LEN(linie$)) and (MID$(linie$,i%,1) = " ")
  162  INCR i%
  163  INCR spalte%
  164 WEND
  165 RETURN

  166 install:
      '-----------------------------------------------------------
      '! prüfe den PRINT Installationsstatus
      '-----------------------------------------------------------

  167  REG 1, &H0100                        '! AX = 0100 -> check
                                            '! Status
  168  CALL INTERRUPT &H2F                  '! Multiplexer INT

  169  IF (REG (1) and &H00FF) <> &HFF THEN  '! PRINT installiert?
  170   PRINT "Bitte installieren Sie zuerst das MS-DOS Programm:"
  171   PRINT "PRINT"
  172   PRINT
  173   END                                  '! Exit
  174  END IF

  175 RETURN

  176 spool:
      '-----------------------------------------------------------
      '! aktiviere PRINT um die Datei xxx.TMP zu drucken
```

```
    '-------------------------------------------------------

177 SHELL "PRINT " + outfile$              '! Spool Datei
178 RETURN
    ' ###### Programm Ende #########
179 END
```

Listing 2.2: SPOOL.BAS

PSLIST: Listings für PostScript-Drucker

Die in den vorhergehenden Abschnitten besprochenen Programme LISTER und SPOOL sind eine gute Hilfe bei der Ausgabe von Listings auf normalen Druckern (Matrix-, Tintenstrahl- und Laserdrucker). Mittlerweile sind aber auch Drucker auf Basis der PostScript-Technologie verfügbar und auf Grund der Preise für einen breiteren Anwenderkreis von Interesse. Es ist daher davon auszugehen, daß solche Geräte bei vielen Anwendern im wahrsten Sinne des Wortes herumstehen. Denn der Pferdefuß beim Einsatz von DOS besteht darin, daß dieses Betriebssystem PostScript nicht unterstützt. Unter DR-DOS 6.0 gibt es zwar ein entsprechendes Hilfsprogramm und auch Windows sowie verschiedene Anwendungen bieten entsprechende Treiber. Aber in PowerBASIC-Programmen kann der Anwender PostScript-Geräte nicht nutzen.

Aus diesem Ansatz heraus entstand die Idee, ein Programm zur Ausgabe von Listings auf PostScript-Geräten zu erstellen. In den vergangenen Jahren habe ich mich im Rahmen einer Zeitschriftenserie und in Form eines Buchprojektes ausgiebig mit der Sprache PostScript beschäftigt. Deshalb reizte mich das Thema, d.h. ich wollte herausfinden, wie aufwendig die Erstellung eines entsprechenden Treibers eigentlich ist. Als zweites finde ich es schon eine »affenscharfe« Sache, unter DOS ein Listing in verschiedenen Schriftgrößen von einem PostScript-Drcuker zu erhalten (wer kann schon damit aufwarten).

Sofern Sie über kein PostScript-Gerät verfügen, können Sie diesen Abschnitt übergehen. Es sei aber darauf hingewiesen, daß es mittlerweile für wenige hundert Mark Emulationsprogramme (z.B. GoScript, Freedom of Press, UltraScript) gibt, die PostScript-Programme unter DOS verarbeiten und die Ausgabe für Nadel-, Tintenstrahl- und Laserdrucker aufbereiten. Damit können Sie für wenig Geld in diese interessante Technik einsteigen. Ich benutze seit fast zwei Jahren einen solchen Emulator, auf dem auch das folgende Programm getestet wurde.

Auf die Frage, was denn eigentlich PostScript ist, kann ich nur mit einer Kurzeinführung aufwarten. Bei PostScript handelt es sich um eine Druckerbeschreibungssprache, die Anfang der achtziger Jahre durch die Firma Adobe definiert wurde. Eine auszugebende Seite wird durch einzelne Befehle für Text und Grafik beschrieben. Im Endgerät (Drucker, Fotobelichter etc.) setzt dann ein Interpreter diese Befehle in die Ausgabeseite um. Damit ist nicht mehr der Rechner sondern das Gerät für

die Aufbereitung der Ausgaben verantwortlich. Der Hauptvorteil besteht deshalb darin, daß Druckausgaben hersteller- und geräteunabhängig formuliert werden können.

Die Sprache selbst besitzt eine Reihe von Befehlen zur Ausgabe von Texten und Grafiken. Weiterhin lassen sich Berechnungen und Ablaufsteuerungen realisieren. Das nachfolgende Listing zeigt ein minimales PostScript-Programm, welches einen Text und einen waagerechten Strich auf einer Druckseite ausgibt. Alle Texte hinter dem Prozentzeichen (%) sind als Kommentare zu verstehen:

```
%--> einfaches PostScript-Programm

/Times-Roman findfont     % Font festlegen
20 scalefont setfont      % Font mit 20 Punkt Größe

newpath                   % Ausgabeseite öffnen
10 500 moveto             % Anfangskoordinaten
(Dies ist ein Text) show % Text ausgeben
10 300 moveto             % nächster Punkt
0.2 setlinewidth          % Linienbreite
0 setgray                 % Farbe schwarz
200 300 lineto            % Pfad (Linie) ziehen
stroke                    % Linie sichtbar machen
showpage                  % Seite ausgeben
% --> Ende
```

Listing 2.3: Einfaches PostScript-Programm

Die Sprache ist stackorientiert und benutzt das Prinzip der umgekehrten polnischen Notation, d.h. zuerst werden die Operanden und dann der Operator auf den Stack gelegt und abgearbeitet. Dies ist sehr gut bei einer Addition zu erkennen. Statt der Anweisung:

```
 3 + 4
```

wird die Form:

```
3 4 add
```

gewählt. Analoges findet sich auch in obigem Listing (z.B. *x y moveto*). Mit den ersten zwei Anweisungen wird eine Schrift (Font) vereinbart. Im Beispiel ist dies *Times-Roman* mit einer Größe von 20 Punkt. *Newpath* öffnet eine Ausgabeseite, auf die sich alle folgenden Ausgaben beziehen. Die Ausgaben werden dann in der Seite virtuell ausgegeben, wobei sich durchaus mehrere Objekte (Texte, Striche etc.) überlagern dürfen. Die Farbe des zuletzt gezeichneten Objektes bestimmt dann, ob die darunterliegenden Objekte verdeckt oder sichtbar sind. Die Anweisungen *x y moveto* verschieben den Ausgabecursor zum angegebenen Koordinatenpunkt, wobei der Ursprung in der linken unteren Ecke liegt. Alle Angaben werden in PostScript übrigens in Punkt (typographischer Punkt = ca. 1,44 mm) ausgegeben. Mit *(Text)show* wird der eigentliche Text, beginnend vom aktuellen Punkt, auf die Ausgabeseite projiziert. Der Befehl *x y lineto* zieht einen Pfad vom aktuellen Punkt zum angegebenen Ziel. Ein Pfad können Sie sich als Linie vorstellen, die mit unsichtbarer

Farbe gezeichnet wurde. Erst der Befehl *stroke* füllt diese Linie mit der gesetzten Farbe (`0 setgray`) und der Strichstärke (`0.2 setlinewidth`) aus. Alle Anweisungen beziehen sich auf eine virtuelle Ausgabeseite. Die Seite wird erst mit der Anweisung *showpage* auf dem Drucker ausgegeben.

Sofern Sie sich näher für die Sprache PostScript interessieren, möchte ich Sie auf die Titel /3 und /4 im Literaturverzeichnis hinweisen.

Der Entwurf

Die Benutzeroberfläche soll sich an den Möglichkeiten von SPOOL anlehnen. Wird das Programm mit der Eingabe:

```
PSLIST
```

aufgerufen, erscheint die Kopfmeldung:

```
P S L I S T                          (c) Born Version 1.0

Optionen  [ /L=10  linker Rand      /R=500  rechter Rand    ]
          [ /O=700 oberer Rand      /U=100  unterer Rand    ]
          [ /F=10  Fontgröße/Punkt  /N      Numerierung Ein ]

File     :
Optionen :
```

Bild 2.7: Kopfmeldung des Programmes PSLIST

Im Gegensatz zu den Programmen LISTER und SPOOL sind die Optionen etwas verändert. Die Zahl der Zeilen pro Seite wird hier nicht mehr angegeben. Dies ist auch nicht nötig, da die Abmessungen des Druckbereiches einer Seite in Punkt spezifiziert werden. Zusammen mit der Fontgröße ergibt sich dann automatisch die Zahl der Zeilen pro Seite. Die oben angegebenen Werte geben die Standardeinstellung wieder. Mit der Option /N läßt sich jedoch wie gewohnt die Zeilennumerierung im Listing einschalten.

Als Dateiname darf jede gültige MS-DOS-Dateibezeichnung einschließlich Laufwerks- und Pfadbezeichnung verwendet werden. Die Abfrage der »Optionen« soll nach der Eingabe des Dateinamens erfolgen. Die Kopfmeldung zeigt mögliche Eingaben für diese Optionen. Die Optionen dürfen zwar in beliebiger Reihenfolge eingegeben werden, das Format ist jedoch gemäß obigen Angaben definiert (/ gefolgt von Großbuchstaben). Nachfolgend finden Sie einige gültige Formatangaben.

```
/N
/L=10 /O=655 /R=550 /U=50 /N
/L=5
/N /R=600
```

Fehlerhafte Eingaben (z.B. linker Rand größer als rechter Rand etc.) sind durch das Programm abzufangen. In diesem Fall erscheint die Fehlermeldung:

```
Bitte Randeinstellung neu setzen
```

Wird kein Dateiname eingegeben, bricht das Programm mit der folgenden Meldung ab:

```
Der Dateiname fehlt
```

Existiert die angegebene Datei nicht, endet das Programm ebenfalls mit der Meldung:

```
Die Datei  <Name>  existiert nicht
```

Wird eine Datei gefunden, beginnt die Ausgabe mit dem Hinweis:

```
Die Datei <Name> wird bearbeitet
```

Name steht dabei für den eingegebenen Dateinamen. Nach Beendigung der Ausgabe in die PostScript-Datei erscheint die Meldung:

```
Die <Datei> wurde im aktuellen Verzeichnis erzeugt
```

Sie können dann die Datei per COPY oder PRINT auf dem PostScript-Gerät ausgeben.

Die eigentliche Ausgabe des Listings entspricht der gewohnten Form, wie sie auch durch das Programm LISTER erzeugt wird. Lediglich die Schriftgröße läßt sich durch die Option /O in Schritten zu einem Punkt variieren. Sinnvolle Angaben dürften dabei zwischen 8 Punkt und 16 Punkt liegen. Bei zu großen Schrifttypen passen die Ausgabezeilen nicht mehr auf eine Seite. Aus Gründen der Vereinfachung begrenzt das Programm die Zeichen pro Zeile auf 75.

Alternativ lassen sich Dateiname und Optionen mit in der Kommandozeile angeben:

```
PSLIST <Filename> <Optionen>
```

Dies ist insbesondere in Batchdateien interessant. Weiterhin kann eine Online-Hilfe mit folgendem Kommando aufgerufen werden:

```
PSLIST /?
```

Auf dem Bildschirm erscheint folgende Meldung:

```
P S L I S T                    (c) Born Version 1.0

Aufruf: PSLIST <Filename> <Optionen>

Optionen :

  /L=10  setzt den linken Rand in Punkt
  /R=500 setzt den rechten Rand
  /O=700 setzt den oberen Rand
  /U=100 setzt den unteren Rand
  /F=10  setzt die Fontgröße in Punkt
  /N     Numerierung ein
```

```
Das Programm gibt ein Listing als PostScript-Datei
mit der Extension xxxx.PS aus, wobei xxxx dem File-
namen entspricht. Die Ergebnisdatei kann dann auf einem
PostScript-Gerät ausgegeben werden.
```

Bild 2.8: Online-Hilfe von PSLIST

Die Implementierung

Bezüglich der Implementierung bietet es sich an, möglichst viele Teile von LISTER oder SPOOL zu übernehmen. Gerade die Decodierung der Eingaben sowie die Formatierung der Ausgaben und Speicherung in eine Textdatei ist in SPOOL bereits gelöst. Lediglich die Teile zur Ansteuerung von PRINT können entfernt werden. Die Beschreibung der einzelnen Module beschränkt sich deshalb auch nur auf die Änderungen zu SPOOL.

Offen ist allerdings noch die Frage, wie die Ausgaben der PostScript-Befehle erfolgen sollen. Hier bieten sich zwei Möglichkeiten an:

- Das Programm generiert alle PostScript-Anweisungen einschließlich der Positionsangaben innerhalb der Seite. Dies setzt die Berechnungen des Layouts der Seite in PSLIST voraus.
- Das Programm generiert lediglich die Anweisungen zur Ausgabe einzelner Zeilen in PostScript. Die Positionierung des Zeilenanfanges erfolgt dann innerhalb des PostScript-Interpreters. Dies setzt aber voraus, daß in PostScript ein entsprechendes Programm vorliegt.

Beide Varianten haben ihre Vor- und Nachteile. Ich habe mich für die zweite Variante entschieden. Hierfür gab es eigentlich zwei Gründe:

- Einmal müssen bestimmte Definitionen (z.B. Umlaute) sowieso in einem PostScript-Programm vorgenommen werden.
- Zweitens stehen mir aus dem erwähnten Buchprojekt verschiedene PostScript-Programme zur Textausgabe zur Verfügung.

Auf Grund dieser Entscheidung konnte ich das Programm in rund vier Stunden implementieren. Zur Vorgehensweise nachfolgend einige Erläuterungen.

PostScript umfaßt eine eigene Sprache, mit der sich Berechnungen und Programmabläufe formulieren lassen. Damit stehen auch Prozeduren und Schleifen zur Verfügung. Somit läßt sich ein PostScript-Programm in zwei Teile gliedern:

- Teil 1 mit den Definitionen und Ausgabeprozeduren.
- Teil 2 mit den eigentlichen Textanweisungen.

Da Teil 1 sich kaum ändert, kann er als Vorspann vor den eigentlichen Text geschrieben werden. Dann muß lediglich der auszugebende Text in

der PostScript-Notation angefügt werden. Diese Aufgabe kann ein kleines Programm leicht erfüllten. Im Grunde sind lediglich alle Zeilen in Anweisungen wie *(..) show* zu packen. (Ganz so einfach ist es zwar nicht, aber der Ansatz ist praktikabel.)

Bleibt noch die Frage, wie Teil 1 des PostScript-Programmes aufzubauen ist. Hier lassen sich folgende Punkte identifizieren:

- Die Abmessungen der müssen definiert werden.
- Der Font muß selektiert und die gewünschte Größe eingestellt werden.
- Da in den Standardfonts von PostScript keine Umlaute definiert sind, müssen diese in der Fontdefinition aktiviert werden.
- Es sind einige Routinen zur formatierten Ausgabe der übergebenen Texte zu integrieren.

Die Abmessungen der Druckseite werden durch den Benutzer beim Aufruf des Programmes definiert (bei fehlenden Angaben werden Standardwerte verwendet). Diese Definitionen müssen also durch das Programm PSLIST generiert werden. Das gleiche gilt für die Fontgröße.

Alle anderen Angaben können jedoch bereits in eine statische Textdatei gespeichert werden. Diese Textdatei muß dann lediglich vor die Textausgaben gestellt werden. Nachfolgendes Listing zeigt den Aufbau des Vorspanns der PostScript-Datei. Dieser Vorspann ist in der Datei HEADER.PS gespeichert.

```
%---> File: HEADER.PS (c) G. Born
%---> Definitionen für Textausgabe mit Umlauten
%--> definiere Konstanten
/LW { CH CH 3 div add } def % Zeilenabstand
/Blank ( ) def         % Leerzeichen

/Buff  12 dict def     % lokales Dictionary

/endtest  % Test ob Seitenende erreicht ist
{
 dup                   % Y duplizieren
 BM lt                 % unterer Rand?
 {
  showpage             % ja-> neue Seite
  pop TM               % neue Y-Koordinate
 } if                  % auf 1. Zeile
} def

/newpage               % Seitenvorschub
{
  showpage             % Seite wechseln
  LM TM moveto         % Startpunkt neue Seite
} def

/crlf                  % Zeilenvorschub
```

```
{
  LM                   % linker Rand
  currentpoint LW sub % y-dy
  exch pop             % X entfernen
  endtest              % neue Seite?
  moveto               % Zeilenanfang
} def

/printword             % Ausgabe eines Wortes
{
  dup                  % String duplizieren
  stringwidth pop      % Länge Text
  currentpoint         % aktuelle Ausgabe
  pop add LM add       % Ausgabelänge berech.
  RM gt
  { crlf } if          % Zeilenvorschub
  show                 % Text ausgeben
  ( ) show             % Leerzeichen
} def

/printline             % Ausgabe eines Satzes
{                      % Beginn der Prozedur
 {                     % Beginn der Schleife
   Blank search        % separiere Wort
   { printword pop }   % Ausgabe Wort
   { printword exit } % Ausgabe Resttext
   ifelse
 } loop                % loop alle Wörter
  crlf                 % Zeilenvorschub
} def

/Redefine               % Umcodierung Font
{ Buff begin            % lokales Dictionary
  /NCodeName exch def   % Hilfsvariable
  /NFontName exch def
  /AFontName exch def

  /AFontDict            % suche alten Font
   AFontName findfont def

  /NeuFont AFontDict    % neue Dictionary schaffen
   maxlength dict def

  AFontDict             % kopiere alten Font
  { exch dup /FID ne    % bis auf FID-Feld
   { dup /Encoding eq
    { exch dup length array copy
      NeuFont 3 1 roll put
    }
    { exch NeuFont 3 1 roll put
    } ifelse
   } { pop pop } ifelse
  } forall

 NeuFont
```

```
 /FontName NFontName put % setze neuen Namen
 NCodeName aload pop     % Werte laden
 NCodeName length 2 idiv % und eintragen
 { NeuFont /Encoding get
   3 1 roll put
 } repeat

 NFontName NeuFont       % definiere neuen Font
 definefont pop
 end
} def    % Ende der Prozedur /Redefine

%--> Hauptprogramm

/Umlaute              % Feld mit Umlautdefinitionen
[ 8#201 /udieresis    % ü
  8#204 /adieresis    % ä
  8#216 /Adieresis    % Ä
  8#224 /odieresis    % ö
  8#231 /Odieresis    % Ö
  8#232 /Udieresis    % Ü
  8#341 /germandbls   % ß
] def

%--> Umcodierung des Fonts
%--> Hier ist der Name des Basis-Fonts einzutragen:
%--> z.B. /Times-Roman oder /AvantGarde oder /Courier
/Courier
%--> Umdefinition des Urfonts in Font mit Umlauten
/Neu-Font-Deutsch            % neuer Font
Umlaute Redefine

/Neu-Font-Deutsch findfont % Font selektieren
CH scalefont setfont   % und initialisieren

LM TM  moveto          % Anfangspunkt

%----> Hier schließt sich der auszugebende Text an
```

Listing 2.4: HEADER.PS

Der Aufbau des Programmes setzt voraus, daß vor den ersten Zeilen bereits die Definitionen für die Abmessungen des Druckbereiches und der Fontgröße stehen. Diese Anweisungen müssen direkt im Programm erzeugt werden:

```
/RM 500 def
/LM 10  def
....
```

Sie sollten sich einmal eine von PSLIST generierte Ausgabedatei ansehen, um den genauen Aufbau des Headers zu analysieren.

Die Prozedur printline (in HEADER.PS) übernimmt einen Textstring und sorgt für die Ausgabe des Inhalts auf dem Drucker. Dabei gilt folgende Syntax:

```
(Text) printline
```

Die Formatierung, der Zeilenvorschub etc. erfolgt dann direkt im PostScript-Programm. Die Prozedur */crlf* führt zum Beispiel einen Zeilenvorschub durch.

Ein Großteil des Programmes widmet sich der Neudefinition des Ausgabefonts. PostScript stellt in den Fontdateien standardmäßig keine Umlaute zur Verfügung, obwohl die Zeichen bereits im Font definiert sind. Die Codes dieser Sonderzeichen müssen daher explizit definiert werden. Hierzu sind die Prozedur */redefine* und die folgenden Definitionen zuständig:

```
/Umlaute              % Feld mit Umlautdefinitionen
[ 8#201 /udieresis    % ü
  8#204 /adieresis    % ä
  8#216 /Adieresis    % Ä
  8#224 /odieresis    % ö
  8#231 /Odieresis    % Ö
  8#232 /Udieresis    % Ü
  8#341 /germandbls   % ß
] def
```

Auf eine genaue Beschreibung der Wirkungsweise muß ich an dieser Stelle verzichten, da diese den Umfang des Buches sprengen würde. Der interessiert Leser sei auf /4/ verwiesen, wo das Programm in erweiterter Form vorliegt und die Wirkungsweise schrittweise erläutert wird. Die Datei HEADER.PS findet sich auf der Begleitdiskette. Sie muß sich beim Aufruf von PSLIST im Verzeichnis von PSLIST oder im aktuellen Verzeichnis befinden.

Nach dieser Vorarbeit kann PSLIST sich bei der Ausgabe darauf beschränken, die Eingabezeilen zu lesen, gegebenenfalls auf 75 Zeichen pro Zeile zu begrenzen und dann die Zeichen in der PostScript-Syntax an die Prozedur *printline* zu übergeben. Hierzu ist der Text in runde Klammern zu stellen und der Name der Prozedur anzuhängen (z.B. *(dies ist ein Text) printline*). Probleme treten lediglich auf, falls im Text selbst diese Klammern auftreten. PostScript sieht in diesem Fall eine Markierung mit dem Zeichen »\« vor. Dies ist beid er Ausgabe zu berücksichtigen.

Als Font für die Ausgabe von Listings würde ich Courier vorschlagen. Dieser Schriftsatz besitzt keine proportionalen Zeichenbreite, d.h. es ergibt sich eine Ausgabe wie auf einem Matrixdrucker. Die Zeichen stehen spaltenweise untereinander. Die Fontgröße und die Abmessungen der Druckseite (oben (TM) - unten (BM)) ergeben dann die Zahl der Zeilen pro Seite. Dieser Wert dient PSLIST zur Formatierung der Ausgabe, d.h. dann wird ein Seitenkopf generiert. Der eigentliche Umbruch der Seite erfolgt dagegen in der PostScript-Prozedur */endtest*.

Hauptmodul

Im Hauptprogramm werden die Eingabeparameter eingelesen und decodiert. Dann öffnet das Programm die Eingabe- und Ausgabedatei. Der Aufruf des Unterprogrammes *vorspann* sorgt dafür, daß Teil 1 der Ausgabedatei generiert wird. In einer Schleife wird dann die Eingabedatei zeilenweise gelesen und über das Unterprogramm *ausgabe* in PostScript-Notation ausgegeben.

vorspann

Das Programm generiert zuerst die Konstanten für die Abmessungen des Druckbereiches in der Ausgabedatei. Dann wird noch die Fontgröße definiert. Anschließend kopiert *vorspann* den Inhalt der Datei HEADER.PS in die Ausgabedatei. Daran schließt sich später das auszugebende Listing an.

Ausgabe

Dieses Modul wurde aus dem Programm SPOOL übernommen. Erwähnt werden soll hier eine kleine Besonderheit. Der Name der Ausgabedatei wird aus dem Namen der Quelldatei bestimmt. Dabei wird lediglich die Extension entfernt und durch die Endung »PS ersetzt. Bei der Ermittlung des Names für die Zieldatei wird davon ausgegangen, dass keine Laufwerks- und Pfadangaben benutzten werden. Das Programm formatiert die Eingabezeile wie in den Modulen LISTER und SPOOL und fügt weiterhin die Steueranweisungen für PostScript hinzu. Der Aufbau der Steueranweisungen wurde oben bereits beschrieben und kann in der Ergebnisdatei leicht nachvollzogen werden.

Weitere Einzelheiten sind nachfolgendem Listing zu entnehmen.

Erweiterungsvorschläge

Das Programm überschreibt die Ausgabedatei ohne Warnung. Hier könnte eine entsprechende Bedienerabfrage eingefügt werden. Weiterhin könnte der Fontname selektierbar sein. Dazu muß lediglich eine Option eingelesen und die entsprechende Definition an den Anfang der PostScript-Datei geschrieben werden. Ein weiterer Punkt ist die Berechnung der Zeichen pro Zeile in Abhängigkeit von der Fontgröße und dem Druckbereich. Zur Zeit werden maximal 75 Zeichen pro Zeile ausgegeben, was bei entsprechenden Fontgrößen zu Problemen führt. Hier ist sicherlich Abhilfe denkbar.

```
X R E F   /Z=55                          (c) Born Version 1.0
Datei : pslist.bas      Datum : 05-13-1992         Seite :  1

 Zeile     Anweisung

      ' ******************************************************
```

```
      ' File       : PSLIST.BAS
      ' Vers.      : 1.0
      ' Last Edit  : 30. 4.92
      ' Autor      : G. Born
      ' File I/O   : INPUT, OUTPUT, FILE, PRINTER
      ' Progr. Spr.: POWERBASIC
      ' Betr. Sys. : DOS 2.1 - 5.0
      ' Funktion: Das Programm dient zur Ausgabe von Listings
      '           auf PostScript-Geräten. Der Text wird in eine
      '           Datei mit der Extension .PS konvertiert. Es
      '           lassen sich beliebige Textdateien mit diesem
      '           Programm aufbereiten. Die Steueranweisungen
      '           für den Interpreter (Seitenumbruch, Randein-
      '           stellung etc.) werden über PSLIST direkt und
      '           über die Datei HEADER.PS generiert.
      '
      ' Aufruf:   PSLIST Filename <Optionen>
      '           Optionen:  /N   Zeilennumerierung ein   [AUS]
      '                      /L=xx linker Rand in Punkten [ 100]
      '                      /R=xx rechter Rand           [ 500]
      '                      /O=xx oberer Rand            [ 700]
      '                      /U=xx unterer Rand           [ 100]
      '                      /F=xx Fontgröße in Punkt     [  10]
      '
      '           Die Werte in [] geben die Standardeinstellung
      '           wieder. Wird das Programm ohne Parameter aufge-
      '           rufen, sind Dateiname und Optionen explizit ab-
      '           zufragen. Mit dem Aufruf:
      '
      '              PSLIST /?
      '
      '           wird ein Hilfsbildschirm ausgegeben.
      ' ********************************************************
      ' Variable definieren
    1 %on = 1: %off = 0
    2 nummer% = %off                        '! keine Zeilennummern
    3 zeile& = 0                            '! Zeilennummer Listing
    4 seite% = 1                            '! Seitennummer Listing

    5 rechts% = 500                         '! rechter Rand in
Punkt
    6 links% = 10                           '! linker Rand
    7 oben% = 700                           '! oberer Rand
    8 unten% = 100                          '! unterer Rand
    9 font% = 10                            '! Fontgröße
   10 rmargin% = 75                         '! 75 Zeichen pro Zeile

   11 indatei% = 1                          '! Dateinummer Eingabe
   12 indatei2% = 3                         '! Dateinummer Header
   13 outdatei% = 2                         '! Dateinummer Ausgabe
   14 errorname$ = ""

   15 ON ERROR GOTO fehler                  '! Fehlerausgang

      '############################################################
```

```
      '#                        Hauptprogramm                          #
      '###############################################################

   16 kommando$ = COMMAND$                    '! Parameter?
   17 IF LEN (kommando$) = 0 THEN             '! User Mode?
   18  CLS                                    '! clear Screen

   19  PRINT "P S L I S T                              (c) Born
Version 1.0"
   20  PRINT
   21  PRINT "Optionen [ /L=10  linker Rand         /R=500 rechter
Rand
       ]"
   22  PRINT "          [ /O=700 oberer Rand         /U=100 unterer
Rand
       ]"
   23  PRINT "          [ /F=10  Fontgröße/Punkt     /N
Numerierung Ein
       ]"
   24  PRINT
   25  INPUT  "File     : ",filename$
   26  INPUT  "Optionen : ",options$
   27  PRINT
   28 ELSE

   29  ptr% = INSTR (kommando$,"/?")          '! Option /?
   30  IF ptr% <> 0 THEN                      '! Hilfsbildschirm
   31   PRINT "P S L I S T                    (c) Born Version 1.0"
   32   PRINT
   33   PRINT "Aufruf: PSLIST <Filename> <Optionen>"
   34   PRINT
   35   PRINT "Optionen :"
   36   PRINT
   37   PRINT "  /L=10  setzt den linken Rand in Punkt"
   38   PRINT "  /R=500 setzt den rechten Rand"
   39   PRINT "  /O=700 setzt den oberen Rand"
   40   PRINT "  /U=100 setzt den unteren Rand"
   41   PRINT "  /F=10  setzt die Fontgröße in Punkt"
   42   PRINT "  /N     Numerierung ein"
   43   PRINT
   44   PRINT "Das Programm gibt ein Listing als PostScript-Datei"
   45   PRINT "mit der Extension xxxx.PS aus, wobei xxxx dem File-"
   46   PRINT "namen entspricht. Die Ergebnisdatei kann dann auf
einem"
   47   PRINT "PostScript-Gerät ausgegeben werden."
   48   PRINT
   49   SYSTEM
   50  END IF
   51       '! Kommando-Modus
   52  ptr% = INSTR (kommando$,"/")          '! Optionen?
   53  IF ptr% = 0 THEN
   54   filename$ = kommando$                '! nur Filename
   55  ELSE
   56   filename$ = LEFT$(kommando$,ptr% -1) '! Filename separieren
   57   options$  = MID$(kommando$,ptr%)     '! Optionen separieren
```

```
   58  END IF
   59 END IF

   60 GOSUB parameter                          '! Optionen decodieren

   61 IF (rechts% < links%) or (oben% < unten%) THEN  '! sinnlose
   62  PRINT                                          '! Einstellung
   63  PRINT "Bitte Randeinstellung neu setzen"     '! Fehlerexit
   64  END                                     '! Exit
   65 END IF

   66 IF filename$ = "" THEN                   '! Leereingabe?
   67  PRINT
   68  PRINT "Der Dateiname fehlt"
   69  END                                     '! Exit
   70 END IF

   71 ptr% = INSTR(filename$,".")              '! hat Datei eine
Extension?
   72 IF ptr% > 0 THEN
   73  outfile$ = LEFT$(filename$,ptr%) + "PS" '! Filename ohne
Extension
   74 ELSE
   75  outfile$ = filename$ + ".PS"            '! Extension anhängen
   76 END IF

      ' prüfe ob Datei vorhanden, nein -> exit

   77 errorname$ = filename$

   78 OPEN filename$ FOR INPUT AS #indatei%  '! Öffne Eingabedatei
   79 OPEN outfile$ FOR OUTPUT AS #outdatei% '! Öffne Ausgabedatei
   80 PRINT
   81 PRINT "Die Datei: ";filename$;" wird bearbeitet"

   82 GOSUB vorspann                           '! Vorspann generieren

   83 WHILE NOT (EOF(indatei%))                '! Datei sequentiell
lesen
   84  LINE INPUT #indatei%, linie$            '! lese Zeile
       '! scan line auf (..) und wandle in \ um
   85  linie1$ = ""
   86  FOR i% = 1 to LEN(linie$)
   87   zchn$ = MID$(linie$,i%,1)
   88   IF (zchn$ = "(") or (zchn$ = ")") THEN
   89    linie1$ = linie1$ + "\"
   90   END IF
   91   linie1$ = linie1$ + zchn$
   92  NEXT i%
   93  linie$ = linie1$
   94  GOSUB ausgabe                           '! schreibe Zeile
   95 WEND

   96 PRINT #outdatei%, "showpage"             '! Abschluß PS-Datei
   97 PRINT #outdatei%, "% END of File"
```

```
   98 CLOSE #indatei%                          '! Datei schließen
   99 CLOSE #outdatei%                         '! Datei schließen
  100 PRINT
  101 PRINT "Die Datei: ";filename$;" wurde im aktuellen
Verzeichnis erzeugt"
  102 END

      '############################################################
      '#                    Hilfsroutinen                         #
      '############################################################

  103 fehler:
      '------------------------------------------------------------
      '! Fehlerbehandlung in PSLIST
      '------------------------------------------------------------

  104 IF ERR = 53 THEN
  105  PRINT "Die Datei ";errorname$;" existiert nicht"
  106 ELSE
  107  PRINT "Fehler : ";ERR;" unbekannt"
  108  PRINT "Programmabbruch"
  109 END IF
  110 END                                  '! MSDOS Exit
  111 RETURN

  112 parameter:
      '------------------------------------------------------------
      '! Decodiere die Eingabeoptionen
      '------------------------------------------------------------

  113 options$ = UCASE$(options$)

  114 ptr% = INSTR (options$,"/N")
  115 IF ptr% > 0 THEN Nummer%=%on          '! Zeilennumerierung

  116 ptr% = INSTR (options$,"/L=")
  117 IF ptr% > 0 THEN CALL getval (links%) '! linker Rand

  118 ptr% = INSTR (options$,"/R=")
  119 IF ptr% > 0 THEN CALL getval (rechts%) '! rechter Rand

  120 ptr% = INSTR (options$,"/O=")
  121 IF ptr% > 0 THEN CALL getval (oben%)   '! oberer Rand

  122 ptr% = INSTR (options$,"/U=")
  123 IF ptr% > 0 THEN CALL getval (unten%)  '! unterer Rand

  124 ptr% = INSTR (options$,"/F=")
  125 IF ptr% > 0 THEN CALL getval (font%)   '! Fontgröße
  126 IF font% > 20 THEN font% = 20          '! max. 20 Punkt

      '! berechne Zeilenzahl aus Seiten- und Fontgröße
  127 maxzeile% = (oben% - unten%) / (font% + font% / 3)
  128 szeile% = maxzeile%+1
```

```
  129 RETURN

  130 SUB getval (wert%)
      '-----------------------------------------------------------
      '! Decodiere den Eingabestring in eine Zahl
      '-----------------------------------------------------------
  131  SHARED options$, ptr%
  132  LOCAL i%

  133  ptr% = ptr% + 3                      '! ptr hinter /x=
  134  i% = 1
  135  WHILE ((ptr%+i%) =< LEN (options$)) and
(MID$(options$,ptr%+i%,1) <> " ")
  136   i% = i% + 1                         '! Ziffernzahl + 1
  137  WEND
  138  wert% = VAL(MID$(options$,ptr%,i%))  '! decodiere die Zahl
  139 END SUB

  140 pageskip:
      '-----------------------------------------------------------
      '! Seitenvorschub mit Kopf (Dateiname, Datum, Seite)
      '-----------------------------------------------------------
  141  IF szeile% < maxzeile% THEN RETURN   '! kein Seitenwechsel
!!

  142  IF seite% > 1 THEN                   '! 1. Seite k. Vorschub
  143   PRINT #outdatei%, "newpage"         '! Vorschub
  144   szeile% = 3                         '! 3 Kopfzeilen
  145  ELSE
  146   PRINT #outdatei%, "(P S L I S T    "; options$; SPACE$(27);
  147   PRINT #outdatei%, "\(c\) Born Version 1.0) printline"
  148   szeile% = 4                         '! 4 Kopfzeilen
  149  END IF
  150  PRINT #outdatei%, "(Datei : ";filename$;"     Datum :
";DATE$;
  151  PRINT #outdatei%, "         Seite : "; seite%; ") printline"
  152  PRINT #outdatei%,
  153  INCR seite%

  154 RETURN

  155 ausgabe:
      '-----------------------------------------------------------
      '! Ausgabe der eingelesenen Zeile in der Datei
      '! rest% gibt an, wieviele Zeichen pro Zeile gedruckt
      '! werden dürfen. Ist die eingelesene Zeile länger, wird
      '! sie auf mehrere Ausgabezeilen aufgeteilt.
      '-----------------------------------------------------------

  156  INCR zeile&                          '! Zeile im Listing + 1
  157  GOSUB pageskip                       '! Seitenvorschub?

  158  spalte% = 0                          '! linker Rand (immer
0)
  159  PRINT #outdatei%, "(";               '! Startklammer
```

```
  160  IF nummer% = %on THEN                '! Zeilennumerierung?
  161   PRINT #outdatei%, USING "###### "; zeile&; '! Zeilennummer
drucken
  162   spalte% = spalte% + 7               '! Spalte 7 setzen
  163  END IF

  164  rest% = rmargin% - spalte%           '! Restzeilenlänge
  165  GOSUB skipblank                      '! merke Blanks
  166  PRINT #outdatei%, LEFT$(linie$,rest%); '! Ausgabe Teilstring
  167  PRINT #outdatei%, ") printline"
  168  linie$ = MID$(linie$, rest% + 1)     '! Reststring
  169  INCR szeile%

  170  WHILE LEN(linie$) > rest%            '! String > Zeile
  171   GOSUB pageskip                      '! Seitenvorschub?
  172   PRINT #outdatei%, "("; SPACE$(spalte%); '! linker Rand
  173   PRINT #outdatei%, LEFT$(linie$,rest%);  '! Teilstring
ausgeben
  174   PRINT #outdatei%, ") printline"
  175   linie$ = MID$(linie$,rest% + 1)     '! Reststring bestimmen
  176   INCR szeile%                        '! Zeile im Listing + 1
  177  WEND

  178  IF LEN(linie$) > 0 THEN
  179   GOSUB pageskip                      '! Seitenvorschub?
  180   PRINT #outdatei%, "("; SPACE$(spalte%);linie$; '!
Reststring ausgeben
  181   PRINT #outdatei%, ") printline"
  182   INCR szeile%                        '! Zeile im Listing + 1
  183  END IF
  184 RETURN
  185 skipblank:
      '-----------------------------------------------------------
      '! zähle führende Blanks
      '-----------------------------------------------------------

  186 i% = 1
  187 WHILE (i% < LEN(linie$)) and (MID$(linie$,i%,1) = " ")
  188  INCR i%
  189  INCR spalte%
  190 WEND
  191 RETURN

  192 vorspann:
      '-----------------------------------------------------------
      '! generiere Vorspann mit PostScript-Anweisungen
      '-----------------------------------------------------------
  193 errorname$ = "HEADER.PS"

  194 OPEN "HEADER.PS" FOR INPUT AS #indatei2%  '! Header öffnen
  195 PRINT "Generiere Fileheader"

  196 PRINT #outdatei%, "%%!PS-Adobe-2.0 EPSF-1.2"
  197 PRINT #outdatei%, "%%Title: ",filename$
```

```
  198 PRINT #outdatei%, "%%Creator: PSLIST 1.0 (c) Born G."
  199 PRINT #outdatei%, "%%EndComments"
  200 PRINT #outdatei%, ""
  201 PRINT #outdatei%, "%%BeginSetup"
  202 PRINT #outdatei%, "/LM ";links%;" def        % linker Rand"
  203 PRINT #outdatei%, "/RM ";rechts%;" def       % rechter Rand"
  204 PRINT #outdatei%, "/TM ";oben%;" def         % oberer Rand"
  205 PRINT #outdatei%, "/BM ";unten%;" def        % unterer Rand"
  206 PRINT #outdatei%, "/CH ";font%;" def         % Fontgröße"
  207 PRINT #outdatei%, ""


  208 WHILE NOT (EOF(indatei2%))            '! Datei sequentiell
lesen
  209  LINE INPUT #indatei2%, linie$        '! lese Zeile
  210  PRINT #outdatei%, linie$             '! schreibe
  211 WEND

  212 PRINT #outdatei%, "%%EndSetup"
  213 PRINT #outdatei%, ""

  214 CLOSE #indatei2%

  215 RETURN

  216 END
```

Listing 2.5: PSLIST.BAS

XREF: Ein Generator zur Erzeugung von Querverweislisten

Die bisher besprochenen Programme zur formatierten Ausgabe von PowerBASIC-Programmlistings unterstützen die Software-Entwicklung nur zum Teil. Zwar lassen sich damit übersichtlich strukturierte Dokumentationen des Programmes erreichen. Aber dies reicht für die tägliche Entwicklerpraxis nicht aus. Wer sich mit den Themen »Softwareentwicklung und PowerBASIC« beschäftig, dem sind wohl folgende Probleme bekannt:

- Es wurde die Endung % ! # & an einer Variablen vergessen, so daß PowerBASIC einen neuen Typ zuweist.
- Das Programm funktioniert nicht oder liefert falsche Ergebnisse, weil ein Variablenname irrtümlich mehrfach belegt oder falsch geschrieben wurde.
- Eine Variable soll in ihrer Bedeutung verändert werden. Das Programmlisting muß nun nach dieser Variable durchsucht werden, um sicherzustellen, daß keine Nebeneffekte auftreten.
- Ein Label soll umbenannt werden. Wo taucht dieser Name überall

im Programm auf?

- Das Programm enthält Variablen, die durch Änderungen überflüssig wurden. Wie können solche »Leichen« gefunden werden?
- Es soll herausgefunden werden, wo überall ein Unterprogramm aufgerufen wird.

Diese und ähnliche Fragen treten häufig bei der Programmentwicklung auf, wobei das Listing dann manuell durchsucht wird. Dies ist nicht nur zeitaufwendig, sondern auch sehr fehleranfällig.

Professionelle Compiler und Assembler unterstützen den Entwickler bei dieser Aufgabe. Sie erzeugen ein Listing mit Zeilennummern und generieren anschließend eine Querverweisliste (Cross-Referenz-Liste). Diese enthält alle Variablen in alphabetischer Reihenfolge sowie die Zeilennummern, in denen die jeweilige Variablen auftritt.

Leider bietet PowerBASIC (wie viele andere populäre Produkte) diesen Service nicht. Nachfolgend wird deshalb ein einfacher Querverweisgenerator für PowerBASIC-Programme beschrieben.

Die Spezifikation

Vor der Entwicklung beginnen wir wieder mit der Festlegung der Anforderungen. Der Generator soll ein beliebiges PowerBASIC-Quellprogramm einlesen und mit einer Querverweisliste versehen wieder ausgeben.

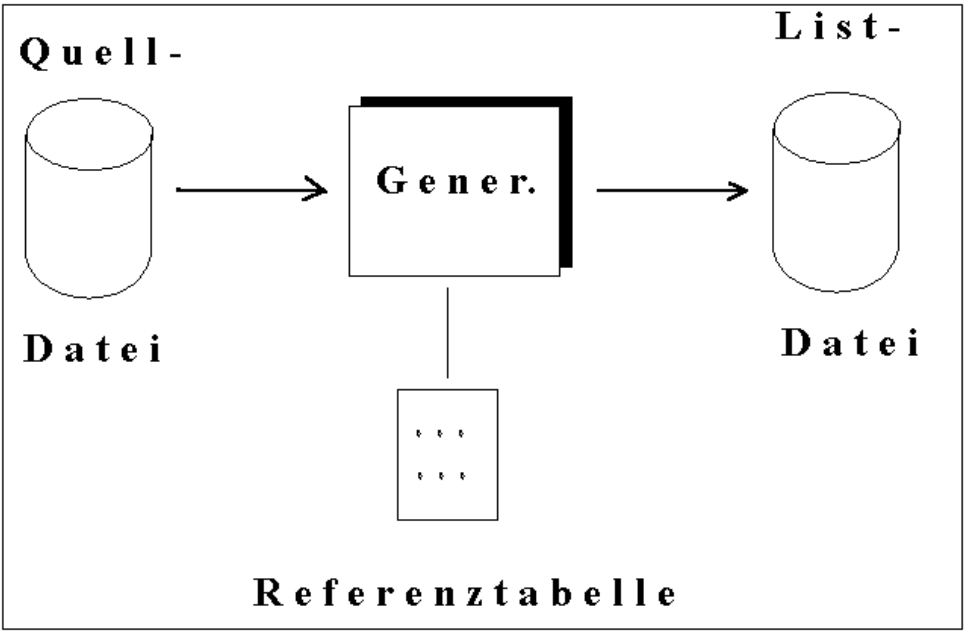


Bild 2.9: Funktionsprinzip eines Cross Referenz Generators

Doch alles der Reihe nach. Zuerst ist die Bedieneroberfläche zu spezifizieren. Das Programm läßt sich durch die Eingabe des Namens:

```
XREF
```

von der MS-DOS-Kommandoebene starten. Nun bestehen (ähnlich wie beim Programm LISTER) wieder zwei Möglichkeiten zum Einlesen des Dateinamens und eventueller Optionen.

Fehlt in der Kommandozeile der Name des einzulesenden Quellprogrammes, verzweigt XREF in den interaktiven Eingabemodus. Der Bildschirm wird gelöscht und es erscheint folgende Meldung:

```
C R O S S  R E F E R E N Z  G E N E R A T O R    (c) Born Version
1.0
Optionen [/L=00 linker Rand        /R=75 rechter Rand ]
         [/Z=60 Zeilen pro Seite                      ]

File    :
Optionen:
```

Bild 2.10: XREF-Eröffnungsmeldung im Dialogmodus

Mit *File* wird der Name der zu bearbeitenden Datei abgefragt. Hier ist jeder gültige MS-DOS-Dateiname einschließlich Laufwerks- und Pfadangaben erlaubt.

Die Abfrage der Optionen erscheint erst nach der Eingabe des Dateinamens. In der Kopfmeldung sind die möglichen Optionen aufgeführt. Ähnlich wie bei LISTER läßt sich die Randeinstellung mit /L und /R beeinflussen. Die Zahl der Zeilen pro Seite wird durch den Schalter /Z angegeben. Eine Leereingabe bei der Optionsabfrage veranlaßt, daß XREF die Standardeinstellung übernimmt.

•Linker Rand bei Spalte • 0 •
•Rechter Rand bei Spalte • 75 •
•Zeilen pro Seite• • 60

Tabelle 2.4: Standardeinstellung in XREF

Diese Voreinstellung wird in der Kopfmeldung bereits angegeben.

Bei fehlender Eingabedatei erscheint die Meldung:

```
Der Dateiname fehlt
```

Dann bricht das Programm ab. Ein anderer Fall tritt auf, wenn sich linke und rechte Randeinstellung überschneiden, oder wenn weniger als zehn Zeilen pro Seite zu drucken sind. Dann bricht XREF mit der folgenden Meldung ab:

```
Bitte Randeinstellung neu setzen
```

Sind alle Eingaben korrekt, eröffnet XREF die Ein- und Ausgabedateien. Als Name für die Ausgabedatei wird der Name der Eingabedatei, allerdings mit der Endung .REF, übernommen. Eine bestehende Datei mit diesem Namen wird ohne Warnung überschrieben. Dies scheint vertretbar, da ja eine Referenzliste beliebig oft generierbar ist. Nur andere Programme dürfen diese Extension nicht verwenden, da sonst das Ergebnis überschrieben wird. Die Datei bleibt auch nach der Bearbeitung erhalten, und läßt sich mit den MS-DOS-Kommandos COPY, TYPE und PRINT ausgeben.

Für die Einbindung in Batchdateien besteht die Möglichkeit, sowohl den Dateinamen als auch die Optionen direkt beim Aufruf mit anzugeben. Sobald ein Dateiname in der Kommandozeile auftritt, geht das Programm in den Kommandomodus über. Dann erscheint keine Kopfmeldung mehr und der Dateiname sowie eventuelle Optionen werden aus der Kommandozeile gelesen. Nachfolgend finden sich einige gültige Aufrufe zur Aktivierung des Kommandomodus.

```
XREF A:LISTER.BAS
XREF SPOOL.BAS /L=5 /Z=50
XREF XREF.BAS /L=5 /R=70 /Z=55
```

Der Dateiname muß als erstes hinter dem Kommando XREF auftauchen. Die Optionen lassen sich wahlweise angeben, wobei die Reihenfolge beliebig ist. Lediglich zur Trennung der einzelnen Parameter muß jeweils ein Leerzeichen verwendet werden.

Der Benutzer wird anschließend über den Programmmablauf mit den Meldungen:

```
Die Datei: <name> wird bearbeitet
Referenzliste erzeugen
Ende  Cross Referenz Generator
```

informiert. Anschließend findet sich auf dem Speichermedium (Platte, Floppy) die Datei mit der Endung .REF. Diese enthält den formatierten Quelltext mit einer laufenden Zeilennumerierung und der erzeugten Referenzliste. Kommentar- und Leerzeilen erhalten keine Zeilennummer. Die Datei wird mit der Information über die Zahl der gelesenen Quellcodezeilen und der ermittelten Variablenzahl abgeschlossen. Bild 2.11 zeigt einen Ausschnitt aus einer solchen Datei.

```
X R E F   /Z=55                          (c) Born Version 1.0
Datei : xref.bas      Datum : 04-25-1992         Seite :  1

 Zeile     Anweisung

      '*****************************************************
      '! File       : XREF.BAS
      '! Vers.      : 1.0
       .
```

```
      .
    1 %true = &HFFFF: %false = 0
    2 tmpx% = 0                              '! Hilfsvariable
    3 zeile% = 0                             '! Zeilennummer Listing
    4 seite% = 1                             '! Seitennummer Listing
    5 maxzeile% = 60                         '! Zeilen pro Seite
      .
      .

     '***** Programm Ende ******

X R E F - T a b e l l e                          (c) Born Version
1.0
Datei : xref.bas      Datum : 09-25-1988         Seite :  16

%false
 1  10  186  202  215  230  417

%maxentry
 14  15  16  93  229

%nil
 261  287  314  347
      .
      .

zeile%
 3  135  141  142  168  177  248  263  273  275  379


XREF Modul Information

Lines read    :  479
Symbols found :  74

End XREF
```

Bild 2.11: Auszug aus einer Querverweisliste

Der Beginn der Referenztabelle wird auf der ersten Seite mit einem Kopftext signalisiert. Die ersten Einträge sind für die Konstanten (%..) reserviert. Daran schließen sich die Variablen an. Die Informationen über die Zahl der gelesenen Anweisungen (ohne Kommentare) und die Variablenzahl sind sicherlich in vielen Fällen interessant. Für mich persönlich war es schon erstaunlich, als nach dem ersten Durchlauf von XREF.BAS zwar ca. 500 Zeilen angegeben wurden, das Programm aber weniger als 80 Variable enthielt. Die Implementierung sieht insgesamt 1.000 Einträge in der Variablenliste vor, so daß sich auch längere Programme ohne Probleme bearbeiten lassen.

Nach der Spezifikation der Bedieneroberfläche und der Anforderungen wenden wir uns der technischen Realisierung zu. Es stellt sich die Frage,

wie läßt sich ein solcher Generator realisieren? Wie können Variablen- und Konstantennamen erkannt werden?

Der Entwurf

Damit sind wir bereit beim Entwurf des Programmes angelangt. Der Generator soll alle Variablen-, Konstanten- und Unterprogrammnamen erkennen und in der Liste eintragen.

Um einen solchen Namen zu erkennen, ist jeweils eine komplette Zeile einzulesen und zeichenweise zu analysieren. Der Aufbau einer Konstanten oder einer Variablen läßt sich mit einem Syntaxdiagramm (Bild 2.12) darstellen.

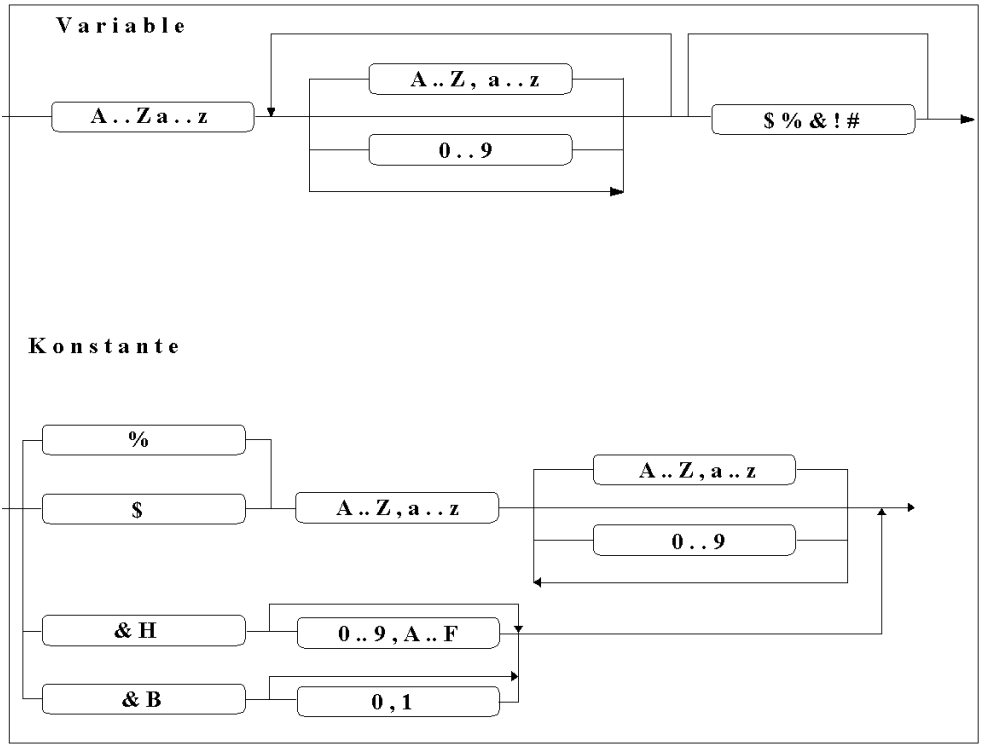


Bild 2.12: Syntaxdiagramm von Basic-Variablen und -Konstanten

Obwohl das Syntaxdiagramm auch numerische Konstanten (&HFFF) oder Steueranweisungen für den Compiler ($STACK) zuläßt, sollen diese beiden Typen nicht weiter verarbeitet werden. In der Liste erscheinen nur Konstanten der Form *%true*, da nur sie eine symbolische Darstellung besitzen. Das Ende einer Konstanten oder Variablen kann durch verschiedene Zeichen signalisiert werden (Leerzeichen, +, -, /, (etc.). Da recht viele Zeichen auftreten können, beschränkt sich die Analyse auf die Erkennung aller gültigen Zeichen einer Variablen (A..Z, a..z, %,&,$,!,#).

Dabei können die Zeichen für die Typdefinition (##, %% etc.) ebenfalls vorkommen. Tritt nun ein anderes Zeichen auf, ist das Ende des jeweiligen Basic-Bezeichners gefunden. Das erste gültige Zeichen signalisiert also den Anfang eines Namens, während das Ende durch das erste auftretende ungültige Zeichen markiert wird. Mit dieser Methode lassen sich recht einfach die einzelnen Bezeichner (Variablen, Konstanten, Labels, Schlüsselwörter etc.) einer Anweisungszeile ermitteln. Das Modul, welches diese Aufgabe übernimmt, wird nachfolgend als *Scanner* bezeichnet. Für die separierten Bezeichner wird der Begriff »token«verwendet. Aus einer eingelesenen Anweisungszeile sind also alle Tokes zu separieren.

Aber wie im täglichen Leben treten noch einige Fußangeln auf. Einmal lassen sich mit obigem Syntaxdiagramm auch PowerBASIC-Schlüsselwörter (IF, END IF, FOR etc.) erzeugen. Es muß nun aber sichergestellt werden, daß diese Schlüsselwörter nicht in der Referenzliste auftauchen. Kein Mensch interessiert sich wohl dafür, wie oft und in welcher Zeile ein IF auftritt. Deshalb wird eine eigene Tabelle mit den Basic-Schlüsselwörtern benötigt. Dann läßt sich jedes *token* überprüfen, und nur wenn es nicht in dieser Tabelle vorkommt, ist eine Einordnung in die Referenztabelle erlaubt.

Der zweite Haken liegt in der Schreibweise der Namen begründet. Die deutsche Sprache kennt Groß- und Kleinschreibung, während PowerBASIC dies zwar erlaubt, aber keine Unterscheidung zwischen Groß- und Kleinbuchstaben vornimmt. Die Namen:

```
TestText% = testtext%
```

sind also für PowerBASIC gleich. Ein Vergleich der Zeichenketten auf Gleichheit wird jedoch das Ergebnis *false* erhalten. Dies liegt daran, daß ein Stringvergleich über die ASCII-Codes erfolgt. Die Codes für Groß- und Kleinbuchstaben sind aber leider unterschiedlich. Jedes gefundene *token* ist also vor der Suche in Großbuchstaben zu konvertieren und mit dem Tabelleneintrag (auch in Großbuchstaben) zu vergleichen. Die Abspeicherung in der Referenztabelle hat aber wieder in der Originalschreibweise zu erfolgen, da der Benutzer dies bei der Ausgabe erwartet. Im Abschnitt über die Implementierung wird ein entsprechendes Modul vorgestellt, welches die Aufgabe löst.

Ein weiteres Problem betrifft die Kommentarzeilen. Innerhalb eines Kommentars können natürlich auch gültige Variablennamen auftreten. Diese dürfen natürlich nicht in die Referenzliste aufgenommen werden, da sie hier ohne Bedeutung sind. Kommentare können sowohl zu Beginn einer Zeile (' oder REM), oder mitten in der Zeile stehen. In beiden Fällen ist die Restzeile zu ignorieren.

Als letzte Schikane sollen nun noch die Textstrings in Basic-Anweisungen erwähnt werden. Falls in einer Anweisung der Text:

```
zeile$ = "Dies ist keine text$ Variable"
```

auftritt, darf nur *zeile$* als Variable erkannt werden. Der String zwischen den Anführungszeichen "..." ist zu ignorieren. Auch dies ist bei der Implementierung zu berücksichtigen.

Nun soll aber das Thema »Separierung der Variablen« beendet werden. Es stellt sich die Frage nach dem Aufbau der Referenztabelle. Die Schlüsselwörter werden in einer einfachen Tabelle abgelegt.

```
ABS
ABSOLUTE
AND
  .
  .
XOR
```

Tabelle 2.5: Basic-Schlüsselwörter

Weiterhin kann diese Tabelle noch einen Code enthalten, der signalisiert, ob eine Basic-Funktion (ABS, UCASE$) vorliegt. Die implementierte Tabelle enthält solche Codes, obwohl sie in der aktuellen Version von XREF nicht weiter ausgewertet werden. Denkbar ist es aber, über eine zuschaltbare Option diese Funktionsnamen in die Referenztabelle aufzunehmen.

Bleibt noch die Aufgabe, eine geeignete Speicherform für die eigentliche Referenztabelle zu finden. Diese besitzt im Prinzip folgende Struktur.

Name	Zeilennummer
aus%	10 15 20 ...
ein%	30 35
.	.
.	.

Tabelle 2.6: Struktur der Referenztabelle

Naheliegend ist deshalb der Gedanke, folgende Struktur im Speicher zu definieren:

```
DIM name$(300)              '! Name der Variablen
DIM lines%(300,30)          '! Zeilennummern
```

In diese Felder lassen sich die Daten dann abspeichern. Über kurz oder lang tauchen aber Probleme auf. Was passiert z.B., wenn im Programm mehr als die oben definierten 300 Variablennamen vorkommen. Eine Vergrößerung der Tabellengröße verschiebt das Problem lediglich, löst es aber nicht, da irgend ein Programm sicher diese Grenze erreicht.

Zusätzlich ist die nächste Schwierigkeit schon absehbar. Nehmen wir an, im Analyseteil wurde eine Variable erkannt und soll nun in die Tabelle eingetragen werden. Stehen die Namen in unsortierter Reihenfolge im Feld *name$*, sind alle belegten Positionen auf Übereinstimmung zu testen. Ein neuer Name wird dann am Tabellenende angefügt. Bei langen Listen bedeutet dies einen hohen Suchaufwand. Abhilfe bringt eine sortierte

Tabelle. Hier lassen sich optimalere Suchverfahren (binäres Suchen etc.) anwenden. Was passiert aber, falls der Name neu ist? Dann sind alle Einträge ab der aktuellen Position um eine Stelle zu verschieben, was ebenfalls Aufwand bedeutet. Verfahren die diese Nachteile umgehen (Hash-Tabellen etc.) lassen sich in Basic nur schwer realisieren und sollen aus diesem Grunde hier nicht diskutiert werden.

Getreu nach dem Spruch: »Aller guten Dinge sind drei« sei noch auf eine weitere Unzulänglichkeit hingewiesen. Pro Variablenname existiert in der Tabelle das Feld *Lines%(i,30)*, d.h. es lassen sich 30 Zeilennummern eintragen. Es gibt aber mit Sicherheit Programme, in denen eine Variable in mehr als diesen 30 Zeilen vorkommt. Das Feld zu vergrößern bringt auch Probleme. Es wird viel Speicherplatz ver(sch)wendet. Denn bei allen Variablen, die nur in wenigen Zeilen auftauchen, schleppen wir unbenutzte Speicherzellen mit, während bei häufig auftretenden Variablen der Platz nicht reicht.

Dies zeigt wieder einmal, daß der Teufel oft im Detail steckt. Der Gedanke, die Tabellen einfach auf Platte oder Diskette zu verlagern, bringt zwar Abhilfe beim Speicherplatzproblem, löst aber nicht die restlichen Schwierigkeiten. Außerdem verschlechtert sich die Zugriffszeit erheblich. (Mit diesen Schwierigkeiten hatte ich auch nicht gerechnet, als vor vielen Jahren die Idee zu XREF entstand.)

Es bleibt also nichts anderes übrig, als aus den verschiedenen Ansätzen eine geeignete Kompromißlösung auszusuchen. Das Problem »Speicherplatz versus Laufzeit« ist weiterhin aktuell. Wie die Lösung aussieht, wird nachfolgend diskutiert.

Der Ansatz in XREF

Um zu halbwegs passablen Verarbeitungsgeschwindigkeiten zu gelangen, scheidet eine Auslagerung auf externe Speichermedien (Floppy, Platte) aus. Dies fällt leicht, da PowerBASIC keine indexsequentielle Dateiverwaltung besitzt. Eine Programmierung einer solchen Dateiverwaltung ist zwar möglich, aber in PowerBASIC zu aufwendig.

Damit deutet sich schon an, daß die Daten im Hauptspeicher zu halten sind. Es stellt sich sofort die Frage nach dem Netto-Speicherplatzbedarf dieser Lösung. PowerBASIC erlaubt eine Länge von 255 Zeichen pro Variablenname. Bei einer Tabellengröße von 1000 Einträgen ergibt sich als Ergebnis:

*1000 Einträge * 255 Bytes = 255 Kbyte*

Dies ist natürlich nicht tragbar, da ja weitere Informationen zu speichern sind. Gottlob ist der Mensch aber schreibfaul, was insbesondere auf Programmierer zutreffen soll. Man kann also davon ausgehen, daß im Mittel pro Variablennamen nicht mehr als etwa zehn Zeichen verwendet werden. Damit sieht die Rechnung bereits anders aus:

*1000 Einträge * 10 Bytes = 10 Kbyte*

Die ist ein akzeptabler Wert. Glücklicherweise besitzt PowerBASIC noch eine effiziente Stringverwaltung, die nur dann Speicher reserviert, wenn auch wirklich Zeichen vorliegen. Auf die Interna möchte ich an dieser Stelle nicht eingehen. In der Praxis wird also relativ wenig Speicher für diese Tabelle belegt. Damit ist klar, daß ein Feld mit 1.000 Elementen vom Typ *String* zur Aufnahme der Variablennamen definiert wird.

Bleibt noch die Speicherung der Zeilennummern. Die Idee mit einem zweidimensionalen Integerfeld:

```
DIM zeilennr%(1000,30)
```

belegt einen Speicherplatz von:

*2 Byte * 30 * 1000 Elemente = 60 Kbyte*

Dies ist recht viel, insbesondere wenn man bedenkt, daß einerseits ein Großteil dieser Einträge unbelegt bleibt, weil eine Variable nur in wenigen Zeilen auftritt. Anderseits reicht der Platz für oft benutzten Variable nicht, da diese mit Sicherheit in mehr als 30 Zeilen auftreten. Bevor mit der Tabellengröße jongliert wird bedienen wir uns doch der Basic-Stringverwaltung. Die Zeilennummern lassen sich direkt als String ablegen, der nur dann Speicher benötigt, wenn auch wirklich Zeichen eingetragen werden.

Damit ergibt sich für die interne Tabelle folgende Struktur:

```
DIM tablename$(1000)              '! Name der Variablen
DIM tableline$(1000)              '! Zeilennummern
```

was eine recht gute Ausnutzung des Hauptspeichers erlaubt.

Nun kann endlich über die Verwaltung dieser Tabelle nachgedacht werden. Wie lassen sich die Variablen eintragen und wie wird geprüft, ob ein Name bereits vorhanden ist?

Die einfachste Lösung besteht darin, die Variablen fortlaufend in die Tabelle einzusetzen. Neue Variablen werden einfach an das freie Ende der Tabelle angehängt. Leider birgt dieser Ansatz einige Nachteile:

- Als Randbedingung soll die Referenztabelle natürlich alphabetisch sortiert ausgegeben werden. Die Liste ist also vor der Ausgabe noch zu sortieren.
- Bei jeder Variable ist die komplette Tabelle zu prüfen, bis feststeht ob der Name bereits eingetragen ist.

Dies verursacht natürlich einen erheblichen Aufwand. Insbesondere die Suchzeit wächst linear mit der Zahl der Einträge in der Tabelle. Falls diese Einträge sortiert vorliegen, kann ein binäres Suchverfahren die Zahl der Vergleiche erheblich reduzieren. Bei dieser Methode wird die Liste durch zwei Zeiger (oben und unten) begrenzt. Im ersten Schritt werden die Zeiger auf das erste und letzte Element gesetzt. Dann wird die Liste halbiert und der Eintrag in der Tabellenmitte mit dem Suchbegriff verglichen. Ist der

Suchbegriff größer als dieser Eintrag, kann die Suche auf die obere Tabellenhälfte reduziert werden. Im nächsten Schritt ist das Intervall also auf die obere Tabellenhälfte reduziert und die Halbierung beginnt wieder. So wird das Intervall sukzessive verkleinert, bis nur noch ein Element übrigbleibt. Stimmt dieses nicht mit dem Suchbegriff überein, liegt kein Eintrag in der Tabelle vor. Die maximale Zugriffszahl reduziert sich auf:

log2(n)

Dabei ist *n* die Zahl der Zugriffe und *2*n* gibt die Tabellengröße an. Bei 256 Elementen ist die Liste nach maximal acht Zugriffen durchsucht. Die lineare benötigt dagegen 256 Zugriffe. Das binäre Suchverfahren wird in XREF verwendet, um einen Namen in der Tabelle mit den Schlüsselwörtern zu finden, da diese für solche Zwecke vorbereitet wurde.

Damit deutet sich schon an: das Verfahren funktioniert nur bei sortierten Tabellen. Dies ist aber bei Variablenlisten im allgemeinen nicht der Fall. Kein Programmierer vergibt die Namen in alphabetisch sortierter Form. Also ist die Liste sortiert aufzubauen. Dies bedeutet, daß bei jedem neuen Namen eine Umsortierung der Tabelle erforderlich wird. In der Regel läuft dies auf ein Verschieben aller nachrangigen Einträge um einen Tabellenplatz hinaus. Was bei der Suchzeit eingespart wird, geht nun beim Sortieren wieder verloren, denn insbesondere das Verschieben von Strings erfordert Aufwand.

Also muß eine andere Strategie her. Um weitere Umwege zu vermeiden, gehe ich direkt auf die verwendete Methode ein. Es kommen verkettete Listen mit einem Zugriff über einen einstufigen Index zum Einsatz. Was steckt hinter diesem Begriff?

Nun, einmal tritt das Problem der Sortierung auf. Die Tabelle ist alphabetisch geordnet auszugeben. Andererseits ist der Aufwand zur Sortierung möglichst gering zu halten. Zur Lösung eignen sich verkettete Listen (Bild 2.13).

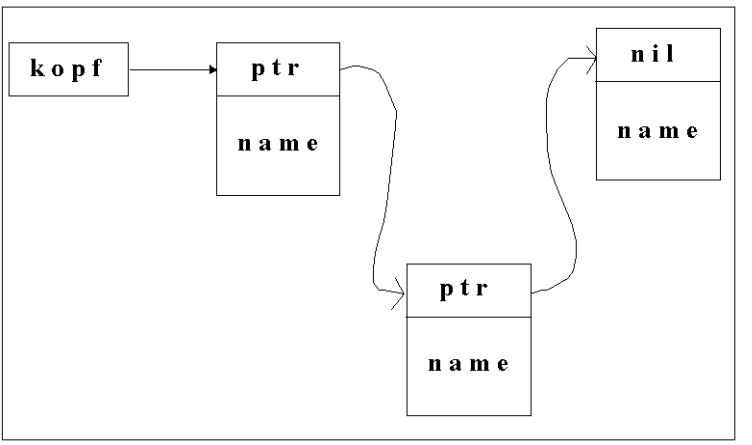


Bild 2.13: Aufbau verketteter Listen

Es werden jeweils zwei Tabellen angelegt. Eine Tabelle dient zur Aufnahme der eigentlichen Namen. Die andere Tabelle dient zur Aufnahme von Zeigern. Da beide Tabellen gleiche Länge besitzen, kann jeweils ein Name und ein Zeiger als Einheit angesehen werden. Die Werte in der Zeigertabelle sind nun so einzustellen, daß sie die Position der logisch folgenden Sätze angeben. Damit lassen sich diese physikalisch an jeder beliebigen Position in der Tabelle abspeichern. Die Zeiger stellen eine logisch verknüpfte Kette her, über die sich dann die Namen in sortierter Reihenfolge ermitteln lassen. Eine separate Variable (*kopf*) enthält einen Zeiger auf das erste Element dieser Liste. Das Ende der Liste wird durch den Zeiger mit dem Wert *nil* markiert. Der Wert *nil* hängt von der Implementierung ab (negative Zahlen oder Null). Eine recht elegante Methode, die eine Sortierung ohne großen Aufwand erlaubt.

Aber die Sache hat wieder einen Nachteil. Da die Liste physikalisch unsortiert vorliegt, klappt die Binärsuche nicht mehr. Also ist die Liste linear zu durchsuchen. Bei 500 Einträgen ein erheblicher Aufwand. Offenbar scheinen sich die Forderungen nach kurzer Suchzeit und geringem Sortieraufwand zu widersprechen.

Diese Fragestellung hat natürlich Generationen von Informatikern beschäftigt, so daß mittlerweile mehrere Lösungen bekannt sind.

Um die Suche in der Tabelle zu verkürzen, wird ein Trick verwendet: Wenn eine Liste zu lang ist, wer hindert uns denn an der Aufteilung in mehrere Teillisten? Diese sind dann wesentlich kürzer und lassen sich demnach schneller durchsuchen. Wie teilt man die Liste auf und in welcher Teilliste ist zu suchen?

Hier gibt es ein einfaches Verfahren. Jeder Variablenname beginnt mit einem Buchstaben des Alphabets. Die Teillisten werden nun so organisiert, daß sie den einzelnen Buchstaben zugeordnet sind. Dann reicht das erste Zeichen eines Namens zur Identifikation der Teiltabelle. Anschließend kann eine weitere Prüfung der Einträge auf Übereinstimmung erfolgen. Wird der Name nicht gefunden, steht auf jeden Fall fest, daß er auch nicht in den anderen Teiltabellen auftritt. Das Verfahren wird als indexsequentielle Suche bezeichnet. Der Aufbau einer solchen Tabelle ist Bild 2.14 zu entnehmen.

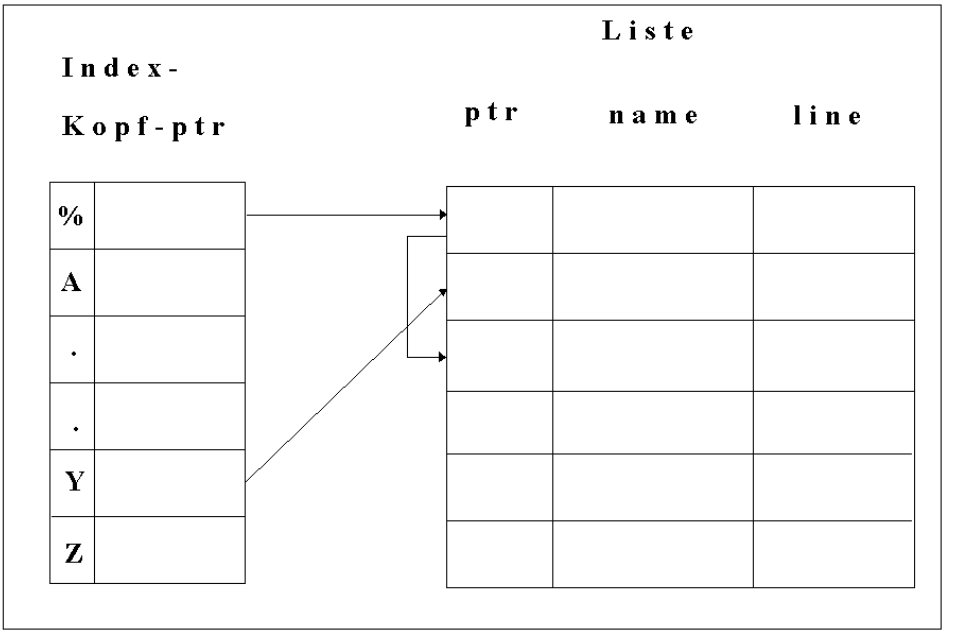


Bild 2.14: Liste mit einer Indextabelle

In unserem Fall enthält die Indextabelle 27 Einträge, da auch Konstanten (%...) zu speichern sind. Die Kopfzeiger der Indextabelle verweisen auf den ersten Eintrag in der Teilliste. Existiert keine Teilliste, wird der Zeiger mit dem Wert *nil* belegt. Bei sehr langen Teiltabellen kann der Index auch mehrstufig angelegt werden. Damit lassen sich dann sehr effiziente Zugriffe erzielen. Die Implementierung in XREF benutzt aber nur einen einstufigen Index.

Damit ist die Aufgabe fast schon gelöst (hoffentlich?). Eine Kleinigkeit soll aber noch angesprochen werden. Es kann vorkommen, daß eine Variable mehrfach in einer Zeile auftritt. Es stört aber, wenn in der Referenzliste die gleiche Zeilennummer bei einer Variablen mehrfach auftritt. Vor dem Eintrag der Zeilennummer ist also zu prüfen, ob diese nicht bereits vorhanden ist.

Weiterhin ist das Problem Groß-/Kleinschreibung zu lösen. Vor jedem Vergleich werden beide Strings in Großbuchstaben konvertiert. Damit wird nun endgültig mit der Implementierung begonnen.

Die Implementierung

Nach diesen Vorüberlegungen besteht noch die Aufgabe, die Konzepte in geeignete Algorithmen zu überführen. Um das Programm transparent und änderungsfreundlich zu halten (welch dehnbarer Begriff), empfiehlt sich

eine Aufgliederung in Teilmodule. Bild 2.15 zeigt eine Übersicht über diese Module und deren Zusammenschaltung.

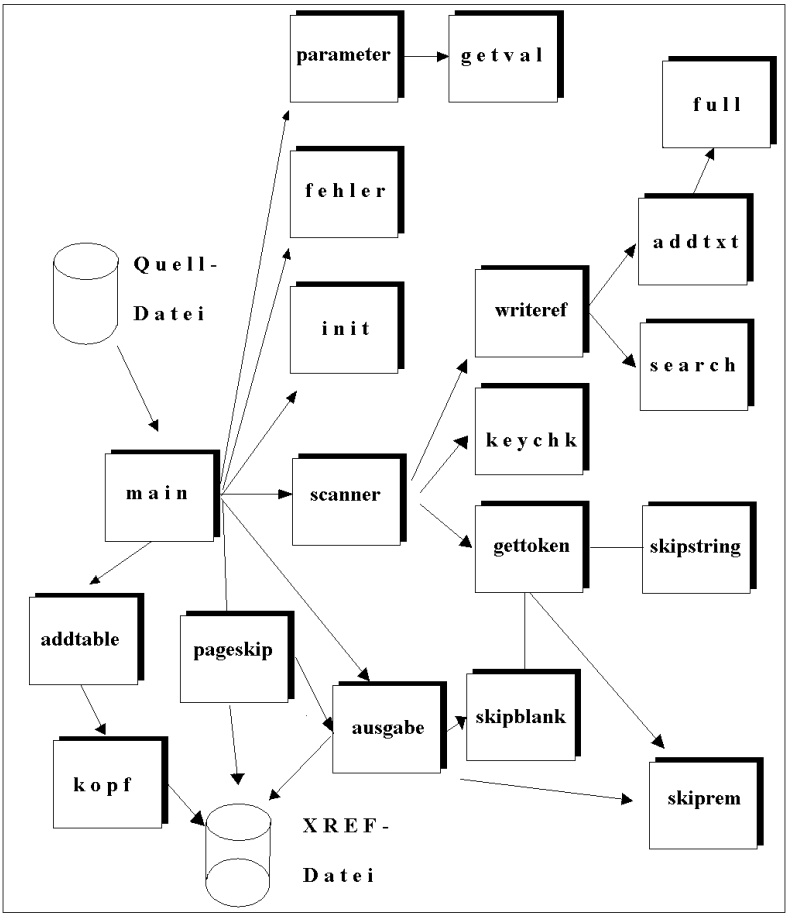


Bild 2.15: Modulhierarchie in XREF.BAS

Das Bild deutet bereits den komplexen Aufbau des Programmes an.

Vor der Diskussion der einzelnen Module sollen die wichtigsten Variablen noch kurz beschrieben werden:

```
%maxentry                Größe der Tabelle mit Schlüsselwörtern
keyword$()               Tabelle mit den Schlüsselwörtern
keycode%()               Tabelle mit den Codes (unbenutzt)

index1%()                Indextabelle mit Kopfzeigern
%tablen                  Größe Referenztabelle (1000)
tableptr%()              Tabelle mit Listenzeigern
tablename$()             Tabelle mit Variablennamen
tableline$()             Tabelle mit Zeilennummern
```

Die Größe der Referenztabelle läßt sich leicht durch die Konstante *%tablen* variieren.

Nun aber zur Beschreibung der einzelnen Module:

Main

Im Hauptmodul werden die wichtigsten Variable reserviert. Dann entscheidet der Eingabemodus (Kommando- oder Interaktivmodus), ob die Kopfmeldung mit der Abfrage des Dateinamens erscheint, oder ob diese Informationen aus der Kommandozeile zu separieren sind. Die Unterprogramme »parameter« und »getval« dienen zur Analyse der Eingaben und wurden bereits im Abschnitt zum Programm LISTER besprochen. Nach dem Öffnen der Dateien beginnt der Aufbau der internen Tabellen (init). In der Schleife:

```
WHILE (NOT (EOF(ein%)
   .
WEND
```

wird die Quelldatei zeilenweise eingelesen. Kommentare sind durch »skiprem« zu erkennen. Das Unterprogramm »ausgabe« sorgt für eine Aufbereitung des Quelltextes (Zeilennummern, Seitenkopf etc.) und dessen Speicherung. Das Modul »scanner« enthält nur Code zur Analyse der eingelesenen Texte. Nach Beendigung der Schleife wird die Referenztabelle an den Quelltext angehängt. XREF schließt die Dateien und terminiert. Die Modularisierung erlaubt einen kompakten Aufbau des Hauptprogramms.

!Syntaxfehler, NURDie Hilfsmodule

Nun werden die neu hinzugekommenen Hilfsmodule vorgestellt.

init

Nach dem Programmstart ist die Tabelle mit den Schlüsselwörtern aufzubauen. Dies erfolgt in *init*, welches die DATA-Anweisungen in das Feld *keyword$()* einliest.

ausgabe

Dieses Modul dient zur formatierten Ausgabe des Quelltextes in die Ergebnisdatei. Seitenwechsel werden durch Aufruf des Programmes »pageskip« gesteuert. Kommentarzeilen erhalten keine Zeilennummer. Der Aufbau lehnt sich an LISTER an.

scanner

Hier ist die eingelesene Zeile (*text$*) auf Variablen- und Labelnamen zu untersuchen. Die eigentliche Analyse erfolgt in einer DO LOOP-Schleife, in der das Unterprogramm »gettoken« aufgerufen wird. Bei erfolgreicher Suche wird die Variable:

```
token% = %true
```

gesetzt. Der gefundene Name ist im Parameter »tokentxt$« gespeichert. Mit dem Aufruf »keychk« prüft das Modul »scanner« ob ein Schlüsselwort als token zurückgegeben wurde. Nur wenn dies nicht zutrifft, sorgt das Modul »writeref« für einen Eintrag in der Referenzliste.

gettoken

Dieses Modul sucht nach gültigen Variablen- und Labelnamen. Durch »skipblank« werden führende Leerzeichen entfernt. Mit »skiprem« und »skipstring« lassen sich Kommentare und Zeichenketten überspring. Wind win Name erkannt, ist dieser im Parameter *token$* zurückzugeben. Die Analyse geschieht in der Schleife:

```
WHILE search% AND ptr% <= lang%
  .
  .
WEND
```

Der Aufbau eines Variablennamens wurde bereits beim Entwurf ausgiebig diskutiert. Zuerst ist der Anfang des *tokens* zu suchen (%,$,&,A..Z,a..z). Wird dieser Anfang gefunden, beginnt die Analyse. Das Ende eines *tokens* markieren Zeichen ungleich (a..z A..Z $ % & ! #). Damit lassen sich auch Schlüsselwörter (IF, THEN, ELSE) und numerische Konstanten (&HFF) auswerten. Am Zeilenende oder bei Beginn eines Kommentars wird die Suche beendet. Das Flag *found%* signalisiert das Ergebnis (true, false).

keychk

Im Parameter *symbol$* wird der Suchbegriff als Text übergeben. Das Modul prüft, ob dieser Begriff in der Tabelle mit den Schlüsselwörtern vorkommt. In diesem Fall handelt es sich um ein Basic-Schlüsselwort. Das Flag *found%* wird auf *true* gesetzt. Zusätzlich geben die Parameter *code%* und *index%* einen Code und die Position in der Schlüsseltabelle an. Beide Parameter werden in XREF aber nicht weiter ausgewertet. Vor der Suche wird der Suchtext in Großbuchstaben konvertiert. Als Strategie wird die Binärsuche verwendet. Damit läßt sich die Zahl der Zugriffe gering halten.

writeref

Die Abspeicherung des Namens (*token$*) und der Zeilennummer (*zeile%*) in die Referenztabelle erfolgt in diesem Modul. Zuerst ist der Name in Großbuchstaben zu wandeln. Dann wird an Hand des erstens Buchstabens die Position in der Indextabelle (*index1%()*) bestimmt. Auf die Abspeicherung der Zeichen (%, A .. Z) in dieser Indextabelle wurde verzichtet, da sich über die ASCII-Codes eine eindeutige Zuordnung ergibt.

Der Zeiger *index1%(ptr%)* verweist nun auf den jeweiligen Listenkopf. Diese Teilliste ist sequentiell auf gültige Einträge zu durchsuchen und gegebenenfalls ist der neue Name in die Teilliste einzufügen. Hierfür existieren die Hilfsprogramme »search« und »addtxt«. Das Listenende wird durch einen Zeiger mit dem Wert %nil markiert. Liegt noch keine Tabelle vor, ist der erste Eintrag direkt mit addtxt auszuführen. Hierfür ist der Steuercode auf 1 zu setzen. Andernfalls beginnt erst die Suche mit search.

Bei gefunden»n Einträgen wird die Zeilennummer an der angegebenen Position an den Teilstring angehängt (*tableline$()*).

Der von *search* zurückgegebene Code (*code%*) steuert die Art der Einfügung in *addtxt*. Die beiden Module *search* und *addtxt* sind so aufgebaut, daß die Steuercodes kompatibel sind. Ist *code% = 0*, dann liegt der Eintrag bereits vor. Der Parameter *ptr%* zeigt auf den entsprechenden Satz.

search

Dieses Modul erhält als Parameter einmal den Zeiger auf den Listenkopf (*ptr%*) sowie den Suchtext in der Variablen *token$*. Dann wird die verkettete Liste sequentiell durchsucht. Hierzu dienen die Zeiger. Vergleiche zwischen Suchtext und Tabellentext dienen zur Ermittlung der Position des neuen Satzes. Das Modul gibt in der Variablen *result%* das Ergebnis der Suchoperation zurück:

Result%	Bemerkung
0	Satz gefunden, ptr% -> aktueller Satz
	Satz nicht gefunden -> einfügen:
1	am Tabellenanfang, ptr% -> Tabellenanfang
2	an Tabellenende, ptr% -> letzter logischer Satz
3	n Tabelle einfügen, ptr% -> logischer Vorgänger

Tabelle 2.7: Ergebniscodes von Search

Diese Codes wurden so gewählt, daß *addtxt* damit den Eintrag steuern kann.

addtxt

Dieses Modul übernimmt das Einfügen eines neuen Variablennamens in die Liste. Der Parameter zeigt auf den Satz, hinter dem der neue Name logisch einzuordnen ist. Physikalisch ist der Satz hinter der Variablen *top%* abzuspeichern, da dieser Zeiger das Tabellenende markiert. Der Parameter *varname$* enthält den ASCII-String des Variablennamens. Mit dem Parameter *code%* wird der Einfügemodus gesteuert. Die Belegung dieser Steuercodes wurde bereits beim Modul *search* besprochen.

addtable

Nach Abschluß der Analyse muß die Referenztabelle an die Ausgabedatei angefügt werden. Dies erfolgt im Unterprogramm *addtable*. Beginnend mit der Indextabelle *index1%()* werden die Teillisten in logischer Folge sequentiell durchlaufen. Die Namen und Zeilennummern sind dann in die Ausgabedatei zu kopieren. Dabei ist natürlich auf eine geeignete Formatierung zu achten. Zum Abschluß werden die Information über Zeilen- und Variablenzahl ausgegeben. Damit ist die Aufbereitung der Referenzliste abgeschlossen.

Weitere Einzelheiten sind dem nachfolgenden Programmlisting zu entnehmen, welches mit XREF erstellt wurde.

Erweiterungsvorschläge

Das Programm gibt zur Zeit noch keine Basic-Funktionen in der Referenzliste aus. Weiterhin fehlt auch die Typangabe bei den Variablen (Integer, Real etc.). Diese beiden Funktionen könnten nachgerüstet werden. Weiterhin ist vorstellbar, daß über eine Option die Ausgabe auf dem Drucker automatisch erfolgt. Auch die Verarbeitung von Include-Dateien ist sicherlich eine lohnende Aufgabe.

```
X R E F   /Z=50                              (c) Born Version 1.0
Datei : xref.bas      Datum : 05-13-1992         Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : XREF.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 22. 4.92
      '! Autor      : G. Born
      '! Files      : INPUT, OUTPUT
      '! Progr. Spr.: PowerBASIC 2.x
      '! Betr. Sys. : DOS 2.1 - 5.0 (+ DR-DOS 5.0/6.0)
      '! Funktion: Das Programm liest den PowerBASIC Quellcode
      '!           ein und gibt diese mit Zeilennummern versehen in
      '!           einer zweiten Datei wieder aus. Gleichzeitig
      '!           wird eine Crossreferenzliste der auftretenden
      '!           Variablen erzeugt. Diese Liste wird dann an die
      '!           Quelldatei angehängt. Die Datei besitzt den
      '!           gleichen Namen wie die Quelldatei, aber mit
      '!           der Extension .REF. Die Datei kann per Editor
      '!           angesehen werden und läßt sich mit dem DOS
      '!           Programm PRINT auf dem Drucker ausgeben.
      '!
      '! Aufruf:   XREF Filename /Optionen
      '!           Optionen:  /Lxx linker Rand           [ 0 ]
      '!                      /Rxx rechter Rand          [75 ]
      '!                      /Zxx Zeilen pro Seite      [60 ]
      '!
      '!           Die Werte in [] geben die Standardeinstellung
      '!           wieder. Wird das Programm ohne Parameter aufge-
      '!           rufen, sind Dateiname und Optionen explizit ab-
      '!           zufragen.
      '*********************************************************
      '! Variable und Konstanten definieren
    1 %true = &HFFFF: %false = 0: %nil = 0
    2 tmpx% = 0                              '! Hilfsvariable
    3 zeile% = 0                             '! Zeilennummer Listing
    4 seite% = 1                             '! Seitennummer Listing
    5 maxzeile% = 60                         '! Zeilen pro Seite
    6 rechts% = 75                           '! rechter Rand
    7 links% = 0                             '! linker Rand
    8 spalte% = 0                            '! Einrückung

    9 remflg% = %false                       '! Kommentar gefunden
```

```
   10 lang% = 0                                '! Länge Zeile

   11 ein% = 1                                 '! Dateinummer Eingabe
   12 aus% = 2                                 '! Dateinummer Ausgabe


      '### Tabellen für XREF, automatischer Init durch Basic
      '
   13 %maxentry = 295                          '! Feldgröße
   14 DIM keyword$(1:%maxentry)                '! Schlüsselwörter
   15 DIM keycode%(1:%maxentry)                '! Schlüsselcodes

   16 DIM index1%(0:26)                        '! Indextabelle
   17 %tablen = 1000                           '! Tabellengröße XREF
   18 DIM tableptr%(1:%tablen)                 '! ptr auf Folgesatz
   19 DIM tablename$(1:%tablen)                '! Variablennamen
   20 DIM tableline$(1:%tablen)                '! Zeilennummern

   21 top% = 0                                 '! oberster Eintrag

   22 ON ERROR GOTO fehler                     '! Fehlerausgang

      '############################################################
      '#                      Hauptprogramm                       #
      '############################################################

   23 kommando$ = COMMAND$                     '! Parameter?
   24 IF LEN (kommando$) = 0 THEN              '! User Mode?
   25  CLS                                     '! clear Screen

   26  PRINT "C r o s s  R e f e r e n z  G e n e r a t o r ";
   27  PRINT "  (c) Born Version 1.0"
   28  PRINT
   29  PRINT "Optionen [ /L=00 linker Rand         /R=75  rechter
Rand   ]
        "
   30  PRINT "          [ /Z=60 Zeilen pro Seite
]
        "
   31  PRINT
   32  INPUT  "File     : ",filename$
   33  INPUT  "Optionen : ",options$
   34  PRINT
   35 ELSE                                     '! Kommando Mode
   36  ptr% = INSTR (kommando$,"/?")           '! Option /?
   37  IF ptr% <> 0 THEN                       '! Hilfsbildschirm
   38   PRINT "X R E F                         (c) Born Version 1.0"
   39   PRINT
   40   PRINT "Aufruf: XREF <Filename> <Optionen>"
   41   PRINT
   42   PRINT "Optionen :"
   43   PRINT

   44   PRINT "  /L=00 setzt den linken Rand"
   45   PRINT "  /R=75 setzt den rechten Rand"
```

```
   46   PRINT "  /Z=60 setzt die Zeilenzahl pro Seite"
   47   PRINT
   48   PRINT "XREF erzeugt aus PowerBASIC Dateien ein Listing mit
einer"
   49   PRINT "Cross-Referenz-Tabelle aller Variablen des
Programmes. Die"
   50   PRINT "Ränder und die Zahl der Zeilen pro Seite lassen sich
im Lis
       ting"
   51   PRINT "einstellen. Die Ausgabe erfolgt in eine Datei mit
dem Namen
       "
   52   PRINT "xxxx.REF, die anschließend ausgedruckt werden kann."
   53   PRINT
   54   SYSTEM
   55  END IF

   56  ptr% = INSTR (kommando$,"/")         '! Optionen?
   57  IF ptr% = 0 THEN
   58   filename$ = kommando$               '! nur Filename
   59  ELSE
   60   filename$ = LEFT$(kommando$,ptr% -1) '! Filename separieren
   61   options$  = MID$(kommando$,ptr%)    '! Optionen separieren
   62  END IF
   63 END IF

   64 GOSUB parameter                       '! Optionen decodieren

   65 IF (rechts% < links%) or (maxzeile% < 10) THEN '! sinnlose
   66  PRINT                                        '! Einstellung
   67  PRINT "Bitte Randeinstellung neu setzen"     '! Fehlerexit
   68  END                                  '! Exit
   69 END IF

   70 IF filename$ = "" THEN               '! Leereingabe?
   71  PRINT
   72  PRINT "Der Dateiname fehlt"
   73  END                                  '! Exit
   74 END IF

   75 ptr% = INSTR(filename$,".")           '! hat Datei eine
Extension?
   76 IF ptr% > 0 THEN
   77  outfile$ = LEFT$(filename$,ptr%) + "REF" '! Filename ohne
Extension
   78 ELSE
   79  outfile$ = filename$ + ".REF"        '! Extension anhängen
   80 END IF


      ' prüfe ob Datei vorhanden, nein -> exit

   81 OPEN filename$ FOR INPUT AS #ein%     '! Öffne Eingabedatei
   82 OPEN outfile$ FOR OUTPUT AS #aus%     '! Öffne Ausgabedatei
   83 PRINT
```

```
   84 PRINT "Die Datei: ";filename$;" wird bearbeitet"

   85 GOSUB init                              '! Tabellen aufbauen
   86 GOSUB pageskip                          '! Seitenkopf ausgeben

   87 WHILE NOT (EOF(ein%))                   '! Datei sequentiell
lesen
   88  LINE INPUT #ein%, linie$               '! lese Zeile
   89  CALL skiprem(1,linie$,remflg%)         '! prüfe auf Kommentar
   90  lang% = LEN(linie$)                    '! ermittle Zeilenlänge
   91  CALL ausgabe (linie$)                  '! drucke Zeile
   92  IF (lang% > 0) AND (NOT remflg%) THEN '! nur Anweisungen
   93   CALL scanner(linie$)                  '! analysiere Satz
   94  END IF
   95 WEND

   96 PRINT "Referenzliste erzeugen"
   97 GOSUB addtable                          '! xref Liste erzeugen

   98 CLOSE                                   '! Dateien schließen
   99 PRINT
  100 PRINT "Ende Cross Referenz Generator"
  101 END

      '############################################################
      '#                     Hilfsroutinen                        #
      '############################################################

  102 fehler:
      '------------------------------------------------------------
      '! Fehlerbehandlung in XREF
      '------------------------------------------------------------

  103 IF ERR = 53 THEN
  104  PRINT "Die Datei ";filename$;" existiert nicht"
  105 ELSE
  106  PRINT "Fehler : ";ERR;" unbekannt"
  107  PRINT "Programmabbruch"
  108 END IF
  109 END                                  '! MSDOS Exit
  110 RETURN


  111 init:
      '!-----------------------------------------------------------
      '! Initialisierung der Tabelle mit den Schlüsselwörtern
      '!-----------------------------------------------------------

  112 FOR i% = 1 to %maxentry
  113  READ keyword$(i%), keycode%(i%)    '! Schlüsselwörter
einlesen
  114 NEXT i%

  115 RETURN
```

```
  116 parameter:
      '-------------------------------------------------------
      '! Decodiere die Eingabeoptionen
      '-------------------------------------------------------

  117 ptr% = INSTR (options$,"/Z=")
  118 IF ptr% > 0 THEN CALL getval (maxzeile%) '! Zeilen / Seite
  119 szeile% = maxzeile% + 1               '! Zeilennr Seite
wechseln

  120 ptr% = INSTR (options$,"/L=")
  121 IF ptr% > 0 THEN CALL getval (links%) '! linker Rand

  122 ptr% = INSTR (options$,"/R=")
  123 IF ptr% > 0 THEN CALL getval (rechts%) '! rechter Rand

  124 RETURN

  125 SUB getval (wert%)
      '-------------------------------------------------------
      '! Decodiere den Eingabestring in eine Zahl
      '-------------------------------------------------------
  126  SHARED options$, ptr%
  127  LOCAL i%

  128  ptr% = ptr% + 3                      '! ptr hinter /x=
  129  i% = 1
  130  WHILE ((ptr%+i%) =< LEN (options$)) AND _
  131      (MID$(options$,ptr%+i%,1) <> " ")
  132   i% = i% + 1                         '! Ziffernzahl + 1
  133  WEND
  134  wert% = VAL(MID$(options$,ptr%,i%))  '! decodiere die Zahl
  135 END SUB

  136 pageskip:
      '-------------------------------------------------------
      '! Seitenvorschub mit Kopf (Dateiname, Datum, Seite)
      '-------------------------------------------------------
  137  IF szeile% < maxzeile% THEN RETURN   '! kein Seitenwechsel
!!

  138  IF seite% > 1 THEN                   '! 1. Seite k. Vorschub
  139   PRINT #aus%, CHR$(12)               '! Vorschub
  140   szeile% = 5                         '! 5 Kopfzeilen
  141  ELSE
  142   PRINT #aus%, "X R E F   "; options$; SPACE$(27);
  143   PRINT #aus%, "(c) Born Version 1.0"
  144   szeile% = 6                         '! 6 Kopfzeilen
  145  END IF
  146  PRINT #aus%, "Datei : ";filename$;"     Datum : ";DATE$;
  147  PRINT #aus%, "         Seite : "; seite%
  148  PRINT #aus%,
  149  PRINT #aus%, SPACE$(links%); " Zeile     Anweisung"
  150  PRINT #aus%,
```

```
  151  INCR seite%

  152 RETURN

  153 SUB ausgabe (text$)
      '-----------------------------------------------------------
      '! Ausgabe der eingelesenen Zeile in die Ausgabedatei.
      '! rest% gibt an, wieviele Zeichen pro Zeile gedruckt
      '! werden dürfen. Ist die eingelesene Zeile länger, wird
      '! sie auf mehrere Ausgabezeilen aufgeteilt.
      '-----------------------------------------------------------
  154  LOCAL linie$, rest%, spalte%
  155  SHARED aus%,links%,rechts%,zeile%
  156  SHARED lang%, szeile%, remflg%

  157  linie$ = text$                       '! kopiere String
  158  GOSUB pageskip                       '! Seitenvorschub?

  159  PRINT #aus%, SPACE$(links%);         '! auf linken Rand

  160  IF (lang% > 0) AND (NOT remflg%) THEN
  161   INCR zeile%                          '! Zeile im Listing +
1
  162   PRINT #aus%, USING "##### "; zeile%; '! Zeilennummer
drucken
  163  ELSE

  164   PRINT #aus%, SPACE$(6);             '! Leerzeichen
  165  END IF

  166  spalte% = links% + 7                 '! linker Rand setzen

  167  rest% = rechts% - spalte%            '! Restzeilenlänge
  168  CALL skipblank(spalte%,linie$)       '! merke Blanks
  169  PRINT #aus%, LEFT$(linie$,rest%)     '! Ausgabe Teilstring
  170  linie$ = MID$(linie$, rest% + 1)     '! Reststring
  171  INCR szeile%

  172  WHILE LEN(linie$) > rest%            '! String > Zeile
  173   GOSUB pageskip                      '! Seitenvorschub?
  174   PRINT #aus%, SPACE$(spalte%);       '! linker Rand
  175   PRINT #aus%, LEFT$(linie$,rest%)    '! Teilstring ausgeben
  176   linie$ = MID$(linie$,rest% + 1)     '! Reststring bestimmen
  177   INCR szeile%                        '! Zeile im Listing + 1
  178  WEND

  179  IF LEN(linie$) > 0 THEN
  180   GOSUB pageskip                      '! Seitenvorschub?
  181   PRINT #aus%, SPACE$(spalte%);linie$ '! Reststring ausgeben
  182   INCR szeile%                        '! Zeile im Listing + 1
  183  END IF
  184 END SUB


  185 SUB scanner(text$)
```

```
      '!-------------------------------------------------------
      '! Scan Quellcode zeilenweise und trage alle Variablen,
Labels
      '! und Prozeduren in die Referenztabelle ein.
      '!-------------------------------------------------------

  186 LOCAL token%, tokentxt$, ptr%, found%, code%
  187 LOCAL zchn$, number%
  188 SHARED linie$, zeile%

  189 ptr% = 1                                  '! start mit 1.
Zeichen

  190 DO
  191  CALL gettoken(ptr%,linie$,tokentxt$,token%) '! suche token
  192  IF token% THEN
  193   zchn$ = MID$(tokentxt$,1,1)               '! 1. Zeichen
  194   IF (zchn$ <> "$") AND (zchn$ <> "&") THEN   '! ignoriere
$... , &...
  195    CALL keychk(tokentxt$,found%,code%,number%)'!
Schlüsselwort?

  196    IF NOT found% THEN                        '! nur Variable
  197     CALL writeref (tokentxt$,zeile%)         '! schreibe
Zeilennr.
  198    END IF
  199   END IF
  200  END IF
  201 LOOP WHILE (token%)                         '! Ende erreicht?

  202 END SUB

  203 SUB gettoken(ptr%,text$,token$,found%)

      '!-------------------------------------------------------
      '! durchsuche Zeile auf Variablennamen, found% = TRUE falls
      '! Variable gefunden. Der Name wird dann in token
zurückgegeben.
      '!-------------------------------------------------------

  204 LOCAL first%, remflg%, search%            '! lokale Variablen
  205 SHARED lang%                              '! globale
Variablen

  206 found% = %false                           '! init Flag nicht
gefunde
       n
  207 search% = %true                           '! init Flag

  208 WHILE search% AND ptr% <= lang%           '! suche
Variablenanfang
  209  CALL skipblank(ptr%,text$)               '! skip führende
Blanks
  210  IF ptr% >= lang% THEN EXIT SUB           '! Zeilenende ->
Exit
```

```
  211  CALL skiprem(ptr%,text$,remflg%)          '! Kommentar?
  212  IF remflg% THEN EXIT SUB                  '! ja -> Zeile
fertig

  213  IF MID$(text$,ptr%,1) = CHR$(34) THEN     '! String "...."?
  214   CALL skipstring(ptr%,text$)              '! skip string
  215   IF ptr% >= lang% THEN EXIT SUB           '! Ende -> Exit
  216  END IF

  217  zchn$ = UCASE$(MID$(text$,ptr%,1))        '! Zeichen
separieren
  218  IF ((zchn$ < "A") OR (zchn$ > "Z")) _     '! Suche Anfang
Name
  219   AND (INSTR("$&%",zchn$) = 0) THEN        '!    "
  220    INCR ptr%                               '! nein -> next
  221  ELSE
  222   search% = %false                         '! ja -> exit
  223  END IF
  224 WEND

  225 IF search% THEN                           '! gefunden?

  226  EXIT SUB                                 '! nein, Zeilenende
-> Exi
       t
  227 END IF

      '! ### Anfang eines Tokens gefunden ####

  228 first% = ptr%                             '! merke Anfang
token
  229 search% = %true                           '! init Flag

  230 WHILE search% AND ptr% <= lang%           '! suche
Variablenende
  231  zchn$ = UCASE$(MID$(text$,ptr%,1))       '! Zeichen
separieren
  232  IF (zchn$ < "A") OR (zchn$ > "Z") THEN   '! Ende Name?
  233    IF (zchn$ < "0") OR (zchn$ > "9") THEN
  234      IF INSTR("$%&!#",zchn$) = 0 THEN
  235       search% = %false                     '! gefunden
  236       DECR ptr%                            '! korrigiere ptr%
wegen Fehler
  237      END IF                                '! da EXIT WHILE
nicht geht!
  238    END IF
  239  END IF
  240  INCR ptr%                                '! nein -> next

  241 WEND

  242 found% = %true                            '! gefunden !
  243 token$ = MID$(text$,first%,(ptr%-first%)) '! get token
```

```
  244 END SUB

  245 SUB keychk(symbol$,found%,code%,index%)
      '!---------------------------------------------------------
      '! prüfe, ob symbol in der Tabelle mit den Schlüsselwörtern
      '! vorliegt, falls ja -> found% = true, code% = Schlüsselcode
      '! sonst -> found% = false, index% = Nummer in Tabelle
      '! Es wird ein binäres Suchverfahren benutzt.
      '!---------------------------------------------------------

  246 LOCAL low%, high%, ptr% , token$
  247 SHARED keycode%(), keyword$()

  248 low% = 1                                    '! Untergrenze
  249 high% = %maxentry                           '! Obergrenze
  250 found% = %false                             '! not found
  251 token$ = UCASE$(symbol$)                    '! Großbuchst.

  252 WHILE (low% + 1 < high%)                    '! Binärsuche in
Tabelle
  253  ptr% = (high% + low%) / 2                  '! calc. Index
  254  IF token$ = keyword$(ptr%) THEN
  255   found% = %true                            '! gefunden
  256   code% = keycode%(ptr%)                    '! Befehlscode
  257   index% = ptr%                             '! Tabellenindex
  258   EXIT SUB                                  '! Exit
  259  ELSE
  260   IF token$ < keyword$(ptr%) THEN           '! welche Hälfte?
  261    high% = ptr%                             '! neue Obergrenze
  262   ELSE
  263    low% = ptr%                              '! neue Untergrenze
  264   END IF
  265  END IF
  266 WEND

  267  IF token$ = keyword$(low%) THEN            '! erstes Keyword
  268   found% = %true                            '! gefunden
  269   code% = keycode%(low%)                    '! Befehlscode
  270   index% = low%                             '! Tabellenindex
  271   EXIT SUB                                  '! ja -> EXIT
  272  END IF

  273  IF token$ = keyword$(high%) THEN           '! letztes Keyword?
  274   found% = %true                            '! gefunden
  275   code% = keycode%(high%)                   '! Befehlscode
  276   index% = high%                            '! Tabellenindex
  277   EXIT SUB                                  '! ja -> EXIT
  278  END IF

  279 END SUB


  280 SUB writeref(token$,zeile%)
      '!---------------------------------------------------------
      '! Eintrag des Variablennamens und der Zeilennummer
```

```
    '! in die Referenzliste. Ist kein Eintrag vorhanden,
    '! wird ein neuer Eintrag mit dem Namen angelegt, sonst
    '! wird nur die Zeilennummer angehängt.
    '!-------------------------------------------------------

281 LOCAL zchn$, ptr%, code%, tmp%
282 SHARED index1%(), tableline$(), top%


283 zchn$ = UCASE$(MID$(token$,1,1))       '! erstes Zeichen
284 IF zchn$ = "%" THEN
285  ptr% = 0                              '! Index %
286 ELSE
287  ptr% = ASC(zchn$) - 64                '! Index A .. Z
288 END IF

289 IF (ptr% > 26) or (ptr% < 0) THEN
290  PRINT "Fehler : Variablenname falsch "
291  EXIT SUB                              '! Exit
292 END IF

293 IF index1%(ptr%) = %nil THEN           '! kein Eintrag?
294  CALL addtxt(ptr%,1,token$)            '! Eintrag anlegen
295  tableline$(top%) = STR$(zeile%)+" "   '! Eintrag Zeilennr.
296 ELSE
297  tmp% = index1%(ptr%)                  '! erster Satz
298  CALL searchf(tmp%,token$,code%)       '! suche Eintrag
299  IF code% > 0 THEN                     '! neu anlegen?
300   IF code% = 1 THEN                    '! an Anfang
301    CALL addtxt(ptr%,code%,token$)      '! ptr% in index1%()
302   ELSE
303    CALL addtxt(tmp%,code%,token$)      '! ptr% in tableptr()
304   END IF
305   tableline$(top%) = STR$(zeile%)+" "  '! Eintrag Zeilennr.
306  ELSE                                  '! Eintrag vorhanden
307   zchn$ = STR$(zeile%)+" "             '! Zeile in ASCII
308   IF INSTR(tableline$(tmp%),zchn$)_    '! Zeile bereits
309      > 0 THEN EXIT SUB                 '! vorhanden -> Exit
310   tableline$(tmp%) = tableline$(tmp%) + zchn$ '! add Zeilenr
311  END IF
312 END IF

313 END SUB   '! write_ref


314 SUB searchf (ptr%,token$,result%)
    '!-------------------------------------------------------
    '! Suche den Variablennamen in der Liste. Es gelten folgende
    '! Bedingungen: ptr% -> pointer auf den ersten Satz in der
    '! Teilkette. Ergebnisse: result% > 0 -> nicht gefunden
    '!  0  gefunden                   -> ptr% zeigt auf Satz
    '!  1  an Tabellenanfang einfügen -> ptr% zeigt auf Anfang
    '!  2  an Tabellenende anhängen   -> ptr% auf letzten Satz
    '!  3  in Tabelle einfügen        -> ptr% auf Vorgängersatz
    '!-------------------------------------------------------
```

```
  315 LOCAL alt%, varname$, tabname$, last%  '! Hilfsvariable
  316 SHARED tableptr%(), index1%(), tablename$()

  317 varname$ = UCASE$(token$)              '! in Großbuchst.
  318 alt% = ptr%                            '! merke Zeiger
  319 WHILE ptr% <> %nil                     '! bis Ende Liste
  320  tabname$ = UCASE$(tablename$(ptr%))   '! Listenname
  321  IF varname$ > tabname$ THEN           '! gefunden?
  322   last% = ptr%                         '! nein -> merke ptr%
  323   ptr% = tableptr%(ptr%)               '! next entry
  324  ELSE
  325   IF varname$ = tabname$ THEN          '! gefunden?
  326    result% = 0: EXIT SUB               '! ja, ready
  327   ELSE
  328    IF ptr% = alt% THEN                 '! Tabellenanfang
  329     result% = 1: EXIT SUB              '! ja, ready
  330    ELSE
  331     result% = 3                        '! nein, einfügen
  332     ptr% = last%: EXIT SUB             '! ptr% auf Vorgänger
  333    END IF
  334   END IF
  335  END IF
  336 WEND

  337 result% = 2                            '! Satz anhängen
  338 ptr% = last%                           '! ptr% auf letzten
Satz

  339 END SUB   '! search


  340 SUB addtxt(ptr%,code%,varname$)
      '!-------------------------------------------------------
      '! Eintrag eines neuen Variablennamens in die Liste
      '! ptr% zeigt auf den Satz hinter dem eingefügt wird.
      '! code bestimmt den Einfügemode:
      '!  1  an Tabellenanfang einfügen -> ptr% in index1% (root)
      '!  2  an Tabellenende anhängen   -> ptr% in tableptr% (last)
      '!  3  in Tabelle einfügen        -> ptr% in tableptr% (pred)
      '! varname$ enthält den Variablennamen
      '!-------------------------------------------------------

  341 SHARED top%, tableptr%(), tablename$()
  342 SHARED index1%()


  343 INCR top%                               '! auf nächsten Satz
  344 IF top% > %tablen THEN CALL full        '! Überlauf
  345 IF code% = 2 THEN                       '! anhängen
  346  tableptr%(top%) = %nil                 '! Ende Liste
  347  tableptr%(ptr%) = top%                 '! Link Folgesatz
  348 ELSE
  349  IF code% = 1 THEN                      '! erster Satz
  350   tableptr%(top%) = index1%(ptr%)       '! Link Folgesatz
```

```
  351   index1%(ptr%) = top%                    '! Link Kopf
  352  ELSE
  353   IF code% = 3 THEN                       '! zwischen Sätze
  354    tableptr%(top%) = tableptr%(ptr%)      '! Link Folgesatz
  355    tableptr%(ptr%) = top%                 '! Link neuen Satz
  356   ELSE
  357    PRINT "Fehler: falscher Code in add"
  358    END                                    '! Fehlerausgang
  359   END IF
  360  END IF
  361 END IF

  362 tablename$(top%) = varname$              '! Name eintragen

  363 END SUB

  364 SUB full
      '! Fehlerabbruch bei Überlauf der Tabellen
  365  PRINT "Fehler : interner Tabellenüberlauf ####"
  366  END
  367 END SUB


  368 addtable:
      '!-----------------------------------------------------------
      '! erzeuge Referenztabelle
      '!-----------------------------------------------------------

  369  PRINT #aus%, CHR$(12)                  '! Seitenwechsel
  370  PRINT #aus%, "X R E F - T a b e l l e"; SPACE$(27);
  371  PRINT #aus%, "(c) Born Version 1.0"
  372  PRINT #aus%, "Datei : ";filename$;"     Datum : ";DATE$;
  373  PRINT #aus%, "         Seite : "; seite%
  374  PRINT #aus%,
  375  INCR seite%                              '! Seite + 1
  376  szeile% = 3                              '! 3 Kopfzeilen

      '!--- gebe die Referenztabelle formatiert aus

  377  FOR i% = 0 TO 26                         '! über
Indextabelle
  378   ptrx% = index1%(i%)                     '! Index in Tabelle
  379   WHILE ptrx% <> %nil                     '! Liste bis Ende
  380    PRINT #aus%, tablename$(ptrx%)         '! Name ausgeben
  381    INCR szeile%                           '! Zeilenzähler + 1
  382    GOSUB kopf                             '! Seitenwechsel?
  383    linie$ = tableline$(ptrx%)             '! hole Zeilennr.
  384    rest% = rechts% - links%               '! Drucklänge
  385    WHILE LEN(linie$) > rest%              '! formatiert
ausgeben
  386     GOSUB kopf                            '! Seitenwechsel?
  387     PRINT #aus%, SPACE$(links%);          '! linker Rand
  388     tmpx% = rest%                         '! Drucklänge
merken
```

```
  389     WHILE (MID$(linie$,tmpx%,1) <> " ")_  '! vermeide daß
letzte
  390       AND (tmpx% > 9)                     '! Zahl
abgeschnitten
  391       DECR tmpx%                          '! wird
  392     WEND
  393     PRINT #aus%, MID$(linie$,1,tmpx%)     '! Teilstring
  394     linie$ = MID$(linie$,tmpx%+1)         '! Rest holen
  395     INCR szeile%                          '! Zeilennr. + 1
  396    WEND
  397    IF LEN(linie$) > 0 THEN
  398     GOSUB kopf                            '! Seitenwechsel
  399     PRINT #aus%, SPACE$(links%);linie$    '! Rest ausgeben
  400     PRINT #aus%,
  401     INCR szeile%, 2
  402    END IF
  403    ptrx% = tableptr%(ptrx%)               '! next entry
  404   WEND
  405  NEXT i%                                  '! next index

  406  szeile% = szeile% + 7                    '! 7 Zeilen res.
  407  GOSUB kopf                               '! Seitenwechsel
  408  PRINT #aus%,
  409  PRINT #aus%,"XREF Modul Information"
  410  PRINT #aus%,
  411  PRINT #aus%,"Lines read    : ";zeile%
  412  PRINT #aus%,"Symbols found : ";top%
  413  PRINT #aus%,
  414  PRINT #aus%,"End XREF"
  415  PRINT #aus%, CHR$(12)                    '! Seitenvorschub

  416 RETURN
  417 kopf:
      '!-------------------------------------------------
      '! Ausgabe des Kopfes für die Referenzliste
      '!-------------------------------------------------

  418  IF szeile% < maxzeile% THEN RETURN        '! Seitenwechsel?

  419  PRINT #aus%, CHR$(12)                  '! Seitenwechsel
  420  PRINT #aus%, "Datei : ";filename$;"     Datum : ";DATE$;
  421  PRINT #aus%, "         Seite : "; seite%
  422  PRINT #aus%,
  423  szeile% = 2                              '! 2 Kopfzeilen
  424  INCR seite%

  425 RETURN

  426 SUB skipblank(ptr%,text$)
      '-------------------------------------------------------
      '! zähle führende Blanks in einer Zeichenkette
      '! text$ = Zeichenkette, zeiger% = Zeiger in Kette
      '-------------------------------------------------------
  427 SHARED lang%
```

```
  428 WHILE (ptr% =< lang%) and (MID$(text$,ptr%,1) = " ")
  429  INCR ptr%
  430 WEND
  431 END SUB

  432 SUB skipstring(ptr%,text$)
      '!-------------------------------------------------------
      '! Es wird geprüft, ob ptr% auf ein " im Text zeigt. In
diesem
      '! Fall liegt ein String "...." vor, dessen Ende (") gesucht
      '! wird. ptr% zeigt nach dem Ablauf auf das Zeichen hinter ".
      '! text$ = Zeichenkette mit Anweisung
      '!-------------------------------------------------------
  433 SHARED lang%

  434 IF MID$(text$,ptr%,1) <> CHR$(34) THEN
  435  EXIT SUB                              '! kein String "..."
  436 END IF

  437 DO                                     '! suche Ende String
  438  INCR ptr%                             '! next char
  439 LOOP UNTIL (MID$(text$,ptr%,1) = CHR$(34)) OR (ptr% >= lang%)

  440 END SUB

  441 SUB skiprem (ptr%,text$,flag%)

      '!-------------------------------------------------------
      '! prüfe auf Kommentare, text$ = String mit Anweisung
      '! ptr% = Zeiger in Text, flag% = true -> Kommentar gefunden
      '!-------------------------------------------------------

  442  CALL skipblank(ptr%,text$)        '! führende blanks
entfernen
  443  IF INSTR(text$,"REM") = ptr% THEN '! scan Anfang = REM oder
'
  444   flag% = %true                    '! Kommentar
  445  ELSE
  446   IF MID$(text$,ptr%,1) = "'" THEN
  447    flag% = %true                    '! Kommentar
  448   ELSE
  449    flag% = %false                   '! kein Kommentar
  450   END IF
  451  END IF
  452 END SUB

      '!-------------------------------------------------------
      '! Data-Anweisungen mit den reservierten Schlüsselwörtern
      '! von PowerBASIC. Der erste Wert enthält das Schlüsselwort,
      '! während der zweite Wert angibt, ob es sich um ein Funktion
      '! oder ein Schlüsselwort für einen Befehl handelt:
      '! Bsp.:     "IF"   , 0   '! Schlüsselwort
      '!          "CHR$" , 1  '! Basic-Funktion
      '!-------------------------------------------------------
      '
```

```
453 DATA "ABS"     ,1, "ABSOLUTE" ,0, "AND"    ,0, "APPEND"  ,0
454 DATA "ARRAY"   ,0, "AS"       ,0, "ASC"    ,1, "ASCEND"  ,0
455 DATA "ASCII"   ,1, "ATN"      ,1, "ATTRIB" ,0, "BASE"    ,0
456 DATA "BEEP"    ,0, "BIN$"     ,1, "BINARY" ,0, "BLOAD"   ,0
457 DATA "BLOCK"   ,0, "BSAVE"    ,0, "CALL"   ,0, "CASE"    ,0
458 DATA "CBCD"    ,0, "CDBL"     ,1, "CEIL"   ,1, "CEXT"    ,1
459 DATA "CFIX"    ,1, "CHAIN"    ,0, "CHDIR"  ,0, "CHR$"    ,1
460 DATA "CINT"    ,1, "CIRCLE"   ,1, "CLEAR"  ,0, "CLNG"    ,1
461 DATA "CLOSE"   ,0, "CLS"      ,0, "COLOR"  ,0, "COLLATE" ,0
462 DATA "COM"     ,0, "COMMAND$" ,1, "COMMON" ,1, "COS"     ,1
463 DATA "CQUD"    ,1, "CSNG"     ,1, "CSRLIN" ,1, "CURDIR$" ,0
464 DATA "CVB"     ,1, "CVD"      ,1, "CVE"    ,1, "CVF"     ,1
465 DATA "CVI"     ,1, "CVL"      ,1, "CVMD"   ,1, "CVMS"    ,1
466 DATA "CVQ"     ,1, "CVS"      ,1, "DATA"   ,0, "DATE$"   ,1
467 DATA "DECLARE" ,0, "DECR"     ,0, "DEF"    ,0, "DEFBCD"  ,0
468 DATA "DEFDBL"  ,0, "DEFEXT"   ,0, "DEFFIX" ,0, "DEFFLX"  ,0
469 DATA "DEFINT"  ,0, "DEFLNG"   ,0, "DEFQUD" ,0, "DEFSNG"  ,0
470 DATA "DEFSTR"  ,0, "DELAY"    ,0, "DELETE" ,0, "DESCEND" ,0
471 DATA "DIM"     ,0, "DIR"      ,0, "DO"     ,0, "DRAW"    ,1
472 DATA "DYNAMIC" ,0, "ELSE"     ,0, "ELSEIF" ,0, "END"     ,0
473 DATA "ENDMEM"  ,0, "ENVIRON"  ,1, "ENVIRON$",1,"EOF"     ,0
474 DATA "EQV"     ,1, "ERADR"    ,1, "ERASE"  ,0, "ERDEV"   ,0
475 DATA "ERDEV$"  ,0, "ERL"      ,0, "ERR"    ,0, "ERROR"   ,0
476 DATA "EXECUTE" ,0, "EXIT"     ,0, "EXP"    ,1, "EXP10"   ,1
477 DATA "EXP2"    ,1, "EXTERNAL" ,0, "EXTRACT$",0,"FIELD"   ,0
478 DATA "FILEATTR",0, "FILES"    ,0, "FIX"    ,1, "FN"      ,0
479 DATA "FOR"     ,0, "FRE"      ,0, "FREEFILE",0, "FUNCTION",0
480 DATA "GET"     ,0, "GET$"     ,0, "GOSUB"   ,0, "GOTO"   ,0
481 DATA "HEX$"    ,1, "IF"       ,0, "INCR"    ,0, "INKEY$" ,0
482 DATA "INLINE"  ,0, "INP"      ,1, "INPUT"   ,0, "INPUT#" ,0
483 DATA "INPUT$"  ,0, "INSERT"   ,0, "INSTAT"  ,0
484 DATA "INSTR"   ,1, "INT"      ,0, "INTERRUPT",0,"IOCTL"  ,0
485 DATA "IOCTL$"  ,0, "KEY"      ,0, "KILL"    ,0, "LBOUND" ,0
486 DATA "LCASE$"  ,1, "LEFT$"    ,1, "LEN"     ,1, "LET"    ,0
487 DATA "LINE"    ,0, "LIST"     ,0, "LLIST"   ,0, "LOC"    ,0
488 DATA "LOCAL"   ,0, "LOCATE"   ,0, "LOCK"    ,0, "LOF"    ,0
489 DATA "LOG"     ,1
490 DATA "LOG10"   ,1, "LOG2"     ,1, "LOOP"    ,0, "LPOS"   ,1
491 DATA "LPRINT"  ,0, "LSET"     ,0, "LTRIM"   ,1, "MAP"    ,0
492 DATA "MAX"     ,1, "MAX%"     ,1, "MAX$  " ,1, "MEMSET"  ,0
493 DATA "MID$"    ,1, "MIN"      ,1, "MIN%"    ,1, "MIN$"   ,1
494 DATA "MKDIR"   ,0, "MKB$"     ,1, "MKD$"    ,1, "MKE$"   ,1
495 DATA "MKF$"    ,1, "MKI$"     ,1, "MKL$"    ,1, "MKMD$"  ,1
496 DATA "MKMS$"   ,1, "MKQ$"     ,1, "MKS$"    ,1, "MOD"    ,0
497 DATA "MTIMER"  ,0
498 DATA "NAME"    ,0, "NEXT"     ,0, "NOT"     ,0, "OCT$"   ,1
499 DATA "OFF"     ,1, "ON"       ,1, "OPEN"    ,0, "OPTION" ,0
500 DATA "OR"      ,0, "OUT"      ,0, "OUTPUT"  ,0, "PAINT"  ,0
501 DATA "PALETTE" ,0, "PEEK"     ,0, "PEEKI"   ,0, "PEEKL"  ,0
502 DATA "PEEK$"   ,0, "PEN"      ,0, "PLAY"    ,0, "PMAP"   ,0
503 DATA "POINT"   ,0, "POKE"     ,0, "POKEI"   ,0, "POKEL"  ,0
504 DATA "POKE$"   ,0, "POS"      ,0, "PRESET"  ,0, "PRINT"  ,0
505 DATA "PRINT#"  ,0, "PSET"     ,0, "PUBLIC"  ,0
506 DATA "PUT"     ,0, "PUT$"     ,0, "RANDOM"  ,0, "RANDOMIZE",0
507 DATA "READ"    ,0, "RECURSIVE",0, "REDIM"   ,0, "REG"    ,0
```

```
508 DATA "REM"     ,0, "REMOVE$"  ,1, "REPEAT$",0, "REPLACE" ,0
509 DATA "RESET"   ,0, "RESTORE"  ,0, "RESUME" ,0, "RETURN"  ,0
510 DATA "RIGHT$"  ,1, "RMDIR"    ,0, "RND"    ,1, "ROUND"   ,1
511 DATA "RSET"    ,0, "RSET"     ,0, "RTRIM$" ,1, "RUN"     ,0
512 DATA "SAVE"    ,0, "SCAN"     ,0, "SCREEN" ,0, "SEEK"    ,0
513 DATA "SEG"     ,0, "SELECT"   ,0, "SERVICE",0, "SGN"     ,1
514 DATA "SHARED"  ,0, "SHELL"    ,0, "SIN"    ,1, "SORT"    ,0
515 DATA "SOUND"   ,0, "SPACE$"   ,1, "SPC"    ,1, "SQR"     ,1
516 DATA "STATIC"  ,1, "STEP"     ,0, "STICK"  ,0, "STOP"    ,0
517 DATA "STR$"    ,1, "STRIG"    ,0, "STRING$",0, "STRPTR"  ,1
518 DATA "STRSEG"  ,1, "SUB"      ,0, "SWAP"   ,0
519 DATA "SYSTEM"  ,0, "TAB"      ,0, "TALLY"  ,1, "TAN"     ,1
520 DATA "THEN"    ,0
521 DATA "TIME$"   ,0, "TIMER"    ,0, "TO"     ,0, "TROF"    ,0
522 DATA "TRON"    ,0, "UBOUND"   ,0, "UCASE$" ,1, "UNTIL"   ,0
523 DATA "UNTIL"   ,0
524 DATA "USING"   ,0, "USR"      ,0, "USR0"   ,0, "USR1"    ,0
525 DATA "USR2"    ,0, "USR3"     ,0, "USR4"   ,0, "USR5"    ,0
526 DATA "USR6"    ,0, "USR7"     ,0, "USR8"   ,0, "USR9"    ,0
527 DATA "VAL"     ,1, "VARPTR"   ,0, "VARPTR$",0, "VARSEG"  ,0
528 DATA "VERIFY"  ,1
529 DATA "VIEW"    ,0, "WAIT"     ,0, "WEND"   ,0, "WHILE"   ,0
530 DATA "WIDTH"   ,0, "WINDOW"   ,0, "WRITE"  ,0, "WRITE#"  ,0
531 DATA "XOR"     ,0

    '***** Programm Ende ******
```

Listing 2.6: XREF.BAS

XFORM: Formatierung von Quellprogrammen

Die vorgestellten Lösungen zur Ausgabe von Listings (LISTER, SPOOL) sowie der Cross-Referenz-Generator bilden sicherlich die Basis für die Softwareentwicklung mit PowerBASIC. Allerdings stieß ich bei der Erstellung der Programme für das vorliegende Buch auf einen Mangel des Compilers, der mir bereits häufig bei anderen Sprachen aufgestoßen ist:

Bei der Entwicklung der Programmablaufsteuerungen werden geschachtelte IF-Anweisungen oder Strukturelemente wie DO, WHILE etc. verwendet. Sofern diese Schachtelung nicht stimmt, d.h. auf jedes IF folgt in der Regel ein Abschluß mit END IF, tritt bei der Übersetzung eine Fehlermeldung auf. Allerdings zeigt der Compiler die Fehlermeldung in der Regel in einer Programmzeile, die mit Sicherheit nichts mehr mit dem fehlenden END IF zu tun hat. Dies ist erklärbar, wenn man sich die Arbeitsweise des Compilers vor Augen hält. In der Praxis führt ein solches Verhalten - insbesondere bei größeren oder komplexeren Programmen - zu einer aufwendigen Fehlersuche. Wer gewissenhaft arbeitet und die Schachtelungen jeweils optisch einrückt, kann hier zwar leichter durch die Strukturen durchfinden. Aber trotz dieser manuell durchgeführten Formatierung meiner Quelldateien trat bei PowerBASIC ein Problem auf, welches mich immer wieder nervt:

Die Syntax der Sprache definiert als Abschluß einer Struktur die Schlüsselwörter:

```
END IF
SUB END
...
```

Da ich häufig in anderen Sprachen programmiere, schreibe ich gewohnheitsmäßig das Schlüsselwort:

```
ENDIF
```

PowerBASIC erkennt dies nicht als END IF und generiert vermutlich eine Variable (oder sonst etwas unsinniges). Bei der Übersetzung tritt eine Fehlermeldung an den unmöglichsten Stellen auf. Die Analyse des Listings ist dann sehr aufwendig, da ich gewohnheitsmäßig auf die Struktur der Einrückungen und auf die Anweisungen schaue. Das fehlende Leerzeichen wird dann leicht übersehen.

Dies war schließlich der Grund, über ein Programm nachzudenken, welches die oben geschilderte Fehlersuche unterstützt.

Die Spezifikation

Zu Beginn steht wieder die Frage, welche Funktionalität benötigt wird?

Das Programm soll eine PowerBASIC-Quelldatei einlesen, bearbeiten und wieder in eine zweite Datei speichern. Was soll aber während der Bearbeitung passieren? Nehmen wir an, Sie haben folgendes kleine Programm vorliegen:

```
IF a% > b% THEN
a% = a% * 20
FOR i% = 1 TO 20
b% = b% * i%
PRINT "Index ", i%
NEXT i%
END IF
```

Die Anweisungen sind in Basic-Manier alle linksbündig aufgeführt. Zwar benötigt PowerBASIC keine Zeilennummern mehr, aber viele Basic-Programme sehen nach wie vor so aus. Während der Bearbeitung soll nun ein strukturiertes Listing entstehen. So sind alle Anweisungen, die innerhalb eines Blockes stehen, einzurücken:

```
IF a% > b% THEN
  a% = a% * 20
  FOR i% = 1 TO 20
    b% = b% * i%
    PRINT "Index ", i%
  NEXT i%
END IF
```

Das kleine Beispiel zeigt bereits, daß durch die Einrückung die Struktur wesentlich deutlicher zum Vorschein tritt. Insbesondere lassen sich die einzelnen Blöcke leicht identifizieren und ein Wechsel der Schachtelungstiefe ist einfach erkennbar. Zusätzlich erhalten Sie ein sauber strukturiertes Quellprogramm, welches für weitere Entwicklungen verwendbar ist.

Was ist aber mit Programmen, die bereits von Hand mit diesen Einrückungen versehen sind? Die Programme in diesem Buch weisen zum Beispiel manuelle Einrückungen auf. Hier wäre es für den Fall der Fälle ebenfalls hilfreich, wenn die Quelldatei sich in der Struktur neu aufbereiten ließe. Bei der Bearbeitung müssen dann alle Einrückungen im Original entfernt und durch den Generator gemäß der Schachtelungstiefe neu formatiert werden. Nach der Implementierung des Programmes habe ich dies versucht und es funktioniert erstaunlich gut.

Bezüglich der oben beschriebenen Problematik der falsch geschriebenen Anweisungen END IF reichte mir diese Funktionalität aber nicht aus. Wenn das Programm bereits die Einrückungen vornimmt, kennt es offensichtlich die Schachtelungstiefe. Für die Fehlersuche ist es daher hilfreich, wenn optional eine sogenannte Levelnummer vor jeder Zeile ausgegeben werden kann.

```
0 IF a% > b% THEN
1   a% = a% * 20
1   FOR i% = 1 TO 20
2     b% = b% * i%
2     PRINT "Index ", i%
1   NEXT i%
0 END IF
```

Das kleine Beispiel zeigt das Prinzip: Am Beginn der Einrückung wird die Levelnummer erhöht und am Ende des Blockes wieder erniedrigt. So ist sofort ersichtlich, wenn die Schachtelung nicht mit dem geplanten Programmzustand übereinstimmt. Da die Analyse der Zeilen einen Parser voraussetzt, muß diesem die eindeutige Schreibweise der Schlüsselwörter mitgeteilt werden. Das bedeutet andererseits, daß falsch geschriebene Wörter nicht als Basic-Schlüsselwörter erkannt werden. Dies macht sich dann in der Struktur und in der Levelnummer bemerkbar.

Bezüglich der Aufrufschnittstellen wird die bei den anderen Programmen verwendete Technik benutzt. Wird das Programm mit der Eingabe:

```
XFORM
```

aktiviert, ist der Bildschirm zu löschen und die folgende Kopfmeldung auszugeben:

```
X F O R M                              (c) Born Version 1.0

Optionen [  /L Levelnummer einblenden   ]

Eingabedatei :
```

```
Ausgabedatei :
Option       :
```

Bild 2.16: Kopfmeldung des Programmes XFORM

Als Dateiname dürfen gültige MS-DOS-Bezeichnungen einschließlich Laufwerks- und Pfadbezeichnungen verwendet werden. Quell- und Zieldatei dürfen jedoch keine identischen Namen aufweisen, sonst erscheint die Fehlermeldung:

```
Eingabedatei = Ausgabedatei nicht erlaubt
```

Auch sollte die Quelldatei gültige Basic-Anweisungen enthalten. Die Abfrage der »Optionen« erscheint erst nach Eingabe der Dateinamen. Wird der Schalter /L gesetzt, wird vor jeder Zeile die Levelnummer mit eingeblendet.

Alternativ kann der Aufruf auch direkt von DOS erfolgen, wodurch eine Verwendung in Batchdateien möglich wird. Sobald Dateinamen in der Kommandozeile auftreten, übernimmt XFORM diese Namen und beginnt mit der Bearbeitung. Für diesen Aufruf gilt folgende Syntax:

```
XFORM Quelldatei Zieldatei </L>
```

Auf jeden Fall müssen zwei Dateinamen eingegeben werden, ansonsten erscheint eine Fehlermeldung:

```
Der Name der Ausgabedatei fehlt
```

Wird die Option /L gesetzt, schaltet XFORM die Generierung von Levelnummern ein.

Als Randbedingung gilt, daß die Dateinamen immer zuerst einzugeben sind. Die Option muß durch mindestens ein Leerzeichen vom Dateinamen getrennt werden. Die folgenden Eingaben:

```
XFORM XFORM.BAS XFORM.LEV
XFORM XFORM.BAS XFORM.LEV /L
```

stellen gültige Aufrufe des Programmes dar.

Als letzter Punkt soll das Programm noch eine Online-Hilfe bieten. In Anlehnung an DOS 5.0 wird diese Online-Hilfe mit der folgenden Option aufgerufen:

```
XFORM /?
```

Dann muß auf dem Bildschirm folgender Hilfstext erscheinen:

```
X F O R M                        (c) Born Version 1.0

Aufruf: XFORM <Eingabefile> <Ausgabefile> </L>

XFORM liest eine PowerBASIC Quelldatei ein und rückt die
Anweisungen zwischen:
```

```
FOR         NEXT
DO          LOOP
WHILE       WEND
IF/THEN     ELSEIF / END IF
SUB         END SUB
```

ein. Dadurch entsteht ein formatiertes Listing, welches sich besser lesen läßt. Über die Option /L kann die Schachtelungstiefe jeder Zeile ausgegeben werden. Danach bricht das Programm ab und der DOS-Prompt erscheint wieder.

Die Implementierung

Das Programm ist zweckmäßigerweise in mehrere Module zu unterteilen, deren Zusammenschaltung in Bild 2.17 gezeigt wird. Insbesondere kann auf viele Techniken zurückgegriffen werden, die bereits aus den bisher vorgestellten Programmen stammen. Gerade die Analyse einer Zeile und Separierung eines Schlüsselwortes (*token*) wurde ausgiebig im Programm XREF genutzt. Daher werden diese Module in XFORM wieder auftauchen.

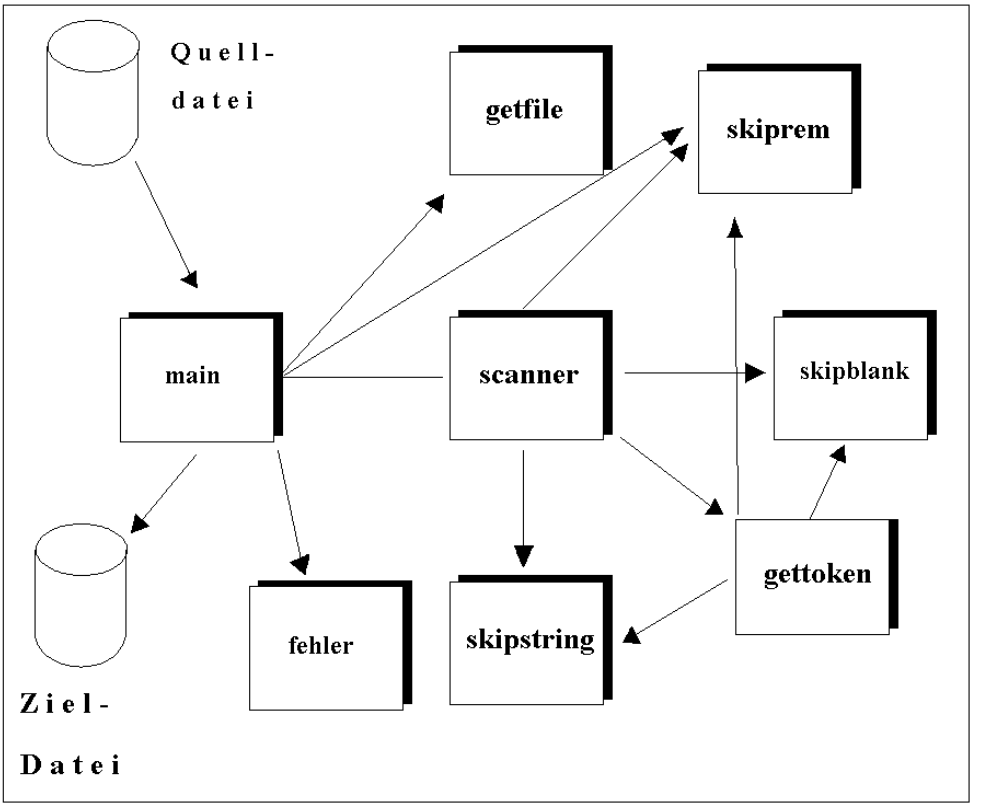


Bild 2.17: Modulhierarchie des Programmes XFORM

Das Programm XFORM besitzt die oben gezeigte Modulstruktur. Aufbau und Funktion dieser Module sollen nun kurz beschrieben werden.

Hauptprogramm

Die Steuerung der Benutzereingaben sowie die Ablaufsteuerung übernimmt wie bisher das Modul *main*. Weiterhin sind in diesem Modul die Variablen zu definieren und zu initialisieren. Falls die Dateinamen in der Kommandozeile enthalten sind, müssen diese separiert werden. In diesem Fall ist auf die Ausgabe der Kopfmeldung zu verzichten. Andernfalls erscheint die Kopfmeldung und der Name der Dateien sowie die Optionen werden abgefragt. Durch die Angabe:

```
 ON ERROR GOSUB fehler
```

werden mögliche Fehler abgefangen. Hierzu zählen auch nicht vorhandene Quelldateien. Mit der Schleife

```
WHILE NOT (EOF(ein%))
 LINE INPUT #ein%, linie$
 CALL skiprem(1,linie$,remflg%)
 lang% = LEN(linie$)
 IF (lang% > 0) AND (NOT remflg%) THEN
  CALL scanner(linie$)
 END IF
 IF (INSTR(UCASE$(kommando$),"/L") > 0) THEN
  PRINT #aus%, USING "## "; lev1%;
  lev1% = lev2%
 END IF
 PRINT #aus%, linie$
WEND
```

wird die Datei zeilenweise gelesen und bearbeitet. Das Modul »skiprem« sorgt dafür, dass Kommentare überlesen werden.

Hinweis: Die Implementierung erkennt Kommentarzeilen zur Zeit nur, wenn sie direkt mit einem Kommentarzeichen eingeleitet werden.

Alle Zeilen, die nicht leer oder als Kommentar erkennbar sind, werden anschließend durch den *Scanner* (siehe gleichnamiges Unterprogramm) bearbeitet. Hier erfolgt die Formatierung der jeweiligen Blöcke. Falls die Option /L gesetzt ist, gibt das Hauptprogramm noch den Levelcode als zweistellige Zahl vor jeder Zeile aus. Der Levelcode der aktuellen Zeile steht in *lev1%*. In *lev2%* findet sich der Levelcode der Folgezeile. Diese Konstruktion ist erforderlich, da bei der Ausgabe einer Zeile am Blockbeginn (z.B. IF ...) noch der alte Levelcode gültig ist, der Scanner aber bereits den Level für die Folgezeile verändert. Ist das Dateiende erreicht, wird die Schleife verlassen, die Dateien geschlossen und das Programm beendet.

Die Hilfsmodule

Als Hilfsmodule gelangen unter anderem Routinen aus dem Cross-Referenz-Generator zum Einsatz (*skiprem*, *gettoken*, *skipstring*). Die Tabellen mit den PowerBASIC-Schlüsselwörtern werden in XFORM nicht

genutzt. Es sind nur wenige Schlüsselwörter von Interesse, so daß die *tokens* direkt auf diese Schlüsselwörter abgeprüft werden.

fehler

Dieses Modul bildet den Fehlerausgang des Programmes LISTER. Es wird immer dann durch Basic aktiviert, falls Laufzeitfehler auftreten.

getfile

Das Unterprogramm *getfile* wird nur benutzt, falls Parameter in der Kommandozeile auftreten. Dann übernimmt das Modul die Separierung der beiden Bezeichnungen aus der Kommandozeile. Pro Aufruf wird ein Filename zurückgegeben. Der Parameter *ptr%* enthält einen Zeiger auf den Anfang des zu analysierenden Teilstrings. Der Dateiname muß durch das Zeilenende oder ein Leerzeichen abgeschlossen werden.

Scanner

Dieses Modul analysiert jede eingelesene Quellzeile auf Schlüsselwörter, die die Einrückungsstufe beeinflussen. Die Zerlegung der Eingabezeile in einzelne *tokens* erfolgt mit dem Modul *gettoken*. Für die einzelnen Schlüsselwörter gelten einige Besonderheiten, die ich nachfolgend kurz vorstellen möchte.

FOR/NEXT

Wird als erstes das Schlüsselwort FOR erkannt, rückt das Programm die Folgezeile um *n* Zeichen ein. Die Zahl *n* ist zur Zeit durch die Variable *indent%* auf 2 festgelegt. Mit NEXT wird die Einrückung wieder erniedrigt. Die gegenwärtige Implementierung geht jedoch davon aus, daß die FOR ..NEXT-Sequenz sich über mehrere Zeilen erstreckt. Sofern die NEXT-Anweisung in der FOR-Zeile steht, wird dies nicht erkannt. Bei Bedarf können Sie diese Eigenschaft aber nachrüsten. Es müssen lediglich alle *tokens* der Zeile auf das Schlüsselwort NEXT überprüft werden. Ein ähnliches Beispiel findet sich bei der Analyse der IF-Anweisung.

DO/LOOP, WHILE/WEND

Für diese Kombinationen von Schlüsselwörter gilt das gleiche wie für FOR .. NEXT.

IF .. THEN .. ELSE, ELSEIF, END IF

Bei der IF-Bedingung sind einige Besonderheiten zu beachten. Die Sequenz:

```
IF .... THEN
 ....
ELSE
 ....
END IF
```

führt zu einem bestimmten Einrückmuster. ELSE reduziert dabei die Schachtelungstiefe nur für die eigene Zeile. Das gleiche gilt für ELSEIF.

Bei END IF ist zu beachten, daß dieses Schlüsselwort sich aus zwei *tokens* zusammensetzt. Bei IF ist zusätzlich der Sonderfall zu beachten, daß sich die Anweisung auf eine Zeile beschränkt. Dann folgt keine END IF-Anweisung:

```
IF a% > 0 THEN b% = 100 / a%
```

Hier darf die Schachtelungstiefe nicht verändert werden. Sobald ein IF erkannt wurde, sucht das Modul die Zeile nach weiteren *tokens* ab. Enthält das letzte *token* den Text THEN, liegt eine mehrzeilige IF-Anweisung vor. Dann wird die folgende Zeile eingerückt.

Der Sonderfall, daß eine IF-Anweisung nur eine Zeile umfassen soll, aber über ein Fortsetzungssymbol auf mehrere Teilzeilen aufgeteilt wurde, ist nicht implementiert:

```
IF a% > b% _
 and  c% = 0 _
 and  d% > 1 THEN a% = 0
```

Hier geht die Bearbeitung durch XFORM schief. Falls Sie diese Form benutzen, sollten Sie XFORM erweitern.

gettoken

Diese Modul übernimmt die Zerlegung einer Zeile in einzelne *tokens* und stammt aus dem Programm XREF. Analoges gilt für *skipblank* und *skiprem*.

Erweiterungsvorschläge

Im Text wurde bereits auf einige Einschränkungen hingewiesen. So werden Kommentarzeilen nur erkannt, falls sie in der ersten Spalte beginnen. Weiterhin läßt sich die Bearbeitung der IF-Anweisungen verbessern. Analoges gilt für die WHILE- und FOR-Schleifen, die sich nur eine Zeile beziehen.

Eine letzte Feinheit besteht in der Markierung der Schachtelungsebenen durch Linien:

```
  +-IF a% > b% THEN
  |   a% = 0
  + END IF
```

Dies ist nicht sonderlich schwierig, Sie müssen lediglich die Ausgabe von Leerzeichen durch die entsprechenden Sonderzeichen ersetzen. Diese Änderungen sind im Modul *scanner* auszuführen.

```
X R E F   /Z=50                          (c) Born Version 1.0
Datei : xform.bas      Datum : 05-13-1992         Seite :  1

 Zeile     Anweisung
```

```
      '*********************************************************
      '! File       : XFORM.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 2. 5.92
      '! Autor      : G. Born
      '! Files      : INPUT, OUTPUT
      '! Progr. Spr.: PowerBASIC 2.x
      '! Betr. Sys. : DOS 2.1 - 5.0 (+ DR-DOS 5.0/6.0)
      '! Funktion: Das Programm liest den PowerBASIC Quellcode
      '!           ein und rückt alle Anweisungen zwischen
      '!
      '!              FOR        NEXT
      '!              DO         LOOP
      '!              WHILE      WEND
      '!              IF/THEN    ELSEIF/END IF
      '!              SUB        END SUB
      '!
      '!           ein.
      '!
      '! Aufruf:   XFORM Filename1 Filename2 /L
      '!
      '!           Wird das Programm ohne Parameter aufgerufen,
      '!           sind die Dateinamen explizit abzufragen.
      '*********************************************************
      '! Variable und Konstanten definieren
    1 %true = &HFFFF: %false = 0
    2 spalte% = 0                          '! '! Einrück. pro
Stufe
    4 remflg% = %false                     '! Kommentar gefunden
    5 lang% = 0                            '! Länge Zeile
    6 lev1% = 0                            '! Level
    7 lev2% = 0
    8 ein% = 1                             '! Dateinummer Eingabe
    9 aus% = 2                             '! Dateinummer Ausgabe

   10 ON ERROR GOTO fehler                 '! Fehlerausgang

      '############################################################
      '#                      Hauptprogramm                       #
      '############################################################

   11 kommando$ = COMMAND$                 '! Parameter?
   12 IF LEN (kommando$) = 0 THEN          '! User Mode?
   13  CLS                                 '! clear Screen

   14  PRINT "X F O R M                        (c) Born Version
1.0"
   15  PRINT
   16  INPUT  "Eingabedatei: ",filename1$
   17  INPUT  "Ausgabedatei: ",filename2$
   18  INPUT  "Option      : ",kommando$
   19  PRINT
   20 ELSE                                 '! Kommando Mode
   21  ptr% = INSTR (kommando$,"/?")       '! Option /?
   22  IF ptr% <> 0 THEN                   '! Hilfsbildschirm
```

```
   23   PRINT "X F O R M                          (c) Born Version 1.0"
   24   PRINT
   25   PRINT "Aufruf: XFORM <Eingabefile> <Ausgabefile> </L>"
   26   PRINT
   27   PRINT "XFORM liest eine PowerBASIC Quelldatei ein und rückt
die"
   28   PRINT "Anweisungen zwischen:
   29   PRINT
   30   PRINT "       FOR        NEXT"
   31   PRINT "       DO         LOOP"
   32   PRINT "       WHILE      WEND"
   33   PRINT "       IF/THEN    ELSEIF/END IF"
   34   PRINT "       SUB        END SUB"
   35   PRINT
   36   PRINT "ein. Dadurch entsteht ein formtierters Listing,
welches sich"
   37   PRINT "besser lesen läßt. Über die Option /L kann die
Schachtelung"
   38   PRINT "jeder Zeile ausgegeben werden."
   39   PRINT
   40   SYSTEM
   41  END IF

      '! separiere Bezeichnungen

   42  ptr% = 1                                    '! Parameter holen
   43  CALL getfile(ptr%, kommando$,filename1$)  '! Name
Eingabedatei1
   44  INCR ptr%
   45  CALL getfile(ptr%, kommando$,filename2$)  '! Name
Ausgabedatei
   46 END IF

   47 IF filename1$ = "" THEN              '! Leereingabe?
   48  PRINT "Der Name der Eingabedatei fehlt"
   49  END
   50 END IF

   51 IF filename2$ = "" THEN              '! Leereingabe?
   52  PRINT "Der Name der Ausgabedatei fehlt"
   53  END
   54 END IF

   55 IF filename1$ = filename2$ THEN          '! gleiche Namen?
   56  PRINT "Eingabedatei = Ausgabedatei nicht erlaubt!"
   57  END
   58 END IF

      ' prüfe ob Datei vorhanden, nein -> exit

   59 OPEN filename1$ FOR INPUT AS #ein%      '! Öffne Eingabedatei
   60 OPEN filename2$ FOR OUTPUT AS #aus%     '! Öffne Ausgabedatei
   61 PRINT
   62 PRINT "Die Datei: ";filename1$;" wird bearbeitet"
```

```
   63 WHILE NOT (EOF(ein%))                  '! Datei sequentiell
lesen
   64  LINE INPUT #ein%, linie$              '! lese Zeile
   65  CALL skiprem(1,linie$,remflg%)        '! prüfe auf Kommentar
   66  lang% = LEN(linie$)                   '! ermittle Zeilenlänge
   67  IF (lang% > 0) AND (NOT remflg%) THEN '! nur Anweisungen
   68   CALL scanner(linie$)                 '! analysiere Satz
   69  END IF
   70  IF (INSTR(UCASE$(kommando$),"/L") > 0) THEN '! Level
ausgeben
   71   PRINT #aus%, USING "## "; lev1%;
   72   lev1% = lev2%
   73  END IF
   74  PRINT #aus%, linie$                   '! Satz speichern
   75 WEND

   76 CLOSE                                  '! Dateien schließen
   77 PRINT
   78 PRINT "Die Datei: ";filename2$;" wurde erzeugt"
   79 END

      '############################################################
      '#                     Hilfsroutinen                        #
      '############################################################

   80 fehler:
      '------------------------------------------------------------
      '! Fehlerbehandlung in XFORM
      '------------------------------------------------------------

   81 IF ERR = 53 THEN
   82  PRINT "Die Datei ";filename1$;" existiert nicht"
   83 ELSE
   84  PRINT "Fehler : ";ERR;" unbekannt"
   85  PRINT "Erradr : ";ERADR
   86  PRINT "Programmabbruch"
   87 END IF
   88 END                                  '! MSDOS Exit
   89 RETURN


   90 SUB getfile(ptr%,text$,result$)
      '!-----------------------------------------------------------
      '! separiere Filename aus Eingabetext (text$)
      '! ptr% -> Anfang Filename, result$ = Filename
      '!-----------------------------------------------------------
   91 LOCAL tmp%, i%, zchn$

   92 CALL skipblank (ptr%,text$)             '! entferne Blanks
   93 tmp% = ptr%                             '! Anfang Filename
   94 FOR i% = ptr% to LEN(text$)             '! suche Ende Filename
   95  zchn$ = MID$(text$,i%,1)
   96  IF (zchn$ = " ") or (zchn$ = "/") THEN
   97   result$ = MID$(text$,ptr%,i%-ptr%)   '! Filename
extrahieren
```

```
   98   ptr% = i%
   99   EXIT SUB
  100  END IF
  101  tmp% = i%
  102 NEXT i%

  103 IF (tmp% = ptr%) THEN
  104   PRINT "Fehler: kein Fileseparator"    '! kein Endeseparator
  105   END                                   '! Exit
  106 ELSE
  107   result$ = MID$(text$,ptr%,tmp%-ptr%+1) '! Filename
extrahieren
  108   ptr% = tmp%
  109 END IF

  110 END SUB

  111 SUB scanner(text$)
      '!-------------------------------------------------------
      '! Scan Quellcode zeilenweise und rücke die Zeilen gegeben-
      '! enfalls ein. Die Zeile wird in text$ zurückgegeben.
      '!-------------------------------------------------------

  112 LOCAL token%, tokentxt$, ptr%, found%, ptr1%, zchn$, tmp%
  113 SHARED lang%, spalte%, indent%, lev1%, lev2%

  114  ptr% = 1                                  '! Start mit 1.
Zeichen
  115  CALL skipblank (ptr%,text$)               '! Blanks
entfernen
  116  ptr1% = ptr%
  117  CALL gettoken(ptr%,text$,tokentxt$,token%) '! suche token
  118  IF token% THEN
  119   tokentxt$ = UCASE$(tokentxt$)             '! Großbuchstaben

      '! bearbeite 1. Token und berechne Einrückung

  120   IF (tokentxt$ = "FOR") THEN
  121    text$ = SPACE$(spalte%) + MID$(text$,ptr1%,lang%)
  122    spalte% = spalte% + indent%
  123    lev2% = lev2% + 1

  124   ELSEIF (tokentxt$ = "DO") THEN
  125    text$ = SPACE$(spalte%) + MID$(text$,ptr1%,lang%)
  126    spalte% = spalte% + indent%
  127    lev2% = lev2% + 1

  128   ELSEIF (tokentxt$ = "WHILE") THEN
  129    text$ = SPACE$(spalte%) + MID$(text$,ptr1%,lang%)
  130    spalte% = spalte% + indent%
  131    lev2% = lev2% + 1

  132   ELSEIF (tokentxt$ = "SUB") THEN
  133    text$ = SPACE$(spalte%) + MID$(text$,ptr1%,lang%)
  134    spalte% = spalte% + indent%
```

```
  135    lev2% = lev2% + 1

  136   ELSEIF (tokentxt$ = "IF") THEN

  137    text$ = SPACE$(spalte%) + MID$(text$,ptr1%,lang%)
  138    token% = %true
  139    DO WHILE (token%)   '! prüfe, ob letztes Token = THEN
  140     CALL gettoken(ptr%,text$,tokentxt$,token%) '! suche token
  141     IF token% THEN oldtoken$ = UCASE$(tokentxt$)
  142    LOOP

  143    IF oldtoken$ = "THEN" THEN
  144     spalte% = spalte% + indent%
  145     lev2% = lev2% + 1
  146    END IF
  147   ELSEIF (tokentxt$ = "ELSE") THEN
  148    tmp% = spalte% - indent%
  149    IF tmp% < 0 THEN tmp% = 0
  150    text$ = SPACE$(tmp%) + MID$(text$,ptr1%,lang%)
  151    IF lev2% > 0 THEN lev1% = lev2% - 1

  152   ELSEIF (tokentxt$ = "ELSEIF") THEN
  153    tmp% = spalte% - indent%
  154    IF lev2% > 0 THEN lev1% = lev2% - 1

  155    IF tmp% < 0 THEN tmp% = 0
  156    text$ = SPACE$(tmp%) + MID$(text$,ptr1%,lang%)

  157   ELSEIF (tokentxt$ = "END") THEN             '! END IF/ END
SUB
  158    CALL gettoken(ptr%,text$,tokentxt$,token%) '! suche token
  159    IF token% THEN
  160     tokentxt$ = UCASE$(tokentxt$)             '!
Großbuchstaben
  161     IF (tokentxt$ = "IF") or (tokentxt$ = "SUB") THEN
  162      spalte% = spalte% - indent%
  163      IF spalte% < 0 THEN spalte% = 0
  164      IF lev2% > 0 THEN lev2% = lev2% - 1
  165      lev1% = lev2%
  166     END IF
  167    END IF
  168    text$ = SPACE$(spalte%) + MID$(text$,ptr1%,lang%)

  169   ELSEIF (tokentxt$ = "NEXT") THEN
  170    spalte% = spalte% - indent%
  171    IF spalte% < 0 THEN spalte% = 0
  172    text$ = SPACE$(spalte%) + MID$(text$,ptr1%,lang%)
  173    IF lev2% > 0 THEN lev2% = lev2% - 1
  174    lev1% = lev2%

  175   ELSEIF (tokentxt$ = "LOOP") THEN
  176    spalte% = spalte% - indent%
  177    IF spalte% < 0 THEN spalte% = 0
  178    text$ = SPACE$(spalte%) + MID$(text$,ptr1%,lang%)
  179    IF lev2% > 0 THEN lev2% = lev2% - 1
```

```
  180    lev1% = lev2%

  181   ELSEIF (tokentxt$ = "WEND") THEN
  182    spalte% = spalte% - indent%
  183    IF spalte% < 0 THEN spalte% = 0
  184    text$ = SPACE$(spalte%) + MID$(text$,ptr1%,lang%)

  185    IF lev2% > 0 THEN lev2% = lev2% - 1
  186    lev1% = lev2%

  187   ELSE
  188    text$ = SPACE$(spalte%) + MID$(text$,ptr1%,lang%)
  189   END IF
  190  END IF

  191 END SUB

  192 SUB gettoken(ptr%,text$,token$,found%)

      '!-------------------------------------------------------
      '! durchsuche Zeile auf Variablennamen, found% = TRUE falls
      '! Variable gefunden. Der Name wird dann in token
zurückgegeben.
      '!-------------------------------------------------------

  193 LOCAL first%, remflg%, search%            '! lokale Variablen
  194 SHARED lang%                              '! globale
Variablen

  195 found% = %false                           '! init Flag nicht
gefunde
       n
  196 search% = %true                           '! init Flag
  197 lang% = LEN (text$)

  198 WHILE search% AND ptr% <= lang%           '! suche
Variablenanfang
  199  CALL skipblank(ptr%,text$)               '! skip führende
Blanks
  200  IF ptr% >= lang% THEN EXIT SUB           '! Zeilenende ->
Exit

  201  CALL skiprem(ptr%,text$,remflg%)         '! Kommentar?
  202  IF remflg% THEN EXIT SUB                 '! ja -> Zeile
fertig

  203  IF MID$(text$,ptr%,1) = CHR$(34) THEN    '! String "...."?
  204   CALL skipstring(ptr%,text$)             '! skip string
  205   IF ptr% >= lang% THEN EXIT SUB          '! Ende -> Exit
  206  END IF

  207  zchn$ = UCASE$(MID$(text$,ptr%,1))       '! Zeichen
separieren
  208  IF ((zchn$ < "A") OR (zchn$ > "Z")) _    '! Suche Anfang
Name
```

```
  209   AND (INSTR("$&%",zchn$) = 0) THEN       '!    "
  210    INCR ptr%                               '! nein -> next
  211  ELSE
  212   search% = %false                         '! ja -> exit
  213  END IF
  214 WEND

  215 IF search% THEN                           '! gefunden?
  216  EXIT SUB                                 '! nein, Zeilenende
-> Exi
      t
  217 END IF

      '! ### Anfang eines Tokens gefunden ####

  218 first% = ptr%                             '! merke Anfang
token
  219 search% = %true                           '! init Flag

  220 WHILE search% AND ptr% <= lang%           '! suche
Variablenende
  221  zchn$ = UCASE$(MID$(text$,ptr%,1))       '! Zeichen
separieren
  222  IF (zchn$ < "A") OR (zchn$ > "Z") THEN   '! Ende Name?
  223    IF (zchn$ < "0") OR (zchn$ > "9") THEN
  224      IF INSTR("$%&!#",zchn$) = 0 THEN
  225       search% = %false                     '! gefunden
  226       DECR ptr%                            '! korrigiere ptr%
wg. Feh
      ler
  227      END IF                                '! da EXIT WHILE
nicht geh
      t !
  228    END IF
  229  END IF
  230  INCR ptr%                                '! nein -> next

  231 WEND

  232 found% = %true                            '! gefunden !
  233 token$ = MID$(text$,first%,(ptr%-first%)) '! get token

  234 END SUB

  235 SUB skipblank(ptr%,text$)
      '-------------------------------------------------------
      '! zähle führende Blanks in einer Zeichenkette
      '! text$ = Zeichenkette, zeiger% = Zeiger in Kette
      '-------------------------------------------------------
  236 SHARED lang%

  237 WHILE (ptr% =< lang%) and (MID$(text$,ptr%,1) = " ")
  238  INCR ptr%
  239 WEND
  240 END SUB
```

```
  241 SUB skipstring(ptr%,text$)

      '!---------------------------------------------------------
      '! Es wird geprüft, ob ptr% auf ein " im Text zeigt. In
diesem
      '! Falls liegt ein String "...." vor, dessen Ende (") gesucht
      '! wird. ptr% zeigt nach dem Ablauf auf das Zeichen hinter ".
      '! text$ = Zeichenkette mit Anweisung
      '!---------------------------------------------------------
  242 SHARED lang%

  243 IF MID$(text$,ptr%,1) <> CHR$(34) THEN
  244  EXIT SUB                              '! kein String "..."
  245 END IF

  246 DO                                     '! suche Ende String
  247  INCR ptr%                             '! next char
  248 LOOP UNTIL (MID$(text$,ptr%,1) = CHR$(34)) OR (ptr% >= lang%)

  249 END SUB

  250 SUB skiprem (ptr%,text$,flag%)

      '!---------------------------------------------------------
      '! prüfe auf Kommentare, text$ = String mit Anweisung
      '! ptr% = Zeiger in Text, flag% = true -> Kommentar gefunden
      '!---------------------------------------------------------

  251  CALL skipblank(ptr%,text$)        '! führende blanks
entfernen
  252  IF INSTR(text$,"REM") = ptr% THEN '! scan Anfang = REM oder
'
  253   flag% = %true                    '! Kommentar
  254  ELSE
  255   IF MID$(text$,ptr%,1) = "'" THEN
  256    flag% = %true                    '! Kommentar
  257   ELSE
  258    flag% = %false                   '! kein Kommentar
  259   END IF
  260  END IF
  261 END SUB
      '***** Programm Ende ******
```

Listing 2.7: XFORM.BAS

3 Werkzeuge zur Behandlung von Textdateien

Bei der Bearbeitung von Textdateien und Programmen treten immer wieder die gleichen Aufgabenstellungen auf. Oft ist der Inhalt der Datei nach bestimmten Kriterien auszuwerten oder bestimmte Dateien sollen kombiniert werden. Die Benutzer von Unix erhalten hier durch das Betriebssystem reichhaltige Unterstützung. Aber unter MS-DOS ist in der Beziehung (abgesehen von Programmen wie MORE, COMP, SORT) nichts zu finden. Aus diesem Grund werden nachfolgend einige kleine Hilfsprogramme erstellt, die hier Unterstützung bieten. Das Programm LISTER.BAS paßt ebenfalls in dieses Kapitel, da es sich zur Bearbeitung von Texten eignet. Nun werden noch einige ergänzende Utilities vorgestellt.

WC: Auswertung von Textdateien

Bei der Erstellung von Textdateien werden manchmal Informationen über den Dateiinhalt benötigt. Auch ein Quellprogramm ist in der Regel nichts anderes als eine spezielle Textdatei. Der Dateiinhalt läßt sich nun im Hinblick auf folgende Fragestellungen analysieren:

- Wieviele Textzeichen sind in der Datei vorhanden?
- Wieviele Wörter enthält der Text?
- Wieviele Zeilen enthält der Text?

Insbesondere der letzte Punkt tritt häufig auf, wenn nach den »Lines of Code« eines Programms gefragt wird. Das Programm ist recht kurz und liefert diese Information.

Die Benutzerschnittstelle ermöglicht verschiedene Varianten der Aktivierung. Nach dem Programmstart mit dem Befehl:

```
WC
```

erscheint folgende Meldung:

```
 W C                              (c) Born Version 1.0

 File : .....
```

auf dem Bildschirm. Mit »File :« wird ein gültiger MS-DOS-Dateiname abgefragt, der auch Laufwerks- und Pfadangaben enthalten darf. Alternativ läßt sich der Dateiname direkt beim Programmstart in der Kommandozeile mit angegeben. Dann gilt die Syntax:

```
WC  Filename
```

Hier kann dann auf die Anzeige des Programmkopfes verzichtet werden.

Existiert die angegebene Datei nicht, bricht das Programm mit folgender Meldung:

File nicht vorhanden

Andernfalls erscheint der Hinweis:

```
Datei <filename> wird bearbeitet
```

»filename« steht für den eingegebenen Dateinamen. Nach der Analyse erscheinen die Ergebnisse in folgnder Form:

```
 Zeichen    : 3698
 Wörter     : 382
 Zeilen     : 117
 Leerzeilen : 14
```

auf dem Bildschirm. Neben der Zeilenzahl wird auch der Anteil an Leerzeilen in diesem Text mit ausgegeben. Obige Daten geben zum Beispiel die Auswertung der Datei WC.BAS wieder.

Über den Aufruf:

```
WC /?
```

steht die Online-Hilfe zur Verfügung. Das Programm gibt dann einen Textbildschirm aus.

```
W C                         (c) Born Version 1.0

Aufruf: WC <Filename>
```

Das Programm analysiert die Textdatei und gibt die Anzahl `der Zeichen, Wörter und Zeilen aus.`

Die Implementierung

Die Implementierung ist recht einfach. Das Programm besteht zunächst aus einem Hauptprogramm zur Variableninitialisierung und zur Dateibehandlung. In einer WHILE-Schleife wird die Textdatei satzweise mit LINE INPUT bearbeitet. Am Programmende wird die Datei geschlossen und die Ergebnisse werden am Bildschirm angezeigt. Das Strichdiagramm in Bild 3.1 zeigt den Ablauf.

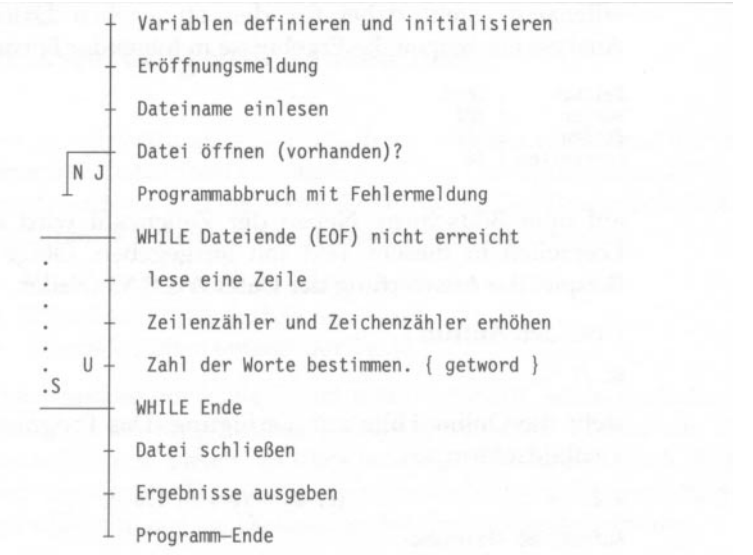


Bild 3.1: Programmablauf in WC.BAS

Nach der Ausgabe der Kopfmeldung wird der Name der auszugebenden Datei in die Stringvariable *filename$* eingelesen. Hierbei sind neben Laufwerksangaben auch Pfadbezeichnungen vor dem Dateinamen erlaubt. Auf die restlichen Einzelheiten wird nicht eingegangen, da diese im Prinzip bereits bei der Entwicklung des Programmes LISTER.BAS diskutiert wurden.

getword

Neben dem Modul »fehler« wird nur ein Unterprogramm benutzt. Aufgabe dieses Moduls ist es, die Zahl der Wörter innerhalb einer Zeile zu ermitteln un in der Variablen *worte%* zu addieren. Als Wörter gelten hier alle Zeichenketten, die mehr als ein Zeichen enthalten. Wörter, werden durch Leerzeichen, oder durch das Zeilenende, getrennt. Leerzeilen werden ignoriert.

Die Einzelheiten sind nachfolgendem Listing zu entnehmen.

Erweiterungsvorschläge

Die Separation der Wörter beschränkt sich darauf, Zeichenketten mit mehr als einem Zeichen zu ermitteln. Für die Auswertung von Textdateien interessieren vielleicht aber nur wirkliche Wörter. Hier kann der Algorithmus so verfeinert werden, daß er nur noch Wörter der deutschen Sprache, einschließlich Umlaute, erkennt. Weiterhin können Leerzeilen bei der Berechnung der Zeilenzahl ignoriert werden. Als letzte Verbesserung besteht die Möglichkeit, daß Programm mit einer Endlosschleife zu versehen, die nach der Auswertung einer Datei wieder an den Programmanfang geht und einen weiteren Dateinamen abfragt. Nur bei

Leereingaben wird dann das Programm verlassen. Diese Erweiterung kann auch bei vielen anderen in diesem Buch vorgestellten Modulen benutzt werden.

```
X R E F   /Z=50                              (c) Born Version 1.0
Datei : wc.bas      Datum : 05-16-1992          Seite :  1

 Zeile     Anweisung

      ' ********************************************************
      ' File       : WC.BAS
      ' Vers.      : 1.0
      ' Last Edit  : 7. 5.92
      ' Autor      : G. Born
      ' Files      : INPUT
      ' Progr. Spr.: PowerBASIC
      ' Betr. Sys. : MS-DOS 2.1 - 5.0
      ' Funktion: Das Programm untersucht einen Textfile und
      '           gibt die Zahl der Zeichen, Wörter und Zeilen
      '           aus. Ein Wort wird nur gezählt, falls es mehr
      '           als einen Buchstaben enthält. Trennzeichen
      '           zwischen Wörtern sind Leerzeichen.
      ' ********************************************************
      ' screens und Variable definieren
    1 zeile& = 0                             '! Zeilen im Text
    2 worte& = 0                             '! Worte im Text
    3 char& = 0                              '! Zeichen im Text
    4 leerz& = 0                             '! Leerzeilen
    5 ein% = 3                               '! I/O Kanal
    6 linie$ = ""                            '! Textpuffer
    7 lang% = 0                              '! Zeilenlänge
    8 ptr% = 0

      '############################################################
      '#                      Hauptprogramm                       #
      '############################################################

    9 ON ERROR GOTO fehler

   10 kommando$ = COMMAND$                    '! Parameter ?
   11 IF LEN (kommando$) = 0 THEN             '! User Mode ?
   12  CLS                                    '! clear Screen
   13  PRINT "W C                           (c) Born Version 1.0"
   14  PRINT
   15  INPUT  "File : ",filename$
   16  PRINT
   17 ELSE
   18  ptr% = INSTR (kommando$,"/?")          '! Option /?
   19  IF ptr% <> 0 THEN                      '! Hilfsbildschirm
   20   PRINT "W C                          (c) Born Version 1.0"
   21   PRINT
   22   PRINT "Aufruf: WC <Filename>"
   23   PRINT
   24   PRINT "Das Programm analysiert die Textdatei und gibt die
Zahl"
```

```
   25   PRINT "der Zeichen, Wörter und Zeilen aus."
   26   PRINT
   27   SYSTEM
   28  END IF
   29 ||||      '! Kommando Mode
   30  filename$ = kommando$               '! nur Filename
   31 END IF

      ' prüfe ob Datei vorhanden, nein -> exit

   32 OPEN filename$ FOR INPUT AS #ein%    '! File öffnen

   33 PRINT "Datei ";filename$;" wird bearbeitet"

   34 WHILE NOT (EOF(ein%))                 '! Datei sequentiell
lesen
   35  LINE INPUT #ein%, linie$             '! lese Zeile
   36  lang% = LEN (linie$)                 '! Zeichen / Zeile
   37  char& = char& + lang%                '! count chars
   38  INCR zeile&                          '! count lines
   39  IF lang% <= 0 THEN
   40   INCR leerz&                         '! Zahl der Leerzeilen
   41  ELSE
   42   GOSUB getword                       '! count words
   43  END IF
   44 WEND

   45 CLOSE #ein%                          '! close datei

   46 PRINT
   47 PRINT "Zeichen    :", char&
   48 PRINT "Wörter     :", worte&
   49 PRINT "Zeilen     :", zeile&
   50 PRINT "Leerzeilen :", leerz&
   51 PRINT
   52 END

      '############################################################
      '#                    Hilfsroutinen                         #
      '############################################################

   53 fehler:
      '------------------------------------------------------------
      '! Fehlerbehandlung in WC
      '------------------------------------------------------------

   54 IF ERR = 53 THEN
   55  PRINT "Die Datei ";filename$;" existiert nicht"
   56 ELSE
   57  PRINT "Fehler : ";ERR;" unbekannt"
   58  PRINT "Programmabbruch"
   59 END IF
   60 END                                  '! MSDOS Exit
   61 RETURN
```

```
62 getword:
   '-------------------------------------------------------
   '! zähle die Zahl der Worte in der Zeile
   '-------------------------------------------------------

63  zahl& = 0                              '! init char count
64  FOR i%= 1 TO lang%                     '! scan line
65   zchn$ = MID$(linie$,i%,1)             '! separate char
66   IF zchn$ <> " " THEN
67    INCR zahl&                           '! zähle Buchstaben
68   ELSE
69    IF zahl& > 1 THEN INCR worte&        '! count words
70    zahl& = 0                            '! clear char count
71   END IF
72  NEXT i%
73  IF zahl& > 1 THEN INCR worte&          '! letztes Wort zählen
74 RETURN
   ' ###### Programm Ende #########
```

Listing 3.1: WC.BAS

CUT: Ein Filter für Textdateien

Bei Textdateien, insbesondere mit Tabellen, sollen öfters bestimmte Spalten entfernt werden. Bei umfangreichen Texten ist nur eine automatische Bearbeitung sinnvoll. Ein Editor hilft hier nicht weiter, da er zeilenorientiert arbeitet. Also bleibt nur die manuelle Bearbeitung, Zeichen für Zeichen. Es stellt sich die Frage, ob diese Aufgabe nicht wesentlich einfacher durch ein Programm zu erledigen ist.

Ausgehend von der Unix-Utility CUT entstand ein solches Programm, welches einige dieser Funktionen übernimmt.

Die Spezifikation

Beginnen wir zuerst mit der Spezifikation der Anforderungen. Das Modul soll zwar Textdateien zeilenorientiert bearbeiten. Aufgabe ist es jedoch, spaltenweise bestimmte Zeichen auszufiltern und den Restsatz in eine Ausgabedatei zu übertragen. Diese Aufgabenstellung tritt zum Beispiel bei Tabelle 3.1 auf.

Artikel	Nr.	Preis (Euro)	Rabatt
4711	5,70	30 - 50	
.	.	.	.
1743	3,80	22	

Tabelle 3.1: Beispiel einer Tabelle mit feldorientiertem Aufbau

Diese enthält neben der Artikelnummer und dem Preis auch die Rabattspanne die jeweils eingeräumt werden kann. Wird nun diese Tabelle gedruckt, ist es nicht immer erwünscht, wenn die Rabattsätze auftauchen.

Nach der herkömmlichen Methode werden also jeweils zwei Listen erstellt und gepflegt. Dies ist zeitraubend und fehlerträchtig. Schaltet man nun das Programm CUT vor die Ausgabe, läßt sich die Rabattspalte leicht entfernen. Damit erscheint nur noch der gewünschte Text.

Ein ähnliches Problem tritt auf, wenn aus einer Adreßdatei mit Name, Anschrift und Telefonnummer die Adresse entfernt werden soll. Die Einträge sind gemäß nachfolgender Darstellung spaltenweise geordnet, wobei die Einträge durch Doppelpunkte getrennt sind.

```
   Name                 Adresse                 Telefonnummer

Müller, Udo : 6000 Frankfurt Wiesenstraße 18 : 069 - 12345
Dr. Post, Josef : 8000 München 1 Schillerstraße 23 : 089 1007
Heinz Peter : 6230 Frankfurt (M) 80 Königstr. 1 : 069 - 33456
  .                .
```

Hier besteht nun die Schwierigkeit, daß keine genaue Spalte angegeben werden kann, ab der der Text zu entfernen ist. Da aber die einzelnen Felder durch ein Trennzeichen (hier ein »:«) markiert sind, kann ein anderes Verfahren angewandt werden. Es wird das Trennzeichen sowie die Nummer des zu entfernenden Feldes spezifiziert. Mit der Feldnummer 2 lässt dich die Adresse entfernen, wodurch nur noch Name und Telefonnummer in der Ausgabedatei stehen. Damit sit bereits obige Forderung erfüllt. Sind mehrere Felder zu entfernen, ist CUT mehrmals aufzurufen.

Nun wollen wir mit der Beschreibung der Ein-/Ausgabemeldungen beginnen. Hier gibt es wieder die zwei Alternativen zur Übergabe der Parameter: den Kommandomodus und den interaktiven Eingabemodus. Wird nun der Name CUT ohne weitere Parameter eingegeben, dann verzweigt das Programm in den interaktiven Eingabemodus. Auf dem Bildschirm erscheint folgende Meldung:

```
C U T                              (c) Born Version 1.0
Optionen : [ /F=xx    Feld Nr.          /D=:  Delimiter       ]
           [ /S=xx    Skip n Lines      /P=xx Process n Lines ]
           [ /=Cx1-x2 Column x1 bis x2  /T    Trace ON        ]

Eingabedatei :
Ausgabedatei :
Optionen     :
```

Bild 3.2: Kopfmeldung von CUT im Eingabemodus

Mit »Eingabedatei« wird ein gültiger MS-DOS Dateiname, der auch Laufwerks- und Pfadangaben enthalten darf, abgefragt. Exisitiert die angegebene Datei nicht, bricht das Programm mit einer Fehlermeldung ab (Tabelle 3.2):

```
Die Datei <name> existiert nicht
```

Mit <*name*> wird hier die eingegebene Datei bezeichnet. Andernfalls erscheint die folgende Abfrage am Bildschirm:

```
Ausgabedatei : .....
```

Existiert die Datei bereits, erfolgt eine Warnung:

```
Die Ausgabedatei existiert bereits, überschreiben (J/N) ?
```

die explizit bestätigt werden muß. Nur durch die Eingabe »J« oder »j« wird die Bearbeitung fortgesetzt und die Abfrage der Optionen erscheint:

```
Optionen :
```

Es muß nun entweder die Feld- (/F) oder die Column-Option (/C) korrekt eingegeben werden. Ist dies nicht der Fall, erscheinen die Fehlermeldungen (Tabelle 3.2):

```
Die Optionen fehlen oder sind falsch
```

Dann ist das Programm erneut zu starten.

Für den Einsatz in Batchdateien besteht die Möglichkeit der Parameterübergabe innerhalb der Kommandozeile. In diesem Fall besitzt der Aufruf folgende Struktur:

```
CUT <Eingabedatei> <Ausgabedatei> <Optionen>
```

Wichtig ist hierbei, daß alle drei Eingabeparameter innerhalb der Kommandozeile auftauchen. Im Kommandomodus erfolgt ebenfalls eine Warnung, falls die Ausgabedatei bereits existiert.

Fehlt die Eingabe in einem Datei- oder Optionsfeld, wird unabhängig vom Kommando- oder Interaktiv-Modus der Programmablauf mit einer Fehlermeldung (Tabelle 3.2) abgebrochen.

Über die Option:

```
CUT /?
```

läßt sich der Bildschirm mit der Online-Hilfe abrufen.

Die Optionen

Nun wollen wir uns noch etwas genauer mit den Optionseingaben beschäftigen.

Die Feld-Option /F

Die Feld-Option erlaubt es, ein komplettes Feld zu entfernen. Daher muß die laufende Nummer des Feldes sowie das Trennzeichen (Delimiter) zwischen den Feldern angegeben werden. Es sind z.B. folgende Eingaben möglich:

```
/F=2  /D=:
/D=;  /F=2
```

In diesem Fall wird das zweite Feld entfernt, wobei die Felder durch einen Doppelpunkt oder ein Semikolon getrennt sind. Wird das Trennzeichen (z.B. /D=:) bei Verwendung der /F-Option nicht angegeben, erfolgt ein Programmabbruch mit einer Fehlermeldung:

```
Option /D fehlt
```

Obiges Format ist bei der Eingabe einzuhalten. Bei der /F-Option ist in jedem Fall auch das Trennzeichen mit /D zu definieren. Falls die Column-Option nicht verwendet wird, ist die Feld-Option zwingend vorgeschrieben.

Die Column-Option /C

Befinden sich die zu entfernenden Zeichen an festen Positionen innerhalb einer Zeile, läßt sich die Column-Option verwenden.

```
/C=x1-x2
```

Mit *x1* wird dabei die Anfangsspalte angegeben, ab der die Zeichen zu entfernen sind. Die Variable *x2* spezifiziert die Spalte, bis zu der die Zeichen noch entfernt werden. Nachfolgend sind einige gültige Eingaben aufgeführt:

```
/C=3-10       entferne Zeichen ab Spalte 3 bis Spalte 10
/C=5                entferne alle Zeichen ab Spalte 5
/C=-12      entferne alle Zeichen bis Spalte 12
```

Es werden alle Zeichen, einschließlich der jeweils spezifizierten Spalten entfernt.

Ist der Wert der Anfangsspalte kleiner als der Wert der Endspalte, bricht das Programm mit einer Fehlermeldung ab:

```
Fehler bei /C=    Ende < Anfang
```

Falls die Feld-Option nicht verwendet wird, ist die Column-Option zwingend vorgeschrieben. Sonst bricht das Programm mit einer Fehlermeldung ab.

Die Skip Line-Option /S

Oft ist es so, daß die zu bearbeitende Tabelle in einen Text integriert ist. Dieser Text darf natürlich nicht bearbeitet werden. Hierfür dient die Option *Skip Line*. Durch die Eingabe:

```
/S=20
```

werden die ersten 20 Zeilen innerhalb der Textdatei überlesen und unbearbeitet in die Ausgabedatei gespeichert. Diese Option ist nicht zwingend vorgeschrieben. Falls sie fehlt, beginnt die Bearbeitung der Datei ab der ersten Zeile.

Die Process-Line Option /P

Diese Option ergänzt die Skip Line-Option. Der Text vor der Tabelle läßt sich durch die /S-Option überlesen. Schließt sich an die Tabelle weiterer Text an, darf dieser natürlich nicht bearbeitet werden. Mit der Eingabe:

```
/P=15
```

wird dem Programm CUT signalisiert, daß nur 15 Zeilen innerhalb der Datei zu bearbeiten sind. Fehlt die /S-Option, sind dies die ersten 15 Zeilen der Datei. Sonst werden xx-Zeilen überlesen (S=xx) und anschließend sind die folgenden 15 Zeilen zu bearbeiten. Dann wird der Rest der Datei unmodifiziert ausgegeben. Die /P-Option ist nicht zwingend vorgeschrieben.

Mit dieser Technik läßt sich eine Tabelle innerhalb eines Textes selektiv bearbeiten. Bei mehreren Tabellen im Text ist CUT mehrmals aufzurufen, wobei pro Durchlauf jeweils eine Tabelle bearbeitet wird.

Die Trace-Option /T

Wird der Schalter /T gesetzt, erscheinen alle Zeilen, die in die Ausgabedatei abgespeichert werden, auch auf dem Bildschirm. Diese Option ist standardmäßig ausgeschaltet, um die Anzeige zu unterdrücken. Waren die Eingaben korrekt, meldet sich das Programm mit:

```
CUT Start
```

Falls der Trace-Mode eingeschaltet ist, wird anschließend der Inhalt der bearbeiteten Datei satzweise auf dem Bildschirm ausgegeben. Die erste Zeile enthält den Originalsatz, während die zweite Zeile die korrigierte Fassung zeigt. Leere Sätze werden übergangen. Findet sich kein Separator oder kein entsprechendes Feld, erscheinen auch bei ausgeschalteten Trace-Mode die Meldungen:

```
kein Separator
```

```
Feld Nr. .. fehlt
```

Der Satz wird dann unbearbeitet abgespeichert. Am Dateiende erfolgt eine Abschlußmeldung:

```
CUT Ende
```

Damit liegt das Ergebnis in der spezifizierten Ausgabedatei vor.

Die Fehlermeldungen

Da das Programm eine Reihe von Fehlermeldungen ausgibt, werden diese nachfolgend zusammengefaßt.

```
-----------------------------------------------------
| Meldung: Der Name der <....datei> fehlt
|
| Ursache: Es fehlt der Name einer Ein- oder Aus-
|          gabedatei. Dies kann insbesondere beim
|          Kommandomodus auftreten.
-----------------------------------------------------
| Meldung: Die Optionen fehlen oder sind falsch
|
| Ursache: Die Eingabe ist zwingend erforderlich.
|          Das Eingabeformat muß eingehalten wer-
```

	den.
Meldung:	Die Datei <name> existiert nicht
Ursache:	Im eingegebenen Verzeichnis wurde die Eingabedatei nicht gefunden.
Meldung:	Fehler : <Nummer> unbekannt
Ursache:	Laufzeitfehler PowerBASIC. Die Nummer gibt die Fehlerart an (PowerBASIC Dokumentation).
Meldung:	Fehler : Options /C oder /F fehlen
Ursache:	Es wurde keine gültige Column- oder Feldoption gefunden.
Meldung:	Option /D fehlt
Ursache:	Es wurde eine Feld Option eingegeben und keine gültige /D Option gefunden.
Meldung:	kein Separator
Ursache:	Die eingelesene Zeile enthält kein Separatorzeichen, der Satz wird komplette in die Ausgabe übernommen.
Meldung:	Feld <nummer> nicht gefunden
Ursache:	In der /F Option wurde ein Wert angegeben, der größer ist als die Zahl der Felder in der Eingabezeile.
Meldung:	Fehler : kein Fileseparator
Ursache:	In der Kommandozeile fehlen die Blanks zwischen den Dateinamen, oder es wurden keine Optionen eingegeben.
Meldung:	Fehler bei /C= Ende < Anfang
Ursache:	Bei der /C Option ist der Wert der Endespalte kleiner als der Wert der Anfangsspalte.

Tabelle 3.2: Fehlermeldungen des Programmes CUT

Die obigen Fehler führen zu einem Programmabbruch, während nach Warnungen (z.B. kein Separator) der Ablauf fortgesetzt wird.

Die Implementierung von CUT

Die Implementierung ist recht einfach. Das Programm besteht wieder aus mehreren Modulen, die im folgenden Hierarchiediagramm (Bild 3.3) aufgeführt sind.

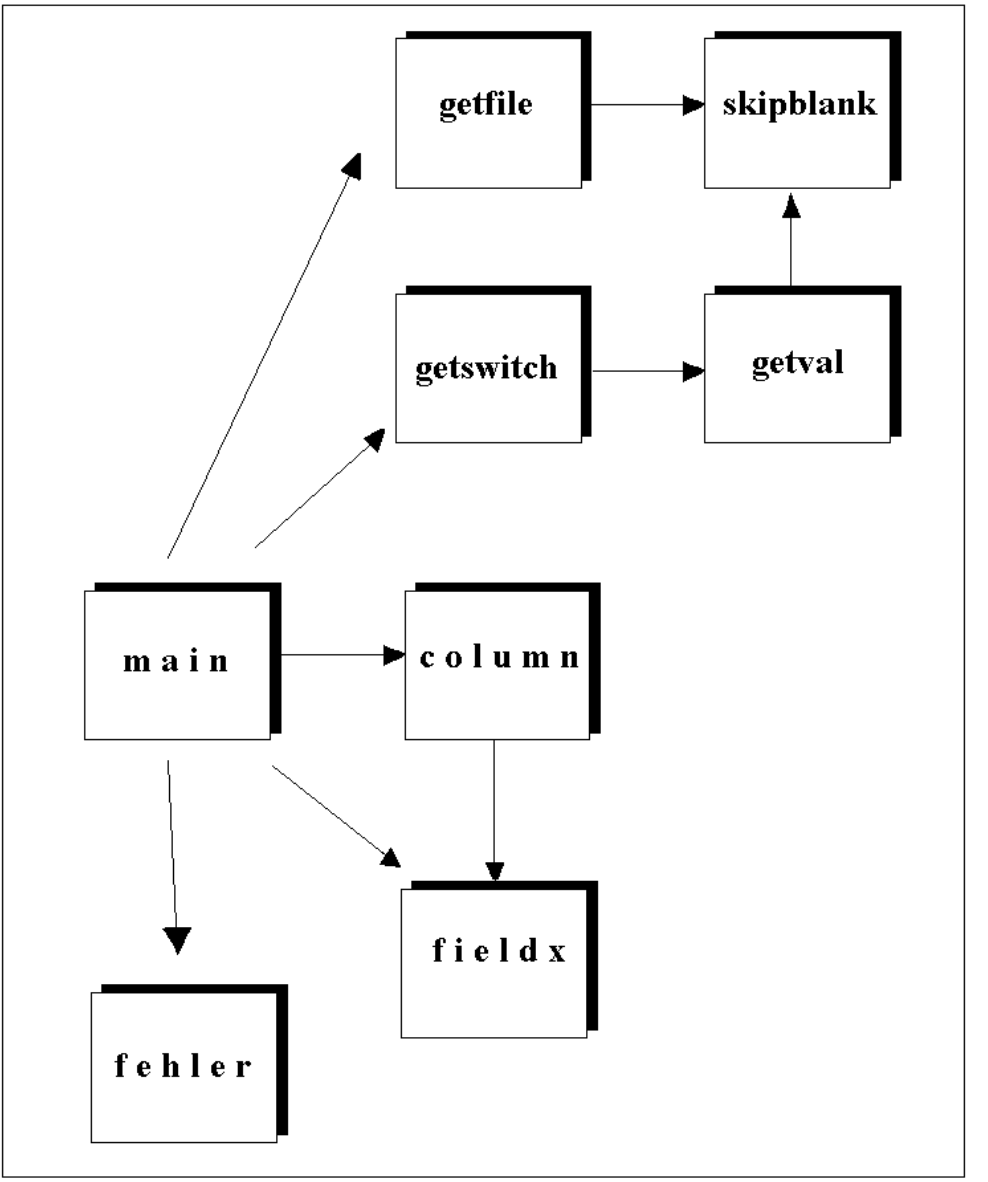


Bild 3.3: Modulhierarchie in CUT.BAS

Das Hauptprogramm dient zur Variableninitialisierung und zur Dateibehandlung. In einer WHILE-Schleife wird die Textdatei satzweise mit

LINE INPUT gelesen und über die Unterprogramme *fieldx* und *column* bearbeitet. Am Programmende wird die Datei geschlossen und die Endemeldung erscheint auf dem Bildschirm. Das Strichdiagramm in Bild 3.4 zeigt den Ablauf.

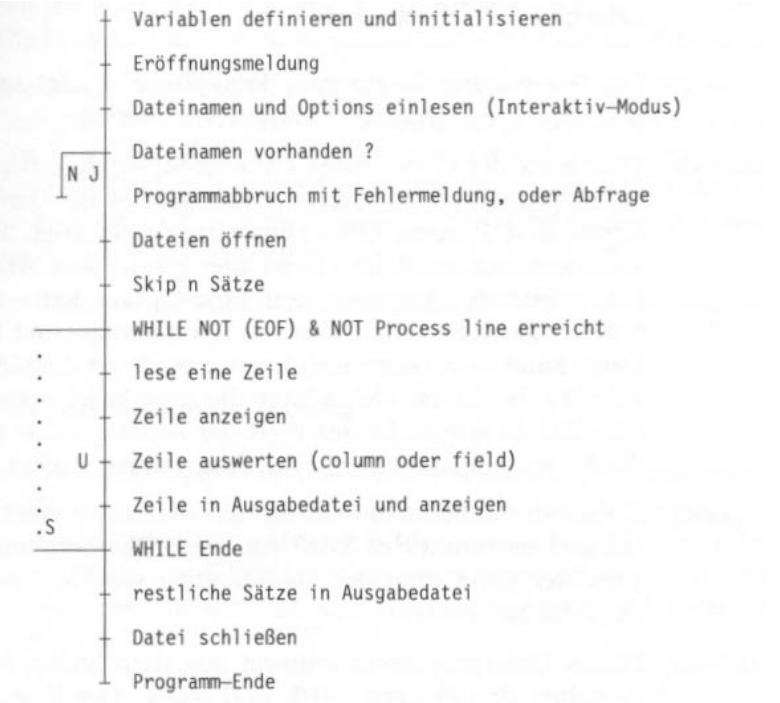


Bild 3.4: Programmablauf in CUT.BAS

Im Hauptmodul werden einige Variable definiert, deren Bedeutung kurz erläutert wird.

```
col1%           Zeiger auf das erste zu entfernende Zeichen
col2%           Zeiger auf das letzte zu entfernende Zeichen
ptr%            Hilfzeiger auf aktuelles Zeichen

feld%           Nummer des zu entfernenden Feldes
colopt%         true = Column-Option   false = Feld-Option
inlinie$        String mit Originalzeile
outlinie$       String mit Ausgabezeile
trace%          Schalter = true -> Trace-Mode
skip%           Zahl der zu überlesenden Zeilen
work&           Zahl der zu bearbeitenden Zeilen
zeile&          bearbeitete Zeile
```

Einzelheiten innerhalb des Hauptmoduls sind dem Listing zu entnehmen.

Die CUT-Hilfsmodule

Die Bearbeitung bestimmter Funktionen erfolgt wie üblich in Hilfsmodulen, die nachfolgend besprochen werden.

getswitch

Hier wird der eingegebene Options-String decodiert. Im ersten Schritt ist zu prüfen, ob die Trace-Option gesetzt ist. Dann werden die Skip Lines- und Process Lines-Optionen ausgewertet. Ist die Feld-Option selektiert, muß auch der /D-Schalter gesetzt sein. Andernfalls erfolgt ein Fehlerabbruch. Alternativ zur Feld-Option kann auch der Column-Schalter aktiviert sein. Dann werden Anfangs- und Endspalte ermittelt. Dabei kann es vorkommen, daß nur ein Wert definiert wird (/C=1 oder /C=-20). In diesen Fällen sind die jeweils fehlenden Parameter auf 0 oder 255 zu setzen. Ist der Wert der Anfangsspalte größer als der Wert der Endespalte, dann bricht das Programm mit einer Fehlermeldung ab.

getval

Dieses Modul liest einen String aus vorzeichenbehafteten Dezimalziffern ein und bestimmt den Wert (Integer). Das Ergebnis wird dann in der Variablen *tmp%* abgelegt. Die Funktion wird aus dem Unterprogramm *getswitch* aufgerufen.

column

Dieses Unterprogramm entfernt aus dem String *inlinie$* alle Zeichen zwischen den Zeigern *col1%* und *col2%*. Der Reststring wird dann in *outlinie$* abgelegt.

fieldx

Bei Anwahl der Feld-Option sucht dieses Modul das entsprechende Feld. Dabei werden die Variablen *col1%* und *col2%* auf die Anfangs- und Endeseparatoren des betreffenden Feldes positioniert. Fehlt der Separator im untersuchten Satz oder wird kein Feld mit der entsprechenden Nummer gefunden, erfolgt eine Warnung an den Bediener. Der Satz wird anschließend unmodifiziert in die Ausgangsdatei kopiert. Wurde ein gültiges Feld gefunden, erfolgt ein Aufruf des Unterprogrammes *column*, da dieses ja bereits die Funktion der Zeichenentfernung erfüllt.

getfile

Dieses Modul separiert einen Dateinamen aus dem übergebenen Optionsstring. Der Name muß durch ein Leerzeichen vom nachfolgenden Text getrennt sein. Der Parameter *ptr%* spezifiziert, ab welcher Position im Optionsstring die Separierung erfolgen soll. Nach dem Aufruf zeigt *ptr%* auf das Leerzeichen hinter dem Dateinamen.

fehler

Mit *fehler* werden Laufzeitfehler des PowerBASIC-Systems abgefangen. Die Fehlernummer wird angezeigt, danach bricht das Programm ab.

skipblank

Das Modul entfernt führende Leerzeichen aus einem übergebenen Text (*string$*). Der Zeiger (*ptr%*) bestimmt beim Aufruf, ab welcher Position der Text bearbeitet wird. Nach dem Aufruf zeigt *ptr%* auf das erste Zeichen, das kein Leerzeichen ist. Der Text wurde aber nicht verändert. Im rufenden Programm lassen sich dann die Leerzeichen leicht durch die Funktion *MID$()* entfernen.

Die Einzelheiten sind dem nachfolgenden Listing zu entnehmen.

Erweiterungsvorschläge

Mit der Feld-Option läßt sich nur jeweils ein Feld pro Programmlauf bearbeiten. Eine Erweiterung hinsichtlich der Möglichkeit zur gleichzeitigen Entfernung mehrerer Felder ist denkbar. Weiterhin besteht die Möglichkeit, den Filteralgorithmus so abzuwandeln, daß ein bestimmtes Zeichen gesucht wird, ab dem dann die eingelesenen Zeilen bearbeitet werden.

```
X R E F   /Z=50                              (c) Born Version 1.0
Datei : cut.bas      Datum : 05-17-1992         Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : CUT.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 6. 5.92
      '! Autor      : G. Born
      '! Files      : INPUT, OUTPUT
      '! Progr. Spr.: POWERBASIC
      '! Betr. Sys. : DOS 2.1 - 5.0
      '! Funktion: Das Programm liest eine Textdatei ein und
filtert
      '!           bestimmte Textspalten aus. Das Ergebnis wird in
      '!           einer zweiten Datei abgelegt. Es sind folgende
      '!           Optionen möglich:
      '!
      '!           /F=x1    entferne Feld x1
      '!
      '!           In diesem Fall muß ein Feldseparator mit
      '!
      '!           /D=x     x = Trennzeichen
      '!
      '!           eingegeben werden. Weiterhin können einzelne
Spalten
      '!           mit
      '!
      '!           /C=x1-x2  x1 = Anfangsspalte  x2 = Endspalte
      '!
      '!           entfernt werden. Es ist jeweils nur die /F oder
      '!           /C Option zulässig. Die Option:
      '!
```

```
      '!           /S=xx
      '!
      '!           erlaubt am Anfang der Textdatei n Zeilen zu
überlesen,
      '!           während bei der
      '!
      '!           /P=xx
      '!
      '!           Option nur n Zeilen bearbeitet werden.
      '!
      '! Aufruf:   CUT                                  '! Interaktiv
Modus
      '!           CUT <datei1> <datei2> <Optionen>    '! Kommando
Modus
      '************************************************************
      '! Variable definieren
    1 %true = &HFFFF: %false = 0            '! Konstante
    2 trace% = %false                       '! No Trace Mode
    3 ein% = 1 : aus% = 2                   '! Kanäle für I/O
    4 col1% = 1                             '! Anfangsspalte
    5 col2% = 255                           '! Endspalte
    6 feld% = 0                             '! Feldnr. für Option
/F
    7 colopt% = %true                       '! Column Option
einstellen
    8 options$ = ""                         '! Optionen
    9 inlinie$ = ""                         '! Puffer Lesedatei
   10 outlinie$ = ""                        '! Puffer Schreibdatei
   11 skip% = 0                             '! Zeilen überlesen
   12 work& = 100000                        '! Zeilen zu bearbeiten
   13 zeile& = 0                            '! bearbeitete Zeilen
   14 ptr% = 0: hilf% = 0                   '! Hilfszeiger
   15 tmp$ = ""                             '! Hilfsstring
   16 sep$ = ""                             '! Delimiter Feld

   17 ON ERROR GOTO fehler

      '############################################################
      '#                      Hauptprogramm                       #
      '############################################################

   18 kommando$ = COMMAND$                  '! Parameter ?
   19 IF LEN (kommando$) = 0 THEN           '! Interaktiv Mode ?
   20  CLS                                  '! ja -> Clear Screen
      '! #####   Kopf ausgeben  ######
   21  PRINT "C U T                         (c) Born Version 1.0"
   22  PRINT
   23  PRINT "Optionen : [ /F=xx    Feld Nummer       /D=x
Delimiter
         ]"
   24  PRINT "           [ /S=xx    Skip n Lines      /P=xx Prozess
n Line
        s ]"
   25  PRINT "           [ /C=x1-x2 Column x1 bis x2  /T    Trace
ON
```

```
          ]"
   26  PRINT
   27  INPUT "Eingabedatei : ",infilename$  '! lese Dateiname
Eingabe
   28  INPUT "Ausgabedatei : ",outfilename$ '! lese Dateiname
Ausgabe
   29  INPUT "Optionen     : ",options$     '! lese Optionen
   30  PRINT
   31 ELSE                                  '! Kommando Mode
   32  ptr% = INSTR (kommando$,"/?")        '! Option /?
   33  IF ptr% <> 0 THEN                    '! Hilfsbildschirm
   34   PRINT "C U T                         (c) Born Version 1.0"
   35   PRINT
   36   PRINT "Optionen : [ /F=xx    Feld Nummer       /D=x
Delimiter
          ]"
   37   PRINT "           [ /S=xx    Skip n Lines      /P=xx
Prozess n Lin
       es ]"
   38   PRINT "           [ /C=x1-x2 Column x1 bis x2  /T    Trace
ON
          ]"
   39   PRINT
   40   PRINT "Das Programm filtert bestimmte Textspalten aus
einer"
   41   PRINT "Eingabedatei heraus. Optionen:"
   42   PRINT
   43   PRINT "/F=xx   filtert das durch Delimiter getrennte Feld
Nr. xx heraus"
   44   PRINT "/D=x    gibt den Delimiter (z.B. , oder ;) für die
Felder a
       n"
   45   PRINT "/C=x1-x2 schneidet die Spalten x1 bis x2 aus der
Datei"
   46   PRINT "/S=xx   überliest n Zeilen, bevor der Filter
aktiviert wird
       "
   47   PRINT "/P=xx   filtert nur n Zeilen und terminiert dann"
   48   PRINT "/T      gibt die bearbeitete Zeile und das Ergebnis
am Bild
       schirm aus"
   49   PRINT
   50   SYSTEM
   51  END IF
      '!
      '! getfile separiert den Dateinamen aus der Kommandozeile
      '! Falls ein Name fehlt, würden die Optionen in die jeweilige
      '! Variable gespeichert. Dies ist abgefangen, da Optionen mit
      '! /.. beginnen. Dann wird ein Leerstring zurückgegeben
      '!
   52  ptr% = 1                                  '! Parameter holen
   53  CALL getfile(ptr%, kommando$,infilename$) '! Name
Eingabedatei
   54  INCR ptr%                                 '! Anfang next
token
```

```
   55  CALL getfile(ptr%, kommando$,outfilename$)'! Name
Ausgabedatei
   56  hilf% = INSTR(kommando$,"/")          '! suche Optionen
   57  IF hilf% >= ptr% THEN                 '! gefunden ?
   58   options$ = MID$(kommando$,hilf%)     '! Reststring mit
Optionen
   59  END IF
   60 END IF

   61 IF infilename$ = "" THEN               '! Leereingabe ?
   62  PRINT "Der Name der Eingabedatei fehlt"
   63  END
   64 END IF

   65 IF outfilename$ = "" THEN               '! Leereingabe ?
   66  PRINT "Der Name der Ausgabedatei fehlt"
   67  END
   68 END IF


   69 IF (LEN(options$) = 0) OR _             '! Optionen prüfen ?
   70  (INSTR(options$,"/") = 0) THEN
   71  PRINT "Die Options fehlen oder sind falsch"
   72  END
   73 END IF

   74 OPEN infilename$ FOR INPUT AS #ein%  '! öffne Eingabedatei
      '! Ausgabedatei vorhanden -> prüfe über OPEN inputdatei

   75 ON ERROR GOTO ok
   76 OPEN outfilename$ FOR INPUT AS #aus% '! existiert
Ausgabedatei%
   77 ON ERROR GOTO fehler
   78 CLOSE #aus%                          '! nein -> Close

   79 INPUT "Ausgabedatei existiert bereits, überschreiben (J/N) ?
", tmp$
   80 PRINT
   81 IF UCASE$(tmp$) <> "J" THEN END      '! stopp -> sonst Datei
weg

   82 ok:
   83 OPEN outfilename$ FOR OUTPUT AS #aus% '! Ausgabedatei open

   84 options$ = UCASE$(options$)           '! in Großbuchstaben
   85 GOSUB getswitch                       '! lese Optionen

   86 PRINT
   87 PRINT "CUT Start "
   88 PRINT

      '!
      '! überlese führende Zeilen falls skip line gesetzt
      '!
```

```
   89 WHILE (skip% > 0) AND (NOT (EOF(ein%)))
   90  DECR skip%                          '! skip% - 1
   91  LINE INPUT #ein%, inlinie$          '! lese eine Zeile
   92  PRINT #aus%, inlinie$               '! in Ausgabedatei
   93  IF trace% THEN                      '! Trace Mode ?
   94   PRINT inlinie$                     '! Anzeige Original
   95   PRINT inlinie$                     '! Anzeige Kopie
   96   PRINT
   97  END IF
   98 WEND

      '!
      '! bearbeite n Zeilen
      '!

   99 WHILE (zeile& < work&) AND (NOT (EOF(ein%)))
  100  LINE INPUT #ein%, inlinie$          '! lese eine Zeile
  101  lang% = LEN (inlinie$)              '! Merke Zeilenlänge
  102  IF colopt% THEN
  103   CALL column (inlinie$, outlinie$)  '! auswerten Zeile
  104  ELSE
  105   CALL fieldx(inlinie$, outlinie$)   '!    "
  106  END IF
  107  PRINT #aus%, outlinie$              '! Zeile in
Ausgabedatei
  108  IF trace% THEN                      '! Trace Mode ?
  109   PRINT inlinie$                     '! display Original
  110   PRINT outlinie$                    '! display Zeile
  111   PRINT
  112  END IF
  113  INCR zeile&                         '! Zeile + 1
  114 WEND

      '!
      '! restliche Zeilen umkopieren
      '!

  115 WHILE NOT (EOF (ein%))
  116  LINE INPUT #ein%, inlinie$          '! lese eine Zeile
  117  PRINT #aus%, inlinie$               '! in Ausgabedatei
  118  IF trace% THEN                      '! Trace Mode ?
  119   PRINT inlinie$                     '! ja -> Anzeige
  120   PRINT inlinie$                     '! 2 x
  121   PRINT
  122  END IF
  123 WEND

  124 CLOSE
  125 PRINT
  126 PRINT "CUT Ende"
  127 END

      '############################################################
      '#                    Hilfsroutinen                         #
      '############################################################
```

```
  128 fehler:
      '-------------------------------------------------------
      '! Fehlerbehandlung in CUT
      '-------------------------------------------------------

  129 IF ERR = 53 THEN
  130  PRINT "Die Datei ";infilename$;" existiert nicht"
  131 ELSE
  132  PRINT "Fehler : ";ERR;" unbekannt"
  133  PRINT "Programmabbruch"
  134 END IF
  135 END                                  '! MSDOS Exit
  136 RETURN

  137 getswitch:
      '-------------------------------------------------------
      '! decodiere eingegebene Optionen
      '! /F = field   /D = separator    /C = column
      '! /S = skip    /P = process      /T = trace
      '-------------------------------------------------------

  138  options$ = UCASE$(options$)

  139  IF INSTR(options$,"/T") > 0 THEN
  140   trace% = %true                        '! Trace Mode ein
  141  END IF

  142  ptr% = INSTR(options$,"/S=")          '! Skip Lines Option ?
  143  IF ptr% > 0 THEN
  144   CALL getval(options$,ptr%+3,skip%)   '! lese Zahl
  145  END IF

  146  ptr% = INSTR(options$,"/P=")          '! Process Lines
Option ?
  147  IF ptr% > 0 THEN
  148   CALL getval(options$,ptr%+3,tmpx%)   '! lese Zahl
  149   work& = tmpx%
  150  END IF
  151  ptr% = INSTR(options$,"/F=")          '! Field Option ?
  152  IF ptr% > 0 THEN
  153   CALL getval(options$,ptr%+3,feld%)   '! lese Zahl
  154   IF feld% <= 0 THEN                   '! falsche Nummer
  155    PRINT "Feld Nummer ";feld%;" nicht zulässig"
  156    END
  157   END IF
  158   ptr% = INSTR(options$,"/D=")          '! check Delimiter
  159   IF ptr% = 0 THEN
  160    PRINT "Option /D fehlt"
  161    END
  162   END IF
  163   sep$ = MID$(options$,ptr%+3,1)       '! get Separator
  164   colopt% = %false                     '! select Field Option
  165   RETURN
  166  END IF
```

```
167  ptr% = INSTR(options$,"/C=")          '! Column Option ?
168  IF ptr% > 0 THEN
    '! es sind folgende Eingaben erlaubt:
    '! /C10-20   /C-20      /C10-
169   INCR ptr%,3                          '! auf 1. Zeichen
170   CALL getval (options$,ptr%,tmpx%)    '! lese 1. Wert
171    IF tmpx% < 0 THEN                   '! negativer Wert ?
172     col2% = -tmpx%                     '! Endespalte
173    ELSE
174     col1% = tmpx%                      '! Startspalte
175     IF MID$(options$,ptr%,1) = "-" THEN
176      CALL getval (options$,ptr%,tmpx%) '! lese 2. Wert
177      tmpx% = - tmpx%                    '! Vorzeichenwechsel
178      IF tmpx% > col1% THEN              '! Endewert?
179       col2% = tmpx%                     '! ja
180      END IF
181     END IF
182   END IF
183  ELSE
184   PRINT "Fehler: Options /C oder /F fehlen"
185   END
186  END IF
187 RETURN

188 SUB getval(text$,ptr%,result%)
    '-------------------------------------------------------
    '! decodiere Eingabewert als Dezimalzahl
    '-------------------------------------------------------
189 LOCAL tmp%, zchn$, leng%, sign%

190 sign% = 1                             '! Vorzeichen +
191 tmp% = 0                              '! Hilfsvariable

192 leng% = LEN(text$)

193 CALL skipblank(ptr%,text$)            '! überlese Leerzeichen
194 zchn$ = MID$(text$,ptr%,1)            '! separ. Zeichen
195 IF zchn$ = "-" THEN                   '! Vorzeichen ?
196  sign% = -1                           '! Vorzeichen  -
197  INCR ptr%                            '! auf 1. Ziffer
198 END IF

199 zchn$ = MID$(text$,ptr%,1)            '! separ. Zeichen

200 WHILE (zchn$ >= "0") AND (zchn$ <= "9")_
201  AND (ptr% <= leng%)                  '! n Ziffern
202  tmp% = tmp% * 10 + VAL(zchn$)        '! Ziffer holen
203  INCR ptr%                            '! nächstes Zeichen
204  zchn$ = MID$(text$,ptr%,1)           '! lese Zeichen
205 WEND
206 result% = tmp% * sign%                '! Vorzeichen

207 END SUB

208 SUB column(quelle$,ziel$)
```

```
      '!----------------------------------------------------------
      '! entferne alle Zeichen von Spalte (start) bis Spalte (ende)
      '!----------------------------------------------------------

  209 SHARED col1%, col2%

  210  IF col1% <= 0 THEN col1% = 1        '! negativer Wert
  211  ziel$ = LEFT$(quelle$,col1%-1)     '! Anfang merken
  212  ziel$ = ziel$ + MID$(quelle$,col2%+1)'! Rest anhängen

  213 END SUB

  214 SUB fieldx (quelle$,ziel$)
      '!----------------------------------------------------------
      '! entferne Feld Nr. n
      '!----------------------------------------------------------
  215 SHARED feld%, col1%, col2%, sep$
  216 LOCAL ptr%, zahl%, anf%, ende%, lang%

  217 lang% = LEN(quelle$)
  218 ptr% = INSTR(quelle$,sep$)            '! suche separator

  219 IF ptr% = 0 THEN                      '! nicht gefunden
  220  ziel$ = quelle$                      '! alles übernehmen
  221  PRINT "kein Separator"
  222  EXIT SUB
  223 END IF

      '! beachte, daß ein Delimiter entfernt wird !!!

  224 anf% = 1: ende% = ptr%                '! init Feldgrenzen
  225 zahl% = 1                             '! Feldzähler

  226 WHILE (zahl% < feld%) AND (ptr% > 0)
  227  anf% = ende% + 1                     '! hinter Delimiter
  228  ptr% = INSTR(ptr%+1,quelle$,sep$)    '! suche Feldende
      '!
      '!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      '! Achtung TB Fehler: Der Befehl INSTR() funktioniert
      '! nicht immer, falls der Suchbegriff nicht vorkommt.
      '! Deshalb muß ptr% = 0 und ptr% < Stringlänge abge-
      '! fragt werden. (In PowerBASIC übernehmen?!)
      '!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      '!
  229  IF (ptr% = 0) OR (ptr% >= lang%) THEN '! letztes Feld ?
  230   ende% = LEN(quelle$)                '! ja -> auf
Stringlänge
  231   DECR anf%                           '! auf Delimiter
  232   ptr% = 0                            '! Ende Schleife !
  233  ELSE
  234   ende% = ptr%                        '! nein -> auf
Delimiter
  235  END IF
  236  INCR zahl%                           '! next Feld
```

```
  237 WEND

  238 IF (zahl% < feld%) THEN                '! Feld gefunden?
  239  ziel$ = quelle$                       '! Feld nicht gefunden
  240  PRINT "Feld Nr. ";feld%;" nicht gefunden"
  241  EXIT SUB
  242 END IF

  243 IF zahl% = feld% THEN                  '! Feld gefunden
  244  col1% = anf%                          '! Feldanfang
  245  col2% = ende%                         '! Feldende
  246  CALL column(quelle$,ziel$)            '! cut field with
column
  247 END IF

  248 END SUB


  249 SUB getfile(ptr%,text$,result$)
      '!-------------------------------------------------------
      '! separiere Filename aus Eingabetext (text$)
      '! ptr% -> Anfang Filename, result$ = Filename
      '!-------------------------------------------------------
  250 LOCAL tmp%

  251 CALL skipblank (ptr%,text$)             '! entferne Blanks
  252 tmp% = INSTR(ptr%,text$," ")            '! suche Separator
  253 IF tmp% = 0 THEN
  254  PRINT "Fehler: kein Fileseparator"     '! kein Endeseparator
  255  END                                    '! Exit
  256 END IF
  257 IF MID$(text$,ptr%,1) = "/" THEN        '! Optionen eingegeben
?
  258  result$ = ""                           '! Leerstring
  259 ELSE
  260  result$ = MID$(text$,ptr%,tmp%-ptr%)   '! Filename
  261  ptr% = tmp%                            '! korrigiere ptr%
  262 END IF

  263 END SUB

  264 SUB skipblank(ptr%,text$)
      '!-------------------------------------------------------
      '! entferne führende Leerzeichen aus text$
      '!-------------------------------------------------------

  265 LOCAL lang%, zchn$

  266 lang% = LEN (text$)                     '! Stringlänge
  267 zchn$ = MID$(text$,ptr%,1)              '! separiere Zeichen

  268 WHILE (zchn$ = " ") AND (ptr% <= lang%) '! Zeichen <> blank
  269  INCR ptr%
  270  zchn$ = MID$(text$,ptr%,1)             '! separiere Zeichen
  271 WEND
```

```
272 END SUB

    '***** Programm Ende ******
```

Listing 3.2: CUT.BAS

PASTE: Vereinigung von Textdateien

CUT ermöglicht es, bestimmte Spalten aus einem Textdokument »herauszuschneiden«. Nun fehlt noch eine Funkton zum Zusammenkleben. Hierbei sind die Texte satzweise zu kombinieren. Bei der Bearbeitung von Tabellen tritt diese Aufgabe zum Beispiel häufiger auf: Informationen sind aus zwei Dateien spaltenweise zu kombinieren. Nachfolgend wird ein solcher Fall konstruiert. Es liegt eine Datei mit Namen und Adressen vor. Eine zweite Datei enthält satzweise die zugehörigen Telefonnummern. Hieraus ist nun ein Textdokument zu erstellen, welches die Daten im folgenden Format enthält:

```
Name    Adresse    Telefonnummer
```

Beide Dateien sind satzweise zu lesen, zusammenzufügen und in einer dritten Datei auszugeben. Eine Arbeit, die per Texteditor nur mit erheblichem manuellen Aufwand zu erledigen ist.

Ein ähnliches Problem tritt auf, wenn zum Beispiel aus folgender Tabelle:

```
Name : Adresse : Telefonnummer
```

die Spalten *Adresse* und Te*lefonnummer* zu tauschen sind. Per Editor kaum zu schaffen, aber mit den Modulen CUT und PASTE elegant nach folgendem Schema durchzuführen.

- Zerlege die Urdatei mit CUT in drei Dateien, die jeweils nur noch eine Spalte besitzen.
- Kombiniere die (Spalten-) Dateien mittels des Programms PASTE in beliebiger Reihenfolge.

Es sind sicherlich einige Aufrufe notwendig, aber die Arbeit wird zuverlässig verrichtet. Notfalls läßt sich hierfür eine kleine Batchdatei erstellen, so daß die Aufgabe beliebig oft wiederholbar ist.

Da das Modul PASTE recht klein ist, wird sich hier auf die Diskussion der Ein-/Ausgabemeldungen beschränkt.

Das Programm ist wie gewohnt mit einer interaktiven Bedieneroberfläche und mit einer Kommandoversion ausgestattet. Bei der Eingabe des Wortes PASTE, ohne weitere Parameter, verzweigt das Programm in den interaktiven Modus und meldet sich nach dem Start mit dem Kopftext:

```
P A S T E                    (c) Born Version 1.0
```

```
Optionen : [ /F=xx    Skip n Lines in File 1    ]
           [ /D=xx    Delimiter   /T Trace ON   ]

Eingabedatei 1:
Eingabedatei 2:
Ausgabedatei  :
Optionen      :
```

Bild 3.5: Kopfmeldung des Programmes PASTE

Mit »Ein-/Ausgabedatei« werden gültige MS-DOS Dateinamen, die auch Laufwerks- und Pfadangaben enthalten dürfen, abgefragt. Existiert eine der angegebenen Eingabedateien nicht, bricht das Programm mit folgender Meldung ab:

```
Die Datei <Name 1> oder <Name 2> existiert nicht
```

An Stelle von <*Name 1*> und <*Name 2*> erscheinen die eingegebenen Dateinamen. Die Abfrage nach der Eingabedatei 2 erscheint erst nach Eingabe des vorhergehenden Dateinamens. Das gleiche gilt für die Abfrage der Ausgabedatei. Existiert die Ausgabedatei bereits, erfolgt eine Warnung:

```
Ausgabedatei existiert bereits, überschreiben (J/N) ?
```

die explizit zu quittieren ist. Der Programmablauf wird mit den folgenden Meldungen angezeigt:

PASTE Start

PASTE Ende

Im Trace-Modus erscheinen zusätzlich die bearbeiteten Zeilen auf dem Bildschirm.

Für den Einsatz in Batchdateien besteht die Möglichkeit zur Eingabe der Dateinamen in der Kommandozeile. Der Kommandomodus besitzt folgende Syntax:

```
PASTE <Eingabedatei 1> <Eingabedatei 2> <Ausgabedatei> <Options>
```

Die Dateinamen dürfen Laufwerks- und Pfadbezeichnungen enthalten. Wildcard-Zeichen (*.*) sind allerdings nicht erlaubt. Die einzelnen Parameter sind durch jeweils ein Leerzeichen voneinander zu trennen.

Über den Aufruf:

```
PASTE /?
```

läßt sich die Online-Hilfe aktivieren. Auf dem Bildschirm erscheint dann folgender Text:

```
P A S T E                          (c) Born Version 1.0

Optionen : [ /S=xx    Skip n Lines in File 1    ]
           [ /D=xx    Delimiter   /T Trace ON   ]
```

```
Das Programm liest zwei Eingabedateien und kombiniert
die Texte zeilenweise in eine Ausgabedatei.

Optionen:

/S=xx  überliest n Zeilen in File 1 bis File 2 angehängt wird
/D=xx  definiert den Delimiter, der die Teilsätze trennt
/T     schaltet den Trace-Modus ein
```

Bild 3.6: Online-Hilfe für PASTE

Anschließend endet das Programm wieder.

Die Optionen

Nun sollten noch kurz die Optionen besprochen werden. Anders als bei CUT sind diese bei der Eingabe nicht zwingend vorgeschrieben. Aber sie ermöglichen in einigen Fällen etwas mehr Komfort beim Umgang mit CUT.

DieSkip Line-Option /S

Oft ist es so, daß die zu ergänzende Tabelle in einen Text integriert ist. Dieser Text darf natürlich nicht bearbeitet werden. Hierfür dient die Option *Skip Line*. Durch die Eingabe:

/S=20

werden die ersten 20 Zeilen innerhalb der Textdatei überlesen und unbearbeitet in die Ausgabedatei gespeichert. Diese Option ist nicht zwingend vorgeschrieben. Falls sie fehlt, beginnt die Bearbeitung der Datei ab der ersten Zeile.

DieTrace-Option /T

Wird der Schalter /T gesetzt, erscheinen alle Zeilen, die in die Ausgabedatei abgespeichert werden, auch auf dem Bildschirm. Diese Option ist standardmäßig ausgeschaltet, um die Anzeige zu unterdrücken.

Die Delimiter-Option /D

Mit dieser Option läßt sich ein Trennzeichen definieren, welches zwischen die zwei zu kombinierenden Sätze eingefügt wird. Dies kann ein Feldtrennzeichen (z.B.: , ,) oder einfach ein Leerzeichen sein. Gültige Eingaben sind:

/D=#
/d=:

Mit der Taste Alt lassen sich auch Zeichen eingeben, die nicht auf der Tastatur vorkommen. Fehlt diese Option, werden die Sätze direkt (auch ohne Leerzeichen) zusammengefügt.

Der Text der *Eingabedatei1* wird immer linksbündig gespeichert. Jeder

Satz wird dann durch den zugehörigen Satz der *Eingabedatei2* ergänzt.

Die Fehlermeldungen

Ähnlich dem Programm CUT gibt PASTE einige Fehlermeldungen aus, die nachfolgend zusammengefaßt werden.

Meldung: Der Name der <....datei> fehlt
Ursache: Es fehlt der Name einer Ein- oder Aus- gabedatei. Dies kann insbesondere beim Kommandomodus auftreten.
Meldung: Die Datei < > oder < > existiert nicht
Ursache: Im eingegebenen Verzeichnis wurde die Eingabedatei nicht gefunden.
Meldung: Fehler : <Nummer> unbekannt
Ursache: Laufzeitfehler PowerBASIC. Die Nummer gibt die Fehlerart an (PowerBASIC Dokumentation).
Meldung: Fehler : kein Fileseparator
Ursache: In der Kommandozeile fehlen die Blanks zwischen den Dateinamen, oder es wurden keine Optionen eingegeben.

Tabelle 3.3: Fehlermeldungen des Programmes PASTE

Bei diesen Fehlern bricht das Programm den Ablauf ab.

Die Implementierung

Das Programm besitzt einen relativ einfachen Aufbau, der in Bild 3.7 dargestellt wird.

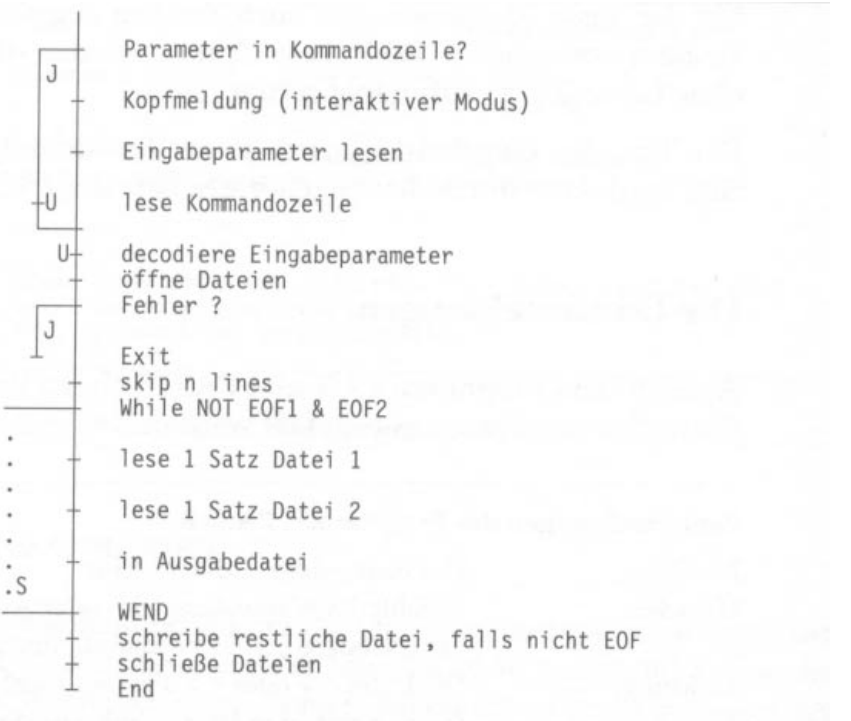


Bild 3.7: Ablauf des Hauptprogrammes PASTE.BAS

Im Hauptmodul werden die Variablen initialisiert. Dann wird geprüft, ob eine Kommandozeile mit Dateinamen vorhanden ist. In diesem Fall erfolgt die Separierung der Eingabedaten mit dem Unterprogramm *getfile*. Andernfalls erscheint die Kopfmeldung und die Daten werden interaktiv abgefragt. Nach einer Überprüfung der Eingaben auf Plausibilität werden die Dateien geöffnet. In einer Schleife werden die eventuell mit der Skip Line-Option spezifizierten Zeilen überlesen. Dann beginnt die Bearbeitung der beiden Eingabedateien. Es wird jeweils ein Satz der Eingabedatei 1 gelesen. Dann ist der entsprechende Satz der Eingabedatei 2 zu lesen. Die Ausgabe erfolgt mit einer PRINT-Anweisung, wobei die gelesenen Texte hintereinander geschrieben werden. Die Variable *delimiter$* kann ein Trennzeichen enthalten (/D= Option). Im Initialisierungsteil wird der Variablen ein Leerstring zugewiesen. Ist das Ende einer der beiden Eingabedateien erreicht, wird die Schleife verlassen. Falls die Längen der Eingabedateien verschieden sind, werden die restlichen Sätze der unbearbeiteten Datei in die Ausgabedatei kopiert. Danach endet das Programm.

Die Hilfsmodule

Das Programm PASTE benutzt einige Hilfsmodule, die kurz vorgestellt werden.

fehler

Mit *fehler* werden Laufzeitfehler des PowerBASIC-Systems abgefangen. Die Fehlernummer wird angezeigt und das Programm bricht ab. Die Fehlerursache ist den PowerBASIC-Handbüchern zu entnehmen.

getswitch

Hier wird der eingegebene Options-String decodiert. Im ersten Schritt ist zu prüfen, ob die Trace-Option gesetzt ist. Dann wird die Skip Lines-Option ausgewertet. Die letzte Prüfung gilt der Delimiter-Option. Ist diese vorhanden, trägt das Modul den gefunden Wert in die Variable *delimiter$* ein.

getval

Dieses Modul liest einen String aus vorzeichenbehafteten Dezimalziffern ein und bestimmt den Wert (Integer). Das Ergebnis wird dann in der Variablen *tmp%* abgelegt. Die Funktion wird aus dem Unterprogramm *getswitch* aufgerufen.

getfile

Dieses Modul separiert einen Dateinamen aus dem übergebenen Optionsstring. Der Name muß durch ein Leerzeichen vom nachfolgenden Text getrennt sein. Der Parameter *ptr%* spezifiziert, ab welcher Position im Optionsstring die Separierung erfolgen soll. Nach dem Aufruf zeigt *ptr%* auf das Leerzeichen hinter dem Dateinamen.

skipblank

Das Modul entfernt führende Leerzeichen aus einem übergebenen Text (*string$*). Der Zeiger (*ptr%*) bestimmt beim Aufruf, ab welcher Position der Text bearbeitet wird. Nach dem Aufruf zeigt *ptr%* auf das erste Zeichen, das kein Leerzeichen ist. Der Text wurde aber nicht verändert. Im rufenden Programm lassen sich dann die Leerzeichen leicht durch die Funktion *MID$()* entfernen.

Einzelheiten sind dem nachfolgenden Listing zu entnehmen.

Erweiterungsvorschläge

Das Programm kann um eine Process-Line-Option erweitert werden, so daß nur *n* Zeilen einer Eingabedatei zu bearbeiten sind. Weiterhin kann eine Option den Ablauf bei ungleichen Längen der Eingabedateien steuern (z.B. Abbruch, falls das Ende der ersten Datei erreicht ist).

```
X R E F   /Z=50                          (c) Born Version 1.0
Datei : paste.bas      Datum : 05-17-1992         Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : PASTE.BAS
```

```
      '! Vers.      : 1.0
      '! Last Edit  : 6. 5.92
      '! Autor      : G. Born
      '! Files      : INPUT, OUTPUT
      '! Progr. Spr.: POWERBASIC
      '! Betr. Sys. : DOS 2.1 - 5.0
      '! Funktion: Das Programm liest zwei Textdateien ein und
      '!           kombiniert diese zeilenweise, so daß aus je-
      '!           weils den zwei Einzelsätzen ein Satz wird.
      '!           Das Ergebnis wird in einer Ausgabedatei abge-
      '!           legt. Es sind folgende Aufrufe möglich:
      '!
      '! Aufruf:   PASTE                              '! Interaktiv
Mode
      '!           PASTE <datei1> <datei2> <datei3> '! Kommando
Mode
      '*************************************************************
      '! Variable definieren
    1 %true = &HFFFF: %false = 0            '! Konstante
    2 ein1% = 1 : ein2% = 2 :  aus% = 3     '! Kanäle für I/O
    3 inlinie1$ = ""                        '! Puffer Lesedatei 1
    4 inlinie2$ = ""                        '! Puffer Lesedatei 2
    5 trace% = %false                       '! Trace Mode aus
    6 skip% = 0                             '! Zeilen überlesen
    7 ptr% = 0: hilf% = 0                   '! Hilfszeiger
    8 delimiter$ = ""                       '! Trennzeichen

      'ON ERROR GOTO fehler

      '############################################################
      '#                      Hauptprogramm                       #
      '############################################################

    9 kommando$ = COMMAND$                  '! Parameter ?
   10 IF LEN (kommando$) = 0 THEN           '! Interaktiv Mode ?
   11  CLS                                  '! ja -> Clear Screen
      '! #####   Kopf ausgeben  ######
   12  PRINT "P A S T E                          (c) Born Version
1.0"
   13  PRINT
   14  PRINT "Optionen : [ /S=xx    Skip n Lines in File 1    ]"
   15  PRINT "           [ /D=xx    Delimiter   /T Trace ON   ]"
   16  PRINT
   17  INPUT "Eingabedatei 1 : ",infilename1$ '! lese Dateiname
Eingabe 1
   18  INPUT "Eingabedatei 2 : ",infilename2$ '! lese Dateiname
Eingabe 2
   19  INPUT "Ausgabedatei   : ",outfilename$ '! lese Dateiname
Ausgabe
   20  INPUT "Optionen       : ",options$     '! lese Optionen
   21  PRINT
   22 ELSE              '! Kommando Mode
   23  ptr% = INSTR (kommando$,"/?")         '! Option /?
   24  IF ptr% <> 0 THEN                     '! Hilfsbildschirm
   25   PRINT "P A S T E                           (c) Born Version
```

```
1.0"
   26   PRINT
   27   PRINT "Optionen : [ /S=xx    Skip n Lines in File 1    ]"
   28   PRINT "           [ /D=xx    Delimiter   /T Trace ON   ]"
   29   PRINT
   30   PRINT "Das Programm liest zwei Eingabedateien und
kombiniert"
   31   PRINT "die Texte zeilenweise in eine Ausgabedatei.
Optionen:"
   32   PRINT
   33   PRINT "/S=xx  überliest n Zeilen in File 1 bis File 2
angehängt wird"
   34   PRINT "/D=xx  definiert den Delimiter, der die Teilsätze
trennt"
   35   PRINT "/T     schaltet den Trace-Modus ein"
   36   PRINT
   37   SYSTEM
   38  END IF
      '!
      '! getfile separiert den Dateinamen aus der Kommandozeile
      '!
   39  kommando$ = UCASE$(kommando$)          '! in Großbuchstaben
   40  ptr% = 1                               '! Parameter holen
   41  CALL getfile(ptr%, kommando$,infilename1$)'! Name
Eingabedatei1
   42  INCR ptr%                                '! Anfang next
token
   43  CALL getfile(ptr%, kommando$,infilename2$)'! Name
Eingabedatei2
   44  INCR ptr%                                '! Anfang next
token
   45  CALL getfile(ptr%, kommando$,outfilename$)'! Name
Ausgabedatei
   46  hilf% = INSTR(kommando$,"/")           '! suche Optionen
   47  IF hilf% >= ptr% THEN                  '! gefunden ?
   48   options$ = MID$(kommando$,hilf%)      '! Reststring mit
Optionen
   49  END IF
   50 END IF

   51 IF infilename1$ = "" THEN              '! Leereingabe ?
   52  PRINT "Der Name der Eingabedatei 1 fehlt"
   53  END
   54 END IF

   55 IF infilename2$ = "" THEN              '! Leereingabe ?
   56  PRINT "Der Name der Eingabedatei 2 fehlt"
   57  END
   58 END IF

   59 IF outfilename$ = "" THEN              '! Leereingabe ?
   60  PRINT "Der Name der Ausgabedatei fehlt"
   61  END
   62 END IF
```

```
   63 OPEN infilename1$ FOR INPUT AS #ein1% '! öffne Eingabedatei
   64 OPEN infilename2$ FOR INPUT AS #ein2% '! öffne Eingabedatei
      '! Ausgabedatei vorhanden -> prüfe über OPEN inputdatei

   65 ON ERROR GOTO ok
   66 OPEN outfilename$ FOR INPUT AS #aus% '! existiert
Ausgabedatei%
   67 ON ERROR GOTO fehler
   68 CLOSE #aus%                          '! nein -> Close

   69 INPUT "Ausgabedatei existiert bereits, überschreiben (J/N) ?
", tmp$
   70 PRINT
   71 IF UCASE$(tmp$) <> "J" THEN END      '! stopp -> sonst Datei
weg

   72 ok:
   73 OPEN outfilename$ FOR OUTPUT AS #aus% '! Ausgabedatei open

   74 options$ = UCASE$(options$)           '! in Großbuchstaben
   75 GOSUB getswitch                       '! lese Optionen

   76 print "opt ";skip%;" ";delimiter$;" ";trace%

   77 PRINT
   78 PRINT "PASTE Start "
   79 PRINT

      '!
      '! überlese führende Zeilen falls skip line gesetzt
      '!

   80 WHILE (skip% > 0) AND (NOT (EOF(ein1%)))
   81  DECR skip%                           '! skip% - 1
   82  LINE INPUT #ein1%, inlinie1$         '! lese eine Zeile
   83  PRINT #aus%, inlinie1$               '! in Ausgabedatei
   84  IF trace% THEN                       '! Trace Mode ?
   85   PRINT inlinie1$                     '! Anzeige Original
   86  END IF
   87 WEND

      '!
      '! bearbeite n Zeilen
      '!
   88  LINE INPUT #ein1%, inlinie1$         '! lese eine Zeile
   89  LINE INPUT #ein2%, inlinie2$         '! lese eine Zeile

   90 WHILE (NOT EOF(ein1%)) AND (NOT EOF(ein2%))
   91  PRINT #aus%, inlinie1$; delimiter$;_ '! Zeile in
Ausgabedatei
   92               inlinie2$               '!  "
   93  IF trace% THEN                       '! Trace Mode ?
   94   PRINT inlinie1$;delimiter$;_        '! Display Zeile
   95         inlinie2$                     '!  "
```

```
 96  END IF
 97  LINE INPUT #ein1%, inlinie1$         '! lese eine Zeile
 98  LINE INPUT #ein2%, inlinie2$         '! lese eine Zeile
 99 WEND

    '!
    '! restliche Zeilen umkopieren bei unterschiedl. Dateilängen
    '!

100 WHILE NOT (EOF (ein1%))               '! Datei 1 fertig?
101  PRINT #aus%, inlinie1$               '! in Ausgabedatei
102  IF trace% THEN                       '! Trace Mode ?
103   PRINT inlinie1$                     '! ja -> Anzeige
104  END IF
105  LINE INPUT #ein1%, inlinie1$         '! lese eine Zeile
106 WEND

107 WHILE NOT (EOF (ein2%))               '! Datei 2 fertig?
108  PRINT #aus%, inlinie2$               '! in Ausgabedatei
109  IF trace% THEN                       '! Trace Mode ?
110   PRINT inlinie2$                     '! ja -> Anzeige
111  END IF
112  LINE INPUT #ein2%, inlinie2$         '! lese eine Zeile
113 WEND

114 CLOSE

115 PRINT
116 PRINT "PASTE Ende"
117 END

    '############################################################
    '#                    Hilfsroutinen                         #
    '############################################################

118 fehler:
    '-----------------------------------------------------------
    '! Fehlerbehandlung in PASTE
    '-----------------------------------------------------------

119 IF ERR = 53 THEN
120  PRINT "Die Datei ";infilename1$;" oder ";infilename2$;_
121        " existiert nicht"
122 ELSE
123  PRINT "Fehler : ";ERR;" unbekannt"
124  PRINT "Programmabbruch"
125 END IF
126 END                                  '! MSDOS Exit
127 RETURN

128 getswitch:
    '-----------------------------------------------------------
    '! decodiere eingegebene Optionen
    '! /S = skip    /D = separator    /T = tracen
    '-----------------------------------------------------------
```

```
129  options$ = UCASE$(options$)

130  IF INSTR(options$,"/T") > 0 THEN
131   trace% = %true                          '! Trace Mode ein
132  END IF

133  ptr% = INSTR(options$,"/S=")             '! Skip Lines Option ?
134  IF ptr% > 0 THEN
135   CALL getval(options$,ptr%+3,skip%)      '! lese Zahl
136  END IF

137  ptr% = INSTR(options$,"/D=")             '! check Delimiter
138  IF ptr% > 0 THEN
139   delimiter$ = MID$(options$,ptr%+3,1) '! get Separator
140  END IF
141 RETURN

142 SUB getval(text$,ptr%,result%)
    '-------------------------------------------------------
    '! decodiere Eingabewert als Dezimalzahl
    '-------------------------------------------------------
143 LOCAL tmp%, zchn$, leng%, sign%

144 sign% = 1                                '! Vorzeichen +
145 tmp% = 0                                 '! Hilfsvariable

146 leng% = LEN(text$)

147 CALL skipblank(ptr%,text$)               '! überlese Leerzeichen
148 zchn$ = MID$(text$,ptr%,1)               '! separ. Zeichen
149 IF zchn$ = "-" THEN                       '! Vorzeichen ?
150  sign% = -1                              '! Vorzeichen  -
151  INCR ptr%                               '! auf 1. Ziffer
152 END IF

153 zchn$ = MID$(text$,ptr%,1)               '! separ. Zeichen

154 WHILE (zchn$ >= "0") AND (zchn$ <= "9")_
155  AND (ptr% <= leng%)                     '! n Ziffern
156  tmp% = tmp% * 10 + VAL(zchn$)           '! Ziffer holen
157  INCR ptr%                               '! nächstes Zeichen
158  zchn$ = MID$(text$,ptr%,1)              '! lese Zeichen
159 WEND
160 result% = tmp% * sign%                   '! Vorzeichen

161 END SUB

162 SUB getfile(ptr%,text$,result$)
    '!-------------------------------------------------------
    '! separiere Filename aus Eingabetext (text$)
    '! ptr% -> Anfang Filename, result$ = Filename
    '!-------------------------------------------------------
163 LOCAL tmp%
```

```
  164 CALL skipblank (ptr%,text$)             '! entferne Blanks
  165 tmp% = INSTR(ptr%,text$," ")            '! suche Separator
  166 IF tmp% = 0 THEN
  167  PRINT "Fehler: kein Fileseparator"     '! kein Endeseparator
  168  END                                    '! Exit
  169 END IF
  170 IF MID$(text$,ptr%,1) = "/" THEN        '! Optionen eingegeben
?
  171  result$ = ""                           '! Leerstring
  172 ELSE
  173  result$ = MID$(text$,ptr%,tmp%-ptr%)   '! Filename
  174  ptr% = tmp%                            '! korrigiere ptr%
  175 END IF

  176 END SUB

  177 SUB skipblank(ptr%,text$)
      '!-------------------------------------------------------
      '! entferne führende Leerzeichen aus text$
      '!-------------------------------------------------------

  178 LOCAL lang%, zchn$

  179 lang% = LEN (text$)                     '! Stringlänge
  180 zchn$ = MID$(text$,ptr%,1)              '! separiere Zeichen

  181 WHILE (zchn$ = " ") AND (ptr% <= lang%) '! Zeichen <> blank
  182  INCR ptr%
  183  zchn$ = MID$(text$,ptr%,1)             '! separiere Zeichen
  184 WEND

  185 END SUB

      '***** Programm Ende ******
```

Listing 3.3: PASTE.BAS

SHOW: Bildschirmanzeige von Textdateien

Bei der Entwicklung von Software oder bei der Erstellung von Texten fallen mit der Zeit eine Reihe Dateien an. Oft stellt sich dann die Frage nach dem Inhalt der Dateien. Nicht immer steht ein Drucker zur Ausgabe der Listings zur Verfügung, oder dieser Weg ist nicht erwünscht. Also wird die MS-DOS-Funktion TYPE zur Anzeige der Texte am Bildschirm benutzt. Sind mehrere längere Dateien zu bearbeiten, muß sich der Anwender in Geduld üben, da ja immer der komplette Inhalt ausgegeben wird. Oft möchte man jedoch nur einige Zeilen des Dateianfangs sehen, um eine bestimmte Datei zu identifizieren. Sind die Programme (wie in diesem Buch) mit einem Kommentarkopf versehen, kann die interne Programmfunktion oft an Hand dieser Informationen bestimmt werden. TYPE oder COPY bieten keine Möglichkeit zur Ausgabe dieser Kommentarköpfe, vielmehr wird der komplette Text ausgegeben, womit

meist der Dateianfang nicht mehr auf dem Bildschirm steht.

Das Betriebssystem Unix bietet ein entsprechendes Hilfsprogramm, welches die Ausgabe des Textanfangs oder des -endes erlaubt. Ein ähnliches Werkzeug soll nun auch für die MS-DOS-Welt entwickelt werden, das folgende Funktionen bietet:

- Bildschirmausgabe des Dateianfangs, wobei die Zeilenzahl vorgegeben werden kann.
- Anzeige des Textendes mit wählbarer Zeilenzahl.
- Ausgabe der kompletten Textdatei auf dem Bildschirm mit zuschaltbarer Zeilennumerierung.
- Seitenweise Ausgabe auf dem Bildschirm analog der DOS-MORE-Funktion.

Der erste Punkt erfüllt obige Forderung, nur den Kommentarkopf auszugeben. Bei der Erstellung von Texten ist Punkt 2 oft hilfreich. Hier läßt sich schnell überprüfen, ob das Dateiende textlich und inhaltlich zum Anfang der folgenden Datei paßt. Die letzten Punkte fallen etwas aus der Reihe. TYPE erlaubt ja bereits die Ausgabe kompletter Dateien. Was aber fehlt ist die Anzeige mit einer vorangestellten Zeilennummer. Gerade dies wird bei den Programmen CUT oder PASTE benötigt. Ist bei CUT eine Tabelle innerhalb eines Textes zu bearbeiten, benötigt der Anwender die Parameter für die Optionen »Skip Lines« und »Process Lines«. Das Modul SHOW liefert nun genau diese Informationen. Weiterhin ist es störend, wenn der ausgegeben Text über den Bildschirm rollt. MS-DOS bietet zwar die Möglichkeit, über den Filter MORE eine seitenweise Ausgabe zu erzwingen. Aber warum sollte ein solche Funktion nicht direkt im eigentlichen Programm implementiert werden?

Nach diesen Vorüberlegungen fehlt noch die genaue Spezifikation der Bedieneroberfläche. Mit der Eingabe:

```
SHOW
```

wird das Programm im Interaktiv-Modus gestartet. Es erscheint folgende Meldung (Bild 3.8):

```
SHOW                          (c) Born Version 1.0

Optionen: [/L=+xx Anzeige n Zeilen am Dateianfang   ]
          [/L=-xx Anzeige n Zeilen am Dateiende     ]
          [/N     Zeilennumerierung   /M  More Ein  ]

Eingabedatei :
Optionen     :
```

Bild 3.8: Kopfmeldung des Programmes SHOW

Der Name der Eingabedatei darf sowohl Laufwerks- als auch Pfadangaben enthalten. Es sind also alle gültigen MS-DOS-Dateinamen erlaubt, lediglich Wildcards (* bzw. ?) werden nicht unterstützt. Die Abfrage nach

den Optionen erscheint erst nach Eingabe des Dateinamens. Im Kopftext werden die möglichen Eingaben dargestellt. Eine Leereingabe bei der Optionsabfrage veranlaßt SHOW, die komplette Datei auszugeben. Die gleiche Funktion bietet das DOS-Kommando TYPE.

Die Optionen

Nun soll das Programm aber einigen Komfort bieten. Weiterhin muß die Zahl der auszugebenden Zeilen am Dateianfang oder -ende spezifiziert werden. Für diese Zwecke besteht die Möglichkeit, beim Aufruf verschiedene Optionen zu wählen.

Die More-Option /M

Wird die More-Option /M gesetzt, unterbricht SHOW die Ausgabe, sobald die untere Bildschirmzeile erreicht wird. In der untersten Zeile erscheint die Meldung:

```
Weiter, bitte eine Taste betätigen ...
```

Wird eine Taste gedrückt, löscht SHOW den Bildschirm und gibt die nächste Seite aus. Damit läßt sich der ausgegebene Text in Ruhe lesen, ohne daß die Bildschirmanzeige mit Strg+S angehalten werden muß. Die More-Option läßt sich mit den anderen Optionen kombinieren.

Die Option Zeilennumerierung /N

Wird diese Option gesetzt, gibt SHOW vor jeder ausgegebenen Zeile eine laufende Zeilennummer aus. Diese Information ist zum Beispiel für die Programme CUT und PASTE wichtig. Ohne die /N-Option entfällt die Ausgabe der Zeilennumerierung. Die /N-Option läßt sich mit den anderen Optionen kombinieren.

Die Line-Option /L

Diese Option erlaubt es, die Anzahl der auszugebenden Zeilen am Textanfang oder -ende zu spezifizieren. Ohne /L wird die komplette Textdatei auf dem Bildschirm ausgegeben. Soll nur der Anfang der Datei angezeigt werden, ist dies mit:

```
/L=+xx
```

anzugeben. Das Pluszeichen signalisiert, daß sich die nachfolgende Zahl (xx) auf den Dateianfang bezieht. Es sind also xx Zeilen ab dem Textanfang auf dem Bildschirm anzugeben. Für xx ist der Zahlenbereich von 1 bis 99 erlaubt. Längere Textdateien sind ohne die /L-Option auszugeben.

Alternativ besteht die Möglichkeit zur Anzeige von n Zeilen am Textende. Hierfür ist die folgende Eingabe vorgesehen:

```
/L=-xx
```

Das Minuszeichen steht für die Ausgabe des Textendes. Als Zahlenbereich für xx sind die Werte zwischen 1 und 99 zulässig. Längere Texte lassen

sich durch TYPE ausgeben. Wird gleichzeitig die /N-Option selektiert, erscheinen die Zeilennummern (bezogen auf den Textanfang) mit in der Anzeige. Die /L=+xx und /L=-xx Optionen lassen sich nicht gleichzeitig selektieren. Um sowohl den Textanfang als auch das Textende auszugeben, besteht die nachfolgend beschriebene Möglichkeit des Aufrufes über den Kommandomodus.

Der Aufruf von SHOW im Kommandomodus

Zur Benutzung in Batchdateien besteht die Möglichkeit der Parametereingabe innerhalb der Kommandozeile. Dieser Aufruf besitzt folgende Syntax:

```
SHOW <dateiname> <Options>
```

Der Dateiname entspricht den gültigen MS-DOS-Konventionen. Das Optionsfeld kann entfallen. In diesem Fall wird die komplette Datei ausgegeben. Falls keine Datei existiert, erscheint bei beiden Eingabeversionen die Fehlermeldung:

```
Datei <Filename> nicht gefunden
```

Bei der Kommandooption lassen sich die Schalter in beliebiger Reihenfolge getrennt durch Leerzeichen eingeben. Nachfolgend finden sich einige gültige Angaben für Optionen:

```
/N /M /L=+10
/L=-99 /N /M
/L= +10 /L= -30 /N
```

Die letzte Zeile enthält zweimal die /L-Option. Dies ist nach der Definition aber nicht zulässig. SHOW übernimmt in diesem Fall nur die zuerst gefundene gültige Option und ignoriert die nachfolgenden Zeichen der /L=-xx-Option.

Bei fehlerhaften Eingaben erfolgt eine Fehlermeldung. Bei der More-Option wird vor Ausgabe der ersten Zeile der Bildschirm gelöscht, während ohne More-Option die Anzeige gescrollt wird.

Die Online-Hilfe

Wie bei DOS 5.0 und den anderen Programmen läßt sich ein Bildschirm mit Online-Informationen über die Eingabe:

```
SHOW /?
```

abrufen. Dann erscheint folgender Text:

```
S H O W                          (c) Born Version 1.0

Aufruf:   SHOW
          SHOW <datei1> <optionen>
```

```
Das Programm gibt n Zeilen am Anfang oder Ende einer Text-
datei aus. Optionen:

/L=+xx   Anzeige n Zeilen vom Dateianfang
/L=-xx   Anzeige n Zeilen vom Dateiende
/N       Zeilennumerierung
/M       More-Option
```

Bild 3.9: Die Online-Hilfe von SHOW

Anschließend bricht das Programm ab, und Sie können weitere DOS-Befehle eingeben.

Der Entwurf

Damit kann mit dem Programmentwurf begonnen werden. Die Optionen /L=+xx und /N sind ohne Probleme zu erfüllen. Hier muß einfach die Datei satzweise gelesen und ausgegeben werden. Die Anzeige der Zeilennummern kann durch einen PRINT-Befehl erfolgen. Bei der More-Option ist ein interner Zähler zu führen, der beim Erreichen des unteren Bildschirmrandes ein eigenes Modul aktiviert. Dieses muß die Benutzereingaben abfragen, die Anzeige löschen und die Ausgabe auf Zeile 1 setzen

Bleibt noch die /L=-xx-Option, die etwas mehr Probleme bringt. Wie läßt sich die Zeile erkennen, ab der der Text bis zum Dateiende auszugeben ist?

In den wenigsten Fällen ist die Zahl der Zeilen in der Textdatei bekannt. Also muß eine geeignete Lösung für diese Aufgabe gesucht werden. Die erste Möglichkeit besteht darin, die Zeilenzahl in einem ersten Durchlauf zu ermitteln. Dann ist die Datei in einem zweiten Durchgang satzweise zu lesen und ab der selektierten Position auszugeben. Dies hat den großen Vorteil, daß kein großer Aufwand für die Aufbereitung entsteht. Aber bei längeren Textdateien wirkt sich das zweimalige Durchsuchen negativ auf das Laufzeitverhalten aus.

Eine andere Alternative besteht darin, sich für jeden Satz den Offset in Byte zum Dateianfang zu merken. Dann läßt sich anschließend der Lesezeiger direkt auf den ersten auszugebenden Satz positionieren. Dies spart zwar Laufzeit, bedingt aber andererseits einen hohen Verwaltungsaufwand. Bei längeren Dateien dauert der Positioniervorgang auch eine gewisse Zeit.

Für die nachfolgende Implementierung wird eine dritte Alternative gewählt. Die Zahl der auszugebenden Zeilen ist per Definition auf maximal 99 begrenzt. Ein interner Ringpuffer dient zur Aufnahme der laufenden Zeilennummern und der jeweils gelesenen Sätze. Der Puffer besitzt eine feste Größe für 99 Einträge. Dies bedeutet, daß in jeder Situation die (maximal 99) zuletzt gelesenen Sätze der Datei im Puffer vorliegen. Sobald die Datei komplett gelesen wurde, läßt sich die gewünschte Anzahl der

Textzeilen am Dateiende ausgeben.

Noch ein paar Bemerkungen zur Pufferverwaltung. Einmal findet sich eine Variable, die die Zahl der Einträge enthält. Dies ist erforderlich, da der Fall auftreten kann, daß mit /L= -xx mehr Zeilen zur Ausgabe spezifiziert werden als die Datei enthält (z.B. Datei = 50 Zeilen und /L= -99). Ein Zeiger adressiert die jeweils nächste zu beschreibende Zeile im Puffer. Da der Puffer eine ringförmige Topologie besitzt, ist der Zeiger mit Modulo 99 zu erhöhen, d.h. bei mehr als 99 Zeilen wird einfach der älteste Eintrag am Pufferanfang überschrieben. Dies hat gegenüber einer linearen Anordnung den Vorteil, daß die Sätze innerhalb des Puffers nicht verschoben werden müssen.

Die Implementierung

Um die Realisierung zu vereinfachen, wurde die Aufgabe in verschiedene Teile strukturiert und in Form von kleineren Modulen implementiert. Das folgende Hierarchiediagramm (Bild 3.10) zeigt die Zusammenschaltung aller Module.

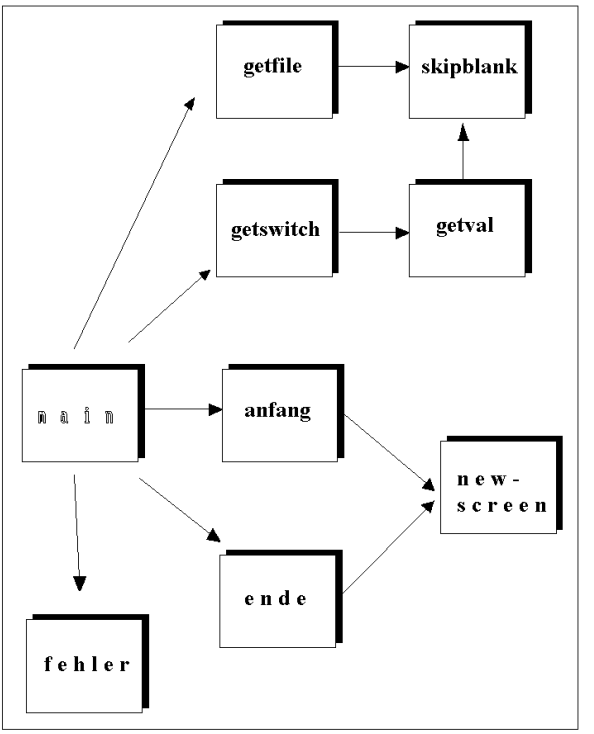


Bild 3.10: Hierarchiediagramm des Programmes SHOW

Es ist ersichtlich, daß bereits in den vorhergehenden Abschnitten entwickelte Module Verwendung finden. Dies gilt insbesondere für die

Unterprogramme *fehler*, *getswitch*, *getval*, *getfile* und *skipblank*. Dadurch beschränkt sich die Implementierung nur noch auf die restlichen Module, deren Aufbau und Funktion nachfolgend beschrieben wird.

Das Hauptmodul

In diesem Programm werden die globalen Variablen und Konstanten definiert. Die Puffergröße zur Aufnahme der eingelesenen Zeilen *(buff$())* und der Zeilennummern *(buffz&())* wird durch die Konstante *%max* spezifiziert. Durch Variation dieser Konstante lassen sich auch mehr als 99 Zeilen ausgeben. Die Bedeutung der Variablen ist im Listing beschrieben. Der Ablauf ähnelt den bereits besprochenen Programmen. Nach den Variablendefinition wird geprüft, ob in der Eingabezeile weitere Parameter vorhanden sind. Ist dies der Fall, sind Dateiname und eventuelle Optionen zu separieren. Andernfalls verzweigt das Modul in den interaktiven Abfragemodus und gibt die Kopfmeldung aus. Nach dem Öffnen der Eingabedatei sind eventuelle Optionen zu decodieren. Die jeweiligen Schalter (*more%, linenr%, outz%*) wurden bei der Initialisierung auf Standardwerte gesetzt. Dann wird je nach dem Wert von *outz%* (*/L=*) die Eingabedatei bearbeitet. Falls keine /L-Option vorliegt (*outz% = 0*), kann der Text ganz normal ausgegeben werden. Mit *outz% > 0* (*/L=+xx*) muß das Modul *anfang* aktiviert werden. Für die /L=-xx-Option ist das Modul *ende* zuständig. Die Unterscheidung erfolgt über den Wert von *outz%* in der CASE-Anweisung. Nach der Bearbeitung ist die Datei zu schließen und das Programm zu beenden.

getswitch

Hier sind die eingegeben Optionen zu decodieren und die jeweiligen Parameter zu setzen. Bei der /L-Option ist auf eine Begrenzung der Eingabewerte zu achten. Dies erfolgt durch Vergleich mit der Konstanten *%max*, was letztlich zur Flexibilität des Programmes beiträgt.

getval

Das Modul wurde so modifiziert, daß es Zahlen mit positiven und negativen Vorzeichen erkennt und decodiert.

anfang

Dieses Unterprogramm ist für die Ausgabe des Textes am Dateianfang verantwortlich. Die Datei wird in der WHILE-Schleife satzweise gelesen und mit PRINT ausgegeben, bis die spezifizierte Zeilenzahl erreicht ist. Dann bricht das Modul die Bearbeitung ab. Falls die /N-Option selektiert ist, erscheint vor jeder Zeile der Wert der Variablen *zeile&* (Zeilennummer). Der Aufruf des Unterprogrammes *newscreen* sorgt dafür, daß bei selektierter More-Option eine Benutzerabfrage am unteren Bildschirmrand erscheint. Das Modul nutzt nur einen Eintrag im Textpuffer (*buff$(1)*) für die Speicherung der Daten.

ende

Dieses Modul gibt n Zeilen am Textende auf dem Bildschirm aus. In einer eigenen WHILE-Schleife wird die Datei zuerst satzweise gelesen. Diese Schleife ist bereits so organisiert, daß die Zeichen direkt in den Ringpuffer (*buff$()*) geschrieben werden. Gleichzeitig dient das Feld *buffz&()* zur Aufnahme der Zeilennummern. In *count%* wird die Zahl der Puffereinträge geführt. Dies ist wichtig, um Dateien mit weniger als *%max* Sätzen zu erkennen, da dort nur ein Teilpuffer auszugeben ist. Nach Erreichen des Dateiendes ist die spezifizierte Zeilenzahl aus dem Puffer anzuzeigen. Vorher muß der Zeiger auf den Puffer so justiert werden, daß er auf den auszugebenden Text zeigt. Dann werden n Zeilen ausgegeben, wobei dieser Wert in *count%* steht. Es ist aber darauf zu achten, daß *count%* immer kleiner oder gleich der mit /L spezifizierten Zeilenzahl ist.

newscreen

Bei der More-Option ist die Ausgabe zu unterbrechen, sobald der untere Bildschirmrand erreicht wird. Dann muß die folgende Benutzermeldung erscheinen:

```
Weiter, bitte ein Taste betätigen ...
```

Diese Aufgabe fällt dem Modul *newscreen* zu. Falls der Wert der Variablen *scrline%* die maximale Zeilenzahl pro Bildschirmseite (*%maxscr*) übersteigt, erscheint diese Meldung. Erst nach Betätigung einer Taste löscht der Befehl CLS den Bildschirm und gibt die weitere Bearbeitung frei.

Weitere Einzelheiten sind dem nachfolgenden Listing zu entnehmen.

Erweiterungsvorschläge

Das Modul SHOW erlaubt nur die Ausgabe von 99 Zeilen am Textanfang oder -ende. Durch Variation der Konstanten *%max* läßt sich dieser Wert vergrößern. Weiterhin kann SHOW so erweitert werden, daß jeder beliebige Ausschnitt aus einer Textdatei ausgegeben werden kann. Denkbar ist auch, daß mit einem Aufruf Anfang und Ende einer Datei angezeigt werden.

```
X R E F   /Z=50                              (c) Born Version 1.0
Datei : show.bas      Datum : 05-17-1992         Seite :  1

 Zeile     Anweisung

      '*******************************************************
      '! File       : SHOW.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 6. 5.92
      '! Autor      : G. Born
      '! Files      : INPUT, OUTPUT
      '! Progr. Spr.: POWERBASIC
      '! Betr. Sys. : DOS 2.1 - 5.0
      '! Funktion: Das Programm liest eine Textdatei ein und gibt
```

```
      '!           diese auf dem Bildschirm aus. Dabei kann
angegeben
      '!           werden, wieviele Zeilen vom Textende oder -
anfang
      '!           angezeigt werden (max. 99).
      '!
      '!           /L=+xx   Anzeige des Dateianfangs
      '!           /L=-xx   Anzeige des Dateiendes
      '!           /N       Zeilennumerierung
      '!           /M       More-Option
      '!
      '! Aufruf:   SHOW                                '! Interaktiv
Mode
      '!           SHOW <datei1> <optionen>           '! Kommando
Mode
      '*********************************************************
      '! Variable definieren
    1 %true = &HFFFF: %false = 0            '! Konstante
    2 ein% = 1                              '! Kanal für I/O
    3 options$ = ""                         '! Optionen
    4 more% = %false                        '! More Option aus
    5 linenr% = %false                      '! Zeilennummern aus
    6 %max = 99
    7 DIM buff$(0:%max-1)                   '! Puffer Lesedatei
    8 DIM buffz&(0:%max-1)                  '! Puffer Zeilennummern
    9 outz% = 0                             '! Ausgaben /L=...
   10 zeile& = 0                            '! bearbeitete Zeilen
   11 %maxscr = 20                          '! Zeilen pro Screen
   12 scrline& = 0                          '! aktuelle Bildzeile

   13 ptr% = 0: hilf% = 0                   '! Hilfszeiger


   14 ON ERROR GOTO fehler

      '############################################################
      '#                      Hauptprogramm                       #
      '############################################################

   15 kommando$ = COMMAND$                  '! Parameter ?
   16 IF LEN (kommando$) = 0 THEN           '! Interaktiv Mode ?
   17  CLS                                  '! ja -> Clear Screen
      '! #####   Kopf ausgeben  ######
   18  PRINT "S H O W                       (c) Born Version 1.0"
   19  PRINT
   20  PRINT "Optionen : [ /L=+xx   Anzeige n Zeilen am Dateianfang
]"
   21  PRINT "           [ /L=-xx   Anzeige n Zeilen am Dateiende
]"
   22  PRINT "           [ /N       Zeilennumerierung  /M  More Ein
]"
   23  PRINT
   24  INPUT "Eingabedatei : ",infilename$  '! lese Dateiname
Eingabe
   25  INPUT "Optionen     : ",options$     '! lese Optionen
```

```
   26  PRINT
   27 ELSE                                    '! Kommando Mode
   28  ptr% = INSTR (kommando$,"/?")          '! Option /?
   29  IF ptr% <> 0 THEN                      '! Hilfsbildschirm
   30   PRINT "S H O W                         (c) Born Version 1.0"
   31   PRINT
   32   PRINT "Aufruf:   SHOW"
   33   PRINT "          SHOW <datei1> <optionen>"
   34   PRINT
   35   PRINT "Das Programm gibt n Zeilen am Anfang oder Ende einer
Text-"
   36   PRINT "datei aus. Optionen:"
   37   PRINT
   38   PRINT "/L=+xx   Anzeige n Zeilen vom Dateianfang"
   39   PRINT "/L=-xx   Anzeige n Zeilen vom Dateiende"
   40   PRINT "/N       Zeilennumerierung"
   41   PRINT "/M       More Option"
   42   PRINT
   43   SYSTEM
   44  END IF
      '!
      '! getfile separiert den Dateinamen aus der Kommandozeile
      '! Falls ein Name fehlt, würden die Optionen in die jeweilige
      '! Variable gespeichert. Dies ist abgefangen, da Optionen mit
      '! /.. beginnen. Dann wird ein Leerstring zurückgegeben
      '!
   45  kommando$ = UCASE$(kommando$) + " "  '! Blank als
Endeseparator
   46  ptr% = 1                              '! Parameter holen
   47  CALL getfile(ptr%, kommando$,infilename$) '! Name
Eingabedatei
   48  INCR ptr%                             '! Anfang next token
   49  hilf% = INSTR(kommando$,"/")          '! suche Optionen
   50  IF hilf% >= ptr% THEN                 '! gefunden ?
   51   options$ = MID$(kommando$,hilf%)     '! Reststring mit
Optionen
   52  END IF
   53 END IF

   54 IF infilename$ = "" THEN               '! Leereingabe ?
   55  PRINT "Der Name der Eingabedatei fehlt"
   56  END
   57 END IF

   58 OPEN infilename$ FOR INPUT AS #ein%    '! öffne Eingabedatei

   59 IF LEN(options$) > 0 THEN
   60  GOSUB getswitch                       '! lese Optionen
   61 END IF

      '!
      '! bearbeite Datei je nach Option
      '!

   62 PRINT                                  '! Leerzeile
```

```
   63 IF more% THEN CLS                        '! clear Screen bei
More

   64 SELECT CASE outz%

   65 CASE > 0
   66  CALL anfang(outz%)                      '! L=+xx Option

   67 CASE = 0
      '!-----------------------------------------------------------
      '! kopiere komplette Textdatei auf Screen
      '!-----------------------------------------------------------

   68  WHILE NOT (EOF(ein%))
   69   INCR zeile&                            '! laufende
Zeilennummer
   70   LINE INPUT #ein%, buff$(1)             '! lese eine Zeile
   71   INCR scrline&                          '! Bildzeile + 1
   72   IF more% THEN CALL newscreen           '! Bildwechsel bei More
?
   73   IF linenr% THEN PRINT zeile&;" ";      '! Zeilennr. ausgeben
   74   PRINT buff$(1)                         '! anzeigen
   75  WEND

   76 CASE < 0
   77  CALL ende (outz%)

   78 END SELECT

   79 CLOSE                                    '! Datei schließen

   80 END

      '############################################################
      '#                    Hilfsroutinen                         #
      '############################################################

   81 fehler:
      '-----------------------------------------------------------
      '! Fehlerbehandlung in SHOW
      '-----------------------------------------------------------

   82 IF ERR = 53 THEN
   83  PRINT "Die Datei ";infilename$;" existiert nicht"
   84 ELSE
   85  PRINT "Fehler : ";ERR;" unbekannt"
   86  PRINT "Programmabbruch"
   87 END IF
   88 END                                  '! MSDOS Exit
   89 RETURN

   90 getswitch:
      '-----------------------------------------------------------
      '! decodiere eingegebene Optionen
      '! /M= more   /N= Zeilennummern
```

```
      '! /L=+xx Textanfang   /L=-xx Textende
      '-------------------------------------------------------

   91  options$ = UCASE$(options$)         '! in Großbuchstaben

   92  IF INSTR(options$,"/M") > 0 THEN
   93   more% = %true                       '! More Mode ein
   94  END IF

   95  IF INSTR(options$,"/N") > 0 THEN     '! Linenr. Option ?
   96   linenr% = %true                     '! Zeilennumerierung
ein
   97  END IF

   98  ptr% = INSTR(options$,"/L=")         '! Anfang oder Ende
Text
   99  IF ptr% > 0 THEN
  100   CALL getval(options$,ptr%+3,outz%)  '! lese Zahl
  101   IF outz% > %max THEN outz% = %max   '! begrenze Wert
  102   IF outz% < -%max THEN outz% = - %max '! auf +99 bis -99

  103  END IF


  104 RETURN

  105 SUB getval(text$,ptr%,result%)
      '-------------------------------------------------------
      '! decodiere Eingabewert als Dezimalzahl
      '-------------------------------------------------------
  106 LOCAL tmp%, zchn$, leng%, sign%

  107 sign% = 1                            '! Vorzeichen +
  108 tmp% = 0                             '! Hilfsvariable

  109 leng% = LEN(text$)

  110 CALL skipblank(ptr%,text$)           '! überlese Leerzeichen
  111 zchn$ = MID$(text$,ptr%,1)           '! separ. Zeichen
  112 IF zchn$ = "+" THEN
  113  INCR ptr%                           '! auf 1. Ziffer
  114 ELSEIF zchn$ = "-" THEN              '! Vorzeichen ?
  115  sign% = -1                          '! Vorzeichen  -
  116  INCR ptr%                           '! auf 1. Ziffer
  117 END IF

  118 zchn$ = MID$(text$,ptr%,1)           '! separ. Zeichen

  119 WHILE (zchn$ >= "0") AND (zchn$ <= "9")_
  120  AND (ptr% <= leng%)                 '! n Ziffern
  121  tmp% = tmp% * 10 + VAL(zchn$)       '! Ziffer holen
  122  INCR ptr%                           '! nächstes Zeichen
  123  zchn$ = MID$(text$,ptr%,1)          '! lese Zeichen
  124 WEND
  125 result% = tmp% * sign%               '! Vorzeichen
```

```
  126 END SUB


  127 SUB getfile(ptr%,text$,result$)
      '!-------------------------------------------------------
      '! separiere Filename aus Eingabetext (text$)
      '! ptr% -> Anfang Filename, result$ = Filename
      '!-------------------------------------------------------
  128 LOCAL tmp%

  129 CALL skipblank (ptr%,text$)            '! entferne Blanks
  130 tmp% = INSTR(ptr%,text$," ")           '! suche Separator

  131 IF tmp% = 0 THEN
  132  PRINT "Fehler: kein Fileseparator"    '! kein Endeseparator
  133  END                                   '! Exit
  134 END IF
  135 IF MID$(text$,ptr%,1) = "/" THEN       '! Optionen eingegeben
?
  136  result$ = ""                          '! Leerstring
  137 ELSE
  138  result$ = MID$(text$,ptr%,tmp%-ptr%)  '! Filename
  139  ptr% = tmp%                           '! korrigiere ptr%
  140 END IF

  141 END SUB

  142 SUB anfang(maxzeile%)
      '!-------------------------------------------------------
      '! kopiere Textanfang bis maxzeile auf Screen
      '!-------------------------------------------------------

  143 SHARED more%, zeile&, ein%, scrline&, linenr%
  144 SHARED buff$()

  145 WHILE NOT (EOF(ein%))
  146  INCR zeile&                           '! laufende
Zeilennummer
  147  IF zeile& > maxzeile% THEN
  148   EXIT SUB                             '! ready
  149  END IF

  150  LINE INPUT #ein%, buff$(1)            '! lese eine Zeile
  151  INCR scrline&                         '! Bildzeile + 1
  152  IF more% THEN CALL newscreen          '! Bildwechsel bei More
?
  153  IF linenr% THEN PRINT zeile&;" ";     '! Zeilennr. ausgeben
  154  PRINT buff$(1)                        '! anzeigen
  155 WEND

  156 END SUB

  157 SUB ende (maxzeile%)
      '!-------------------------------------------------------
```

```
      '! kopiere Ende der Textdatei auf Screen. Hierzu werden
      '! %max Zeilen in einen Umlaufpuffer geschrieben. Am Datei-
      '! ende liegen dann die letzten Zeilen vor und lassen sich
      '! einfach ausgeben.
      '!-------------------------------------------------------

  158 LOCAL ptr%, count%

  159 SHARED more%, zeile&, ein%, buff$(), scrline&
  160 SHARED linenr%, buffz&()

  161 ptr% = 0                                '! Zeiger auf Buffer
  162 count% = 0                              '! keine Einträge
  163 maxzeile% = -maxzeile%                  '! Betrag bilden

  164 WHILE NOT (EOF(ein%))                   '! lese Datei
  165  INCR zeile&                            '! laufende
Zeilennummer
  166  IF count% < %max THEN INCR count%      '! zähle bis %max
  167  LINE INPUT #ein%, buff$(ptr%)          '! lese eine Zeile
  168  buffz&(ptr%) = zeile&                  '! merke Zeilennummer
  169  ptr% = (ptr% + 1) MOD %max             '! ptr auf next entry
  170 WEND

      '!
      '! Text aus dem Ringpuffer ausgeben, count definiert die
Satzzahl
      '! falls der Puffer nicht voll ist, findet sich der 1. Satz
      '! in buff$(0), sonst steht er in buff$(ptr%)
      '!

  171 IF count% < %max THEN                   '! Puffer voll ?
  172  ptr% = ptr% - maxzeile%                '! nein -> auf Anfang
  173  IF ptr% < 0 THEN ptr% = 0
  174 ELSE
  175  ptr% = ptr% - maxzeile%                '! Position im
Ringpuffer
  176  IF ptr% < 0 THEN ptr% = ptr% + %max    '! berechnen
  177 END IF                                  '! ja -> ptr% lassen
  178 IF count% > maxzeile% THEN              '! Ausgabezahl auf
  179  count% = maxzeile%                     '! /L=-xx begrenzen
  180 END IF

  181 scrline& = 0
  182 WHILE count% > 0                        '! Sätze ausgeben
  183  INCR scrline&
  184  IF more% THEN CALL newscreen           '! Bildwechsel bei More
?
  185  IF linenr% THEN PRINT buffz&(ptr%);" ";'! Zeilennr. ausgeben
  186  PRINT buff$(ptr%)                      '! anzeigen
  187  ptr% = (ptr% + 1) MOD %max             '! next entry
  188  DECR count%                            '! 1 Zeile weniger
  189 WEND

  190 END SUB
```

```
191 SUB newscreen

    '-----------------------------------------------------------
    '! More Abfrage bei vollem Bildschirm
    '-----------------------------------------------------------
192 SHARED scrline&

193 IF scrline& > %maxscr THEN
194  PRINT
195  PRINT "Weiter, bitte eine Taste betätigen ..."
196  WHILE LEN(INKEY$) = 0                '! warte auf Taste
197  WEND
198  scrline& = 0                         '! auf 1. Zeile
199  CLS                                  '! clear Screen
200 END IF

201 END SUB

202 SUB skipblank(ptr%,text$)
    '!-----------------------------------------------------------
    '! entferne führende Leerzeichen aus text$
    '!-----------------------------------------------------------

203 LOCAL lang%, zchn$

204 lang% = LEN (text$)                    '! Stringlänge
205 zchn$ = MID$(text$,ptr%,1)             '! separiere Zeichen

206 WHILE (zchn$ = " ") AND (ptr% <= lang%) '! Zeichen <> blank
207  INCR ptr%
208  zchn$ = MID$(text$,ptr%,1)            '! separiere Zeichen
209 WEND

210 END SUB

    '***** Programm Ende ******
```

Listing 3.4: SHOW.BAS

CRDEL: Ein Filter zum Entfernen von Returnzeichen

Seit den Anfangstagen der Textverarbeitung besteht das Problem, daß diese Produkte eine andere Satzformatierung verwenden als zum Beispiel die DOS-Editoren. Wer nur mit einem Textverarbeitungsprogramm oder nur mit einem Editor arbeitet, den braucht dies nicht zu stören. Aber es gibt eine Menge Leute, die vor der Aufgabe stehen, einen vorhandenen ASCII-Text in eines der vielen Textverarbeitungsprogramme zu übernehmen. Dabei stören insbesondere die »harten« Returns, die bei normalen Editioren nach jeder Zeile eingefügt werden. Dieses Zeichen werden aber bei vielen Textverarbeitungsprogrammen als Absatzmarken interpretiert. Zur Bearbeitun per Textsystem sind alle diese Returns zu

entfernen. Die manuelle Filterung im jeweiligen Textverarbeitungsprogramm ist sicherlich eine abendfüllende Beschäftigung. Abhilfe versprechen automatische Konvertierprogramme. Aber diese Tools müssen erst einmal vorhanden sein. Die Sache hat jedoch noch einen weiteren Haken: Sobald Tabellen oder fest formatierte Texte in der Datei auftreten, entfernt der Konverter auch hier die Returns. Damit müssen diese Tabellen doch wieder mühsam nachbearbeitet werden. Aus der Notwendigkeit heraus, ca. 600 Seiten ASCII-Text in das MS-Word-Format zu konvertieren, entstand die Idee für das Programm CRDEL. Da das Programm in Pascal vorlag wurde eine Portierung nach PowerBASIC vorgenommen, um auch diesem Anwenderkreis die Möglichkeit zur Dateikonvertierung zu bieten.

CRDEL beschreitet einen etwas anderen Ansatz als die meisten Konvertierprogramme. Die Textdatei wird eingelesen und auf dem Bildschirm dargestellt. Dann wird bei jeder Zeile abgefragt, ob das Return am Zeilenende zu entfernen ist. Damit ist zwar einerseits eine manuelle Interaktion notwendig. Aber der Anwender kann andererseits selbst entscheiden, wo die Returns entfernt werden sollen. Da die Steuerung des Programmes nur zwei Tasten erfordert, lassen sich auch längere Dateien in kurzer Zeit konvertieren.

Die Anforderungen

Bevor mit der Erstellung des Programmes begonnen wird, sind wieder die Anforderungen zu definieren. Das Programm soll eine ASCII-Datei satzweise einlesen und je nach Benutzereingabe die Returnzeichen entfernen. Nach der Eingabe:

```
CRDEL
```

befindet sich das Programm im Interaktiv-Modus und gibt folgende Maske aus:

```
CRDEL                                        (c) Born Version 1.0

<RET> löschen -> J eingeben, sonst eine beliebige Taste betätigen

Eingabedatei :
Ausgabedatei :
```

Die Abfrage der Dateinamen erfolgt sequentiell. Die Namen dürfen Laufwerks- und Pfadangaben enthalten. Nach Eingabe des Namens der Eingabedatei wird der Name der Ausgabedatei abgefragt. Falls beide Namen identisch sind, bricht das Programm mit einer Fehlermeldung ab:

```
Fehler:  Eingabedatei = Ausgabedatei nicht erlaubt
```

Existiert eine Eingabedatei nicht, erscheint ebenfalls eine Meldung:

```
Die Datei <Name> existiert nicht
```

und das Programm endet. Falls bereits eine Datei mit dem Namen der Ausgabedatei existiert, erfolgt eine Warnung:

```
Ausgabedatei existiert bereits, überschreiben (J/N) ?
```

Durch Eingabe des Zeichens »J« wird die Datei überschrieben. Alle anderen Eingaben beenden das Programm. Fehlende Namen für Ein- und Ausgabedateien führen ebenfalls zu einem Programmabbruch mit der meldung:

Der Name der Eingabedatei fehlt

```
Der Name der Ausgabedatei fehlt
```

Alternativ besteht die Möglichkeit, die Dateinamen bereits in der Kommandozeile beim Aufruf anzugeben. Hierfür gilt folgende Syntax:

```
  CRDEL <Eingabedatei> <Ausgabedatei>
```

Im Kommandomodus erscheint keine Kopfmeldung. Es gelten aber die bereits oben beschriebenen Fehlermeldungen und Warnungen, die auf dem Bildschirm erscheinen.

Alternativ besteht die Möglichkeit, die Online-Hilfe über die Eingabe:

```
 CRDEL /?
```

zu aktivieren. Dann erscheint folgender Text auf dem Bildschirm:

```
C R D E L                          (c) Born Version 1.0

Aufruf: CRDEL
oder    CRDEL <Quellfile> <Zielfile>

Das Programm filtert die harten Returns aus einer Textdatei.
Löschen -> J eingeben, sonst eine beliebige Taste betätigen
```

Anschließend endet das Programm wieder.

Sobald CRDEL mit der Bearbeitung der Datei beginnt, erscheinen die ersten n Zeilen der ASCII-Datei auf dem Bildschirm. Durch die Anzeige mehrerer Zeilen lassen sich Absätze und Tabellen vorausschauend erkennen und behandeln. Die aktuell zu bearbeitende Zeile wird am Textende durch die Zeichen «--- markiert. Nun muss die Taste J betätigt werden, um die Absatzmarke zu entfernen. Falls am Satzende ein getrenntes Wort auftritt, ist gleichzeit der Trennstrich zu entfernen. Wird ein andere Taste gedrückt, ist die Zeile unmodufiziert zu speichern. Anschließend rollt der Text eine Zeile nach oben und am unteren Bildrand erscheint eine neue Zeile. Dies wiederholt sich solange, bis die komplette Datei eingelesen und bearbeitet wurde. Durch diese Technik lassen sich auch große Dateien in kurzer Zeit bearbeiten.

Der Entwurf

Die Aufgabe ist recht einfach, so daß auf eine breitere Diskussion verzichtet wird. Es soll nur kurz das Konzept zur Dateibearbeitung erläutert werden. Aus Anwendersicht muß ein kompletter Textabschnitt auf dem Bildschirm erscheinen. Innerhalb dieses Textes werden die Zeilen dann bearbeitet. Dies hat zur Konsequenz, daß ein Zwischenpuffer zur Bearbeitung der eingelesenen Zeilen erforderlich wird. Dieser wird als Ringpuffer mit einem Schreib- und einem Lesezeiger implementiert. Zuerst ist der Puffer zu füllen und der Inhalt auf dem Bildschirm anzuzeigen. Dann wird jeweils ein Zeile aus dem Puffer gelesen, bearbeitet und in die Ausgabedatei gespeichert. Gleichzeitig muß der Puffer mit einem neuen Satz aus der Eingabedatei wieder aufgefüllt werden. Der Filteralgorithmus wird durch die Benutzereingaben gesteuert. Bei der Eingabe »J« ist das Return-Zeichen zu entfernen. Dies ist recht einfach, da der LINE INPUT-Befehl nur den reinen Text im Puffer ablegt. Wird hinter der Augabeanweisung (PRINT) ein Semikolon gesetzt, schreibt Basic den Text ohne Return in die Ausgabedatei. Jetzt muß aber aber noch das Textende aufbereitet werden. Wird ein Return entfernt, ist am Satzende ein Leerzeichen einzufügen, um den Satz vom nachfolgenden Wort zu trennen. Ein Sonderfall tritt bei getrennten Wörtern auf. Trennzeichen »-« sind zu entfernen, wobei kein Lerrzeichen eigefügt werden darf. Das getrennte Wort soll ja in der kombinierten Zeile zusammenstehen. Allerdings gibt es den Fall, wo der Bindestrich nicht als Trennzeichen fungiert (z.B. MS-DOS). Hier ist es nicht erwünscht, wenn CRDEL das Zeichen »-« entfernt. Deshalb wird geprüft, vo vor dem »-« ein Lerrzeichen steht. Ind iesem Fall wird das Zeichen nicht entfernt, sondern mit einem anhänden Lerrzeichen ausgegeben. Damit ist das Thema abgeschlossen.

Die Implementierung

Das Hauptmodul initialisiert die Variable und liest je nach Modus die Namen der Ein-/Ausgabedateien ein. Nach einer Überprüfung werden die Dateien geöffnet und die Bearbeitung beginnt. Zuerst sind n Zeilen aus der Datei in den Puffer zu lesen und am Bildschirm anzuzeigen (1. FOR-Schleife). Der Puffer (*inline$()*) wird über den Schreibzeiger (*zeile%*) und den Lesezeiger (*hilf%*) verwaltet. Nachdem der Puffer gefüllt ist, beginnt die eigentliche Bearbeitung des Textes. Es wird jeweils ein Satz auf dem Puffer gelesen, bearbeitet und abgespeichert. Gleichzeitig wird ein neuer Eingabesatz aus der Quelldatei in den Puffer übertragen. Mit den LOCATE-Befehlen wird der Cursor auf die erforderlichen Positionen gesetzt. Dies ist einmal für die Markierung «--- erforderlich. Weiterhin wird durch eine PRINT-Anweisung in Zeile 25 ein Bildsroll erzwungen.

Bei Erreichen des Dateiendes sorgt eine zweite FOR-Schleife für die Bearbeitung und Ausgabe des restlichen Pufferinhaltes. Dann beendet CRDEL die Bearbeitung.

fehler

Das Modul fängt PowerBASIC-Laufzeitfehler ab. Der Aufbau wurde bereits besprochen.

getfile

Das Unterprogramm separiert die Dateinamen aus dem übergebenen Kommandostring.

Einzelheiten sind dem nachfolgenden Listing zu entnehmen.

Erweiterungsvorschläge

Bei Fehleingaben besteht in der vorliegenden Version keine Möglichkeit zur Korrektur. Eine Erweiterung um eine »Undo«-Funktion für die letzte Eingabe ist sicherlich wünschenswert.

```
X R E F   /Z=50                          (c) Born Version 1.0
Datei : crdel.bas      Datum : 05-17-1992         Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : CRDEL.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 6. 5.92
      '! Autor      : G. Born
      '! Files      : INPUT, OUTPUT
      '! Progr. Spr.: POWERBASIC
      '! Betr. Sys. : DOS 2.1 - 5.0
      '! Funktion: Das Programm liest eine Textdatei ein und
      '!           filtert auf Benutzeranweisung die "harten
      '!           RETURN" Zeichen am Zeilenende aus der Datei.
      '!           Damit kann diese mit Textverarbeitungsprogrammen
      '!           wie Word bearbeitet werden.
      '!
      '! Aufruf:   CRDEL                      '! Interaktiv Mode
      '!           CRDEL <datei1> <datei2>    '! Kommando Mode
      '*********************************************************
      '! Variable definieren
    1 %max = 15                               '! Konstante
    2 ein% = 1 : aus% = 2                     '! Kanäle für I/O
    3 DIM inlinie$(0:%max-1)                  '! Puffer Lesedatei
    4 zeile% = 0
    5 ptr% = 0: hilf% = 0                     '! Hilfszeiger
    6 y% = 0                                  '! Cursor

    7 ON ERROR GOTO fehler

      '#########################################################
      '#                      Hauptprogramm                    #
      '#########################################################

    8 kommando$ = COMMAND$ + " "              '! Parameter ?
```

```
    9 IF LEN (kommando$) <= 1 THEN           '! Interaktiv Mode ?
   10  CLS                                   '! ja -> Clear Screen
      '! #####   Kopf ausgeben  ######
   11  PRINT "C R D E L                      (c) Born Version 1.0"
   12  PRINT
   13  PRINT "löschen -> J eingeben, sonst eine beliebige Taste
betätigen"
   14  PRINT
   15  INPUT "Eingabedatei : ",infilename$   '! lese Dateiname
Eingabe
   16  INPUT "Ausgabedatei : ",outfilename$  '! lese Dateiname
Ausgabe
   17  PRINT
   18 ELSE                                   '! Kommando Mode
   19  ptr% = INSTR (kommando$,"/?")        '! Option /?
   20  IF ptr% <> 0 THEN                    '! Hilfsbildschirm
   21   PRINT "C R D E L                        (c) Born Version
1.0"
   22   PRINT
   23   PRINT "Aufruf: CRDEL"
   24   PRINT " oder   CRDEL <Quellfile> <Zielfile>"
   25   PRINT
   26   PRINT "Das Programm filtert die harten Returns aus einer
Textdatei.
       "
   27   PRINT "Löschen -> J eingeben, sonst eine beliebige Taste
betätigen
       "
   28   PRINT
   29   SYSTEM
   30  END IF
      '!
      '! getfile separiert den Dateinamen aus der Kommandozeile
      '! Falls ein Name fehlt, würden die Optionen in die jeweilige
      '! Variable gespeichert. Dies ist abgefangen, da Optionen mit
      '! /.. beginnen. Dann wird ein Leerstring zurückgegeben
      '!
   31  ptr% = 1                             '! Parameter holen
   32  CALL getfile(ptr%, kommando$,infilename$) '! Name
Eingabedatei
   33  INCR ptr%                              '! Anfang next
token
   34  CALL getfile(ptr%, kommando$,outfilename$)'! Name
Ausgabedatei
   35 END IF

   36 IF infilename$ = "" THEN              '! Leereingabe ?
   37  PRINT "Der Name der Eingabedatei fehlt"
   38  END
   39 END IF

   40 IF outfilename$ = "" THEN             '! Leereingabe ?
   41  PRINT "Der Name der Ausgabedatei fehlt"
   42  END
   43 END IF
```

```
   44 IF outfilename$ = infilename$ THEN     '! Namen gleich ?
   45  PRINT "Eingabedetei = Ausgabedatei nicht erlaubt"
   46  END
   47 END IF

   48 OPEN infilename$ FOR INPUT AS #ein%   '! öffne Eingabedatei
      '! Ausgabedatei vorhanden -> prüfe über OPEN inputdatei

   49 ON ERROR GOTO ok
   50 OPEN outfilename$ FOR INPUT AS #aus%   '! existiert
Ausgabedatei%
   51 ON ERROR GOTO fehler
   52 CLOSE #aus%                            '! nein -> Close

   53 INPUT "Ausgabedatei existiert bereits, überschreiben (J/N) ?
",zchn$
   54 PRINT
   55 IF UCASE$(zchn$) <> "J" THEN END       '! stopp -> sonst
Datei weg

   56 ok:
   57 OPEN outfilename$ FOR OUTPUT AS #aus%  '! Ausgabedatei open

   58 CLS                                    '! Clear Screen
   59 PRINT                                  '! Leerzeile
      '!
      '! lese die ersten n Sätze aus der Datei und den Puffer und
      '! zeige sie an
      '!

   60 FOR i% = 0 TO %max-2
   61  LINE INPUT #ein%, inlinie$(i%)       '! lese eine Zeile
   62  PRINT inlinie$(i%)                   '! Anzeige auf Screen
   63  IF EOF(ein%) THEN GOTO weiter
   64  zeile% = i%
   65 NEXT i%

   66 weiter:
      '!
      '! Lese weitere Sätze aus der Datei und bearbeite den
      '! Puffer, zeige gelesene Sätze an.
      '!

   67 INCR zeile%
   68 WHILE (NOT (EOF(ein%)))
   69  lang% = LEN (inlinie$(hilf%))        '! Länge Satz ermitteln

   70  y% = csrlin - 1                      '! merke Zeile Cursor

   71  LOCATE 2,lang%+2                     '! Marke an akt.
   72  PRINT "«;                          '! Zeile setzen»   73
zchn$ = INPUT$ (1)                   '! lese Taste
   74  LOCATE 2,lang%+2                     '! clear Marke
   75  PRINT "  ";                          '!  "
```

```
   76  IF UCASE$(zchn$) = "J" THEN
   77   IF lang% > 0 THEN
   78    IF RIGHT$(inlinie$(hilf%),1) = "-" THEN   '! Ende = "-" ?
   79     IF MID$(inlinie$(hilf%),lang%-1,1)_      '! Ende = " -" ?
   80               <> " "  THEN                   '! END = " -"?
   81      PRINT #aus%, LEFT$(inlinie$(hilf%),_    '! entferne "-"
   82                   lang% - 1);
   83     ELSE
   84      PRINT #aus%, inlinie$(hilf%);" ";  '! volle Zeile
   85     END IF
   86    ELSE
   87     PRINT #aus%, inlinie$(hilf%);" ";   '! volle Zeile
   88    END IF
   89   END IF
   90  ELSE
   91   PRINT #aus%, inlinie$(hilf%)        '! mit RETURN ausgeben
   92  END IF

   93  LINE INPUT #ein%, inlinie$(zeile%)   '! lese neue Zeile

   94  LOCATE 25,1                          '! Cursor an unteres
Bild
   95  PRINT                                '! -> Scroll
   96  LOCATE y% ,1                         '! restore Cursor
   97  PRINT inlinie$(zeile%)               '! Anzeige auf Screen
   98  zeile% = (zeile% + 1) MOD %max       '! Adresse berechnen
   99  hilf% = (hilf% + 1) MOD %max         '!  "

  100 WEND
      '!
      '! bearbeite restlichen Puffer
      '!

  101 rest% = %max - hilf% + zeile% - 2
  102 FOR i% = 0 TO rest%
  103  lang% = LEN (inlinie$(hilf%))        '! Länge Satz ermitteln

  104  y% = csrlin - 1                      '! merke Zeile Cursor

  105  LOCATE 2,lang%+2                     '! Marke an akt.
  106  PRINT "«;                          '! Zeile setzen»  107
zchn$ = INPUT$ (1)                    '! lese Taste
  108  LOCATE 2,lang%+2                     '! clear Marke
  109  PRINT "  ";                          '!  "

  110  IF UCASE$(zchn$) = "J" THEN
  111   IF lang% > 0 THEN
  112    IF RIGHT$(inlinie$(hilf%),1) = "-" THEN   '! Ende = "-" ?
  113     IF MID$(inlinie$(hilf%),lang%-1,1) _     '! END = " -"?
  114           <> " " THEN
  115      PRINT #aus%, LEFT$(inlinie$(hilf%),_    '! entferne "-"
  116                   lang%-1);
  117     ELSE
  118      PRINT #aus%, inlinie$(hilf%);" ";  '! volle Zeile
```

```
  119     END IF
  120    ELSE
  121     PRINT #aus%, inlinie$(hilf%);" ";   '! volle Zeile
  122    END IF
  123   END IF
  124  ELSE
  125   PRINT #aus%, inlinie$(hilf%)        '! mit RETURN ausgeben
  126  END IF

  127  LOCATE 25,1                          '! Cursor an unteres
Bild
  128  PRINT                                '! -> Scroll
  129  LOCATE y% ,1                         '! restore Cursor
  130  PRINT                                '! Leerzeile
  131  hilf% = (hilf% + 1) MOD %max         '! Zeiger erhöhen
  132 NEXT i%

  133 CLOSE
  134 PRINT
  135 PRINT "CRDEL Ende"
  136 END

      '############################################################
      '#                    Hilfsroutinen                         #
      '############################################################

  137 fehler:
      '-----------------------------------------------------------
      '! Fehlerbehandlung in CRDEL
      '-----------------------------------------------------------

  138 IF ERR = 53 THEN
  139  PRINT "Die Datei ";infilename$;" existiert nicht"
  140 ELSE
  141  PRINT "Fehler : ";ERR;" unbekannt"
  142  PRINT "Programmabbruch"
  143 END IF
  144 END                                 '! MSDOS Exit
  145 RETURN


  146 SUB getfile(ptr%,text$,result$)
      '!----------------------------------------------------------
      '! separiere Filename aus Eingabetext (text$)
      '! ptr% -> Anfang Filename, result$ = Filename
      '!----------------------------------------------------------
  147 LOCAL tmp%

  148 CALL skipblank (ptr%,text$)           '! entferne Blanks
  149 tmp% = INSTR(ptr%,text$," ")          '! suche Separator
  150 IF tmp% = 0 THEN
  151  PRINT "Fehler: kein Fileseparator"   '! kein Endeseparator
  152  END                                  '! Exit
  153 END IF
  154 IF MID$(text$,ptr%,1) = "/" THEN      '! Optionen eingegeben
```

```
?
  155  result$ = ""                            '! Leerstring
  156 ELSE
  157  result$ = MID$(text$,ptr%,tmp%-ptr%)   '! Filename
  158  ptr% = tmp%                             '! korrigiere ptr%
  159 END IF

  160 END SUB

  161 SUB skipblank(ptr%,text$)
      '!-----------------------------------------------------------
      '! entferne führende Leerzeichen aus text$
      '!-----------------------------------------------------------

  162 LOCAL lang%, zchn$

  163 lang% = LEN (text$)                     '! Stringlänge
  164 zchn$ = MID$(text$,ptr%,1)              '! separiere Zeichen

  165 WHILE (zchn$ = " ") AND (ptr% <= lang%) '! Zeichen <> blank
  166  INCR ptr%
  167  zchn$ = MID$(text$,ptr%,1)             '! separiere Zeichen
  168 WEND

  169 END SUB

      '***** Programm Ende ******
```

Listing 3.5: CRDEL.BAS

PSCRIPT: Textausgabe für PostScript-Drucker

Ausgehend von dem bereits vorgestellten Programm PSLIST möchte ich nun noch ein etwas universelleres Werkzeug vorstellen, welches beliebige Textdateien für die Ausgabe auf PostScript-Geräten aufbereitet.

Der Entwurf

Auf Grund der Vorarbeiten an den bisherigen Programmen läßt sich der Entwurf auf einige wenige Einzelheiten beschränken. Die restlichen Überlegungen und Techniken können direkt aus PSLIST übernommen werden.

Die Benutzeroberfläche soll sich an den Möglichkeiten von PSLIST anlehnen. Wird das Programm mit der Eingabe:

```
PSCRIPT
```

aufgerufen, erscheint die Kopfmeldung:

```
P S C R I P T                          (c) Born Version 1.0

Optionen  [ /L=10  linker Rand      /R=500  rechter Rand    ]
```

```
          [ /O=700 oberer Rand      /U=100  unterer Rand    ]
          [ /F=10  Fontgröße/Punkt  /H      Kopf 1. Seite   ]

File     :
Optionen :
```

Bild 3.11: Kopfmeldung des Programmes PSCRIPT

Im Gegensatz zum Programm PSLIST sind die Optionen etwas verändert. Die Zeilennumerierung wird bei Textdateien in der Regel nicht mehr benötigt. Was aber erwünscht ist, ist ein Kopftext auf der ersten Seite, der den Dateinamen und das aktuelle Erstellungsdatum definiert. Diese Option soll sich jedoch zu- und abschalten lassen. Die Einstellungen für die Ränder können aus PSLIST übernommen werden.

Als Dateiname darf jeder gültige MS-DOS-Dateiname, einschließlich Laufwerks- und Pfadbezeichnung, verwendet werden. Die Abfrage der Optionen soll nach der Eingabe des Dateinamens erfolgen. Die Kopfmeldung zeigt die möglichen Eingaben für diese Optionen. Die Optionen dürfen zwar in beliebiger Reihenfolge eingegeben werden, das Format ist jedoch gemäß obigen Angaben aufzubauen. Jede Option beginnt mit dem Zeichen »/« gefolgt von einem Großbuchstaben. Bei numerischen Werten ist zwischen Zahl und Buchstabe ein Gleichheitszeichen erforderlich. Die Optionen sind durch ein Leerzeichen zu trennen. Hier sind einige gültige Optionen:

```
/H
/L=10 /O=655 /R=550 /U=50 /H
/L=5
/H /R=600
```

Fehlerhafte Eingaben (z.B. linker Rand größer als rechter Rand etc.) sind durch das Programm abzufangen. In diesem Fall erscheint die Fehlermeldung:

```
Bitte Randeinstellung neu setzen
```

Wird kein Dateiname eingegeben, bricht das Programm mit folgender Meldung ab:

```
Der Dateiname fehlt
```

Existiert die angegebene Datei nicht, endet das Programm ebenfalls mit der Meldung:

```
Die Datei  <Name>  existiert nicht
```

Wird eine Datei gefunden, beginnt die Ausgabe mit dem Hinweis:

```
Die Datei <Name> wird bearbeitet
```

Name steht dabei für den eingegebenen Dateinamen. Nach Beendigung der Ausgabe in die PostScript-Datei erscheint die Meldung:

```
Die <Datei> wurde im aktuellen Verzeichnis erzeugt
```

Sie können dann die Datei per COPY oder PRINT auf dem PostScript-Gerät

ausgeben.

Die eigentliche Ausgabe des Textes entspricht der gewohnten Form, lediglich die Schriftgröße läßt sich durch die Option /F in Schritten zu 1 Punkt variieren. Sinnvolle Angaben dürften dabei zwischen 8 Punkt und 16 Punkt liegen. Bei zu großen Schrifttypen passen die Ausgabezeilen nicht mehr auf eine Seite. Allerdings existiert in PSCRIPT keine Begrenzung der Zeilenlänge auf 75 Zeichen. Vielmehr werden alle Formatierungen innerhalb des PostScript-Programmes (HEADER.PS) vorgenommen.

Alternativ lassen sich Dateiname und Optionen mit in der Kommandozeile angeben:

```
PSCRIPT <Filename> <Optionen>
```

Dies ist insbesondere in Batchdateien interessant. Weiterhin kann eine Online-Hilfe mit dem folgenden Kommando aufgerufen werden:

```
PSCRIPT /?
```

Auf dem Bildschirm erscheint folgende Meldung:

```
P S C R I P T                    (c) Born Version 1.0

Aufruf: PSCRIPT <Filename> <Optionen>

Optionen :

  /L=10  setzt den linken Rand in Punkt
  /R=500 setzt den rechten Rand
  /O=700 setzt den oberen Rand
  /U=100 setzt den unteren Rand
  /F=10  setzt die Fontgröße in Punkt
  /H     Header 1. Seite

Das Programm gibt ein Listing als PostScript-Datei
mit der Extension xxxx.PS aus, wobei xxxx dem File-
namen entspricht. Die Ergebnisdatei kann dann auf einem
PostScript-Gerät ausgegeben werden.
```

Bild 3.12: Online-Hilfe von PSCRIPT

Anschließend endet das Programm und der DOS-Prompt erscheint wieder.

Die Implementierung

Bezüglich der Implementierung bietet es sich an, möglichst viele Teile von PSLIST übernehmen. Bild 3.13 zeigt das Moduldiagramm des Programmes.

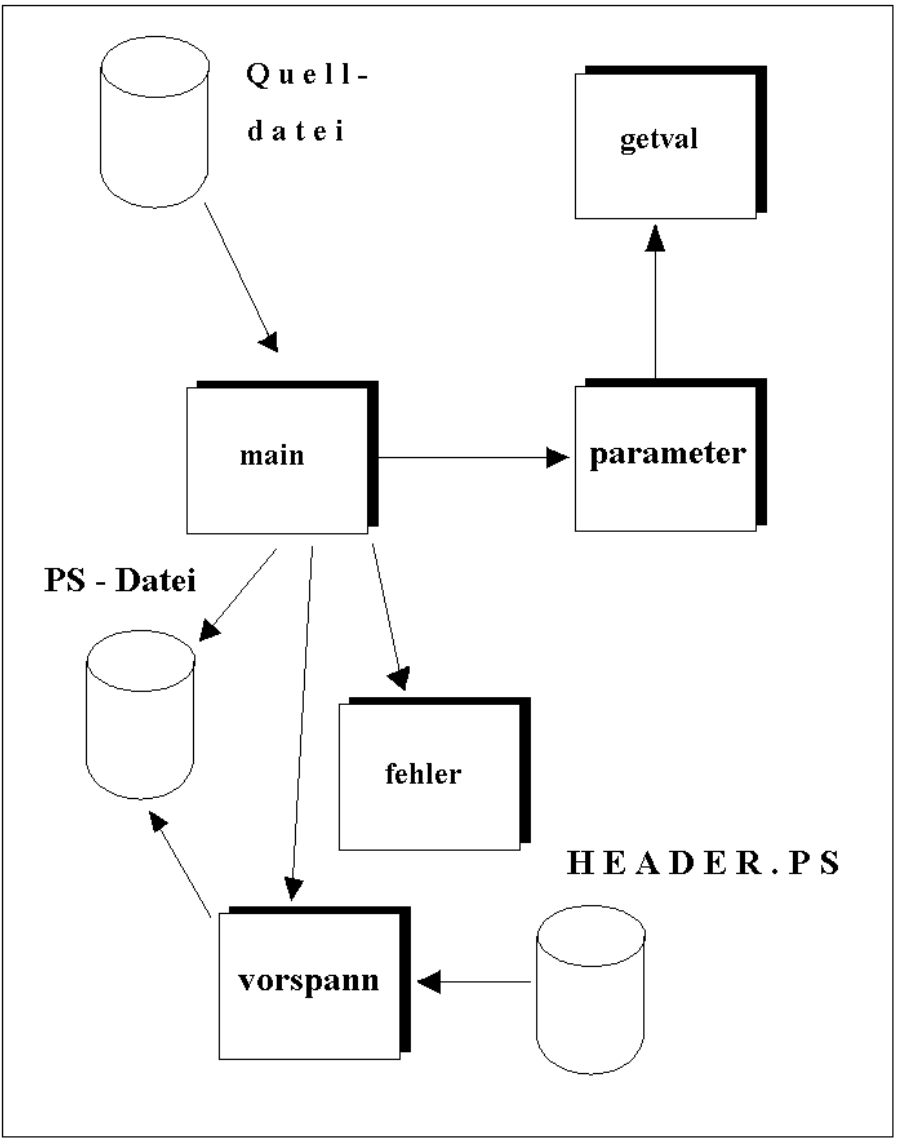


Bild 3.13: Moduldiagramm von PSCRIPT.BAS

Gerade die Formatierung der Ausgaben ist über die Datei HEADER.PS bereits gelöst. Bei der Implementierung möchte ich noch einen Schritt weitergehen. In HEADER.PS sind alle Routinen zum Zeilenumbruch und zum Seitenwechsel enthalten. Damit braucht lediglich der Text für die Ausgabe aufbereitet werden (*(text....) printline*).

Die gegenwärtige Implementierung sieht allerdings vor, daß nach jeder Zeile, die durch *printline* ausgegeben wird, ein Zeilenvorschub ausgeführt

wird. Wird bei der Ausgabe der Zeile der rechte Rand erreicht, beginnt das PostScript-Programm automatisch eine neue Zeile. Soll ein fortlaufender Text ausgegeben werden, sind die Zeilenumbrüche am Ende einer jeden Ausgabezeile störend. Dies kann aber leicht durch Modifikation der Datei HEADER.PS erreicht werden. Nachfolgend ist der Inhalt der Datei HEADER.PS zu sehen. Es muß lediglich der *crlf*-Aufruf in der Prozedur *printline* entfernt werden (die betreffende Zeile ist markiert).

```
%---> File: HEADER.PS (c) G. Born
%---> Definitionen für Textausgabe mit Umlauten
%--> definiere Konstanten
/LW { CH CH 3 div add } def % Zeilenabstand
/Blank ( ) def         % Leerzeichen

/Buff  12 dict def     % lokales Dictionary

/endtest  % Test ob Seitenende erreicht ist
{
 dup                   % Y duplizieren
 BM lt                 % unterer Rand?
 {
  showpage             % ja-> neue Seite
  pop TM               % neue Y-Koordinate
 } if                  % auf 1. Zeile
} def

/newpage               % Seitenvorschub
{
  showpage             % Seite wechseln
  LM TM moveto         % Startpunkt neue Seite
} def

/crlf                  % Zeilenvorschub
{
  LM                   % linker Rand
  currentpoint LW sub % y-dy
  exch pop             % X entfernen
  endtest              % neue Seite?
  moveto               % Zeilenanfang
} def

/printword             % Ausgabe eines Wortes
{
  dup                  % String duplizieren
  stringwidth pop      % Länge Text
  currentpoint         % aktuelle Ausgabe
  pop add LM add       % Ausgabelänge berech.
  RM gt
  { crlf } if          % Zeilenvorschub
  show                 % Text ausgeben
  ( ) show             % Leerzeichen
} def

/printline             % Ausgabe eines Satzes
{                      % Beginn der Prozedur
```

```
 {                    % Beginn der Schleife
   Blank search       % separiere Wort
   { printword pop }  % Ausgabe Wort
   { printword exit } % Ausgabe Resttext
   ifelse
 } loop               % loop alle Worte
%### die folgende Zeile muß eventuell entfernt werden
  crlf                % Zeilenvorschub ########
%###
} def

/Redefine             % Umcodierung Font
{ Buff begin          % lokales Dictionary
  /NCodeName exch def  % Hilfsvariable
  /NFontName exch def
  /AFontName exch def

  /AFontDict          % suche alten Font
   AFontName findfont def

  /NeuFont AFontDict   % neue Dictionary schaffen
   maxlength dict def

  AFontDict           % kopiere alten Font
  { exch dup /FID ne   % bis auf FID-Feld
   { dup /Encoding eq
    { exch dup length array copy
      NeuFont 3 1 roll put
    }
    { exch NeuFont 3 1 roll put
    } ifelse
   } { pop pop } ifelse
  } forall

 NeuFont
 /FontName NFontName put % setze neuen Namen
 NCodeName aload pop     % Werte laden
 NCodeName length 2 idiv % und eintragen
 { NeuFont /Encoding get
   3 1 roll put
 } repeat

 NFontName NeuFont       % definiere neuen Font
 definefont pop
 end
} def    % Ende der Prozedur /Redefine

%--> Hauptprogramm

/Umlaute              % Feld mit Umlautdefinitionen
[ 8#201 /udieresis    % ü
  8#204 /adieresis    % ä
  8#216 /Adieresis    % Ä
  8#224 /odieresis    % ö
  8#231 /Odieresis    % Ö
```

```
  8#232 /Udieresis    % Ü
  8#341 /germandbls   % ß
] def

%--> Umcodierung des Fonts
%--> Hier ist der Name des Basis-Fonts einzutragen:
%--> z.B. /Times-Roman oder /AvantGarde oder /Courier
/Courier
%--> Umdefinition des Urfonts in Font mit Umlauten
/Neu-Font-Deutsch            % neuer Font
Umlaute Redefine

/Neu-Font-Deutsch findfont % Font selektieren
CH scalefont setfont   % und initialisieren

LM TM  moveto          % Anfangspunkt

%----> Hier schließt sich der auszugebende Text an
```

Listing 3.6: HEADER.PS

Sollen am Ende eines Absatzes eine oder mehrere Leerzeilen eingefügt werden, sind eine betreffende Anzahl von *crlf*-Anweisungen in die Ausgabedatei einzufügen. Dann sorgt HEADER.PS dafür, daß die Zeilenumbrüche erfolgen:

```
(Dies ist das Ende eines Absatzes, an dem) printline
(ein Zeilenvorschub erfolgen soll) printline
crlf
crlf
```

Alle anderen Vorgaben bleiben dabei erhalten.

Hauptmodul

Im Hauptprogramm werden die Eingabeparameter eingelesen und decodiert. Dann öffnet das Programm die Eingabe- und Ausgabedatei. Der Aufruf des Unterprogrammes *vorspann* sorgt dafür, daß Teil 1 der Ausgabedatei generiert wird. In einer Schleife wird dann die Eingabedatei zeilenweise gelesen und direkt in die Ausgabedatei übertragen. Auf eine Formatierung der einzelnen Seiten kann verzichtet werden, da diese Aufgabe direkt im PostScript-Programm erfolgt. Lediglich bei gesetzter /H-Option wird der Dateiname und das aktuelle Datum auf der ersten Seite vor dem eigentlichen Text als PostScript-Text ausgegeben.

vorspann

Das Programm generiert zuerst die Konstanten für die Abmessungen des Druckbereiches in der Ausgabedatei. Dann wird noch die Fontgröße definiert. Anschließend kopiert *vorspann* den Inhalt der Datei HEADER.PS in die Ausgabedatei. Daran schließt sich später das auszugebende Listing an.

Erweiterungsvorschläge

Das Programm überschreibt die Ausgabedatei ohne Warnung. Hier könnte eine entsprechende Bedienerabfrage eingefügt werden. Weiterhin könnte der Fontname selektierbar sein. Hier muß lediglich eine Option eingelesen und die entsprechende Definition an den Anfang der PostScript-Datei geschrieben werden. Ein weiterer Punkt ist die Verfeinerung des Zeilenumbruchs, wie er bereits oben beschrieben wurde. Zur Zeit erfolgt nach jeder Ausgabezeile ein Zeilenwechsel.

```
X R E F   /Z=50                              (c) Born Version 1.0
Datei : pscript.bas      Datum : 05-17-1992         Seite :  1

 Zeile     Anweisung

      ' ********************************************************
      ' File       : PSCRIPT.BAS
      ' Vers.      : 1.0
      ' Last Edit  : 29. 4.92
      ' Autor      : G. Born
      ' File I/O   : INPUT, OUTPUT, FILE, PRINTER
      ' Progr. Spr.: POWERBASIC
      ' Betr. Sys. : DOS 2.1 - 5.0
      ' Funktion: Das Programm dient zur Ausgabe von Textdateien
      '           auf PostScript-Geräten. Der Text wird in eine
      '           Datei mit der Extension .PS konvertiert. Es
      '           lassen sich beliebige Textdateien mit diesem
      '           Programm aufbereiten. Die Steueranweisungen
      '           für den Interpreter (Seitenumbruch, Randein-
      '           stellung, etc.) werden in PSCRIPT direkt und
      '           über die Datei HEADER.PS generiert.
      '
      ' Aufruf:   PSCRIPT Filename <Optionen>
      '           Optionen:  /H   Kopftext auf 1. Seite
      '                      /L=xx linker Rand in Punkt    [  10]
      '                      /R=xx rechter Rand            [ 500]
      '                      /O=xx oberer Rand             [ 700]
      '                      /U=xx unterer Rand            [ 100]
      '                      /F=xx Fontgröße in Punkt      [  10]
      '
      '           Die Werte in [] geben die Standardeinstellung
      '           wieder. Wird das Programm ohne Parameter aufge-
      '           rufen, sind Dateiname und Optionen explizit ab-
      '           zufragen. Mit dem Aufruf:
      '
      '              PSCRIPT /?
      '
      '           wird ein Hilfsbildschirm ausgegeben.
      ' ********************************************************
      ' Variable definieren
    1 rechts% = 500                          '! rechter Rand (Punkt)
    2 links% = 10                            '! linker Rand (Punkt)
    3 oben% = 700                            '! oberer Rand (Punkt)
    4 unten% = 100                           '! unterer Rand (Punkt)
    5 font% = 10                             '! Fontgröße (Punkt)
```

```
    6 indatei% = 1                          '! Dateinummer Eingabe
    7 indatei2% = 3                         '! Dateinummer Header
    8 outdatei% = 2                         '! Dateinummer Ausgabe
    9 errorname$ = ""                       '! Dateiname bei
Fehlern

   10 ON ERROR GOTO fehler                  '! Fehlerausgang

      '############################################################
      '#                       Hauptprogramm                      #
      '############################################################

   11 kommando$ = COMMAND$                  '! Parameter ?
   12 IF LEN (kommando$) = 0 THEN           '! User Mode ?
   13  CLS                                  '! clear Screen

   14  PRINT "P S C R I P T                          (c) Born
Version 1.0"
   15  PRINT
   16  PRINT "Optionen [ /L=10  linker Rand         /R=500 rechter
Rand
        ]"
   17  PRINT "          [ /O=700 oberer Rand        /U=100 unterer
Rand
        ]"
   18  PRINT "          [ /F=10  Fontgröße/Punkt    /H     Kopf 1.
Seite
        ]"
   19  PRINT
   20  INPUT  "File     : ",filename$
   21  INPUT  "Optionen : ",options$
   22  PRINT
   23 ELSE

   24  ptr% = INSTR (kommando$,"/?")       '! Option /?
   25  IF ptr% <> 0 THEN                   '! Hilfsbildschirm
   26   PRINT "P S C R I P T               (c) Born Version 1.0"
   27   PRINT
   28   PRINT "Aufruf: PSCRIPT <Filename> <Optionen>"
   29   PRINT
   30   PRINT "Optionen :"
   31   PRINT
   32   PRINT "  /L=10  setzt den linken Rand in Punkt"
   33   PRINT "  /R=500 setzt den rechten Rand"
   34   PRINT "  /O=700 setzt den oberen Rand"
   35   PRINT "  /U=100 setzt den unteren Rand"
   36   PRINT "  /F=10  setzt die Fontgröße in Punkt"
   37   PRINT "  /H     Kopfzeile auf 1. Seite"
   38   PRINT
   39   PRINT "Das Programm wandelt einen Textfile in eine
PostScript-"
   40   PRINT "Datei mit der Extension xxxx.PS um, wobei xxxx dem
File-"
   41   PRINT "namen entspricht. Die Ergebnisdatei kann dann auf
```

```
einem"
   42   PRINT "PostScript-Gerät ausgegeben werden."
Zeile     Anweisung

   43   PRINT
   44   SYSTEM
   45  END IF
   46 ||||      '! Kommando Mode
   47  ptr% = INSTR (kommando$,"/")         '! Optionen ?
   48  IF ptr% = 0 THEN
   49   filename$ = kommando$               '! nur Filename
   50  ELSE
   51   filename$ = LEFT$(kommando$,ptr% -1) '! Filename separieren
   52   options$  = MID$(kommando$,ptr%)    '! Optionen separieren
   53  END IF
   54 END IF

   55 GOSUB parameter                      '! Optionen decodieren

   56 IF (rechts% < links%) or (oben% < unten%) THEN  '! sinnlose
   57  PRINT                                          '! Einstellung
   58  PRINT "Bitte Randeinstellung neu setzen"     '! Fehlerexit
   59  END                                 '! Exit
   60 END IF

   61 IF filename$ = "" THEN               '! Leereingabe ?
   62  PRINT
   63  PRINT "Der Dateiname fehlt"
   64  END                                 '! Exit
   65 END IF

   66 ptr% = INSTR(filename$,".")          '! hat Datei eine
Extension?
   67 IF ptr% > 0 THEN
   68  outfile$ = LEFT$(filename$,ptr%) + "PS" '! Filename ohne
Extension
   69 ELSE
   70  outfile$ = filename$ + ".PS"        '! Extension anhängen
   71 END IF

      ' prüfe ob Datei vorhanden, nein -> exit

   72 errorname$ = filename$

   73 OPEN filename$ FOR INPUT AS #indatei%  '! Öffne Eingabedatei
   74 OPEN outfile$ FOR OUTPUT AS #outdatei% '! Öffne Ausgabedatei
   75 PRINT
   76 PRINT "Die Datei: ";filename$;" wird bearbeitet"

   77 GOSUB vorspann                       '! Vorspann generieren

   78 WHILE NOT (EOF(indatei%))            '! Datei sequentiell
lesen
   79  LINE INPUT #indatei%, linie$        '! lese Zeile
```

```
   80  '! scan line auf (..) und wandle in \( oder \) um
   81  linie1$ = ""
   82  FOR i% = 1 to LEN(linie$)
   83   zchn$ = MID$(linie$,i%,1)
   84   IF (zchn$ = "(") or (zchn$ = ")") THEN
   85    linie1$ = linie1$ + "\"
   86   END IF
   87   linie1$ = linie1$ + zchn$
   88  NEXT i%

   89  PRINT #outdatei%, "(";linie1$;") printline" '! schreibe
Zeile
   90 WEND

   91 PRINT #outdatei%, "showpage"          '! Abschluß PS-Datei
   92 PRINT #outdatei%, "% END of File"

   93 CLOSE #indatei%                       '! Datei schließen
   94 CLOSE #outdatei%                      '! Datei schließen
   95 PRINT
   96 PRINT "Die Datei: ";filename$;" wurde im aktuellen
Verzeichnis erzeugt"
   97 END

      '#############################################################
      '#                    Hilfsroutinen                          #
      '#############################################################

   98 fehler:
      '-----------------------------------------------------------
      '! Fehlerbehandlung in PSCRIPT
      '-----------------------------------------------------------

   99 IF ERR = 53 THEN
  100  PRINT "Die Datei ";errorname$;" existiert nicht"
  101 ELSE
  102  PRINT "Fehler : ";ERR;" unbekannt"
  103  PRINT "Programmabbruch"
  104 END IF
  105 END                                  '! MSDOS Exit
  106 RETURN

  107 parameter:
      '-----------------------------------------------------------
      '! Decodiere die Eingabeoptionen
      '-----------------------------------------------------------

  108 options$ = UCASE$(options$)

  109 ptr% = INSTR (options$,"/L=")
  110 IF ptr% > 0 THEN CALL getval (links%) '! linker Rand

  111 ptr% = INSTR (options$,"/R=")
  112 IF ptr% > 0 THEN CALL getval (rechts%) '! rechter Rand
```

```
  113 ptr% = INSTR (options$,"/O=")
  114 IF ptr% > 0 THEN CALL getval (oben%)   '! oberer Rand

  115 ptr% = INSTR (options$,"/U=")
  116 IF ptr% > 0 THEN CALL getval (unten%)  '! unterer Rand

  117 ptr% = INSTR (options$,"/F=")
  118 IF ptr% > 0 THEN CALL getval (font%)   '! Fontgröße

  119 RETURN

  120 SUB getval (wert%)
      '-----------------------------------------------------------
      '! Decodiere den Eingabestring in eine Zahl
      '-----------------------------------------------------------
  121  SHARED options$, ptr%
  122  LOCAL i%

  123  ptr% = ptr% + 3                       '! ptr hinter /x=
  124  i% = 1
  125  WHILE ((ptr%+i%) =< LEN (options$)) and
(MID$(options$,ptr%+i%,1) <
       > " ")
  126   i% = i% + 1                          '! Ziffernzahl + 1
  127  WEND
  128  wert% = VAL(MID$(options$,ptr%,i%))  '! decodiere die Zahl
  129 END SUB

  130 vorspann:
      '-----------------------------------------------------------
      '! generiere Vorspann mit PostScript-Anweisungen
      '-----------------------------------------------------------
  131 errorname$ = "HEADER.PS"

  132 OPEN "HEADER.PS" FOR INPUT AS #indatei2%  '! Header öffnen
  133 PRINT "Generiere Fileheader"

  134 PRINT #outdatei%, "%%!PS-Adobe-2.0 EPSF-1.2"
  135 PRINT #outdatei%, "%%Title: ",filename$
  136 PRINT #outdatei%, "%%Creator: PSCRIPT 1.0 (c) Born G."
  137 PRINT #outdatei%, "%%EndComments"
  138 PRINT #outdatei%, ""
  139 PRINT #outdatei%, "%%BeginSetup"
  140 PRINT #outdatei%, "/LM ";links%;" def        % linker Rand"
  141 PRINT #outdatei%, "/RM ";rechts%;" def       % rechter Rand"
  142 PRINT #outdatei%, "/TM ";oben%;" def         % oberer Rand"
  143 PRINT #outdatei%, "/BM ";unten%;" def        % unterer Rand"
  144 PRINT #outdatei%, "/CH ";font%;" def         % Fontgröße"
  145 PRINT #outdatei%, ""

  146 WHILE NOT (EOF(indatei2%))            '! Datei sequentiell
lesen
  147  LINE INPUT #indatei2%, linie$        '! lese Zeile
  148  PRINT #outdatei%, linie$             '! schreibe
  149 WEND
```

```
  150 PRINT #outdatei%, "%%EndSetup"
  151 PRINT #outdatei%, ""

  152 CLOSE #indatei2%

      '! generiere Kopfzeile, falls Option /H gesetzt
  153 IF (INSTR (options$,"/H") <> 0) THEN
  154  PRINT #outdatei%, "(P S C R I P T    "; options$;
SPACE$(27);
  155  PRINT #outdatei%, "(c) Born Version 1.0) printline"
  156  PRINT #outdatei%, "(Datei : ";filename$;"     Datum :
";DATE$;
  157  PRINT #outdatei%, ") printline"
  158  PRINT #outdatei%,
  159 END IF

  160 RETURN

  161 END
```

Listing 3.7: PSCRIPT.BAS

4 Werkzeuge für den Umgang mit dem PC

Dieses Kapitel beschäftigt sich mit Hilfsmitteln zur Unterstützung des Betriebssystemes MS-DOS. Ein Tool zur Ausgabe beliebiger Dateien im Hexformat gehört ebenso dazu, wie ein Utility zur Textsuche in EXE- und COM-Dateien. Beginnen wir mit dem ersten Programm dieser Reihe.

CALC: Rechnen in verschiedenen Zahlensystemen

Bei der Entwicklung von Programmen fallen häufig kleinere Berechnungen an. Dies gilt sowohl bei der maschinennahen Programmierung in Assembler, wo Adreßberechnungen im Dezimal- und Hexadezimalsystem durchzuführen sind, als auch für die Programmierung in Hochsprachen. Einige Softwarepakete (z.B. grafische Benutzeroberflächen) bieten zwar auch kleine »Rechner« die meist per Maus zu bedienen sind. Neben der Tatsache, daß dies recht umständlich ist, fehlt häufig die Möglichkeit, Ein- und Ausgaben in verschiedenen Zahlensystemen vorzunehmen.»Wünschenswert ist es, zur Programmentwicklung einen Rechner zu haben, der ein gemischte Eingabe arithmetischer Ausdrücke in verschiedenen Zahlenformaten erlaubt. Die Eingaben sollten zweckmäßigerweise per Tastatur erfolgen. Für professionelle Softwareentwickler werden entsprechende Geräte im Handel angeboten. Aus Kostengründen bietet es sich aber an, den eigenen Personalcomputer mit einem kleinen Programm auszustatten, welches genau diese Funktionen übernimmt. Nachfolgend wird eine solche Lösung in PowerBASIC entwickelt.

Die Spezifikation

Zuerst sollen die Anforderungen hinsichtlich Funktion und Bedieneroberfläche spezifiziert werden. Das Programm muß folgende Aufgaben erfüllen:

- Eingabemöglichkeiten für Zahlen im Dezimal, Hexadezimal- und Binärsystem.
- Verarbeitung gemischter arithmetischer Ausdrücke.
- Ausführung der Grundrechenarten (+ - * /).
- Darstellung des Ergebnisses in den verschiedenen Zahlensystemen.
- Anzeige der invertierten Zahlenwerte.
- Anzeige des jeweiligen ASCII-Äquivalents.

- Anzeige des eingegebenen Ausdrucks.
- Anzeige von Fehlermeldungen.

Die Eingabe gemischter Zahlen im Dezimal-, Binär- und Hexadezimalsystem kommt in der Praxis immer wieder vor. Weiterhin sind öfter Ausdrücke zu berechnen, wobei einzelne Zahlen durchaus verschiedene Darstellungen haben können. Die Kennzeichnung der Zahlenbasis soll (abweichend von der Notation in PowerBASIC) durch einen nachgestellten Buchstaben erfolgen:

```
B  Binärwert       (z.B.  1010B)
T  Dezimalwert     (z.B.    20T)
H  Hexadezimalwert (z.B.    10H)
```

Dies kommt der normalen Schreibweise entgegen, wobei bei Dezimalzahlen auf die Angabe der Zahlenbasis (T) verzichtet werden kann. Wird nur ein Wert eingegeben, erscheint das Ergebnis in der Darstellung der jeweils anderen Zahlensysteme. Bei den vier Grundrechenarten gilt die Regel »Punktrechnung geht vor Strichrechnung« Die Möglichkeit der Klammerung von Ausdrücken wird in vorliegendem Programm nicht implementiert.»Da der Bildschirm über mehrere Zeilen zu je 80 Spalten verfügt, kann die Darstellung der Ergebnisse recht ausführlich erfolgen. Einmal läßt sich eine Darstellung des Ergebnisses in den verschiedenen Zahlensystemen nebeneinander erreichen. In einigen Fällen interessiert auch die invertierte Darstellung. Diese ist ebenfalls leicht realisierbar. Bei Werten zwischen 20H und FFH soll weiterhin das jeweilige ASCII-Zeichen eingeblendet werden. Für Kontrollzwecke und bei Fehleingaben ist der eingegebene Ausdruck auf den Bildschirm auszugeben.

Bei der dezimalen Zahlendarstellung gibt der Rechner die Zahlen mit einem entsprechenden Vorzeichen aus (-32768 bis 32767). Bei Binär- und Hexadezimalzahlen ist dies aber nicht sinnvoll, da hier eigentlich nur die interne Darstellung des 16-Bit-Wertes interessiert. Hier wird also auf das Vorzeichen verzichtet, was zu einer Anzeige des Ganzzahlenbereichs von 0 .. FFFFH führt. Wird beispielsweise ein negativer Wert eingegeben oder berechnet, erscheint das Ergebnis in der jeweiligen Repräsentation als Hexadezimalzahl. Diese auf den ersten Blick unsinnige Form erweist sich bei Adreßberechnungen als vorteilhaft. Bei 16-Bit-Prozessoren (auch der 8086) reicht der Adreßraum von 0 bis 64 Kbyte. Beim Überlauf wird einfach das führende Bit abgeschnitten und der Rest dargestellt. Die Addition von FFFEH + 20H führt dann zum Ergebnis 001FH. Eine Anzeige negativer Werte ist hier nutzlos, während sich die Hexzahl direkt weiterverwenden läßt. Der negative Wert läßt sich übrigens aus der Inversdarstellung durch Addition einer 1 bestimmen, da die Inversdarstellung im Einerkomplement erfolgt. Das bedeutet, daß alle Bits lediglich invertiert werden. Die Eingabe -10 führt beispielsweise zu folgenden Anzeige:

```
        Ergebnis       invertiert
DEZ.      -9               9
HEX.    FFF6            0009
```

Das Programm baut nach dem Start eine Maske auf (Bild 4.1), in der sowohl die Eingaben erfolgen als auch die Ergebnisse angezeigt werden.

```
  D E Z  -  H E X  -  B I N  -  R E C H N E R


           Ergebnis       invertiert
         +-----------+    +----------+
DEZ.      xxxxxxxxxx       xxxxxxxx
         +-----------+    +----------+
HEX      xxxxxxxxxx       xxxxxxxx
         +-----------+    +----------+
BIN.      xxxxxxxxxx       xxxxxxxx
         +-----------+    +----------+
ASCII     x            Eingabe xxxxx

Die Eingabe: EXIT beendet das Programm

Eingabe : ?
```

Bild 4.1: Ergebnisdarstellung des Rechners

Die Benutzereingaben erfolgen im untersten Feld (*Eingabe : ?*). Die Eingabezeile ist durch Betätigung der Eingabetaste abzuschließen, woraufhin die Ergebnisse am Bildschirm angezeigt werden. Folgende Terme werden z.B. als gültig akzeptiert:

```
100H + 12T * 20H + +14
10T - 101B / 2 + 110B --3T
15H / 3T + 3 * 5H + -101B
```

Nach der Betätigung der EingabeTaste wird das Eingabefeld in der untersten Zeile gelöscht, um neue Benutzereingaben zu ermöglichen. Vor der Betätigung der Eingabetaste lassen sich die PowerBASIC-Editiermöglichkeiten nutzen. Neben der Anzeige der Ergebnisse erscheint aber der eingegebene Text zusätzlich im Feld »Eingabe« Wird ein Fehler in der Eingabezeile erkannt, erscheint in der Eingabezeile eine entsprechende Fehlermeldung. Zusätzlich markiert CALC die Fehlerstelle mit dem Zeichen ^.

```
ASCII     x          Eingabe 30 + 15 - D
                                       ^
Eingabe : ungültige Ziffer, bitte die Return-Taste betätigen
```

Nachdem die Fehlermeldung mit der Eingabetaste quittiert wurde, steht der Rechner für neue Eingaben zur Verfügung. Insgesamt sind folgende Fehlermeldungen (Tabelle 4.1) möglich:

errx	err_txt$	Bedeutung
0		kein Fehler
1	ungültige Ziffer	falsche Ziffer innerhalb der Zahl z.B. 1013B
2		unbelegt
3	ungültiger Operator	Operator nicht + - * /
4	ungültige Zahlenbasis	Basis nicht T B H

Tabelle 4.1: Fehlermeldung des Rechners

Das Programm CALC läßt sich durch Eingabe des Wortes EXIT beenden. Damit soll die Beschreibung der Bedieneroberfläche abgeschlossen werden.

Der Entwurf

Nachdem die Ein-/Ausgaben festgelegt sind, soll noch kurz auf einige spezielle Überlegungen hinsichtlich der Realisierung einiger Funktionen eingegangen werden.

Einmal stellt sich die Frage, wie die unterschiedlichen Zahlensysteme behandelt werden. Es wird davon ausgegangen, daß der Rechner intern im binären Zahlenformat arbeitet. Die Auflösung soll auf 16 Bit (2-Byte-Integer-Werte) begrenzt bleiben. Damit besteht eigentlich nur noch die Aufgabe, die Eingaben jeweils in die interne Darstellung des Rechners umzuwandeln und das Ergebnis in Form der verschiedenen Zahlenformate auf dem Bildschirm anzuzeigen. Um eine Dezimalzahl einzulesen, existiert die Basic-Funktion *VAL()*. Problematisch ist es jedoch, wenn die Zeichenkette ungültige Ziffern enthält. In diesem Fall tritt ein Laufzeitfehler auf, der zwar durch die Routine *fehler* abgefangen wird. Aber das Programm bricht nach Ausgabe der Basic-Fehlermeldung ab, da leider keine Möglichkeit besteht, an die Unterbrechungsstelle im Programm zurückzukehren (ein Fehler kann ja an vielen Stellen auftreten). Zur Decodierung einer Dezimalzahl wird deshalb der Eingabetext Zeichen für Zeichen separiert. Jedes dieser Zeichen entspricht dann einer Dezimalziffer. Vor der Decodierung läßt sich nun prüfen, ob das Zeichen im Bereich von 0 bis 9 liegt. Andernfalls kann die Decodierung mit einer internen Fehlermeldung abgebrochen werden. Im Modul *dec1* wird genau diese Technik angewandt, um einen ASCII-Text in eine Dezimalzahl zu konvertieren. Analog kann bei der Konvertierung von Binär- und Hexadezimalzahlen verfahren werden.

Es soll aber noch auf ein Problem eingegangen werden, welches bei der Implementierung in PowerBASIC auftritt. Der Rechner soll intern mit 2-Byte-Integer-Werten arbeiten. Nun werden diese Zahlen aber mit einem Vorzeichen versehen, so daß der Darstellungsbereich das Intervall -32768 bis +32767 umfaßt. Es können also auch nur Zahlen in diesem Wertebereich verarbeitet werden. Wichtig ist bei der Berechnung des Ergebnisses, daß kein Überlauf auftritt, da sonst das Programm mit einem Laufzeitfehler (Overflow) abbricht. Bei der Decodierung der Dezimalzahlen verträgt PowerBASIC diese Behandlung. Aber bei der Wandlung von Hexzeichen in eine Hexzahl treten Probleme auf. Normalerweise läßt sich eine Hexzahl gemäß folgendem Algorithmus bestimmen:

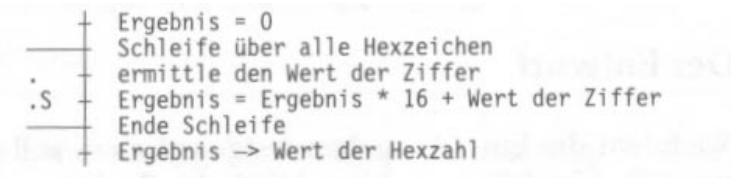


Bild 4.2: Wandlung einer Hexadezimalzahl

Es werden also die einzelnen Hexzeichen gelesen, in den äquivalenten Hexwert umgerechnet und zum Ergebnis addiert. Der Ergebniswert ist vor jeder Addition mit dem Wert 16 zu multiplizieren. Damit wird lediglich eine Verschiebung um eine Stelle nach links erreicht (0FFH * 16 = 0FF0H). In Power BASIC führt dieses Verfahren aber bei Hexzahlen größer 7FFFH zu einem Overflowfehler. Der Grund liegt in der vorzeichenbehafteten Darstellung der 2-Byte-Integer-Zahlen. Falls der Wert 8000H eingelesen werden soll, liegt folgende Situation vor:

```
1 Ziffer   8     Wert -> 8H
2 Ziffer   0     Wert -> 80H
3 Ziffer   0     Wert -> 800H
4 Ziffer   0     Wert -> 8000H -> Overflow
```

Bei den ersten drei Ziffern liegt das Ergebnis im positiven Zahlenbereich. Sobald jedoch die Multiplikation 800H * 16 durchgeführt wird, tritt ein Laufzeitfehler auf. Das Ergebnis der Multiplikation 800H * 16 ist in der Hexdarstellung der Wert 8000H, was normalerweise keine Probleme bereitet. In der Dezimalnotation tritt jedoch ein Vorzeichenwechsel auf:

```
800H                          = +2048T
800H * 16 = 8000H             = -32768
```

In der Dezimaldarstellung liegt plötzlich eine negative Zahl vor. Da eine Multiplikation zweier positiver Zahlen keine negatives Ergebnis erzeugt, nimmt PowerBASIC einen numerischen Überlauf an und bricht mit einem Laufzeitfehler ab. Diese Eigenart muß natürlich abgefangen werden. Hierzu wird vor Addition der vierten Ziffer geprüft, ob das Ergebnis größer 7FFH ist. In diesem Fall wird vor der Multiplikation das oberste Bit gelöscht. Dann ist die Multiplikation und Addition auszuführen. Dies verhindert einen Überlauf, bringt aber ein falsches Ergebnis. Also ist zum

Schluß noch das oberste Bit zu setzen. Dieser Trick wird im Modul *hex1* angewandt, da andere Lösungen nicht zufriedenstellend arbeiten.

Der nächste Punkt gilt der Frage, wie arithmetische Ausdrücke auszuwerten sind. Diese können sowohl beliebige Zahlenformate, als auch ein beliebige Schachtelung der Operatoren (*/+-) aufweisen. Es muß also ein Analyseprogramm erstellt werden, welches einmal die Zahlen mit den jeweiligen Zahlenbasen sowie die Operatoren erkennt. Eine separierte Zahl läßt sich dann in die interne Darstellung konvertieren. Auch das Erkennen der Operatoren ist nicht sonderlich schwierig. Problematischer ist aber dann die Auswertung des Ausdruckes. Da bei der Eingabe die übliche Schreibweise benutzt wird (auf eine umgekehrte polnische Notation wurde verzichtet), läßt sich der Ausdruck nicht einfach von links nach rechts auswerten:

```
13 + 15H * 11B + 18 / 2
```

Es gilt nach wie vor, daß die Punktrechnung vor der Strichrechnung durchzuführen ist. Tritt bei der Auswertung eine Multiplikation oder Division auf, muß erst das Zwischenergebnis bestimmt werden. Eine Lösungsmöglichkeit besteht darin, den Algorithmus zur Auswertung des Ausdrucks rekursiv zu formulieren. Sobald eine Punktrechnung auftritt, wird die Rekursion gestartet, um den Teilausdruck auszuwerten. Aus Aufwandsgründen wurde aber in vorliegendem Fall auf diese Lösung verzichtet. Vielmehr kommt ein einfacherer Ansatz zum Tragen. Dieser beruht auf der Überlegung, daß alle Zahlen (Operanden) und die Operatoren in Felder eingelesen werden. Für die Zahlen existiert ein eigenes Feld, in dem alle Wert nacheinander abgelegt werden. Die Operatoren werden ebenfalls decodiert und in einem zweiten Feld abgelegt. Nach der Analyse liegt der obige Ausdruck in folgender Form vor:

```
     13     +  15H    *  11B    -  18     /   2
-> wert(1) + wert(2) * wert(3) - wert(4) / wert(5)
```

Nun ist bei der Auflösung einfach die Punktrechnung vorzuziehen, d.h. die Multiplikation (*wert(2) * wert(3)* sowie die Division (*wert(4) / wert(5)*) sind zuerst auszuführen. Die Ergebnisse werden dann im niedrigen Feldelement abgespeichert:

```
wert(2) = wert(2) * wert(3)
wert(4) = wert(4) * wert(5)
```

Der Ausdruck reduziert sich dann auf die Form:

```
-> wert(1) + wert(2)     ---    - wert(4)    ---
```

Die Einträge für die Operatoren (* /) sowie die Operatoren (*wert(3), wert(5)*) werden gelöscht. Da nun nur noch Strichrechnung vorkommt, ist der verbleibende Ausdruck von links nach rechts auszuwerten. Gelöschte Einträge sind zu überlesen. Die Lösung ist sicherlich nicht die eleganteste, erfüllt ihren Zweck aber voll und ganz. Damit soll der Entwurf abgeschlossen werden.

Die Implementierung

Das Programm gliedert sich in mehrere Module, deren Zusammenschaltung nachfolgendem Hierarchiediagramm (Bild 4.3) zu entnehmen ist.

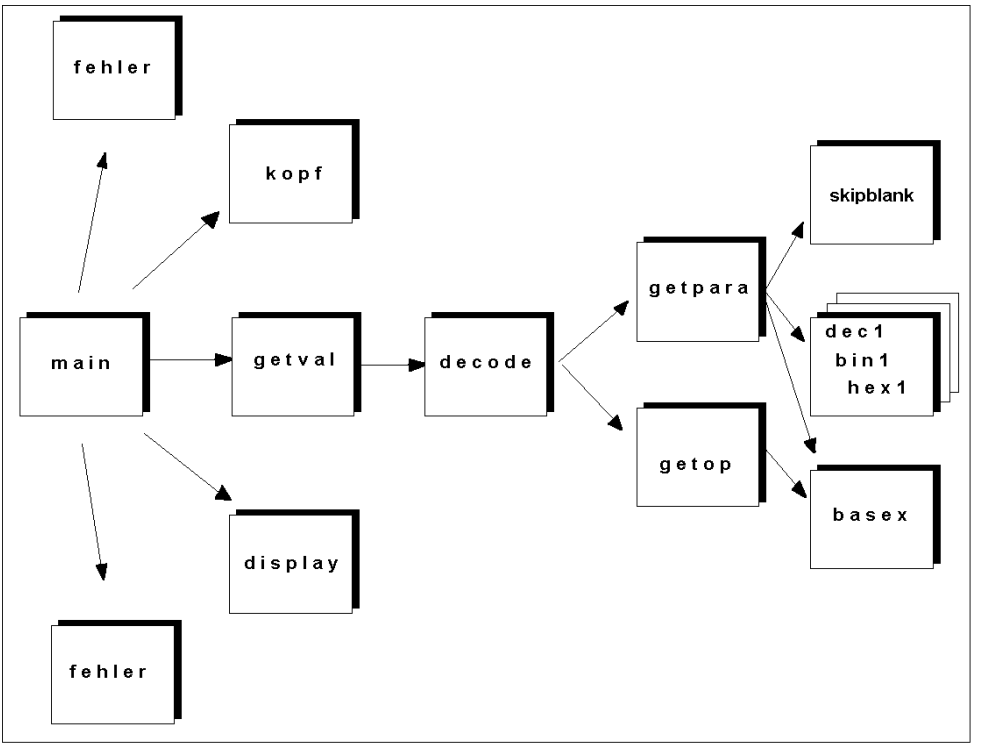


Bild 4.3: Modulhierarchie des Rechners

Damit kann die Implementierung nach der üblichen Methode erfolgen.

Das Hauptprogramm

Nach dem Definitionsteil für die Variablen schließt sich das Hauptprogramm (*main*) an. Neben der Ausgabe der statischen Bildschirmmaske mittels des Unterprogrammes *kopf* beginnt eine Endlosschleife. In dieser werden die Werte eingelesen sowie die Ergebnisse berechnet (*decode*) und ausgegeben (*display*). Werden Eingabefehler erkannt, sind diese über das Unterprogramm *fehler* auszugeben. Solche Fehler lassen sich durch den Benutzer quittieren, während Laufzeitfehler des PowerBASIC-Systems über das Modul *fehler1* abgefangen sind. Laufzeitfehler führen allerdings zu einem Programmabbruch. Mit Strg+Untbr läßt sich das Programm abbrechen. Die restlichen Aufgaben werden von den Hilfsmodulen bearbeitet.

fehler

Bei Fehleingaben übernimmt das Modul *fehler* die Ausgabe einer entsprechenden Fehlermeldung. Die Meldungen stehen im Textfeld *errtxt$()*, wobei die übergebene Variable *errx%* die Fehlerart anzeigt. Das Modul gibt gleichzeitig den eingegeben Ausdruck aus und markiert die fehlerhafte Stelle durch das Zeichen ^. Die Position wird in etwa durch die Variable *errptr%* angegeben. Durch drücken eine beliebigen Taste wird der Fehler quittiert und eine neue Eingabe ist möglich. Die Fehlertexte wurden bereits vorgestellt.

fehler1

Dieses Modul wird bei PowerBASIC-Laufzeitfehlern aktiviert und gibt eine Basic-Fehlernummer aus. Dann bricht das Programm ab.

kopf

Hier erfolgt die Ausgabe des statischen Maskenteils. Die Koordinaten sind in den Konstanten *%tx* und *%ty* abgelegt.

decode

Dieses Modul ist für die Auswertung eines Ausdruckes zuständig. In einer Schleife werden alle Eingabewerte (*getval*) und die Operatoren (*getop*) gelesen. Dann erfolgt die Berechnung des Ausdruckes. Hierbei ist die Priorität der Grundrechenarten zu berücksichtigen. Es wird dabei die im Abschnitt über den Programmentwurf diskutierte Technik verwendet. Das Feld *wertx%()* enthält alle eingegebenen Zahlen. Das Ergebnis einer Operation auf zwei Zahlen wird in das obere Feldelement zurückgespeichert.

```
wert(1) *  wert(2)  -> wert(2) / wert(3) -> wert(3)
```

Dadurch läßt sich die Rechnung wesentlich vereinfachen. Die Entscheidung, ob eine Punktrechnung vorliegt, kann durch Auswertung der jeweiligen Feldvariable *opc%()* erfolgen. In einem zweiten Schritt wird die Strichrechnung dann nachgezogen. Operationen, die Punktrechnung erfordern, sind dabei zu übergehen. Am Ende der Auswertung liegt das Ergebnis in der Variablen *wert%* vor.

getval

Dieses Modul separiert aus dem Eingabetext die durch *ptr%* adressierte Zahl und ermittelt über die Unterprogramme *basex*, *dec1*, *hex1* und *bin1* sowohl die Zahlenbasis als auch den Wert der Zahl.

getop

Analog dem Programm *getval* dient dieses Modul zu Analyse des Eingabetextes. Es liest aber keine Zahlen, sondern die durch *ptr%* adressierten Operatoren, und gibt das Ergebnis in *opcode%* zurück.

display

Dieses Modul übernimmt die Ausgabe der berechneten Ergebnisse. Die Ausgabekoordinaten sind durch Konstanten definiert. Die Variable *wert%* enthält das Originalergebnis. In *nwert%* steht die inverse Darstellung. Die Ausgabe der Dezimalzahlen erfolgt mit einer einfachen PRINT-Anweisung, während für Hex- und Binärzahlen die Funktionen HEX$() und BIN$() verwendet werden. Leider kann hier kein Format angegeben werden. Dies führt aber zu einer Darstellung mit Leerzeichen an den unbelegten Stellen. Deshalb werden diese Leerstellen durch die nachfolgende Anweisung mit Nullen belegt:

```
PRINT STRING$(x - LEN(res$),"0");res$
```

Bei Werten im Bereich 20H bis FFH wird die entsprechende ASCII-Darstellung mit eingeblendet. Die Binärziffern werden in Gruppen zu je 8 Bit dargestellt, um die Lesbarkeit zu erhöhen.

basex

In diesem Unterprogramm wird das letzte Zeichen einer separierten Zahl untersucht. Abhängig von diesem Zeichen wird die Zahlenbasis (hex, bin, dez) für die nachfolgende Decodierung festgelegt.

hex1, bin1, dez1

Alle drei Module besitzen die Aufgabe, den markierten String in eine Zahl zu wandeln. Dabei ist jedes der Module für eine bestimmte Zahlenbasis zuständig. Falsche Zeichen sind zu erkennen und mit einer Fehlermeldung abzuweisen. Die Tricks bei der Decodierung der Hexadezimalzahlen wurden bereits beim Entwurf besprochen.

Damit sind die wesentlichen Module des Programmes CALC vorgestellt. Einzelheiten können dem Listing entnommen werden.

Erweiterungsvorschläge

Das Programm erlaubt derzeit keine Eingaben von Ausdrücken, die Klammern enthalten. Eine Erweiterung in dieser Hinsicht wäre denkbar. Die interne Darstellung mit 32-Bit-Zahlen ist ebenfalls möglich.

```
X R E F   /Z=50                              (c) Born Version 1.0
Datei : calc.bas      Datum : 05-31-1992          Seite :  1

 Zeile     Anweisung

      '*******************************************************
      '! File       : CALC.BAS
      '! Vers.      : 1.0
      '! Last Edit  :  7.5.92
      '! Autor      : G. Born
      '! Files      : INPUT, OUTPUT
      '! Progr. Spr.: POWER Basic
      '! Betr. Sys. : DOS 2.1 - 5.0
```

```
      '! Funktion: Das Programm simuliert einen Hex-Dez-Bin-
Rechner.
      '!           Die Eingaben werden übernommen, decodiert und
das
      '!           Ergebnis wird im Bildschirmspeicher jeweils
normal
      '!           und invertiert als Dezimal-, Hexadezimal- und
      '!           Binärzahl angezeigt. Zusätzlich werden ASCII -
      '!           Darstellung und Eingabe eingeblendet.
      '************************************************************
      '! Variable definieren
      '! globale Konstanten
    1  %true = -1: %false = 0
    2  %hex = 1: %bin = 2: %dec = 3          '! code für
Zahlenbasis
    3  %add = 1: %sub = 2: %mul = 3: %div = 4'! code für
Operationen
      '! globale Variablen
    4  %maxentry = 10
    5  DIM wertx%(1:%maxentry)               '! Speicher für 10
Parameter
    6  DIM opc%(1:%maxentry)                 '!     "     "  10
Operatoren
    7  wert% = 0                             '! Ergebnis
    8  count% = 0                            '! Zahl der Parameter
    9  errx% = 0                             '! Fehlernummer
   10  errptr% = 0                           '! ptr auf Fehler im
Text

      '! definiere die Koordinaten der Ein- / Ausgabefelder
   11  %y1 = 8: %y2 = 10: %y3 = 12: %y4 = 14 '! y Koordinaten
   12  %y5 = 19
   13  %x1 = 12: %x2 = 42: %x3 = 28: %x4 = 15 '! x Koordinaten
   14  %tx1 = 12: %ty1 = 2                   '! Textkoordinaten
   15  %tx2 = 17: %tx3 =45: %ty2 = 5
   16  %tx4 = 5: %tx5 = 18

   17  lang% = 0                             '! Länge Eingabetext
   18  text$ = ""                            '! Eingabetext
   19  DIM errtxt$(1:4)                      '! Fehlertexte
   20  errtxt$(1) = "ungültige Ziffer"
   21  errtxt$(2) = "----------------"
   22  errtxt$(3) = "ungültiger Operator"
   23  errtxt$(4) = "ungültige Zahlenbasis"

      '############################################################
      '#                       Hauptprogramm                      #
      '############################################################

   24 ON ERROR GOTO fehler1

   25 CALL kopf                          '! Bildschirmmaske ausgeben
   26 WHILE %true                        '! Endlosschleife
   27  errx% = 0                         '! clear Fehlervariable
   28  LOCATE %y5,%x4,1                  '! Auf Eingabefeld
```

```
29  INPUT "";text$                    '! Eingabe lesen und decod.

30  '!  Test auf EXIT-Befehl
31  IF (INSTR(UCASE$(text$),"EXIT") <> 0) THEN
32   END
33  END IF

34  LOCATE %y5,%x4,1
35  PRINT SPACE$(30)                  '! clear Eingabefeld
36  lang% = LEN(text$)                '! Stringlänge
37  CALL decode (text$)               '! Eingabe decodieren
38  IF errx% = 0 THEN
39   CALL display (wert%)             '! Ergebnis ausgeben
40  ELSE
41   CALL fehler (errx%)              '! Fehlermeldung ausgeben
42  END IF
43  wert% = 0
44  FOR a% = 1 to %maxentry           '! clear results
45   wertx%(a%) = 0
46  NEXT a%
47 WEND
48 END

   '#############################################################
   '#                     Hilfsroutinen                         #
   '#############################################################

49 SUB kopf
   '!-----------------------------------------------------------
   '! Ausgabe des statischen Maskenteils auf dem Bildschirm
   '!-----------------------------------------------------------

50  CLS
51  LOCATE %ty1,%tx1,0                  '! setze Cursor
52  PRINT "D E Z   -   H E X   -    B I N   -   R e c h n e r"
53  LOCATE %ty2,%tx2,1
54  PRINT "Ergebnis";
55  LOCATE %ty2,%tx3,0
56  PRINT "invertiert";

57  LOCATE %y1-1,%x4-4,0
58  PRINT "Ú------------------¿";       '! zeichne Rahmen
59  PRINT SPACE$(10);
60  PRINT "Ú------------------¿";

61  LOCATE %y1,%tx4,0
62  PRINT "DEZ.  |";SPACE$(18);"|";
63  PRINT SPACE$(10);"|";SPACE$(18);"|";

64  LOCATE %y2-1,%x4-4,0
65  PRINT "Ã------------------´";
66  PRINT SPACE$(10);
67  PRINT "Ã------------------´";

68  LOCATE %y2,%tx4,0
```

```
 69  PRINT "HEX.  |";SPACE$(18);"|";
 70  PRINT SPACE$(10);"|";SPACE$(18);"|";

 71  LOCATE %y3-1,%x4-4,0
 72  PRINT "Ã------------------´";
 73  PRINT SPACE$(10);
 74  PRINT "Ã------------------´";

 75  LOCATE %y3,%tx4,0
 76  PRINT "BIN.  |";SPACE$(18);"|";
 77  PRINT SPACE$(10);"|";SPACE$(18);"|";

 78  LOCATE %y4-1,%x4-4,0
 79  PRINT "À------------------Ù";
 80  PRINT SPACE$(10);
 81  PRINT "À------------------Ù";

 82  LOCATE %y4,%tx4,0
 83  PRINT "ASCII";
 84  LOCATE %y5,%tx4,0
 85  PRINT "Eingabe :";
 86  LOCATE %y5-2,%tx4,0
 87  PRINT "Die Eingabe: EXIT beendet das Programm";
 88  LOCATE %y4,%tx5,0
 89  PRINT "Eingabe :";

 90 END SUB


 91 SUB decode (text$)

    '!-----------------------------------------------------------
    '! bearbeite Eingabe und berechne Ergebnis
    '!-----------------------------------------------------------
 92 LOCAL ptr%, l%, flag%

 93 SHARED lang%, count%, errx%, wert%
 94 SHARED wertx%(), opc%()

 95  ptr% = 1: errx% = 0: count% = 1     '! init lokale variable
 96  WHILE ptr% <= lang%                 '! scan String
 97   CALL getval (ptr%,wertx%(count%))  '! ermittle 1. Parameter
 98   IF errx% > 0 THEN EXIT SUB         '! Error Exit
 99   IF wertx%(count%) = 0 THEN GOTO ready '! WHILE EXIT
100   CALL getop (ptr%,opc%(count%))     '! ermittle 1. Operator
101   IF errx% > 0 THEN EXIT SUB         '! Error Exit
102   INCR ptr%                          '! hinter Operator
103   INCR count%                        '! nächste Zelle
104  WEND

105  DECR count%                         '! Zahl der Werte

106 ready:
    '! nur 1 Parameter gefunden, oder Leereingabe ?
107  IF count% < 2 THEN
```

```
  108   wert% = wertx%(1)
  109   EXIT SUB
  110  END IF

      '! Punktrechnung "*"  "/" vorziehen !!!
  111  FOR l% = 1 TO count%-1
  112   IF opc%(l%) = %mul THEN
  113    wertx%(l%+1) = wertx%(l%) * wertx%(l%+1)
  114   ELSE
  115    IF opc%(l%) = %div THEN
  116     wertx%(l%+1) = INT(wertx%(l%) / wertx%(l%+1))
  117    END IF
  118   END IF
  119  NEXT l%

      '! Strichrechnung "+"  "-" nachziehen !!!
  120  FOR l% = 1 TO count%-1
  121   WHILE (opc%(l%) = %mul) OR (opc%(l%) = %div) '! skip A * B
+ ...
  122    INCR l%
  123   WEND
  124   j% = l%+1: flag% = %false              '! clear gefunden
  125   WHILE (opc%(j%) = %mul) OR (opc%(j%) = %div) '! skip A + B
* C ...
  126    INCR j% : flag% = %true               '! setze gefunden
  127   WEND

  128   IF opc%(l%) = %add THEN
  129    wertx%(j%) = wertx%(l%) + wertx%(j%)
  130   ELSE
  131    IF opc%(l%) = %sub THEN
  132     wertx%(j%) = wertx%(l%) - wertx%(j%)
  133    END IF
  134   END IF
  135   IF flag% THEN l% = j%-1
  136  NEXT l%

      ' FOR n% = 1 TO count%
      '  PRINT "n= ";n%;" wert ";wertx%(n%);" opc "; opc%(n%)
      ' NEXT n%

  137  wert% = wertx%(count%)                 '! Endergebnis

  138 END SUB

  139 SUB getval (ptr%, wert%)

      '!-------------------------------------------------------
      '! lese eine Zahl ein und decodiere sie
      '!-------------------------------------------------------

  140 SHARED text$, errx%, count%, lang%, vorz%
  141 LOCAL tmp%, zchn$, basis%, first%, last%

  142  vorz% = 1 : tmp% = 0               '! init Hilfsvariablen
```

```
      '! suche Anfang und Ende der Zahl
  143  CALL skipblank  (ptr%,text$)        '! skip führende
Leerzeichen

      '! Vorzeichen bearbeiten
  144  zchn$ = MID$ (text$,ptr%,1)         '! hole Zeichen

  145  IF (zchn$ = "-") THEN
  146   vorz% = -1                         '! negative Zahl
  147   INCR ptr%
  148  ELSE
  149   IF (zchn$ = "+") THEN              '! Vorz. überlesen
  150    INCR ptr%
  151   END IF
  152  END IF
  153  zchn$ = MID$ (text$,ptr%,1)         '! hole Zeichen
  154  first% = ptr%                       '! merke Anfang Zahl

      '! suche Ende der Zahl = " ";"+";"-";"*";"/"
  155  WHILE (INSTR(" +-*/",zchn$) = 0) _
  156                AND ptr% <= lang%
  157   INCR ptr%                          '! hole nächstes Zeichen
  158   zchn$ = MID$ (text$,ptr%,1)        '! hole Zeichen
  159  WEND
      '! merke Ende der Zahl
  160  IF ptr% < lang% THEN
  161   last% = ptr% - 1                   '! auf letzte Ziffer
  162  ELSE
  163   last% = lang%                      '! auf Textende
  164  END IF

      '! decodiere Zahlenbasis
  165  zchn$ = UCASE$(MID$ (text$,last%,1))'! hole Zeichen
  166  CALL basex (basis%,zchn$,last%)     '! decodiere Basis
  167  IF errx% > 0 THEN EXIT SUB          '! error exit

  168  SELECT CASE basis%                  '! decodiere Zahl

  169  CASE %hex
  170   CALL hex1 (first%,last%,wert%)     '! Hexzahl

  171  CASE %bin
  172   CALL bin1 (first%,last%,wert%)     '! Binärzahl

  173  CASE %dec
  174   CALL dec1 (first%,last%,wert%)     '! Dezimalzahl

  175  END SELECT

  176 END SUB

  177 SUB getop (ptr%,opcode%)

      '!-----------------------------------------------------------
```

```
    '! ermittle operator (+ - * / )
    '!-------------------------------------------------------

178 SHARED errx%, errptr%, text$, lang%
179 LOCAL  zchn$, tmp%

180  CALL skipblank (ptr%,text$)         '! überlese Leerzeichen

181  IF ptr% >= lang% THEN
182   opcode% = 0                        '! nichts gefunden
183   EXIT SUB
184  END IF

185  zchn$ = MID$(text$,ptr%,1)          '! hole Zeichen
186  tmp% = INSTR ("+-*/",zchn$)         '! decodiere Operator

187  SELECT CASE tmp%                    '! Zuweisung Opcode

188  CASE 1
189   opcode% = %add                     '! Addition

190  CASE 2
191   opcode% = %sub                     '! Subtraktion

192  CASE 3
193   opcode% = %mul                     '! Multiplikation

194  CASE 4
195   opcode% = %div                     '! Division

196  CASE ELSE
197   errx% = 3                          '! ungültiger Operator
198   errptr% = ptr%                     '! Zeiger setzen
199   opcode% = 0

200  END SELECT

201 END SUB

202 SUB display (wert%)

    '!-------------------------------------------------------
    '! Ausgabe des Ergebnisses auf dem Bildschirm
    '!-------------------------------------------------------
203 LOCAL nwert%, res$
204 SHARED text$

205  nwert% = (NOT wert%)              '! Complement
    '! Ausgabe der Werte in DEZ HEX BIN
206  LOCATE %y1,%x1+12,0
207  PRINT USING "######"; wert%       '! Dezimalzahl
208  LOCATE %y1,%x2+12,0
209  PRINT USING "######"; nwert%;     '! Einerkomplement
210  LOCATE %y2,%x1+14,0
211  res$ = HEX$(wert%)                '! Hexausgabe mit
```

```
  212  PRINT STRING$(4-LEN(res$),"0");res$  '! führend. Nullen
  213  LOCATE %y2,%x2+14,0
  214  res$ = HEX$(nwert%)                 '! Hexausgabe
  215  PRINT STRING$(4-LEN(res$),"0");res$
  216  LOCATE %y3,%x1+1,0
  217  res$ = BIN$(wert%)                  '! Binärausgabe mit
  218  res$ = STRING$(16-LEN(res$),"0") + res$ '! führ. Nullen
  219  PRINT MID$(res$,1,8);":";MID$(res$,9)
  220  LOCATE %y3,%x2+1,1
  221  res$ = BIN$(nwert%)                 '! Binärausgabe mit
  222  res$ = STRING$(16-LEN(res$),"0") + res$ '! führ. Nullen
  223  PRINT MID$(res$,1,8);":";MID$(res$,9)
  224  LOCATE %y4,%x1,1
  225  PRINT " "
  226  IF (wert% >= &H20) AND (wert% < 256) THEN
  227   LOCATE %y4,%x1,1
  228   PRINT CHR$(wert% MOD 256);         '! ASCII Wert
  229  ELSE
  230   LOCATE %y4,%x1,1
  231   PRINT " "                          '! ASCII Feld
  232  END IF
  233  LOCATE %y4,%x3,1
  234  PRINT text$;
  235  PRINT SPACE$(30-LEN(text$));       '! clear Restfeld

  236 END SUB


  237 SUB skipblank(ptr%,text$)
      '-------------------------------------------------------
      '! zähle führende Leerzeichen in einer Zeichenkette
      '! text$ = Zeichenkette, zeiger% = Zeiger in Kette
      '-------------------------------------------------------
  238 SHARED lang%

  239 WHILE (ptr% =< lang%) and (MID$(text$,ptr%,1) = " ")
  240  INCR ptr%
  241 WEND
  242 END SUB

  243 SUB basex (basis%, zchn$, last%)

      '!-------------------------------------------------------
      '! decodieren der Zahlenbasis
      '! "H" basis = 1, "B" basis = 2, "T" basis = 3
      '!-------------------------------------------------------

  244 LOCAL tmp%
  245 SHARED text$, errx%, errptr%

  246  DECR last%                         '! Zeiger auf letzte
Ziffer
  247  tmp% = INSTR("HBT",zchn$)          '! decodiere Zeichen

  248  SELECT CASE tmp%
```

```
  249  CASE 1
  250   basis% = %hex                    '! Hexzahl

  251  CASE 2
  252   basis% = %bin                    '! Binärzahl

  253  CASE 3
  254   basis% = %dec                    '! Dezimalzahl

  255  CASE ELSE

  256   IF (zchn$ >= "0") AND (zchn$ =< "9") THEN  '! Ziffer = 0 ..
9
  257    INCR last%
  258    basis% = %dec                   '! Dezimalzahl
  259   ELSE
  260    errx% = 4                       '! 1 Fehler
  261    errptr% = last%                 '! Fehler Exit
  262   END IF

  263  END SELECT

  264 END SUB


  265 SUB hex1 (first%,last%,wert%)

      '!-------------------------------------------------------
      '! decodieren der Hexzahl
      '!-------------------------------------------------------

  266 SHARED text$, vorz%, errx%, errptr%
  267 LOCAL tmp%, b%

  268  wert% = 0                        '! init

  269  FOR b% = first% TO last%         '! alle Ziffern
  270   zchn$ = UCASE$(MID$(text$,b%,1)) '! hole Ziffer

  271   tmp% = INSTR("0123456789ABCDEF",zchn$) '! decodiere Ziffer

  272   IF tmp% = 0 THEN                '! Wert gefunden ?
  273    errx% = 1: errptr% = b%         '! Nein -> Fehler
  274    EXIT SUB
  275   END IF
      '!
      '! Achtung: Power Basic kann bei I*2 keine Zahlen größer
      '! 7FFFH verarbeiten (8000H führt zu Overflow). Deshalb
      '! wird das oberste Bit bei Zahlen > 8000H gelöscht und
      '! zum Abschluß wieder gesetzt (sorry).
      '!

  276   IF (b% = 4) AND wert% > &H7FF THEN '! Teste auf Overflow
  277     wert% = wert% AND &H7FF        '! clear oberes Bit
```

```
  278     wert% = wert% * 16 + (tmp% - 1)'! Ziffer auf Zahl
addieren
  279     wert% = wert% OR &H8000        '! setze oberes Bit
  280   ELSE
  281    wert% = wert% * 16 + (tmp% - 1) '! Ziffer auf Zahl
addieren
  282   END IF
  283   wert% = wert% * vorz%

  284  NEXT b%

  285 END SUB


  286 SUB bin1 (first%,last%,wert%)

      '!-------------------------------------------------------
      '! decodieren der Binärzahl
      '!-------------------------------------------------------

  287 SHARED text$, vorz%, errx%, errptr%
  288 LOCAL tmp%, b%

  289  wert% = 0                         '! init
  290  FOR b% = first% TO last%          '! alle Ziffern
  291   zchn$ = MID$(text$,b%,1)         '! lese Ziffer

      '! gültige Ziffer ????
  292   IF zchn$ < "0" OR zchn$ > "1" THEN
  293    errx% = 1                       '! Fehlerausgang
  294    errptr% = b%
  295   ELSE
  296    tmp%  = VAL(zchn$)              '! Wert der Ziffer
  297    wert% = wert% * 2 + tmp% * vorz% '! Ziffer auf Zahl
addieren
  298   END IF
  299  NEXT b%

  300 END SUB


  301 SUB dec1 (first%,last%,wert%)

      '!-------------------------------------------------------
      '! decodieren der Dezimalzahl
      '!-------------------------------------------------------

  302 SHARED text$, vorz%, errx%, errptr%
  303 LOCAL tmp%, b%

  304  wert% = 0                         '! init

  305  FOR b% = first% TO last%          '! alle Ziffern
  306   zchn$ = MID$(text$,b%,1)         '! hole Ziffer
```

```
        '! Achtung VAL funktioniert nicht, da 0 bei Fehler
geliefert wird
  307   tmp% = INSTR ("0123456789",zchn$) '! Wert der Ziffer
  308   tmp% = tmp% - 1
      '! gültige Ziffer ????
  309   IF tmp% < 0 THEN
  310    errx% = 1                      '! Fehlerexit
  311    errptr% = b%
  312   ELSE
  313    wert% = wert% * 10 + (tmp% * vorz%) '! Ziffer auf Zahl
addieren
  314   END IF
  315  NEXT b%

  316 END SUB

  317 fehler1:
      '!-------------------------------------------------------
      '! Abfangroutine für Power Basic Fehler
      '!-------------------------------------------------------

  318 IF ERR = 6 THEN
  319  LOCATE %y4, %x3,0
  320  PRINT "Overflow Error";               '! Fehlermeldung
  321 ELSE
  322  PRINT "Fehler : ";ERR;
  323 END IF

  324 END
  325 RETURN


  326 SUB fehler (fehlernr%)

      '!-------------------------------------------------------
      '! Fehlerausgabe auf dem Bildschirm
      '!-------------------------------------------------------
  327 SHARED text$, errtxt$()

  328  LOCATE %y4,%x3,0
  329  PRINT text$;                         '! display Eingabe
  330  PRINT SPACE$(30-LEN(text$));         '! lösche Restfeld
  331  LOCATE (%y4+1%),(%x3+2+errptr%),0
  332  PRINT "^";                           '! Fehlerstelle
markieren
  333  LOCATE %y5,%x4,1
  334  PRINT errtxt$(fehlernr%);            '! Fehlermeldung
ausgeben

  335  INPUT ", bitte die Return Taste betätigen ", text$
  336  LOCATE %y5,%x4,0
  337  PRINT SPACE$(60)                     '! clear Meldung
  338  LOCATE (%y4+1),(%x3+2+errptr%)
  339  PRINT " "                            '! Clear ^
```

```
340 END SUB

    '**** Programm Ende ****
```

Listing 4.1: CALC.BAS

DUMP: Dateiausgabe im Hexformat

Wer sich unter MS-DOS den Inhalt bestimmter Dateien anschauen möchte, erlebt manchmal einige Überraschungen. Textdateien lassen sich mit den Kommandos COPY, LIST oder PRINT ausgeben. Aber bei COM- und EXE-Dateien klappt dies nicht mehr. Auf dem Bildschirm erscheinen merkwürdige Zeichen, die keinen Sinn ergeben. Ähnliches tritt bei allen Dateien auf, die kein ASCII-Format enthalten. Es gibt für Insider zwar einen Ausweg über DEBUG, aber dies ist mit einigen Tücken verbunden. Am Markt angebotene Zusatztools erlauben ebenfalls die Anzeige von Dateien im Hexmode, kosten aber Geld und lohnen nicht immer die Anschaffung. Nachfolgend wird deshalb eine entsprechende Lösung in PowerBASIC vorgestellt.

Die Anforderungen

Beginnen wir zunächst mit der Beschreibung der Programm Ein-/Ausgabemeldungen. Es soll der Inhalt beliebiger Dateien auf dem Bildschirm ausgeben werden. Geeignet ist für diese Zwecke die Darstellung im Hexadezimalformat, wobei jeweils 16 Bytes pro Zeile aufgegeben werden. Vor jeder Zeile sollte eine fortlaufende Adresse stehen. Eine ASCII-Darstellung in der Folgezeile ist auch recht nützlich. Nicht darstellbare Steuerzeichen werden durch einen Punkt (».« markiert. Damit ergibt sich in etwa folgende Anzeige:»

```
Adr          Werte

0000 00 01 FF 40 41 53 43 49 49 20 23 23 04 05 FE F7
      .  .  .  @  A  S  C  I  I     #  #  .  .  .  .
0010 12 0A 0D .....
      .  .  .
```

Bild 4.4: DUMP-Ausgabeformat

Nachfolgend wird eine präzisere Spezifikation der Bedieneroberfläche erstellt. Das Programm soll durch die folgende Eingabe gestartet werden:

```
DUMP
```

In diesem Fall meldet es sich mit dem Kopftext:

```
 D U M P                          (c) Born  Version 1.0
 Options   [ /Wide  /More  /Print                    ]
 File :
 Options :
```

Bild 4.5: Kopfanzeige nach dem Programmstart

Mit der Abfrage »File : wird der Dateiname eingelesen. Generell sind gültige Pfadangaben erlaubt. Die Abfrage nach »Options :« erscheint erst nach Eingabe des Dateinamens. Wird die Datei nicht gefunden, erscheint die Meldung:

```
Die Datei <Name> existiert nicht
```

und das Programm bricht ab. Andernfalls beginnt das Programm mit der Ausgabe der Werte in oben gezeigtem Format. Dieses Format wird nachfolgend als Normal-Modus bezeichnet, d.h. die Standardeinstellung ist wirksam.

Bei einem gefüllten Bildschirm wird ein »scroll« ausgeführt. Das Programm bricht ab, falls das Dateiende erreicht ist oder falls die Tasten STRG+Unterbr gedrückt werden.

Für Auswertungen der Anzeige ist der »Bildscroll« störend. Weiterhin soll eine Ausgabe auf dem Drucker möglich sein. Solche Eigenschaften lassens ich über das Optionsfeld selektieren.

Die More-Option /M

Die More-Option erlaubt eine seitenorientierte Bildschirmausgabe. Sobald die Ausgabe den unteren Bildschirmrand erreicht, unterbricht das Programm mit der Meldung:

```
Weiter, bitte eine Taste betätigen ..
```

Soll die Ausgabe fortgesetzt werden, ist eine Taste zu betätigen. Dann wird der Bildschirm gelöscht und die nächste Seite angezeigt. Die Option wird mit der Eingabe /M im Optionsfeld aktiviert. Am Bildschirm erscheint folgende Ausgabe:

```
 Adr           Werte
0000 00 01 FF 40 41 53 43 49 49 20 23 23 04 05 FE F7
      .  .  .  @  A  S  C  I  I     #  #  .  .  .  .
0010 12 0A 0D .....
      .  .  .
      .  .  .

Weiter, bitte eine Taste betätigen ...
```

Bild 4.6: Ausgabe mit More-Option im Normal-Modus

Die More-Option ist nur bei Ausgaben auf dem Bildschirm wirksam.

Die Wide-Option /W

Alternativ besteht die Möglichkeit, die ASCII-Darstellung nicht in der zweiten Zeile unter den Hexadezimalwerten sondern am Zeilenende auszugeben. Die führt zu einer komprimierteren Anzeige.

```
 Adr          Werte
0000 00 01 FF 40 41 ..... 20 23 23 04 05 FE F7  ...@ASCII ##....
0010 12 0A 0D 30 31 ..... 35 36 37 38 39 40 41  ...012 3456789@A
0020 .  .  .
```

Bild 4.7: Anzeige mit der Wide-Option

Diese Option läßt sich mit der More-Option kombinieren:

```
 Adr          Werte
0000 00 01 FF 40 41 ..... 20 23 23 04 05 FE F7  ...@ASCII ##....
0010 12 0A 0D 30 31 ..... 35 36 37 38 39 40 41  ...012 3456789@A
0020 .  .  .
     .  .  .

Weiter, bitte eine Taste betätigen ..
```

Bild 4.8: Wide- und More-Option kombiniert

Dann erfolgt die Ausgabe seitenorientiert.

Die Printer-Option /P

Weiterhin soll die Möglichkeit bestehen, die Ausgaben vom Bildschirm auf den Drucker umzuleiten. Hierfür ist die Printer-Option vorgesehen. Eine Kombination mit der Wide-Option ist erlaubt, während die More-Option unterdrückt wird. Die Eingabe /P aktiviert die Druckerausgabe. Jede Seite beginnt mit einem Kopf, welcher den Dateinamen und die Seitennummer enthält. Die Seitenlänge ist fest auf 60 Zeilen pro Seite eingestellt. Während des Ausdrucks erscheint auf dem Bildschirm die Meldung:

```
Ausgabe auf dem Drucker
```

In Bild 4.9 ist ein Auszug eines solchen Protokolls dargestellt.

```
File : xxxxxx                                    Seite : 1

 Adr          Werte
0000 00 01 FF 40 41 ..... 20 23 23 04 05 FE F7  ...@ASCII ##....
0010 12 0A 0D 30 31 ..... 35 36 37 38 39 40 41  ...012 3456789@A
0020 .  .  .
```

Bild 4.9: Ausgabe auf dem Drucker im Wide-Modus

Bisher wurden alle Eingaben interaktiv abgefragt. Um DUMP auch in Batchdateien verwenden zu können, besteht auch hier die Möglichkeit zur Aktivierung des Kommandomodus. Dann können Dateiname und eventuelle Optionen direkt in der Kommandozeile eingegeben werden. Der Aufruf besitzt folgende Syntax:

```
DUMP <Filename> <Optionen>
```

Der Dateiname hinter der Eingabe DUMP gilt dabei als Kriterium zur Selektion der Kommandooption. Fehlt der Filename, aber die Optionen sind vorhanden, dann bricht das Programm mit einer Fehlermeldung ab. Die Optionen sind dagegen nicht zwingend vorgeschrieben.

In Anlehnung an die bisherigen Module und im Hinblick auf DOS 5.0 läßt sich mit:

```
DUMP /?
```

eine Online-Hilfe abfragen. Auf dem Bildschirm erscheint ein entsprechender Hilfstext.

Zusammenfassung der Eingaben und Optionen

Nachfolgend ist eine knappe Zusammenstellung der DUMP aller Eingabemöglichkeiten und Optionen aufgeführt.

File:•Abfrage eines gültigen MS-DOS Dateinamens. Pfadangaben sind möglich. Ist die Datei nicht vorhanden, erscheint die Fehlermeldung: Die Datei < > existiert nicht und das Programm bricht ab.

Options:•Abfrage der Optionen zur Ausgabesteuerung. Fehlen die Optionen, wird die Standardeinstellung (Normal-Modus) übernommen.

/M•Die Ausgabe wird im Normal- und Wide-Modus unterbrochen, falls der Bildschirm voll ist. Nach Quittierung der Meldung Weiter, bitte eine Taste betätigen ... wird der Ablauf fortgesetzt. Die /P-Option hebt die /M-Option auf.

/W•Ausgabe der ASCII-Darstellung in der Ausgabezeile hinter den Hexcode. Die Option läßt sich mit der /M- oder /P-Option kombinieren.

/P•Hiermit wird die Ausgabe auf den Drucker umgeleitet. Eine selektierte More-Option wird aufgehoben, während /P und /W kombinierbar sind. Jede Druckseite enthält im Kopf Dateiname und Seitennummer.

Die Optionen lassen sich in beliebiger Reihenfolge eingeben, solange als erster Parameter der Dateiname mit einem nachfolgenden Leerzeichen auftritt. Nachfolgend sind einige gültige Optionen angegeben:

```
/P /W
/M / W
/W/P
```

Damit soll die Beschreibung der Ein-/Ausgaben abgeschlossen werden.

Der Entwurf

Nachdem konkrete Vorstellungen über die Programmfunktionen bestehen, beginnen wir mit dem Entwurf. Um einen transparenten Aufbau zu erhalten, ist ein modularer Aufbau zu empfehlen. Die nachfolgende Abbildung gibt die Struktur der einzelnen Programmodule wieder.

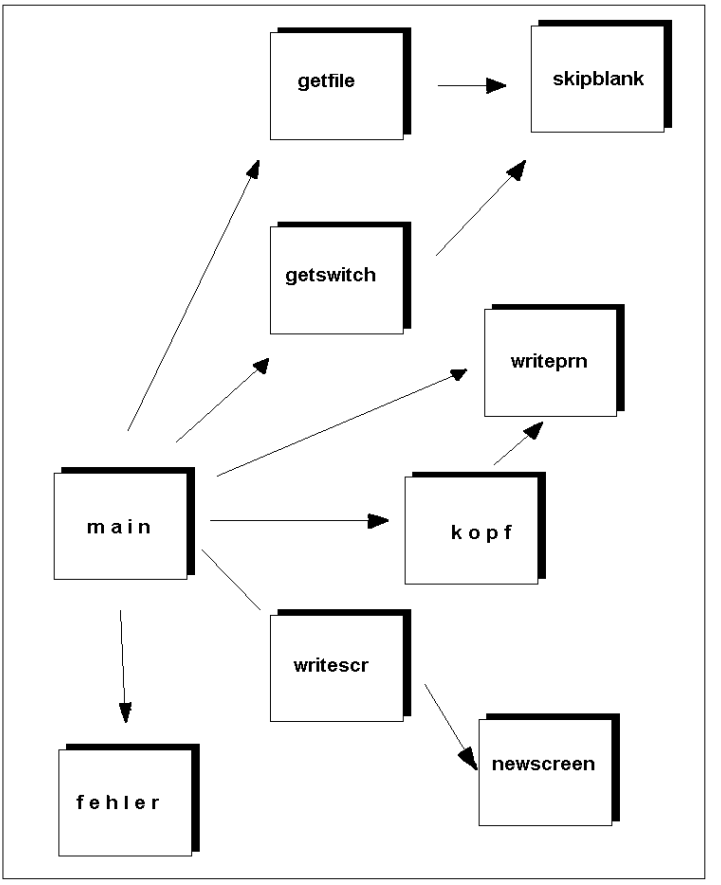


Bild 4.10: Modulhierarchie in DUMP.BAS

Da die Struktur nicht allzu komplex ist, kann auf die Implementierung eingegangen werden.

Die Implementierung

Durch den modularen Aufbau lassen sich bereits bestehende Module übernehmen und die grobe Struktur gleicht den bereits realisierten Programmen. Es soll aber noch kurz auf die Frage eingegangen werden, wie sich in PowerBASIC beliebige Dateien lesen lassen. Über die normale OPEN/INPUT-Sequenz treten Probleme auf, da PowerBASIC die Datei dann im ASCII-Modus interpretiert. Der Code 1AH wird in diesem Fall als Endekennzeichen verwendet. Falls ein solcher Wert in einer Binärdatei auftritt, bricht die INPUT-Funktion an dieser Stelle ab und meldet das Dateiende. Damit lassen sich die nachfolgenden Daten nicht mehr lesen. Da dies unerwünscht ist, muß die Datei im BINARY-Modus geöffnet werden. Dann lassen sich alle Bytes mit GET lesen. Allerdings ist noch zu klären, wie sich einzelne Bytes in PowerBASIC einlesen lassen. Der Befehl:

```
GET #ein%, 1, code%
```

liest ja jeweils einen 2-Byte-Integerwert ein. Aber nicht alle Dateien enthalten eine gerade Anzahl von Bytes, womit beim letzten Byte Probleme auftreten. Aber mit einem Trick klappt es doch! Wird jeweils ein Zeichen in eine ASCII-Variable gelesen:

```
GET #ein%, 1, zchn$
```

und anschließend in einen Hexwert konvertiert, dann ist das Problem gelöst.

Das Hauptprogramm

Nach der Variablendeklaration und dem Einlesen der Parameter (wahlweise im Interaktiv- oder Kommandomodus) beginnt die Ausgabe. Hierzu wird die Datei byteweise gelesen und die Werte in Blöcken zu 16 Bytes ausgegeben. Das Feld *code%()* dient zur Aufnahme der gelesenen Werte. Die Variable *adr* gibt die Anfangsadresse (Offset vom Dateianfang) des ersten Byte eines Blockes an. Die Variablen *more%*, *druck%* und *wide%* dienen zur Einstellung der jeweiligen Optionen. In *maxzeile%* findet sich die maximale Zeilenzahl pro Seite für die Bildschirm- und Druckerausgabe.

Die Decodierung der Optionen erfolgt mit dem Aufruf *getswitch*. *kopf* gibt einen Seitenkopf aus, während die Unterprogramme *writescr* und *writeprn* zur Ausgabe auf Bildschirm und Drucker zuständig sind. Sobald 16 Werte gelesen wurden, sind diese Module aufzurufen. Nach erreichen des Dateiendes wird die WHILE-Schleife verlassen und eventuell vorliegende Werte sind noch auszugeben, bevor die Datei geschlossen und das Programm beendet wird. Nachfolgendes Strichdiagramm (Bild 4.11) beschreibt den Programmablauf.

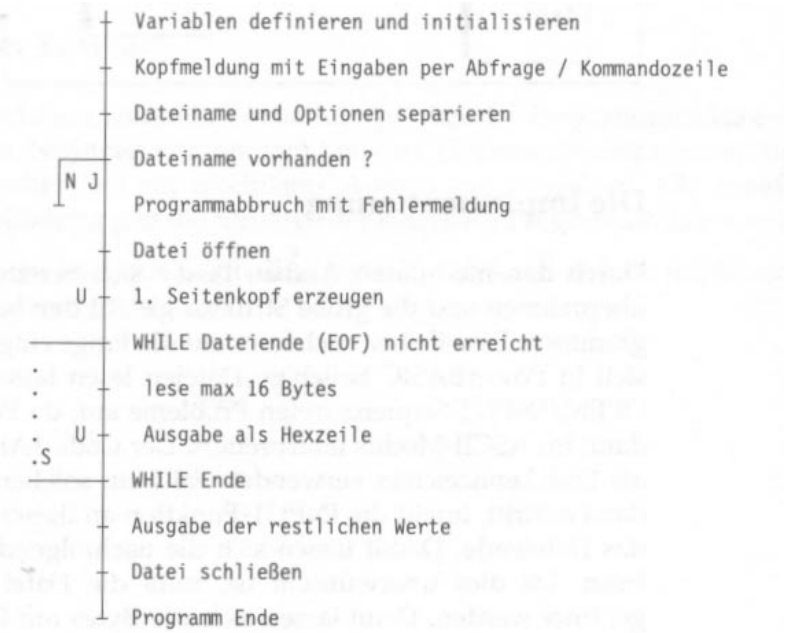


Bild 4.11: Programmablauf im Hauptmodul (main)

Die Funktionen zur Decodierung der Optionen, Ausgabe der Zeilen etc. werden als Unterprogramme realisiert.

getswitch

Nach dem Programmstart sind eventuelle Optionen zu decodieren. Dies ist Aufgabe von *getswitch*. Die Analyse erfolgt dadurch, daß der String *option$* mit Hilfe der INSTR-Funktion durchsucht wird. Falls eine Option vorkommt, gibt *ptr%* die Position im String wieder. Dann ist der Schalter zu setzen, andernfalls bleibt die Standardeinstellung erhalten. Drei Abfragen reichen zur Decodierung der Optionen.

newscreen

Bei der More-Option ist die Ausgabe zu unterbrechen, sobald der untere Bildschirmrand erreicht wird. Dann muß die folgende Benutzermeldung erscheinen:

```
Weiter, bitte ein Taste betätigen ...
```

Diese Aufgabe fällt dem Modul *newscreen* zu. Falls der Wert der Variablen *scrline%* die maximale Zeilenzahl pro Bilschirmseite (*%maxscr*) übersteigt, erscheint diese Meldung. Erst nach Betätigung einer Taste löscht der Befehl CLS den Bildschirm und gibt die weitere Bearbeitung frei.

kopf

Das Modul gibt den Seitenkopf bei jeder neuen Bildschirm- oder Druckerseite aus. Die Variable *maxzeile%* enthält den Wert, ab dem ein

Seitenwechsel erforderlich ist. Dieser Grenzwert wird je nach Option dynamisch (Bildschirm, Drucker) angepaßt.

writescr und writeptr

Diese Module übernehmen die Ausgabe auf dem Bildschirm (*writescr*) und dem Drucker (*writeprn*). In jeder Zeile steht am Anfang eine fortlaufende Adresse (fünf Hexziffern). Das Strichdiagramm in Bild 4.12 gibt den Ablauf wieder.

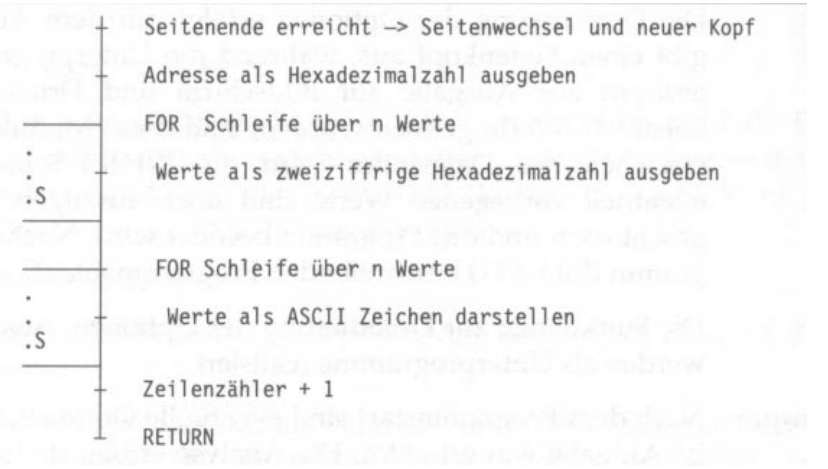


Bild 4.12: Ablauf in writescr und writeprn

Damit wird die Beschreibung der Programme beendet. Einzelheiten sind dem Listing zu entnehmen.

Erweiterungsvorschläge

Die Ausgabe der Werte erfolgt zur Zeit in Hexadezimalnotation. Es wäre möglich, die Ausgaben wahlweise im Dezimal- oder Hexadezimalsystem vorzusehen.

```
X R E F   /Z=50                              (c) Born Version 1.0
Datei : dump.bas      Datum : 05-31-1992          Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : DUMP.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 7.5.92
      '! Autor      : G. Born
      '! Files      : INPUT, OUTPUT, FILE
      '! Progr. Spr.: POWER Basic
      '! Betr. Sys. : DOS 2.1 - 5.0
      '! Funktion: Das Programm gibt den Inhalt einer Datei in Form
      '!           von Hexzahlen auf den Screen oder Printer aus.
```

```
      '! Ausgabe : Adr    16 Bytes
      '!           0000 xx xx xx .....   xx xx xx <- Hex Werte
      '!                a  a  a .....    a  a  a <- ASCII Werte
      '!           Nicht darstellbare Codes werden in der ASCII
      '!           Anzeige durch einen "." markiert. Mit der Option
      '!           /P erfolgt die Ausgabe auf dem Drucker. Mit /W
      '!           wird der Wide Mode aktiviert, welcher die
      '!           ASCII Zeichen hinter die Hexausgabe
positioniert.
      '!           Mit /M wird die More Option angewählt, die die
      '!           Ausgabe anhält, sobald der Screen voll ist.
      '!
      '! Aufruf:   DUMP                              interaktiver
Mode
      '!           DUMP <datei> <options>           Kommando Mode
      '************************************************************
      '! Variable definieren
    1 %true = &HFFFF: %false = 0            '! Konstante
    2 ein% = 1                              '! Kanal für I/O
    3 options$ = ""                         '! Optionen
    4 more% = %false                        '! More Option aus
    5 druck% = %false                       '! keine Printerausgabe
    6 wide% = %false                        '! Normalmode

    7 DIM code%(0:15)                       '! Puffer für 16 Bytes
    8 adr& = 0                              '! Anfangsadresse Zeile
    9 ptr% = 0

   10 %maxscr = 19                          '! Zeilenzahl pro
Screen
   11 %maxprt = 60                          '! Zeilenzahl pro
Druckseite
   12 maxzeile% = %maxscr                   '! Zeilenzahl für
Screen
   13 zeile% = 2                            '! Zeilennummer der
Seite
   14 seite% = 1                            '! Seitennummer
   15 spacex% = 1                           '! Zwischenraum ASCII
   16 kommando$ = ""
   17 filename$ = ""                        '! Dateiname
   18 hilf% = 0

   19 ON ERROR GOTO fehler

      '############################################################
      '#                       Hauptprogramm                      #
      '############################################################

   20 kommando$ = COMMAND$                  '! Parameter ?
   21 IF LEN (kommando$) = 0 THEN           '! Interaktiv Mode ?
   22  CLS                                  '! ja -> Clear Screen
      '! #####   Kopf ausgeben  ######
   23  PRINT "D U M P                          (c) Born Version
1.0"
```

```
   24  PRINT "Options [/Wide /More /Print
]"
   25  PRINT
   26  INPUT "File    : ",filename$        '! lese Dateiname
   27  INPUT "Options : ",options$         '! lese Optionen
   28  PRINT
   29 ELSE                                 '! Kommando Mode
   30  ptr% = INSTR (kommando$,"/?")       '! Option /?
   31  IF ptr% <> 0 THEN                   '! Hilfsbildschirm
   32   PRINT "D U M P                          (c) Born Version
1.0"
   33   PRINT
   34   PRINT "Aufruf:  DUMP <Filename> <Optionen>
   35   PRINT
   36   PRINT "Das Programm gibt den Inhalt einer Binärdatei als
Hex-"
   37   PRINT "Dump auf dem Bildschirm oder Drucker aus. Optionen:"
   38   PRINT
   39   PRINT "/W  WIDE-Mode mit Hexzahlen und Text in einer Zeile"
   40   PRINT "/M  More-Mode, seitenweise Ausgabe am Bildschirm"
   41   PRINT "/P  Ausgabe auf den Drucker"
   42   PRINT
   43   SYSTEM
   44  END IF
      '!
      '! getfile separiert den Dateinamen aus der Kommandozeile
      '! Falls ein Name fehlt, würden die Optionen in die jeweilige
      '! Variable gespeichert. Dies ist abgefangen, da Optionen mit
      '! /.. beginnen. Dann wird ein Leerstring zurückgegeben
      '!
   45  kommando$ = UCASE$(kommando$) + " "  '! Blank als
Endeseparator
   46  ptr% = 1                            '! Parameter holen
   47  CALL getfile(ptr%, kommando$,filename$) '! Name Eingabedatei
   48  INCR ptr%                           '! Anfang next token
   49  hilf% = INSTR(kommando$,"/")        '! suche Optionen
   50  IF hilf% >= ptr% THEN               '! gefunden ?
   51   options$ = MID$(kommando$,hilf%)   '! Reststring mit
Optionen
   52  END IF
   53 END IF

   54 IF filename$ = "" THEN              '! Leereingabe ?
   55  PRINT "Der Name der Eingabedatei fehlt"
   56  END
   57 END IF

   58 OPEN filename$ FOR INPUT AS #ein%   '! Eingabedatei
vorhanden?
   59 CLOSE                              '! Ja -> öffne als
Binärdatei

   60 OPEN filename$ FOR BINARY AS #ein%  '! öffne Eingabedatei

   61 IF LEN(options$) > 0 THEN
```

```
   62  GOSUB getswitch                      '! lese Optionen
   63 END IF

   64 CALL kopf                            '! 1. Seitenkopf ausgeben

      '!
      '! bearbeite Datei je nach Option
      '!
   65 IF druck% THEN
   66  PRINT "Ausgabe auf dem Drucker"
   67 END IF

   68 ptr% = 0                             '! auf 1. Eintrag
   69 WHILE NOT (EOF(ein%))                '! lese sequentiell
   70  GET$ #ein%, 1, zchn$                '! lese 1 Byte aus
Binärdatei
   71  code%(ptr%) = ASC(zchn$)            '! wandele in Hex
   72  IF ptr% > 14 THEN                   '! 16 Bytes gelesen ?
   73   IF druck% THEN
   74    CALL writeprn                     '! auf Printer ausgeben
   75   ELSE
   76    CALL writescr                     '! auf Screen ausgeben
   77   END IF
   78  ELSE
   79   INCR ptr%                          '! next entry
   80  END IF

   81 WEND

   82 IF druck% THEN                       '! Restdaten ausgeben
   83  CALL writeprn                       '! auf Printer ausgeben
   84 ELSE
   85  CALL writescr                       '! auf Screen ausgeben
   86 END IF

   87 CLOSE
   88 PRINT "Ende Dump"
   89 END

      '############################################################
      '#                    Hilfsroutinen                         #
      '############################################################

   90 fehler:
      '-----------------------------------------------------------
      '! Fehlerbehandlung in XREF
      '-----------------------------------------------------------

   91 IF ERR = 53 THEN
   92  PRINT "Die Datei ";filename$;" existiert nicht"
   93 ELSE
   94  PRINT "Fehler : ";ERR;" unbekannt"
   95  PRINT "Programmabbruch"
   96 END IF
   97 END                                  '! MSDOS Exit
```

```
   98 RETURN

   99 getswitch:
      '-------------------------------------------------------
      '! decodiere eingegebene Optionen
      '! option$ ist der String mit den Optionen
      '-------------------------------------------------------

  100  options$ = UCASE$(options$)

  101  IF INSTR(options$,"/M") > 0 THEN
  102   more% = %true                          '! More  Mode ein
  103  END IF

  104  IF INSTR(options$,"/W") > 0 THEN        '! wide Option ?
  105   wide% = %true                          '! Wide ein
  106   spacex% = 0                            '! kein Zwischenraum
  107 END IF

  108  IF INSTR(options$,"/P") > 0 THEN        '! Printer Option ?
  109   druck% = %true                         '! ja -> setze Mode
  110   more% = %false                         '! More ausschalten
  111   maxzeile% = %maxprt                    '! Zeilenzahl für
Printer
  112  END IF

  113 RETURN


  114 SUB getfile(ptr%,text$,result$)
      '!-------------------------------------------------------
      '! separiere Filename aus Eingabetext (text$)
      '! ptr% -> Anfang Filename, result$ = Filename
      '!-------------------------------------------------------
  115 LOCAL tmp%

  116 CALL skipblank (ptr%,text$)             '! entferne
Leerzeichen
  117 tmp% = INSTR(ptr%,text$," ")            '! suche Separator
  118 IF tmp% = 0 THEN
  119  PRINT "Fehler: kein Fileseparator"     '! kein Endeseparator
  120  END                                    '! Exit
  121 END IF
  122 IF MID$(text$,ptr%,1) = "/" THEN        '! Optionen eingegeben
?
  123  result$ = ""                           '! Leerstring
  124 ELSE
  125  result$ = MID$(text$,ptr%,tmp%-ptr%)   '! Filename
  126  ptr% = tmp%                            '! korrigiere ptr%
  127 END IF

  128 END SUB


  129 SUB newscreen
```

```
      '-------------------------------------------------------
      '! More Abfrage bei vollem Bildschirm
      '-------------------------------------------------------
  130 SHARED zeile%

  131 IF zeile% > maxzeile% THEN
  132  PRINT
  133  PRINT "Weiter, bitte eine Taste betätigen ..."
  134  WHILE LEN(INKEY$) = 0                '! warte auf Taste
  135  WEND
  136  CALL kopf                             '! Seitenkopf
  137 END IF
  138 END SUB


  139 SUB kopf
      '-------------------------------------------------------
      '! Seitenvorschub und Ausgabe des Seitenkopfes auf Screen
oder
      '! Drucker. Bei der Druckerausgabe werden Filename und
Seiten-
      '! nummer mit ausgegeben.
      '-------------------------------------------------------
  140 SHARED seite%, filename$, zeile%, druck%

  141  IF druck% THEN
  142   IF seite% > 1 THEN LPRINT CHR$(12); '! Seitenvorschub
  143   LPRINT "File : "; filename$,
  144   LPRINT SPACE$(15);"Seite : "; seite%
  145   LPRINT " Adr               Werte"   '! auf Screen
  146   LPRINT
  147   INCR seite%
  148   zeile% = 3
  149  ELSE
  150   CLS
  151   PRINT " Adr               Werte"     '! auf Screen
  152   PRINT
  153   zeile% = 2
  154  END IF

  155 END SUB

  156 SUB writescr
      '-------------------------------------------------------
      '! Ausgabe der 16 gelesenen Werte im HEX und ASCII Format
      '! auf dem Bildschirm
      '! code (16)         16 Bytes aus der Datei
      '-------------------------------------------------------
  157 LOCAL res$, i%, zchn$
  158 SHARED zeile%, maxzeile%, code%(), adr&, ptr%
  159 SHARED spacex%, more%, wide%

  160  IF (zeile% > maxzeile%) AND more% THEN
  161   CALL newscreen                       '! Bildwechsel
  162  END IF
```

```
163  res$ = HEX$(adr&)                      '! Hexausgabe mit
164  PRINT STRING$(5-LEN(res$),"0");res$;" "; '! führend. Nullen
165  FOR i% = 0 TO ptr%                     '! Codes ausgeben
166   res$ = HEX$(code%(i%) AND &HFF)       '! Hexausgabe mit
167   PRINT STRING$(2-LEN(res$),"0");res$;" "; '! führend. Nullen
168  NEXT i%

169  IF NOT wide% THEN
170   PRINT: PRINT "     ";                 '! keine Wide Option
171  ELSE
172   IF ptr% < 15 THEN                     '! Zeile nicht voll !!
173    PRINT SPACE$((15 - ptr%) * 3);       '! Vorschub n Zeichen
174   END IF
175   PRINT " ";                            '! Leerzeichen
176  END IF

177  FOR i% = 0 TO ptr%                     '! ASCII ausgeben
178   IF code%(i%) > &H1F THEN
179    zchn$ = CHR$(code%(i%))              '! ASCII Wert
180   ELSE
181    zchn$ = "."                          '! nicht darstellbar
182   END IF
183   PRINT SPACE$(spacex%);zchn$; _        '! ASCII Darstellung
184         SPACE$(spacex%);                '! ausgeben
185  NEXT i%
186  PRINT
187  INCR adr&, 16                          '! Adresse + 16
188  IF wide% THEN
189   INCR zeile%                           '! wide -> Zeile + 1
190  ELSE
191   INCR zeile%, 2                        '! Zeile + 2
192  END IF
193  ptr% = 0                               '! reset ptr%

194 END SUB


195 SUB writeprn
    '!-------------------------------------------------------
    '! Ausgabe der Werte auf dem Drucker
    '!-------------------------------------------------------
196 LOCAL res$, i%, zchn$
197 SHARED zeile%, maxzeile%, code%(), adr&, ptr%
198 SHARED spacex%, wide%

199  IF zeile% > maxzeile% THEN
200   CALL kopf                             '! Seitenwechsel
201  END IF

202  res$ = HEX$(adr&)                      '! Hexausgabe mit
203  LPRINT STRING$(5-LEN(res$),"0");res$;" "; '! führend. Nullen
204  FOR i% = 0 TO ptr%                     '! Codes ausgeben
205   res$ = HEX$(code%(i%) AND &HFF)       '! Hexausgabe mit
```

```
  206   LPRINT STRING$(2-LEN(res$),"0");res$;" "; '! führend.
Nullen
  207  NEXT i%

  208  IF NOT wide% THEN
  209   LPRINT: LPRINT "     ";             '! keine Wide Option
  210  ELSE
  211   IF ptr% < 15 THEN                   '! Zeile nicht voll !!
  212    LPRINT SPACE$((15 - ptr%) * 3);    '! Vorschub n Zeichen
  213   END IF
  214   LPRINT " ";                         '! Leerzeichen
  215  END IF

  216  FOR i% = 0 TO ptr%                   '! ASCII ausgeben
  217   IF code%(i%) > &H1F THEN
  218    zchn$ = CHR$(code%(i%))             '! ASCII Wert
  219   ELSE
  220    zchn$ = "."                        '! nicht darstellbar
  221   END IF
  222   LPRINT SPACE$(spacex%);zchn$; _     '! ASCII Darstellung
  223         SPACE$(spacex%);              '! ausgeben
  224  NEXT i%
  225  LPRINT
  226  INCR adr&, 16                        '! Adresse + 16
  227  IF wide% THEN
  228   INCR zeile%                         '! Zeile + 1 -> wide%
  229  ELSE
  230   INCR zeile%, 2                      '! Zeile + 2
  231  END IF
  232  ptr% = 0                             '! reset ptr%

  233 END SUB

  234 SUB skipblank(ptr%,text$)
      '!-----------------------------------------------------------
      '! entferne führende Leerzeichen aus text$
      '!-----------------------------------------------------------

  235 LOCAL lang%, zchn$

  236 lang% = LEN (text$)                    '! Stringlänge
  237 zchn$ = MID$(text$,ptr%,1)             '! separiere Zeichen

  238 WHILE (zchn$ = " ") AND (ptr% <= lang%) '! Zeichen <> blank
  239  INCR ptr%
  240  zchn$ = MID$(text$,ptr%,1)            '! separiere Zeichen
  241 WEND

  242 END SUB

      '***** Programm Ende ******
```

Listing 4.2: DUMP.BAS

FORMAT: Formatschutz für Festplatten

Das MS-DOS-Programm FORMAT.COM (oder FORMAT.EXE) dient zum Formatieren von Disketten und Festplatten. Werden beim Aufruf keine Parameter angegeben, bezieht sich FORMAT auf das Standardlaufwerk. Als Einstellparameter werden dann Standardwerte gesetzt. Im Prinzip ist gegen diese Aufrufkonvention nichts einzuwenden, da sie sich bei vielen MS-DOS-Programmaufrufen bestens bewährt. Leider hat dies, insbesondere bei älteren Versionen des Programmes FORMAT, fatale Konsequenzen. Ist im System eine Festplatte vorhanden, befinden sich üblicherweise das Betriebssystem und die entsprechenden Hilfsprogramme auf dieser Platte. Dies bedeutet, daß in der Regel die Platte (C:, D: etc.) als Standardlaufwerk eingestellt wird. Alle Eingaben ohne Laufwerksangabe beziehen sich nun auf diese Platte. Wird bei der Eingabe:

```
FORMAT
```

die Bezeichnung des Laufwerks vergessen, bezieht sich das Kommando auf die Festplatte. Besonders bei älteren Versionen vom FORMAT wird dann die Platte ohne Warnung formatiert. Auch bei einer einfachen J/N-Abfrage ist diese schnell quittiert. Wem dies einmal passiert ist, kennt die tragischen Folgen eines solchen Formataufrufes. Es stellt sich sofort die Frage nach den Sicherungskopien, die natürlich in diesem Augenblick fehlen. Auch wenn diese vorhanden sind, bleibt eine Menge Arbeit mit der Restaurierung der Festplatte.

Eine Möglichkeit besteht zwar darin, das Programm FORMAT nur auf Disketten zu halten. Damit läßt sich nur nach Einsatz einer Diskette mit dem Programm FORMAT arbeiten, ist aber in der Handhabung äußerst hinderlich. DOS 5.0 und verschiedene Utilities bieten zwar eine UNFORMAT-Anweisung. Aber auch dies ist mit Aufwand zur Restaurierung der Festplatte verbunden. Also ist ein Weg zu suchen, der zu einem besser abgesicherten FORMAT-Programm führt. Naheliegend ist sicherlich der Gedanke, FORMAT so zu verändern, daß eine irrtümliche Formatierung der Festplatte kaum mehr möglich ist. Dies setzt aber sehr genaue Kenntnisse des Programmcodes voraus. Als zweites Problem erweist sich die Tatsache, daß bei jeder neuen Version die Anpassung zu wiederholen ist. Dies ist in der Regel kein gangbarer Weg. Es stellt sich deshalb die Frage nach einer Alternativstrategie. Wie läßt sich das Programm FORMAT im Hinblick auf eine sichere Handhabung beeinflussen, ohne daß auf versionsspezifische Eigenarten Rücksicht genommen werden muß?

Mit etwas Nachdenken stößt man auf eine einfache und dennoch elegante Lösung. MS-DOS bietet die Möglichkeit der Tochterprozesse, d.h. ein laufendes Programm kann ein weiteres Programm laden und aktivieren. Das aktuelle Programm wird dabei zwar suspendiert, aber nach Beendigung des neuen Programmes erhält es die Kontrolle zurück. Diese Vorgehensweise läßt sich auch für obiges Problem verwenden. Zuerst wird ein Programm aufgerufen, welches die Eingaben überprüft. Nur wenn das

Aufrufkommando korrekt ist, wird das eigentliche FORMAT-Programm aktiviert. Ein solches Prüfprogramm läßt sich natürlich in PowerBASIC leicht erstellen. Bleibt nur noch die Frage offen, wie der Aufruf erfolgen soll? Die Eingabe:

```
TEST FORMAT  A:  /S
```

ist sicherlich nicht zumutbar, da der normale DOS-Benutzer nichts von der Existenz des Programmes TEST weiß. Er wird sich also nach wie vor mit der Eingabe FORMAT begnügen und damit seine Platte ruinieren. Aber niemand sagt, daß das MS-DOS Programm FORMAT nicht einen anderen Dateinamen erhalten kann. Es besteht doch die Möglichkeit, die Datei FORMAT.COM oder FORMAT.EXE mit einem neuen Namen zu belegen. Das PowerBASIC- Prüfprogramm erhält dann den Namen FORMAT.

Damit bleibt für den Benutzer die gewohnte Aufruffolge erhalten und er merkt im Grunde nichts davon, daß ein anderes Programm zwischengeschaltet wurde. Nur wenn sich ein Aufruf auf die Festplatte bezieht, erscheint eine deutliche Warnung.

Vor der Implementierung soll nun noch kurz die genaue Benutzerschnittstelle festgelegt werden. Das Original-MS-DOS-Programm FORMAT wird in FORMATX umbenannt (mit RENAME ist dies kein Problem). Das PowerBASIC-Programm erhält dann den Namen FORMAT.BAS. Damit aktiviert jede Eingabe der Form:

```
FORMAT <Laufwerk> <Optionen>
```

das BASIC-Programm. Die Eingabe der Optionen kann entfallen. Wird aber auch der Laufwerksname weggelassen, erscheint die Meldung:

```
### FORMAT Aufruf ohne Laufwerksangabe unzulässig ###
```

und das Programm bricht ab. Wird als Laufwerk die Bezeichnung A: oder B: eingetragen, ist das Programm FORMATX zu aktivieren. Dieser Aufruf wird dem Benutzer aber nicht angezeigt. Auf dem Bildschirm erscheinen lediglich die gewohnten Meldungen des DOS-FORMAT-Programmes. Tritt die Bezeichnung einer Festplatte im Kommando auf:

```
 FORMAT C: /V /S
```

dann erscheint eine deutliche Warnung an den Benutzer (Bild4.13):

```
########################################################
# Wollen Sie wirklich Ihre Festplatte formatieren?   #
# Bitte, geben Sie dann den Code 69 ein, sonst eine  #
# beliebige Taste betätigen....                      #
########################################################

Code :
```

Bild 4.13: Benutzerwarnung beim Aufruf von FORMAT

Nur wenn der Benutzer die Zahl 69 eingibt, läßt sich die Festplatte formatieren. Die eingegebenen Zeichen erscheinen dabei nicht auf dem Bildschirm. Wird obiger Text so verändert, daß die Codenummer nicht mehr angezeigt wird, ist die Nummer mit einem Paßwort zu vergleichen. Nur bei Eingabe des korrekten Paßwortes läßt sich die Platte formatieren. Es besteht zwar auch die Möglichkeit, die Zeichen J/N abzufragen. Da solche Abfragen aber bei mehreren anderen Programmen auftreten, ist die Gefahr einer irrtümlichen Quittierung zu hoch. Die Abfrage der Zahl 69 stellt hier eine zusätzliche Sicherheit dar.

Da unser PowerBASIC-Programm das Eingabekommando nur auf gültige Laufwerksbezeichnungen prüft, können im Optionsfeld beliebige Zeichen auftreten. Falsche Optionen werden anschließend vom DOS FORMAT-Programm erkannt und abgewiesen.

Damit wird die Beschreibung der Benutzeroberfläche sowie die Diskussion der prinzipiellen Vorgehensweise abgeschlossen.

Die Implementierung

Die Implementierung des Programmes ist recht einfach, so daß auf ein Hierarchiediagramm an dieser Stelle verzichtet wird. Das Programm erhält den Dateinamen FORMAT.BAS und besteht nur aus dem Hauptmodul.

Es sind nur Eingaben über die Kommandozeile erlaubt. Diese wird mittels der Funktion COMMAND$ in die Stringvariable *kommando$* eingelesen. Ist die Länge = 0, fehlen Laufwerks- und Optionseingaben. In diesem Fall bricht das Programm mit einer Fehlermeldung ab.

Nun gibt es aber noch den Fall, daß zwar die Laufwerksbezeichnung fehlt, aber Optionen eingegeben wurden. Hier muß eine erweiterte Prüfung erfolgen, da die Stringlänge größer Null ist. Es gibt nun einen einfachen Weg zur Überprüfung auf gültige Laufwerksnamen. Alle Laufwerkskennzeichnungen sind mit einem Doppelpunkt abzuschließen (A:, B:, a:, b: etc.). In vorliegender Implementierung wird nun überprüft, ob ein solcher Doppelpunkt im Kommandostring auftritt. Fehlt dieses Zeichen, liegt keine gültige Laufwerksbezeichnung vor und das Programm ist zu beenden.

Damit ist die Abbruchbedingung bei fehlender Laufwerksbezeichnung hinreichend besprochen. Es bleibt nur noch der Fall, daß als Laufwerk eine Festplatte angegeben wird. Wie werden nun diese Eingaben erkannt. Zuerst nehmen wir an, daß bekannt ist, welche Platten im System vorhanden sind (C:, D: etc.). Damit läßt sich das Programm beliebig an das System anpassen. In vorliegender Implementierung wird deshalb nur das Laufwerk C: überprüft. Hierzu ist der String in der Variablen *kommando$* in Großbuchstaben zu konvertieren (UCASE$). Dann wird mit der Funktion INSTR geprüft, ob die Zeichenkombination (C:) in der Kommandozeile vorkommt. Ist dies nicht der Fall, darf das in FORMATX.COM umbenannte DOS-FORMAT-Programm mit den eingegeben Optionen aufgerufen werden. Hierzu ist der Eingabetext mit

dem String »FORMATX« zu verknüpfen. Dies ist erforderlich, da die PowerBASIC-Funktion COMMAND$ nur die Eingabeparameter ohne den Namen des aktiven Programmes liefeert. Dann wird as MS-DOS-Formatprogramm über die Basic-Funktion SHELL aufgerufen. Die Einzelheiten sind dem nachfolgenden Listing zu entnehmen.

Erweitungsvorschläge

Das Programm gibt vor der Formatierung einer Festplatte den Hinweis daß sich diese durch Eingabe des Codes »69« formatieren lässt. Einmal kann natürlich die Eingabe J/N geändert werden. Dadurch steigt aber auch die Gefahr von Fehleingaben.

Wer die Festplatte noch sicherer vor ungewollter Formatierung schützen möchte, kann innerhalb der Warnung ein Kennwort abfragen. Der Codewert darf im Text dann nicht mehr vorkommen. Wird das ursprüngliche DOS-Programm FORMAT.COM zusätzlich noch mit einem unverfänglichen Namen versehen und gegebenenfalls in einer »Hidden«-Datei versteckt, kann nur ein Insider diesen Schutz umgehen.

```
X R E F   /Z=50                              (c) Born Version 1.0
Datei : format.bas      Datum : 05-31-1992          Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : FORMAT.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 16. 5.92
      '! Autor      : G. Born
      '! Files      : INPUT, OUTPUT
      '! Progr. Spr.: Power Basic
      '! Betr. Sys. : DOS 2.1 - 5.0
      '! Funktion: Das Programm wird mit der Eingabe:
      '!
      '!           FORMAT <Laufwerk> <Optionen>
      '!
      '!           aufgerufen. Es überprüft dann, ob als Lauf-
      '!           werk ein Festplattenname eingegeben wurde.
      '!           In diesem Fall erfolgt eine Warnung mit Ab-
      '!           frage, ob die Festplatte formatiert werden
      '!           soll. Das Programm aktiviert nach einer
      '!           Quittierung, oder bei Floppylaufwerken an-
      '!           schließend das DOS FORMAT Programm, welches
      '!           in FORMATX umbenannt wurde.
      '*********************************************************
      '! Variable definieren
    1 zchn$ = ""                            '! Hilfsvariable
    2 kommando$ = ""                        '! Kommandozeile
    3 ptr% = 0                              '! Hilfzeiger

      '#########################################################
      '#                     Hauptprogramm                     #
```

```
      '############################################################

    4 kommando$ = UCASE$(COMMAND$)          '! Parameter ?
    5 IF LEN (kommando$) = 0 THEN           '! Laufwerksname ?
    6  PRINT "### Formataufruf ohne Parameter unzulässig       ###"
    7  END                                  '! Exit
    8 END IF

    9 ptr% = INSTR(kommando$,":")           '! suche Laufwerk
   10 IF ptr% = 0 THEN                      '! Laufwerksname ?
   11  PRINT "### Formataufruf ohne Laufwerksangabe unzulässig ###"
   12  END                                  '! Exit
   13 END IF

      '!
      '!-----------------------------------------------------------
      '! prüfe ob als Laufwerk eine Festplatte angegeben wurde ?
      '!-----------------------------------------------------------
      '!

   14 IF INSTR(kommando$,"C:") > 0 THEN
   15  CLS
   16  PRINT
   17  PRINT
"#########################################################"
   18  PRINT "# Wollen Sie wirklich Ihre Festplatte formatieren ?
#"
   19  PRINT "# In diesem Fall geben Sie den Code 69 ein, sonst
#"
   20  PRINT "# bitte eine beliebige Taste betätigen ...
#"
   21  PRINT
"#########################################################"
   22  PRINT
   23  PRINT "Code : ";
   24  antw$ = INPUT$ (1)                   '! lese 1. Zeichen
   25  IF antw$ <> "6" THEN END             '! Exit

   26  antw$ = INPUT$ (1)                   '! lese 2. Zeichen
   27  IF antw$ <> "9" THEN END             '! Exit
   28 END IF
      '!
      '! Die Eingabe bezieht sich auf eine Floppy, oder der
Benutzer
      '! möchte die Festplatte formatieren -> starte FORMATX
      '!
   29 SHELL "FORMATX " + kommando$          '! execute Format
   30 END                                   '! Ende

      '***** Programm Ende ******
```

Listing 4.3: FORMAT.BAS

DELX: Physikalisches Löschen von Dateien

Jeder MS-DOS Anwender kennt die Kommandos DEL und ERASE, mit denen sich Dateien von Platten und Floppys löschen lassen. Der normale Benutzer geht dann davon aus, daß die Daten damit gelöscht sind. Aus Zeitgründen beschränkt sich DOS aber darauf, nur den ersten Buchstaben des Dateinamens im Inhaltsverzeichnis auf den Wert E5H zu setzen. Dies ist der vereinbarte Code für unbenutzte Dateieinträge. Insider haben bereits vor Jahren herausgefunden, daß die Daten weiterhin auf der Platte/Floppy erhalten bleiben. Wird der Eintrag im Inhaltsverzeichnis auf ein gültiges ASCII-Zeichen zurückgesetzt, läßt sich die Originaldatei restaurieren. Auf dieser Erkenntnis beruhen die angebotenen Programme zur Restaurierung versehentlich gelöschter Dateien (z.B. UNDELETE bei DOS 5.0). So schön dies bei unabsichtlich gelöschten Dateien ist, um so problematischer wirkt sich dies im Hinblick auf den Datenschutz aus. Mit den normalen DOS-Kommandos ist es unmöglich, eine Datei wirklich zu löschen. Mit Hilfe bestimmter Tools lassen sich die ursprünglichen Daten wieder restaurieren. Bei sensitivem Material ist dies natürlich unerwünscht.

Nachfolgend wird deshalb ein Programm entwickelt, welches eine Datei physikalisch von der Platte/Floppy löscht. Doch zuerst zu den Anforderungen aus Benutzersicht.

Einmal soll die Datei so gelöscht werden, daß eine Restaurierung nicht mehr möglich ist. Das Programm DELX bietet wieder zwei Möglichkeiten zur Eingabe der Dateinamen. Mit dem Kommando:

```
DELX
```

wird der interaktive Eingabemodus selektiert. Auf dem Bildschirm erscheint die Meldung:

```
D E L X                          (c) Born Version 1.0
Löschen einer Datei

Datei :
```

Bild 4.14: Meldung von DELX

Das Programm erwartet dann den Namen der zu löschenden Datei. Es sind die gültigen MS-DOS-Konventionen zu beachten. Bei Leereingaben bricht DELX mit der folgenden Meldung ab:

```
Der Name der Datei fehlt
```

Falls die Datei nicht existiert, erscheint die Nachricht:

```
Die Datei <Name> existiert nicht
```

und das Programm wird ebenfalls beendet. Mit <*Name*> wird der eingegebene Dateiname bezeichnet.

Alternativ besteht die Möglichkeit, den Dateinamen bereits in der Kommandozeile mit anzugeben. Es gilt dabei folgende Syntax:

```
DELX <Dateiname>
```

Im Kommandomodus erscheint die Kopfmeldung nicht mehr. Ansonsten gelten die gleichen Fehlermeldungen wie bei der interaktiven Eingabe. Mit der Eingabe:

```
DELX /?
```

läßt sich noch die Online-Hilfe aktivieren.

Falls die Datei existiert, beginnt das Löschen der Datei. Dies wird durch die Meldung:

```
Die Datei <Name> wird gelöscht
Die Datei <Name> ist gelöscht
```

kommentiert. Der Vorgang dauert, insbesondere bei längeren Dateien, etwas länger als das entsprechende DOS-Kommando. Anschließend sind Datei und Daten gelöscht.

Die Implementierung

Das Programm ist so einfach, daß auf eine Diskussion des Entwurfs verzichtet werden kann. Das Hauptprogramm übernimmt wieder die Aufgabe, den Dateinamen interaktiv oder aus der Kommandozeile einzulesen. Dann wird die Datei im Binary-Modus eröffnet. Dies ist erforderlich, da auf die Datei wahlfrei zugegriffen werden muß. Da DOS nur den Eintrag im Inhaltsverzeichnis aber nicht die Daten löscht, muß dies von DELX übernommen werden. Hierzu dient die WHILE/WEND-Schleife. Im Prinzip ist jedes Byte der Datei mit dem Wert FFH zu überschreiben. Dann läßt sich der ursprüngliche Wert nicht mehr rekonstruieren. Leider ist die Länge der Datei nicht bekannt. Um zu vermeiden, daß DOS die Datei auf andere Sektoren kopiert, muß die zu schreibende Datei die gleiche Länge besitzen wie die Eingabedatei. Deshalb liest DELX die Datei sequentiell und schreibt gleichzeitig die Werte FFH zurück. Bei Binary-Dateien ist dies möglich. Um einen schnelleren Ablauf zu erhalten, werden jeweils 512 Byte gelesen und beschrieben. Dies entspricht der Sektorgröße auf der Floppy/Platte im MS-DOS-Format. Lediglich der letzte Satz muß nicht immer 512 Byte besitzen. Deshalb wird die Zahl der gelesenen Zeichen registriert, um zum Abschluß die Restdaten zu schreiben. Nach dieser Prozedur kann die Datei mit dem Basic-Kommando KILL gelöscht werden. Einzelheiten sind dem nachfolgenden Listing zu entnehmen.

Erweiterungsvorschläge

Mit dieser Lösung ist es recht einfach, eine Datei mitsamt den Daten zu löschen. Allerdings sei darauf hingewiesen, daß das Programm keine

Daten außerhalb der Datei löscht. Beim normalen DOS-Betrieb werden jedoch Dateien kopiert, gelöscht und in ihrer Größe verändert. Dies führt dazu, daß auf dem Medium freie Sektoren vorhanden sind, die zur Aufnahme neuer Dateien dienen. In diesen Sektoren können aber durchaus Daten aus alten Dateien vorliegen. Mit DELX lassen sich diese Werte nicht überschreiben, da die Sektoren nicht zur angegeben Datei gehören. Mit einem Trick läßt sich aber das Programm DELX so erweitern, daß alle freien Sektoren einer Platte/Floppy überschrieben werden. Hierzu wird einfach ein temporäre Datei eröffnet und solange mit &HFF beschrieben, bis das Medium voll ist. Dann sind alle leeren Sektoren dieser Datei zugeordnet. Wird die Datei dann gelöscht, finden sich nur noch FFH-Werte in den freien Sektoren, und die alten Daten sind mit Sicherheit überschrieben.

```
X R E F   /Z=50                              (c) Born Version 1.0
Datei : delx.bas      Datum : 05-31-1992          Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : DELX.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 16.5.92
      '! Autor      : G. Born
      '! Files      : INPUT, OUTPUT
      '! Progr. Spr.: Power Basic
      '! Betr. Sys. : DOS 2.1 - 5.0
      '! Funktion: Das Programm löscht eine Datei pyhsikalisch,
d.h.
      '!           die Daten werden erst mit FFH überschrieben.
      '!
      '! Aufruf:   DELX                          '! Interaktiv Mode
      '!           DELX <datei1>                 '! Kommando Mode
      '!           DELX /?                       '! Online-Hilfe
      '*********************************************************
      '! Variable definieren

    1 ein% = 1                                  '! I/O Kanal
    2 ptr% = 0: hilf& = 0                       '! Hilfszeiger
    3 konst$ = ""

    4 ON ERROR GOTO fehler

      '############################################################
      '#                      Hauptprogramm                       #
      '############################################################

    5 kommando$ = COMMAND$                      '! Parameter ?
    6 IF LEN (kommando$) = 0 THEN               '! Interaktiv Mode ?
    7  CLS                                      '! ja -> Clear Screen
      '! #####   Kopf ausgeben  ######
    8  PRINT "D E L X                          (c) Born Version 1.0"
    9  PRINT "Löschen einer Datei"
   10  PRINT
```

```
   11  INPUT "Datei : ",infilename$  '! lese Dateiname Eingabe
   12  PRINT
   13 ELSE                                    '! Kommando Mode
   14  ptr% = INSTR (kommando$,"/?")          '! Option /?
   15  IF ptr% <> 0 THEN                      '! Hilfsbildschirm
   16   PRINT "D E L X                         (c) Born Version 1.0"
   17   PRINT
   18   PRINT "Aufruf:   DELX"
   19   PRINT "          DELX <datei>"
   20   PRINT
   21   PRINT "Das Programm löscht eine Datei, wobei deren Inhalt
über-"
   22   PRINT "schrieben wird."
   23   PRINT
   24   SYSTEM
   25  END IF
      '!
      '! getfile separiert den Dateinamen aus der Kommandozeile
      '!
   26  kommando$ = UCASE$(kommando$) + " "  '! Blank als
Endeseparator
   27  ptr% = 1                               '! Parameter holen
   28  CALL getfile(ptr%, kommando$,infilename$) '! Name
Eingabedatei
   29 END IF

   30 IF infilename$ = "" THEN               '! Leereingabe ?
   31  PRINT "Der Name der Datei fehlt"
   32  END
   33 END IF

   34 OPEN infilename$ FOR INPUT AS #ein%    '! Datei vorhanden ?
   35 CLOSE                                  '! ja-> open als
   36 OPEN infilename$ FOR BINARY AS #ein%   '! Binary Datei
   37 konst$ = STRING$(512,CHR$(&HFF))       '! auf FFH setzen

   38 PRINT "Die Datei ";infilename$;
   39 PRINT " wird gelöscht"

   40 WHILE NOT (EOF(ein%))
   41  GET$ #ein%, 512, zchn$                '! lese 512 Bytes
   42  SEEK #ein%, hilf&                     '! auf Anfang Satz
   43  IF LEN(zchn$) = 512 THEN              '! voller Satz ?
   44   PUT$ #ein% , konst$                  '! schreibe 512 * FFH
   45  ELSE
   46   PUT$ #ein%, STRING$(LEN(zchn$),CHR$(&HFF)) '! n Bytes
   47   EXIT LOOP                            '! EOF erreicht
   48  END IF
   49  hilf& = hilf& + LEN(zchn$)            '! Zeiger + n
   50 WEND

   51 CLOSE                                  '! Datei schließen

   52 outfile$ = LEFT$(infilename$,INSTR(infilename$,":"))
   53 outfile$ = outfile$ + "Born.G"       '! neuer Name
```

```
   54 NAME infilename$ AS outfile$          '! umbenennen

   55 KILL outfile$                         '! lösche Datei

   56 PRINT "Die Datei ";infilename$;
   57 PRINT " ist gelöscht"

   58 END

      '############################################################
      '#                      Hilfsroutinen                       #
      '############################################################

   59 fehler:
      '-----------------------------------------------------------
      '! Fehlerbehandlung in XREF
      '-----------------------------------------------------------

   60 IF ERR = 53 THEN
   61  PRINT "Die Datei ";infilename$;" existiert nicht"
   62 ELSE
   63  PRINT "Fehler : ";ERR;" unbekannt"
   64  PRINT "Programmabbruch"
   65 END IF
   66 END                                  '! MSDOS Exit
   67 RETURN

   68 SUB getfile(ptr%,text$,result$)
      '!----------------------------------------------------------
      '! separiere Filename aus Eingabetext (text$)
      '! ptr% -> Anfang Filename, result$ = Filename
      '!----------------------------------------------------------
   69 LOCAL tmp%

   70 CALL skipblank (ptr%,text$)           '! entferne
Leerzeichen
   71 tmp% = INSTR(ptr%,text$," ")          '! suche Separator
   72 IF tmp% = 0 THEN
   73  PRINT "Fehler: kein Fileseparator"   '! kein Endeseparator
   74  END                                  '! Exit
   75 END IF
   76 IF MID$(text$,ptr%,1) = "/" THEN      '! Optionen eingegeben
?
   77  result$ = ""                         '! Leerstring
   78 ELSE
   79  result$ = MID$(text$,ptr%,tmp%-ptr%) '! Filename
   80  ptr% = tmp%                          '! korrigiere ptr%
   81 END IF

   82 END SUB

   83 SUB skipblank(ptr%,text$)
      '!----------------------------------------------------------
      '! entferne führende Leerzeichen aus text$
```

```
   '!-------------------------------------------------------

84 LOCAL lang%, zchn$

85 lang% = LEN (text$)                      '! Stringlänge
86 zchn$ = MID$(text$,ptr%,1)               '! separiere Zeichen

87 WHILE (zchn$ = " ") AND (ptr% <= lang%) '! Zeichen <> blank
88  INCR ptr%
89  zchn$ = MID$(text$,ptr%,1)              '! separiere Zeichen
90 WEND

91 END SUB

   '***** Programm Ende ******
```

Listing 4.4: DELX.BAS

TEXTS: Textsuche in EXE-, SYS- und COM-Dateien

Manchmal ist es von Interesse EXE-, COM- oder SYS-Dateien auf Texte zu untersuchen. Dies ist zum Beispiel bei unbekannten Programmen erforderlich, wenn mögliche Fehler- oder Benutzermeldungen ermittelt werden sollen. Ein anderer Fall ist durch Viren in Computerprogrammen aktuell geworden. Bei unbekannten Programmen läßt sich teilweise an Hand der Meldungen auf die Funktion schließen. Damit besteht zumindest bei einigen Viren die Chance eine infizierte Programmdatei durch die Textstrings der Virusmeldungen im Programmcode identifizieren.

Wie dem auch immer sei, als normaler DOS-Benutzer ist eine Analyse des Programmcodes kaum durchführbar. Die Befehle COPY, PRINT oder TYPE produzieren nur »Schrott« bei Dateien mit Programmcode. Das bereits vorgestellte Programm DUMP schafft Abhilfe. Allerdings werden ASCII- und HEX-Zeichen gemeinsam dargestellt, dass eine weitere Auswertung erforderlich wird. Schöner wäre es wohl, wenn der Rechner die Analyse des Codes übernimmt und die Textstrings in aufbereiteter Form auf dem Bildschirm ausgibt (Bild 4.15).

```
Die Datei: c:\dos\command.com  wird bearbeitet
X[u
!PSQ
....
(A)bbrechen
, (W)iederholen
, (I)gnorieren
, (U)ebergehen
beim Lesen von
beim Schreiben auf
 Laufwerk %
 Gerät %
```

```
Bitte Datenträger %
 Seriennummer %
 einlegen
Fehlerhafte Dateizuordnungstabelle, Laufwerk %
Eine beliebige Taste drücken, um fortzusetzen
Stapelverarbeitung abbrechen (J/N)?
.....
/DEV/CON
COMMAND
COM
AUTOEXEC
BAT
KAUTOEXE
BAT
PATH
COMSPEC
COMMAND
COM
....
```

Bild 4.15: Analyse von COMMAND.COM mit TEXTS

Für diesen Zweck wird nachfolgend ein kleines Programm entwickelt. Es läßt sich mit der Eingabe:

```
TEXTS
```

im interaktiven Eingabemodus starten. Dann wird der Bildschirm gelöscht und es erscheint folgende Meldung:

```
 T e x t s        (c) Born Version 1.0

 Optionen  [/L=03  minimale Zeichenzahl pro Wort ]

 File     :
 Optionen :
```

Bild 4.16: Startmeldung von TEXTS

Das Programm erwartet dann den Namen der zu bearbeitenden Datei. Es sind die gültigen MS-DOS-Konventionen zu beachten. Bei Leereingaben bricht TEXTS mit folgender Meldung ab:

```
Der Dateiname fehlt
```

Falls die Datei nicht existiert, erscheint die Nachricht:

```
Die Datei <Name> kann nicht geöffnet werden
```

und das Programm wird ebenfalls beendet. Mit *<Name>* wird der eingegebene Dateiname bezeichnet. Das Programm analysiert die eingelesenen Binärwerte der Datei auf zusammenhängende darstellbare Zeichen. Finden sich mehr als drei darstellbare Zeichen hintereinander, wird diese Folge als Text interpretiert. Die Wahrscheinlichkeit für unsinnige Zeichenfolgen ist jedoch recht hoch. Deshalb kann über die Option /L=xx die Schwelle der darstellbaren Zeichen pro Wort zwischen 2

und 30 Zeichen verändert werden. Ist eine Zeichenfolge kürzer als dieser eingestellte Wert, wird sie nicht angezeigt. Mit der Option:

```
/L=10
```

werden nur Zeichenketten angezeigt, die aus mindestens zehn aufeinanderfolgenden Zeichen bestehen. Mit dieser Option läßt sich die Erkennung von Texten beeinflussen.

Alternativ besteht die Möglichkeit, den Dateinamen und die Option bereits in der Kommandozeile mit anzugeben. Es gilt dabei folgende Syntax:

```
TEXTS <Dateiname> <Optionen>
```

Im Kommandomodus erscheint die Kopfmeldung nicht mehr. Ansonsten gelten die gleichen Bedingungen und Fehlermeldungen wie bei der interaktiven Eingabe.

Allerdings läßt sich bei diesem Modus die DOS Ein-/Ausgabeumleitung verwenden, d.h. die Ausgaben von TEXTS können in eine Datei oder an den Drucker umgeleitet werden. Das Kommando:

```
TEXTS TEXTS.EXE /L=6 >TEXTS.TMP
```

erzeugt eine Textdatei mit allen Textstrings, die in der Datei TEXTS.EXE gespeichert sind. Diese Datei kann dann in Ruhe analysiert werden.

Nach dem Aufruf beginnt die Prüfung, ob die Datei existiert. Falls ja, liest TEXTS alle Bytes der Datei sequentiell ein und bearbeitet diese Bytefolge. Dies wird durch die Meldung:

```
Die Datei <Name> wird bearbeitet
```

kommentiert. In Anlehnung an die bisherigen Programme kann die Online-Hilfe mit der Anweisung:

```
TEXTS /?
```

aktiviert werden. Dann erscheint der folgende Text auf dem Bildschirm:

```
T e x t s                              (c) Born Version 1.0

Aufruf: TEXTS <Filename> <Optionen>

Optionen:

    /L=3   min. Zeichen pro Wort

Das Programm liest eine Binärdatei ein und gibt alle
Texte aus, die mehr als n Zeichen /L=xx) enthalten.
```

Bild 4.17: Online-Hilfe von TEXTS

Die Implementierung

Das Programm ist so einfach, daß auf eine Diskussion des Entwurfs verzichtet werden kann. Das Hauptprogramm übernimmt die Aufgabe, den Dateinamen aus der Kommandozeile oder interaktiv einzulesen. Dann beginnt die Analyse auf gültige Zeichenketten in der WHILE-Schleife. Die Anweisung:

```
IF (INSTR(txt$,zchn$) > 0 THEN
```

übernimmt die Analyse des Objectcodes auf Zeichenketten. Das Verfahren ist recht einfach: Alle Bytes werden daraufhin untersucht, ob sie den ASCII-Zeichen (A..Z, a..z, Ä, Ü, Ö, ä, ü, ö, ß) entsprechen. Die betreffenden Zeichen werden im Programmkopf in der Variablen *txt$* definiert. Jedes gefundene Zeichen erhöht die Variable *tmp%* um 1. Das Zeichen selbst wird an *text$* angehängt. Sobald ein nicht darstellbares Zeichen auftritt, erfolgt die Überprüfung auf die Stringlänge. Ist diese größer als die eingestellte Schwelle, wird der betreffende Text ausgegeben.

Noch ein paar Worte zur DOS-Ein-/Ausgabeumleitung. Mit:

```
TEXTS file > prn:
```

soll die Ausgabe zum Beispiel auf den Drucker umgeleitet werden. Der PRINT-Befehl schreibt die Texte jedoch direkt in den Bildschirmspeicher der Grafikkarte, die Umleitung funktioniert daher nicht. Hier muß man zu einem Trick greifen und PowerBasic zwingen, die Ausgaben an die DOS-Geräte vorzunehmen. Hierzu gibt es die OPEN-Anweisung:

```
OPEN "CONS:" FOR OUTPUT AS #2
```

die eine Ausgabeeinheit öffnet. Dann lassen sich die Zeichen mit:

```
PRINT #2, ...
```

an diese Einheit ausgeben. CONS: ist jedoch die Standard-DOS-Einheit, die auch Umleitungen zuläßt. Damit ist obiges Problem gelöst.

Die Unterprogramme *parameter* und *getval* dienen zur Decodierung der Optionen und wurden bereits in früheren Kapiteln besprochen. Einzelheiten sind dem nachfolgenden Listing zu entnehmen.

Erweiterungsvorschläge

Die Erkennung von Wörtern ist noch verbesserungsfähig. So wird nach einem Großbuchstaben in der Regel eine Sequenz von Kleinbuchstaben stehen. Die Ausgabe läßt sich zur Zeit nicht mit Strg+S anhalten. die Implementierung der More-Option sollte deshalb keine größeren Probleme bereiten.

```
X R E F                              (c) Born Version 1.0
Datei : texts.bas     Datum : 11-04-1992         Seite :  1
```

```
 Zeile     Anweisung

      ' ********************************************************
      ' File       : TEXTS.BAS
      ' Vers.      : 1.0
      ' Last Edit  : 20. 4.92
      ' Autor      : G. Born
      ' File I/O   : INPUT, OUTPUT, FILE, PRINTER
      ' Progr. Spr.: POWER BASIC
      ' Betr. Sys. : DOS 2.1 - 5.0
      ' Funktion: Das Programm liest beliebige Binärdateien ein
      '           und versucht daraus Texte zu extrahieren und
      '           anzuzeigen.
      '
      ' Aufruf:   TEXTS Filename /Optionen
      '           Optionen:  /L=XX min. Wortlänge        [ 3 ]
      '
      '           Die Werte in [] geben die Standardeinstellung
      '           wieder. Wird das Programm ohne Parameter aufge-
      '           rufen, sind Dateiname und Optionen explizit ab-
      '           zufragen. Mit dem Aufruf:
      '
      '              TEXTS /?
      '
      '           wird ein Hilfsbildschirm ausgegeben.
      ' ********************************************************
      ' Variable definieren
    1 %on = 1: %off = 0
    2 txt$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜ"
    3 txt$ = txt$ + "abcdefghijklmnopqrstuvwxyzäöüß"
    4 txt$ = txt$ + " -_+#!$%&/(){}*><[]?'`';,"

    5 datei% = 1
    6 lang% = 3                              '! 3 Zeichen für Wort

    7 ON ERROR GOTO fehler                   '! Fehlerausgang

      '############################################################
      '#                      Hauptprogramm                       #
      '############################################################

    8 kommando$ = COMMAND$                   '! Parameter ?
    9 IF LEN (kommando$) = 0 THEN            '! User Mode ?
   10  CLS                                   '! clear Screen

   11  PRINT "T e x t s                            (c) Born Version
1.0"
   12  PRINT
   13  PRINT "Optionen [ /L=03 minimale Zeichenzahl pro Wort   ]"
   14  PRINT
   15  INPUT  "File     : ",filename$
   16  INPUT  "Optionen : ",options$
   17  PRINT
   18 ELSE
   19  ptr% = INSTR (kommando$,"/?")        '! Option /?
```

```
   20  IF ptr% <> 0 THEN                    '! Hilfsbildschirm
   21   PRINT "T e x t s                    (c) Born Version 1.0"
   22   PRINT
   23   PRINT "Aufruf: TEXTS <Filename> <Optionen>"
   24   PRINT
   25   PRINT "Optionen :"
   26   PRINT
   27   PRINT "  /L=3  min. Zeichen pro Wort"
   28   PRINT
   29   PRINT "Das Programm liest eine Binärdatei ein und gibt
alle"
   30   PRINT "Texte aus, die mehr als n Zeichen (/L=xx)
enthalten."
   31   PRINT
   32   SYSTEM
   33  END IF
   34 ||||      '! Kommando Mode
   35  ptr% = INSTR (kommando$,"/")         '! Optionen ?
   36  IF ptr% = 0 THEN
   37   filename$ = kommando$               '! nur Filename
   38  ELSE
   39   filename$ = LEFT$(kommando$,ptr% -1)'! Filename separieren
   40   options$  = MID$(kommando$,ptr%)    '! Optionen separieren
   41  END IF
   42 END IF

   43 GOSUB parameter                      '! Optionen decodieren

   44 IF (lang% < 2) OR (lang% > 30) THEN   '! sinnlose
   45  PRINT                               '! Einstellung
   46  PRINT "Bitte Einstellung für Länge neu setzen" '! Fehlerexit
   47  SYSTEM
   48 END IF

   49 IF filename$ = "" THEN               '! Leereingabe ?
   50  PRINT
   51  PRINT "Der Dateiname fehlt"
   52  SYSTEM
   53 END IF

      ' prüfe ob Datei vorhanden, nein -> exit

   54 OPEN filename$ FOR BINARY AS #datei%

      '! öffne Ausgabeeinheit für I/O-Umleitung
   55 OPEN "CONS:" FOR OUTPUT AS #2

   56 PRINT #2,
   57 PRINT #2, "Die Datei: ";filename$;" wird bearbeitet"

   58 tmp% = 0
   59 text$ = ""

   60 WHILE NOT (EOF(datei%))              '! Datei sequentiell
lesen
```

```
   61  GET$ #datei%, 1, zchn$               '! lese 1 Byte aus
Binärdatei
   62  IF (INSTR(txt$,zchn$) > 0) THEN      '! Zeichen gehört zu
Wort
   63   INCR tmp%                           '! zähle Buchstaben
   64   text$ = text$ + zchn$               '! merke Zeichen
   65  ELSE
   66   IF tmp% >= lang% THEN PRINT #2, (text$) '! gebe Satz aus
   67   tmp% = 0
   68   text$ = ""
   69  END IF
   70 WEND

   71 PRINT
   72 PRINT "Ausgabe beendet"
   73 CLOSE                                '! Dateien schließen

   74 END

      '############################################################
      '#                    Hilfsroutinen                         #
      '############################################################

   75 fehler:
      '-----------------------------------------------------------
      '! Fehlerbehandlung in TEXTS
      '-----------------------------------------------------------

   76 IF ERR = 53 THEN
   77  PRINT "Die Datei ";filename$;" existiert nicht"
   78 ELSE
   79  PRINT "Fehler : ";ERR;" unbekannt"
   80  PRINT "Programmabbruch"
   81 END IF
   82 END                                  '! MSDOS Exit

   83 parameter:
      '-----------------------------------------------------------
      '! Decodiere die Eingabeoptionen
      '-----------------------------------------------------------

   84 ptr% = INSTR (options$,"/L=")
   85 IF ptr% > 0 THEN CALL getval (lang%)  '! min. Zeichen / Wort

   86 RETURN

   87 SUB getval (wert%)
      '-----------------------------------------------------------
      '! Decodiere den Eingabestring in eine Zahl
      '-----------------------------------------------------------
   88  SHARED options$, ptr%
   89  LOCAL i%

   90  ptr% = ptr% + 3                     '! ptr hinter /x=
   91  i% = 1
```

```
   92  WHILE ((ptr%+i%) =< LEN (options$)) and
(MID$(options$,ptr%+i%,1) <
        > " ")
   93   i% = i% + 1                          '! Ziffernzahl + 1
   94  WEND
   95  wert% = VAL(MID$(options$,ptr%,i%))  '! decodiere die Zahl
   96 END SUB


      ' ###### Programm Ende #########
```

Listing 4.5: TEXTS.BAS

5 DOS-Befehlserweiterungen zur Stapelverarbeitung

In den bisherigen Kapiteln haben Sie eine Reihe von Werkzeugen kennengelernt, mit denen einzelne DOS-Programme in ihrer Funktionalität ergänzt werden. Ein noch gänzlich unbearbeitetes Feld ist die Stapelverarbeitung unter DOS. Auch in der Version 5.0 kommt dieses Betriebssystem nur mit einigen spartanischen Befehlen zur Stapelverarbeitung daher. Tastaturabfragen, Schleifen und Berechnungen oder Statusabfragen, dies sind in der Regel Fremdworte beim Erstellen von Batchdateien für DOS. Einzig Digital Research hat mit DR-DOS 6.0 eine einfache Möglichkeit zur Benutzerabfrage in Batchdateien geschaffen. Das folgende Kapitel enthält ausgewählte Beispiele dieser Befehlserweiterungen die nach PowerBASIC portiert wurden. Damit erhalten Anwender von PowerBASIC die Möglichkeit (zumindest in Teilbereichen) hinter die Kulissen zu schauen, denn alle Module liegen im Quellcode vor.

ASK: Benutzerabfragen aus Batchprogrammen

Eine der am häufigsten vermißten Funktionen in Batchprogrammen ist die Möglichkeit zur Abfrage von Benutzereingaben. Zwar lassen sich beim Aufruf einer Stapelverarbeitungsdatei Parameter angeben und auswerten. Was aber nicht klappt ist die Abfrage der Tastatur im laufenden Stapelverarbeitungsprogramm. Wie oft habe ich in der Vergangenheit darüber geflucht, daß DOS keinen entsprechenden Befehl kennt. Jegliche Benutzersteuerung, Menüs zum Aufruf verschiedener Anwendungen, Abfragen von Laufwerken etc. ist mit den Standard-DOS-Befehlen unmöglich. Ein entsprechendes Schattendasein ist daher den meisten Batchprogrammen vorbestimmt. Vor einigen Jahren stieß ich dann aber in einer US-Computerzeitschrift auf einen Hinweis, wie eine Benutzerabfrage aus Batchprogrammen zu realisieren sei. Das daraus entwickelte Programm ASK leistet mir seither in vielen Fällen gute Dienste. Auch wenn die Technik bei vielen Programmierern mittlerweile als alter Hut gilt, möchte ich den Lesern die PowerBASIC-Version nicht vorenthalten. Vielleicht ist das Programm doch für einige unter Ihnen noch neu und hilfreich.

Der Entwurf

Als erstes möchte ich wieder auf die Anforderungen an das Programm eingehen. Das Programm ASK erlaubt es, innerhalb einer Stapeldatei Benutzereingaben abzufragen. Es besitzt dabei folgende Aufrufstruktur:

```
ASK <Text>
```

Wird ein <*Text*> in der Kommandozeile angegeben, erscheint dieser nach dem Aufruf von ASK auf dem Bildschirm, unabhängig von der ECHO-Einstellung (ECHO ON/ECHO OFF) der Batchdatei. Der Text darf beliebige Zeichen enthalten und ist durch ein Leerzeichen von ASK zu trennen. Damit erhalten Sie die Möglichkeit, beim Ablauf der Batchdatei eine Nachricht an den Benutzer zu senden, z.B:

```
ASK Soll das Programm beendet werden (J/N) ?
```

Eventuelle Benutzereingaben (hier »J« oder »N«) sollen dabei direkt hinter der Abfrage am Bildschirm erscheinen.»Bei einem Aufruf ohne Parameter wird kein Text auf dem Bildschirm angezeigt. Dies ist teilweise erwünscht, wenn die Nachrichten an den Benutzer per ECHO-Befehl oder aus einer Datei ausgegeben werden sollen. In diesen Fällen erscheint dann allerdings auch die Benutzereingabe am Anfang einer neuen Zeile.

ASK wartet anschließend auf eine Benutzereingabe von der Tastatur. Der betreffende Tastaturcode ist dann an die Batchdatei zu übergeben. Dort kann die Ablaufsteuerung entsprechend reagieren. Nun ist aber noch die Frage offen, wie sich die Ergebnisse der Tastaturabfragen von ASK an DOS übergeben und im Batchprogramm auswerten lassen.

Hier bietet DOS einen trickreichen Mechanismus, der für unsere Zwecke benutzt werden kann. Wird ein Programm beendet, kann es einen Fehlercode an DOS zurückgeben. Dieser Fehlercode ist bei einem normal beendeten Programm gleich 0. Allerdings sind Werte zwischen 0 und 255 erlaubt. Unter DOS kann dieser Fehlercode durch die Batchanweisung:

```
ERRORLEVEL
```

abgefragt werden. Die Anweisung:

```
IF ERRORLEVEL 33 GOTO L1
```

innerhalb einer Batchdatei veranlaßt eine Programmablaufsteuerung. Immer wenn der interne Wert des Fehlercodes >= 33 ist, wird zur Marke *L1* verzweigt. Andernfalls führt DOS den auf die IF-Abfrage folgenden Befehl aus. In einer Batchdatei darf die ERRORLEVEL-Funktion beliebig häufig aufgerufen werden. Wichtig ist lediglich, daß ein Vergleich innerhalb der IF-Anweisungen mit ERRORLEVEL immer auf »größer gleich« erfolgt. Obiges Beispiel ist dann als:

```
IF ERRORLEVEL >= 33 THEN
```

zu interpretieren. Zwar nutzen nur wenige DOS-Programme (wie BACKUP) diesen Mechanismus und geben Fehlercodes beim Abbruch zurück. Aber das Prinzip läßt sich in ASK verwenden. Ein eingelesener Tastencode muß lediglich als Fehlercode (Bytewert zwischen 0 und 255) zurückgegeben werden.

Weiterhin soll sich das Programm mit der Option:

```
ASK /?
```

aktivieren lassen. Dann erscheint eine Anzeige mit folgenden Erläuterungen auf dem Bildschirm:

```
 A S K                              (C) Born G.   Version 1.0

 Aufruf:   ASK <Text>

 Das Programm erlaubt Benutzerabfragen aus Batchdateien
 und gibt die Tastaturcodes als Fehlercode an DOS zurück.
 Die Fehlercodes lassen sich durch ERRORLEVEL auswerten.
 Das Feld <Text> ist optional und erlaubt die Angabe einer
 Benutzermeldung, die beim Aufruf erscheint.
```

Bild 5.1: Meldung des Programmes ASK

Ein Beispielprogramm

Nachfolgendes Beispiel demonstriert die Verwendung von ASK in einer Batchdatei. Es soll eine einfache Menüsteuerung entworfen werden. Nach dem Aufruf des Batchprogrammes erscheint folgende Maske auf dem Bildschirm:

```
 ##########################################################
 #                    M  E  N  Ü                          #
 ##########################################################
 #  ***     P r o g r a m m a u s w a h l       ***   #
 #                                                        #
 #                  0 Ende                                #
 #                  1 Textverarbeitung                    #
 #                  2 Tabellenkalkulation                 #
 #                  3 Datenbank                           #
 #                  4 ---                                 #
 #                                                        #
 ##########################################################
Bitte eine Kennziffer eingeben :
```

Bild 5.2: Menüsteuerung mit ASK

Der Benutzer kann nun die verschiedenen Programme durch Eingabe einer Ziffer aktivieren. Bei ungültigen Eingaben wird die Maske aufgefrischt und die Abfrage erscheint erneut. Das Programm läßt sich durch die Eingabe der Zahl 0 beenden. Das zugehörige Batchprogramm besitzt dabei den in Listing 5.1 gezeigten Aufbau.

```
ECHO OFF
:============================================
: Programm:  MENU.BAT
: Version:   1.0  (25.5.92) (c) Born
: Funktion:  Demonstration einer Menüverwaltung
:            unter Verwendung von ASK
:============================================
```

```
: Maske aufbauen
:LOOP
CLS
ECHO ########################################################
ECHO #                    M  E  N  Ü                        #
ECHO ########################################################
ECHO #  ***     P r o g r a m m a u s w a h l       ***    #
ECHO #                                                      #
ECHO #                  0 Ende                              #
ECHO #                  1 Textverarbeitung                  #
ECHO #                  2 Tabellenkalkulation               #
ECHO #                  3 Datenbank                         #
ECHO #                  4 ---                               #
ECHO #                                                      #
ECHO ########################################################
ECHO Bitte eine Kennziffer eingeben :
ECHO.
:
: Hier wird ASK zur Benutzerabfrage verwendet
:
ASK Bitte eine Kennziffer eingeben :
:
: Auswertung der Benutzereingaben
: werte zuerst immer die höheren Codes aus
: Zahl 0 -> Code = 48, 1 = 49, usw.
:
IF ERRORLEVEL 52 GOTO LOOP
IF ERRORLEVEL 51 GOTO L3
IF ERRORLEVEL 50 GOTO L2
IF ERRORLEVEL 49 GOTO L1
IF ERRORLEVEL 48 GOTO Exit
GOTO loop
:L1      Aufruf der Textverarbeitung
CD TEXTE
WORD
CD..
GOTO loop
:L2      Aufruf der Tabellenkalkulation
CD 123
LOTUS
CD..
GOTO loop
:L3      Aufruf des Datenbankprogrammes
CD DBASE
DBASE
CD..
GOTO loop
:Exit
ECHO ON
```

Listing 5.1: Batchprogramm zur Menüsteuerung

Die eigentliche Maske wird mit ECHO-Befehlen aufgebaut. Am Ende der Sequenz erlaubt der ASK-Befehl die Tastaturabfrage, wobei gleichzeitig eine Benutzermeldung abgesetzt wird. Die Auswertung der Benutzereingabe erfolgt über die IF-Sequenz. Die Tasten 0 bis 4 haben die ASCII-Codes 48 bis 52. Wichtig ist dabei die Reihenfolge der Abfrage, da immer:

Errorlevel >= Tastencode

ausgewertet wird. Es führt an dieser Stelle zu weit, auf alle Einzelheiten der Batchdatei einzugehen. Der interessierte Leser sei hier auf /5/ verwiesen, wo die Thematik detailliert an vielen Beispielen beschrieben wird.

Noch ein Wort zu den zurückgegebenen Codes der einzelnen Tasten. Solange Sie eine Taste mit Buchstaben oder Ziffern betätigen, gibt ASK den entsprechenden ASCII-Code zurück. Im Anhang findet sich eine Tabelle mit den Codes der einzelnen Buchstaben und Zeichen. Für ASK müssen Sie in der Spalte mit den Dezimalzahlen nachsehen. Als Sonderfall sind jedoch die Funktionstasten oder Tastenkombinationen (z.B. Alt+F) zu betrachten. Hier liefert die Tastatur einen 2-Byte-Tastencode (Extended-ASCII-Code) zurück, wobei der erste Wert 0 ist. An DOS kann jedoch immer nur ein Byte als Fehlercode zurückgegeben werden. Bei Funktionstasten wäre bei Verwendung des 1. Byte dann immer der Wert 0 in ERRORLEVEL zu finden. Die Rückgabe des zweiten Codebyte führt auch nicht weiter, da dieser Wert den Codes normaler Tasten entspricht. Alle Tasten die einen sogenannten *Extended-ASCII-Code* zurückgeben, sind damit für ASK tabu. Um diese Tasten in einer Batchdatei einfacher abzufangen, soll ASK in diesem Fall den Fehlercode 255 an DOS zurückgeben. Das nachfolgend vorgestellte Programm FKEY.BAS erlaubt dagegen die explizite Abfrage von Funktionstasten aus Batchdateien.

Auf der Begleitdiskette findet sich übrigens das folgende Programm (E.BAT), welches die Programmentwicklung mit ASK erleichtert.

```
@ECHO OFF
 :=========================================
 : Programm zur Ermittlung von ERRORLEVEL
 :=========================================
 %1 %2 %3 %4 %5 %6 %7 %8 %9
 : ermittle Errorlevel
 FOR %%a IN (0 1 2) DO IF ERRORLEVEL %%a00  SET $1=%%a
 GOTO %$1%
 :2     ERRORLEVEL 200 - 255
 FOR %%a IN (0 1 2 3 4 5) DO IF ERRORLEVEL 2%%a0  SET $2=%%a
 FOR %%a IN (0 1 2 3 4 5 6 7 8 9) DO IF ERRORLEVEL 2%$2%%%a  SET
$3=%%a
 : korrigiere Überlauf über 255
 IF NOT '%$1%%$2%%$3%' == '259' GOTO SET_E
 FOR %%a IN (0 1 2 3 4 5) DO IF ERRORLEVEL 2%$2%%%a SET $3=%%a
 GOTO SET_E
 :1    100 - 199
```

```
 :0     00 -  99
 FOR %%a IN (0 1 2 3 4 5 6 7 8 9) DO IF ERRORLEVEL %$1%%%a0  SET
$2=%%a
 FOR %%a IN (0 1 2 3 4 5 6 7 8 9) DO IF ERRORLEVEL %$1%%$2%%%a  SET
$3=%%a
 :SET_E
 SET $ERROR=%$1%%$2%%$3%
 SET $1=
 SET $2=
 SET $3=
 ECHO.
 ECHO ERRORLEVEL IST %$ERROR%
 ECHO ON
```

Listing 5.2: E.BAT

Wird ASK über das Modul folgendermaßen aktiviert:

```
E ASK Bitte eine Taste eingeben
```

zeigt E.BAT anschließend den DOS-ERRORLEVEL-Code im Klartext auf dem Bildschirm an. Damit läßt sich recht einfach feststellen, welche Taste welchen Code liefert. Weitere Hinweise zu diesem Thema sowie eine erweiterte Version findet sich in /5/.

Die Implementierung

Doch nun möchte ich auf die Implementierung zu sprechen kommen. Das Programm ASK ist vom Aufbau recht einfach, so daß auf ein Hierarchiediagramm verzichtet wird. Es besteht nur aus einem Hauptmodul, welches als erstes die Kommandozeile auf den Parameter */?* hin analysiert, gegebenenfalls den Hilfebildschirm ausgibt und dann (mit dem Fehlercode 0) abbricht.

Ohne Option wird der eventuell in der Kommandozeile vorhandene Text mittels der PowerBASIC-Funktion COMMAND$ ermittelt und per PRINT-Kommando ausgegeben. Liegt kein Text vor, gibt COMMAND$ einen Leerstring zurück.

Dann wird ein Zeichen von der Tastatur gelesen. Hierzu eignet sich die PowerBASIC-Funktion INPUT$, da sie eine vorgegebene Anzahl von Zeichen einliest, ohne daß diese mit der Eingabetaste zu quittieren sind. Wird eine Funktionstaste betätigt, liefert die Funktion den Code 0.

Der Fehlercode läßt sich in PowerBASIC leicht über die Konstruktion:

```
END (Fehlercode)
```

an DOS zurückgeben. Einzelheiten sind dem nachfolgenden Listing zu entnehmen.

Erweiterungsvorschläge

Das Programm ASK kann so optimiert werden, daß nur bestimmte Tastencodes (z.B. J, j, N, n) akzeptiert werden. Dann ist die Auswertung in einer Batchdatei etwas einfacher.

```
X R E F   /Z=50                              (c) Born Version 1.0
Datei : ask.bas      Datum : 05-29-1992         Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : ASK.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 16.5.92
      '! Autor      : G. Born
      '! Progr. Spr.: PowerBASIC
      '! Betr. Sys. : DOS 2.1 - 5.0 (DR-DOS 5.0/6.0)
      '! Funktion: Das Programm wird mit der Eingabe:
      '!
      '!           ASK <Text>
      '!
      '!           aufgerufen. Es erlaubt Benutzerabfragen in
      '!           Batch-Dateien. Der <Text> wird auf dem
      '!           Bildschirm ausgegeben. Dann wartet ASK auf
      '!           eine Eingabe. Das Ergebnis wird als Errorcode
      '!           an DOS übergeben und läßt sich über ERRORLEVEL
      '!
      '!             IF ERRORLEVEL 3 ...
      '!
      '!           abfragen.
      '*********************************************************
      '! Variable definieren
    1 zchn$ = ""
    2 ptr% = 0
    3 kommando$ = ""

      '############################################################
      '#                      Hauptprogramm                       #
      '############################################################

    4 kommando$ = COMMAND$                  '! Parameter ?

    5 ptr% = INSTR (kommando$,"/?")         '! Option /?
    6 IF ptr% <> 0 THEN                     '! Hilfsbildschirm
    7   PRINT "A S K                         (c) Born Version 1.0"
    8   PRINT
    9   PRINT "Aufruf: ASK <Text>"
   10   PRINT
   11   PRINT "Das Programm erlaubt Benutzerabfragen aus
Batchdateien"
   12   PRINT "und gibt die Tastaturcodes als Fehlercode an DOS
zurück"
   13   PRINT "Die Fehlercodes lassen sich durch ERRORLEVEL
auswerten."
```

```
   14   PRINT "Das Feld <Text> ist optional und erlaubt die Angabe
einer"
   15   PRINT "Benutzermeldung, die beim Aufruf erscheint."
   16   PRINT
   17   SYSTEM
   18 END IF

   19 PRINT kommando$;" ";                    '! Text ausgeben
   20 zchn$ = INPUT$ (1)                      '! Benutzereingabe

   21 IF ASC(zchn$) = 0 THEN
   22  END (255)                              '! Funktionstaste
   23 END IF

   24 END ASC(zchn$)                          '! Ende

      '***** Programm Ende ******
```

Listing 5.3: ASK.BAS

ESC:Steuersequenzen im Klartext

Ein weiteres Thema ist die Ausgabe von Texten mit eingebetteten Steuersequenzen an Bildschirm und Drucker. Mit ECHO lassen sich zwar recht einfach Texte an den Bildschirm oder andere Einheiten ausgeben. Kritisch wird es allerdings, falls nicht darstellbare Zeichen auszugeben sind. Beispiele sind der Zeilenvorschub (Code = 10) oder das Zeichen mit dem Code 0. Solche Zeichen werden häufig zur Ansteuerung von Druckern und Peripheriegeräten benötigt. Um hier über eine komfortable Eingabe zu verfügen, habe ich das Programm ESC entwickelt. Es ist auf der Begleitdiskette beigefügt und erlaubt die Ausgabe beliebiger Zeichen an die aktuelle Ausgabeeinheit.

Der Entwurf

Für ESC gilt folgende Aufrufsyntax:

```
ESC <Param1> <Param2> .... <Param n>
```

Hinter dem Programmnamen lassen sich mehrere Parameter angeben, die durch Leerzeichen zu trennen sind. Als Parameter sind dabei beliebige Folgen von zweiziffrigen Hexadezimalzahlen und Zeichenketten erlaubt. Zeichenketten müssen durch Anführungszeichen geklammert werden. Die zweiziffrigen Hexcodes (z.B. 0A 0D 0C) werden dann in das entsprechende ASCII-Zeichen konvertiert und zur Standardausgabeeinheit weitergeleitet. Die Codefolge 0A 0D löst dann zum Beispiel einen Zeilenvorschub aus. Texte müssen in Anführungszeichen stehen (z.B. "Hallo") und werden direkt zur Ausgabeeinheit weitergeleitet. Hierdurch lassen sich beliebige Steuersequenzen zusammenstellen.

Das Programm soll weiterhin die DOS-Ein-/Ausgabeumleitungen unterstützen, damit sich die Zeichen dann zu jeder beliebigen Einheit

umdirigieren lassen. Nachfolgend möchte ich einige Beispiele für solche Aufrufe zeigen:

```
ESC 0C >PRN:|||(Vorschub auf Drucker)
ESC 07||||(Bell auf Bildschirm)
ESC 41 42 >A:TST.TXT|(Zeichen AB in die Datei TST.TXT)
```

Die folgende Anweisung zeigt, wie sich Zeichenketten und Hexzahlen mischen lassen.

```
ESC 41 20 "Hallo, dies ist eine Zeichenkette " 0d 0a
```

Diese Anweisung gibt den Text:

```
A Hallo, dies ist eine Zeichenkette
```

auf dem Bildschirm aus. Der Code 41 steht dabei für den Buchstaben *A* (siehe ASCII-Tabelle im Anhang). Mit *0d 0a* wird nach der Textausgabe ein Zeilenvorschub ausgelöst.

Nun noch einige Hinweise zur Behandlung von Fehleingaben. Fehlt das erste Anführungszeichen bei einem String, bricht ESC mit einer Fehlermeldung ab. ERRORLEVEL enthält dann den Wert 255. Fehlt das letzte Anführungszeichen eines Strings, interpretiert ESC die restlichen Parameter einfach als Text. Das Kommando:

```
ESC 41 Hallo dies ist ein Text"
```

führt demnach zu einer Fehlermeldung, während:

```
ESC "Hallo dies ist ein Text 0d 0a
```

den kompletten Text der Zeile ausgibt (*0a 0d* werden als Text interpretiert). Parameter und Text dürfen beliebig gemischt werden und sind durch Leerzeichen zu trennen. Bei Hexdezimalzahlen dürfen die Werte zwischen 00 und FF liegen. Eine Hexziffer kann die Zeichen 0..9,A,B,C,D,E,F und die entsprechenden Kleinbuchstaben a,b,c,d,e,f enthalten. Alle anderen Zeichen führen zu einem Fehlerabbruch.

Weiterhin läßt sich mit dem Befehl:

```
ESC /?
```

die Online-Hilfe des Programmes abrufen. Auf dem Bildschirm erscheint dann ein entsprechender Fehlertext.

```
E S C                          (c) Born Version 1.0

Aufruf: ESC <Param 1> <Param 2> .. <Param n>

Das Programm gibt in den Parametern angegebene Texte
oder Hexzahlen (2 Ziffern) an die Standardausgabeeinheit
aus. Beispiel:

ESC 0C > PRN:
```

```
ESC "Hallo" 0D 0A
```

Bild 5.3: Online-Hilfe von ESC

Anwendungsbeispiele

Um Ihnen die Möglichkeiten des Programmes etwas zu erläutern, möchte ich einige kleine Anwendungen vorstellen.

Druckeransteuerung mit ESC

Das folgende kleine Beispiel zeigt, wie sich mit ESC die Schriftarten an einem EPSON-Drucker umschalten lassen. Die Druckerhersteller geben in den meisten Fällen die notwendigen Steueranweisungen für die Umstellung der Schriftarten in ihren Handbüchern an. Mit dieser Kenntnis ist es relativ einfach, den Drucker umzustellen.

Bei Epson-Druckern und den dazu kompatiblen Geräten gelten folgende Codesequenzen (Angaben im Hexadezimalmodus) zur Formatumstellung:

```
Drucker Reset       : 1B 40
Near Letter Quality : 1B 78 01
Draft               : 1B 78 00
Roman               : 1B 6B 00
Sans Serif          : 1B 6B 01
Pica                : 1B 21 00
Elite               : 1B 21 01
Kursiv Ein          : 1B 34
Kursiv Aus          : 1B 35
Fett Ein            : 1B 45
Fett Aus            : 1B 46
```

Die Schriftarten »Roman« und »Sans Serif« stehen teilweise aber nur im »Near Letter«-Modus zur Verfügung. Weitere Codesequenzen (z.B. für Unterstreichen etc.) lassen sich an Hand der Informationen aus den Druckerhandbüchern erstellen. Mit dem Programm ESC lassen sich natürlich die Steuersequenzen recht elegant an den Drucker übergeben. D»s Kommando:

```
ESC 1B 34 > PRN:
```

stellt die Kursivschrift ein, während der Befehl:

```
ESC 1B 35 > PRN:
```

den Modus wieder abschaltet. Damit läßt sich ein kleines Batchprogramm zur Fontumstellung erstellen. Das nachfolgende Listing zeigt den Aufbau der Batchdatei. Bei der Umstellung auf »Roman« und » Sans Serif« ist gleichzeitig der »Letter«-Modus einzuschalten, da diese Typen nur in diesem Modus definiert sind (EPSON LX 800).»

```
ECHO OFF
:============================================================
```

```
: Programm: FONT.BAT
: Version:  31.5.92 (c) Born
: Batchdatei zur Fontumstellung eines Druckers.
:=============================================================
CLS
ECHO ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
ECHO º Utility zur Umstellung der Schriftart des Druckers º
ECHO ÌÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¹
ECHO º                                                    º
ECHO º                  0 Draft                           º
ECHO º                  1 Pica                            º
ECHO º                  2 Roman                           º
ECHO º                  3 Sans Serif                      º
ECHO º                  4 Kursiv Ein                      º
ECHO º                  5 Kursiv Aus                      º
ECHO º                                                    º
ECHO ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
ECHO.
ASK Bitte eine Zahl eingeben
: prüfe Eingabeparameter
:CHECK
IF ERRORLEVEL 54 GOTO EXIT
IF ERRORLEVEL 53 GOTO L5
IF ERRORLEVEL 52 GOTO L4
IF ERRORLEVEL 51 GOTO L3
IF ERRORLEVEL 50 GOTO L2
IF ERRORLEVEL 49 GOTO L1
IF ERRORLEVEL 48 GOTO L0
GOTO EXIT
:L0
ECHO Draft Modus einschalten
ESC 1B 78 00 > PRN:
GOTO EXIT
:L1
ECHO Pica Modus einschalten
ESC 1B 21 00 > PRN:
GOTO EXIT
:L2
ECHO Roman Modus einschalten
ESC 1B 78 01 1B 6B 00   > prn:
GOTO EXIT
:L3
ECHO Sans Serif Modus einschalten
ESC 1B 78 01 1B 6B 01   > prn:
GOTO EXIT
:L4
ECHO Kursiv Schrift einschalten
ESC 1B 34   > prn:
GOTO EXIT
:L5
ECHO Kursiv Schrift ausschalten
ESC 1B 35   > prn:
GOTO EXIT
:EXIT
ECHO ON
```

Listing 5.4: Programm zur Fontumschaltung

Der Drucker kann durch einfaches Aus- und wieder Einschalten auf den alten Schriftmodus zurückgesetzt werden. Notfalls läßt sich auch eine entsprechende Steuersequenz erzeugen, die diese Aufgabe erledigt. In der Grundeinstellung ist der Schrifttyp »Elite« mit dem »Draft«-Modus wirksam. Die obigen Überlegungen sind auch auf andere Drucker umsetzbar. Sie müssen sich lediglich die erforderlichen Sequenzen aus den Druckerhandbüchern beschaffen. Damit lassen sich unterschiedliche Schriftarten auch unter DOS nutzen.

Abfrage des Druckerstatus aus Batchdateien

Weiterhin können mit ESC auch COM-Dateien erzeugt werden. Die Abfrage des Druckerstatus kann über das kleine Programm LPTCHK.COM erfolgen. Hierzu müssen Sie lediglich die folgenden Anweisungen eingeben:

```
ESC b4 02 31 d2 cd 17 > LPTCHK.COM
ESC 88 e0 b4 4c cd 21 >> LPTCHK.COM
```

Die Codes entsprechen den Maschinenbefehlen zur Abfrage des Druckerstatus an LPT1. Das Programm gibt den Status über ERRORLEVEL an DOS zurück. Ein Aufruf erfolgt dann mit LPTCHK:

```
LPTCHK
IF ERRORLEVEL .....
......
```

Die betreffenden Codes für verschiedene Druckerstörungen (Drucker ausgeschaltet, Papierende, Offline etc.) können Sie mit dem Programm E.BAT ermitteln.

Print-Screen aus Batchdateien

Ein anderes einfaches COM-Programm läßt sich mit dem Befehl:

```
ESC CD 05 C3 > PRTSCR.COM
```

erzeugen. Sobald Sie das Programm von der DOS-Ebene mit:

```
PRTSCR
```

aufrufen, wird eine Hardcopy des Bildschirminhaltes auf dem Drucker ausgegeben. Offenbar besitzt das Programm die gleiche Wirkung wie die Druck-Taste des Rechners. Sie sehen, mit etwas Know how und den beschriebenen Utilities sind interessante Dinge möglich.

Die Implementierung

Der Aufbau des Programmes ist recht einfach, so daß ich an dieser Stelle auf ein Hierarchiediagramm verzichten möchte.

Das Hauptprogramm

Im Hauptprogramm wird zunächst geprüft, ob die Option */?* gesetzt ist. Ist dies der Fall, gibt ESC den Text der Online-Hilfe aus und endet dann.

Andernfalls beginnt die Decodierung der Parameter der Kommandozeile. Für diese Aufgabe ist das Modul *getpara* zuständig.

Um die Ein-/Ausgabeumleitung zu ermöglichen, muß der DOS-Kanal geöffnet werden:

```
OPEN "CONS:" FOR OUTPUT AS #1
```

Die betreffende Technik wurde bereits im vorherigen Kapitel (TEXTS) vorgestellt.

getpara

In diesem Modul werden als erstes führende Leerzeichen vor einem Parameter entfernt (*skipblank*). Bei der Decodierung der Parameter sind dann Hexzahlen und Strings zu unterscheiden. Beginnt ein Parameter mit Anführungszeichen (z.B: "Textzeile...), liegt per Definition eine Zeichenkette vor. Dann aktiviert *getpara* das Unterprogramm *WRString*. Andernfalls liegt eine Hexzahl vor, die mit *WRVal* decodiert und ausgegeben wird.

WRVal

Dieses Unterprogramm bearbeitet immer zwei Zeichen aus der Kommandozeile und versucht diese als Hexzahl zu interpretieren. Die Decodierung erfolgt über die Sequenz:

```
tmp%=INSTR("0123456789ABCDEF",zchn$)
```

Liegt keine gültige Hexziffer vor, ist *tmp%=0*, und das Modul bricht mit einer Meldung und dem Fehlercode 255 ab. Andernfalls wird *tmp%* um 1 erniedrigt und gibt damit direkt den Wert der Hexziffer wieder. Sobald eine Hexzahl errechnet wurde, gibt *WRVal* diese über den folgenden Befehl aus:

```
PRINT #1, CHR$(wert%)
```

Damit werden die Hexzeichen der Kommandozeile in den zugehörigen ASCII-Code gewandelt.

;**WRString;**

Dieses Unterprogramm übernimmt die Ausgabe von Texten. Es wird aufgerufen, falls ein Parameter mit einem Anführungszeichen (") beginnt. Dann sucht *WRString* den Abschluß des Strings. Fehlt dieser, wird einfach die Restzeile ausgegeben. Andernfalls beschränkt sich die Ausgabe auf den String zwischen den Anführungszeichen. Innerhalb des Textes kann deshalb kein Anführungszeichen verwendet werden. Ist dies erforderlich, kann das Zeichen direkt als Hexcode definiert werden (z.b: "Hallo " 22 "Du da" 22).

fehler

Dieses Modul fängt die Laufzeitfehler von PowerBASIC ab.

Anmerkung: Beim Test des Programmes ist mir noch eine Anomalie aufgefallen, die vermutlich in der Implementierung von PowerBASIC liegt. Die Anweisung:

```
••• ESC 20 "Test" 0d 0a "Text"
```

erzeugt lediglich die erste Zeile, während das Wort *"Text"* nicht mehr auf dem Bildschirm erscheint. Offenbar werden alle Zeichen hinter dem Zeilenvorschub abgeschnitten. Wird die obige Zeile dagegen mit einem zweiten Zeilenvorschub (0d 0a) abgeschlossen, erscheinen die zwei Zeilen. Der Fehler tritt hingegen nicht auf, falls die Ausgabe an eine andere DOS-Einheit umgeleitet wird.

Erweiterungsvorschläge

Die Erkennung von Anführungszeichen im Text ist eine sinnvolle Ergänzung von ESC. Weiterhin kann die Separierung von Parametern verbessert werden, so daß auch andere Trennzeichen erlaubt sind. Als dritte Ergänzung kann die Decodierung der Hexzahlen so erweitert werden, daß auch Zahlen mit einer Ziffer erkannt werden.

```
X R E F   /Z=50                          (c) Born Version 1.0
Datei : esc.bas      Datum : 06-02-1992         Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : ESC.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 16.5.92
      '! Autor      : G. Born
      '! Progr. Spr.: PowerBASIC
      '! Betr. Sys. : DOS 2.1 - 5.0 (DR-DOS 5.0/6.0)
      '! Funktion: Das Programm wird mit der Eingabe:
      '!
      '!           ESC <Para 1> <Para 2> .. <Para n>
      '!
      '!           aufgerufen. Es liest die Parameter und gibt
      '!           den Inhalt auf der Standard Ausgabeeinheit aus.
      '!           Bei Zahlen als Parameter werden diese als
Hexwerte
      '!           interpretiert und in ASCII Codes gewandelt.
Beispiel:
      '!
      '!             ESC 20 "Hallo" 0D 0A
      '!
      '!           Zahlen werden als Hexwerte mit je 2 Ziffern
interpre-
      '!           tiert. Parameter sind durch Blanks zu
separieren.
```

```
      '!           Zeichenketten sind in ".." einzuschließen.
      '*********************************************************
      '! Variable definieren
    1 ptr% = 0
    2 kommando$ = ""
    3 lang% = 0

      '#############################################################
      '#                       Hauptprogramm                       #
      '#############################################################

    4  ON ERROR GOTO fehler

    5  kommando$ = COMMAND$                  '! Parameter ?

    6  ptr% = INSTR (kommando$,"/?")         '! Option /?
    7  IF ptr% <> 0 THEN                     '! Hilfsbildschirm
    8   PRINT "E S C                         (c) Born Version 1.0"
    9   PRINT
   10   PRINT "Aufruf: ESC <Param 1> <Param 2> .. <Param n>
   11   PRINT
   12   PRINT "Das Programm gibt in den Parametern angegebene
Texte"
   13   PRINT "oder Hexzahlen (2 Ziffern) an die
Standardausgabeeinheit"
   14   PRINT "aus. Beispiel:"
   15   PRINT
   16   PRINT "ESC 0C > PRN:"
   17   PRINT "ESC "Hallo" 0D 0A"
   18   PRINT
   19   SYSTEM
   20  END IF

   21  OPEN "CONS:" FOR OUTPUT AS #1

   22  ptr% = 1
   23  lang% = LEN(kommando$)                        '! Länge
Parameterstring

   24  WHILE ptr% <= lang%                           '! separiere
Parameter
   25   CALL getpara (ptr%, kommando$)
   26  WEND

   27  CLOSE #1

   28 END                                            '! Ende

      '#############################################################
      '#                     Hilfsroutinen                         #
      '#############################################################

   29 SUB getpara (ptr%, text$)

      '!-----------------------------------------------------------
```

```
      '! lese die Parameter und geben sie aus
      '!-------------------------------------------------------
   30 LOCAL zchn$

      '! suche Anfang des Parameters
   31  CALL skipblank (ptr%,text$)        '! skip führende blanks

      '! liegt ein String vor ?

   32  zchn$ = MID$(text$,ptr%,1)
   33  IF (zchn$ = CHR$(34)) THEN
   34   CALL WRSTRING (ptr%, text$)       '! String ausgeben
   35  ELSE
   36   CALL WRVal (ptr%, text$)          '! Hexwert ausgeben
   37  END IF
   38 END SUB

   39 SUB skipblank(ptr%,text$)
      '-------------------------------------------------------
      '! überlese führende Blanks in einer Zeichenkette
      '! text$ = Zeichenkette, ptr% = Zeiger in Kette
      '-------------------------------------------------------
   40 SHARED lang%

   41 WHILE (ptr% =< lang%) and (MID$(text$,ptr%,1) = " ")
   42  INCR ptr%
   43 WEND
   44 IF ptr% >= lang% THEN
   45  CALL Ende (0)                   '! Textende erreicht
   46 END IF
   47 END SUB


   48 SUB WRVal (ptr%,text$)

      '!-------------------------------------------------------
      '! decodieren der Hexzahl
      '!-------------------------------------------------------

   49 LOCAL tmp%, zchn$, wert%

   50  zchn$ = UCASE$(MID$(text$,ptr%,1)) '! hole 1. Ziffer
   51  tmp%  = INSTR("0123456789ABCDEF",zchn$) '! decodiere Ziffer

   52  IF tmp% = 0 THEN                 '! Wert gefunden ?
   53   PRINT "Fehler in Parameter ";MID$(text$,ptr%,10)  '! Text
ausgeben
   54   CALL Ende (255)
   55  END IF

   56  wert% = (tmp%-1) * 16

   57  INCR ptr%
   58  zchn$ = UCASE$(MID$(text$,ptr%,1)) '! hole 2. Ziffer
   59  tmp%  = INSTR("0123456789ABCDEF",zchn$) '! decodiere Ziffer
```

```
   60  IF tmp% = 0 THEN                  '! Wert gefunden ?
   61   PRINT "Fehler in Parameter ";MID$(text$,ptr%,10)  '! Text
ausgeben
   62   CALL Ende (255)
   63  END IF

   64  wert% = wert% + (tmp% - 1)

   65  PRINT #1, CHR$(wert%);             '! Hexzahl als ASCII-Code
   66  INCR ptr%                          '! auf Folgezeichen
   67 END SUB

   68 SUB WRString (ptr%,text$)

      '!-----------------------------------------------------------
      '! Ausgabe des Strings
      '!-----------------------------------------------------------

   69 LOCAL tmp%, zchn$, wert%
   70 SHARED lang%

      '! suche Ende des Strings

   71  INCR ptr%
   72  anf% = ptr%                          '! merke Anfang
   73  WHILE (MID$(text$,ptr%,1) <> CHR$(34)) AND ptr% <= lang%
   74   INCR ptr%                           '! hole nächstes Zeichen
   75  WEND

   76  PRINT #1, MID$(text$,anf%,ptr%-anf%); '! Text ausgeben
   77  INCR ptr%

   78 END SUB

   79 fehler:
      '!-----------------------------------------------------------
      '! Abfrageroutine für PowerBASIC Fehler
      '!-----------------------------------------------------------

   80  PRINT "Fehler : ";ERR;
   81  CALL Ende(255)
   82 RETURN

   83 SUB Ende (errx%)
      '!
      '! schließe Dateien und terminiere
      '!
   84  CLOSE #1
   85  END (errx%)
   86 END SUB

      '!***** Programm Ende ******
```

Listing 5.5: ESC.BAS

GET: Abfrage von Systemparametern aus Batchdateien

Eine andere interessante Aufgabe ist die Abfrage von Systemparametern aus Batchdateien. Das oben beschriebene Programm LPTCHK ist lediglich ein einfacher Ansatz. Die Abfrage von Datum, Uhrzeit und freiem Speicher sind ebenfalls interessante Alternativen. Mit dem Programm GET läßt sich dies leicht erreichen. Der Befehl:

```
GET /?
```

aktiviert die Online-Hilfe des Moduls. Auf dem Bildschirm erscheint folgender Text:

```
G E T                          (c) Born Version 1.0

Aufruf: GET  <Option>

Erlaubt die Abfrage bestimmter Parameter
GET /DAY   ermittelt den Tag (1-31)
GET /MONTH ermittelt den Monat (1-12)
GET /YEAR  ermittelt das Jahr (0-99)
GET /WEEK  ermittelt den Wochentag (0-6)
GET /SEC   ermittelt die Sekunden (0-59)
GET /MIN   ermittelt die Minuten (0-59)
GET /STD   ermittelt die Stunden (0-23)
GET /MEM   freier Hauptspeicher in 4 KB-Blöcken
```

Bild 5.4: Online-Hilfe von GET

Die Aufrufe zur Abfrage der Parameter besitzen folgende Syntax:

```
GET Kommando
```

Der Parameter »Kommando« enthält ein Schlüsselwort zur Auswahl des auszuführenden Kommandos. Nachfolgend werden die einzelnen Kommandos besprochen.

Abfrage des Datum und der Zeit

Der GET-Befehl ermöglicht die Abfrage der Uhrzeit und des Datums aus einer Batchdatei heraus. Hierzu ist hinter GET der Name des betreffenden Parameters anzugeben. Der Wert wird dann über ERRORLEVEL zurückgegeben. Für GET sind die Aufrufe gemäß Tabelle 5.1 zulässig.

Kommando	Ergebnis
DAY	ermittelt den TAG 1-31
MONTH	ermittelt den Monat 1-1
YEAR	ermittelt das Jahr 00-99
WEEK	ermittelt den Wochentag
STD	ermittelt die Stunde 0-23

MIN	ermittelt die Minuten 0-59
SEC	ermittelt die Sekunden 0-59

Tabelle 5.1: Kommandos von GET

Der Code für den Wochentag ist gemäß der Aufstellung aus Tabelle 5.2 codiert:

Code	Wochentag
0	Sonntag
1	Montag
2	Dienstag
3	Mittwoch
4	Donnerstag
5	Freitag
6	Samstag

Tabelle 5.2: Codierung der Wochentage

Über die Datums- und Uhrzeitabfrage lassen sich Programme zu bestimmten Uhrzeiten aus einem Batchprogramm heraus starten. Weiterhin kann über eine Batchdatei der jeweilige Wochentag berechnet und ausgegeben werden.

Abfrage des freien DOS-Speichers

Mit dem Kommando:

```
GET /MEM
```

läßt sich der freie Speicher im 640-Kbyte-DOS-RAM abfragen. Der Befehl gibt eine Zahl in ERRORLEVEL zurück, die den freien Speicher in Einheiten zu 4 Kbyte spezifiziert. Zur Berechnung des freien Speichers ist der Wert aus ERRORLEVEL mit 4096 zu multiplizieren:

*Wert (Kbyte) = ERRORLEVEL * 4095*

Das Ergebnis entspricht im wesentlichen dem unter DOS für Programme verfügbaren Speicher. Das DOS-Kommando MEM wird zwar in der Regel geringfügig andere Werte liefern, die Abweichung ist aber tolerierbar. Um Ihnen eine Vorstellung von den Möglichkeiten des Programmes GET zur vermitteln, möchte ich eine kleine Anwendung vorstellen.

Automatischer Programmstart

In manchen Fällen ist es erwünscht, ein Programm zu einer bestimmten Uhrzeit oder zu einem bestimmten Datum automatisch zu starten. Denkbar ist zum Beispiel, daß ein System Nachts Daten aus einem Mailboxrechner abrufen soll. Unter DOS ist dies scheinbar nicht zu machen, da das Betriebssystem keine Zeitsteuerung besitzt. Hier muß vielmehr zur gewünschten Zeit das Programm explizit gestartet werden.

Mit GET läßt sich eine einfache Lösung entwickeln, die ein oder mehrere Programme zu einer bestimmten Uhrzeit startet. Es muß sich lediglich um ein selbst ablaufendes Programm handeln und der Rechner muß eingeschaltet bzw. das Batchprogramm START.BAT muß aktiv sein. Dieses Programm überprüft zyklisch die Uhrzeit und startet ein Programm, sobald diese Zeit erreicht ist. Nachfolgendes Listing zeigt den Aufbau des Moduls:

```
ECHO OFF
:=========================================
: File: START.BAT  (C) Born G. V 1.0
: Aufruf: START STD MIN Programm <Parameter>
: Das Programm überwacht die Uhrzeit und
: aktiviert ein Programm zu einer vorge-
: gebenen Uhrzeit.Als Parameter sind die
: Startzeit in Stunden und Minuten, sowie
: der Programmname zu übergeben.
:=========================================
IF '%1' == '' GOTO FEHLER
IF '%2' == '' GOTO FEHLER
IF '%3' == '' GOTO FEHLER
: Meldung, daß Programm aktiv ist
ECHO START ist aktiv, das Programm %3
ECHO wird um %1:%2 aktiviert
ECHO Abbruch durch Betätigung der ESC Taste
: Hilfsgrößen berechnen
XCALC %1 + 1
SET STEP=M1
SET VAR=STD1
GOTO ENV
:M1   Minuten berechnen
XCALC %2 + 1
SET STEP=LOOP
SET VAR=MIN1
GOTO ENV
:LOOP    prüfe ob die Stunde erreicht ist
GET /STD
: Stunde >= 1. Parameter ?
IF ERRORLEVEL %STD1% GOTO END0
IF ERRORLEVEL %1 GOTO NEXT
GOTO TASTE
:NEXT   prüfe ob die Minuten erreicht sind
GET /MIN
IF ERRORLEVEL %MIN1% GOTO END0
IF ERRORLEVEL %2 GOTO START
```

```
:TASTE   prüfe ob ESC gedrückt
VKEY *
IF ERRORLEVEL 28 GOTO LOOP
IF ERRORLEVEL 27 GOTO END
GOTO LOOP
:START   starte das Programm
ECHO Die Zeit:  %1:%2 Uhr wurde erreicht
ECHO Das Programm: %3  wird gestartet
:hier folgt die Anweisung zum Start des Programmes !!!!!
%3
GOTO END
:FEHLER
ECHO Falscher Aufruf, Parameter fehlt
ECHO Aufruf: START Std Min Programm
GOTO END
:END0
ECHO Die Startzeit ist bereits abgelaufen
GOTO END
:**************************************************
: "Unterprogramm" ENV, schreibt ERRORLEVEL in das
:  in das Environment
:        STEP = Marke Rücksprungadresse
:        VAR  = Name der Zielvariablen
:**************************************************
:ENV        ermittle Errorlevel
FOR %%a IN (0 1 2) DO IF ERRORLEVEL %%a00  SET $1=%%a
GOTO %$1%
:2     ERRORLEVEL 200 - 255
FOR %%a IN (0 1 2 3 4 5) DO IF ERRORLEVEL 2%%a0  SET $2=%%a
FOR %%a IN (0 1 2 3 4 5 6 7 8 9) DO IF ERRORLEVEL 2%$2%%a  SET
$3=%%a
: korrigiere Überlauf über 255
IF NOT '%$1%%$2%%$3%' == '259' GOTO SET_E
FOR %%a IN (0 1 2 3 4 5) DO IF ERRORLEVEL 2%$2%%a SET $3=%%a
GOTO SET_E
:1    100 - 199
:0     00 -  99
FOR %%a IN (0 1 2 3 4 5 6 7 8 9) DO IF ERRORLEVEL %$1%%a0  SET
$2=%%a
FOR %%a IN (0 1 2 3 4 5 6 7 8 9) DO IF ERRORLEVEL %$1%%$2%%a  SET
$3=%%a
:SET_E
SET %VAR%=%$1%%$2%%$3%
IF NOT '%$1%'== '0' GOTO OK
SET %VAR%=%$2%%$3%
IF NOT '%$2%'== '0' GOTO OK
SET %VAR%=%$3%
:OK
SET $1=
SET $2=
SET $3=
: Rücksprung zur angegebenen Marke
GOTO %STEP%
:END
SET STD1=
```

```
SET MIN1=
ECHO ON
```

Listing 5.6: Automatischer Programmstart

Das Programm ist mit den drei Parametern:

```
START Std Min Programm
```

aufzurufen. In »Std« ist die Startzeit in Stunden anzugeben. In »Min« steht die Startzeit in Minuten und der dritte Parameter enthält den Namen der auszuführenden Programmdatei. Die Startzeit von 1:09 muss damit folgendermaßen angegeben werden:

```
START 1 9 PROG1
```

Eine 0 vor einer Ziffer ist nicht zulässig. In obigen Beispiel wird das Programm »PROG1« um 1:09 gestartet. Anschließend bricht das Batchprogramm ab. Um das Batchprogramm innerhalb der Schleife abzubrechen, brauchen Sie nur die ESC-Taste zu drücken. Im Vorgriff wurden zur Abfrage die Programm VKEY und XCALC (siehe unten) benutzt.

Das Programm START.BAT benutzt noch einen weiteren Trick. Die ERRORLEVEL-Abfrage muß so formuliert werden, daß alle Zeiten oberhalb der Startzeit ausgeschlossen werden (>). Deshalb werden die Stunden- und Minutenwert sowie die um 1 erhöhten Werte benötigt. Die Berechnungen:

STD + 1 MIN + 1

lassen sich mit XCALC ausführen. Doch wie können die Ergebnisse zwischengespeichert werden? XCALC liefert alles als ERRORLEVEL zurück. Hier habe ich das Programm E.BAT so modifiziert, daß es den Wert aus ERRORLEVEL ermittelt und in einer Umgebungsvariablen ablegt. Leider kennt der DOS-Batchbefehlssatz keine Unterprogramme. Deshalb muß das »Unterprogramm« ENV mit GOTO-Sprüngen simuliert werden. In der Umgebungsvariablen SETP wird der Name einer Rücksprungadresse (z.B. LOOP) übergeben. Der Name ist vor dem Einsprung in ENV zu setzen. In der Umgebungsvariablen VAR muss dann noch der Name der Zielvariablen übergeben werden.

ENV ermittelt dann den Wert aus ERRORLEVEL speichert diesen in den angegebenen Umgebungsvariablen und springt zur angegebenen Marke zurück. (Die ist zwar von hinten durch die Brust ins Auge, funktioniert aber recht gut). Die Technik läßt sich zusammen mit XCALC sehr gut einsetzen, um Ergebnisse von Berechnungen zwischenzuspeichern. Wer sich mit der Benutzung von Umgebungsvariablen in Batchprogrammen nicht auskennt, sei wieder auf /5/ verwiesen, wo dieses Thema erschöpfend behandelt wird.

Die Implementierung

Die Implementierung von GET ist recht einfach. Im Hauptprogramm wird wieder die /?-Option in der Kommandozeile gesucht und gegebenenfalls die Online-Hilfe aktiviert. Andernfalls prüft GET mit den Anweisungen:

```
ptr%=INST(kommando$,"/....")
IF ptr% > 0 THEN CALL .....
```

das betreffende Kommando. Bei Datum und Uhrzeit wird der betreffende Parameter über die Unterprogramme GETDATE und GETTIME ermittelt.

Der freie Speicher ist über den Aufruf der Funktion 51H des INT 21 und über den INT 12 zu ermitteln. Hintergrundinformationen zu diesem Thema finden sich im folgenden Kapitel im Zusammenhang mit dem Programm HWINFO.BAS.

getdateDas Unterprogramm ermittelt das aktuelle Datum über die DATE$-Funktion und gibt den über *Nr%* bezeichneten Parameter (Tag, Monat, Jahr, Wochentag) an DOS als Fehlercode zurück. Dieser Parameter läßt sich über ERRORLEVEL auswerten. Der Wochentag muß direkt durch Aufruf der Funktion 2AH des INT 21 (siehe /1/) ermittelt werden. Die Funktion gibt den Wert in AL zurück.

gettimeDas Unterprogramm ermittelt die aktuelle Zeit über die TIME$-Funktion und gibt den über *Nr%* bezeichneten Parameter (Sek., Min., Std.) an DOS als Fehlercode zurück. Dieser Parameter läßt sich über ERRORLEVEL auswerten.

Erweiterungsvorschläge

In /5/ findet sich das Programm CHECK, welches die Funktionalität von GET stark erweitert. Es erlaubt zum Beispiel die Abfrage der Versionsnummer von DOS, die Ermittlung des aktuellen Laufwerkes, der Plattenkapazität, der Bildschirmmodi etc. Bei Bedarf können Sie diese Funktionen in GET ebenfalls nachrüsten.

```
X R E F   /Z=50                              (c) Born Version 1.0
Datei : get.bas      Datum : 06-03-1992         Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : GET.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 26.5.92
      '! Autor      : G. Born
      '! Progr. Spr.: PowerBASIC
      '! Betr. Sys. : DOS 2.1 - 5.0 (DR-DOS 5.0/6.0)
      '! Funktion: Das Programm wird mit der Eingabe:
      '!
      '!           GET <Option>
      '!
```

```
      '!           aufgerufen. Es erlaubt die Abfrage verschiedener
      '!           Systemparameter und gibt die Werte als Error-
      '!           code an DOS zurück. Das Ergebnis läßt sich über
      '!           ERRORLEVEL auswerten.
      '!
      '!             IF ERRORLEVEL 3 ...
      '!
      '*************************************************************
      '! Variable definieren
    1 ptr% = 0
    2 kommando$ = ""

      '############################################################
      '#                       Hauptprogramm                      #
      '############################################################

    3 kommando$ = UCASE$(COMMAND$)         '! Parameter ?

    4 ptr% = INSTR (kommando$,"/?")        '! Option /?
    5 IF ptr% <> 0 THEN                    '! Hilfsbildschirm
    6   PRINT "G E T                        (c) Born Version 1.0"
    7   PRINT
    8   PRINT "Aufruf: GET  <Option>"
    9   PRINT
   10   PRINT "Erlaubt die Abfrage bestimmter Parameter"
   11   PRINT "GET /DAY   ermittelt den Tag (1-31)"
   12   PRINT "GET /MONTH ermittelt den Monat (1-12)"
   13   PRINT "GET /YEAR  ermittelt das Jahr (0-99)"
   14   PRINT "GET /WEEK  ermittelt den Wochentag (0-6)"
   15   PRINT "GET /SEC   ermittelt die Sekunden (0-59)"
   16   PRINT "GET /MIN   ermittelt die Minuten (0-59)"
   17   PRINT "GET /STD   ermittelt die Stunden (0-23)"
   18   PRINT "GET /MEM   freier Hauptspeicher in 4 KB-Blöcken"
   19   PRINT
   20   PRINT "Die Fehlercodes lassen sich durch ERRORLEVEL
auswerten."
   21   PRINT
   22   SYSTEM
   23 END IF

      '! Sprungverteiler für die einzelnen Kommandos
   24 ptr% = INSTR (kommando$,"/DAY")       '! Option /DAY
   25 IF ptr% > 0 THEN CALL GETDATE (1)

   26 ptr% = INSTR (kommando$,"/MONTH")     '! Option /MONTH
   27 IF ptr% > 0 THEN CALL GETDATE (2)

   28 ptr% = INSTR (kommando$,"/YEAR")      '! Option /YEAR
   29 IF ptr% > 0 THEN CALL GETDATE (3)

   30 ptr% = INSTR (kommando$,"/WEEK")      '! Option /Wochentag
   31 IF ptr% > 0 THEN CALL GETDATE (4)

   32 ptr% = INSTR (kommando$,"/SEC")       '! Option /SEC
   33 IF ptr% > 0 THEN CALL GETTIME(1)
```

```
   34 ptr% = INSTR (kommando$,"/MIN")       '! Option /MIN
   35 IF ptr% > 0 THEN CALL GETTIME(2)

   36 ptr% = INSTR (kommando$,"/STD")       '! Option /STD
   37 IF ptr% > 0 THEN CALL GETTIME(3)

   38 ptr% = INSTR (kommando$,"/MEM")       '! Option /MEM
   39 IF ptr% > 0 THEN
   40  REG 1, &H5100                         '! AX = 5100 -> read
                                             '! PSP Segm. Adr
   41  CALL INTERRUPT &H21                   '! Dispatcher INT
   42  IF (REG (0) AND &H01) > 0 THEN        '! Fehler ?
   43   END (0)                              '! Fehlercode
   44  ELSE
   45   start& = REG(2)                      '! merke Adresse
   46  END IF

   47  CALL INTERRUPT &H12                   '! BIOS: GET RAM SIZE
   48  ram& = REG(1)                         '! RAM Größe
   49  ram& = ram& * &H3FF
   50  frei& = ram& - (start& * 16)          '! berechne. freie
Größe
   51  END (frei& / 4096)                    '! Korrektur 4 Kbyte
Block
   52 END IF

   53 END                                   '! Ende

      '############################################################
      '#                    Hilfsroutinen                         #
      '############################################################

   54 SUB GETDATE (Nr%)
      '!----------------------------------------------------------
      '! Ermittelt das Datum und gibt es als Fehlercode an DOS
      '! 1 = Tag, 2 = Monat, 3 = Jahr, 4 = Wochentag
      '!----------------------------------------------------------

   55  LOCAL dat$

   56  SELECT CASE Nr%

   57   CASE 1
   58    dat$ = MID$(DATE$,1,2)              '! Tag ermitteln
   59    END (VAL(dat$))

   60   CASE 2
   61    dat$ = MID$(DATE$,4,2)              '! Monat ermitteln
   62    END (VAL(dat$))

   63   CASE 3
   64    dat$ = MID$(DATE$,9,2)              '! Jahr ermitteln
   65    END (VAL(dat$))
```

```
   66   CASE 4
   67    REG 1, &H2A00                          '! INT 21, 2AH GET
DATE
   68    CALL INTERRUPT &H21                    '! Wochentag in AL
   69    END (REG(1) AND &HFF)

   70   END SELECT
   71 END SUB


   72 SUB GETTIME (Nr%)
      '!-------------------------------------------------------
      '! Ermittelt die Zeit und gibt sie als Fehlercode an DOS
      '! 1 = Sek., 2 = Min. 3 = Std.
      '!-------------------------------------------------------

   73  LOCAL tim$

   74  SELECT CASE Nr%

   75   CASE 1
   76    tim$ = MID$(TIME$,7,2)                 '! Sek. ermitteln
   77    END (VAL(tim$))

   78   CASE 2
   79    tim$ = MID$(TIME$,4,2)                 '! Min. ermitteln
   80    END (VAL(tim$))

   81   CASE 3
   82    tim$ = MID$(TIME$,1,2)                 '! Std. ermitteln
   83    END (VAL(tim$))

   84   END SELECT
   85 END SUB

      '***** Programm Ende ******
```

Listing 5.7: GET.BAS

FKEY: Abfragen von Funktionstasten aus Batchprogrammen

Das Modul ASK bietet keine Möglichkeit zur Abfrage von Funktionstasten aus Batchdateien. Die Ursache liegt in der zurückgelieferten Codefolge (2 Byte). Deshalb möchte ich noch ein kurzes Programm vorstellen, welches explizit die Abfrage von Funktionstasten aus Batchdateien erlaubt.

Der Entwurf

Als erstes möchte ich auf die Anforderungen an das Programm eingehen. Das Programm FKEY erlaubt es, innerhalb einer Stapeldatei Funktionstasten abzufragen. Es besitzt dabei folgende Aufrufstruktur:

```
FKEY <Text>
```

Wird ein <*Text*> in der Kommandozeile angegeben, erscheint dieser nach dem Aufruf von FKEY auf dem Bildschirm, unabhängig von der ECHO-Einstellung (ECHO ON/ECHO OFF) der Batchdatei. Der Text darf beliebige Zeichen enthalten und ist durch ein Leerzeichen vom Programmnamen zu trennen. Damit erhalten Sie auch hier die Möglichkeit, beim Ablauf der Batchdatei eine Nachricht an den Benutzer zu senden:

```
FKEY Bitte betätigen Sie eine Funktionstaste
```

Bei einem Aufruf ohne Parameter entfällt die Textausgabe auf dem Bildschirm. FKEY wartet anschließend auf eine Benutzereingabe von der Tastatur. Der betreffende Tastaturcode wird dann an DOS übergeben und läßt sich über die ERRORLEVEL-Funktion aus Batchdateien abfragen. Hierbei gelten die gleichen Bedingungen für die Formulierung der Abfrage wie bei dem Programm ASK.

Weiterhin soll sich das Programm mit der Option:

```
FKEY /?
```

aktivieren lassen. Dann erscheint eine Anzeige mit folgenden Erläuterungen auf dem Bildschirm:

```
F K E Y                                (C) Born G. Version 1.0
Aufruf:   FKEY <Text>

Das Programm erlaubt Abfragen von Funktionstasten
aus Batchdateien und gibt die Codes als Fehlercode an
DOS zurück. Die Fehlercodes lassen sich durch ERRORLEVEL
auswerten. Das Feld <Text> ist optional und erlaubt die
Angabe einer Benutzermeldung, die beim Aufruf erscheint.
```

Bild 5.5: Meldung des Programmes FKEY

Das Programm FKEY gibt nach Betätigung einer Funktionstaste das zweite Byte des *Extended-ASCII-Codes* zurück. Bei normalen Tasten wird dagegen der Wert 255 in ERRORLEVEL gespeichert. Tabelle 5.3 gibt eine grobe Übersicht der Codes der einzelnen Funktionstasten.

Taste	Code
F1	59
F2	60
F3	61
F4	62
F5	63
F6	64

F7	65
F8	66
F9	67
F10	68

Tabelle 5.3: Codes der Funktionstasten

Weitere Tastencodes lassen sich der Dokumentation der Tastatur (siehe PowerBASIC-Handbuch) entnehmen.

Ein Beispielprogramm

Nachfolgendes Beispiel demonstriert die Verwendung von FKEY in einer Batchdatei. Hierbei wurde das Programm MENU.BAT an die Anforderungen von FKEY angepaßt. Nach dem Aufruf des Batchprogrammes erscheint folgende Maske auf dem Bildschirm:

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º ºººººººººººººº   M  E  N  Ü    ºººººººººººººººº º
ÌÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¹
º  ***    P r o g r a m m a u s w a h l      ***   º
º                                                   º
º                 F1 Ende                           º
º                 F2 Textverarbeitung               º
º                 F3 Tabellenkalkulation            º
º                 F4 Datenbank                      º
º                 F5 ---                            º
º                                                   º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼

Bitte eine Funktionstaste betätigen:
```

Bild 5.6: Menüsteuerung mit FKEY

Der Benutzer kann nun die verschiedenen Programme durch Eingabe einer Funktionstaste aktivieren. Bei ungültigen Eingaben wird die Maske aufgefrischt und die Abfrage erscheint erneut. Das Programm läßt sich durch die Eingabe der Taste F1 beenden. Das zugehörige Batchprogramm besitzt dabei folgenden Aufbau:

```
ECHO OFF
:===============================================
: Programm:  MENU1.BAT
: Version:   1.0  (25.5.92) (c) Born
: Funktion:  Demonstration einer Menüverwaltung
:            unter Verwendung von FKEY
:===============================================
: Maske aufbauen
:LOOP
CLS
```

```
ECHO ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
ECHO º ºººººººººººººº    M  E  N  Ü    ºººººººººººººººº º
ECHO ÌÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¹
ECHO º  ***     P r o g r a m m a u s w a h l      ***   º
ECHO º                                                    º
ECHO º                 F1 Ende                            º
ECHO º                 F2 Textverarbeitung                º
ECHO º                 F3 Tabellenkalkulation             º
ECHO º                 F4 Datenbank                       º
ECHO º                 F5 ---                             º
ECHO º                                                    º
ECHO ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
ECHO.
:
: Hier wird FKEY zur Benutzerabfrage verwendet
:
FKEY Bitte eine Funktionstaste betätigen:
:
: Auswertung der Benutzereingaben
: werte zuerst immer die höheren Codes aus
: Zahl F1 -> Code = 59, F2 = 60, usw.
:
IF ERRORLEVEL 63 GOTO LOOP
IF ERRORLEVEL 62 GOTO L3
IF ERRORLEVEL 61 GOTO L2
IF ERRORLEVEL 60 GOTO L1
IF ERRORLEVEL 59 GOTO Exit
GOTO loop
:L1      Aufruf der Textverarbeitung
CD TEXTE
WORD
CD..
GOTO loop
:L2      Aufruf der Tabellenkalkulation
CD 123
LOTUS
CD..
GOTO loop
:L3      Aufruf des Datenbankprogrammes
CD DBASE
DBASE
CD..
GOTO loop
:Exit
ECHO ON
```

Listing 5.8: Batchprogramm zur Menüsteuerung

Einzige Änderungen an dem Programm gegenüber MENU.BAT sind die Texte der Maske sowie die Codes zur Abfrage der Funktionstasten.

Die von den Funktionstasten zurückgelieferten Codes lassen sich übrigens mit dem Programm E.BAT ermitteln. Wird FKEY über das Modul aktiviert, zeigt E.BAT anschließend den DOS-ERRORLEVEL-Code im Klartext auf

dem Bildschirm an. Damit läßt sich recht einfach feststellen, welche Funktionstaste welchen Code liefert.

Die Implementierung

Doch nun möchte ich auf die Implementierung zu sprechen kommen. Das Programm entspricht vom Aufbau dem Modul ASK, so daß auf ein Hierarchiediagramm verzichtet wird. Es besteht nur aus einem Hauptmodul. Findet sich in der Kommandozeile der Parameter */?*, gibt FKEY den Hilfebildschirm aus und bricht (mit dem Fehlercode 0) ab.

Ohne Option wird der eventuell in der Kommandozeile vorhandene Text mittels der PowerBASIC-Funktion COMMAND$ ermittelt und per PRINT-Kommando ausgegeben. Liegt kein Text vor, gibt COMMAND$ einen Leerstring zurück.

Dann wird die Tastatur mit INKEY$ gelesen. Liegen keine Zeichen im Tastaturpuffer vor, wartet das Programm in einer Eingabeschleife. Bei normalen Tasteneingaben endet das Programm mit dem Code 255. Bei Funktionstasten wird das 2. Codebyte als Code an DOS zurückgegeben. Einzelheiten sind dem nachfolgenden Listing zu entnehmen.

Erweiterungsvorschläge

Das Programm FKEY kann so optimiert werden, daß nur bestimmte Funktionstasten (z.B. F1, F2 etc.) akzeptiert werden. Dann kann die Auswertung in einer Batchdatei etwas einfacher gestaltet werden.

```
X R E F   /Z=50                              (c) Born Version 1.0
Datei : fkey.bas      Datum : 06-02-1992         Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : FKEY.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 26.5.92
      '! Autor      : G. Born
      '! Progr. Spr.: PowerBASIC
      '! Betr. Sys. : DOS 2.1 - 5.0 (DR-DOS 5.0/6.0)
      '! Funktion: Das Programm wird mit der Eingabe:
      '!
      '!           FKEY <Text>
      '!
      '!           aufgerufen. Es erlaubt die Abfragen von
Funktions-
      '!           tasten aus Batch-Dateien. Der <Text> wird auf
dem
      '!           Bildschirm ausgegeben. Dann wartet ASK auf die
      '!           Eingabe. Das Ergebnis wird als Errorcode an DOS
      '!           übergeben und läßt sich über ERRORLEVEL
      '!
```

```
      '!             IF ERRORLEVEL 3 ...
      '!
      '!           abfragen.
      '*********************************************************
      '! Variable definieren
    1 zchn$ = ""
    2 ptr% = 0
    3 kommando$ = ""

      '#############################################################
      '#                        Hauptprogramm                      #
      '#############################################################

    4 kommando$ = COMMAND$                  '! Parameter ?

    5 ptr% = INSTR (kommando$,"/?")         '! Option /?
    6 IF ptr% <> 0 THEN                     '! Hilfsbildschirm
    7   PRINT "F K E Y                        (c) Born Version 1.0"
    8   PRINT
    9   PRINT "Aufruf: FKEY <Text>"
   10   PRINT
   11   PRINT "Das Programm erlaubt die Abfragen von
Funktionstasten"
   12   PRINT "aus Batchdateien und gibt die Codes als Fehlercode
an"
   13   PRINT "DOS zurück. Die Fehlercodes lassen sich durch
ERRORLEVEL"
   14   PRINT "auswerten. Das Feld <Text> ist optional und erlaubt
die"
   15   PRINT "Angabe einer Benutzermeldung, die beim Aufruf
erscheint."
   16   PRINT
   17   SYSTEM
   18 END IF

   19 PRINT kommando$;" ";                   '! Text ausgeben
   20 zchn$ = INKEY$                         '! Tastatur abfragen
   21 WHILE (LEN(zchn$) = 0)                 '! Leereingabe
   22  zchn$ = INKEY$                        '! Tastatur abfragen
   23 WEND

   24 IF LEN(zchn$) = 1 THEN
   25  END (255)                             '! keine
Funktionstaste
   26 ELSE
   27  END (ASC(MID$(zchn$,2,1)))            '! Tastencode
   28 END IF

   29 END ASC(zchn$)                         '! Ende

      '***** Programm Ende ******
```

Listing 5.9: FKEY.BAS

XCALC: Berechnungen in Batchdateien

Eine weitere Sache, die ich bisher vermisse, ist die Möglichkeit von Berechnungen in Batchprogrammen. In /5/ habe ich daher ein entsprechendes Modul vorgestellt, welches gänzlich neue Möglichkeiten eröffnet (Schleifen, Berechnungen, etc.). Aus diesem Ansatz heraus wurden die Funktionen (mit einer kleinen Einschränkung) nach PowerBASIC portiert. Die Einschränkung besteht darin, daß das Programm die Ergebnisse an DOS als Fehlercode zurückgibt, also auf den Zahlenbereich zwischen 0 und 255 begrenzt ist. Trotzdem ergeben sich damit interessante Möglichkeiten.

Die Anforderungen

Der XCALC-Befehl erlaubt Berechnungen innerhalb eines Batchprogrammes. Die Ergebnisse werden dabei als Fehlercode an DOS zurückgegeben und lassen sich per ERRORLEVEL abfragen. Mit dem Aufruf:

```
XCALC /?
```

läßt sich die Online-Hilfe aktivieren.

Allgemein besitzt XCALC jedoch folgende Aufrufsyntax:

```
XCALC Ausdruck
```

Der Parameter *Ausdruck* ist zwingend vorgeschrieben und nimmt den zu berechnenden Ausdruck auf. XCALC beherrscht nur die vier Grundrechenarten:

```
+  Addition
-  Subtraktion
*  Multiplikation
/  Division
```

Dabei wird die Punktrechnung vor der Strichrechnung ausgeführt. Klammern sind in dem Ausdruck nicht erlaubt. Gültige Ausdrücke sind dann zum Beispiel:

```
3 + 15 * 2 - 10
3 * 4 /  3 + 4
-5 - 17
13 - -2
```

Als Werte lassen sich vorzeichenbehaftete Ganzzahlen verwenden. Die Ergebnisse dürfen im Zahlenbereich von 0 bis 255 liegen. Ist das Ergebnis einer Operation größer als 255 oder kleiner 0, tritt ein Überlauf auf und das Programm bricht mit folgender Meldung ab:

```
Bereichsüberlauf - Wert : xxxx
```

An DOS wird der Fehlercode 255 zurückgeliefert. Dies ist bei der Verwendung von CALC zu beachten. Ungültige Ausdrücke sind zum Beispiel:

```
255 + 1          ! führt zum Überlauf
100 * 3          ! führt zum Überlauf
```

Ähnliches gilt, falls ein ungültiger Ausdruck als Parameter angegeben wird. Dann erscheint die Meldung:

```
Fehler im Ausdruck: xxxxxxx
```

und der Code 255 wird zurückgegeben.

Interessant ist die Möglichkeit, als Ausdruck den Inhalt einer Umgebungsvariablen anzugeben. Mit:

```
XCALC %a% + 1
```

wird zur Laufzeit des Batchprogrammes der Inhalt der Umgebungsvariablen *a* eingesetzt. Steht dort zum Beispiel die Zahl 3, wird diese in die Berechnung eingebracht.

```
ECHO OFF
:=========================================
: File: LOOP.BAT  (C) Born G. V 1.0
: Aufruf: START
: Das Programm demonstriert die Verwendung
: von Schleifen in Batchprogrammen.
:=========================================
: Meldung, daß Programm aktiv ist
ECHO LOOP wird nun aktiv und beginnt mit der Schleife
SET C1=1
:LOOP    Beginn der Schleife
ECHO Schleifenzähler = %C1%
:
: Schleifenzähler erhöhen
XCALC %C1% + 1
SET STEP=M1
SET VAR=C1
GOTO ENV
:M1 Ende erreicht?
IF ERRORLEVEL 10 GOTO EXIT
GOTO LOOP
:************************************************
: "Unterprogramm" ENV, schreibt ERRORLEVEL in das
:  in das Environment
:        STEP = Marke Rücksprungadresse
:        VAR  = Name der Zielvariablen
:************************************************
:ENV        ermittle Errorlevel
FOR %%a IN (0 1 2) DO IF ERRORLEVEL %%a00  SET $1=%%a
GOTO %$1%
:2     ERRORLEVEL 200 - 255
FOR %%a IN (0 1 2 3 4 5) DO IF ERRORLEVEL 2%%a0  SET $2=%%a
```

```
FOR %%a IN (0 1 2 3 4 5 6 7 8 9) DO IF ERRORLEVEL 2%$2%%%a  SET
$3=%%a
: korrigiere Überlauf über 255
IF NOT '%$1%%$2%%$3%' == '259' GOTO SET_E
FOR %%a IN (0 1 2 3 4 5) DO IF ERRORLEVEL 2%$2%%%a SET $3=%%a
GOTO SET_E
:1    100 - 199
:0     00 -  99
FOR %%a IN (0 1 2 3 4 5 6 7 8 9) DO IF ERRORLEVEL %$1%%%a0  SET
$2=%%a
FOR %%a IN (0 1 2 3 4 5 6 7 8 9) DO IF ERRORLEVEL %$1%%$2%%%a  SET
$3=%%a
:SET_E
SET %VAR%=%$1%%$2%%$3%
IF NOT '%$1%'== '0' GOTO OK
SET %VAR%=%$2%%$3%
IF NOT '%$2%'== '0' GOTO OK
SET %VAR%=%$3%
:OK
SET $1=
SET $2=
SET $3=
: Rücksprung zur angegebenen Marke
GOTO %STEP%
:EXIT
SET C1=
ECHO ON
```

Listing 5.10: Schleifen in Batchprogrammen

Die Implementierung

Bei der Implementierung kann auf die Überlegungen bezüglich des Moduls CALC zurückgegriffen werden. Lediglich folgende Unterschiede sind zu beachten:

- Die Eingaben werden aus der Kommandozeile übernommen.
- Nach einer Berechnung wird das Ergebnis als Errorcode an DOS zurückgegeben und XCALC endet.
- Auf die Unterstützung verschiedener Zahlensystem kann bei XCALC verzichtet werden.

Nachfolgendes Bild gibt das Hierarchiediagramm von XCALC wieder.

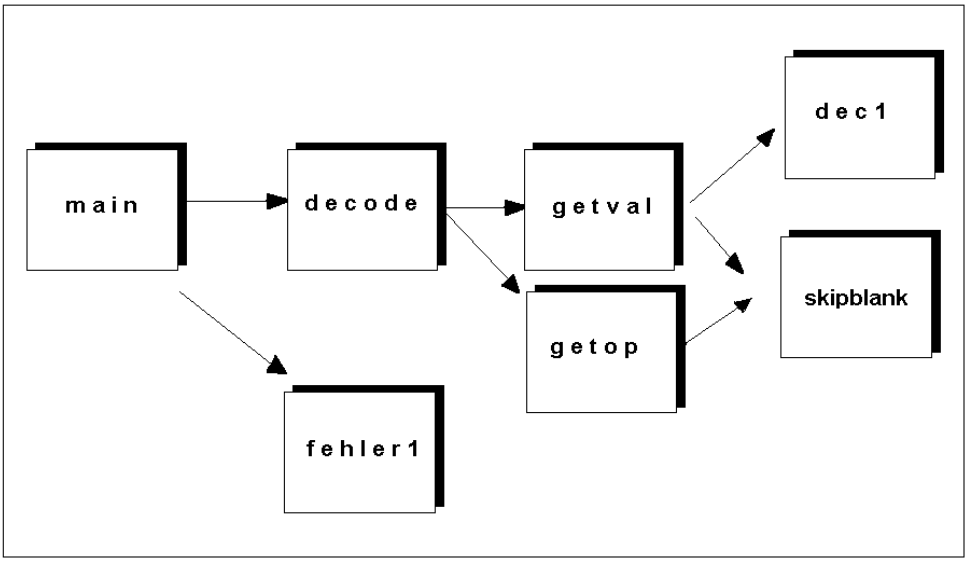


Bild 5.7: Hierarchiediagramm von XCALC

Die Beschreibung der einzelnen Module findet sich im Kapitel 3 (CALC). Aus diesem Programm wurden die Module *decode* zur Dekodierung der Eingaben, *getval* zur Separierung der Terme, *getop* zur Separierung der Operatoren und *skipblank* zum Entfernen der Leerzeichen übernommen. Das Unterprogramm *dec1* wandelt die Strings in Dezimalzahlen um. Weitere Einzelheiten sind dem Listing und der Beschreibung des Programmes CALC zu entnehmen.

```
X R E F   /Z=50                              (c) Born Version 1.0
Datei : xcalc.bas      Datum : 06-02-1992         Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : XCALC.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 16.5.92
      '! Autor      : G. Born
      '! Progr. Spr.: PowerBASIC
      '! Betr. Sys. : DOS 2.1 - 5.0 (DR-DOS 5.0/6.0)
      '! Funktion: Das Programm wird mit der Eingabe:
      '!
      '!           XCALC <Ausdruck>
      '!
      '!           aufgerufen. Es liest den Ausdruck ein und
berechnet
      '!           das Ergebnis (muß zwischen 0 und 255 liegen) und
      '!           gibt dieses in ERRORLEVEL zurück.
      '!
      '!             XCALC 3+5
```

```
      '!
      '!           Damit lassen sich Berechnungen in Batchdateien
      '!           ausführen.
      '*********************************************************
      '! Variable definieren
      '! globale Konstanten
    1  %true = -1: %false = 0
    2  %add = 1: %sub = 2: %mul = 3: %div = 4'! code für
Operationen
      '! globale Variablen
    3  %maxentry = 10
    4  DIM wertx%(1:%maxentry)               '! Speicher für 10
Parameter
    5  DIM opc%(1:%maxentry)                 '!     "     "  10
Operatoren
    6  wert% = 0                             '! Ergebnis
    7  errx% = 0                             '! Fehlernummer
    8  count% = 0                            '! Zahl der Parameter

    9  lang% = 0                             '! Länge Eingabetext
   10  text$ = ""                            '! Eingabetext

   11  ptr% = 0

      '#############################################################
      '#                       Hauptprogramm                       #
      '#############################################################

   12  ON ERROR GOTO fehler1

   13  text$ = COMMAND$                     '! Parameter ?

   14  ptr% = INSTR (text$,"/?")            '! Option /?
   15  IF ptr% <> 0 THEN                    '! Hilfsbildschirm
   16   PRINT "X C A L C                    (c) Born Version 1.0"
   17   PRINT
   18   PRINT "Aufruf: XCALC <Ausdruck>"
   19   PRINT
   20   PRINT "Das Programm berechnet den angegebenen Ausdruck und"
   21   PRINT "gibt das Resultat als Fehlercode an DOS zurück."
   22   PRINT "Die Ergebnisse dürfen zwischen 0 und 255 liegen und"
   23   PRINT "lassen sich mit ERRORLEVEL auswerten. Beispiel:"
   24   PRINT
   25   PRINT "XCALC 3 + 5"
   26   PRINT
   27   SYSTEM
   28  END IF

   29  ptr% = 1
   30  lang% = LEN(text$)                        '! Länge
Parameterstring

   31  CALL decode (text$)                       '! berechne
Ergebnis
```

```
32  IF errx% <> 0 THEN                          '! Fehlerabbruch
33   PRINT "Fehler im Ausdruck : ";text$
34   END 255
35  END IF

36  IF (wert% > 255) OR (wert% < 0) THEN
37   PRINT "Bereichsüberlauf Wert : ";wert%    '! Fehlerexit
38   END (255)
39  ELSE
40   END (wert%)
41  END IF

42 END                                           '! Ende

   '#############################################################
   '#                     Hilfsroutinen                         #
   '#############################################################

43 SUB decode (text$)

   '!-----------------------------------------------------------
   '! bearbeite Eingabe und berechne Ergebnis
   '!-----------------------------------------------------------
44 LOCAL ptr%, l%, flag%

45 SHARED lang%, count%, errx%, wert%
46 SHARED wertx%(), opc%()

47  ptr% = 1: errx% = 0: count% = 1     '! init lokale variable
48  WHILE ptr% <= lang%                 '! scan String
49   CALL getval (ptr%,wertx%(count%))  '! ermittle 1. Parameter
50   IF errx% > 0 THEN EXIT SUB         '! Error Exit
51   ' IF wertx%(count%) = 0 THEN GOTO ready '! WHILE EXIT
52   CALL getop (ptr%,opc%(count%))     '! ermittle 1. Operator
53   IF errx% > 0 THEN EXIT SUB         '! Error Exit
54   INCR ptr%                          '! hinter Operator
55   INCR count%                        '! nächste Zelle
56  WEND

57  DECR count%                         '! Zahl der Werte

58 ready:
   '! nur 1 Parameter gefunden, oder Leereingabe ?
59  IF count% < 2 THEN
60   wert% = wertx%(1)
61   EXIT SUB
62  END IF

   '! Punktrechnung "*"  "/" vorziehen !!!
63  FOR l% = 1 TO count%-1
64   IF opc%(l%) = %mul THEN
65    wertx%(l%+1) = wertx%(l%) * wertx%(l%+1)
66   ELSE
67    IF opc%(l%) = %div THEN
68     wertx%(l%+1) = INT(wertx%(l%) / wertx%(l%+1))
```

```
   69    END IF
   70   END IF
   71  NEXT l%

      '! Strichrechnung "+"  "-" nachziehen !!!
   72  FOR l% = 1 TO count%-1
   73   WHILE (opc%(l%) = %mul) OR (opc%(l%) = %div) '! skip A * B
+ ...
   74    INCR l%
   75   WEND
   76   j% = l%+1: flag% = %false              '! clear gefunden
   77   WHILE (opc%(j%) = %mul) OR (opc%(j%) = %div) '! skip A + B
* C ...
   78    INCR j% : flag% = %true               '! setze gefunden
   79   WEND

   80   IF opc%(l%) = %add THEN
   81    wertx%(j%) = wertx%(l%) + wertx%(j%)
   82   ELSE
   83    IF opc%(l%) = %sub THEN
   84     wertx%(j%) = wertx%(l%) - wertx%(j%)
   85    END IF
   86   END IF
   87   IF flag% THEN l% = j%-1
   88  NEXT l%

      ' FOR n% = 1 TO count%
      '  PRINT "n= ";n%;" wert ";wertx%(n%);" opc "; opc%(n%)
      ' NEXT n%

   89  wert% = wertx%(count%)                  '! Endergebnis

   90 END SUB

   91 SUB getval (ptr%, wert%)

      '!-----------------------------------------------------------
      '! lese eine Zahl ein und decodiere sie
      '!-----------------------------------------------------------

   92 SHARED text$, errx%, count%, lang%, vorz%
   93 LOCAL tmp%, zchn$, first%, last%

   94  vorz% = 1 : tmp% = 0                '! init Hilfsvariablen

      '! suche Anfang und Ende der Zahl
   95  CALL skipblank  (ptr%,text$)        '! skip führende blanks

      '! Vorzeichen bearbeiten
   96  zchn$ = MID$ (text$,ptr%,1)         '! hole Zeichen

   97  IF (zchn$ = "-") THEN
   98   vorz% = -1                         '! negative Zahl
   99   INCR ptr%
  100  ELSE
```

```
101   IF (zchn$ = "+") THEN              '! Vorz. überlesen
102    INCR ptr%
103   END IF
104  END IF
105  zchn$ = MID$ (text$,ptr%,1)         '! hole Zeichen
106  first% = ptr%                       '! merke Anfang Zahl

    '! suche Ende der Zahl = " ";"+";"-";"*";"/"
107  WHILE (INSTR(" +-*/",zchn$) = 0) AND ptr% <= lang%
108   INCR ptr%                          '! hole nächstes Zeichen
109   zchn$ = MID$ (text$,ptr%,1)        '! hole Zeichen
110  WEND
    '! merke Ende der Zahl
111  IF ptr% < lang% THEN
112   last% = ptr% - 1                   '! auf letzte Ziffer
113  ELSE
114   last% = lang%                      '! auf Textende
115  END IF

116  CALL dec1 (first%, last%, wert%)    '! Zahl decodieren

117 END SUB

118 SUB getop (ptr%,opcode%)

    '!-------------------------------------------------------
    '! ermittle operator (+ - * / )
    '!-------------------------------------------------------

119 SHARED errx%, text$, lang%
120 LOCAL  zchn$, tmp%

121  CALL skipblank (ptr%,text$)         '! überlese blanks

122  IF ptr% >= lang% THEN
123   opcode% = 0                        '! nichts gefunden
124   EXIT SUB
125  END IF

126  zchn$ = MID$(text$,ptr%,1)          '! hole Zeichen
127  tmp% = INSTR ("+-*/",zchn$)         '! decodiere Operator

128  SELECT CASE tmp%                    '! Zuweisung Opcode

129  CASE 1
130   opcode% = %add                     '! Addition

131  CASE 2
132   opcode% = %sub                     '! Subtraktion

133  CASE 3
134   opcode% = %mul                     '! Multiplikation

135  CASE 4
136   opcode% = %div                     '! Division
```

```
  137  CASE ELSE
  138   errx% = 3                          '! ungültiger Operator
  139   opcode% = 0

  140  END SELECT

  141 END SUB

  142 SUB skipblank(ptr%,text$)
      '-------------------------------------------------------
      '! zähle führende Blanks in einer Zeichenkette
      '! text$ = Zeichenkette, zeiger% = Zeiger in Kette
      '-------------------------------------------------------
  143 SHARED lang%

  144 WHILE (ptr% =< lang%) and (MID$(text$,ptr%,1) = " ")
  145  INCR ptr%
  146 WEND
  147 END SUB


  148 SUB dec1 (first%,last%,wert%)

      '!-------------------------------------------------------
      '! decodieren der Dezimalzahl
      '!-------------------------------------------------------

  149 SHARED text$, vorz%, errx%
  150 LOCAL tmp%, b%

  151  wert% = 0                         '! init

  152  FOR b% = first% TO last%          '! alle Ziffern
  153   zchn$ = MID$(text$,b%,1)         '! hole Ziffer

  154   '! Achtung VAL funktioniert nicht, da 0 bei Fehler
geliefert wird
  155   tmp% = INSTR ("0123456789",zchn$) '! Wert der Ziffer

      '! gültige Ziffer ????
  156   IF tmp% = 0 THEN
  157    errx% = 1                       '! Fehlerexit
  158   ELSE
  159    tmp% = tmp% - 1                 '! Wert korrigieren
  160    wert% = wert% * 10 + (tmp% * vorz%) '! Ziffer auf Zahl
addieren
  161   END IF
  162  NEXT b%

  163 END SUB

  164 fehler1:
      '!-------------------------------------------------------
      '! Abfangroutine für PowerBASIC Fehler
```

```
    '!---------------------------------------------------------

 165 IF ERR = 6 THEN
 166  PRINT "Overflow Error";                '! Fehlermeldung
 167 ELSE
 168  PRINT "Fehler : ";ERR;
 169 END IF

 170 END
 171 RETURN

    '!***** Programm Ende ******
```

Listing 5.11: XCALC.BAS

WAIT: Zeitverzögerung in Batchprogrammen

WAIT ist ein weiteres Programm, welches sich auf der beiliegenden Diskette befindet. Es ermöglicht, den Ablauf eines Batchprogrammes für eine gewisse Zeit zu unterbrechen. Während bei ASK und PAUSE der Ablauf bis zur ersten Benutzereingabe ruht, gibt WAIT die Kontrolle nach einer gewissen Zeitdauer wieder an DOS zurück und der Batchjob kann fortgesetzt werden. Mit:

```
WAIT /?
```

wird die Online-Hilfe aktiviert. Allgemein gilt folgende Aufrufform:

```
WAIT Time
```

Mit *Time* ist die Verzögerungszeit in Sekunden zu definieren. Hier lassen sich auch Kommazahlen mit angeben. Falls dieser Parameter fehlt oder falsch angegeben wurde, erscheint eine Fehlermeldung:

```
Zeitangabe ungültig
```

und der Fehlercode 255 wird an DOS übergeben. Gültige Werte für die Verzögerungszeit sind zum Beispiel:

```
WAIT 1
WAIT 0.5
WAIT 20
```

Die Zeit kann allerdings von System zu System leicht variieren, da die Ladezeiten unterschiedlich sind. Das folgende kleine Beispiel demonstriert den Einsatz innerhalb eines Programmes:

```
ECHO OFF
:=========================================
: File: WARTE.BAT  (C) Born G. V 1.0
: Aufruf: WARTE
: Das Programm demonstriert den Einsatz von
: WAIT.
:=========================================
```

```
: Meldung, daß Programm aktiv ist
ECHO Überprüfe den Druckerstatus an LPT1
:LOOP
LPTCHK
IF ERRORLEVEL 145 GOTO FEHLER
IF ERRORLEVEL 144 GOTO EXIT
:FEHLER   Meldung
ECHO Drucker gestört
ESC 07
WAIT 5
GOTO LOOP
:EXIT
ECHO ON
```

Listing 5.12: Anwendung von WAIT

Hier kommt übrigens bereits das kleine COM-Programm LPTCHK zum Einsatz. Es prüft, ob der Drucker gestört ist. In diesem Fall wird eine Fehlermeldung ausgegeben und fünf Sekunden bis zur nächsten Prüfung gewartet.

Die Implementierung

Die Implementierung ist recht einfach, so daß WAIT nur aus dem Hauptprogramm besteht. Wird beim Aufruf die Option */?* angegeben, dann erscheint der Bildschirm mit der Online-Hilfe. Andernfalls liest WAIT den Parameter und speichert diesen in *zeit@@*. Dann wird die Funktion DELAY aufgerufen. Nach der Wartezeit endet das Programm. Einzelheiten sind dem Listing zu entnehmen.

```
X R E F   /Z=50                          (c) Born Version 1.0
Datei : wait.bas      Datum : 06-02-1992         Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : WAIT.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 16.5.92
      '! Autor      : G. Born
      '! Progr. Spr.: PowerBASIC
      '! Betr. Sys. : DOS 2.1 - 5.0 (DR-DOS 5.0/6.0)
      '! Funktion: Das Programm wird mit der Eingabe:
      '!
      '!           WAIT <Sekunden>
      '!
      '!           aufgerufen. Das Programm wartet die im Para-
      '!           meter <Sekunden> angegebene Zeit und terminiert
      '!           anschließend. Damit läßt sich der Ablauf einer
      '!           Batchdatei für n Sekunden verzögern. (n = 1 bis
      '!           255 Sekunden.
      '*********************************************************
      '! Variable definieren
      '!*********************************************************
```

```
 1  Zeit@@ = 0
 2  kommando$ = ""

   '############################################################
   '#                      Hauptprogramm                       #
   '############################################################

 3  kommando$ = COMMAND$                  '! Parameter ?

 4  ptr% = INSTR (kommando$,"/?")         '! Option /?
 5  IF ptr% <> 0 THEN                     '! Hilfsbildschirm
 6   PRINT "W A I T                       (c) Born Version 1.0"
 7   PRINT
 8   PRINT "Aufruf: WAIT <Sekunden>"
 9   PRINT
10   PRINT "Das Programm wartet n (1 bis 255) Sekunden."
11   PRINT
12   SYSTEM
13  END IF

14  Zeit@@ = VAL(kommando$)

15  IF Zeit@@ = 0 THEN
16   PRINT "Zeitangabe ungültig"
17   SYSTEM
18  END IF

19  DELAY Zeit@@

20 END                                    '! Ende

   '***** Programm Ende ******
```

Listing 5.13: WAIT.BAS

VKEY: Abfragen des Tastaturpuffers aus Batchprogrammen

Die Module ASK und FKEY unterbrechen den Ablauf des Batchprogrammes und warten auf eine Benutzereingabe. Manchmal ist es jedoch erwünscht, daß lediglich geprüft wird, ob die Tastatur zwischenzeitlich betätigt wurde, ohne das Programm zu unterbrechen. Für diesen Zweck wurde das Programm VKEY (Verify Key) entwickelt.

Der Entwurf

Das Programm VKEY wird wie die beiden anderen Programme ASK und FKEY innerhalb einer Stapeldatei aufgerufen:

```
VKEY <Text>
```

Wird ein <*Text*> in der Kommandozeile angegeben, erscheint dieser nach dem Aufruf von VKEY auf dem Bildschirm, unabhängig von der ECHO-Einstellung (ECHO ON/ECHO OFF) der Batchdatei. Der Text darf beliebige Zeichen enthalten und ist durch ein Leerzeichen vom Programmnamen zu trennen. Dann prüft VKEY den Tastaturpuffer auf Zeichen und gibt einen Code an DOS zurück. Hierbei gilt:

Code	
0	keine Zeichen im Puffer
255	Funktionstaste gedrückt
1-254	Code der gedrückten Taste

Der betreffende Code läßt sich über die ERRORLEVEL-Funktion aus Batchdateien abfragen. Hierbei gelten die gleichen Bedingungen für die Formulierung der Abfrage wie bei dem Programm ASK.

Weiterhin soll sich das Programm mit der Option:

```
VKEY /?
```

aktivieren lassen. Dann erscheint eine Anzeige mit dem Hilfstext auf dem Bildschirm.

Die Implementierung

Das Programm entspricht vom Aufbau dem Modul ASK, so daß auf ein Hierarchiediagramm verzichtet wird. Es besteht nur aus einem Hauptmodul. Findet sich in der Kommandozeile der Parameter */?*, gibt VKEY den Hilfebildschirm aus und bricht (mit dem Fehlercode 0) ab.

Ohne Option wird der eventuell in der Kommandozeile vorhandene Text mittels der PowerBASIC-Funktion COMMAND$ ermittelt und per PRINT-Kommando ausgegeben. Liegt kein Text vor, gibt COMMAND$ einen Leerstring zurück.

Dann wird die Tastatur mit INKEY$ gelesen. Liegen keine Zeichen im Tastaturpuffer vor, endet das Programm mit dem Code 0. Bei normalen Tasteneingaben terminiert das Programm mit dem Code der Taste. Bei Extended-ASCII-Codes wird der Wert 255 zurückgegeben. Einzelheiten sind dem nachfolgenden Listing zu entnehmen.

```
X R E F   /Z=50                          (c) Born Version 1.0
Datei : vkey.bas      Datum : 06-04-1992         Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : VKEY.BAS
```

```
      '! Vers.      : 1.0
      '! Last Edit  : 26.5.92
      '! Autor      : G. Born
      '! Progr. Spr.: PowerBASIC
      '! Betr. Sys. : DOS 2.1 - 5.0 (DR-DOS 5.0/6.0)
      '! Funktion: Das Programm wird mit der Eingabe:
      '!
      '!           VKEY <Text>
      '!
      '!           aufgerufen. Es prüft ob Tasteneingaben vorliegen
      '!           und gibt den Code an DOS zurück. <Text> wird auf
      '!           dem Bildschirm ausgegeben. Dann wartet VKEY auf
die
      '!           Eingabe. Das Ergebnis läßt sich über ERRORLEVEL
      '!
      '!             IF ERRORLEVEL 3 ...
      '!
      '!           abfragen.
      '*************************************************************
      '! Variable definieren
    1 zchn$ = ""
    2 ptr% = 0
    3 kommando$ = ""

      '#############################################################
      '#                      Hauptprogramm                        #
      '#############################################################

    4 kommando$ = COMMAND$                  '! Parameter ?

    5 ptr% = INSTR (kommando$,"/?")         '! Option /?
    6 IF ptr% <> 0 THEN                     '! Hilfsbildschirm
    7   PRINT "V K E Y                        (c) Born Version 1.0"
    8   PRINT
    9   PRINT "Aufruf: VKEY <Text>"
   10   PRINT
   11   PRINT "Das Programm prüft, ob eine Taste betätigt wurde
und"
   12   PRINT "gibt den Code an DOS zurück:"
   13   PRINT
   14   PRINT "ERRORLEVEL =    0 keine Taste"
   15   PRINT "              255 Funktionstaste"
   16   PRINT "            1-254 Tastencode"
   17   PRINT
   18   PRINT "Das Feld <Text> ist optional und erlaubt die
Ausgabe"
   19   PRINT "einer Benutzermeldung beim Aufruf."
   20   PRINT
   21   SYSTEM
   22 END IF

   23 PRINT kommando$;" ";                  '! Text ausgeben
   24 zchn$ = INKEY$                        '! Tastatur abfragen
   25 IF (LEN(zchn$) = 0) THEN              '! Puffer Leer
   26  END (0)                              '! 0 zurückgeben
```

```
27 ELSEIF LEN(zchn$) = 1 THEN
28  END (ASC(zchn$))                    '! Tastencode
29 ELSE
30  END (255)                           '! Funktionstaste
31 END IF

32 END
   '***** Programm Ende ******
```

Listing 5.14: VKEY.BAS

6 Strandgut

Im Rahmen meiner bisherigen Tätigkeit haben sich eine Reihe von Programmen und Modulen angesammelt, die für Entwickler und Anwender von Interesse sein können. Ich habe deshalb eine Auswahl diese Module in diesem Kapitel unter dem Motto »Strandgut« zusammengefasst. Es handelt sich um eine lose Sammlung an Tools, die sich nicht in die vorhergehenden Kapitel einordnen lassen.

Ein Menüsystem für PowerBASIC

Alle in den vorhergehenden Kapiteln gezeigten Programme sind mit einer recht einfachen Benutzersteuerung ausgestattet. Diese reicht für die benötigte Funktionalität aus und lehnt sich zudem an das DOS-5.0-Bedienkonzept an. Bei umfangreicheren Programmfunktionen ist aber in der Regel eine Menüführung erforderlich. Von Standardprogrammen sind und Pop-up-Menüs bekannt. PowerBASIC bietet hier aber keine Unterstützung. Ansätze im Shareware-Bereich existieren zwar, diese dürfen aber nicht frei weiterverwendet werden. Weiterhin fehlt meist der Quellcode, welcher für Erweiterungen erforderlich ist und die gebotene Funktionalität ist nicht immer überzeugend. Deshalb entstand bei mir die Idee, eine Menüsteuerung in PowerBASIC zu implementieren und die Bibliothek nachfolgend zu veröffentlichen. Als Ausgangspunkt fand sich in der Zeitschrift Toolbox (DMV-Verlag) ein kleines Programm (88 Zeilen) in TurboBASIC für Pop-up-Menüs. Das Listing war schnell abgetippt und lauffähig. Allerdings fehlten viele Funktionen, so daß ich mich entschloß, das Paket grundlegend zu überarbeiten. Nach zwei Tagen stand eine komplette Bibliothek zur Menüsteuerung in PowerBASIC, die mit dem Urprogramm kaum noch etwas zu tun hat. Diese Bibliothek sowie zwei einfache Demonstrationsprogramme für Pop-up- und Pull-down-Menüs möchte ich nachfolgend vorstellen.

Der Entwurf der Bibliothek

Die Bibliothek soll alle Funktionen bereitstellen, mit denen sich eine einfache Menüsteuerung in PowerBASIC realisieren läßt. Die nachfolgenden Vorüberlegungen definieren deshalb die Anforderungen:

Um ein Pop-up-Menü zu generieren, muß der Programmierer die gewünschten Menütexte zusammenstellen und die Position der Pop-up-Box definieren. Dann muß die Bibliothek das betreffende Menü selbständig auf dem Bildschirm darstellen. Bild 6.1 zeigt ein Beispiel für solches Menü:

Bild 6.1: Beispiel eines Pop-up-Menü

Die vier Menüpunkte werden mit einem Rahmen versehen und auf dem Bildschirm an der definierten Position ausgegeben. Die markierte Zeile symbolisiert den Auswahlcursor. Dieser Cursor läßt sich mittels der Tasten Pfeil oben und Pfeil unten zwischen den einzelnen Menüpunkten verschieben. Sobald die Tasten Esc, Pfeil links, Pfeil rechts oder Eingabe betätigt werden, wird das Menü beendet und das Anwenderprogramm erhält die Kontrolle zurück. Der Abbruch über die Cursortasten sollte sich aber bei Bedarf sperren lassen. Rückgabeparameter geben dann an, welcher Menüpunkt zuletzt per Cursor selektiert war. Weiterhin ist von Interesse, ob das Menü per Esc-Taste abgebrochen oder ein Menüpunkt per Eingabe quittiert wurde. Dann kann das Anwenderprogramm auf die Benutzereingaben reagieren.

Neben diesen einfachen Grundanforderungen lassen sich aber noch einige Zusatzwünsche formulieren. Die Menüsteuerung übernimmt zwar die Ausgabe des Menüs an der vorgegebenen Position und zeichnet auch einen Rahmen um den Text. Die Abmessungen des Rahmen werden dabei automatisch aus der Länge der Menütexte berechnet. Als Option ist nun die Auswahl des Rahmentyps denkbar (einfacher Rahmen, Doppelrahmen, kein Rahmen). Weiterhin ist es teilweise erwünscht, in der Kopf- und Fußzeile des Rahmens Texte einzublenden (Bild 6.2).

Bild 6.2: Pop-up-Menü mit Kopf- und Fußtexten

Der Kopftext kann einen Hinweis auf die Funktion des Menüs geben, während im Fußtext Fehlermeldungen oder Bedienhinweise erscheinen können. Der nächste Punkt betrifft die Selektion der Farben für Text und Hintergrund. Beide sollten vor Aufruf des Menüs definierbar sein. Hilfreich ist weiterhin die Möglichkeit, die Position des Cursors im Menüsystem vor dem Aufruf zu definieren.

Mit diesen Optionen sind die Wünsche hinsichtlich der Programmschnittstelle für Pop-up-Menüs abgedeckt. Vor der

Implementierung sollten jedoch noch einige Gedanken bezüglich der Funktionalität der Software diskutiert werden.

Beim Aufbau eines Pop-up-Menüs wird zum Beispiel der betreffende Bildschirmbereich durch das Menü überschrieben. Wird dann das Menü wieder geschlossen, sind die vorher vorhandenen Daten verloren. Daher muß der betreffende Bildschirmbereich vor Ausgabe des Menüs gesichert werden.

Nachdem ein Menü vom Benutzer mittels Esc oder Eingabe (oder Cursortasten) beendet wird, geht die Kontrolle wieder an das Anwenderprogramm zurück. Denkbar ist es nun, gleichzeitig das Menü zu löschen und den alten Bildschirminhalt zu restaurieren. Dies ist bei einstufigen Menüs (Bild 6.1) tolerierbar. Was ist aber, wenn mehrere Menüs gleichzeitig (Bild 6.3) sichtbar sein sollen? Hier muß die Kontrolle, wann ein Menü geschlossen und der Bildschirm restauriert wird, beim Anwenderprogramm liegen.

Bild 6.3: Mehrstufiges Pop-up-Menü

Bei der Ausgabe eines Pop-up-Menüs sind deshalb folgende Schritte erforderlich:

- Sicherung des Bildschirmbereiches der Menübox.
- Ausgabe des Pop-up-Menüs und Benutzersteuerung.
- Schließen der Menübox und Restaurieren des Bildschirmbereiches.

Naheliegend wäre es, alle diese Funktionen automatisch mit einem Funktionsaufruf abzuwickeln. Wegen der oben beschriebenen Problematik bei mehrstufigen Menüs geht dies aber nicht. Weiterhin treten bei der Implementierung verschiedene Probleme auf; so muß die Größe der Puffer für die zu sichernden Bildschirmausschnitte innerhalb der Menübibliothek bekannt sein. Diese ist aber abhängig von der Anzahl und den Abmessungen der Menüboxen. Puffer innerhalb der Bibliothek müssen aber vorab definiert werden, was eine Limitierung der Menüsteuerung zur Folge hat. Daher soll die Implementierung der Menübibliothek so erfolgen, daß das Anwendungsprogramm die komplette Kontrolle über die Menüsteuerung behält. Damit ist das Programm auch für die Sicherung und Restaurierung der Bildschirmbereiche verantwortlich. Als positiver Vorteil ist zu sehen, daß damit die Zahl der gleichzeitig offenen Menüs nur noch durch das Anwenderprogramm bestimmt wird.

Erweiterung um Pull-down-Funktionen

Alle bisherigen Diskussionspunkte beziehen sich nur auf Pop-up-Menüs. Bei Pull-down-Menüs besteht die einzige wirkliche Erweiterung in der Menüleiste, die am oberen Bildschirmrand ausgegeben wird. Ein geöffnetes Pull-down-Menü läßt sich dann als Pop-up-Menü mit einer festen Position interpretieren (Bild 6.4).

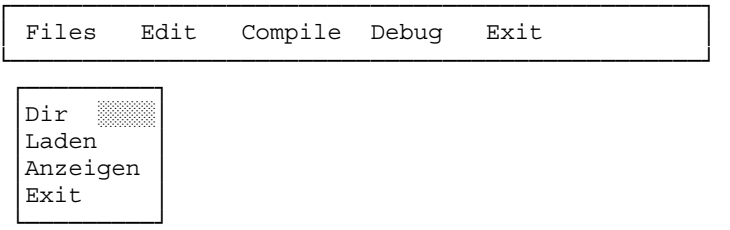


Bild 6.4: Pull-down-Menü

Damit muß die Bibliothek lediglich um eine Funktion zur Ausgabe der Menüleiste erweitert werden. Die Cursorsteuerung kann dann vom Anwenderprogramm übernommen werden. Damit läßt sich auch die Funktionalität des Pull-down-Menüs bestimmen (siehe Anwendungsbeispiel).

Die Implementierung

Die Implementierung der Bibliothek erfolgt in PowerBASIC. Die einzelnen Module werden in einer eigenen Bibliotheksdatei namens MENU.INC abgelegt:

Auf die Verwendung einer Unit wurde hier aus Aufwandsgründen verzichtet.

Alle von der Bibliothek benutzten globalen Variablen sind entweder intern als SHARED definiert oder als Übergabeparameter zu vereinbaren. Nachfolgend werden erst die für das Anwenderprogramm relevanten Prozeduren beschrieben. Der Abschnitt läßt sich daher auch als Handbuch für die Bibliothek interpretieren. Anschließend folgen die intern benutzten Hilfsprozeduren.

MenuInit (screenseg%)

Diese Prozedur muß als erstes vom Anwendungsprogramm aufgerufen werden, um einige zentrale (shared) Variable zu initialisieren. Der Übergabeparameter *screenseg%* definiert die Segmentadresse des Bildschirmadapters. Diese wird zur Sicherung des Bildschirminhaltes benötigt. Da die Menüs nur im Textmodus funktionieren, beschränken sich die Adressen auf:

```
&HB000      für Monochromkarten (HGC)
&HB800      für Colorkarten (CGA, EGA, VGA)
```

Der Parameter wird in der globalen Variable *scrseg%* gespeichert. Die Variablen *xmax%* und *ymax%* definieren die Bildschirmabmessungen (z.B. 80x25). Hier können bei Bedarf auch andere Abmessungen (z.B. 80x43) eingestellt werden. Der Versuch, ein Menü außerhalb dieser Grenzen auszugeben, wird von der Menübibliothek abgefangen.

Die globale Variable *initflg%* signalisiert durch den Wert 1, daß die Initialisierung erfolgreich durchgeführt wurde. Falls vom Anwendungsprogramm das Modul *MenuInit* nicht aufgerufen wurde, läßt sich kein Pop-up- oder Pull-down-Menü über *PopMenu*, *PullMenu* ausgeben.

PopMenu (x%, y%, text$(1), items%, fcol%, bcol%, title$, foot$, style%, status%, nr%)

Dies ist die eigentliche Prozedur zur Erzeugung eines Pop-up-Menüs. Die Prozedur ist vom Anwenderprogramm aufzurufen und benötigt folgende Parameter:

```
x%|X-Position linke obere Ecke
y%|Y-Position linke obere Ecke
text$()|Feld mit den Menütexten
items%|Zahl der Menüzeilen in text$()
fcol%|Vordergrundfarbe (0..31)
bcol%|Hintergrundfarbe (0..7)
title$|Titeltext
foot$|Fußtext
style%|Rahmentyp (0,1,2)
status%|Status des Aufrufs und Steuerparameter
nr%||selektierter Menüpunkt
```

Das Modul öffnet eine Box an der mit *x%,y%* angegebenen Stelle. Breite und Höhe der Box werden automatisch aus den Längen der Menütexte und dem Titel- und Fußtext bestimmt. Diese Texte sind in den Parametern *text$(), title$, foot$* zu übergeben. Die Anzahl der Einträge in *text$()* (Menüzeilen) muß in *items%* übergeben werden. Falls kein Kopf- oder Fußtext erscheinen soll, sind Leerstrings als Parameter in *title$* und *foot$* zu übergeben.

Die Farbauswahl für die Menübox erfolgt durch die Parameter *fcol%* und *bcol%*. Die Codierung der Werte ist im PowerBASIC-Handbuch beim COLOR-Befehl beschrieben:

0 schwarz••4 rot
1 blau••5 magenta
2 grün••6 braun
3 zyan••7 weiß

Weitere Codes für die Vordergrundfarben sind dem Basic-Handbuch zu entnehmen.

Der Parameter *style%* definiert den Rahmentyp (Tabelle 6.2). In *status%* findet sich nach der Rückkehr die Information über den Ablauf der Operation. Hierbei gelten die Werte aus Tabelle 6.1:

status%	Bedeutung
	beim Aufruf
0	kein Exit bei Cursor links/rechts
1	Exit bei Cursor links/rechts
	bei der Rückkehr
-2	Initialisierung fehlt
-1	Menü paßt nicht auf Bildschirm
0	Exit mit RETURN-Taste
1	Exit mit ESC-Taste
2	Exit mit Cursor rechts
3	Exit mit Cursor links

Tabelle 6.1: Statuscodes von PopMenu

Wird vor dem Aufruf der Wert von *status% = 1* gesetzt, gibt das Modul die Kontrolle an die Anwendung zurück, wenn die Tasten Pfeil links oder Pfeil rechts erkannt werden (kann bei Pull-down-Menüs relevant sein). Bei *status% = 0* wird dies blockiert.

Die Rückgabewerte in *status%* (0 bis 3) erlauben dem Anwenderprogramm die Unterscheidung, ob das Menü mit Esc, den *Cursortasten* oder durch Selektion eines Menüpunktes über die Eingabe-Taste verlassen wurde. Über Esc kann ein Benutzer ein versehentlich geöffnetes Menü schließen. Der zuletzt durch den Cursor selektierte Menüpunkt findet sich in allen Fällen im Parameter *nr%*. Wird beim Aufruf von *PopMenu* in *nr%* ein Wert zwischen *1* und *items%* übergeben, positioniert das Modul den Cursor auf den zugehörigen Menüpunkt. Andernfalls findet sich der Cursor auf dem 1. Menüpunkt. Dies ist bei mehrstufigen Menüsystemen wichtig, da dann das Anwenderprogramm für die Cursorpositionierung verantwortlich ist (siehe auch Beispielprogramm).

PopMenu kümmert sich nicht um die Sicherung und Restaurierung des Bildschirmes. Dies muß durch die Module *OpenBox* und *CloseBox* im Anwenderprogramm erfolgen. Das Menü bleibt auch nach Verlassen der Prozedur *PopMenu* sichtbar bis es mit *CloseBox* geschlossen wird. Damit kann ein Menü jederzeit reaktiviert werden. Der Aufbau der Prozedur *PopMenu* ist dem folgenden Listing zu entnehmen. Die Anweisungen sind ausgiebig kommentiert, so daß auf eine Diskussion an dieser Stelle verzichtet wird.

PullMenu (text$(1), items%, fcol%, bcol%, xpos%(1), status%, nr%)

Dies ist die Prozedur zur Erzeugung einer Pull-down-Menüleiste. Die Prozedur ist vom Anwenderprogramm aufzurufen und benötigt folgende Parameter:

`text$()`|`Feld mit den Menütexten`

```
items%│Zahl der Menüzeilen in text$()
fcol%│Vordergrundfarbe (0..31)
bcol%│Hintergrundfarbe (0..7)
xpos%()│Rückgabeparameter x-Pos. der Menüeinträge
status%│Status nach dem Aufruf, Steuereingang
nr%│selektierter Menüpunkt
```

Das Modul gibt eine Menüzeile in der obersten Bildschirmzeile aus. Der Bildbereich kann vorher mit *SaveArea* im Anwenderprogramm gesichert werden. Die Anzahl der Einträge in *text$()* (Menüpunkte) muß in *items%* übergeben werden.

Die Farbauswahl für die Menüleiste erfolgt durch die Parameter *fcol%* und *bcol%*. Die Kodierung der Werte entspricht der Prozedur *PullMenu* und ist im PowerBASIC-Handbuch beim COLOR-Befehl beschrieben.

Der Parameter *status%* steuert beim Aufruf die Option zur Cursorsteuerung. Wird der Wert *status% = 1* vor dem Aufruf gesetzt, gibt das Modul die Kontrolle an die Anwendung zurück, sobald die Tasten Pfeil links oder Pfeil rechts erkannt werden. Bei *status% = 0* wird dies blockiert. Die Rückgabewerte in *status%* (0 bis 3) erlauben dem Anwenderprogramm die Unterscheidung, ob das Menü mit Esc, den *Cursortasten* oder durch Selektion eines Menüpunktes über die Eingabe-Taste verlassen wurde. Die Codierung entspricht den Vorgaben in Tabelle 6.1. Über Esc kann ein Benutzer ein versehentlich geöffnetes Menü schließen.

Der zuletzt durch den Cursor selektierte Menüpunkt findet sich in allen Fällen im Parameter *nr%*. Wird beim Aufruf von *PopMenu* in *nr%* ein Wert zwischen *1* und *items%* übergeben, positioniert das Modul den Cursor auf den zugehörigen Menüpunkt. Andernfalls findet sich der Cursor auf dem 1. Menüpunkt. Dies ist bei geöffneten Pull-down-Menüs wichtig, da dann das Anwenderprogramm für die Cursorpositionierung verantwortlich ist.

PullMenu kümmert sich nicht um die Sicherung und Restaurierung des Bildschirmes. Dies muß durch die Module *SaveArea* und *CloseBox* im Anwenderprogramm erfolgen. Die Menüzeile bleibt auch nach Verlassen der Prozedur *PullMenu* sichtbar, bis sie mit *CloseBox* geschlossen wird. Damit kann sie jederzeit reaktiviert werden. Der Aufbau der Prozedur *PullMenu* ist dem folgenden Listing zu entnehmen.

OpenBox (x%, y%, text$(1), items%, title$, foot$, buff%(1))

Dies ist eine der wichtigsten Routinen für das Anwenderprogramm. Vor dem Aufbau eines Pop-up-Menüs muß die Prozedur aufgerufen werden um den Bildschirmbereich unterhalb der Menübox im Parameter *buff%()* zu sichern. Da die Puffer im Anwenderprogramm definiert werden, läßt sich deren Größe und Anzahl in Abhängigkeit von der Anwendung festlegen. Zur Sicherung eines kompletten Bildschirms werden 2004 Elemente (4008 Byte) benötigt. Jedes Zeichen des Bildschirms besteht dabei aus zwei Byte (Zeichen+Attribut). Die ersten vier Elemente dienen zur Aufnahme der Parameter für Lage und Größe des Bildbereiches. In der Regel kann der Puffer jedoch kleiner als 2004 sein, da die Menübox keinen

kompletten Bildschirm umfaßt. Um Lage und Größe des zu sichernden Bereiches in *OpenBox* zu berechnen, müssen deshalb die Parameter *x%, y%, text$(1), items%, title$* und *foot$* beim Aufruf übergeben werden.

OpenBox darf nur einmal vor dem Öffnen eines Pop-up-Menüs aufgerufen werden (siehe auch Beispielprogramm). Das Modul aktiviert dann die Prozedur *SaveArea*.

SaveArea (x%, y%, breite%, hohe%, buff%())

Die Routine erlaubt die Sicherung eines Bildschirmbereiches an der Stelle *x%,y%* mit den Abmessungen *breite% x hohe%* in *buff%()*. Der Puffer muß im Anwenderprogramm definiert werden. Damit läßt sich dessen Größe in Abhängigkeit von der Anwendung festlegen. Zur Sicherung eines kompletten Bildschirms werden 2004 Elemente (4008 Byte) benötigt. Vier Elemente dienen zur Sicherung der Parameter *x%, y%, breite%* und *hohe%*. *SaveBox* wird von *OpenBox* benutzt und darf auch von Anwendungsprogrammen aufgerufen werden. Das Modul greift direkt auf den Bildschirmspeicher zurück und benötigt deshalb die Segmentadresse aus der globalen Variablen *scrseg%*. Diese muß vorher in *MenuInit* definiert werden.

CloseBox (buff%(1))

Durch Aufruf dieser Prozedur wird eine geöffnete Menübox wieder geschlossen und der Hintergrund des Bildschirms restauriert. Wichtig ist die Übergabe des korrekten Puffers. Andernfalls wird der Bildschirm nicht korrekt restauriert.

Die Hilfsroutinen

Der nachfolgende Abschnitt beschreibt einige Hilfsroutinen, die nur intern durch die Bibliothek benutzt werden können.

MenuLine (lo$, lu$, ro$, ru$, li$, lup$, style%)

Diese Hilfsprozedur wird nur intern benutzt und initialisiert den Linientyp für den Rahmen. Der Parameter *style%* definiert dabei die Linienart (Tabelle 6.2).

style	Rahmentyp
1	einfacher Rahmen
2	Doppelrahmen
sonst	kein Rahmen (Blanks)

Tabelle 6.2: Rahmenart

Die Prozedur legt dann die benötigten Zeichen für den Rahmen in den folgenden Variablen ab:

```
lo$  linke obere Ecke
lu$  linke untere Ecke
```

```
ro$||rechte obere Ecke
ru$||rechte untere Ecke
li$||horizontale Linie
lup$|vertikale Linie
```

Die Zeichen werden in *PopMenu* zur Konstruktion des Rahmens verwendet. Daher wird *MenuLine* vor jeder Ausgabe einer Menübox von *PopMenu* aufgerufen.

GetMaxLen (text$(1), items%, title$, foot$, maxlen%)

Dies ist eine interne Hilfsroutine, die die Breite der Menübox für Pop-up-Menüs ermittelt. Diese berechnet sich aus dem längsten String, der in *text%()*, *title$* oder *foot$* vorkommt. Das Ergebnis wird in *maxlen%* zurückgegeben. Die Textbox selbst ist dann zwei Zeichen breiter als *maxlen%*.

PutLine (text$, xlen%)

Dies ist ebenfalls eine nur intern benutzte Routine, die durch *PopMenu* aktiviert wird. Aufgabe ist es, jeweils den Text einer Menüzeile auszugeben und den Bereich bis zum rechten Rahmen der Box mit Leerzeichen aufzufüllen. Der Parameter *text$* enthält den Text der Menüzeile, während *xlen%* die Breite des Textbereiches angibt.

Damit möchte ich die Beschreibung der Bibliotheksroutinen beenden. Sicherlich sind noch Verbesserungen denkbar. So kann die Auswahl eines Menüpunktes durch Eingabe des ersten Buchstabens erfolgen. Dies erfordert nur wenige zusätzliche Zeilen in *PopMenu*. Nachfolgend findet sich das Listing mit den einzelnen Bibliotheksroutinen.

```
X R E F   /Z=55                          (c) Born Version 1.0
Datei : menu.inc      Datum : 07-19-1992         Seite :  1

 Zeile     Anweisung

      '!#####################################################
      '! File:    MENU.INC
      '! Version: 1.0 v. 10.7.92 (c) G. Born
      '!          Subroutinen zur Menüsteuerung
      '!          Pull Down und Pop Up
      '!#####################################################

    1 SUB MenuInit (screenseg%)
      '!-----------------------------------------------------
      '!  Subroutine zur Initialisierung der Variablen
      '!-----------------------------------------------------

    2 SHARED xmax%, ymax%, initflg%, scrseg%

    3  scrseg% = screenseg%                '! Seg. Adr. Adapter
    4 ||||     '! B800H oder B000H
    5  xmax% = 80                          '! rechter Bildrand
    6  ymax% = 25                          '! unterer Bildrand
```

```
    7  initflg% = 1                          '! Init ok

    8 END SUB

    9 SUB MenuLine (lo$,lu$,ro$,ru$,li$,lup$,style%)
      '!-----------------------------------------------------
      '!  Subroutine zur Einstellung des Rahmentyps
      '!-----------------------------------------------------

   10  SELECT CASE style%
   11   CASE = 1                              '! Linientyp 1
   12    lo$ = "Ú"                            '! Ecke links oben
   13    lu$ = "À"                            '! Ecke links unten
   14    ro$ = "¿"                            '! Ecke rechts oben
   15    ru$ = "Ù"                            '! Ecke rechts unten
   16    li$ = "-"                            '! Linie waagerecht
   17    lup$ = "│"                           '! Linie senkrecht
   18   CASE = 2
   19    lo$ = "É"                          '! Linientyp 2
   20    lu$ = "È"
   21    ro$ = "»"
   22    ru$ = "¼"
   23    li$ = "Í"
   24    lup$ = "°"
   25   CASE ELSE
   26    lo$ = " "                          '! Linientyp 3
   27    lu$ = " "
   28    ro$ = " "
   29    ru$ = " "
   30    li$ = " "
   31    lup$ = " "
   32   END SELECT

   33 END SUB

   34 SUB PopMenu (x%, y%, text$(1), items%, fcol%, bcol%, title$,
foot$,
       style%, status%, nr%)
      '!-----------------------------------------------------
      '!  Subroutine für PopUp-Menü
      '!
      '!  Die Routine gibt die Menübox mit dem Text aus.
      '!
      '! x%, y%   Anfangskoordinaten linke obere Ecke
      '! text$()  Texte mit Menüpunkten
      '! items%   Zahl der Menüpunkte
      '! title$   Text Kopfzeile
      '! foot$    Text Fußzeile
      '! style%   Rahmentyp (1 = einfach, 2 = doppelt, sonst blank
      '! fcol%    Vordergrundfarbe
      '! bcol%    Hintergrundfarbe
      '! status%  vor Aufruf:
      '!           0 kein Exit bei Cursor rechts/links
      '!           1 Exit bei Cursor rechts/links
      '!          Ergebnis des Aufrufes:
```

```
   '!          -2 Fehler: Initialisierung fehlt
   '!          -1 Fehler: Box paßt nicht auf Bildschirm
   '!           0 ok, Menü mit RETURN beendet
   '!           1 ok, Menü mit ESC beendet
   '!           2 ok, Menü mit Cursor Rechts beendet
   '!           3 ok, Menü mit Cursor Links beendet
   '! nr%      Aufruf: vorselektierter Menüpunkt
   '!          Return: Nummer des selektierten Menüpunktes
   '!-------------------------------------------------

35 LOCAL maxlen%, i%, zchn$, zeile%, old%, flag%
36 LOCAL lo$, lu$, ro$, ru$, li$, lup$
37 SHARED xmax%, ymax%, initflg%

38 flag% = status%                              '! merke status

   '! prüfe ob INIT durchgeführt

39  IF initflg% <> 1 THEN
40   status% = -2
41   EXIT SUB
42  END IF

   '! Emittle Länge des Menüpunktes

43  CALL GetMaxLen (text$(),items%, title$, foot$, maxlen%)

   '! Paßt das Menü auf den Bildschirm ?

44  IF (x% + maxlen% + 2) > xmax% THEN
45   status% = -1
46   EXIT SUB
47  END IF

48  IF (y% + items% + 2) > ymax% THEN
49   status% = -1
50   EXIT SUB
51  END IF

   '! Rahmentyp setzen

52  CALL MenuLine(lo$,lu$,ro$,ru$,li$,lup$,style%)

   '! Rahmen zeichnen

53  COLOR fcol%,bcol%
54  LOCATE y%, x%                    '! linke obere Ecke
55  PRINT lo$;
56  IF (LEN(title$) > 0) THEN
57   PRINT title$;                   '! Titel Textbox
58  END IF
59  IF LEN(title$) < maxlen% THEN
60   FOR i% = LEN(title$) TO maxlen%-1 : PRINT li$; : NEXT i%
61  END IF
62  PRINT ro$
```

```
   63  FOR i% = 1 TO items%
   64   LOCATE (y%+i%), x%
   65   PRINT lup$;
   66   CALL PutLine (text$(i%),maxlen%)
   67   PRINT lup$
   68  NEXT i%

   69  LOCATE (y%+items%+1), x%
   70  PRINT lu$;
   71  IF (LEN(foot$) > 0) THEN
   72   PRINT foot$;                        '! Fußtext
   73  END IF
   74  IF LEN(foot$) < maxlen% THEN
   75   FOR i% = LEN(foot$) TO maxlen%-1 : PRINT li$; : NEXT i%
   76  END IF
   77  PRINT ru$

      '! Init Variable für Selection
      '! ermittle die Position der invertierten Zeile
      '! diese kann in nr% vorgegeben werden (1 - items%)

   78  old% = items%                       '! letzte Zeile

   79  IF (nr% < 1) OR (nr% > items%) THEN '!
   80   zeile% = 1                         '! 1. Menüpunkt
   81  ELSE
   82   zeile% = nr%                       '! vorgegebener Punkt
   83   IF nr% = items% THEN old% = 1
   84  END IF

   85  zchn$ = CHR$(0)                     '! Init Puffer

      '! decodiere Benutzerauswahl
      '! Warte bis CR oder ESC betätigt wurde
   86  loopx% = 0
   87  WHILE (loopx% = 0)

      '! prüfe ob sich die Cursorauswahl gegenüber dem letzten
Durchlauf
      '! verändert hat. Markiere betreffende Auswahlzeile des Menüs
durch
      '! inverse Textdarstellung

   88   IF zeile% <> old% THEN             '! Änderung
   89    LOCATE (y%+old%), (x%+1)          '! Cursor auf alte Zeile
   90    COLOR fcol%,bcol%                 '! normal darstellen
   91    CALL PutLine(text$(old%),maxlen%) '! schreiben
   92    LOCATE (y%+zeile%), (x%+1)        '! Cursor auf neue Zeile
   93    COLOR bcol%,fcol%                 '! invers darstellen
   94    CALL PutLine(text$(zeile%),maxlen%) '! schreiben
   95   END IF

      '! lese Tastatur aus und decodiere ggf. die Tastencodes
```

```
   zchn$ = INKEY$                    '! lese Tastencodes
   IF LEN(zchn$) = 2 THEN            '! Extended ASCII-Code
    zchn$ = RIGHT$(zchn$,1)          '! entferne  1. Zeichen
    SELECT CASE zchn$
     CASE = CHR$(72)                 '! Cursor UP
      old% = zeile%                  '! merke Zeile
      zeile% = zeile% - 1            '! neue Position
      IF zeile% = 0 THEN zeile% = items% '! wrap
     CASE = CHR$(80)                 '! Cursor DOWN
      old% = zeile%                  '! merke Zeile
      zeile% = zeile% + 1            '! neue Position
      IF zeile% > items% THEN zeile% = 1 '! wrap
     CASE = CHR$(75)                 '! Cursor RIGHT
      status% = 2                    '! Exitcode
      IF flag% = 1 THEN loopx% = 1   '! Exit
     CASE = CHR$(77)                 '! Cursor LEFT
      status% = 3
      IF flag% = 1 THEN loopx% = 1   '! Exit
     END SELECT
   ELSE
    SELECT CASE zchn$
     CASE = CHR$(13)                 '! Return
      status%=0                      '! Exitcode CR
      loopx% = 1
     CASE = CHR$(27)
      status%=1                      '! Exitcode ESC
      loopx% = 1
     END SELECT
   END IF
 WEND

 nr% = zeile%                         '! gebe Auswahlcode
zurück

END SUB

SUB GetMaxLen (text$(1),items%,title$,foot$,maxlen%)
    '!-----------------------------------------------------
    '!  Subroutine zur Ermittlung der Breite der Box
    '!
    '!-----------------------------------------------------

 LOCAL i%

    '! Ermittle die Länge des größten Menüpunktes

 maxlen% = 0
 FOR i% = 1 TO items%
  IF (LEN(text$(i%)) > maxlen%) THEN
   maxlen% = LEN(text$(i%))
  END IF
 NEXT i%

    '! Ist die Titellänge > als maxlen?
 IF (LEN(title$) > maxlen%) THEN
```

```
  137   maxlen% = LEN(title$)
  138  END IF

      '! Ist die Fußlänge > als maxlen?
  139  IF (LEN(foot$) > maxlen%) THEN
  140   maxlen% = LEN(foot$)
  141  END IF

  142 END SUB

  143 SUB PutLine (text$,xlen%)
      '!-------------------------------------------------
      '!  Subroutine zur Ausgabe einer Menüzeile
      '!
      '!  Die Routine gibt den text$ der Länge xlen% in der
      '!  gesetzten Farbe in der Menübox aus.
      '!-------------------------------------------------

  144  LOCAL i%

  145  PRINT text$;                        '! Original Text ausgeben
  146  FOR i% = 1 TO xlen%-LEN(text$)      '! mit Blanks auffüllen
  147   PRINT " ";
  148  NEXT i%

  149 END SUB

  150 SUB OpenBox (x%, y%, text$(1), items%, title$, foot$,
buff%(1))
      '!-------------------------------------------------
      '!  Subroutine zur Sicherung des Bildschirmausschnittes
      '!  unter der Menübox. (Variable siehe PopMenu)
      '!-------------------------------------------------

  151  SHARED scrseg%                     '! Seg. Adr. Screen
  152  LOCAL maxlen%

      '! ermittle Länge des Menüpunktes

  153  CALL GetMaxLen (text$(),items%, title$, foot$, maxlen%)
  154  maxlen% = maxlen% + 2              '! wegen Rahmen

      '! sichere Bildschirmausschnitt

  155  CALL SaveArea (x%, y%, maxlen%,items%+2, buff%())

  156 END SUB

  157 SUB SaveArea (x%, y%, breite%, hohe%, buff%(1))
      '!-------------------------------------------------
      '!  Subroutine zur Sicherung eines Bildschirmausschnittes
      '!  ab Punkt x,y mit den Abmessungen breite x hohe in
      '!  buff().
      '!-------------------------------------------------
```

```
  158  SHARED scrseg%                    '! Seg. Adr. Screen
  159  LOCAL i%, j%, ptr%, ofs%

      '! sichere die Parameter des Bildausschnitts im Buffer

  160  buff%(1) = x%
  161  buff%(2) = y%

  162  buff%(3) = breite%
  163  buff%(4) = hohe%

      '! sichere den Bildschirmbereich

  164  DEF SEG = scrseg%                  '! Screen Segment
  165  ptr% = 5                           '! Beginn Screen-Puffer
  166  FOR i% = 1 TO hohe%                '! alle Zeilen
  167   ofs% = ((y%+i%-2)*80 + (x%-1))*2  '! Anfangsadresse
  168   FOR j% = 0 TO breite%             '! alle Spalten
  169    buff%(ptr%) = PEEKI(ofs%+j%*2)
  170    ptr% = ptr% + 1
  171   NEXT j%
  172  NEXT i%
  173  DEF SEG
  174 END SUB

  175 SUB CloseBox (buff%(1))
      '!-----------------------------------------------------
      '!  Subroutine zur Restaurierung des Bildschirmausschnittes
      '!  unter der Menübox.
      '!-----------------------------------------------------

  176  SHARED scrseg%                    '! Seg. Adr. Screen
  177  LOCAL i%, j%, ptr%, ofs%, x%, y%, hohe%, breite%

      '! lese Parameter des Ausschnittes

  178  x% = buff%(1)
  179  y% = buff%(2)
  180  breite% = buff%(3)
  181  hohe%   = buff%(4)

  182  DEF SEG = scrseg%                 '! Screen Segment
  183  ptr% = 5
  184  FOR i% = 1 TO hohe%
  185   ofs% = ((y%+i%-2)*80 + (x%-1))*2
  186   FOR j% = 0 TO breite%
  187    POKEI (ofs%+j%*2),buff%(ptr%)
  188    ptr% = ptr% + 1
  189   NEXT j%
  190  NEXT i%
  191  DEF SEG
  192 END SUB

  193 SUB PullMenu (text$(1), items%, fcol%, bcol%, xpos%(1),
status%,
```

```
    nr%)
    '!-----------------------------------------------------
    '!  Subroutine für PullDown-Menü
    '!
    '!  Die Routine gibt die Menüzeile mit dem Text aus.
    '!
    '! text$()  Texte mit Menüpunkten
    '! items%() Zahl der Punkt in der Menüzeile
    '! fcol%    Vordergrundfarbe
    '! bcol%    Hintergrundfarbe
    '! xpos%()  x-Koordinate Menüpunkt
    '! status%  vor Aufruf
    '!           1 Exit bei Cursor rechts/links
    '!           0 kein Exit bei Cursor rechts links
    '!          Ergebnis des Aufrufes
    '!          -2 Fehler: Initialisierung fehlt
    '!          -1 Fehler: Zeile paßt nicht auf Bildschirm
    '!           0 ok, Menü mit RETURN beendet
    '!           1 ok, Menü mit ESC beendet
    '!           2 ok, Menü mit Cursor Rechts beendet
    '!           3 ok, Menü mit Cursor Links beendet
    '! nr%      vor Aufruf Position invert. Menüpunkt
    '!          nach Aufruf:
    '!          Nummer des selektierten Menüpunktes
    '!-----------------------------------------------------

194 LOCAL maxlen%, i%, zchn$, spalte%, old%, tmp%
195 LOCAL loopx%, flag%, leer$
196 SHARED xmax%, ymax%, initflg%

197  flag% = status%                    '! merke Flag

    '! prüfe ob INIT durchgeführt

198  IF initflg% <> 1 THEN
199   status% = -2
200   EXIT SUB
201  END IF

    '! Paßt die Menüzeile auf den Bildschirm ?

202  tmp% = 1                             '! 1. Leerzeichen
203  FOR i% = 1 TO items%
204   tmp% = tmp% + LEN(text$(i%)) + 1
205  NEXT i%

206  IF ptr% > xmax% THEN
207   status% = -1
208   EXIT SUB
209  END IF

    '! ermittle ob Zwischenraum vergrößert werden kann
210  leer$ = " "                          '! 1 Leerzeichen
211  IF tmp%+items% < xmax% THEN
212   tmp% = (xmax%-tmp%) / items%
```

```
213   IF tmp% > 8 THEN tmp% = 8            '! max 8 Blanks
214   leer$ = SPACE$(tmp%)                 '! n Leerzeichen
215  END IF

    '! gebe Menüzeile mit Hauptpunkten aus
216  spalte% = nr%
217  IF spalte% < 1 THEN spalte% = 1
218  IF spalte% > items% THEN spalte% = items%

219  LOCATE 1,1
220  COLOR fcol%,bcol%                    '! normal darstellen
221  PRINT " ";                           '! Leerzeichen
222  FOR i% = 1 TO items%
223   xpos%(i%) = POS (x)                 '! merke x-pos Menüpunkt
224   IF i% = spalte% THEN
225    COLOR bcol%,fcol%                  '! invers darstellen
226   ELSE
227    COLOR fcol%,bcol%                  '! normal darstellen
228   END IF
229   PRINT text$(i%);leer$;              '! Menüpunkt ausgeben
230  NEXT i%

231  COLOR fcol%,bcol%                    '! normal darstellen
232  DO WHILE POS(x) < xmax%              '! Rest mit Blanks füllen
233   PRINT " ";
234  WEND
235   PRINT " ";

    '! Init Variable für Selection
    '! ermittle die Position der invertierten Zeile
    '! diese kann in nr% vorgegeben werden (1 - items%)

236  old% = items%                        '! letzte Spalte

237  IF (nr% < 1) OR (nr% > items%) THEN '!
238   spalte% = 1                         '! 1. Menüpunkt
239  ELSE
240   spalte% = nr%                       '! vorgegebener Punkt
241   IF nr% = items% THEN old% = 1
242  END IF

    '! decodiere Benutzerauswahl
    '! Warte bis CR, ESC, oder CurR/CurL betätigt wurde

243  loopx% = 0

244  WHILE (loopx% = 0)

    '! lese Tastatur aus und decodiere ggf. die Tastencodes

245   zchn$ = INKEY$                      '! lese Tastencodes
246   IF LEN(zchn$) = 2 THEN              '! Extended ASCII-Code
247    zchn$ = RIGHT$(zchn$,1)            '! entferne  1. Zeichen
248    SELECT CASE zchn$
249     CASE = CHR$(80)                   '! Cursor DOWN (unbelegt)
```

```
  250      status% = 3                    '! Exitcode
  251      IF flag% = 1 THEN loopx% = 1   '! Exit
  252     CASE = CHR$(77)                 '! Cursor RIGHT
  253      old% = spalte%                 '! merke Zeile
  254      spalte% = spalte% + 1          '! neue Position
  255      IF spalte% > items% THEN spalte% = 1 '! wrap
  256      status% = 2                    '! Exitcode
  257     CASE = CHR$(75)                 '! Cursor LEFT
  258      old% = spalte%                 '! merke Zeile
  259      spalte% = spalte% - 1          '! neue Position
  260      IF spalte% = 0 THEN spalte% = items% '! wrap
  261      status% = 3
  262     END SELECT
  263    ELSE
  264     SELECT CASE zchn$
  265      CASE = CHR$(13)                '! Return
  266       status%=0                     '! Exitcode CR
  267       loopx% = 1
  268      CASE = CHR$(27)
  269       status%=1                     '! Exitcode ESC
  270       loopx% = 1
  271      END SELECT
  272    END IF

      '! prüfe ob sich die Cursorauswahl gegenüber dem letzten
Durchlauf
      '! verändert hat. Markiere betreffende Auswahlzeile des Menüs
durch
      '! inverse Textdarstellung

  273   LOCATE 1,2                       '! Text neu ausgeben
  274   FOR i% = 1 TO items%
  275    IF i% = spalte% THEN
  276     COLOR bcol%,fcol%              '! invers darstellen
  277    ELSE
  278     COLOR fcol%,bcol%              '! normal darstellen
  279    END IF
  280    PRINT text$(i%);leer$;         '! Menüpunkt ausgeben
  281   NEXT i%
  282   COLOR fcol%,bcol%               '! normal darstellen

  283  WEND
  284  nr% = spalte%                    '! gebe Auswahlcode
zurück

  285 END SUB
      '! Ende INC-File
```

Listing 6.1: Bibliotheksroutinen für Pop-up-Menüs

Ein Anwendungsbeispiel für Pop-up-Menüs

Um den Umgang mit der Menüsteuerung etwas näher zu erläutern, habe ich ein einfaches Beispielprogramm erstellt. Um ein Menü aufzubauen, sind mehrere Schritte erforderlich. Diese sind nachfolgend abstrahiert dargestellt:

```
DIM BUF%(600)             '! Puffer vereinbaren
DIM MEN$(3)               '! Feld für Menütexte

CLS                       '! clear Screen
CALL MenuInit (&HB800)    '! Init Lib, (Colorkarte)

'! Aufbau des 1. Menüs mit 3 Einträgen
MEN$(1) = "Files"         '! Text Menüzeilen
MEN$(2) = "----"
MEN$(3) = "Exit"
weiss% = 7                '! Farbkonstanten
blau%  = 1
'! definiere Textbox (save Screen)
CALL OpenBox (10,3,MEN$(),3,"","",buf%())
'! aktiviere Menübox
CALL PopMenu (10,3,MEN$(),3,weiss%,blau%,"","",2, status%,nr%)
'! schließe Menübox
CALL CloseBox (buf%())
'! prüfe ob status% ok
IF status% < 0 THEN
 '! Fehlerbehandlung
ELSEIF status% = 0 THEN
'! nr% enthält den selektierten Punkt
 SELECT CASE nr%
  CASE = 1
   ....
  CASE = 2
   ....
  CASE = 3
   ....
 END SELECT
END IF
END
```

Das Programm erzeugt ein Pop-up-Menü. Anschließend wird dieses gelöscht und die selektierten Parameter über die CASE-Struktur decodiert.

Das nachfolgende Listing des Beispielprogramme zeigt, wie sich mehrstufige Menüs erzeugen lassen. Solange *CloseBox* nicht aufgerufen wird, bleibt ein Pop-up-Menü sichtbar und kann mit *PopMenu* reaktiviert werden. Wichtig ist lediglich, daß die ursprünglichen Aufrufparameter verwendet werden.

Es wird auch gezeigt, wie die Menü-Funktionen zur Textausgabe nutzbar sind. (Eine eigene Funktion zur Anzeige von Textboxen wird in einem der folgenden Abschnitte vorgestellt). Weiterhin enthält die Demo eine

Sequenz, in der eine Menübox mehrfach neu positioniert wird. Einzelheiten sind dem nachfolgenden Listing zu entnehmen. Zur Erzeugung einer EXE-Datei müssen der Quellcode und die Datei MENU.INC im gleichen Verzeichnis gespeichert sein.

```
X R E F   /Z=50                          (c) Born Version 1.0
Datei : popmenu.bas      Datum : 07-19-1992         Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : POPMENU.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 10.7.92
      '! Autor      : G. Born
      '! Progr. Spr.: PowerBasic
      '! Betr. Sys. : DOS 2.1 - 5.0 (DR-DOS 5.0/6.0)
      '! Funktion: Das Programm wird mit der Eingabe:
      '!
      '!           POPMENUE
      '!
      '!           aufgerufen. Es demonstriert die Verwendung
      '!           der Routinen zur Menüsteuerung (MENU.INC).
      '*********************************************************
      '! definiere Variable
    1 DIM MEN$(7)                              '! Menütexte
    2 nr% = 1                                  '! Zeilennr. Cursor
    3 tmp% = 1

    4 DIM buff1%(600)                          '! temporäre Puffer
    5 DIM buff2%(500)                          '! max. 2004 Elemente
    6 DIM buff3%(500)                          '! für kompletten
    7 DIM buff4%(500)                          '! Screen

    8 DIM xp%(3)                                '! Koordinaten Jump
Menü
    9 DIM yp%(3)                                '! vereinbaren
   10 xp%(1) = 15
   11 xp%(2) = 20
   12 xp%(3) = 30
   13 yp%(1) = 5
   14 yp%(2) = 10
   15 yp%(3) = 15


   16 black% = 0                               '! Farben
   17 blue% = 1
   18 green% = 2
   19 zyan% = 3
   20 red% = 4
   21 magen% = 5
   22 white% = 7

   23 d% = 2                                    '! Doppelrahmen
```

```
   24 e% = 1                                   '! Einfachrahmen

      '! Bildschirm löschen und mit Zeichen füllen

   25 CLS
   26 FOR i% = 1 TO 1999: PRINT "°"; : Next i% '! Screen füllen

      '!-----------------------------------------------------
      '! Init Variable des Menüsystems, der Bildschirmadapter
      '! liegt bei Coloradaptern bei Segmentadr. B800H,
      '!-----------------------------------------------------

   27 CALL MenuInit (&HB800)                   '! Init Variable

   28 done% = 0                                '! Hilfsflag löschen

      '!=====================================================

   29 DO WHILE 1                               '! Schleife über
Menüsystem

   30  Kopf$ = ("1.Menue")                     '! Titeltext für Menübox
   31  MEN$(1) = "1. Submenü"                  '! Texte für Menü
definieren
   32  MEN$(2) = "2. Submenü"
   33  MEN$(3) = "3. Textbox"
   34  MEN$(4) = "4. ---"
   35  MEN$(5) = "5. ---"
   36  MEN$(6) = "6. Jump Menü"
   37  MEN$(7) = "7. Exit"

      '!-----------------------------------------------------
      '! Aufruf des Hauptmenüs mit Kopftext ohne Fußtext, 7 entries
      '! als erstes muß der Fensterbereich gesichert werden
      '! Achtung: dies darf nur 1 x erfolgen, deshalb Flag done
      '!-----------------------------------------------------

   38  IF done% = 0 THEN                       '! Box offen?
   39   CALL OpenBox(8,5,MEN$(),7,kopf$,"",buff1%())
   40   done% = 1                              '! markiere offene Box
   41  END IF

   42  status% = 0
   43  CALL
PopMenu(8,5,MEN$(),7,white%,blue%,kopf$,"",d%,status%,nr%)
   44  tmp% = nr%                              '! merke Selektion
   45  IF status% < 0 THEN                     '! Fehler beim Aufruf?
   46   CLS
   47   PRINT "Fehler: Menübox paßt nicht auf Bildschirm"
   48   END
   49  ELSE
   50   IF status% = 1 THEN                    '! ESC gedrückt?
   51    CALL CloseBox(buff1%())               '! Schließe Box
   52    END                                   '! Ja -> Exit
   53   END IF
```

```
   54  END IF

      '! werte selektierten Menüpunkt in nr% aus
      '! status% = 2 oder 3 wird hier ignoriert und
      '! wirkt daher wie RETURN !!!!

   55  SELECT CASE nr%

   56   CASE = 1                          '! 1. Auswahlcode
Hauptmenü
      '!-------------------------------------------------------
      '! baue ein Submenü mit 3 Einträgen auf
      '!-------------------------------------------------------
   57    MEN$(1) = "Test"                 '! Texte für Menü
definieren
   58    MEN$(2) = "------"
   59    MEN$(3) = "EXIT"

   60    CALL OpenBox(21,8,MEN$(),3,"Sub-Menu1","",buff2%())
   61    CALL PopMenu(21,8,MEN$(),3,red%,blue%,"Sub-Menu1","",d%,
         status%,nr%)
   62    IF status% < 0 THEN               '! Fehler beim Aufruf?
   63     CLS
   64     PRINT "Fehler: Menübox paßt nicht auf Bildschirm"
   65     END
   66    ELSE                '! werte nur status% = 0 aus
   67     IF status% = 0 THEN              '! RETURN gedrückt

      '!-------------------------------------------------------
      '!
      '! Trick: Hier wird die Menüauswahl als Textbox benutzt
      '!
      '!-------------------------------------------------------

   68      MEN$(1) = "Auswahl: " + STR$(nr%)
   69      MEN$(2) = "Bitte ESC- oder RET-Taste betätigen
   70      CALL OpenBox(30,10,MEN$(),2,"","",buff3%())
   71      CALL
PopMenu(30,10,MEN$(),2,zyan%,red%,"","",e%,status%,nr%)
   72      CALL CloseBox(buff3%())               '! close Textbox
   73     END IF
   74    END IF

   75   CALL CloseBox(buff2%())                 '! close Submenü

   76   CASE = 2
        '! baue ein 2. Submenü mit 4 Einträgen auf
   77    head$ = "Sub-Menue 2"            '! Titeltext für Menübox
   78    fuss$ = "Exit->ESC"              '! Fußtext für Menübox
   79    MEN$(1) = "Test 1"               '! Texte für Menü
definieren
   80    MEN$(2) = "Test 2"
   81    MEN$(3) = "Test 3"
   82    MEN$(4) = "EXIT"
```

```
    '! Aufruf des Menüs
 83    CALL OpenBox(15,12,MEN$(),4,head$,fuss$,buff2%())
 84    CALL PopMenu(15,12,MEN$(),4,white%,black%,head$,fuss$,d%,
       status%,nr%)
 85    IF status% < 0 THEN               '! Fehler beim Aufruf?
 86     CLS
 87     PRINT "Fehler: Menübox paßt nicht auf Bildschirm"
 88     END
 89    ELSE
 90     IF status% = 0 THEN              '! RETURN gedrückt
    '!
    '! Trick: Hier wird die Menüauswahl als Textbox benutzt
    '!
 91      MEN$(1) = "Auswahl: " + STR$(nr%)
 92      MEN$(2) = "Bitte ESC- oder RET-Taste betätigen
 93      CALL OpenBox(20,10,MEN$(),2,"","",buff3%())
 94      CALL PopMenu(20,10,MEN$(),2,2,4,"","",e%,status%,nr%)
 95      CALL CloseBox(buff3%())
 96     END IF
 97    END IF

 98    CALL CloseBox(buff2%())

 99   CASE = 3                             '! Textanzeige

100   '! schließe das Hauptmenü
101    CALL CloseBox(buff1%())
102    done%=0                             '! reset flag%


    '!-----------------------------------------------------
    '! baue den Text der Box auf
    '!-----------------------------------------------------

103    MEN$(1) = "Dies ist ein Beispiel für den Aufbau einer"
104    MEN$(2) = "Textbox. Der Text muß auf die einzelnen Zeilen"
105    MEN$(3) = "aufgeteilt werden. Um die Box zu schließen,"
106    MEN$(4) = "drücken Sie bitte die ESC-Taste."

107    CALL OpenBox(20,10,MEN$(),4,"Sub-Menu1","",buff2%())
108    CALL PopMenu(20,10,MEN$(),4,white%,green%,"Sub-Menu1","",
       e%,status%,nr%)
109    IF status% < 0 THEN               '! Fehler beim Aufruf?
110     CLS
111     PRINT "Fehler: Textbox paßt nicht auf Bildschirm"
112     END
113    END IF

114   CALL CloseBox(buff2%())

115   CASE = 4
116   CASE = 5
117   CASE = 6
118    CALL CloseBox(buff1%())             '! schließe Hauptmenü
```

```
      '!-------------------------------------------------------
      '! Demo wie eine BOX verwendet wird (Jump Menü)
      '!-------------------------------------------------------

  119    Kopf$ = ("Jump Menü")              '! Titeltext für Menübox
  120    MEN$(1) = "Punkt 1"                '! Texte für Menü
definieren
  121    MEN$(2) = "Punkt 2"
  122    MEN$(3) = "Punkt 3"

      '! n. Aufrufe des Hauptmenüs an leicht veränderter Position

  123   status% = 0
  124   nr% = 1

  125   DO WHILE status% = 0
  126    tmp% = nr%                 '! Achtung: alte nr% merken
  127 |||     '! und buff4%() verwenden,
  128 |||     '! da buff1() noch benutzt !!!!
  129    CALL OpenBox(xp%(nr%),yp%(nr%),MEN$(),3,kopf$,"",buff4%())
  130    CALL
PopMenu(xp%(nr%),yp%(nr%),MEN$(),3,white%,blue%,kopf$,
        fuss$,e%,status%,nr%)
  131    CALL CloseBox(buff4%())

  132   WEND

  133   CASE = 7
      '! schließe Hauptmenü und terminiere
  134    CALL CloseBox(buff1%())
  135    END
  136  END SELECT

  137  nr% = tmp%                          '! setze
Selektionspunkt

  138 WEND
  139 END

      '! Routinen zur Menüsteuerung einbinden

  140 $INCLUDE "menu.inc"

      '! Ende
```

Listing 6.2: POPMENU.BAS

Ein Anwendungsbeispiel für Pull-down-Menüs

Um den Umgang mit der Menüsteuerung etwas näher zu erläutern, habe ich ein weiteres Beispielprogramm erstellt, welches ein Pull-down-Menü implementiert. Um ein Menü aufzubauen, sind mehrere Schritte erforderlich. Diese sind nachfolgend abstrahiert dargestellt:

- Initialisiere Variable.
- Sichere obere Bildschirmzeile.
- Sichere einen Bereich unterhalb dieser Zeile für die Pull-down-Menüs.
- Gib die Menüzeile aus.
- Werte die Tastencodes aus und erzeuge gegebenenfalls die Pull-down-Boxen oder aktiviere die Anwendung.

Das Programm erzeugt zuerst die oberste Menüleiste. Vorher wird mit *SaveArea* der Bildschirminhalt dieser Zeile gesichert, was aber nicht zwingend erforderlich ist. Da immer nur ein Pull-down-Menü offen ist, reicht es anschließend, einige Zeilen unterhalb der Menüleiste in einem Puffer zu sichern. Dort werden die Menüs sichtbar und können jeweils mit *CloseBox* mit dem Puffer überschrieben werden. Dadurch braucht der Bildschirmbereich nur einmal gesichert zu werden.

Wird in der Menüzeile ein Punkt mit Eingabe selektiert, muß das betreffende Pull-down-Fenster geöffnet werden. Hierzu sind die Routinen der Pop-up-Menüsteuerung zu verwenden. Die x-Position des Fensters findet sich in *xpos%(nr%)*, wobei *nr%* dem Rückgabeparameter von *PullMenü* entspricht. Die y-Position ist fest auf die zweite Zeile eingestellt. Mit diesem Trick ist der Aufbau eines Pull-down-Menüs recht einfach. Das nachfolgende Beispielprogramm zeigt die Einzelheiten der Implementation. Was noch fehlt ist die Möglichkeit, bei geöffnetem Pull-down-Fenster die Cursor-rechts/links-Steuerung anzuwenden. Dann sollte direkt das nächste Fenster geöffnet werden. In der aktuellen Version wird das Fenster geschlossen und zum jeweiligen Menüpunkt der Leiste verzweigt. Das Pull-down-Fenster läßt sich dann über die EingabeTaste öffnen.

Einzelheiten sind nachfolgendem Listing zu entnehmen. Zur Erzeugung einer EXE-Datei müssen der Quellcode und die Datei MENU.INC im gleichen Verzeichnis gespeichert sein.

```
X R E F   /Z=50                              (c) Born Version 1.0
Datei : pullmenu.bas      Datum : 07-19-1992         Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : PULLMENU.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 10.7.92
      '! Autor      : G. Born
      '! Progr. Spr.: PowerBasic
      '! Betr. Sys. : DOS 2.1 - 5.0 (DR-DOS 5.0/6.0)
      '! Funktion: Das Programm wird mit der Eingabe:
      '!
      '!           PULLMENUE
      '!
      '!           aufgerufen. Es demonstriert die Verwendung
```

```
      '!           der Routinen zur Steuerung von PullDown Menüs.
      '!           (MENU.INC).
      '*********************************************************
      '! definiere Variable
    1 DIM MEN1$(3)                          '! Menütexte
    2 DIM MEN2$(3)                          '! Menütexte
    3 DIM xpos%(3)                          '! Position Menüpunkte
    4 DIM buff1%(100)                       '! Buffer für Save
    5 DIM buff2%(2000)                      '! "


    6 black% = 0                            '! Farben
    7 blue% = 1
    8 green% = 2
    9 zyan% = 3
   10 red% = 4
   11 magen% = 5
   12 white% = 7

      '! Bildschirm löschen und mit Zeichen füllen

   13 CLS
   14 FOR i% = 1 TO 1999: PRINT "°"; : Next i% '! Screen füllen

      '!-------------------------------------------------
      '! Init Variable des Menüsystems, der Bildschirmadapter
      '! liegt bei Coloradaptern bei Segmentadr. B800H,
      '!-------------------------------------------------

   15  CALL MenuInit (&HB800)             '! Init Variable

   16  MEN1$(1) = "Files"                  '! Texte für Menü
definieren
   17  MEN1$(2) = "Edit"
   18  MEN1$(3) = "Exit"
   19  items% = 3                         '! 3 Punkte im Hauptmenü

      '!-------------------------------------------------
      '! Aufruf des Menüzeile des Pull Down Menüs
      '!-------------------------------------------------

   20  CALL SaveArea (1,1,80,1,buff1%()) '! sichere 1. Zeile Screen
   21  CALL SaveArea (1,2,80,20,buff2%())'! sichere weitere Zeilen
   22  level% = 0                        '! nur Pull Down Zeile
   23  punkt% = 0                        '! 1. Menüpunkt

   24 DO WHILE 1                         '! Endlosschleife
   25  nr% = punkt%
   26  status% = 1                       '! Exit bei Cursor down
   27  CALL
PullMenu(MEN1$(),items%,white%,blue%,xpos%(),status%,nr%)
   28  punkt% = nr%

   29  IF status% < 0 THEN               '! Fehler beim Aufruf?
   30   CLS
```

```
   IF status% = -1 THEN
    PRINT "Fehler: Menübibliothek nicht initialisiert"
   ELSE
    PRINT "Fehler: Menüzeile paßt nicht auf Bildschirm"
   END IF
   END
  END IF

  IF status% = 1 THEN                '! ESC gedrückt?
   IF level% = 0 THEN                '! EXIT
    CLS
    END
   ELSE
    CALL CloseBox (buff2%())         '! clear Pull Down Box
    level% = 0
   END IF
  END IF

     '! Selektion eines Menüpunktes in der Menüzeile
     '! klappe Menü auf

  IF ((status% = 0) OR (status% = 3)) AND (level% = 0) THEN
   SELECT CASE punkt%

    CASE = 1                          '! 1. Pull Down Menü
     level% = 1
     MEN2$(1) = "Load     "
     MEN2$(2) = "Save"
     MEN2$(3) = "List"
     nr% = 1
     status% = 1                       '! Exit bei Cursor
rechts/links
     CALL
PopMenu(xpos%(punkt%),2,MEN2$(),3,white%,blue%,"","",1,
     status%,nr%)
     IF status% < 0 THEN              '! Fehler beim Aufruf?
      CLS
      PRINT "Fehler: Menübox paßt nicht auf Bildschirm"
      END
     ELSE
      SELECT CASE status%
       CASE = 0                       '! RETURN?
        LOCATE 10,2
        PRINT "Auswahl "; punkt%; ":"; nr%
        PRINT "Bitte eine Taste betätigen"
        DO WHILE INKEY$ = "":WEND
       CASE ELSE
        CALL CloseBox(buff2%())      '! Schließe Box
        level% = 0
        IF status% = 3 THEN          '! Cursor rechts
         punkt% = punkt% + 1
         IF punkt% > items% THEN punkt% = 1
        ELSEIF status% = 2 THEN      '! Cursor links
         punkt% = punkt% - 1
         IF punkt% <= 0 THEN punkt% = items%
```

```
   77         END IF
   78        END SELECT
   79      END IF

   80     CASE = 2
   81      level% = 1
   82      MEN2$(1) = "Cut"
   83      MEN2$(2) = "Paste"
   84      nr% = 1
   85      status% = 1                      '! Exit bei Cursor
rechts/links
   86      CALL
PopMenu(xpos%(punkt%),2,MEN2$(),2,white%,blue%,"","",
         1,status%,nr%)
   87      IF status% < 0 THEN              '! Fehler beim Aufruf?
   88       CLS
   89       PRINT "Fehler: Menübox paßt nicht auf Bildschirm"
   90       END
   91      ELSE
   92       SELECT CASE status%
   93        CASE = 0                       '! RETURN?
   94         LOCATE 10,2
   95         PRINT "Auswahl "; punkt%; ":"; nr%
   96         PRINT "Bitte eine Taste betätigen"
   97         DO WHILE INKEY$ = "":WEND
   98        CASE ELSE
   99         CALL CloseBox(buff2%())       '! Schließe Box
  100         level% = 0
  101         IF status% = 3 THEN           '! Cursor rechts
  102          punkt% = punkt% + 1
  103          IF punkt% > items% THEN punkt% = 1
  104         ELSEIF status% = 2 THEN       '! Cursor links
  105          punkt% = punkt% - 1
  106          IF punkt% <= 0 THEN punkt% = items%
  107         END IF
  108        END SELECT
  109      END IF

  110     CASE 3                             '! Exit
  111      CLS
  112      END
  113    END SELECT
  114   END IF
  115 WEND

  116 END

      '! Routinen zur Menüsteuerung einbinden

  117 $INCLUDE "menu.inc"

      '! Ende
```

Listing 6.3: PULLMENU.BAS

Maussteuerung für PowerBASIC-Programme

Ein weiteres Thema ist die Mausunterstützung in PowerBASIC. Bis zur Version 2.1 ist keinerlei Maussupport vorhanden. Deshalb habe ich eine kleine Bibliothek von Routinen implementiert, die eine Mausansteuerung unter PowerBASIC ermöglichen.

Der Entwurf

Die Ansteuerung der Maus aus Anwenderprogrammen ist unter DOS recht einfach. Es muß lediglich ein Maustreiber (kompatibel zu Microsoft) geladen werden. Dann ist der Treiber aus dem Anwenderprogramm zu aktivieren. Die Ausgabe des Mauscursors übernimmt der Treiber, wobei automatisch Text- und Grafikmodus erkannt und unterstützt werden. Der Treiber selbst stellt eine Kommunikationsschnittstelle unter dem INT 33H zur Verfügung, über die sich einzelne Funktionen ansprechen lassen. Einzelheiten über diese Schnittstelle finden sich in /1/. Aufgabe der Bibliothek ist es nun, die Schnittstelle zwischen dem INT 33H und den PowerBASIC-Anwendungen aufzubereiten. So sollte sich die Initialisierung des Treiber über einen Aufruf der Routine:

```
CALL MouseInit (status%)
```

aus der Anwendung vornehmen lassen. Die aktivierte Bibliotheksroutine wertet dann die Übergabeparameter aus und ruft den Maustreiber über den INT 33H auf. Einzelheiten sind der folgenden Beschreibung zu entnehmen.

Die Implementierung

Die Anbindung der PowerBASIC-Anwendungen an den Maustreiber erfolgt über eine Sammlung von Prozedur- und Funktionsaufrufen, die die Umsetzung für den INT 33H übernehmen. Nicht alle Funktionen, die der Maustreiber anbietet, sind nachfolgend implementiert. Vielmehr wurde sich auf die notwendigen Grundfunktionen beschränkt. Erweiterungen sind aber leicht durchführbar. Alle Module werden in der Datei MAUS.INC abgelegt und lassen sich per INCLUDE-Anweisung:

```
$INCLUDE "MAUS.INC"
```

in die Anwendung integrieren. Nachfolgend möchte ich kurz die Schnittstellen zu den einzelnen Routinen vorstellen.

MausInit (status%)

Diese Routine muß als erstes zur Initialisierung des Maustreibers aufgerufen werden. Nur dann lassen sich weitere Funktionen ansprechen. *status%* gibt das Ergebnis des Aufrufes an. Ist der Rückgabewert 0, wurde der Treiber erkannt und initialisiert. Mit *status% = 1* als Rückgabewert ist kein Treiber vorhanden.

ShowCursor

Dieser Prozeduraufruf veranlaßt, daß der Treiber den Mauszeiger auf dem Bildschirm einblendet. Die Zeigerform wird dabei in Abhängigkeit vom Bildschirmmodus (Text oder Grafik) gewählt.

HideCursor

Dieser Prozeduraufruf schaltet den Mauszeiger ab, d.h. das Symbol ist nicht mehr sichtbar.

GetPara (x%, y%, button%)

Dieser Prozeduraufruf fragt die folgenden Parameter der Maus ab:

x%	aktuelle x-Position des Cursors
y%	aktuelle y-Position des Cursors
button%	Zustand der Maustasten

Die beiden Koordinaten *x%,y%* werden dabei in Mausschritten (*Mikeys*) angegeben. Diese Einheit läßt sich über einen INT 33-Aufruf verändern, der aber in der Bibliothek nicht implementiert wurde. In *button%* findet sich die Information, welche Maustasten gerade gedrückt sind:

Bit	Taste
0	linke Maustaste
1	rechte Maustaste
2	mittlere Maustaste

Ist das betreffende Bit = 1, wurde die Taste gerade gedrückt. Die mittlere Maustaste ist nur bei Mäusen mit drei Tasten vorhanden.

XPos%

Die Funktion gibt die x-Position des Mauszeigers zurück.

YPos%

Die Funktion gibt die y-Position des Mauszeigers zurück.

Buttons%

Die Funktion gibt den Zustand der Maustasten zurück. Es gilt die Codierung wie bei der Prozedur *GetPara*.

SetXY (x%, y%)

Die Prozedur setzt den Mauszeiger auf die in den Parametern *x%, y%* angegebenen Koordinaten. Diese sind in Mausschritten anzugeben.

SetXRange (xmin%, xmax%)

Die Prozedur setzt ein Intervall auf der X-Achse, innerhalb dessen sich der Mauszeiger bewegen darf. Zusammen mit *SetYRange* läßt sich so ein Fenster für den Mauszeiger definieren. Dieses Fenster kann dann mit dem Zeiger nicht mehr verlassen werden.

SetYRange (ymin%, ymax%)

Die Prozedur setzt ein Intervall auf der Y-Achse, innerhalb dessen sich der Mauszeiger bewegen darf. Zusammen mit *SetXRange* läßt sich so ein Fenster für den Mauszeiger definieren. Dieses Fenster kann dann mit dem Zeiger nicht mehr verlassen werden.

PenEmul (switch%)

Der Maustreiber kann einen Lichtgriffel emulieren. Über die Prozedur *PenEmul* läßt sich die Emulation ein- oder ausschalten. Der Wert von *switch%* bestimmt dabei den Emulationsmode:

```
switch% = 0  Emulation ausgeschaltet
switch% = 1  Emulation eingeschaltet
```

Die Routine wird im folgenden Beispielprogramm allerdings nicht verwendet.

SetTCursor (ymin%, ymax%)

Der Maustreiber wird in Abhängigkeit vom Anzeigemodus der Bildschirmkarte den Cursor darstelen:

```
Pfeil|Graphikmode
Viereck|Textmode
```

Das Anwendungsprogramm kann nun die Form des Cursors neu definieren. Für den Grafikmodus ist die Funktion 09H des INT 33H aufzurufen. Für den Textmodus läßt sich die Prozedur *SetTCursor* aktivieren. Die beiden Parameter *ymin%* und *ymax%* geben die Zahl der Rasterzeilen für den Cursor an.

Damit möchte ich die Beschreibung der Mausbibliothek beenden. Nicht alle Funktionen des INT 33H wurden implementiert. Falls Sie die Bibliothek erweitern möchten, kann dies analog zu den bereits vorgestellten Routinen erfolgen. Eine detaillierte Beschreibung der INT 33H-Aufrufe findet sich in den Literaturhinweisen (/1/). Einzelheiten bezüglich der Mausbibliothek sind dem folgenden Listing zu entnehmen:

```
X R E F   /Z=50                              (c) Born Version 1.0
Datei : mouse.inc      Datum : 07-19-1992         Seite :  1

 Zeile     Anweisung

      '!#####################################################
      '! File:    MOUSE.INC
      '! Version: 1.0 v. 10.7.92 (c) G. Born
      '!          Subroutinen zur Maussteuerung
      '!#####################################################

    1 SUB MouseInit (status%)
      '!-----------------------------------------------------
      '!  Initialisierung der Maus
      '!  status% = 0 -> ok, 1 -> keine Maus
```

```
   '!-----------------------------------------------------
 2  REG 1,0
 3  CALL INTERRUPT &H33
 4  status% = 0
 5  IF REG(1) <> -1 THEN status% = 1
 6 END SUB

 7 SUB ShowCursor
   '!-----------------------------------------------------
   '!  mache Mauscursor sichtbar
   '!-----------------------------------------------------
 8  REG 1,1
 9  CALL INTERRUPT &H33
10 END SUB

11 SUB HideCursor
   '!-----------------------------------------------------
   '!  mache Mauscursor unsichtbar
   '!-----------------------------------------------------
12  REG 1,2
13  CALL INTERRUPT &H33
14 END SUB

15 SUB GetPara (x%, y%, button%)
   '!-----------------------------------------------------
   '!  lese Mausstatus
   '!  x%, y%  = Koordinate Mauszeiger
   '!  button% = Tastenstatus
   '!              Bit 0 = 1 linke Taste gedrückt
   '!              Bit 1 = 1 rechte Taste gedrückt
   '!              Bit 2 = 1 mittlere Taste gedrückt
   '!-----------------------------------------------------
16  REG 1,3
17  CALL INTERRUPT &H33
18  x% = REG(3)
19  y% = REG(4)
20  button% = REG(2)
21 END SUB

22 Def FN XPos%
   '!-----------------------------------------------------
   '!  lese X-Position Mauscursor
   '!-----------------------------------------------------
23  REG 1,3
24  CALL INTERRUPT &H33
25  FN XPos% = REG(3)
26 END DEF

27 Def FN YPos%
   '!-----------------------------------------------------
   '!  lese Y-Position Mauscursor
   '!-----------------------------------------------------
28  REG 1,3
29  CALL INTERRUPT &H33
30  FN YPos% = REG(4)
```

```
31 END DEF

32 Def FN Button%
   '!-----------------------------------------------------
   '!  Mausknopf gedrückt?
   '!  Wert > 0 ja, Bitcodierung wie GetParam
   '!  Bit 0 = links, Bit 1 = rechts, Bit 2 = Mitte
   '!-----------------------------------------------------
33  REG 1,3
34  CALL INTERRUPT &H33
35  Button% = REG(2)
36 END DEF

37 SUB SetXY (x%, y%)
   '!-----------------------------------------------------
   '!  setze Mausposition
   '!  x%, y%  = Koordinate Mauszeiger
   '!-----------------------------------------------------
38  REG 1,4
39  REG 3,x%
40  REG 4,y%
41  CALL INTERRUPT &H33
42 END SUB

43 SUB SetXRange (xmin%, xmax%)
   '!-----------------------------------------------------
   '!  setze Mausintervall xmin .. xmax
   '!-----------------------------------------------------
44  REG 1,7
45  REG 2,0
46  REG 3,xmin%
47  REG 4,xmax%
48  CALL INTERRUPT &H33
49 END SUB

50 SUB SetYRange (ymin%, ymax%)
   '!-----------------------------------------------------
   '!  setze Mausintervall ymin .. ymax
   '!-----------------------------------------------------
51  REG 1,8
52  REG 2,0
53  REG 3,ymin%
54  REG 4,ymax%
55  CALL INTERRUPT &H33
56 END SUB


57 SUB PenEmul (switch%)
   '!-----------------------------------------------------
   '!  schalte Light Pen Emulation:
   '!  switch% = 0 -> aus,  = 1 -> ein
   '!-----------------------------------------------------

58  IF switch% = 0 THEN
59   REG 1,&H0E              '! aus
```

```
60  ELSE
61   REG 1,&H0D              '! on
62  END IF
63  CALL INTERRUPT &H33
64 END SUB

65 SUB SetTCursor (ymin%, ymax%)
   '!-------------------------------------------------
   '!  setze Größe des Mauscursors im Textmode xmin .. xmax
   '!-------------------------------------------------
66  REG 1,10
67  REG 2,1
68  REG 3,xmin%
69  REG 4,xmax%
70  CALL INTERRUPT &H33
71 END SUB

   '! ENDE
```

Listing 6.4: Bibliotheksroutinen zur Maussteuerung

Ein Beispielprogramm zur Maussteuerung in PowerBASIC

Aufbauend auf der oben gezeigten Bibliothek möchte ich nun ein kurzes Beispielprogramm vorstellen, welches den Umgang mit den Routinen der Mausbibliothek aufzeigt. Das Programm läßt sich wahlweise im Text- oder Grafikmodus betreiben, so daß die unterschiedlichen Cursorsymbole sichtbar werden. Die Abfrage bezüglich des Modus erfolgt nach dem Programmstart.

Als erster Schritt muß der Maustreiber initialisiert werden. Dies geschieht über die Anweisung:

```
CALL MouseInit (status%)
```

Fehlt der Maustreiber, bricht das Programm mit einer Fehlermeldung ab. Nach erfolgreichem Aufruf lassen sich die einzelnen Routinen jederzeit aktivieren. Um den Cursor sichtbar zu machen, ist die Prozedur *ShowCursor* aufzurufen. Soll der Cursor abgeschaltet werden, ist die Routine *HideCursor* zu aktivieren. Nach einem Aufruf von *HideCursor* läßt sich die Anzeige mit *ShowCursor* wieder einblenden.

Anschließend wird der Anzeigebereich für den Mauscursor auf ein Fenster innerhalb des Bildschirms mit den Prozeduren *SetXRange* und *SetYRange* begrenzt. Die Größenangaben erfolgen in Mausschritten (1 Mikey = 1/1200 Zoll). Anschließend wird *ShowCursor* aufgerufen.

Der nächste Abschnitt demonstriert, wie sich der Mauscursor mit der Prozedur *SetCursor* auf dem Bildschirm positionieren läßt. Der Cursor wird in Schritten quer über den Bildschirm verschoben. Dann enthält das Beispielprogramm eine Schleife, die nur durch Drücken der Esc-Taste beendet werden kann. Innerhalb der Schleife wird der aktuelle Mauszustand zyklisch abgefragt. Die Position (x,y) sowie der Tastenstatus

werden anschließend auf dem Bildschirm angezeigt. Mit diesem kurzen Beispiel läßt sich aufzeigen, wie einfach die Maussteuerung in PowerBASIC zu integrieren ist. Weitere Einzelheiten sind nachfolgendem Listing zu entnehmen:

```
X R E F   /Z=50                              (c) Born Version 1.0
Datei : maus.bas      Datum : 07-19-1992         Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : MAUS.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 10.7.92
      '! Autor      : G. Born
      '! Progr. Spr.: PowerBasic
      '! Betr. Sys. : DOS 2.1 - 5.0 (DR-DOS 5.0/6.0)
      '! Funktion: Das Programm wird mit der Eingabe:
      '!
      '!           MAUS
      '!
      '!           aufgerufen. Es demonstriert die Verwendung
      '!           der Routinen zur Maussteuerung (MAUS.INC).
      '*********************************************************
      '!

    1 CLS
    2 PRINT "Graphik (J/N) ?"
    3 tmp$ = INPUT$ (1)
    4 IF (UCASE$(tmp$) = "J") THEN SCREEN 2 '! Graphik ein
    5 CLS

    6 CALL MouseInit (status%)              '! Init Mauslib

    7 IF status% <> 0 THEN                  '! Init ok?
    8  PRINT "Keine Maus vorhanden"
    9  END
   10 END IF

      '! setze Fenster für Cursor (in Pixeln)

   11 CALL SetXRange (5,610)
   12 CALL SetYRange (30,150)

   13 CALL ShowCursor                        '! Cursor einblenden

      '! Cursor horizontal über Screen bewegen

   14 FOR i% = 10 TO 600
   15  CALL SetXY (i%,i% / 6)                '! setze Cursor
   16 NEXT i%

   17 LOCATE 1,5
   18 PRINT "Mausdemo:     Abbruch mit der ESC-Taste"
```

```
19 LOCATE 22,5
20 PRINT "Cursorposition : "


21 WHILE INKEY$ <> CHR$(27)

22   CALL GetPara (x%, y%, button%)   '! lese Status

23   LOCATE 22,22
24   PRINT x%; " ";y%

25   LOCATE 23,5
26   IF (button% AND 1) > 0 THEN
27    PRINT " linke Maustaste "
28   ELSE
29    PRINT "                 "
30   END IF

31   LOCATE 23, 20
32   IF (button% AND 2) > 0 THEN
33    PRINT " rechte Maustaste "
34   ELSE
35    PRINT "                  "
36   END IF

37 WEND

38 END

   '! Mauslib einbinden

39 $INCLUDE "MOUSE.INC"

   '! Ende
```

Listing 6.5: Ein Beispiel zur Maussteuerung

BIOS-Spielereien

Die Verbindung zwischen dem Betriebssystem (DOS) und der Hardware wird durch eine eigene Softwareschicht realisiert. Innerhalb dieser Software erfolgt die Anpassung des maschinenunabhängigen Betriebssystems an herstellerspezifische Hardwarekonfigurationen. Diese Software wird als BIOS (Basic Input Output System) bezeichnet. Für PowerBASIC-Programmierer sind die BIOS-Funktionen in der Regel unbekannt, da keine Zugriffe auf diese Funktionen erfolgen. Allerdings gibt es einige recht nützliche Funktionen im BIOS, die die Möglichkeiten für PowerBASIC-Programme erweitern. Nachfolgend möchte ich einige Module vorstellen, die oft verblüffend einfach aber dennoch praktisch verwertbar sind.

NUMOFF: Abschalten der NUMLOCK-Taste

Bei Rechnern mit MF2-Tastatur (sie besitzen einen separaten Block mit numerischen Tasten) schaltet das BIOS bei jedem Systemstart automatisch die NUMLOCK-Funktion ein. Dies wird an der entsprechenden Leuchtdiode sichtbar, die beim Start aktiv wird. Die Tasten geben dann bei einer Betätigung die Ziffern 0..9 zurück. Sollen die betreffenden Tasten jedoch zur Cursorsteuerung verwendet werden, muß zuerst die NUMLOCK-Taste manuell abgeschaltet werden.

Nachdem der erste Rechner mit MF2-Tastatur auf meinem Schreibtisch stand, trat permanent ein Problem auf: Ich benutze den Ziffernblock meist zur Cursorsteuerung. Wurde das System gestartet, war jedoch die numerische Funktion aktiv. Bei Aufruf einer Tabellenkalkulation, einer Textverarbeitung oder einer Datenbank wurden dann meist die ersten Einträge durch Ziffern überschrieben, wenn ich die »Cursorsteuerung« benutzte. Da der Ansatz recht interessant ist, möchte ich nachfolgend die nach PowerBASIC portierte Lösung vorstellen.

Die Grundlagen

Beim Systemstart überprüft das BIOS die Hardware/Systemkonfigurierung und nimmt verschiedene Initialisierungen vor. Dabei wird auch die NUMLOCK-Funktion eingeschaltet. Die Systemkonfiguration wird teilweise in einem eigenen BIOS-Datenbereich gespeichert. Dieser Datenbereich umfaßt 256 Byte und liegt an den Adressen 0000:0400 bis 0000:04FF. (Die genaue Beschreibung ist in /1/ enthalten.)

Das Tastatur-Statusflag

Beim Systemstart schaltet das BIOS die Tastatur in einen bestimmten Zustand, der anschließend in einem Flag ab Adresse 0000:0417 im BIOS-Datenbereich gespeichert wird. Dieses Flag besitzt dabei die Codierung gemäß Bild 6.5.

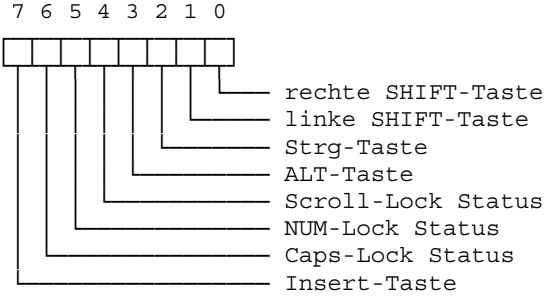


Bild 6.5: Tastaturstatusflag

Die einzelnen Bits geben an, ob die jeweilige Taste gedrückt wurde (Bit = 1), bzw. ob die betreffende Funktion aktiv (Bit = 1) oder abgeschaltet (Bit = 0) ist. Das BIOS überprüft zyklisch dieses Flag und stellt die Tastatur bei Änderungen um.

Interessant ist in diesem Zusammenhang Bit 5, welches den Zustand des NUMLOCK-Schalters anzeigt. Ist das betreffende Bit gesetzt, ist die NUMLOCK-Funktion aktiv. Wird das betreffende Flag gelöscht, schaltet das BIOS die NUMLOCK-Funktion wieder ab. Dies ist aber genau die benötigte Funktion um NUMLOCK per Software abzuschalten.

Die Implementierung

Mit obigem Wissen ist die Implementierung ein Kinderspiel. Das Programm erhält den Namen NUMOFF.BAS und besteht nur aus wenigen Anweisungen. Deshalb kann auf die Angabe von Hierarchiediagrammen etc. verzichtet werden. Das betreffende Flag wird mit:

```
DEF SEG &H40
a% = PEEK (&H0017)
```

gelesen. Beachten Sie, daß das korrekte Datensegment gesetzt werden muß (DEF SEG). Ich habe auf die Segmentangabe &H0000 verzichtet, da diese die 20-Bit-Adressierung in PowerBASIC einschaltet. Vielmehr wird das Segment 40H und der Offset 17H verwendet, was ebenfalls die Adresse 0000:0417 ergibt. Dann wird Bit 5 mittels der AND-Anweisung:

```
a% AND &HDF
```

gelöscht und das Ergebnis wieder in die Speicherzelle zurückgeschrieben. Abschließend muß noch das ursprüngliche Segment mit DEF SEG restauriert werden.

Nachdem das Programm in eine EXE-Datei übersetzt wurde, muß es als letzte Anweisung in die AUTOEXEC.BAT-Datei aufgenommen werden. Dann wird die NUMLOCK-Funktion bei jedem Systemstart abgeschaltet. Damit ist die Aufgabe bereits gelöst, Einzelheiten sind dem nachfolgenden Listing zu entnehmen.

Erweiterungsvorschläge

Sie können das Programm so modifizieren, daß sich die NUMLOCK-Taste wahlweise ein- oder ausschalten läßt. Weiterhin ist es denkbar, auch die anderen Bits des Tastaturflags über entsprechende Parameter zu modifizieren.

```
X R E F   /Z=50                              (c) Born Version 1.0
Datei : numoff.bas      Datum : 07-10-1992          Seite :  1

 Zeile     Anweisung

      '******************************************************
      '! File       : NUMOFF.BAS
```

```
'! Vers.      : 1.0
'! Last Edit  : 16.6.92
'! Autor      : G. Born
'! Progr. Spr.: PowerBasic
'! Betr. Sys. : DOS 2.1 - 5.0 (DR-DOS 5.0/6.0)
'! Funktion: Das Programm wird mit der Eingabe:
'!
'!           NUMOFF
'!
'!           aufgerufen. Es schaltet die NUMLOCK-Funktion
'!           der Tastatur ab und gibt eine entsprechende
'!           Meldung auf dem Bildschirm aus.
'*************************************************************
'! Benutzernachricht ausgeben

1   PRINT "NUMOFF            (c) Born Version 1.0"
2   PRINT
3   PRINT "NUMLOCK-Funktion abgeschaltet"

4   DEF SEG = &H40               '! setze Segmentadresse
5   a% = PEEK (&H0017)           '! lese Flag
6   POKE &H0017, (a% AND &HDF)   '! clear Bit 5 und speichern
7   DEF SEG                      '! altes Segment

'***** Programm Ende ******
```

Listing 6.6: NUMOFF.BAS

LPTSWAP: Vertauschen der Druckerausgänge

Ein anderes Problem tritt häufiger auf, wenn mehrere Drucker an einem Rechner angeschlossen sind. Zum Beispiel befindet sich ein Laserdrucker an LPT1: und ein Nadeldrucker an LPT2:. Soll nun wahlweise auf den beiden Geräten gedruckt werden, muß die Anwendung dies unterstützen. Aber nicht jedes Anwendungsprogramm bietet diese Möglichkeit. Bestes Beispiel ist der LPRINT-Befehl, der sich auf LPT1: bezieht. Ist an LPT1: nun der Nadeldrucker angeschaltet und wollen Sie auf dem Laserdrucker ausgeben, müssen die Anschlußkabel am Rechner umgesteckt werden. Dies ist lästig und führt bei häufiger Durchführung auch zum Materialverschleiß. Eine Alternative wäre ein Durckerumschalter, der jedoch einiges kostet und nicht immer funktioniert (bei Laserdruckern kann es elektrische Probleme geben). Die Idee, die Ausgabe über das DOS-MODE-Kommando von LPT1: auf LPT2: umzuleiten, funktioniert nur, falls die Software die DOS-Ausgaberoutinen des INT 21H benutzt. Viele Programme greifen jedoch direkt auf die Ausgaberoutinen des BIOS-INT 17H zurück. Dann funktioniert die Umleitung jedoch nicht mehr. Ähnliche Überlegungen gelten auch für die seriellen Schnittstellen (COM1: bis COM4:). Hier wäre ein Programm hilfreich, welches softwaremäßig die beiden Schnittstellen LPT1: und LPT2: vertauscht. Ausgehend von dieser Überlegung entstand bei mir vor Jahren ein kurzes Programm, welches

genau diese Aufgabe übernimmt. Nachfolgend möchte ich die Version für PowerBASIC vorstellen.

Der Entwurf

Um die Aufgabe zu lösen, sind wieder detaillierte Kenntnisse des DOS-Betriebssystems und des BIOS erforderlich. Jede Druckerausgabe einer Anwendung wird normalerweise über die INT 21-Funktionen abgewickelt. Diese Funktionen bedienen sich des BIOS-INT 17, um einzelne Zeichen an den Ausgabeport LPTx: zu schicken. Einzelne Anwenderprogramme greifen ebenfalls direkt auf den INT 17 zu.

Das BIOS verwaltet nun in seinem BIOS-Datenbereich die Informationen über die Hardwarekonfiguration. Beim Systemstart wird zum Beispiel festgestellt, wieviele Schnittstellen das System besitzt und unter welchen Adapteradressen die Ausgabeports LPTx: und COMx: angesprochen werden können. Diese Informationen werden dann ab der Adresse 0000:0400 wortweise im BIOS-Datenbereich gespeichert. Tabelle 6.3 enthält eine Aufstellung der Belegung dieses Datenbereiches.

Adr.	Bedeutung
0:0400	Portadresse COM1 (meist 03F8H)
0:0402	Portadresse COM2 (meist 02F8H)
0:0404	Portadresse COM3
0:0406	Portadresse COM4
0:0408	Portadresse LPT1 (meist 03BCH)
0:040A	Portadresse LPT2 (meist 03B8H)
0:040C	Portadresse LPT3 (meist 02B8H)
0:040E	Portadresse LPT4

Tabelle 6.3: Adressen von LPTx: und COMx: im BIOS-Datenbereich

Ist der betreffende Adapter nicht vorhanden, trägt das BIOS unter der zugehörigen Adresse den Wert 0000 ein. Mit einem Debugger läßt sich dann leicht feststellen, welche Karten vorhanden sind.

Für unseren Zweck ist aber noch eine weitere Eigenschaft interessant. Das BIOS prüft zum Beispiel vor der Ausgabe auf LPT1: die Adresse des Adapters (0000:0408). Steht dort der Wert 03BCH, erfolgt die Ausgabe auf die Karte mit dieser Adresse. Wird aber unter 0000:0408 der Wert 03B8H eingetragen, leitet das BIOS einfach die Zeichen an eine zweite Karte mit der betreffenden Adresse weiter. Durch einfaches Umsetzen der BIOS-Adressen läßt sich offenbar die Eingabe umleiten. Was deshalb benötigt wird, ist lediglich ein einfaches Programm, welches die betreffenden Einträge im BIOS-Datenbereich modifiziert.

Die Implementierung

Für unsere Aufgabe muß das Programm lediglich die Portadressen für LPT1: und LPT2: vertauschen. Eine Anpassung an andere LPTx-Ausgänge ist leicht durchführbar. Selbst bei COM-Ausgängen funktioniert die Sache. Was allerdings nicht geht ist die Umleitung von COM-Ausgängen auf LPT-Ausgänge.

Der Zugriff auf die Daten des BIOS kann über die PEEK- und POKE-Anweisungen erfolgen. Da ich in den nachfolgenden Beispielen auch Assemblerprogramme benutze, möchte ich in diesem kleinen Beispiel mit INLINE-Codes arbeiten. Das Programm wird in Maschinensprache codiert und in einer Prozedur implementiert. Einzelheiten sind dem folgenden Listing zu entnehmen.

Sofern Sie sich für die Assemblerprogrammierung interessieren, möchte ich auf /8/ verweisen. Dort wird auch eine Fassung von LPTSWAP in Assembler vorgestellt. Weiterhin gibt es eine Reihe von weiterer Literatur zu Assembler. Wer einen preisgünstigen Assembler sucht, sei auf den A86-Assembler (Shareware) und das Begleitbuch /9/ aus dem Systhema-Verlag verwiesen. Weiterhin ist für die Version 3.0 von PowerBASIC ein integrierter Assembler vorgesehen. Ich plane zu dieser Version ein eigenes Buch herauszugeben, welches die Thematik behandelt. In den nachfolgenden Abschnitten und im Anhang erfahren Sie zusätzlich, wie Sie zumindest kleinere Programme auch ohne Assembler bearbeiten können, ohne jedesmal mühsam die Maschinencodebefehle einzugeben.

Da das Programm sehr einfach gehalten ist (es besteht nur aus wenigen Zeilen), möchte ich direkt das Listing vorstellen. Die Assembleranweisungen wurden in Maschinencode als INLINE integriert. Die Befehle und deren Funktion sind in jeder Zeile als Kommentar aufgeführt.

```
X R E F   /Z=55                              (c) Born Version 1.0
Datei : lptswap.bas      Datum : 07-19-1992          Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : LPTSWAP.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 16.7.92
      '! Autor      : G. Born
      '! Progr. Spr.: PowerBasic
      '! Betr. Sys. : DOS 2.1 - 5.0 (DR-DOS 5.0/6.0)
      '! Funktion: Das Programm wird mit der Eingabe:
      '!
      '!           LPTSWAP
      '!
      '!           aufgerufen. Es vertauscht LPT1 und LPT2
      '!           und gibt eine entsprechende Meldung
      '!           auf dem Bildschirm aus.
      '*********************************************************
```

```
      '! Benutzernachricht ausgeben

    1   PRINT "LPTSWAP            (c) Born Version 1.0"
    2   PRINT
    3   PRINT "LPT1: und LPT2: vertauscht"

    4   CALL LPTSwap                '! vertausche Schnittstellen
    5   END

      '!----------------------------------------------------------
      '! Hilfsroutinen
      '!----------------------------------------------------------

    6 SUB LPTSwap
      '!----------------------------------------------------------
      '! Die Routine vertauscht die Portadressen von LPT1 / LPT2
      '! im BIOS-Datenbereich (Adr. 0:408 und 0:40A).
      '!
      '! Implementiere die Routine als Assemblerprozedur
      '!----------------------------------------------------------

    7 $INLINE &H1E                  '! PUSH DS     ; merke
Datensegment
    8 $INLINE &H31, &HC0            '! XOR AX,AX   ; Setze DS = 0
    9 $INLINE &H8E, &HD8            '! MOV DS,AX   ; "
   10 $INLINE &HA1, &H08, &H04      '! MOV AX,[408]; Adr. LPT1
lesen
   11 $INLINE &H8B, &H1E, &H0A, &H04 '! MOV BX,[40A]; Adr. LPT2
lesen
   12 $INLINE &H89, &H1E, &H08, &H04 '! MOV [40A],BX; Adr. LPT2
setzen
   13 $INLINE &HA3, &H0A, &H04      '! MOV [408],AX; Adr. LPT1
setzen
   14 $INLINE &H1F                  '! POP DS      ; restauriere
DS

   15 END SUB

      '***** Programm Ende ******
```

Listing 6.7: LPTSWAP.BAS

Der BIOS-Kommunikationsbereich

Ein weiteres Problem betrifft die Übergabe von Informationen zwischen verschiedenen Programmen. So kann ein Programm zum Beispiel bestimmte Initialisierungen (z.B. Druckereinstellungen) vornehmen, die von anderen Programmen mit benutzt werden. Denkbar ist nun, daß das erste Programm nach der Initialisierung ein Flag im Rechner setzt, so daß andere Programme erkennen, daß die Initialisierung bereits erfolgt ist. Allerdings bleibt das Problem, wie diese Information zwischen den Programmen übergeben werden kann. Denkbar ist es, die Information in einer Datei zu speichern. Das Problem besteht aber darin, daß die Datei

auch nach dem Abschalten des Rechners erhalten bleibt, die Initialisierung aber verloren geht. Beim Start müßte die entsprechende Datei gelöscht werden. Zwar geht dies über ein entsprechendes Kommando in der AUTOEXEC.BAT-Datei, aber die Lösung ist nicht zuverlässig. Besser wäre es, die Signatur im RAM abzulegen. Dieses RAM wird bei jedem Rechnerneustart gelöscht. Doch wo lassen sich die Daten speichern?. Der Speicherbereich des Programmes wird freigegeben, sobald das Programm endet. Der Schreibzugriff auf den Umgebungsbereich funktioniert aus PowerBASIC nicht. Hier hilft nur noch die Kenntnis bezüglich der Belegung des BIOS-Datenbereiches weiter. Die Adressen:

```
0000:04F0 .. 0000:04FF
```

werden vom BIOS als »Kommunikationsbereich für Zwischenanwendungen« reserviert. Damti sind 16 Byte verfügbar, die bei jedem Programmstart gelöscht werden. Um auf den Bereich zuzugreifen, lassen sich PEEK- und POKE-Anweisungen verwenden.

```
DEF SEG &H4F
FOR i% = 0 TO 15
 PRINT HEX$(PEEK(i%))
NEXT i%
```

werden die 16 Byte als Hexzahlen auf dem Bildschirm ausgegeben.

Sofern Sie also einen Datenbereich zur Parameterübergabe benötigen, können Sie die 16 Byte benutzen.

XPARK: Parken der Festplatte

Ein anderes »Problem**!Syntaxfehler, DER** geschieht dies in einem Bereich wo keine Daten vorhanden sind. Viele Systeme besitzen hierzu ein eigenes Programm, welches meist mit dem Namen PARK versehen ist. Falls Sie jedoch nicht über ein solches Programm verfügen, können Sie nachfolgendes Utility für diesen Zweck verwenden.

Der Entwurf

Das Programm erhält den Namen XPARK (zur Unterscheidung von PARK) und soll alle Schreib-/Leseköpfe von Festplatten in eine Parkposition bringen. Hierzu sind mehrere Schritte nötig:

- Ermittle die Zahl der Festplatten.
- Positioniere die Köpfe in einem sicheren Bereich.
- Gib eine Benutzernachricht zur Abschaltung des Rechners aus.

Wichtig ist vor allem, daß der Rechner bei laufendem PARK-Programm abgeschaltet wird. Wurde die Parkposition angefahren, verschiebt die Rückkehr nach DOS sofort die Köpfe wieder in eine andere Position (das DOS-Programm COMMAND.COM muß ja geladen werden). Deshalb darf

das Programm nicht enden, sondern muß in einer Endlosschleife verbleiben.

Die Zahl der Laufwerke sowie die Verschiebung der Köpfe kann über die Routinen des BIOS-INT 13H erfolgen. Über diesen Interrupt erfolgt die Ansteuerung von Disketten und Festplatten. Die Zahl der Festplatten im System läßt sich über den INT 13 mit AH = 08H und DX = 80H abfragen. Die Köpfe werden über die Funktion AH = 0CH geparkt. Vorher müssen die Laufwerksparameter ermittelt werden. Dies erfolgt über die Funktion AH = 08H, wobei in DX der Wert 80H plus die Laufwerksnummer (1, 2, 3 etc.) übergeben wird. Der Aufruf gibt die Laufwerksdaten (Zahl der Köpfe, Zahl der Zylinder etc.) zurück. Dann werden diese Daten der Funktion 0CH übergeben. Der Kopf des Laufwerkes wird auf die letzte Spur positioniert. Weitere Einzelheiten bezüglich der Schnittstelle des INT 13H finden sich in /1/.

Die Implementierung

Das Programm ist recht einfach aufgebaut, so daß hier nur kurz die verschiedenen Module besprochen werden.

drivesx

Aufgabe dieses Unterprogramms ist es, die Zahl der Festplatten im System zu ermitteln. Es wird die Funktion AH = 08H des INT 13 verwendet. Das Ergebnis wird als Parameter an das rufende Programm zurückgegeben.

park

Diese Routine parkt das im Parameter *lw%* angegebene Laufwerk, indem die Köpfe auf die letzte Spur verschoben werden. Hierzu benutzt das Programm die Funktion AH = 08H des INT 13, um die Parameter zu ermitteln. Dann wird der Kopf durch die Funktion AH = 0CH des INT 13 geparkt.

Das Hauptprogramm ermittelt die Zahl der Festplattenlaufwerke, gibt eine Meldung auf dem Bildschirm aus und parkt dann alle Köpfe. Anschließend verzweigt der Programmablauf in eine Endlosschleife. Wird nun der Rechner abgeschaltet, sind die Köpfe geparkt. Falls eine Taste betätigt wird, endet das Programm mit einer Benutzermeldung. Die Köpfe sind daraufhin nicht mehr geparkt. Weitere Einzelheiten sind dem folgenden Listing zu entnehmen.

```
X R E F   /Z=50                          (c) Born Version 1.0
Datei : park.bas      Datum : 07-19-1992         Seite :  1

 Zeile     Anweisung

      '******************************************************
      '! File       : PARK.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 16.5.92
      '! Autor      : G. Born
```

```
   '! Files      : INPUT, OUTPUT
   '! Progr. Spr.: PowerBasic
   '! Betr. Sys. : DOS 2.1 - 5.0
   '! Funktion: Das Programm wird mit der Eingabe:
   '!
   '!           PARK
   '!
   '!           aufgerufen und parkt die Festplattenköpfe.
   '************************************************************
   '! Variable definieren
 1 drive% = 0

 2 ON ERROR GOTO fehler

 3 PRINT
 4 PRINT "P A R K                    (c) Born Version 1.0"
 5 PRINT

 6 CALL DRIVESX                            '! Laufwerkszahl

 7 PRINT "Es wurden ";drive%;" Platten erkannt"
 8 PRINT

 9 SELECT CASE drive%
10  CASE 0
11   PRINT "Es sind keine Festplatten vorhanden"

12  CASE 1
13   CALL park (1)
14   PRINT "Festplatte 1 geparkt"

15  CASE 2
16   CALL park (1)
17   PRINT "Festplatte 1 geparkt"
18   CALL park (2)
19   PRINT "Festplatte 2 geparkt"
20 END SELECT

21 PRINT "Schalten Sie nun Ihren Rechner aus."
22 PRINT "Falls Sie ein Taste betätigen, ist"
23 PRINT "das Parken der Festplatte(n) aufgehoben."
24 PRINT
25 WHILE INKEY$ = ""                      '! Endlosschleife
26 WEND
27 PRINT "Ende PARK, Platte nicht geparkt"
28 END                                    '! Ende

   '############################################################
   '#                     Hilfsroutinen                        #
   '############################################################

29 fehler:
   '------------------------------------------------------------
   '! Fehlerbehandlung in PARK
   '------------------------------------------------------------
```

```
30  PRINT "Fehler : ";ERR;" unbekannt"
31  PRINT "Programmabbruch"
32 END                                  '! MSDOS Exit

33 SUB drivesx
   '!-------------------------------------------------------
   '! ermittle die Zahl der Plattendrives per INT 13
   '! CALL:   AH = 800H DX = 0080H
   '! RETURN: DL = drives
   '!-------------------------------------------------------

34  SHARED drive%

35  REG 1, &H0800                       '! AX = 0800
36  REG 4, &H80                         '! Drives
37  CALL INTERRUPT &H13                 '! BIOS INT
38  drive% = REG (4) AND &HFF           '! lese Wert in DL
39 END SUB

40 SUB park (lw%)
   '!-------------------------------------------------------
   '! positioniere Kopf auf letzte Spur
   '! CALL:   AH = C00H DX = 0080H + drive_nr
   '!-------------------------------------------------------

41  REG 1, &H0800                       '! AX = 0800
42  REG 4, &H80 + lw%                   '! Drive
43  CALL INTERRUPT &H13                 '! ermittle Drivedaten
44  '! Kopf des Laufwerkes auf letzte Spur positionieren

45  REG 1,H0C00                          '! Seek
46  REG 3,REG (3)                        '! Inhalt von CX
47  REG 4, &H80 + lw%                    '! Laufwerk
48  CALL INTERRUPT &H13
49 END SUB

   '***** Programm Ende ******
```

Listing 6.8: PARK.BAS

Bildschirmsteuerung über den INT 10H

Ein weiterer interessanter Interrupt ist der BIOS-INT 10. Über diesen Interrupt erfolgt die Bildschirmausgabe. Die Funktionen werden daher direkt vom BIOS der Grafikkarte zur Verfügung gestellt. Nachfolgend möchte ich eine kleine Bibliothek vorstellen, die einige der BIOS-INT 10-Funktionen direkt benutzt.

Der Entwurf

Der BIOS-INT 10 wird über den Befehl INTERRUPT aus PowerBASIC angesprochen. Die erforderlichen Parameter müssen in den Registern AX bis DX übergeben werden. Der Inhalt von AH steuert dabei die auszuführende Funktion. Da diese Art des Aufrufes recht unkomfortabel ist, sollen die einzelnen Funktionen über Prozeduren aufrufbar sein. Dann kann ein Anwendungsprogramm leichter auf die BIOS-Funktionen zugreifen. Auf eine Beschreibung der INT 10-Funktionen wird an dieser Stelle verzichtet. Der interessierte Leser sei auf /1/ verwiesen, wo sich eine detaillierte Beschreibung aller Aufrufe findet.

Die Implementierung

Die einzelnen Routinen bilden bestimmte BIOS-INT-10-Funktionen für PowerBASIC ab. Diese Routinen befinden sich in der Datei *INT10.INC* und werden nachfolgend beschrieben.

SetMode (mode%)

Diese Routine schaltet die Adapterkarte des Bildschirms in einen bestimmten Modus. Dieser Modus ist im Parameter *mode%* zu übergeben. Hierbei gilt:

```
mode% = 0  40 x 25 Zeichen Monochrom
        1  40 x 25 Zeichen Farbe
        2  80 x 25 Zeichen Monochrom
        3  80 x 25 Zeichen Farbe
        4  320 x 200 Pixel Farbe
        5  320 x 200 Pixel Monochrom
        6  640 x 200 Pixel Monochrom
```

Damit läßt sich der Bildschirm in den gewünschten Modus umschalten. Mit EGA- und VGA-Karten stehen weitere Modi zur Verfügung, die hier aber nicht behandelt werden.

GetMode (mode%)

Diese Routine ermittelt über das BIOS den aktuellen Modus der Adapterkarte des Bildschirms. Hierbei gelten die bei *SetMode* definierten Modi als Übergabeparameter.

CursorSize (ymin%, ymax%)

Der Cursor wird aus mehreren Zeilen aufgebaut. Dabei kann der Cursor maximal sieben Zeilen (Monochrom) und 13 Zeilen (Farbe) umfassen. Die Prozedur *Cursorsize* erlaubt es, die Größe des Cursors zu verändern. Als Parameter sind die unterste und die oberste Zeile zu übergeben.

SetCursor (xpos%, ypos%)

Mit der Prozedur läßt sich der Cursor innerhalb des Bildschirms positionieren. Die beiden Parameter geben dabei die Position in Spalten (x) und Zeilen (y) an. Im Textmode sind 80 Spalten (Zeichen) erlaubt. Um

Zeichen per BIOS auszugeben, muß vorher die betreffende Cursorposition gesetzt werden.

GetCursor (xpos%, ypos%)

Mit der Prozedur läßt sich die Cursorposition abfragen. Es gelten die gleichen Übergabeparameter wie bei *SetCursor*.

Scroll (x1%, y1%, x2%, y2%, lines%, attribut%)

Dies ist die interessanteste Prozedur der kompletten Bibliothek. Mit den Parametern x1,y1 und x2,y2 läßt sich ein Fenster auf dem (Text-) Bildschirm definieren. Dann kann der Text innerhalb des Bildschirms gescrollt werden. Die Zahl der zu scrollenden Zeilen wird in *lines%* definiert. Ist der Wert negativ, wird der Fensterinhalt um n Zeilen nach oben verschoben. Andernfalls wird der Text n Zeilen nach unten verschoben. Die neu eingefügte Zeile ist mit Leerzeichen gefüllt. Der Parameter *attribut%* definiert dabei, wie die neue Zeile darzustellen ist. Es gilt dabei die Codierung für die Attribute von Monochrom- und Farbkarten.

Weitere Einzelheiten zu den BIOS-Aufrufen finden sich in /1/. Die Prozeduren sind in dem folgenden Listing ausgiebig kommentiert.

```
X R E F   /Z=50                              (c) Born Version 1.0
Datei : int10.inc      Datum : 07-19-1992         Seite :  1

 Zeile     Anweisung

      '!#####################################################
      '! File:    INT10.INC
      '! Version: 1.0 v. 16.7.92 (c) G. Born
      '!          Subroutinen zur Bildschirmsteuerung
      '!#####################################################

    1 SUB SetMode (mode%)
      '!-----------------------------------------------------
      '!  setze den Bildschirmmode
      '!   mode% = 0  40 x 25 Zeichen Monochrom
      '!           1  40 x 25 Zeichen Color
      '!           2  80 x 25 Zeichen Monochrom
      '!           3  80 x 25 Zeichen Color
      '!           4  320 x 200 Pixel Color
      '!           5  320 x 200 Pixel Monochrom
      '!           6  640 x 200 Pixel Monochrom
      '!-----------------------------------------------------
    2  REG 1, 0 + (mode% AND &H0F)
    3  CALL INTERRUPT &H10
    4 END SUB

    5 SUB GetMode (mode%)
      '!-----------------------------------------------------
      '!  lese den Bildschirmmode
      '!   mode% = 0  40 x 25 Zeichen Monochrom
      '!           1  40 x 25 Zeichen Color
```

```
   '!           2  80 x 25 Zeichen Monochrom
   '!           3  80 x 25 Zeichen Color
   '!           4  320 x 200 Pixel Color
   '!           5  320 x 200 Pixel Monochrom
   '!           6  640 x 200 Pixel Monochrom
   '!-------------------------------------------------
 6  REG 1, &H0F00
 7  CALL INTERRUPT &H10
 8  mode% = REG (1) AND &H0F
 9 END SUB

10 SUB CursorSize (ymin%,ymax%)
   '!-------------------------------------------------
   '!  stelle die Cursorgröße ein
   '!-------------------------------------------------
11  REG 1,&H0100
12  REG 3, (ymax% AND &H0F)*16 + (ymin% AND &H0F)
13  CALL INTERRUPT &H10
14 END SUB

15 SUB SetCursor (xpos%, ypos%)
   '!-------------------------------------------------
   '!  setze den Cursor auf Bildschirmseite 0
   '!-------------------------------------------------
16  REG 1,&H0200
17  REG 2,0
18  REG 4, ((ypos%+1) AND &HFF) * 256 + ((xpos%+1) AND &HFF)
19  CALL INTERRUPT &H10
20 END SUB

21 SUB GetCursor (xpos%, ypos%)
   '!-------------------------------------------------
   '!  lese die Cursorposition auf Bildschirmseite 0
   '!-------------------------------------------------
22  REG 1,&H0300
23  REG 2,0
24  CALL INTERRUPT &H10
25  xpos% = (REG (4) AND &HFF) + 1
26  ypos% = ((REG (4) AND &HFF) / 256) + 1
27 END SUB

28 SUB Scroll (x1%, y1%, x2%, y2%, lines%, attribut%)
   '!-------------------------------------------------
   '!  up-/downscroll des Fensters um n lines
   '!  x%, y%  = Koordinaten Fenster
   '!  lines%  = Zahl der zu scrollenden Zeilen
   '!            negativ -> upscroll, sonst downscroll
   '!  attribut%= Attribut der neuen Zeile
   '!              Bit 7 = Blinkbit
   '!                  3 = Intensitätsbit
   '!                4-6 = Hintergrundfarbe
   '!                0-2 = Vordergrundfarbe
   '!-------------------------------------------------

29  IF lines% < 0 THEN                   '! scroll up
```

```
   30   REG 1,&H0600 + (-lines% AND &HFF)    '! Zahl der Zeilen
   31  ELSE                                  '! scroll down
   32   REG 1,&H0700 + (lines% AND &HFF)     '! Zahl der Zeilen
   33  END IF

   34  REG 3, (y1%-1 AND &HFF)*256  + (x1%-1 AND &HFF) '! Window
   35  REG 4, (y2%-1 AND &HFF)*256  + (x2%-1 AND &HFF) '! Window
   36  REG 2, (attribut% AND &HFF)

   37  CALL INTERRUPT &H10
   38 END SUB

      '! ENDE
```

Listing 6.9: Bibliothek zur Bildschirmsteuerung (INT10.INC)

Ein Beispielprogramm zu Bildschirmansteuerung

Das nachfolgende kleine Beispielprogramm erläutert den Umgang mit obigen BIOS-Funktionen. Das Programm benutzt die Pop-up-Menüs zur Steuerung des Ablaufes.

In den ersten beiden Menüpunkten läßt sich die Größe des Cursors verändern. Der folgende Menüpunkt demonstriert, wie sich Bildschirmausschnitte scrollen lassen.

Textbox

Um Text auf dem Bildschirm auszugeben, wird eine eigenen Prozedur mit dem Namen *TextBox* eingeführt. Diese Prozedur wurde aus dem Modul *PopMenu* der Bibliothek *MENU.INC* extrahiert. *PopMenu* erfüllt bereits alle Funktionen zur Textausgabe in einer Box. Lediglich die Cursorsteuerung zur Auswahl eines Menüpunktes und damit auch der Parameter *nr%* kann entfallen. Die Prozedur *TextBox* besitzt daher auch die gleichen Parameter wie *PopMenu*. Bei Bedarf sollten Sie den Code mit in *MENU.INC* integrieren.

Soll eine Textbox wieder gelöscht werden, muß vorher der Bildschirminhalt mit *OpenBox* gesichert werden. Die Prozedur *TextBox* erlaubt es, Tastaturabfragen und Benutzereingaben direkt aus dem Anwendungsprogramm zu steuern. Damit lassen sich recht komfortabel Eingabemasken gestalten. Einziges Handikap ist die Positionierung des Eingabecursors. Hier könnte das Modul *Textbox* um eine weitere Funktion zur Gestaltung von Benutzereingaben erweitert werden. Weitere Einzelheiten sind folgendem Listing zu entnehmen.

```
X R E F   /Z=50                          (c) Born Version 1.0
Datei : screen.bas      Datum : 07-19-1992         Seite :  1

 Zeile     Anweisung

      '*******************************************************
      '! File       : SCREEN.BAS
```

```
      '! Vers.      : 1.0
      '! Last Edit  : 11.7.92
      '! Autor      : G. Born
      '! Progr. Spr.: PowerBasic
      '! Betr. Sys. : DOS 2.1 - 5.0 (DR-DOS 5.0/6.0)
      '! Funktion: Das Programm wird mit der Eingabe:
      '!
      '!           SCREEN
      '!
      '!           aufgerufen. Es demonstriert die Verwendung
      '!           der Routinen zur Bildschirmsteuerung
(INT10.INC).
      '************************************************************
      '!
    1 DIM MEN$(7)                              '! Menütexte
    2 nr% = 1                                  '! Zeilennr. Cursor
    3 DIM buff1%(600)                          '! temporäre Puffer
    4 DIM buff2%(500)                          '! max. 2000 Elemente
    5 black% = 0                               '! Farben
    6 white% = 7

    7 CALL SetMode (3)                       '! 80 x 25 Zeichen

      '! Bildschirm löschen und mit Zeichen füllen

    8 CLS
    9 FOR i% = 1 TO 1999: PRINT "°"; : Next i% '! Screen füllen

      '!-------------------------------------------------------
      '! Init Variable des Menüsystems, der Bildschirmadapter
      '! liegt bei Coloradaptern bei Segmentadr. B800H,
      '!-------------------------------------------------------

   10 CALL MenuInit (&HB800)             '! Init Variable

   11 done% = 0                          '! Hilfsflag löschen

   12 DO WHILE 1                         '! Schleife über
Menüsystem

   13  Kopf$ = ("INT10 Demo")            '! Titeltext für Menübox
   14  MEN$(1) = "Cursor Size 1"         '! Texte für Menü
definieren
   15  MEN$(2) = "Cursor Size 2"
   16  MEN$(3) = "Scroll Demo"
   17  MEN$(4) = "Exit"

      '!-------------------------------------------------------
      '! Aufruf des Hauptmenüs mit Kopftext ohne Fußtext, 7 entries
      '! als erstes muß der Fensterbereich gesichert werden
      '! Achtung: dies darf nur 1 x erfolgen, deshalb Flag done
      '!-------------------------------------------------------

   18  IF done% = 0 THEN                 '! Box offen?
   19   CALL OpenBox(8,10,MEN$(),4,kopf$,"",buff1%())
```

```
   20   done% = 1                          '! markiere offene Box
   21  END IF

   22  status% = 0
   23  CALL
PopMenu(8,10,MEN$(),4,white%,black%,kopf$,"",1,status%,nr%)
   24  tmp% = nr%                          '! merke Selektion
   25  IF status% < 0 THEN                '! Fehler beim Aufruf?
   26   CLS
   27   PRINT "Fehler: Menübox paßt nicht auf Bildschirm"
   28   END
   29  ELSE
   30   IF status% = 1 THEN               '! ESC gedrückt?
   31    CALL CloseBox(buff1%())          '! Schließe Box
   32    END                              '! Ja -> Exit
   33   END IF
   34  END IF

      '! werte selektierten Menüpunkt in nr% aus
      '! status% = 2 oder 3 wird hier ignoriert und
      '! wirkt daher wie RETURN !!!!

   35  IF (nr% = 1) OR (nr% = 2) THEN

      '!-------------------------------------------------
      '! baue ein Textfenster mit dem Cursor auf
      '!-------------------------------------------------

   36    MEN$(1) = "Cursor :     "      '! Texte für Menü
definieren

   37    CALL OpenBox(25,10,MEN$(),1,"","",buff2%())
   38    CALL TextBox(25,10,MEN$(),1,white%,black%,"","Exit ->
ESC",1,status%)
   39    IF status% < 0 THEN               '! Fehler beim Aufruf?
   40     CLS
   41     PRINT "Fehler: Textbox paßt nicht auf Bildschirm"
   42     END
   43    END IF

   44    CALL SetCursor (32,9)
   45    IF nr% = 1 THEN
   46     CALL CursorSize (1,2)             '! Cursorgröße 1
   47    ELSE
   48     CALL CursorSize (1,7)             '! Cursorgröße 2
   49    END IF

   50    DO WHILE INKEY$ = "" : WEND       '! warte auf Input

   51    CALL CloseBox(buff2%())                  '! close Submenü

   52  ELSEIF nr% = 3 THEN

   53    head$ = "Scroll-Demo"             '! Titeltext für Menübox
   54    fuss$ = "Exit->ESC"               '! Fußtext für Menübox
```

```
   55    MEN$(1) = " "              '! Texte für Menü definieren
   56    MEN$(2) = "Scroll Demo"
   57    MEN$(3) = "Scroll Demo"
   58    MEN$(4) = "Scroll Demo"
   59    MEN$(5) = "Scroll Demo"
   60    MEN$(6) = " "

      '! Aufruf des Menüs
   61    CALL OpenBox(25,10,MEN$(),6,head$,fuss$,buff2%())
   62    CALL
TextBox(25,10,MEN$(),6,white%,black%,head$,fuss$,1,status%)
   63    IF status% < 0 THEN               '! Fehler beim Aufruf?
   64     CLS
   65     PRINT "Fehler: Menübox paßt nicht auf Bildschirm"
   66     END
   67    END IF

   68    flag% = 0
   69    DO WHILE INKEY$ <> CHR$(27)
   70     IF flag% = 0 THEN
   71      CALL Scroll (26, 11, 36, 15, -1, 7)
   72      flag% = 1
   73     ELSE
   74      CALL Scroll (26, 11, 36, 15, 1, 7)
   75      flag% = 0
   76     END IF
   77     DELAY 1
   78    WEND

   79    CALL CloseBox(buff2%())

   80   nr% = tmp%
   81  ELSE
   82   CALL CloseBox(buff1%())
   83   END
   84  END IF
   85 WEND

   86  CALL CloseBox(buff1%())

   87 END
      '! Routine Textbox

   88 SUB TextBox (x%, y%, text$(1), items%, fcol%, bcol%, title$,
foot$,
       style%, status%)
      '!-----------------------------------------------------
      '!  Subroutine für Textausgabe
      '!
      '!  Die Routine gibt die Textbox mit dem Text aus.
      '!
      '! x%, y%   Anfangskoordinaten linke obere Ecke
      '! text$()  Texte mit Menüpunkten
      '! items%   Zahl der Menüpunkte
      '! title$   Text Kopfzeile
```

```
    '! foot$    Text Fußzeile
    '! style%   Rahmentyp (1 = einfach, 2 = doppelt, sonst blank
    '! fcol%    Vordergrundfarbe
    '! bcol%    Hintergrundfarbe
    '! status%  Ergebnis des Aufrufes:
    '!          -2 Fehler: Initialisierung fehlt
    '!          -1 Fehler: Box paßt nicht auf Bildschirm
    '!           0 ok
    '!-------------------------------------------------

 89 LOCAL maxlen%, i%, flag%
 90 LOCAL lo$, lu$, ro$, ru$, li$, lup$
 91 SHARED xmax%, ymax%, initflg%

    '! prüfe ob INIT durchgeführt

 92  IF initflg% <> 1 THEN
 93   status% = -2
 94   EXIT SUB
 95  END IF

    '! Emittle Länge des Menüpunktes

 96  CALL GetMaxLen (text$(),items%, title$, foot$, maxlen%)

    '! Paßt das Menü auf den Bildschirm ?

 97  IF (x% + maxlen% + 2) > xmax% THEN
 98   status% = -1
 99   EXIT SUB
100  END IF

101  IF (y% + items% + 2) > ymax% THEN
102   status% = -1
103   EXIT SUB
104  END IF

    '! Rahmentyp setzen

105  CALL MenuLine(lo$,lu$,ro$,ru$,li$,lup$,style%)

    '! Rahmen zeichnen

106  COLOR fcol%,bcol%
107  LOCATE y%, x%                    '! linke obere Ecke
108  PRINT lo$;
109  IF (LEN(title$) > 0) THEN
110   PRINT title$;                   '! Titel Textbox
111  END IF
112  IF LEN(title$) < maxlen% THEN
113   FOR i% = LEN(title$) TO maxlen%-1 : PRINT li$; : NEXT i%
114  END IF
115  PRINT ro$

116  FOR i% = 1 TO items%
```

```
117   LOCATE (y%+i%), x%
118   PRINT lup$;
119   CALL PutLine (text$(i%),maxlen%)
120   PRINT lup$
121  NEXT i%

122  LOCATE (y%+items%+1), x%
123  PRINT lu$;
124  IF (LEN(foot$) > 0) THEN
125   PRINT foot$;                          '! Fußtext
126  END IF
127  IF LEN(foot$) < maxlen% THEN
128   FOR i% = LEN(foot$) TO maxlen%-1 : PRINT li$; : NEXT i%
129  END IF
130  PRINT ru$

131  status% = 0

132 END SUB

    '! Libs einbinden

133 $INCLUDE "MENU.BAS
134 $INCLUDE "INT10.INC"

    '! Ende
```

Listing 6.10: Beispiel zur Bildschirmsteuerung

DBVIEW: Zugriff auf dBase (DBF)-Dateien mit PowerBASIC

Im Bereich der Datenbanken nimmt dBase auf MS-DOS Rechnern eine dominierende Stellung ein. Für Datenbankanwendungen bietet die interne Programmiersprache eine gute Unterstützung. Aufwendig bis unmöglich sind umfangreichere Berechnungen oder die Ausgabe von Grafiken. Zwar gibt es mittlerweile Zusatzprogramme, aber diese sind auch nicht immer hilfreich.

Anders sieht es bei PowerBASIC aus, wo gerade in Richtung Berechnungen und Grafiken alle Möglichkeiten offen stehen. Dafür ist das Thema »Datenbankfunktionen« ein Schwachpunkt. Im folgenden Abschnitt wird nun gezeigt, wie dies für PowerBASIC funktioniert (auch wenn die Implementierung wegen der fehlenden Möglichkeit zur Definition eigener Datentypen unerwartet aufwendig wurde). Die dabei vorgestellten Routinen können in eigenen Programmen benutzt werden.

Der Aufbau der DBF-Dateien in dBase III

Bevor aber über eine Lösung nachgedacht wird, sollte das Format dieser Dateien bekannt sein. Deshalb wird nachfolgend die Struktur der DBF-Dateien etwas tiefergehend diskutiert.

dBase speichert in diesen Dateien alle Informationen, die es zur Bearbeitung und Auswertung braucht. Der Vorteil, daß alle Fakten in kompakter Form vorliegen, wird allerdings mit dem Nachteil eines aufwendigeren Zugriffsmechanismus auf die Daten erkauft. Die Entwickler sind bei der Definition einige Kompromisse eingegangen, die sich auf das Laufzeitverhalten erheblich auswirken. Neben Tricks zur Optimierung der Zugriffgeschwindigkeit sind auch Schwächen in der Abspeicherung der Einzeldaten erkennbar. Es ist daher aus meiner Sicht interessant, sich intensiver mit der internen Struktur der DBF-Dateien auseinanderzusetzen. Dies trägt nicht nur zur Lösungsfindung bei, sondern verbessert sicher auch das Verständnis für den Umgang mit dBase.

Jede DBF-Datei setzt sich aus drei Teilen zusammen: dem Header, der Feldbeschreibung (Datensatzdefinition) und den eigentlichen Datensätzen. Die Informationen sind dabei gemischt im ASCII- und Binärformat gespeichert. Der Header enthält alle Informationen über den Aufbau der Datei, die Struktur ist in Tabelle 6.4 aufgeführt.

Offset	Bytes	Bedeutung
00H	1	Nummer der dBase-Version 02H dBase II 03H dBase III 83H dBase III mit Memofeld
01H	3	Datum letzter Schreibzugriff im Binärformat (JJMMTT)
04H	4	Zahl der Datensätze im File
08H	2	Headerlänge in Byte
0AH	2	Datensatzlänge in Byte
0CH	20	reserviert
20H	n*32	Feldbeschreibungen
..H	1	0DH als Header Ende

Tabelle 6.4: Format eines dBase-III-DBF-Headers

Das erste Byte des Headers besitzt bereits eine Doppelfunktion. Zum einen dient es dBase zur Identifizierung, ob es sich um eine gültige DBF-Datei

handelt und welche Version vorliegt. Davon ist dann auch die nachfolgende Datenstruktur abhängig. Die alten dBase-II-Datenbanken besitzen den Wert 02H im ersten Byte. Auf deren Struktur soll nicht weiter eingegangen werden. Ab dBase III findet sich im unteren Nibble (Bit 0 .. 3) der Wert 3H. Das oberste Bit (7) dient zur Markierung, ob die Datei Memofelder enthält. In diesem Fall ist der DBF-Datei ein DBT-File mit den Texten zugeordnet, und das Byte enthält demnach den Code 83H. In allen anderen Fällen findet sich der Wert 03H.

Das nächste Feld umfaßt drei Byte mit dem im Binärformat codierten Datum des letzten Schreibzugriffes in der Form JJMMTT. Dies bedeutet, im ersten Byte steht das Jahr (0 .. 99). Warum die Entwickler dieses Feld vorgesehen haben, ist mir schleierhaft, da DOS bereits im Directory zu jedem Dateinamen auch das Datum und vor allem die Zeit des letzten Schreibzugriffes führt. Beim Kopieren kompletter Dateien wird dagegen das Datum der Quelldatei mit übernommen. Das folgende Feld umfaßt vier Byte, in denen die Zahl der Datensätze in der DBF-Datei geführt wird. Die Bytes werden als vorzeichenlose 32-Bit-Zahl interpretiert, wobei die üblichen Intel-Konventionen zur Speicherbelegung (Low Byte der Zahl auf der untersten Adresse) gelten. Auch hier lohnt sich eine Beschäftigung mit den Interna. Was heißt denn eigentlich »Zahl der Datensätze« dBase hängt bei jedem APPEND BLANK-Befehl einen neuen (leeren) Datensatz mit den definierten Feldern an die DBF-Datei an. Nun gibt es noch den DELETE-Befehl, der einen Satz aus der Datei entfernt. Aus Effizienzgründen beschränkt sich dBase aber darauf, einen gelöschten Satz mit einer Markierung »*« im ersten Byte zu kennzeichnen. Dies führt zu schnellen DELTE-Operationen und ermöglichst sogar ein Undelete ohne größeren Aufwand. Aber die Sätze verbleiben nach wie vor in der Datenbanktabelle. So kann eine solche Tabelle viele hundert gelöschte Datensätze aufweisen, die als Daten noch vorhanden sind. Zugriffe ohne Index werden dadurch recht langsam, da für einen Zugriff auf den Folgesatz alle dazwischen liegenden Sätze (gelöscht oder ungelöscht) zu lesen sind. Um diesen Nachteil zu korrigieren, bietet dBase den PACK-Befehl, mit dem die gelöschten Sätze aus der DBF-Datei entfernt werden. An die Stelle des gelöschten Records wird ein nachfolgender gültiger Satz kopiert. Der Eintrag im Kopf enthält immer die Gesamtzahl der Records in der Datenbank, unabhängig von ihrer Gültigkeit. Erst mit einer PACK-Anweisung reduziert sich der Eintrag im Header. An dieser Stelle möchte ich noch einen weiteren Hinweis geben. Nach einer PACK-Operation muß sich nicht unbedingt die Größe der DBF-Datei geändert haben. DOS verwaltet Dateien über sogenannte Cluster, die je nach Medium eine verschiedene Größe besitzen. Ein Cluster ist dann die kleinste belegte Einheit auf einem Medium. Die Implementierung von dBase III bildet nun die interne Datenbankstruktur auf DOS-Dateien mit Clustern ab. Auch wenn nun Datensätze mit PACK gelöscht werden, verändert dBase die Größe der Datei nicht, sondern beschränkt sich auf das Verschieben der gültigen Sätze auf gelöschte Positionen. Das Ende des gültigen Datenbereiches wird anschließend durch das EOF-Zeichen 1AH markiert. Ein Lesezugriff über die DOS-Funktionen mit einer EOF-Abfrage

funktioniert nicht, denn im Binärmodus stimmt das DOS-Dateiende nicht mit dem logischen dBase-Dateiende überein. Vielmehr stehen die alten (gelöschten oder verschobenen) Sätze nach wie vor in der Datei und können auch gelesen werden. Wer einen Zugriff im Textmodus versucht - dieser erkennt 1AH als EOF - wird ebenfalls scheitern, da eine DBF-Datei ja ASCII- und Binärdaten enthält. Bereits im Datumsfeld kann der Eintrag 1AH vorkommen. Das logische Dateiende läßt sich damit nur über die Recordzahl im Header identifizieren. Sobald dieser Zeiger zerstört ist, besitzt dBase keine Möglichkeit zur Restaurierung der Datenbank mehr. Erst wenn die Datenbank mit dem dBase-Befehl COPY FILE TO in eine andere Datei kopiert wird, verkleinert sich die Dateilänge, da dann nur die gültigen Sätze übertragen werden.

Dies bringt natürlich einige Nachteile mit sich, angefangen von dem aufwendigeren Zugriff bis hin zu dem unnötig belegten Speicherplatz auf der Diskette/Platte. Ein Vorteil soll aber nicht unerwähnt bleiben: Dadurch, daß sich die Dateigröße nicht immer bei PACK-Operationen ändert, wird einer Fragmentierung der Platte vorgebeugt. Dieser Effekt tritt unter DOS immer dann auf, wenn Dateien ständig in ihrer Größe variiert werden und mit der Zeit über verschiedene - nicht zusammenhängende - Cluster verstreut sind. Dies verschlechtert natürlich (wegen der vielen Kopfbewegungen) die Zugriffszeiten auf die Dateien.

Das nächste Feld im Header umfaßt eine vorzeichenlose 16-Bit-Zahl, in der die Länge des Headers in Byte steht.

Die Länge eines Datensatzes wird im nächsten Feld als vorzeichenlose 16-Bit-Zahl geführt. Dieser Wert ist immer um ein Byte höher als die rechnerische Summe der einzelnen Feldlängen. Dies ist darin begründet, daß am Satzanfang ein Byte zur Markierung gelöschter Sätze reserviert wird.

Nun kommt ein reservierter Bereich mit 20 Byte, der nicht weiter interessiert. Damit sind im Prinzip die in Tabelle 6.4 beschriebenen Headerinformationen abgehandelt.

Was noch fehlt sind die Informationen über den Aufbau der Datensätze. Diese finden sich in Form einzelner Feldbeschreibungen, die im Anschluß an den Header gespeichert sind. Für jedes Feld der Datenbank findet sich ein Satz mit 32 Byte, der das in Tabelle 6.5 gezeigte Format besitzt.

Offset	Bytes	Bedeutung
00H	11	Name des Feldes in ASCII-Zeichen mit 00H abgeschl.
0BH	1	Feldtyp in ASCII (C, N, L, D, M)
0CH	4	Feldadresse im Speicher
10H	1	Feldlänge in Byte

11H	1	Nachkommastellen in Byte
12H	2	reserviert
14H	1	ID für Arbeitsbereich
15H	2	reserviert
17H	1	SET FIELDS Marke
18H	8	reserviert

Tabelle 6.5: Format der dBase-III-DBF-Feldbeschreibung

Die ersten elf Byte der Feldbeschreibung sind für den Namen des Feldes reserviert, der als ASCII-Text abgelegt wird. Das Zeichen 00H schließt eine Zeichenkette ab. Umfasst der Feldname keine elf Zeichen, werden die restlichen Bytes deshalb mit den Werten 00H belegt.

Im nächsten Byte steht das ASCII-Zeichen für den Feldtyp. Dieser wird gemäß der in Tabelle 6.6 gezeigten Notation codiert.

Zeichen	Typ	erlaubte Zeichen
C	Char.	ASCII-Zeichen
N	Num.	0..9, -
L	Logic.	JjNnTtFf?
D	Datum	JJJJMMTT
M	Memo	DBT Blocknummer

Tabelle 6.6: Codierung der dBase-III-Feldtypen

An den Feldtyp schließt sich ein 4-Byte-Vektor an, der die Felddatenadresse enthält. Diese Adresse wird nur für die Bearbeitung im Hauptspeicher benötigt und besitzt auf dem Speichermedium keine Bedeutung.

Die Feldlänge findet sich im Header ab Offset 16 und ist in einem Byte als Binärcode abgelegt. Damit kann ein Feld maximal 256 Zeichen umfassen. Bei numerischen Feldern gibt der Wert die Zahl der Stellen (einschließlich des Dezimalpunktes) an. Bei Memo-Feldern beträgt die Feldlänge genau zehn Byte, in denen eine Blocknummer für die zugehörige DBT-Datei gespeichert wird.

»Logical«-Felder besitzen die Länge 1, während bei Datums-Feldern immer acht Byte reserviert werden.»Bei numerischen Feldern spezifiziert das Folgebyte die Zahl der Nachkommastellen. Bei allen anderen Feldtypen

besitzt der Eintrag den Wert 00H. Wichtig ist, daß die Zahl der Nachkommastellen immer kleiner als die Feldlänge ist.

Die restlichen 14 Byte sind für interne Zwecke reserviert und interessieren für den Zugriff auf die Datenbanken nicht weiter. Sie müssen lediglich beim Zugriff auf die Feldbeschreibung gelesen werden. Auch das SET FIELDS-Byte ist nicht weiter relevant, da dBase diesen Eintrag offensichtlich nur im Speicher benutzt. Einzelheiten sind der Tabelle 6.3 zu entnehmen.

Für jedes Feld der Datenbank findet sich ein eigener 32-Byte-Satz mit obigen Informationen. Das Ende der Felddefinitionen wird durch das Zeichen 0DH markiert. Auch hier merkt man den ASCII-orientierten Aufbau der DBF-Dateien. Damit ist der Header komplett beschrieben und alle Informationen über den Aufbau der Datensätze liegen vor.

Die eigentlichen Datensätze werden von dBase an den Definitionsteil angehängt. Die Recordlänge richtet sich nach der Länge der jeweiligen Felder und wird im Header der Datei geführt. Der Wert ist immer um 1 Byte größer als die Summe der Feldlängen, da vor jedem Record ein Eintrag für Markierungszwecke reserviert wird. Die Datensätze werden im reinen ASCII-Format ohne Trennzeichen gespeichert. Damit ist der Import und Export von Daten mit der dBase-SDF-Option natürlich recht einfach. Weiterhin lassen sich alle Felder, unabhängig von ihrem Typ, als ASCII-Text bearbeiten. Allerdings wird dies mit einem erheblichen Aufwand für die Bearbeitung und Speicherung numerischer Werte erkauft. Bei Berechnungen ist vor jeder Schreib-/Leseoperation eine Konvertierung erforderlich. Das erste Byte im Satz dient zur Markierung gelöschter Daten. Bei neuen Sätzen wird hier ein Leerzeichen eingetragen. Die »Delete«-Funktion schreibt nur ein »*« in dieses Byte. Damit ist der Satz als gelöscht markiert und kann durch den PACK-Befehl entfernt werden. Wird ein neuer Satz mit APPEND BLANK angehängt, fügt dBASE eine entsprechende Anzahl an Leerzeichen an das Dateiende ein. Der Abschluß des gültigen Datenbereiches wird durch das Zeichen 1AH markiert, d.h. die »EOF-Marke« wird nicht durch DOS, sondern durch dBASE verwaltet. Die EOF-Abfrage aus DOS funktioniert aus diesem Grunde nicht bei DBD-Dateien. Hinter dem Zeichen 1AH können durch korrekt aufgebaute Sätze auftauchen, die aber nicht mehr zum gültigen Bereich der Datenbank gehören. Hier wäre es aus meiner Sicht besser gewesen, wenn die Entwickler die DOS-I/O-Funktionen zur Veränderung der Dateigröße benutzt hätten.

Der Entwurf

Nach dieser etwas ausführlicheren Diskussion der DBF-Dateistruktur wenden wir uns der eigentlichen Aufgabe zu. Aus Programmen soll direkt auf die DBF-Datei zugegriffen werden. Da die Struktur bekannt ist, liegt die Lösung nahe. Zuerst ist der Header zu lesen, dann müssen die Feldbeschreibungen decodiert werden. Anschließend kann auf die Datensätze der Datei zugegriffen werden. Da alle Daten als ASCII-Zeichen

vorliegen, ist keine Konvertierung erforderlich. Günstig ist es jedoch, wenn diese Operationen nicht dediziert im Programm vorkommen, sondern in einzelnen Modulen versteckt werden. Dann lassen sich transparente und änderungsfreundliche Programme erstellen.

Benötigt werden folgende Funktionen:

- Lies einen Satz aus der DBF-Datei.
- Schreibe einen Satz in die DBF-Datei.
- Hänge einen Leersatz an.
- Positioniere den Schreib-/Lesezeiger.

Die Details der Implementierung werden im nachfolgenden Abschnitt besprochen.

Die Implementierung

Für die Bearbeitung der DBF-Dateien werden folgende Funktionen vorgesehen, die in der Datei *DBF_LIB.INC* gespeichert sind.

Die Datei *DB_DEF.INC* enthält alle globale Variablen, die von den Bibliotheksmodulen benötigt werden. Diese Definitionen sind nachfolgend aufgeführt.

```
'!************************************************
'! Include-Datei DB_DEF.INC für den DBF-Header mit
'! der Feldbeschreibung. Die Definition muß im Haupt-
'! programm eingebunden werden!
'! Die restlichen Definitionen finden sich in den
'! Funktionen zum Zugriff (Variable sind SHARED !!!!)
'!************************************************
'! Aufbau des Headers einer DBF-Datei
'! ver      STRING * 1          '! Version 03H oder 83H
'! datum    STRING * 3          '! Datum JJ MM TT
'! rec&     LONG                '! Records in Datenbank
'! headb    INTEGER             '! Zahl der Bytes im Kopf
'! recbyte  INTEGER             '! Zahl der Bytes pro Record
'! reserve  STRING * 20         '! reservierte Bytes

'! Aufbau der Feldbeschreibung der DBF-Datei
'! feldname  STRING * 11        '! Feldname 11 Zeichen
'! ftyp      STRING * 1         '! C N L D M
'! dummy1    STRING * 4         '! Dummy Feld
'! laenge    STRING * 1         '! Zahl der Stellen
'! komma     STRING * 1         '! Zahl der Nachkommastellen
'! dummy2    STRING * 2         '! reservierte Bytes
'! id        STRING * 1         '! ID Byte
'! dummy3    STRING * 11        '! reserviert
'-------------------------------------------------------

header$  = ""                   '! nimmt 32-Byte Header auf
ver%     = 0                    '! 1. Header Byte nach USE
```

```
rec&     = 0                    '! Zahl der DBF-Records
headb%   = 0                    '! Headerlänge in Byte
reclen% = 0                     '! Recordlänge in Byte
anzahl%  = 0                    '! Zahl der Felder in DBF

DIM feldname$(128)              '! Name DBF-Feld 10 Zchn
DIM ftyp$(128)                  '! Feldtyp DBF-Feld "C,N,T,.."
DIM laenge%(128)                '! Feldlänge
DIM komma%(128)                 '! Nachkommastellen
DIM feldinh$(128)               '! Buffer für Feldinhalt

recofs&  = 0                    '! Offset Anfang aktueller Satz

'! Hilfsvariable für MOVE
DIM a%(2)
tmp$ = " "
! Ende Definition
```

Die Datei muß im Hauptprogramm im Kopf mit INCLUDE eingebunden werden, damit die Variable auch der Anwendung zur Verfügung stehen. Die Bedeutung der einzelnen Variable ist den Kommentaren zu entnehmen.

Die verschiedenen Funktionen zum Zugriff auf die DBF-Dateien wurden nach Möglichkeit als PowerBASIC-Prozeduren definiert. Lediglich ein Modul zur Typkonvertierung mußte in Assembler implementiert werden. Nachfolgend werden die einzelnen Module vorgestellt.

MOVE (len%, ziel,quell)

Dies ist die einzige Prozedur, die in Assembler formuliert wurde. Aufgabe ist es, den Inhalt der Quelle in das Ziel zu verschieben. Die Zahl der Bytes wird dabei im Parameter *len%* angegeben. Als Quelle und Ziel müssen die Adressen der jeweiligen Variable angegeben werden. Bei Integervariablen wird die Adresse standardmäßig vom Compiler eingesetzt. Die Routine *MOVE* wird jedoch verwendet, um Strings in Integervariablen abzulegen. Bei Strings wird aber die Adresse auf einen Descriptor als Parameter übergeben. Um die Adresse des ersten Zeichen des Strings als Parameter zu übergeben ist folgende Sequenz erforderlich:

```
DIM a%(2)
a%(1) = STRPTR(txt$)
a%(2) = STRSEG(txt$)
CALL MOVE (2,reclen%, a%(1))
```

Obige Sequenz verschiebt zwei Byte aus dem String *txt$* in die Variabel *reclen%*. Dies wird im Modul *USE* bei der Konvertierung von Binärwerten des Headers benötigt. Der Header wird byteweise in einen String gelesen. Dann werden die einzelnen Binärwerte in Integervariable konvertiert.

Das Programm selbst wurde in Assembler codiert. Da kein Assembler zur Übersetzung zur Verfügung stand, habe ich das DOS-Programm DEBUG zur Assemblierung verwendet. Die Binärdatei wird dann mit $INLINE

"MOVE.COM" in Basic integriert. Einzelheiten bezüglich der Einbindung sind dem Listing der Bibliothek zu entnehmen.

Interessant ist jedoch noch die Parameterübergabe von Basic an Assembler. Basic übergibt die Adressen der jeweiligen Parameter auf dem Stack. Die Prozedur wird dabei als CALL FAR aufgerufen. Damit ergibt sich für obigen Aufruf von MOVE folgendes Stackabbild:

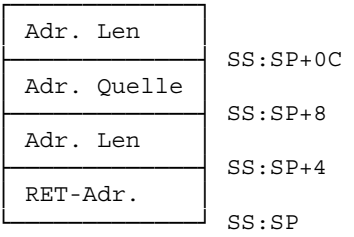


Bild 6.6: Übergabeparameter

Jeder Eintrag auf dem Stack besteht in diesem Fall aus vier Byte, wobei das Low-Word eines Parameters auf den unteren Stackadressen gespeichert ist. Dies muß beim Zugriff auf die Stackdaten berücksichtigt werden. Als letzter Parameter wird die Rücksprungadresse auf dem Stack gehalten. Um den Inhalt der Parameter zu lesen, muß die Adresse vom Stack als Zeiger benutzt werden. Die Einzelheiten sind nachfolgendem Listing zu entnehmen:

```
a 100
;--------------------------------------
; Routine zum Verschieben von n Bytes
; Aufruf: CALL MOVE (Len, Ziel, Quelle)
;         Len    = Zahl der Bytes
;         Ziel   = Adresse Ziel
;         Quelle = Adresse Quelladresse
;--------------------------------------
PUSH BP              ; BP
MOV BP,SP            ; BP auf Stack
ADD BP,02            ; alter SP
PUSH ES              ; merke ES
PUSH DS              ; merke DS
PUSHF
PUSH BX
PUSH AX
PUSH CX
PUSH SI
PUSH DI
; lese Adresse der Quelladresse vom Stack
MOV SI, [BP+04]      ; Ofs-PTR auf Quelle
MOV DS, [BP+06]      ; Seg-PTR auf Quelle
MOV AX, [SI]         ; ermittle Adresse String
MOV CX, [SI+2]       ;  "
MOV SI, AX           ;  "
MOV DS, CX           ;  "
; lese Len - Parameter und speichere in CX
```

```
MOV BX, [BP+0C]      ; Ofs. Len
MOV AX, [BP+0E]      ; Seg. Len
MOV ES, AX
ES: MOV CX,[BX]      ; LEN in CX
; lese Zieladresse von Stack
MOV DI, [BP+8]       ; Ofs. Ziel
MOV ES, [BP+0A]      ; Seg. Ziel
; verschiebe Speicherbereich
REP
MOVSB
POP DI
POP SI
POP CX
POP AX
POP BX
POPF
POP DS               ; restauriere DS
POP ES               ; restauriere ES
POP BP               ; restauriere BP
NOP                  ; Dummy zur Sicherheit

n MOVE.COM
r cx
3A
W
q
```

Listing 6.11: MOVE.ASM

Beispiele für die Verwendung finden sich im Modul USE.

Die nachfolgend beschriebenen Module wurden alle in Basic implementiert.

USE (handle%,filename$,status%)

Mit dieser Prozedur ist (in Anlehnung an die dBase-Notation) eine DBF-Datei zu öffnen. Der Dateiname muß dabei in *filename$* übergeben werden. Der Parameter *handle%* muß vor dem Aufruf mit einer Dateinummer belegt werden, die noch keiner offenen Datei zugewiesen wurde. Über diese Dateinummer erfolgen die Zugriffe auf die Daten, und mit der Nummer wird auch die Datei geschlossen. Das Modul prüft nach dem Aufruf, ob die Datei vorhanden ist. Im Fehlerfall wird das Programm abgebrochen. Deshalb ist im Anwendungsprogramm eine Fehlerroutine zu definieren. Ist die Datei vorhanden, wird der Header gelesen und ausgewertet. Einzelne Parameter werden in den globalen Variablen (siehe DB_DEF.INC) gespeichert und sind im Hauptprogramm zugänglich. Der Parameter *status%* enthält nach der Rückkehr einen Code mit folgender Bedeutung:

Status	Bedeutung
0	ok
1	---
2	keine gültige DBF-Datei
3	dBase-II-File (illegal)

```
4          EOF beim Lesen des Headers
5          kein EndOfHeader gefunden
```

Tabelle 6.7: Statuscodes von USE

Nur wenn *status%=0* zurückgegeben wird, konnte die DBF-Datei korrekt geöffnet werden. Andernfalls sollte das Programm beendet werden, da keine dBase-III-Datei vorliegt. Vorher ist die Datei mit der PowerBASIC-Anweisung CLOSE zu schließen. Die Dateinummer ist mit *handle%* identisch. Nach einem Aufruf von USE steht der Schreib-/Lesezeiger auf dem ersten Datensatz des Headers.

GetRecord (handle%,status%,buffer%)

Dieses Modul liest einen Datensatz aus der mit *handle%* spezifizierten Datenbank und überträgt die Informationen in den Textpuffer *buffer$*. Der Lesezeiger muß bereits auf den Anfang des Satzes gesetzt sein (*skip*). Ein weiterer Aufruf von *GetRecord* liest den gleichen Satz der Datenbank, da die Zeiger nur durch *Skip*, *GotoBottom*, *GotoTop* und *AppendBlank* verändert werden.

Ein unmittelbar darauffolgender *PutRecord*-Aufruf schreibt somit die Daten über den alten Datensatz. Die Prozedur separiert zusätzlich die einzelnen Feldinhalte und legt die Teilstrings in der globalen Feldvariablen *feldinh$()* ab. Der Parameter *status%* gibt nach dem Aufruf einen Hinweis auf Fehler (0 -> ok, 1 -> Fehler: EOF erreicht). Die einzelnen Feldinhalte lassen sich auch aus der Variablen *buffer$* extrahieren.

PutRecord (handle%,status%,buffer$)

Dieses Modul schreibt den Inhalt des Puffers in die mit *handle%* spezifizierte Datenbank zurück. Die Länge des Puffers (*buffer$*) muß mit der Länge der dBase-Datensätze übereinstimmen. Es wird der Datensatz überschrieben, der aktuell durch den Schreib-/Lesezeiger adressiert wird. Die Positionierung dieses Zeigers erfolgt durch *Skip*, *AppendBlank*, *GotoTop* und *GotoBottom*. In *status%* findet sich nach dem Aufruf der Fehlercode (0 -> ok, 1 -> Fehler: falsche Satzlänge).

AppendBlank (handle%)

Dieses Modul fügt einen neuen (leeren) Datensatz am Ende der Datenbank an. Daten lassen sich anschließend mit der Funktion *PutRecord* in diesen Satz übertragen. Gleichzeitig wird der Schreib-/Lesezeiger auf diesen neuen Datensatz gesetzt. Der Parameter *handle%* spezifiziert dabei die Nummer der mit USE geöffneten Datei.

Skip (handle%, status%, records%)

Da die *GetRecord*- und *PutRecord*-Aufrufe die relative Lage des internen Datenzeigers nicht verändern, wird das Modul *Skip* zur Positionierung benutzt. Jeder Aufruf verschiebt die Schreib-/Leseposition der durch *handle%* definierten Datei um *record%* Sätze. Die Richtung wird dabei durch das Vorzeichen von *record%* bestimmt. Negative Werte verschieben

den Zeiger zum Dateianfang. Wird der Dateianfang oder das Dateiende erreicht, enthält *status%* nach dem Aufruf den Wert 1. Sonst wird 0 zurückgegeben.

GotoBottom (handle%,status%)

Um die Datensätze zu adressieren, werden weitere Positionierungsfunktionen benötigt. Mit *GotoBottom* läßt sich der Schreib-/Lesezeiger auf den ersten Satz der mit *handle%* adressierten Datenbank positionieren. Bei diesem Satz kann es sich durchaus um einen als gelöscht markierten Record (»*« als erstes Byte) handeln. In *status%* wird immer der Wert 0 zurückgegeben.

GotoTop (handle%,status%)

Mit *GotoTop* läßt sich der Schreib-/Lesezeiger auf den letzten Satz der mit *handle%* adressierten Datenbank positionieren. Bei diesem Satz kann es sich durchaus um einen als gelöscht markierten Record (»*« als erstes Byte) handeln. In *status%* wird immer der Wert 0 zurückgegeben.

DBEof (handle%,status%)

Nun fehlt noch eine Funktion, die das Dateiende der DBF-Datenbank nach einer *Skip*-Anweisung erkennt. Die DOS-EOF-Funktion ist ja hierzu nicht in der Lage. Die Prozedur *DBEof* übernimmt diese Aufgabe.

Es ließen sich sicher noch einige Erweiterungen einbringen. Aber zur Demonstration und für den grundsätzlichen Umgang mit DBF-Dateien reichen die Module aus. Einzelheiten sind dem nachfolgenden Listing zu entnehmen.

```
X R E F   /Z=55                              (c) Born Version 1.0
Datei : dbf_lib.inc      Datum : 07-20-1992         Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : DBF_LIB.INC
      '! Vers.      : 1.0
      '! Last Edit  : 10.6.92
      '! Autor      : G. Born
      '! Progr. Spr.: PowerBasic 4.0 / 4.5
      '! Funktion: Library mit Routinen zum Zugriff auf dBase DBF-
Files
      '*********************************************************
      '!
    1 SUB MOVE INLINE
      '---------------------------------------------------------
      '! CALL MOVE (LEN, ZIEL, QUELLE)
      '! Die Prozedur verschiebt n Byte eines Strings in die
      '! Zieladresse. Achtung: der String muß mit seiner Adresse
      '! angegeben werden.
      '! Bsp.:   A$="AB"      String
      '!         DIM a%(2)    Adress Dummy
      '!         X% = 0       Ziel
```

```
   '!         a%(1) = STRPTR (A$)
   '!         a%(2) = STRSEG (A$)
   '!         CALL MOVE (2, X%, a5(1)  verschiebe 2 Byte
   '-------------------------------------------------------

 2 $INLINE "move.com"

 3 END SUB

 4 SUB USE (handle%, filename$, status%)
   '-------------------------------------------------------
   '! Die Routine öffnet eine gültige DBASE III-Datei.
   '! Parameter:  handle%   = Nummer Filehandle
   '!             filename$ = Dateiname
   '!             status%   = Fehlerstatus USE-Aufruf
   '!                         0 -> ok,
   '!                         2 -> keine DBF-Datei (dBase III)
   '!                         3 -> DBF-Datei (dBase II)
   '!                         4 -> EOF erreicht
   '!                         5 -> kein Header Ende
   '!
   '-------------------------------------------------------

 5 SHARED header$, ver%, rec&, headb%, reclen%, anzahl%
 6 SHARED feldname$(), ftyp$(), laenge%(), komma%(), recofs&
 7 LOCAL i%; headend$, tmp$
 8 DIM a%(2)

   '! öffne die DBF-Datei im BINARY-Mode
   '! Achtung: da PB bei fehlender Datei diese
   '! anlegt, wird erst im INPUT-Mode geprüft,
   '! ob die Datei vorhanden ist!!!

 9 OPEN filename$ FOR INPUT AS #handle%  '! Datei vorhanden?
10 CLOSE #handle%
11 OPEN filename$ FOR BINARY AS #handle% '! öffne als Binary

12 GET$ #handle%, 32, header$            '! lese Kopf der Datei

13 ver% = (ASC(MID$(header$,1,1)))       '! decodiere Signatur
14 IF (ver% <> &H83) AND (ver% <> &H03) THEN
15  CLOSE #handle%                       '! schließen, da
16  status% = 2                          '! keine DBF
17  EXIT SUB
18 ELSEIF (ver% = &H02) THEN             '! DBASE II Header
19  CLOSE #handle%                       '! schließen, da
20  status% = 3                          '! dBase II DBF
21  EXIT SUB
22 END IF

23 tmp$ = MID$(header$,5,4)               '! Elemente decodieren
24 a%(1) = STRPTR(tmp$)
25 a%(2) = STRSEG(tmp$)
26 CALL MOVE (4,rec&,a%(1))               '! Zahl der Records
27 tmp$ = MID$(header$,9,2)
```

```
   28 a%(1) = STRPTR(tmp$)
   29 a%(2) = STRSEG(tmp$)
   30 CALL MOVE (2,headb%,a%(1))              '! Headerlänge
   31 tmp$ = MID$(header$,11,4)
   32 a%(1) = STRPTR(tmp$)
   33 a%(2) = STRSEG(tmp$)
   34 CALL MOVE (2,reclen%,a%(1))             '! Recordlänge

      '!-------------------------------------------------------
      '! lese und decodiere die Feldbeschreibung der DBASE III-
      '! Datei, es sind maximal 128 Felder zulässig
      '!-------------------------------------------------------

   35 anzahl% = ((headb% - 1) / 32) - 1       '! Zahl der Felder

   36 SEEK #handle%, 32                       '! setze Zeiger auf 1.
Feld
   37 FOR i% = 1 TO anzahl%                   '! lese n
Felddefinitionen
   38  GET$ #handle%,32, feld$                '! lese Definition
Feld
   39  IF EOF(handle%) THEN                   '! Fehler abfangen?
   40   CLOSE #handle%                        '! schließen, da
   41   status% = 4                           '! EOF erkannt
   42   EXIT SUB
   43  END IF

   44  feldname$(i%) = MID$(feld$,1,10)        '! Feldname
   45  ftyp$(i%) = MID$(feld$,12,1)            '! Feldtyp
   46  laenge%(i%) = ASC(MID$(feld$,17,1))     '! Länge
   47  komma%(i%) = ASC(MID$(feld$,18,1))      '! Dezimalstellen

   48 NEXT i%

      '!-------------------------------------------------------
      '! prüfe ob nächstes Byte das Header Ende signalisiert
      '!-------------------------------------------------------

   49 GET$ #handle%,1, headend$               '! lese Zeichen
   50 IF headend$ <> CHR$(&H0D) THEN          '! Ende = 0DH
   51   CLOSE #handle%                        '! schließen, da kein
   52   status% = 5                           '! Ende Header da
   53   EXIT SUB
   54 END IF

   55 recofs& = SEEK (handle%)                '! merke Offset 1.
Datensatz

   56 END SUB  '! ***** use ******



   57 SUB GetRecord (handle%, status%, buffer$)
      '!-------------------------------------------------------
      '! lese einen Satz aus der DBASE III - Datenbank und
```

```
      '! gebe das Ergebnis in buffer$ zurück. Die Daten sind
      '! als ASCII - Text in der Datenbank abgelegt.
      '! handle% = filehandle, status% = 0 ok., 1 = EOF
      '!-------------------------------------------------------
      '!
   58 SHARED header$, ver%, rec&, headb%, reclen%, anzahl%
   59 SHARED feldname$(), ftyp$(), laenge%(), komma%(), recofs&
   60 SHARED feldinh$()
   61 LOCAL i%, ptr%, lang%

   62  status% = 0
   63  SEEK #handle%, recofs&              '! auf Satzanfang
   64  GET$ #handle%, reclen% ,buffer$     '! lese Satz in Buffer
   65  IF EOF(handle%) THEN
   66   status% = 1    '! Error
   67  ELSE
   68   ptr% = 2
   69   FOR i% = 1 to anzahl%              '! separiere Felder
   70    lang% = laenge%(i%)               '! Feldlänge
   71    feldinh$(i%) = MID$(buffer$,ptr%,lang%)
   72    ptr% = ptr% + lang%
   73   NEXT i%
   74  END IF

   75 END SUB  '! ****** GetRecord *****
   76 SUB PutRecord (handle%, status%, buffer$)
      '!-------------------------------------------------------
      '! Schreibe einen Satz in die DBASE III - Datenbank.
      '! Die Daten sind als ASCII - Text im Puffer, geordnet
      '! nach Feldern, abzulegen. Achtung: Die Bufferlänge
      '! muß gleich der Recordlänge in DBASE sein !!!
      '! Der Inhalt des Puffers wird an der aktuellen Stelle
      '! in die Datenbank abgespeichert.
      '! handle% = filehandle, status% = 0 ok., 1 = Fehler
      '!-------------------------------------------------------
      '!
   77 SHARED header$, ver%, rec&, headb%, reclen%, anzahl%
   78 SHARED feldname$(), ftyp$(), laenge%(), komma%(), recofs&
   79 LOCAL datum$

   80  IF (LEN(buffer$) <> reclen%) THEN   '! Buffer = Satzlänge
   81   status% = 1                        '! Satzlänge falsch
   82   EXIT SUB
   83  END IF

   84  SEEK #handle%, recofs&               '! auf Satzanfang
   85  PUT$ #handle%, buffer$               '! schreibe Buffer in
DB


   86  datum$ = CHR$(VAL(MID$(DATE$,9,2)))_  '! Jahr
   87         + CHR$(VAL(MID$(DATE$,1,2)))_  '! Monat
   88         + CHR$(VAL(MID$(DATE$,4,2)))   '! Tag

   89  SEEK #handle%, 1
```

```
 90  PUT$ #handle%, datum$                  '! Datum aktualisieren
 91  status% = 0

 92 END SUB  '! ****** PutRecord *****


 93 SUB AppendBlank (handle%)
    '!-------------------------------------------------------
    '! Hänge einen leeren Satz in die DBASE III - Datenbank an.
    '! nach dem Aufruf steht der Schreiblesezeiger auf diesem
    '! Satz, d.h. PutRecord kann direkt Daten speichern.
    '!-------------------------------------------------------
    '!
 94 SHARED header$, ver%, rec&, headb%, reclen%, anzahl%
 95 SHARED feldname$(), ftyp$(), laenge%(), komma%(), recofs&

 96 LOCAL buf$, satz$, i%, tmp&

 97  satz$ = SPACE$(reclen%) + CHR$(&H1A) '! Leersatz mit EOF
 98  rec& = rec& + 1
 99  recofs& = headb% + (rec& * reclen%)  '! Endezeiger

100  SEEK #handle%, recofs&               '! an Dateiende
101  PUT$ #handle%, satz$                 '! append Leersatz

    '! Datum und Recordzahl im Header korrigieren

102  buf$   = CHR$(VAL(MID$(DATE$,9,2)))_  '! Jahr
103         + CHR$(VAL(MID$(DATE$,1,2)))_  '! Monat
104         + CHR$(VAL(MID$(DATE$,4,2)))   '! Tag

105  tmp& = rec&
106  FOR i% = 1 to 4
107   buf$ = buf$ + CHR$(tmp& AND &HFF)    '! in String
108   tmp& = tmp& / &H100
109  NEXT i%
110  SEEK #handle%, 1
111  PUT$ #handle%, buf$                  '! aktualisieren
112  status% = 0

113 END SUB  '! ****** AppendBlank *****

114 SUB Skip (handle%, status%, n%)
    '!-------------------------------------------------------
    '! Positioniere den Schreib- / Lesezeiger n Sätze weiter.
    '!-------------------------------------------------------
    '!
115 SHARED header$, ver%, rec&, headb%, reclen%, anzahl%
116 SHARED feldname$(), ftyp$(), laenge%(), komma%(), recofs&

117 LOCAL min1&, max1&, tmp&

118  status% = 0
119  min1& = headb%                      '! Grenzen
120  max1& = headb% + (rec& * reclen%)
```

```
 tmp& = recofs& + (reclen% * n%)
 IF tmp& < min1& THEN               '! Untergrenze prüfen
  status% = 1
 ELSEIF tmp& > max1& THEN          '! Obergrenze prüfen
  status% = 1
 ELSE
  recofs& = tmp&
 END IF

END SUB  '! ****** Skip *****

SUB GotoBottom (handle%, status%)
    '!-------------------------------------------------------
    '! Positioniere den Schreib- / Lesezeiger auf Satz 1.
    '!-------------------------------------------------------
    '!
SHARED header$, ver%, rec&, headb%, reclen%, anzahl%
SHARED feldname$(), ftyp$(), laenge%(), komma%(), recofs&

 status% = 0
 recofs& = headb%                    '! 1. Satz

END SUB  '! ****** GotoBottom *****

SUB GotoTop (handle%, status%)
    '!-------------------------------------------------------
    '! Positioniere den Schreib- / Lesezeiger auf letzen Satz
    '!-------------------------------------------------------
    '!
SHARED header$, ver%, rec&, headb%, reclen%, anzahl%
SHARED feldname$(), ftyp$(), laenge%(), komma%(), recofs&

 status% = 0
 recofs& = headb% + (reclen% * rec&)  '! letzter Satz

END SUB  '! ****** GotoTop *****

SUB DBEof (handle%, status%)
    '!-------------------------------------------------------
    '! Prüfe, ob EOF() der Datenbank erreicht ist
    '!-------------------------------------------------------
    '!
SHARED header$, ver%, rec&, headb%, reclen%, anzahl%
SHARED feldname$(), ftyp$(), laenge%(), komma%(), recofs&
LOCAL tmp&

 status% = 1
 tmp& = headb% + (reclen% * rec&)
 IF recofs& < tmp& THEN
  status% = 0                        '! True
 END IF

END SUB  '! ****** DBEof *****

    '! **** Ende ****
```

Listing 6.12: DBF_LIB.INC

DBDOC: Ein Anwendungsbeispiel

Um den Umgang mit einer dBase-Datei zu demonstrieren, wurde das Programm DBDOC erstellt. Es soll den Inhalt einer DBF-Datei lesen und am Bildschirm dokumentieren. Nachfolgendes Bild zeigt einen Ausschnitt aus der Bildschirmdarstellung:

```
DBASE III (DBF) DOC          (c) Born Version 1.0
File : .......

Header der DBASE III Datei

Version ..
Datum   ..
Records ..
Header Länge ...
Record Länge ...

Weiter bitte eine Taste betätigen

Feldbeschreibungen der Datei
.....
```

Bild 6.7: Ausgabe von DBDOC

Dabei kann gleichzeitig der Umgang mit den Funktionen erläutert werden.

Als erstes ist die Definition der globalen Variablen vorzunehmen. Hierzu ist die Anweisung:

```
$INCLUDE "DB_DEF.INC"
```

in das Programm aufzunehmen.

Nach der Kopfmeldung wird der Name einer DBF-Datei angefordert. Bei gültigen Eingaben ist anschließend die Datei mit USE zu öffnen. Bei fehlender Datei bricht das Programm über die Errorroutine ab. Bei erfolgreichem USE-Aufruf werden die Daten der globalen Variablen des Headers ausgegeben. Daran schließen sich die Feldbeschreibungen an.

Nachdem im Hauptprogramm die Kopfdaten dokumentiert wurden, läßt sich auf die Sätze zugreifen. Der Aufruf *GotoBottom* positioniert den internen Schreib-/Lesezeiger auf den ersten Datensatz. Dann demonstriert eine FOR..NEXT-Schleife, wie sich die Datenbank satzweise lesen läßt. Die Positionierung auf die Folgesätze übernimmt das Modul *Skip*.

Im nächsten Schritt verändert das Hauptprogramm den letzten Datensatz und sichert diesen mit *PutRecord* in der Datei. Beim Zugriff auf die Daten liegt der komplette Satz als ASCII-Text vor. An Hand der Felddefinition

läßt sich jedes Feld separieren. Beachten Sie aber, daß das erste Byte des Satzes zur Markierung gelöschter Daten dient.

Der Zugriff auf die Sätze der Datenbank ist sicherlich nicht immer über die gezeigt FOR-Schleife erwünscht. Deshalb wird zusätzlich der Umgang mit der Prozedur *DBEof* gezeigt. Mit ihr lassen sich die Sätze in einer WHILE-Schleife lesen. Da *GetRecord* nach dem Lesen eines Satzes die Feldinhalte separiert und in der globalen Feldvariablen *feldinh$()* ablegt, lassen sich diese recht einfach auf dem Bildschirm anzeigen.

Zum Abschluß des Demoprogrammes (DBDOC.BAS) wird noch ein leerer Datensatz mit *AppendBlank* eingefügt und mit dem Inhalt des vorletzten Satzes überschrieben.

Bei Bedarf können sicherlich noch einige weitere Funktionen zur Ergänzung der Bibliothek entstehen.

Das Beispielprogramm und die Module sind ausführlich kommentiert, so daß dem Umgang mit dBase-Dateien aus PowerBASIC-Programmen nichts mehr im Wege steht. Weitere Erläuterungen finden sich in nachfolgendem Listing.

```
X R E F   /Z=55                              (c) Born Version 1.0
Datei : dbdoc.bas      Datum : 07-20-1992          Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : DBDOC.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 10.6.92
      '! Autor      : G. Born
      '! Files      : DBASE File
      '! Progr. Spr.: PowerBasic 4.0 / 4.5
      '! Betr. Sys. : DOS 2.1 - 3.3
      '! Funktion: Demonstration des Zugriffs auf DBASE III Daten-
      '!           bankfiles aus PowerBasic. Das Programm gibt den
      '!           Inhalt einer DBASE III Datei auf dem Screen aus.
      '!           Dabei wird insbesondere der Umgang mit den ein-
      '!           zelnen Unterprogrammen gezeigt.
      '!
      '! Aufruf:   DBDOC
      '*********************************************************
      '! definiere die Header Datenstrukturen !!!!!!

    1 $INCLUDE "DB_DEF.INC"

    2 filename$ = ""                        '! Dateiname
    3 ein%      = 1                         '! Filehandle

    4 ON ERROR GOTO fehler                  '! Fehlerhandler

      '#########################################################
      '#                      Hauptprogramm                    #
      '#########################################################
```

```
   '! #####   Kopf ausgeben  ######
 5 CLS                                    '! Clear Screen
 6 PRINT "DBASE III (DBF) DOC              (c) Born Version 1.0"
 7 PRINT
 8 INPUT "File    : ", filename$          '! lese Dateiname
 9 PRINT

10 IF filename$ = "" THEN                 '! Leereingabe ?
11  PRINT "Der Name der Eingabedatei fehlt"
12  END
13 END IF

   '!-------------------------------------------------------
   '!     *** Bearbeitung der DBASE III Datei ***
   '!-------------------------------------------------------

14 CALL USE (ein%, filename$,status%)        '! öffne Datei

15 IF status% <> 0 THEN
16  PRINT "Fehler : "; status%
17  END
18 END IF

   '!-------------------------------------------------------
   '! ***  Ausgabe des Headers der DBASE III Datei ***
   '! Die Version gibt dabei an, ob intern Memofelder be-
   '! nutzt wurden (Version = 83H -> Memodatei)
   '!-------------------------------------------------------

19 PRINT "Header der DBASE III Datei"
20 PRINT
21 PRINT "Version       ";HEX$(ver%)
22 PRINT "Datum        ";ASC(MID$(header$,4,1));".";
23 PRINT ASC(MID$(header$,3,1));".";
24 PRINT ASC(MID$(header$,2,1))
25 PRINT "Records      "; rec&
26 PRINT "Header Länge "; headb%
27 PRINT "Record Länge "; reclen%
28 PRINT

29 INPUT "Weiter, bitte die <RET> Taste betätigen", tmp$

   '!-------------------------------------------------------
   '! lese und decodiere die Feldbeschreibung der DBASE III-
   '! Datei, es sind maximal 128 Felder zulässig
   '!-------------------------------------------------------

30 PRINT "Feldbeschreibung der Datei ";filename$
31 PRINT
32 PRINT "Feldname        Typ        Stellen      Kommastellen"
33 PRINT "-----------Å----------Å---------Å-------------"

34 FOR i% = 1 TO anzahl%                  '! n Felddefinitionen
```

```
   35  PRINT feldname$(i%);"     ";                 '! Name des
Feldes
   36  SELECT CASE ftyp$(i%)                  '! gebe Feldtyp aus
   37  CASE "N"
   38   PRINT "Numerisch   ";
   39  CASE "C"
   40   PRINT "Character   ";
   41  CASE "L"
   42   PRINT "Logical     ";
   43  CASE "D"
   44   PRINT "Datum       ";
   45  CASE "M"
   46   PRINT "Memo        ";
   47  END SELECT

   48  PRINT USING "\  \##";"    ";laenge%(i%);    '! Feldlänge
   49  PRINT "         ";komma%(i%)               '!
Nachkommastellen
   50 NEXT i%
   51 PRINT "-----------Á----------Á---------Á-------------"

      '! *** Hinweis: Die Recordlänge ist 1 Byte größer als dies
      '!              aus den Feldlängen ersichtlich ist, da
      '!              im ersten Byte des Records die Information
      '!              für gelöschte Sätze steht (*).

   52 PRINT "Recordlänge in Byte       "; reclen%
   53 PRINT

   54 INPUT "Weiter, bitte die <RET> Taste betätigen", tmp$

      '!-----------------------------------------------------
      '! lese und decodiere die Datensätze der DBASE III Datei
      '!-----------------------------------------------------

   55 PRINT "Datensätze der DBASE III Datei "; filename$
   56 PRINT
      '!-----------------------------------------------------
      '! Hier wird gezeigt, wie der Inhalt der Datei satzweise
      '! per FOR Schleife gelesen werden kann.
      '!-----------------------------------------------------
   57 CALL GotoBottom (ein%,status%)         '! auf 1. Satz
   58 FOR i& = 1 TO rec&                     '! Schleife über alle
Records
   59  CALL GetRecord (ein%, status%, satz$) '! lese Satz
   60  PRINT satz$                           '! dokumentiere Satz
   61  CALL Skip (ein%, status%, 1)          '! nächster Satz
   62 NEXT i&

      '!-----------------------------------------------------
      '! Der Inhalt des aktuellen Satzes wird verändert und in die
      '! Datenbank zurückgespeichert
      '!-----------------------------------------------------

   63 satz1$ = " " + "Hallo" + MID$(satz$,7) '! ändere Feld 1
```

```
   64 CALL PutRecord (ein%, status%, satz1$) '! speichere Satz

      '!-------------------------------------------------------
      '! Alternativ besteht die Möglichkeit, die Datei satzweise
      '! zu lesen, bis EOF() erreicht ist. Hierfür dient die
      '! Funktion DBEof().
      '!-------------------------------------------------------

   65 PRINT "Lese Datei nochmals"
   66 CALL GotoBottom (ein%,status%)         '! auf 1. Satz
   67 CALL DBEOF(ein%,status%)               '! EOF erreicht ?
   68 WHILE status% = 0
   69  CALL GetRecord (ein%,status%,satz$)   '! lese Satz

   70  FOR i% = 1 TO anzahl%                 '! gebe Felder aus
   71   PRINT feldname$(i%), " : "; feldinh$(i%)
   72  NEXT i%

   73  INPUT "Weiter, bitte die <RET> Taste betätigen", tmp$

   74  CALL Skip (ein%, status%, 1)          '! nächster Satz
   75  CALL DBEOF(ein%,status%)              '! EOF erreicht ?
   76 WEND
   77 PRINT "EOF Erreicht"

   78 INPUT "Weiter, bitte die <RET> Taste betätigen", tmp$

      '!-------------------------------------------------------
      '! Es wird ein leerer Satz angefügt und mit dem Inhalt des
      '! vorletzten Satzes überschrieben
      '!-------------------------------------------------------

   79 PRINT "Leersatz anhängen"
   80 CALL AppendBlank (ein%)                '! Leersatz anhängen
   81 CALL PutRecord (ein%, status%, satz$)  '! alten Satz
speichern

   82 INPUT "Weiter, bitte die <RET> Taste betätigen", tmp$

   83 CLOSE

   84 PRINT "Ende DBDOC"
   85 END

      '############################################################
      '#                      Hilfsroutinen                       #
      '############################################################

   86 fehler:
      '-------------------------------------------------------
      '! Fehlerbehandlung in DBDOC
      '-------------------------------------------------------

   87 IF ERR = 53 THEN
   88  PRINT "Fehler: Datei nicht gefunden"
```

```
89  CLOSE
90  END
91 ELSE
92  PRINT "Fehler : "; ERR; " unbekannt"
93  PRINT "Programmabbruch"
94  END
95 END IF
96 END                                    '! MSDOS Exit
97 RETURN

98 $INCLUDE "DBF_LIB.INC

   '! **** Ende ****
```

Listing 6.13: Listing des Programms DBDOC.BAS

PCXV: Anzeige von PCX-Dateien

Die weite Verbreitung des Programmes Paintbrush (ZSoft) führte dazu, daß das PCX-Format als Standard zur Speicherung von Pixelgrafiken benutzt wird. Viele Fremdprogramme können Bilder in diesem Format bearbeiten. Auch die Windows-Version von Paintbrush unterstützt das PCX-Format. Deshalb entstand die Idee, ein kleines PowerBASIC-Programm zur Anzeige von PCX-Dateien zu schreiben.

Die nachfolgend aufgezeigte Lösung ist allerdings nur als Demonstrationsbeispiel für den Zugriff auf PCX-Files zu verstehen. Das Programm weist daher noch einige Mängel auf. So wird die Farbpalette aus der PCX-Datei nicht ausgewertet, die Bilder können demnach in falschen Farben angezeigt werden. Außerdem ist die Ausgabegeschwindigkeit auch auf 80386-Rechnern alles andere als berauschend. Für professionelle Zwecke kann wohl nicht auf Assembler und C verzichtet werden.

Der Entwurf

Das Programm soll nach dem Start den Namen einer PCX-Datei sowie gegebenenfalls eine Option erfragen. Anschließend ist die Datei zu öffnen, auf einen gültigen PCX-Header zu prüfen und als Grafik auszugeben. Alternativ kann der Dateiname bereits in der Kommandozeile mit angegeben werden:

```
PCXV filename [/D]
```

Die Option /D schaltet dabei den DEBUG-Modus ein, so daß vor der Grafikausgabe einige Parameter der Datei mit angezeigt werden. Über den Befehl:

```
PCXV /?
```

läßt sich der Text der Online-Hilfe aktivieren. In dieser Hinsicht gleicht das Programm den bisher vorgestellten Lösungen.

Vor der Implementierung möchte ich aber noch einige Hinweise bezüglich des PCX-Formates geben.

Das PCX-Format

Dieses Format erlaubt die Speicherung von Pixelgrafiken in Dateien. Die Datei besteht aus einem Header von 128 Byte und dem anschließenden Grafikteil. Der Header besitzt eine Struktur gemäß Tabelle 6.8.

Offset	Bytes	Bemerkungen
00H	1	0AH als PCX-Signatur
01H	1	PCX-Version 0 = Version 2.5 2 = Version 2.8 mit Palette 3 = Version 2.8 ohne Palette 5 = Version 3.0
02H	1	Komprimierungsflag 0 = keine Komprimierung 1 = RLE Komprimierung
03H	1	Bit pro Pixel (meist 1)
04H	8	Koordinaten des Bildes
0CH	2	horiz. Auflösung Bildpunkt
0EH	2	vert. Auflösung Bildpunkt
10H	48	Color Map (16x3)
40H	1	reserviert
41H	1	Zahl der Farbebenen (max. 4)
42H	2	Bytes pro Bildzeile (gerade)
44H	2	Palettedaten 1 = Farbe - S/W 2 = Graustufen
45H	58	reserviert

Tabelle 6.8: Format des PCX-Headers

Das erste Byte enthält eine Signatur für gültige PCX-Dateien. Daran schließt sich eine Versionsnummer für die PCX-Version an. In der Version 5 sind die Palettedaten nicht mehr im Header sondern am Dateiende gespeichert. Das Byte ab Offset 02H definiert die Codierungsart. Mit 0 werden die Daten uncodiert und mit 1 per RLE-Codierung gespeichert. Mit der RLE-Codierung läßt sich Speicherplatz sparen. Offset 3 gibt die Zahl der Bit pro Pixel an, wobei der Wert meist auf 1 gesetzt ist.

Die Bildkoordinaten (X1,Y1,X2,Y2) finden sich als Worte ab Offset 04H. Interessant ist das Byte ab Offset 65 (41H), welches die Zahl der Farbebenen angibt. Der Wert ab Offset 66 definiert die Länge einer (unkomprimierten) Zeile in Byte.

Bei der PCX-Version 2 sind ab Offset 16 (10H) die 16 Farben á 3 Byte der Palette gespeichert. Bei der Version 5 befindet sich die Palette mit 256 Farben á 3 Byte in den letzten 769 Byte der Datei. Das erste Byte des Anhangs enthält dann die Signatur 0CH.

Die Bilddaten schließen sich an den Header an. Ein Bild wird dabei zeilenweise abgetastet und in einzelnen Punkten gespeichert. Bei Farbbildern wird das Bild vorher in vier Ebenen mit den Grundfarben Rot, Grün, Blau und Intensität aufgeteilt. Dann wird die erste Zeile mit den Punkten der Farbe Rot gespeichert. Daran schließt sich die Zeile mit den Punkten der Farbe Grün an. Nach Grün folgt Blau und die Intensität. Erst wenn alle Farbebenen der ersten Zeile gespeichert wurden, schließt sich die nächste Zeile an. Nun muß noch unterschieden werden, ob die Daten codiert oder uncodiert vorliegen. Bei einer uncodierten Speicherung können die Bildpunkte gelesen und auf dem Bildschirm ausgegeben werden. Die Zahl der Bytes pro Zeile steht dann im Header.

Bei der RLE-Codierung versucht man, mehrere gleiche Bildpunkte zusammenzufassen, um damit eine Bildkomprimierung zu erhalten. Für die RLE-Codierung gilt:

- Sind die beiden oberen Bits (6,7) eines Bytes auf 1 gesetzt, liegt eine komprimierte Information vor. Die restlichen Bits 0 bis 5 sind dann als Wiederholzähler zu interpretieren. Das Folgebyte ist dann n mal zu wiederholen um die ursprüngliche Bildfolge zu erhalten.
- Sind die beiden oberen Bits (6,7) eines Bytes auf 0 gesetzt, liegt ein einfaches Datenbyte vor. Dann sind alle Bits als Bilddaten zu interpretieren.

Die Hexcodesequenz:

C3 33 07 41

entspricht der Bilddatenfolge:

33 33 33 07 41

Bei Bildbereichen mit gleicher Farbe wird häufig die RLE-Codierung benutzt.

Damit möchte ich die Beschreibung des PCX-Formates beenden. Wer sich für die Thematik interessiert, sei auf /7/ verwiesen, wo sich eine detaillierte Beschreibung verschiedener Formate findet.

Die Implementierung

Die Implementierung erfolgt in PowerBASIC, wobei verschiedene Module benutzt werden. Nachfolgend möchte ich kurz die Struktur des Programmes vorstellen.

fehler

Diese Prozedur fängt Laufzeitfehler ab und beendet das Programm mit einer Fehlermeldung.

MOVE (len%, Ziel, Quelle)

Dies ist die bereits im Modul DBDOC benutzte Assemblerprozedur zum Kopieren mehrerer Bytes. Die Prozedur wird zur Typkonvertierung der Headerdaten verwendet.

getheader (status%, header$)

Dieses Modul erwartet in *header$* die 128 Byte des PCX-Headers. Dann wird geprüft ob eine gültige PCX-Signatur vorliegt. Bei einem Fehler wird *status%=1* zurückgegeben. Andernfalls zerlegt *getheader* die Headerdaten und speichert diese in globalen Variablen. Diese lassen sich dann durch das Hauptprogramm auswerten. Die Prozedur *MOVE* wird zur Typkonvertierung der Headerdaten benutzt.

Plotline (x1%, y1%, lenx%)

Diese Prozedur gibt *lenx%* Pixel einer Zeile an der Position x1,y1 aus. Die Bildpunkte selbst sind in *pixel%(a,b)* gespeichert. Beachten Sie, daß pro Zeile bei der Farbdarstellung vier Pixelreihen auszugeben sind. Der PSET-Befehl in PowerBASIC erwartet diese Werte im Parameter *bit%*. Deshalb sind die aufwendigen Maskierungen innerhalb der FOR-Schleife erforderlich. Weiterhin müssen 8 Bit pro Byte gespeichert werden.

Das Hauptprogramm

Das Hauptprogramm ist recht einfach aufgebaut. Nach der Variablendefinition ermittelt es den Dateinamen und die Optionen. Dann wird die Datei geöffnet und die 128 Byte des Headers gelesen. Nach der Decodierung lassen sich die Parameter für Testzwecke anzeigen.

Nach der Aktivierung des Grafikmodus beginnt die Decodierung der Bilddaten. Eine FOR-Schleife geht dann über alle Zeilen eines Bildes. Die innere FOR-Schleife berücksichtigt dabei die Farbebenen. Die Zeile wird durch die WHILE-Schleife bearbeitet. Die Schleife wird abgebrochen, sobald die Zahl der Datenbytes pro Zeile erreicht ist. Mit *Plotline* lassen sich die Grafikdaten einer Zeile ausgeben.

Die zahlreichen Bitoperationen sowie die Ausgabe über PSET führen zu einem recht langsamen Bildaufbau. Die direkte Codierung in Assembler bringt hier deutliche Vorteile. Da es sich lediglich um ein Demonstrationsprogramm handelt, habe ich auch darauf verzichtet, die Palettedaten auszuwerten. Daher kann die Farbdarstellung der Bilder vom Original abweichen. Hier ergeben sich sicherlich einige Verbesserungsmöglichkeiten. Die Einzelheiten sich dem folgenden Listing zu entnehmen.

```
X R E F   /Z=50                              (c) Born Version 1.0
Datei : pcxv.bas      Datum : 07-21-1992          Seite :  1

 Zeile     Anweisung

      ' ********************************************************
      ' File       : PCXV.BAS
      ' Vers.      : 1.0
      ' Last Edit  : 20. 5.92
```

```
      ' Autor      : G. Born
      ' File I/O   : INPUT, OUTPUT, FILE, PRINTER
      ' Progr. Spr.: POWERBASIC
      ' Betr. Sys. : DOS 2.1 - 5.0
      ' Funktion: Das Programm zeigt einen PCX-Datei an.
      '
      ' Aufruf:   PCXV Filename
      '
      ' ***********************************************************
      '! Headervariable definieren
    1 signatur%=0
    2 Version%=0
    3 encoding%=0
    4 bits%=0
    5 x1%=0
    6 y1%=0
    7 x2%=0
    8 y2%=0
    9 hres%=0
   10 vres%=0
   11 planes%=0
   12 bytel%=0
   13 palinfo%=0
   14 head$ = ""

   15 debug% = 0                              '! Ausgabe Header
ausschalten

      '! Puffer für 1 Zeile mit Bilddaten
   16 DIM pixel%(4,1024)                      '! 4 Ebenen a 1024
Pixel

   17 breite% = 0                             '! Pixel pro Zeile
   18 hoehe% = 0                              '! Zeilen pro Bild


   19 ON ERROR GOTO fehler                    '! Fehlerausgang

      '############################################################
      '#                      Hauptprogramm                       #
      '############################################################

   20 kommando$ = COMMAND$                    '! Parameter ?
   21 IF LEN (kommando$) = 0 THEN             '! User Mode ?
   22  CLS                                    '! clear Screen

   23  PRINT "P C X  - V i e w                  (c) Born Version
1.0"
   24  PRINT
   25  INPUT  "File     : ",filename$
   26  PRINT
   27  INPUT  "Option /D: ",options$
   28  PRINT
   29 ELSE
   30  ptr% = INSTR (kommando$,"/?")          '! Option /?
```

```
31  IF ptr% <> 0 THEN                       '! Hilfsbildschirm
32   PRINT "P C X - V i e w                 (c) Born Version 1.0"
33   PRINT
34   PRINT "Aufruf: PCXV <Filename> [/D]"
35   PRINT
36   PRINT "Zeigt eine PCX-Datei am Bildschirm an. Die"
37   PRINT "Option /D schaltet den DEBUG-Mode ein."
38   PRINT
39   SYSTEM
40  END IF
41                              '! Kommando Mode
42  ptr% = INSTR (kommando$,"/")           '! Optionen ?
43  IF ptr% = 0 THEN
44   filename$ = kommando$                 '! nur Filename
45   options$ = ""
46  ELSE
47   filename$ = LEFT$(kommando$,ptr% -1)'! Filename separieren
48   options$  = MID$(kommando$,ptr%)      '! Optionen separieren
49  END IF

50  IF filename$ = "" THEN                  '! Leereingabe ?
51   PRINT
52   PRINT "Der Dateiname fehlt"
53   SYSTEM
54  END IF

55 END IF

   '! DEBUG-Option gesetzt (/D)

56  options$ = UCASE$(options$)
57  ptr% = INSTR (options$,"/D")           '! Debug-Option ?

58  IF ptr% > 0 THEN
59   debug% = 1                            '! DEBUG-Mode ein
60  END IF

   ' prüfe ob Datei vorhanden, nein -> exit

61  OPEN filename$ FOR INPUT AS #1         '! File exist?
62  CLOSE #1
63  OPEN filename$ FOR BINARY AS #1        '! öffne Datei

64  GET$ #1, 128, head$                    '! lese Header

65  CALL GetHeader (status%, head$)        '! decodiere Header

66  IF (status% <> 0) THEN
67   PRINT "Keine gültige PCX-Datei"
68   CLOSE
69   SYSTEM
70  END IF

71  IF debug% = 1 THEN
72   PRINT "Header "; signatur%; " "; Version%
```

```
   73   PRINT "Encoding ";encoding%;" Bits ";bits%
   74   PRINT "Bild ";X1%;" ";Y1%;" ";x2%;" ";Y2%; " Pixel ";x2%-
x1%
   75   PRINT "Planes ";planes%;" Bytes/(Zeile) ";bytel%
   76   PRINT
   77   INPUT "Weiter, bitte eine Taste betätigen", tmp$
   78  END IF

   79   SCREEN 12: CLS                        '! Graphikmode

   80 breite% = x2% - x1%                    '! Pixel / Zeile
   81 hoehe% = y2% - y1%                     '! Zeilen / Bild

   82 FOR i% = 0 TO hoehe%                   '! alle Zeilen
   83  FOR k% = 1 to planes%                 '! alle Farbebenen
   84   bcount% = 0                          '! decodierte Bytes
   85   ptr% = 1                             '! Hilfszeiger
   86   WHILE bcount% < bytel%               '! lese n Bytes

      '! Datei sequentiell lesen und byteweise decodieren
   87    GET$ #1, 1, zchn$                   '! lese 1 Byte
   88    byte% = ASC(zchn$)                  '! konvert. in Byte

   89    IF byte% > &HC0 THEN                '! komprimiert ?
   90     count% = byte% AND &H3F            '! Wiederholfaktor
   91     bcount% = bcount% + count%         '! Zahl der Bildbytes
   92     GET$ #1, 1, zchn$                  '! Datenbyte
   93     byte% = ASC(zchn$)
   94     FOR l% = 1 to count%               '! generiere Daten
   95      pixel%(k%,ptr%) = byte%           '! speichere Byte
   96      INCR ptr%
   97     NEXT l%
   98    ELSE
   99     INCR bcount%                        '! Zahl der Bildbytes
  100     pixel%(k%,ptr%) = byte%             '! speichere Byte
  101     INCR ptr%
  102    END IF
  103   WEND
  104  NEXT k%
      '! Bildzeile ausgeben
  105  Call Plotlinie (x1%, y1%+i%, bcount%-1)
  106 NEXT i%

  107 tmp$ = INPUT$ (1)

  108 LOCATE 24,1

  109 PRINT
  110 PRINT "Ausgabe beendet"
  111 CLOSE                                 '! Dateien schließen

  112 END


      '############################################################
```

```
    '#                    Hilfsroutinen                          #
    '###########################################################

113 fehler:
    '-----------------------------------------------------------
    '! Fehlerbehandlung in TEXTS
    '-----------------------------------------------------------

114 IF ERR = 53 THEN
115  PRINT "Die Datei ";filename$;" existiert nicht"
116 ELSE
117  PRINT "Fehler : ";ERR;" unbekannt"
118  PRINT "Programmabbruch"
119 END IF
120  END                                    '! MSDOS Exit
121 RETURN


122 SUB getheader (status%, header$)
    '-----------------------------------------------------------
    '! Lese die Daten des Headers in Variable
    '-----------------------------------------------------------
123 SHARED signatur%, Version%, encoding%, bits%, x1%, y1%
124 SHARED x2%, y2%, hres%, vres%, colormap%, planes%, bytel%
125 SHARED palinfo%
126 LOCAL ptr%, tmp$
127 DIM a%(2)

    '! setze die Infos im Header in Variable um

128 status% = 0
129 signatur% = ASC(MID$(header$,1,1))     '! Signatur PCX-File
130 IF signatur% <> 10 THEN                '! teste Signatur
131  status% = 1
132  EXIT SUB
133 END IF

134 Version% = ASC(MID$(header$,2,1))      '! Versionsnummer
135 encoding% = ASC(MID$(header$,3,1))     '! Kodierungsflag
136 bits% = ASC(MID$(header$,4,1))         '! Bits pro Ebene

137 tmp$ = MID$(header$,5,2)               '! Xmin decodieren
138 a%(1) = STRPTR(tmp$)
139 a%(2) = STRSEG(tmp$)
140 CALL MOVE (2,x1%,a%(1))

141 tmp$ = MID$(header$,7,2)               '! Ymin decodieren
142 a%(1) = STRPTR(tmp$)
143 a%(2) = STRSEG(tmp$)
144 CALL MOVE (2,y1%,a%(1))

145 tmp$ = MID$(header$,9,2)               '! Xmax decodieren
146 a%(1) = STRPTR(tmp$)
147 a%(2) = STRSEG(tmp$)
148 CALL MOVE (2,x2%,a%(1))
```

```
  149 tmp$ = MID$(header$,11,2)               '! Ymax decodieren
  150 a%(1) = STRPTR(tmp$)
  151 a%(2) = STRSEG(tmp$)
  152 CALL MOVE (2,y2%,a%(1))

  153 planes% = ASC(MID$(header$,66,1))      '! Planes decodieren

  154 tmp$ = MID$(header$,67,2)               '! Bytes pro Zeile
decodiere
       n
  155 a%(1) = STRPTR(tmp$)
  156 a%(2) = STRSEG(tmp$)
  157 CALL MOVE (2,bytel%,a%(1))
  158 bytel% = bytel%
  159 END SUB


  160 SUB Plotlinie (x1%,y1%,lenx%)
      '-------------------------------------------------------
      '! CALL Plotline (....)
      '! Die Prozedur gibt die Bilddaten in einer Zeile aus.
      '-------------------------------------------------------
  161 SHARED planes%, breite%
  162 SHARED pixel%()()
  163 LOCAL i%, k%, ptr%

  164 ptr% = x1%
  165 FOR i% = 1 TO lenx%+1
  166  mask% = 128                                  '! oberstes
Bit
  167  FOR k% = 8 TO 1 STEP - 1                     '! alle Bits
       bit%=0
  168   IF (pixel%(1,i%) AND mask%) <> 0 THEN bit%=1
  169   IF (pixel%(2,i%) AND mask%) <> 0 THEN bit%=bit%+2
  170   IF (pixel%(3,i%) AND mask%) <> 0 THEN bit%=bit%+4
  171   IF (pixel%(4,i%) AND mask%) <> 0 THEN bit%=bit%+8

  172   PSET (ptr%,y1%), (bit%)
  173   INCR ptr%                                    '! next Point
  174   mask% = mask% / 2                            '! next Bit
  175  NEXT k%
  176 NEXT i%
  177 END SUB

  178 SUB MOVE INLINE
      '-------------------------------------------------------
      '! CALL MOVE (LEN, ZIEL, QUELLE)
      '! Die Prozedur verschiebt n Byte eines Strings in die
      '! Zieladresse. Achtung: der String muß mit seiner Adresse
      '! angegeben werden.
      '! Bsp.:   A$="AB"      String
      '!         DIM a%(2)    Adress Dummy
      '!         X% = 0       Ziel
      '!         a%(1) = STRPTR (A$)
```

```
    '!          a%(2) = STRSEG (A$)
    '!          CALL MOVE (2, X%, a5(1)  verschiebe 2 Byte
    '-------------------------------------------------------

179 $INLINE "move.com"

180 END SUB

    ' ###### Programm Ende #########
```

Listing 6.14: PCXV.BAS

HWINFO: Konfigurationsprüfung per Software

Wer sich mit Rechnern beschäftigt, dem ist wohl folgendes Problem bestens bekannt: Da steht ein Personalcomputer zur Verfügung, über dessen Konfiguration keine Unterlagen vorhanden sind. Dann geht die Raterei los: wie groß ist der RAM-Bereich, wo liegen die BIOS-ROMs und welche Schnittstellen sind schon im Gerät vorhanden? Den Programmierer interessieren Informationen über die interne Belegung des Speichers oder der Festplatte. Das Studium der Hardwareunterlagen (falls überhaupt beschaffbar) ist aufwendig und führt nicht immer zum Erfolg. Man denke nur an die vielen importierten Systeme, zu denen keine Dokumentation existiert. Eine Analyse der technischen Unterlagen und der Hardware setzt weiterhin entsprechende Erfahrungen und Kenntnisse voraus. Fragen zur internen Aufteilung des Speichers oder der Festplatte lassen sich über diesen Ansatz prinzipiell nicht lösen. Wesentlich eleganter scheint die Möglichkeit einer Konfigurationsabfrage per Software. Wie dies funktioniert, wird in nachfolgendem Programm vorgestellt. Dieses Programm dokumentiert die wichtigsten Parameter des Systems per Bildschirm.

Der Ansatz geht davon aus, daß MS-DOS ja ebenfalls wissen muß, welche Konfiguration vorliegt. Peter Norton und andere Insider haben diesen Gedanken bereits vor Jahren aufgegriffen, ohne aber die internen Funktionen offenzulegen. Der folgende Abschnitt beschreibt deshalb den Lösungsweg, um einem breiteren Kreis von Anwendern die Möglichkeit zu eigenen Erweiterungen zu bieten.

Die Anforderungen

Vor der Entwicklung dieses Programmes werden wieder die Anforderungen hinsichtlich der Funktionalität festgelegt. Wichtig ist sicherlich die Dokumentation der Belegung des Hauptspeichers. Hier interessiert nicht nur die Größe des Speichers, sondern auch die Lage des Bildschirm-RAM sowie die Adressen der diversen BIOS-ROMs. Bei der Softwareentwicklung ist vielleicht noch die Größe des durch das System belegten Speichers und die Startadresse des ersten Anwenderprogrammes relevant. Bild 6.8 zeigt den genauen Aufbau der Ausgaben des Programmes HWINFO.

```
S y s t e m   I N F O          (c) Born  Version 1.0


Bios Version ............

MS - DOS Version 3.20

RAM from     0000:0000 len 512 Kbytes
RAM from     0000:0000
ROM from     E800:0000 len 8 Kbytes
ROM from     F800:0000
User Progr. Adresse : 0E01:0000
MS-DOS Free Memory  : 464608 Bytes
MS-DOS Used Memory  :  59664 Bytes

Disks              A:

Disk               C: 21204992 Bytes
Sektoren pro Cluster : 4
Bytes pro Sektor     : 512
Cluster              : 10354
Freie Kapazität      : 11415552 Bytes (53,8 %)

Colorcard 80 x 25 Monochrom

Serial   Interfaces : 1
Parallel Interfaces : 1

Ende System Info
```

Bild 6.8: Ausgabe der Konfiguration per Bildschirm

Als erstes wird das Copyright des BIOS-ROM angezeigt. Dann folgt die Versionsnummer des Betriebssystems. Nur bei den Versionen unterhalb 2.0 wird keine Unterversion angezeigt. Der Hauptspeicher muß ab Adresse 0000:0000 beginnen und hat eine Länge von n Kbyte. Beim Bildschirmspeicher wird nur dessen Anfangsadresse im Speicher angegeben. Aus dieser Information sowie aus dem Bildschirmmodus läßt sich auf die Speichergröße schließen.

Die Prüfung auf ROM-Bereiche gibt in jedem Fall die Startadresse an. Läßt sich auch die Länge ermitteln, wird diese Information ebenfalls mit ausgegeben. Ansonsten wird der Speicher in 32-Kbyte-Schritten auf solche ROMs überprüft. Bei unvollständig codierten Adreßbereichen wird ein ROM an mehreren Stellen gespiegelt. Dies wird aus Aufwandsgründen nicht abgefangen, sondern die Adressen werden angezeigt.

Die Startadresse des ersten Anwenderprogrammes ist in der Segment-Offset-Notation anzugeben (z.B. 0890:0000). Bei dieser Notation wird der 1-Mbyte-Adreßraum in Segmente zu je 16 Byte unterteilt. Dies ergibt genau 65535 Segmente, die sich mit einem 16-Bit-Register adressieren lassen. Der Offsetwert gibt dann die Adresse relativ zum Segmentanfang

an. Die absolute (physikalische) Adresse läßt sich dann folgendermaßen berechnen:

Adresse = Segment * 16 + Offset

Die Zahl der Diskettenlaufwerke wird durch den jeweiligen Laufwerksbuchstaben (A:, B: etc.) angezeigt. Bei Festplatten und RAM-Disks erscheinen zusätzlich Informationen über die interne Organisation. Platten werden in Sektoren zu n Byte unterteilt. MS-DOS speichert standardmäßig 512 Byte pro Sektor. Zur besseren internen Verwaltung werden mehrere Sektoren zu einem Cluster kombiniert. Ausführliche Informationen über die interne Verwaltung von Festplatten und Floppys finden sich in /1/. Die Clustergröße sowie die Anzahl der Cluster auf dem Medium sind ebenfalls in der Anzeige dargestellt. Aus diesen Angaben läßt sich auch die Speicherkapazität eines Mediums bestimmen.

Zum Abschluß ist die Zahl der Schnittstellen (seriell/parallel) und die Art des Grafikadapters anzuzeigen.

Der Entwurf

Nach der Definition der Anforderungen stellt sich die Frage, wie die Informationen zu ermitteln sind. Da MS-DOS die Konfiguration beim Systemstart ermittelt, läßt sich diese über verschiedene System- und BIOS-Funktionen ermitteln.

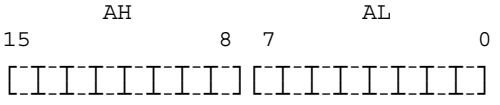
Die Versionsnummer läßt sich einmal über das DOS-Kommando VER ermitteln. Durch den Basic-SHELL-Befehl kann dieses Kommando aktiviert werden. Aber DOS bietet einen anderen Weg über die Funktionen des Interrupt 21. (Leider können die Systemaufrufe nur kurz vorgestellt werden, da sonst der Umfang dieses Buches gesprengt wird. Für Leser die sich für den internen Aufbau von MS-DOS und die Schnittstellen zu den BIOS- und Systemaufrufen (INT 21) interessieren, bietet /1/ auf über 800 Seiten ausführliche Informationen zu diesem Themen.) Doch nun zurück zur Ermittlung der Versionsnummer über die INT-21-Funktion 30H. Dieser gibt den Versionscode im Register AX zurück. Weitere Hinweise finden sich im Abschnitt über die Implementierung.

Der nächste Schritt dient zur Bestimmung der RAM- und ROM-Bereiche. Hier besteht die Möglichkeit, innerhalb des 1-Mbyte-Adreßraumes einen Speichertest durchzuführen. In Schritten zu 4 Kbyte ließe sich prüfen, ob eine Adresse beschreibbar (RAM) ist. Enthält der Speicher nach dem Schreibvorgang (Wert auf 55H setzen) den Wert 0FFH, ist die Adresse mit hoher Wahrscheinlichkeit unbelegt. Alle anderen Werte ungleich dem geschriebenen Wert deuten auf ROMs hin. Bei der Ermittlung der ROM-Bereiche wird diese Strategie auch verwendet. Ein RAM-Test ist in PowerBASIC aber recht aufwendig (PEEKE, POKE). Insbesondere ist vor dem Test das Interruptsystem zu sperren, um Systemabstürze zu verhindern. Deshalb wird die Speichergröße über den INT 12 des BIOS ermittelt. Dieser Aufruf gibt die Größe in Kbyte im Register AX zurück. Die Startadresse liegt fest auf dem Wert 0000:0000. Die Ermittlung des

Bildschirmspeichers geht ebenfalls recht einfach. Er kann auf den Segmentadressen A000H, B000H oder B800H liegen. Es wird deshalb geprüft, ob an diesem Stellen Werte ungleich FFH vorliegen. In solchen Fällen wird ein RAM-Bereich gemeldet. Der BIOS-ROM-Bereich wird in Schritten zu 32 Kbyte auf Werte ungleich FFH abgefragt. Damit lassen sich auch die Anfangsadressen der ROMs bestimmen.

Bleibt noch die Ermittlung der Startadresse des ersten Anwenderprogrammes und des freien Speicherbereiches. Hier wird mit der (undokumentierten) INT-21-Funktion 51H gearbeitet. Diese gibt die Segmentadresse des Programm-Segment-Prefix-Bereiches (PSP) wieder. Hierbei handelt es sich um einen Datenbereich von 255 Byte, der von MS-DOS vor jedes Anwenderprogramm geladen wird. Die PSP-Adresse des Programmes HWINFO gibt somit auch die Anfangsadresse des Anwenderprogrammes an. Aus der Speichergröße und dieser Adresse läßt sich dann der belegte und freie Speicher berechnen. Es wird aber vorausgesetzt, daß sich im oberen Speicherbereich keine residenten Programme befinden.

Als nächstes ist die Zahl der Diskettenlaufwerke, der seriellen und parallelen Schnittstellen und der Modus des Bildschirmadapters zu ermitteln. Alle diese Daten werden vom BIOS direkt verwaltet und lassen sich durch den INT 11 abfragen. Bild 6.9 zeigt die Belegung des in Register AX zurückgegebenen Konfigurationswortes.




```
Bit 0: 1 = Disk Drive vorhanden
Bit 1: --
Bit 2-3:     Speicherbänke auf der Hauptplatine
             00 = 1, 01 = 2, 10 = 3,  11 = 4
Bit 4-5:     default Screen Mode
             00 illegal
             01 Color Card 40x25 monochr.
             10 Color Card 80x25 monochr.
             11 Monochrom Card 80x25
Bit 6-7:     falls Bit 1 = 1 => Zahl d.Floppy Drives
             00 = 1, 01 = 2, 10 = 3, 11 = 4
Bit 8:       ---
Bit 9-11:    Zahl der seriellen Schnittstellen
Bit 12:      Spieleadapter vorhanden
Bit 13:      ---
Bit 14-15:   Zahl der parallelen Schnittstellen.
```

Bild 6.9: Codierung des Register AX beim INT 11 (Hardware Test)

Leider wird die Zahl der Plattenlaufwerke nicht mit angegeben. Dies ist historisch begründet, da in den Anfängen von MS-DOS Plattenlaufwerke zu teuer waren, um in Personalcomputern eingesetzt zu werden. Der INT-21-Aufruf 1CH (get drive data) eignet sich aber zur Ermittlung der Daten von Festplattenlaufwerken. Mit Hilfe dieses DOS-INT-21 spricht das Modul

verschiedene Laufwerke an. Existiert ein Laufwerk, meldet DOS dies über das AX-Register. Bei Floppylaufwerken mißlingt der Versuch, falls sich keine Diskette im Laufwerk befindet. Auf dem Bildschirm erscheint die Fehlermeldung:

```
Laufwerk nicht bereit ..
Wiederholen, Ignorieren, Abbrechen ..
```

Da aber die Zahl der Floppy-Laufwerken vom BIOS-Aufruf bekannt ist, sind nur noch die Kennbuchstaben (C:, D:) potentieller Festplatten abzufragen. Die Kapazität der Platte läßt sich anschließend durch den INT-21-Aufruf 36H ermitteln.

Die Ausgabe der BIOS-Signatur setzt voraus, daß diese ab F800H abgelegt ist. Einige Hersteller verwenden allerdings modifizierte BIOS-Bausteine, die die Signatur an anderer Stelle enthalten.

Die Implementierung

Nach diesen Vorüberlegungen kann die Umsetzung in ein Programm erfolgen. Dieses gliedert sich wieder in mehrere Teile, um einen transparenten Aufbau zu erlangen. Auf ein Hierarchiediagramm wird an dieser Stelle verzichtet, da der Aufbau recht einfach ist.

Nachfolgend werden die Funktionen der einzelnen Module beschrieben. Das Hauptprogramm ist recht kompakt, da es im wesentlichen die Unterprogrammaufrufe zur Ermittlung und Ausgabe der Konfiguration enthält.

fehler

Dieses Modul reagiert auf PowerBASIC-Laufzeitfehler. In diesen Fällen wird die Fehlernummer ausgegeben und HWINFO bricht ab.

bios

Es wird der Text ab Adresse F800 ausgeben. Bei vielen Rechnern findet sich hier die BIOS-Signatur.

dosvers

Die Versionsnummer der aktuellen DOS-Version wird per INT 21, Funktion 30H abgefragt. Ist der Wert in AL = 0, dann liegt eine DOS-1.x-Version vor. Ab DOS 2.0 enthält das Register AL die Hauptversion, während AH die Unterversionsnummer enthält.

memory

Das Programm ermittelt zuerst die RAM-Größe über den BIOS-INT 12. Dieser gibt das Ergebnis im Register AX in Kbyte zurück. Die Lage des Bildschirmspeichers wird durch Zugriffe auf die Segmente A000H, B000H und B800H ermittelt. Finden sich hier Werte ungleich FFH, wird ein RAM-Bereich gemeldet. Dies ist zwar nicht ganz korrekt, funktioniert aber in der Regel. Dann ist der Bereich ab C000:0H in Schritten zu 32 Kbyte auf

ROMs abzufragen. Wird ab Offset 0 der Wert 55AAH gefunden, liegt ein BIOS-Erweiterungs-ROM vor. Die Länge steht dann ab Offset 3H. Bei anderen ROM-Bausteinen findet sich auf der betreffenden Offsetadresse meist ein Wert ungleich FFH, was zur Erkennung genügt.

freemem

Das Modul errechnet den freien und belegten DOS-Speicher aus der Programmstartadresse und der Speichergröße. Die betreffenden Module sind daher zuerst aufzurufen.

progradr

Das Programm ermittelt die Anfangsadresse des ersten Anwenderprogrammes über die INT-21-Funktion 51H (get PSP). Nach dem Aufruf findet sich die PSP-Adresse im Register BX. Diese ist identisch mit der Startadresse des ersten Anwenderprogrammes.

config

Hier erfolgt die Abfrage des BIOS-INT 11, der Informationen über die Hardwarekonfiguration liefert. Ist Bit 1 gesetzt, zeigt *config* über die Zahl der registrierten Laufwerke die logischen Laufwerksnamen an.

Dann ermittelt das Programm per INT-21-Funktion 3600H, ob Hard- oder RAM-Disks vorhanden sind. Die Nummer der interessierenden Einheit ist im Register DX zu übergeben (1 = A:, 2 = B: etc.). Die Zahl der möglichen Laufwerke wird per Software auf maximal drei begrenzt. Der INT-21-Aufruf gibt auch die Informationen über die Aufteilung des Mediums wieder. Falls AX = 0FFFFH gesetzt ist, existiert das Laufwerk nicht. Andernfalls gilt:

```
AX = 0FFFF -> Die Laufwerksnummer ist falsch
     sonst -> Zahl der Sektoren pro Cluster
BX = Zahl der freien Cluster
CX = Byte pro Sektor
DX = Cluster pro Platte
```

Mit diesen Informationen läßt sich der freie Speicherplatz in Byte und in Prozent bestimmen.

Zum Abschluß wird die Zahl der parallelen und seriellen Schnittstellen ermittelt und ausgegeben. Diese befinden sich ebenfalls im Konfigurationswort, welches durch den INT 11 ermittelt wurde.

Einzelheiten sind dem nachfolgenden Listing zu entnehmen.

Verbesserungsvorschläge

Insbesondere die Bestimmung der RAM- und ROM-Bereiche läßt sich durch einen direkten Speichertest erheblich verbessern. Auch die Anzeige der BIOS-Signatur ist noch verbesserungsfähig.

```
X R E F   /Z=55                          (c) Born Version 1.0
```

```
Datei : hwinfo.bas      Datum : 07-22-1992         Seite :  1

 Zeile     Anweisung

      '*********************************************************
      '! File       : HWINFO.BAS
      '! Vers.      : 1.0
      '! Last Edit  : 16.5.92
      '! Autor      : G. Born
      '! Files      : INPUT, OUTPUT
      '! Progr. Spr.: PowerBasic
      '! Betr. Sys. : DOS 2.1 - 5.0
      '! Funktion: Das Programm wird mit der Eingabe:
      '!
      '!           HWINFO
      '!
      '!           aufgerufen. Es ermittelt die Konfiguration
      '!           des PCs und gibt das Ergebnis auf dem
      '!           Bildschirm aus.
      '*********************************************************
      '! Variable definieren
    1 frei& = 0                              '! freier Speicher
    2 start& = 0                             '! Startadresse

      '############################################################
      '#                      Hauptprogramm                       #
      '############################################################

    3 ON ERROR GOTO fehler

    4 PRINT
    5 PRINT "S y s t e m   I N F O             (c) Born Version
1.0"
    6 PRINT

    7 CALL bios                              '! Bios Copyright
    8 CALL dosvers                           '! DOS Versionsnummer
    9 CALL memory                            '! Speicherbelegung
   10 CALL progadr                           '! Adr. Userprogramme
   11 CALL freemem                           '! freier Speicher

   12 PRINT "Weiter bitte eine Taste betätigen"
   13 DO WHILE INKEY$ = "": WEND

   14 CALL config                            '!
Systemkonfigurierung

   15 PRINT
   16 PRINT "Ende System Info"
   17 END                                    '! Ende

      '############################################################
      '#                     Hilfsroutinen                        #
      '############################################################
```

```
18 fehler:
   '-------------------------------------------------------
   '! Fehlerbehandlung in HWINFO
   '-------------------------------------------------------

19  PRINT "Fehler : ";ERR;" unbekannt"
20  PRINT "Programmabbruch"
21 END                                  '! MSDOS Exit

22 SUB bios
   '!-------------------------------------------------------
   '! lese Bios Signatur
   '!-------------------------------------------------------
23 LOCAL i%

24 DEF SEG = &HF800                     '! Segm. Bios ROM

25 PRINT
26 PRINT "Bios Version ";
27 FOR i% = 3 TO 50
28  PRINT CHR$(PEEK (i%));
29 NEXT i%
30 PRINT

31 END SUB

32 SUB dosvers
   '!-------------------------------------------------------
   '! ermittle die DOS Version über die INT 21 Funktion 30
   '! CALL: AH = 30   RETURN: AL = Untervers. AH = Hauptvers.
   '!-------------------------------------------------------

33  LOCAL AH%, AL%

34  REG 1, &H3000                         '! AX = 3000 -> read
                                          '! Version
35  CALL INTERRUPT &H21                   '! Dispatcher INT

36  AH% = (REG (1) / 256) AND &HFF        '! lese Wert in AH
37  AL% = REG (1) AND &HFF                '! lese Wert in AL
38  PRINT
39  IF AL% = 0 THEN                       '! AL = leer ?
40   PRINT "MS-DOS Version 1.x"
41  ELSE
42   PRINT "MS-DOS Version ";             '! Versionsnummer
43   PRINT USING "##&"; AL%; ".";         '! AL = Hauptversion
44   PRINT USING "##"; AH%                '! AH = Unterversion
45  END IF
46 END SUB

47 SUB memory
   '!-------------------------------------------------------
   '! ermittle die RAM und ROM Speichergröße
   '!-------------------------------------------------------
48 LOCAL i&
```

```
49 PRINT

   '!
   '! ermittle die RAM Größe über den BIOS INT 12
   '! Ergebnis in AX in Kbyte
   '!

50  CALL INTERRUPT &H12                    '! BIOS: GET RAM SIZE
51  PRINT "RAM from  0000:0000 len ";
52  PRINT REG (1);" Kbytes"                '! AX = Größe


   '!
   '! Prüfe die Lage des Bildschirmadapters auf Segment
   '! A000H, B000H, B800H
   '!
53  DEF SEG = &HA000                       '! setze Segment
54  IF (PEEK(0) <> &HFF) AND _             '! Bildschirm RAM ?
55      (PEEK(1) <> &HFF) THEN
56    PRINT "RAM from  A000:0000"          '! Segmentadr.
57  END IF

58  DEF SEG = &HB000                       '! setze Segment
59  IF (PEEK(0) <> &HFF) AND _             '! Bildschirm RAM ?
60      (PEEK(1) <> &HFF) THEN
61    PRINT "RAM from  B000:0000"          '! Segmentadr.
62  END IF


63  DEF SEG = &HB800                       '! setze Segment
64  IF (PEEK(0) <> &HFF) AND _             '! Bildschirm RAM ?
65      (PEEK(1) <> &HFF) THEN
66    PRINT "RAM from  B800:0000"          '! Segmentadr.
67  END IF

   '!
   '! Prüfe ab C000:0 in 32 Kbyte Schritten auf ROMs
   '!

68  FOR i& = &HC000 TO &HFFFF STEP &H800   '! teste ROM
69   DEF SEG = i&                          '! setze Segment
70   IF (PEEK(0) = &H55) AND _             '! ROM Signatur ?
71      (PEEK(1) = &HAA) THEN
72    PRINT "ROM from  "; HEX$(i&);":0000";'! Segmentadr.
73    PRINT " len "; PEEK(2);              '! Kbytes
74    PRINT " Kbyte"
75   ELSEIF (PEEK(0) <> &HFF) AND _        '! ROM Signatur ?
76      (PEEK(1) <> &HFF) THEN
77    PRINT "ROM from  "; HEX$(i&);":0000" '! Segmentadr.
78   END IF
79  NEXT i&

80 END SUB
```

```
   81 SUB freemem
      '!-------------------------------------------------------
      '! ermittle den freien Speicher
      '!-------------------------------------------------------
   82 SHARED start&
   83 LOCAL frei&, ram&

   84  CALL INTERRUPT &H12                    '! BIOS: GET RAM SIZE

   85  ram& = REG(1)                          '! RAM Größe
   86  ram& = ram& * &H3FF
   87  frei& = ram& - (start& * 16)           '! berechne. freie
Größe

   88  PRINT
   89  PRINT "MS-DOS Free Memory  : ";
   90  PRINT frei&;" Kbyte"                   '! freier Speicher
   91  PRINT "MS-DOS Used Memory  : ";
   92  PRINT start& * 16;" Kbyte"            '! belegter Speicher

   93 END SUB

   94 SUB progadr
      '!-------------------------------------------------------
      '! Ermittle die Anfangsadresse des Anwenderprogrammes
      '! über die undokumentierte Funktion 51H des INT 21.
      '! Diese gibt im Register BX die Segmentadresse des
      '! aktuellen PSP an. Dies ist gleichzeitig die Start-
      '! adresse der Anwenderprogramme.
      '!-------------------------------------------------------
   95  LOCAL res$
   96  SHARED start&

   97  REG 1, &H5100                          '! AX = 5100 -> read
                                              '! PSP Segm. Adr
   98  CALL INTERRUPT &H21                    '! Dispatcher INT

   99  IF (REG (0) AND &H01) > 0 THEN         '! Fehler ?
  100   PRINT "Fehler beim Systemaufruf "; REG (1) '! AX =
Fehlercode
  101  ELSE
  102   PRINT "User Progr. Adresse : ";
  103   start& = REG(2)                       '! merke Adresse
  104   res$ = HEX$(REG(2))                   '! Segm. Adr in BX
  105   res$ = STRING$(4-LEN(res$),"0") + res$ '! führende Nullen
  106   PRINT res$;":0000"
  107  END IF

  108 END SUB

  109 SUB config
      '!-------------------------------------------------------
      '! lese die Konfiguration der Hardware über den INT 11
      '!-------------------------------------------------------
  110 LOCAL ax%, drive%, bytes&, frei&, disp%, i% , tmp&
```

```
  111 DIM a%(2)                              '! Hilfsvariable für
MOVE

  112  CALL INTERRUPT &H11                   '! Lese Konfiguration

  113  ax% = REG(1)                          '! Wert in ax%
  114  PRINT
  115  IF (ax% AND 1) = 0 THEN               '! Floppys ?
  116   PRINT "keine Diskettenlaufwerke gefunden "
  117  ELSE
  118   drive% = (ax% AND &HC0) / 64         '! Floppys -> b14, b15
  119   PRINT "Disks"; SPACE$(15);
  120   FOR i% = 0 TO drive%                 '! n Drives
  121    PRINT CHR$(&H41 + i%);": ";         '! Kennbuchstabe A: ..
F:
  122   NEXT i%
  123  END IF
  124  PRINT
  125  PRINT

      '!
      '! ist eine Festplatte im System integriert ?
      '! Platten befinden sich oberhalb der Floppys (C:, D:, etc.).
      '! Prüfe per INT 21, Funktion 1CH ob Platten vorhanden sind.
      '!
  126  IF drive% < 2 THEN                    '! weniger als 1
Floppy?
  127   drive% = 3                           '! Platte ab c:
  128  ELSE
  129   INCR drive%                          '! oberhalb Floppy
  130  END IF

  131  FOR i% = drive% to drive% + 2         '! max. 3 Disks
  132   REG 1, &H3600                        '! Funktion 3600
  133   REG 4, i% AND &HFF                   '! Laufwerkscode
  134   CALL INTERRUPT &H21                  '! INT 21
  135   IF (REG(1) AND &HFF) <> &HFF THEN    '! Platte gefunden ?
  136    PRINT "Disk "; SPACE$(15); CHR$(&H40+i%);": ";
  137    tmp% = REG(4)                       '! Cluster

      '!
      '! Achtung: wegen I*2 ist eine Typkonvertierung erforderlich
!!!!
      '!
  138    a%(1) = VARPTR(tmp%)                '! Adr. Variable
übergeben!!!
  139    a%(2) = VARSEG(tmp%)
  140    clust& = 0
  141    CALL MOVE (2,clust&,a%(1))

  142    bytes& = clust& * REG(3) * REG(1)
  143    PRINT bytes&;                       '! Speichergröße
  144    PRINT " Bytes"
  145    PRINT "Sektoren pro Cluster : "; REG(1)
  146    PRINT "Bytes pro Sektor     : "; REG(3)
```

```
  147    PRINT "Cluster              : "; clust&

  148    tmp% = REG(2)                       '! freie Cluster
      '!
      '! Achtung: wegen I*2 ist eine Typkonvertierung erforderlich
!!!!
      '!
  149    a%(1) = VARPTR(tmp%)                '! Adr. Variable
übergeben!!!
  150    a%(2) = VARSEG(tmp%)
  151    fclust& = 0
  152    CALL MOVE (2,fclust&,a%(1))

  153    frei& = fclust& * REG(3) * REG(1)
  154    PRINT "Freie Kapazität      : "; frei&;    '! in Bytes
  155    PRINT USING "Bytes (##.#";(frei& * 100.0 / bytes&);
  156    PRINT " %)"                         '! in %
  157   END IF
  158  NEXT i%

      '!
      '! Anzeige weiterer Informationen aus dem BIOS Aufruf
      '!

  159  PRINT
  160   disp% = (ax% AND &H30) / 16          '! Bildschirmadapter
  161   SELECT CASE disp%
  162    CASE 0
  163     PRINT "Illegal Display Mode"
  164    CASE 1
  165     PRINT "Colorcard 40 x 25 Monochrom"
  166    CASE 2
  167     PRINT "Colorcard 80 x 25 Monochrom"
  168    CASE 3
  169     PRINT "Monochromcard 80 x 25"
  170   END SELECT

  171  PRINT
  172  PRINT "Serial Interfaces   : "; (ax% AND &H0E00) / 512
  173  tmp& = (ax% AND &HC000) / &HFF / &H2F
  174  PRINT "Parallel Interfaces : "; tmp&

  175 END SUB

  176 SUB MOVE INLINE

  177 $INLINE "MOVE.COM"

  178 END SUB

      '***** Programm Ende ******
```

Listing 6.15: HWINFO.BAS

Anhang A: ASCII-Tabellen

Hex	Dez	ASCII
00	0	NUL
01	1	SOH
02	2	STX
03	3	ETX
04	4	EOT
05	5	ENQ
06	6	ACK
07	7	BEL
08	8	BS
09	9	HT
0A	10	LF
0B	11	VT
0C	12	FF
0D	13	CR
0E	14	SO
0F	15	SI
10	16	DLE
11	17	DC1
12	18	DC2
13	19	DC3
14	20	DC4
15	21	NAK
16	22	SYN
17	23	ETB
18	24	CAN
19	25	EM
1A	26	SUB
1B	27	ESC
1C	28	FS
1D	29	GS
1E	30	RS
1F	31	US
20	32	SPACE
21	33	!
22	34	"
23	35	#
24	36	$
25	37	%
26	38	&
27	39	'

28	40	(
29	41	)
2A	42	*
2B	43	+
2C	44	,
2D	45	-
2E	46	.
2F	47	/
30	48	0
31	49	1
32	50	2
33	51	3
34	52	4
35	53	5
36	54	6
37	55	7
38	56	8
39	57	9
3A	58	:
3B	59	;
3C	60	<
3D	61	=
3E	62	>
3F	63	?
40	64	@
41	65	A
42	66	B
43	67	C
44	68	D
45	69	E
46	70	F
47	71	G
48	72	H
49	73	I
4A	74	J
4B	75	K
4C	76	L
4D	77	M
4E	78	N
4F	79	O
50	80	P
51	81	Q
52	82	R
53	83	S
54	84	T
55	85	U
56	86	V
57	87	W

58	88	X
59	89	Y
5A	90	Z
5B	91	[
5C	92	\
5D	93	]
5E	94	^
5F	95	_
60	96	`
61	97	a
62	98	b
63	99	c
64	100	d
65	101	e
66	102	f
67	103	g
68	104	h
69	105	i
6A	106	j
6B	107	k
6C	108	l
6D	109	m
6E	110	n
6F	111	o
70	112	p
71	113	q
72	114	r
73	115	s
74	116	t
75	117	u
76	118	v
77	119	w
78	120	x
79	121	y
7A	122	z
7B	123	{
7C	124	|
7D	125	}
7E	126	~
7F	127	•
80	128	Ç
81	129	ü
82	130	é
83	131	â
84	132	ä
85	133	à
86	134	å
87	135	ç

88	136	ê
89	137	ë
8A	138	è
8B	139	ï
8C	140	î
8D	141	ì
8E	142	Ä
8F	143	Å
90	144	É
91	145	æ
92	146	Æ
93	147	ô
94	148	ö
95	149	ò
96	150	û
97	151	ù
98	152	ÿ
99	153	Ö
9A	154	Ü
9B	155	ø
9C	156	£
9D	157	Ø
9E	158	×
9F	159	ƒ
A0	160	á
A1	161	í
A2	162	ó
A3	163	ú
A4	164	ñ
A5	165	Ñ
A6	166	ª
A7	167	º
A8	168	¿
A9	169	®
AA	170	¬
AB	171	½
AC	172	¼
AD	173	¡
AE	174	«
AF	175	«
B0	176	°
B1	177	±
B2	178	²
B3	179	
B4	180	´
B5	181	Á
B6	182	Â
B7	183	À

B8	184	©
B9	185	¹
BA	186	º
BB	187	»
BC	188	¼
BD	189	¢
BE	190	¥
BF	191	¿
C0	192	À
C1	193	Á
C2	194	Â
C3	195	Ã
C4	196	-
C5	197	Å
C6	198	ã
C7	199	Ã
C8	200	È
C9	201	É
CA	202	Ê
CB	203	Ë
CC	204	Ì
CD	205	Í
CE	206	Î
CF	207	¤
D0	208	ð
D1	209	Ð
D2	210	Ê
D3	211	Ë
D4	212	È
D5	213	_
D6	214	Í
D7	215	Î
D8	216	Ï
D9	217	Ù
DA	218	Ú
DB	219	Û
DC	220	Ü
DD	221	¦
DE	222	Ì
DF	223	ß
E0	224	Ó
E1	225	ß
E2	226	Ô
E3	227	Ò
E4	228	õ
E5	229	Õ
E6	230	µ
E7	231	Þ

E8	232	þ
E9	233	Ú
EA	234	Û
EB	235	Ù
EC	236	ý
ED	237	Ý
EE	238	¯
EF	239	´
F0	240	-
F1	241	±
F2	242	=
F3	243	¾
F4	244	¶
F5	245	§
F6	246	÷
F7	247	¸
F8	248	°
F9	249	¨
FA	250	·
FB	251	¹
FC	252	
FD	253	²
FE	254	_
FF	255	

Anhang B: Literaturhinweise

/1/ Born, Günter: Das DOS 5 Programmierhandbuch, Markt&Technik Verlag, München, 1992, 862 Seiten,
ISBN 3-87791-316-4

/2/ Schäpers, Arne: DOS 5 für Programmierer, Addison-Wesley Verlag, Bonn, 1991, 1123 Seiten,
ISBN 3-89319-350-2

/3/ Adobe: PostScript Language Reference Manual, Addison-Wesley Verlag, Bonn, 1991, 764 Seiten,
ISBN 0-201-18127-4

/4/ Born, Günter: PostScript enträtselt, Systhema Verlag, München, 1991, 272 Seiten,
ISBN 3-89390-363-1

/5/ Born, Günter: DOS 5 Tuning, Markt&Technik Verlag, München, 1991, 388 Seiten,
ISBN N3-87791-196-X

/6/ Born, Günter: Systemtuning mit TSR-Programmen, Addison-Wesley Verlag, Bonn, 1990, 252 Seiten, ISBN 3-89319-262-X

/7/ Born, Günter: Referenzhandbuch Dateiformate, Addison-Wesley Verlag, Bonn, 1992, 830 Seiten, ISBN 3-89319-446-0

/8/ Born, Günter: Assembler enträtselt, Systhema Verlag, München, 1991, 269 Seiten,
ISBN 3-89390-378-X

/9/ Born, Günter: A86/D86, Assemblerprogrammierung für Einsteiger, Systhema Verlag, München, 1991, 275 Seiten,
ISBN 3-89390-111-6

Stichwortverzeichnis