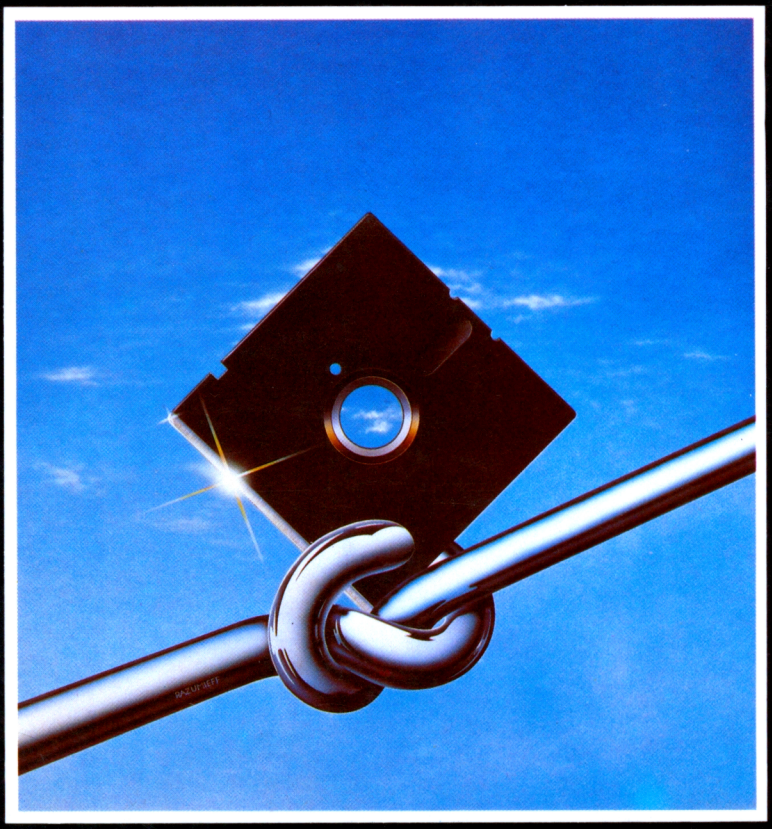
SYBEX

Das dBASE II Profibuch



Alan Simpson

Das dBASE II Profibuch

Das dBASE II Profibuch

Alan Simpson

DÜSSELDORF · BERKELEY · PARIS

Anmerkung:

CP/M ist ein Warenzeichen von Digital Research Inc.
dBASE II ist ein Warenzeichen von Ashton-Tate
IBM Personal Computer ist ein eingetragenes Warenzeichen von International Business Machines Corporation
WordStar und MailMerge sind Warenzeichen von MicroPro International

Originalausgabe in Englisch
Titel der amerikanischen Ausgabe: Advanced Techniques in dBASE II
Original Copyright © 1985 by SYBEX Inc., Berkeley, California, USA

Deutsche Übersetzung: Elmar A. Weiler

Umschlagentwurf: Nicolae Razumieff/tgr
Satz: tgr – typo-grafik-repro gmbh, Remscheid
Gesamtherstellung: Clausen & Bosse, Leck

Der Verlag hat alle Sorgfalt walten lassen, um vollständige und akkurate Informationen zu publizieren. SYBEX-Verlag GmbH, Düsseldorf, übernimmt keine Verantwortung für die Nutzung dieser Informationen, auch nicht für die Verletzung von Patent- und anderen Rechten Dritter, die daraus resultieren.

ISBN 3-88745-610-6
1. Auflage 1986

Alle Rechte vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder einem anderen Verfahren) ohne schriftliche Genehmigung des Verlages reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Printed in Germany
Copyright © 1986 by SYBEX-Verlag GmbH, Düsseldorf

Inhaltsverzeichnis

Einführung 11

Warum in dBASE II programmieren? 13
Die Struktur dieses Buches 14
Welche Version sollten Sie benutzen 14
Vereinbarungen zur Schreibweise 14

1 Befehlsdateien erstellen 15
Befehlsdateien mit dem MODIFY COMMAND schreiben . . . 15
Befehlsdateien mit WordStar schreiben 17
Strukturiertes Programmieren 19
Zusammenfassung 23

2 Optimierte Leistung 25
Minuten werden zu Sekunden mit FIND und WHILE 25
Schneller rechnen 28
Schnellere Berichte 29
Schneller kopieren 29
Schneller editieren 29
Schneller sortieren 31
Die Leistungssteigerung schätzen 32
Mit mehreren Index-Dateien arbeiten 32

Nachteile der Leistungssteigerung 33
Auf Leistung programmieren 35
Zusammenfassung 37

3 Programmentwurf und Programmentwicklung 39
Software-Entwurf und Entwicklung 40
Das Ziel des Projekts bestimmen 40
Die Datenbankstruktur erstellen 42
Die Programmstruktur erstellen 44
Die einzelnen Befehlsdateien entwerfen und entwickeln 45
Zusammenfassung 51

4 Benutzerfreundliche Dateneingabemasken 53
Formatierte Masken erstellen 54
Arbeiten mit ZIP 55
FMT-Dateien benutzen 58
Eingabeformate festlegen 60
Die Programmdatei für die Eingabemaske 61
Zusammenfassung 62

5 Menügeführtes Suchen und Sortieren 63
Das REPORTS-Programm 63
Menügesteuertes Sortieren 68
Menügesteuertes Suchen 69
Zusammenfassung 76

6 Berichte 77
Bildschirm- und Druckformatierung 77
Mehrfache Etikettenkopien mit Justierung 86
Eine MailMerge-Datei erstellen 94
Zusammenfassung 99

7 Programme zum Ändern und Löschen 101
Automatisches Ändern der Datenbank 101
Datensätze aus einer Datenbank löschen 109
Zusammenfassung 116

8 Zusätzliche Optionen 119
Schnellsuche nach einem einzelnen Eintrag 119
Duplikate schnell finden 127
Zusammenfassung 134

9 Ein Inventarsystem 135
Die Ziele des Inventarsystems 137
Die Datenbanken des Inventarsystems 138
Gestaltung der Inventarprogramme 140
Das Inventar-Hauptmenü 141
Zusammenfassung 145

10 Die Haupt-Inventar-Datei 147
Die Gestaltung der Hauptdatei 148
Die Hauptdatei 148
Neue Teilenummer eingeben 149
Hauptdatei-Reports 155
Bericht über den aktuellen Lagerbestand 157
Bericht über Nachbestellungen 158
Bericht der bestellten Teile 159
Teile bestellen 160
Die Report-Optionen auf den Bildschirm bringen 161
Änderungen in der Hauptdatei 173
Zusammenfassung 177

11 Inventar – Verkaufssystem 179
Die Struktur des Verkaufssystems 180
Das Menü des Verkaufssystems 180
Eingabe der Verkaufsdaten 181
Verkaufsberichte 190
Eine Bemerkung zum Datum 202
Zusammenfassung 204

12 Inventar – Wareneingangsystem 205
Wareneingangsystem Programmstruktur 206
Menü Wareneingang 206
Das Eingabeprogramm 206
Wareneingang-Berichte 211
Zusammenfassung 220

13 Aktualisierung des Inventarsystems 221
Aktualisierung der Hauptdatei 221
Ändern in der Datei Verkauf 229
Änderungen in der Datei Wareneingang 238
Zusammenfassung 246

14 Ein Buchhaltungssystem 249
Die Ziele des Buchhaltungssystems 249
Entwurf der Datenbanken des Buchhaltungssystems 250
Die Programmstruktur des Buchhaltungssystems 254
Zusammenfassung 256

15 Installations- und Menüprogramm für das Buchhaltungssystem . 257
Buchhaltungs-Installationsprogramm 258
Buchhaltungssystem Hauptmenü 268
Zusammenfassung 270

16 Vorgänge eingeben und aktualisieren 271
Buchhaltungsvorgänge speichern 271
Die Kontenaufstellung aktualisieren 282
Zusammenfassung 290

17 Das Scheckregister bearbeiten 291
Einnahmen speichern, Bilanzen errechnen 293
Das Scheckregister drucken 297
Zusammenfassung 304

18 Die Berichte des Buchhaltungssystems 305
Buchhaltungssystem Berichtsmenü 305
Einnahmen-/Ausgaben-Übersicht 306
Bericht Kontenübersicht 315
Schecks schreiben 327
Zusammenfassung 340

19 Daten der Buchhaltung ändern 341
Die Änderungsmasken der Buchhaltung 343
Buchhaltungdaten eingeben, ändern oder löschen 346
Berichtigungsbuchungen 357
Erweiterungsmöglichkeiten der Buchhaltung 359
Zusammenfassung 360

20 Ein Literatur-Verwaltungs-Programm 363
Die Vorteile der Literatur-Verwaltung 363
Die Struktur der Datenbank 364
Die Dateien des Literatur-Verwaltungs-Systems 365
Das Hauptmenü der Literatur-Verwaltung 366

Neue Einträge aufnehmen 368
Einträge ändern 368
Einträge löschen 376
Zusammenfassung 378

21 Literaturverwaltungs-Berichte 381
Nach Referenzen suchen 381
Unterstreichungen 409
Stichwörterbericht der Literaturverwaltung 413
Zusammenfassung 422

Stichwortverzeichnis 425

Einführung

Wenn Sie eine großartige Idee für ein Computerprogramm im Kopf haben und diese mit dBASE II in die Realität umsetzen wollen, dann haben Sie jetzt genau das richtige Buch in Händen. Auf den hier folgenden Seiten werden Sie lernen, wie man ausgefallene Ideen in dBASE II-Programme verwandelt, die genau das erfüllen, was Sie von ihnen erwarten: hohe Verarbeitungsgeschwindigkeit und Effektivität.

Es ist nicht unbedingt notwendig, daß Sie bereits jetzt schon jeden dBASE-Befehl haargenau kennen und verstehen. Einige der komplexeren Befehle werden in diesem Buch ausführlich diskutiert – speziell die, die im dBASE-Handbuch vernachlässigt wurden. Wenn Ihnen in einem der Programme ein unbekannter Befehl begegnet, sehen Sie dazu dann noch einmal ins Handbuch.

Ein dBASE-Befehl ist immer das erste Wort in einer Programmzeile. Wenn Sie zum Beispiel diese Zeile sehen:

```
UPDATE ON code FROM trans ADD betrag REPLACE datum
```

können Sie unter der Rubrik zum Befehl UPDATE im dBASE II-Handbuch weitere Informationen nachschlagen. Das erste Wort in einem dBASE-Satz ist der Hauptbefehl für den gesamten Satz.

Auf eine dBASE II-Funktion folgt meistens – in Klammern gesetzt – ein Feldname oder eine Speichervariable. In dieser Zeile zum Beispiel:

```
STORE STR(x,5) TO y
```

ist STORE der Befehl, und STR ist die Funktion, die aus der Variablen X eine „Zeichenkette" macht (in unserem Beispiel eine Zeichenkette mit fünf Zeichen). Wenn Ihnen diese oder andere Funktionen unbekannt sind, nehmen Sie sich die Zeit, noch einmal in das dBASE-Handbuch zu sehen. Sie sparen dadurch auf die Dauer viel Zeit und verstehen die hier gezeigten Programme besser.

Denken Sie auch daran, daß dieses Buch als Lehrbuch geschrieben wurde und als solches verstanden und gelesen werden sollte. Es beginnt mit recht einfachen Programmen und baut auf diesen immer komplexer werdende auf. Wenn Sie also einfach die ersten neun Kapitel überschlagen, werden Sie im zehnten Kapitel ganz schön staunen. Sie sollten sich die Zeit nehmen, das Buch Seite für Seite zu lesen.

Um ein anwenderfreundliches Programmsystem in dBASE II zu schreiben, müssen Sie mehreren Anforderungen genügen. Zuerst einmal müssen Sie die dBASE II-Sprache kennen. Mein Einführungsbuch „Arbeiten mit dBASE II" ist geeignet, Ihnen die grundlegenden Befehle von dBASE II beizubringen. Zweitens müssen Sie wissen, wie man programmiert. Das Ziel dieses Buches ist, Ihnen zu zeigen, wie man in dBASE II programmiert. Wir werden dieses Ziel erreichen, indem wir hauptsächlich drei verschiedene Techniken anwenden:

1. *Programmgestaltungs-Techniken:* Ein Programmsystem zu gestalten, erfordert mehrere Einzelschritte: Das Niederschreiben der Grundziele des Projekts, die Entscheidung, was in der Datenbank oder mehreren Datenbanken gespeichert werden soll, und das Zerlegen der großen Aufgabe in viele kleine, die dann recht leicht zu programmieren sind. Dann beginnt das wirkliche Programmieren. Sie werden eine Reihe von kleinen, einfachen Routinen schreiben, die nach der Zusammenfassung zu einem Gesamtsystem so zusammenarbeiten, daß Sie Ihr Ziel erreichen. In diesem Buch folgen wir den eben vorgestellten Schritten, wenn wir Programme schreiben.

2. *Techniken zur Leistungssteigerung:* Ein Programm, das eine bestimmte Aufgabe in einer Stunde erledigt, kann ein gutes Programm sein. Ein Programm, das dieselbe Aufgabe in fünf Minuten schafft, ist vorzuziehen. Die Leistung eines Programmsystems zu optimieren, ist ein Grundanliegen in diesem Buch. In Kapitel 2 finden Sie viele Techniken besprochen, die für mehr Leistung in dBASE-Programmen sorgen.

3. *Fundamentale Algorithmen:* Eine weitere Programmiertechnik besteht darin, grundlegende Algorithmen (Routinen, die Standardprobleme lösen) zu erstellen und sie später in anderen Programmen und Programmsituationen wieder zu verwenden. Beispielsweise fin-

den Sie im Kapitel 18 ein Programm, das Schecks für eine Buchhaltung schreibt. Wenn Sie sich also daran machen, ein Buchhaltungsprogramm zu schreiben, können Sie einfach dieses Programm verwenden, statt ein anderes von Grund auf zu erstellen. Die grundsätzliche Logik, der Algorithmus zum Schreiben von Schecks, ist bei allen Systemen immer gleich.

Warum in dBASE II programmieren?

dBASE II besitzt so viele Befehle, daß Sie viele Aufgaben ohne großen Programmieraufwand lösen können. Aber es gibt schließlich drei gute Gründe, warum Sie sich entschließen sollten, in dBASE zu programmieren:

1. *Die Möglichkeiten von dBASE erweitern:* Obwohl Ihnen dBASE viele Befehle anbietet, die Sie direkt am Bildschirm eintippen können, reichen diese längst noch nicht aus. Wenn Sie beispielsweise Briefetiketten mit den Adressen aus Ihrer Datenbank drucken wollen, gibt es keinen Befehl dafür. Sie müssen ein dBASE-Programm schreiben, um diese Aufgabe zu bewältigen. Sie sehen selbst: Um möglichst viel Leistung aus dBASE herauszuholen, müssen Sie Programme schreiben.
2. *Anwenderfreundliche Systeme entwickeln:* Bis jetzt haben Sie möglicherweise gelernt, wie Datenbanken eingerichtet und Daten bewegt werden können. Zweifelsohne mußten Sie dafür etwas Zeit investieren. Es könnte jedoch sein, daß jemand anders mit Ihrem Programm arbeiten soll, nachdem Sie es entwickelt haben, damit Sie sich einem neuen dBASE-Projekt zuwenden können. Diese andere Person ist aber möglicherweise nicht bereit, viel Zeit zum Erlernen der dBASE-Sprache zu investieren. Dafür können Sie eine Reihe von Programmen schreiben, die es auch einem Anfänger ermöglichen, mit Daten zu arbeiten, indem ihm oder ihr verschiedene Wahlmöglichkeiten per Menü angeboten werden. Das schafft Ihnen den Freiraum, sich Ihrem neuen dBASE-Projekt zuzuwenden.
3. *Erhöhte Programmierleistung:* Wenn Sie ein professioneller Programmierer sind, werden Sie wissen, wie lange es dauert, ein benutzerfreundliches Programmsystem zu entwickeln. Jedoch – die meisten Programmsysteme verlangen immer die gleiche Arbeit: Der Anwender soll in der Lage sein, Daten in die Datenbank einzugeben, zu sortieren, zu suchen und zu verändern. Zusätzlich sollen noch Berichte erstellt und eventuell Berechnungen ausgeführt werden. In den Programmiersprachen C, PASCAL, COBOL oder FORTRAN kostet die Erfüllung dieser Anforderungen viel Zeit. dBASE kann die meisten Aufgaben mit einem einzigen Befehl erledigen, was die Programmierzeit spürbar verkürzt.

Da dBASE bereits alle Routinen zum Eingeben neuer Daten, Sortieren, Suchen, Verändern und zum Drucken von Berichten enthält, können Sie in recht kurzer Zeit ein Programm schreiben, das alle Aufgaben erfüllt. Außerdem bietet dBASE beachtliche Fehlersuch-Möglichkeiten, wodurch die Entwicklungszeit für Programme weiter verkürzt wird. Es gibt noch einen anderen Grund, warum Sie das Programmieren in dBASE erlernen sollten. Das hervorragende Design des Produkts, und das Marketinggeschick von ASHTON-TATE, ließen dBASE zum Industriestandard werden. Ob Sie nun ein gestandener Profi oder ein Neugieriger sind, der mit der Zeit gehen möchte, Sie werden zahlreiche Vorteile genießen, wenn Sie dBASE II beherrschen.

Die Struktur dieses Buches

In Kapitel 1 finden Sie grundlegende Techniken zum Erstellen von dBASE II-Programmen, darin eingeschlossen auch das Arbeiten mit dem MODIFY COMMAND, WordStar und dem Maskengenerator ZIP. Kapitel 2 bietet grundlegende Techniken zur Verbesserung der Leistungsfähigkeit Ihrer Programme.

Die Kapitel 3 bis 8 bringen eine Adressenverwaltung und zeigen grundsätzliche Programmiertechniken für Menüs, Eingabemasken und Berichte. In den Kapiteln 9 bis 13 werden ein Inventarsystem und die grundlegenden Techniken zum Arbeiten mit mehreren Datenbanken behandelt. In den Kapiteln 20 bis 21 schließlich finden Sie eine Literaturverwaltung mit fortschrittlichen Techniken zum menügesteuerten Suchen und zur Suche nach Stichwörtern und einer Methode des Zeilenumbruchs.

Welche Version sollten Sie benutzen?

Um das Leben ein wenig einfacher zu machen, wurden alle Programmsysteme in diesem Buch mit dBASE II Version 2.4 erstellt. Wenn sie immer noch eine frühere Version benutzen, sollten Sie endlich auf 2.4 umsteigen, weil dort viele Fehler der alten Version beseitigt wurden. Außerdem finden Sie in der neuen Version mehr Möglichkeiten beim Programmerstellen, was die Programmierung vereinfacht.

Vereinbarungen zur Schreibweise

Alt-H Die Tasten ALT und H gleichzeitig drücken.

^D Die Tasten CTRL und D gleichzeitig drücken.

Befehlsdateien erstellen

1

In diesem Kapitel erfahren Sie, wie man mit dem Befehl MODIFY COMMAND und dem Textprogramm WordStar Programme schreiben kann. Zudem lernen Sie auch strukturierte Programmierung kennen.

Befehlsdateien mit dem MODIFY COMMAND schreiben

Eine Befehlsdatei ist nichts anderes, als eine Aneinanderreihung von Befehlen, die dBASE II lesen und ausführen kann. Für das Schreiben solcher Dateien bietet dBASE II einen Texteditor an. Sobald dBASE II sein Bereitschaftszeichen (Prompt) zeigt – das ist der Punkt (.) in der ersten Spalte – benutzen Sie das Kommando MODIFY COMMAND (oder die Abkürzung davon: MODI COMM), um den Texteditor aufzurufen und Ihr Programm zu schreiben.

Lassen Sie uns ein kleines Programm mit dem Namen TEST auf die Diskette im Laufwerk B schreiben. Schreiben Sie:

```
MODI COMM B:TEST
```

Beachten Sie dabei: Wenn Sie bereits vorher SET DEFAULT TO B eingegeben haben, können Sie die Laufwerksbezeichnung B natürlich weglassen. dBASE wird Ihnen kurz die Meldung NEUE DATEI zeigen und dann einen leeren Bildschirm präsentieren, auf den Sie Ihr Programm schreiben dürfen.

Sie schreiben Ihr Programm, indem Sie Zeile für Zeile eingeben und am Zeilenende RETURN drücken. Versuchen Sie einmal, diesen einfachen Zweizeiler einzutippen:

```
ERASE
? "Hallo, ich bin ein Testprogramm"
```

Um dieses Meisterwerk abzuspeichern, geben Sie den Befehl ^W ein. (Sie drücken dazu die Control-Taste, halten sie gedrückt und tippen dann auf W.) dBASE wird diese Datei auf der Diskette im angemeldeten Laufwerk unter dem Namen TEST abspeichern und die Kennung CMD (oder PRG bei Computern unter dem Betriebssystem MS-DOS oder PC-DOS) automatisch anhängen. Danach sehen Sie wieder das dBASE-Prompt.

Um das Programm jetzt laufen zu lassen, geben Sie DO ein, gefolgt von dem Namen (wenn nötig mit der Laufwerksbezeichnung) der Datei. In unserem Beispiel tippen Sie ein:

```
DO B:TEST
```

und drücken dann auf RETURN. dBASE wird die beiden Befehlszeilen in dem Programm ausführen: Es wird zuerst den Bildschirm leeren und dann die Textmeldung in der zweiten Zeile bringen. Der Bildschirm sollte jetzt so aussehen:

```
Hallo, ich bin ein Testprogramm
```

Jetzt werden Sie das TEST-Programm überarbeiten. Geben Sie ein:

```
MODI COMM B:TEST
```

und drücken Sie anschließend auf RETURN. Der Bildschirm wird gelöscht, und das Listing des Programms wird angezeigt. Der Cursor steht auf dem ersten Zeichen in der ersten Zeile oben links. Das sieht so aus:

```
ERASE
? "Hallo, ich bin ein Testprogramm"
```

Von jetzt an können Sie Zeilen und Zeichen dazuschreiben, ändern und löschen, indem Sie diese Steuerzeichen benutzen:

^E Cursor eine Zeile aufwärts
^X Cursor eine Zeile abwärts
^D Cursor ein Zeichen nach rechts
^S Cursor ein Zeichen nach links
^G Zeichen rechts über dem Cursor löschen
^T Zeile löschen
^N eine neue, leere Zeile einfügen
^V Einfügen ein-/ausschalten

Bei einigen Terminals, wie zum Beispiel bei IBM, funktionieren auch die normalen Cursortasten wie gewohnt. Zusätzlich kann auch die DEL-Taste zum Löschen eines Zeichens und die INS-Taste zum Umschalten des Einfügemodus benutzt werden.

Wenn Sie beispielsweise ^D drücken (oder die Cursor-abwärts-Taste), geht der Cursor in der Datei TEST eine Zeile tiefer, wie Sie hier sehen:

```
ERASE
? "Hallo, ich bin ein Testprogramm"
```

Wenn Sie ^N eingeben, wird eine neue Zeile in die Datei eingefügt, was dann so aussieht:

```
ERASE

? "Hallo, ich bin ein Testprogramm"
```

Jetzt können Sie eine neue Zeile eingeben, wie es hier gezeigt wird:

```
ERASE
? "Dies ist eine neue Zeile"
? "Hallo, ich bin ein Testprogramm"
```

Sind Sie mit der Programmerweiterung fertig, benutzen Sie ^W, um die Änderungen abzuspeichern. Sollten Sie beim Ändern des Programms einen Fehler machen, können Sie die Eingabe mit dem Befehl ^Q beenden. Damit bleibt die Datei unverändert auf der Diskette stehen.

Nachdem Sie eine Datei geändert und mit ^W abgespeichert haben, erstellt dBASE automatisch eine Sicherheitskopie mit der Kennung BAK (für BAcKup). Erinnern Sie sich daran, daß die BAK-Datei immer den Stand vor der letzten Änderung repräsentiert.

Befehlsdateien mit WordStar schreiben

Sie können selbstverständlich auch die meisten Standard-Textprogramme zum Schreiben eines Programms benutzen. WordStar von MicroPro beispielsweise bietet Programmierern viele wertvolle Hilfsmittel dazu an. Um mit WordStar zu arbeiten, legen Sie die Diskette mit den WordStar-Dateien in das angemeldete Laufwerk (A> kennzeichnet das Laufwerk A als das angemeldete). Dann rufen Sie das Programm auf, indem Sie WS eintippen und auf RETURN drücken.

Daraufhin erscheint das Hauptmenü von WordStar auf dem Bildschirm und wartet auf eine Eingabe von Ihnen. Wählen Sie die Option N, den Programm-Modus, um eine Datei ohne irgendwelche Sonderzeichen zu

erstellen (im Text-Modus verwendet WordStar das achte Bit jedes Wortes, um Trennungen oder den Randausgleich zu vermerken). WordStar fragt dann:

```
NAME DER DATEI ZUM BEARBEITEN?
```

Geben Sie die Laufwerksbezeichnung, den Dateinamen und die Kennung ein. Möchten Sie die Datei auf dem Laufwerk B erstellen, geben Sie den Dateinamen so an: B:TEST.CMD (auf einem Computer mit CP/M-Betriebssystem). Unter MS-DOS schreiben Sie den Namen so: B:TEST.PRG, weil dBASE hier ein PRG als Kennung erwartet.

WordStar präsentiert Ihnen dann einen fast leeren Bildschirm, auf dem Sie Ihr Programm zusammenstellen können. (Am oberen Ende des Bildschirms sehen Sie kurze Informationen, wie Sie den Cursor bewegen können.) Hier sind die Grundkommandos für den Cursor:

^E Cursor eine Zeile aufwärts
^X Cursor eine Zeile abwärts
^D Cursor ein Zeichen nach rechts
^S Cursor ein Zeichen nach links
^G Zeichen rechts über dem Cursor löschen
^Y Zeile löschen
^N eine neue, leere Zeile einfügen
^V Einfügen ein-/ausschalten

Um die Datei zu speichern und wieder in das Betriebssystem zurückzukehren, geben Sie ^KX ein. Vom Prompt A> aus rufen Sie wie gewohnt dBASE auf und befehlen DO B:TEST. Sie können den Befehl MODI COMM für kleinere Änderungen benutzen oder auch dBASE mit QUIT verlassen und wieder mit dem Programm-Modus von WordStar ändern.

Haben Sie dBASE und WordStar auf derselben Diskette stehen, so können Sie die Programme im Programm-Modus schreiben und mit ^KD abspeichern. WordStar wird die Datei auf die Diskette schreiben und wieder zu seinem Hauptmenü zurückkehren. Von hier aus verlangen Sie mit dem Befehl R, daß ein anderes Programm ausgeführt wird. WordStar wird Sie daraufhin fragen:

```
PROG-Name?
```

und Sie geben an:

```
DBASE TEST
```

und dBASE wird das Programm ausführen.

Die Programmdatei können Sie mit WordStar ändern, wenn Sie dBASE auf dem normalen Wege mit QUIT verlassen. Daraufhin erscheint die Meldung:

```
Mit beliebiger Eingabe zurück zu WordStar:
```

auf dem Bildschirm, und Sie sind wieder Hauptmenü von WordStar. Hier bearbeiten Sie Ihre Datei wieder über die Option N, wie gehabt. Dieses Verfahren ist genauso gut, als hätten Sie WordStar in dBASE eingebaut.

Die wertvollste Einrichtung in WordStar ist für dBASE-Programmierer, daß man damit Textblöcke markieren und dann verschieben, kopieren, löschen oder in eine Datei auf der Diskette schreiben kann. Sie können auch eine beliebige Datei irgendwo in die gerade bearbeitete Datei einlesen. Das ist besonders für den Gelegenheitsprogrammierer wertvoll, weil er Teile von anderen Programmen zusammenstellen und damit ein neues Programm schreiben kann. Diese Möglichkeit verringert die Programmierzeit und die Fehlersuche entscheident. Sie können nämlich bereits ausgetestete Routinen in Ihr Programm übernehmen, ohne sie noch einmal schreiben und die Fehler suchen zu müssen.Hier sehen Sie die WordStar-Befehle, mit denen die Blockoperationen durchgeführt werden:

^KB	Markiert den Blockanfang
^KK	Markiert das Blockende
^KC	Kopiert den Block an die Cursor-Position
^KV	Verschiebt den Block an die Cursor-Position
^KY	Löscht den markierten Block
^KW	Schreibt markierten Block auf Diskette
^KR	Liest eine Datei in die geöffnete Datei ab der Cursor-Position
KR	ein

Obwohl Sie viele andere Texteditoren für die Erstellung von Programmen ebenso verwenden können, ist WordStar einer der besten. Der Zeitaufwand für das Erlernen eines guten Textprogramms lohnt sich auf jeden Fall, und Sie programmieren damit leichter.

Strukturiertes Programmieren

Als Antwort auf die Probleme beim Ändern und Verwalten von großen Programmen auf noch größeren Computern entwickelte man Mitte der 70er Jahre das strukturierte Programmieren. Dieselbe Technik kann auch bei dBASE II zum Modifizieren und bei der Fehlersuche angewendet werden. Die Faustregeln zum strukturierten Programmieren lauten:

1. Füge deutlich sichtbare Kommentare in das Programm ein, so daß du leicht jede Stelle finden kannst, die eine spezielle Aufgabe bearbeitet.

2. Verwende bei Programmzeilen mit Schleifen oder Entscheidungen einen Einzug, damit du leicht den Anfang und das Ende dieser Routinen erkennen kannst.
3. Versuche Befehle und Variablennamen selbsterklärend zu gestalten, statt mit geheimnisvollen Abkürzungen zu arbeiten.

Lassen Sie uns einmal zwei Programme vergleichen: eines, das die Methode der strukturierten Programmierung nicht verwendet und eines, das sie verwendet. Beide Programme bewirken genau dasselbe: Sie zeigen das Hauptmenü eines hypothetischen Literatur-Verwaltungspro-

```
* HAUPTMENÜ
USE B:library
DO WHILE T
ERASE
§ 1,20 SAY "LITERATUR-VERWALTUNG"
§ 3,25 SAY "1. Neue Einträge"
§ 4,25 SAY "2. Bericht drucken"
§ 5,25 SAY "3. Daten verändern"
§ 6,25 SAY "4. ENDE"
STORE " " TO wahl
§ 8,20 SAY "Bitte wählen (1-4) " GET wahl
READ
* Entsprechend verzweigen
IF wahl = "1"
   APPEND
   ELSE
     IF wahl ="2"
      REPORT FORM library
      ELSE
        IF wahl = "3"
          EDIT
          ELSE
            * Ende
            IF wahl = "4"
               RETURN
            ENDIF
         ENDIF
     ENDIF
ENDIF
ENDDO
```

Abb. 1.1: Das unstrukturierte Programm HAUPTMENÜ

gramms, fragen den Benutzer nach seiner Wahl aus dem Menü und führen anschließend das Gewünschte aus.

Beachten Sie, daß zwar einige Kommentare in diesem Programm stehen (Zeilen, die mit einem Stern beginnen), diese aber nicht besonders gut zu entdecken sind. Beispielsweise müssen Sie erst viele dBASE-Zeile überfliegen, bevor Sie die Anweisung * ENDE entdecken.

```
*****************************************LIBRARY.PRG
********************** HAUPTMENÜ für
******************Literaturverwaltung
USE B:library
STORE " " TO wahl

DO WHILE wahl # "4"
ERASE
§ 1,20 SAY "Literatur-Verwaltung"
§ 3,25 SAY "1. Neue Einträge"
§ 4,25 SAY "2. Bericht drucken"
§ 5,25 SAY "3. Daten verändern"
§ 6,25 SAY "4. Ende"
STORE " " To wahl
§ 8,20 SAY "Bitte wählen (1-4) " GET wahl
READ
* Entsprechend der Benutzereingabe verzweigen
DO CASE

CASE wahl = "1"
APPEND

CASE wahl = "2"
REPORT FORM B:library

CASE wahl = "3"
EDIT

ENDCASE

ENDDO (while wahl # 4)
**************** bei wahl=4, ENDE
RETURN
```

Abb. 1.2: Das strukturierte Programm HAUPTMENÜ

Zweitens machen die Einzüge den Programmtext nicht gerade leichter lesbar. Die Einzüge sind ungeschickt und verwirrend, besonders am Ende des Programms, wo eine Reihe offensichtlich zusammenhangloser ENDIF-Anweisungen steht. Wenn dieses Programm aus irgendwelchen Gründen hängenbleibt, wird es nicht einfach sein festzustellen, wo der Fehler steckt.

Drittens sind einige der Befehle nicht selbstdokumentierend. Zum Beispiel heißt die dritte Zeile von oben DO WHILE T. In dBASE ist die logische Variable T (true) immer wahr. Die Schleife erweckt also den Eindruck, endlos zu laufen. Wenn Sie sich aber die Zeilen darunter ansehen, werden Sie feststellen, daß die Schleife durchaus nicht ewig läuft. Gibt der Benutzer nämlich eine 4 als Auswahl ein, führt dBASE als nächstes den Befehl RETURN aus und springt in gewissem Sinne aus der Schleife zurück zum dBASE-Prompt. Achten Sie auch auf den ENDDO-Befehl am Ende des Programms, der da ganz alleine, ohne eine Erklärung steht. Wäre das ein großes Programm mit vielen DO WHILE-Schleifen, wäre es ziemlich schwierig festzustellen, zu welchem DO WHILE dieses ENDDO gehört. Dasselbe gilt für viele ENDIF-Befehle.

Vergleichen Sie dieses Programm mit dem in der Abb. 1.2, das nach den einfachen Regeln der strukturierten Programmierung geschrieben wurde.

Sehen Sie sich an, wieviel einfacher dieses Programm zu lesen ist gegenüber dem ersten. Einerseits sind alle Kommentare auf den ersten Blick zu erkennen, weil sie mit vielen Sternchen gekennzeichnet sind, die sie aus dem Progamm hervorheben. Außerdem sind Sie in klarem Deutsch geschrieben (Groß- und Kleinschreibung), so daß sie von allein ins Auge springen. Am Anfang des Programms sehen Sie auch einen Programmnamen und eine kurze Beschreibung darüber, was das Programm macht.

Beachten Sie auch, daß alle Zeilen mit DO WHILE ... ENDDO-Schleifen jeweils drei Zeichen eingezogen wurden. Sie können ganz leicht den Beginn und das Ende einer Schleife finden, indem Sie nur mit dem Finger in der entsprechenden Spalte entlangfahren und dabei das DO WHILE und das ENDDO auf gleicher Höhe finden.

Das Programm benutzt auch die Befehle DO CASE ... ENDCASE, um zu entscheiden, welche Aktion auf die Eingabe des Benutzers hin folgen soll. Das beseitigt die verwirrenden ENDIF-Anweisungen des ersten Programms und macht das Programm leichter lesbar. Die Befehle innerhalb der DO CASE ... ENDCASE-Konstruktionen sind ebenfalls eingerückt, wodurch die Schritte nach jeder Fallunterscheidung deutlich erkennbar werden.

Schließlich sind auch die Befehle in dem strukturierten Programm anschaulicher als die Befehle in dem nicht strukturierten Programm. Der Befehl DO WHILE OPTION # 4 sagt uns sofort, daß diese Schleife nicht ewig läuft. Sie bricht ab, wenn die Speicher-Variable OPTION gleich 4 ist. Wir brauchen nicht erst das geheimnisvolle Ende einer scheinbar unendlichen Schleife zu suchen. Zusätzlich steht neben dem ENDDO-Befehl am Ende des Programms eine Erklärung (while option 4), die uns gleich sagt, zu welchem Schleifenbeginn dieses ENDDO gehört. So präpariert, ist es zu einem späteren Zeitpunkt leicht, mit dem Programm zu arbeiten oder etwas zu ändern. In dBASE wird alles, was hinter einem ENDDO oder einem ENDIF steht, als Kommentar gewertet und hat keinen Einfluß auf den Programmablauf. Tun Sie sich deshalb selbst den Gefallen jedesmal aufzuschreiben, zu welchem DO WHILE oder IF ein ENDDO oder ENDIF gehört.

Zusammenfassung

Um eine dBASE-Befehlsdatei zu erstellen, benutzen Sie entweder MODIFY COMMAND oder einen Texteditor wie WordStar. Das Programm starten Sie dann, indem Sie dBASE laden und dann DO plus den Dateinamen eingeben (Beispiel: DO B:TEST). Folgen Sie den einfachen Regeln für die strukturierte Programmierung, wenn sie Ihre Programme schreiben; dadurch werden Ihre Programme leichter lesbar und übersichtlicher, wenn Sie später einmal Fehler suchen müssen oder etwas ändern wollen.

Im nächsten Kapitel werden Sie etwas über die Techniken zur Leistungssteigerung Ihrer bedienerfreundlichen dBASE-Programmsysteme erfahren.

Optimierte Leistung

2

Wir wollen alle mehr Leistung aus unserem Computer herausholen. Wir freuen uns, daß unser Computer Aufgaben in Minuten erledigt, für die ein Mensch Stunden braucht. Aber nicht jeder hat schon gemerkt, daß auch die Minuten noch weiter auf ein paar Sekunden verkürzt werden können. In diesem Kapitel wollen wir grundsätzliche Methoden der Leistungssteigerung bei dBASE II diskutieren und miteinander vergleichen.

Minuten werden zu Sekunden mit FIND und WHILE

Lassen Sie uns einige unterschiedliche Techniken für grundlegende Arbeiten mit dBASE vergleichen. Wir werden eine einfache Adressen-Datenbank für dieses Beispiel benutzen. Sie heißt POST.DBF und hat diese Struktur:

```
FLD   NAME       TYPE       WIDTH      DEC
001   NACHNAME   C           020        000
002   VORNAME    C           020        000
003   STRASSE    C           020        000
004   ORT        C           020        000
005   LAND       C           010        000
006   PLZ        C           010        000
007   BETRAG     N           009        002
```

Lassen Sie uns davon ausgehen, daß die Datei POST.DBF bereits 1000 Einträge enthält. Zehn Personen in dieser Datenbank haben den Nachnamen Müller (Nachname-Feld). Wie lange würde es wohl dauern, alle Müllers mit den Befehlen LIST, COUNT oder COPY in eine andere Datenbank zu bekommen? Wie lange würde es dauern, einen formatierten Bericht nur über die Müllers zu drucken oder die Beträge für alle Müllers zusammenzuzählen? Selbstverständlich hängt die Dauer für solche Operationen von der verwendeten Technik ab.

Um nun verschiedene Techniken miteinander vergleichen zu können, gehen wir davon aus, daß die Datenbank bereits über das Feld NACHNAME mit folgenden Befehlen indiziert wurde:

```
USE POST
INDEX ON nachname TO namen
```

Nachdem Sie diese Zeilen eingetippt haben, existiert eine Datei NAMEN.NDX auf Ihrer Diskette und Sie können den Befehl USE POST INDEX NAMEN benutzen, um mit der Datenbank POST und der Index-Datei NAMEN gleichzeitig zu arbeiten.

Lassen Sie uns die Verarbeitungszeiten untersuchen, die bei drei verschiedenen Befehlen und Methoden herauskommen. Zuerst wird der Befehl LIST FOR verwendet, um alle Müllers zu finden. Die Befehlsdatei würde dann so aussehen:

```
********************** Methode 1: mit LIST FOR
ERASE
USE POST INDEX NAMEN
ACCEPT "Alle Personen mit diesem Nachnamen aufli-
                          sten?" TO suchen
LIST FOR nachname=suchen
```

Wenn Sie dieses Programm laufen lassen, wird zuerst der Bildschirm geleert und dann die Frage gestellt:

```
Alle Personen mit diesem Nachnamen auflisten?
```

Angenommen, Sie geben den Namen Müller ein und drücken dann RETURN. Das Programm wird dann alle zehn Datensätze der Personen namens Müller auf dem Bildschirm auflisten. Es dauert auf einem Disketten-System etwa 148 Sekunden (2 1/2 Minuten), bis das Programm alle Namen auf den Bildschirm gebracht hat und wieder zum dBASE-Prompt zurückkehrt.

Bei der zweiten Möglichkeit lösen wir das gleiche Problem mit dem Befehl FIND, lassen ihn den ersten Müller in der Index-Datei

NAMEN.NDX finden und arbeiten dann mit einer DO WHILE-Schleife, um die restlichen neuen Müllers anzuzeigen. Die Programmdatei für die zweite Problemlösung sieht so aus:

```
************** Methode 2: mit FIND und DO..WHILE
ERASE
USE POST INDEX NAMEN
ACCEPT "Alle Personen mit diesem Nachnamen aufli-
                          sten?" TO suchen

FIND &suchen
DO WHILE nachname=suchen
   DISPLAY
   SKIP
ENDDO (while nachname=suchen)
```

Wenn Sie dieses Programm laufen lassen, werden Sie wieder nach den Namen der zu zeigenden Personen gefragt und bekommen die entsprechende Liste dann auf den Bildschirm. Dieses Programm braucht nur 29 Sekunden (weniger als ein halbe Minute), um alle zehn Müllers anzuzeigen und zum dBASE-Prompt zurückzukehren. Das bedeutet: es braucht nur etwa ein Drittel der Zeit des ersten Programms.

Eine noch einfachere und schnellere Methode besteht darin, den FIND-Befehl für das Finden des ersten Müller zu benutzen und dann die restlichen Personen mit diesem Namen durch den Befehl LIST WHILE anzeigen zu lassen. Die dritte Möglichkeit wird in dem hier gezeigten Beispiel-Programm verwendet:

```
************** Methode 3: mit FIND und LIST WHILE
ERASE
ACCEPT "Alle Personen mit diesem Nachnamen aufli-
                          sten?" TO suchen
FIND &suchen
LIST WHILE nachname=suchen
```

Jetzt dauert es sogar weniger als neun Sekunden, alle zehn Müllers anzuzeigen und dann zum dBASE-Prompt zurückzukehren. Offensichtlich sollte man die dritte Methode vorziehen (leider ist sie nicht in allen Situationen einsetzbar, wie Sie noch in folgenden Kapiteln sehen werden).

Sie können die Unterschiede der drei Vorgehensweisen in der Tabelle 2.1 sehen. Obwohl jede der drei Methoden haargenau das gleiche macht, unterscheiden sich die Ausführungszeiten doch erheblich.

Schneller rechnen

Die Arbeitsmethode mit FIND und WHILE bei einer indizierten Datenbank kann erhebliche Zeitersparnis bedeuten, auch wenn sie mit anderen Befehlen als LIST benutzt wird.

Methode Nr.	Benutzte Befehle	Zeitaufwand
1	LIST FOR	148,75 Sekunden
2	FIND und DO WHILE	29,91 Sekunden
3	FIND und LIST WHILE	8,94 Sekunden

Tab. 2.1: Vergleich der Verarbeitungszeiten bei drei verschiedenen Befehlen

Wenn Sie zum Beispiel dBASE fragen, wie viele Müllers in der Datenbank stehen, können Sie diese Befehle benutzen:

```
USE POST
COUNT FOR nachname="Müller"
```

Nach dieser Methode vergehen 55,5 Sekunden, bis Sie wissen, daß zehn Müllers in der Datenbank sind, und dBASE wieder zu seinem Prompt zurückkehrt. Sie können die Verarbeitungszeit deutlich verkürzen, wenn Sie diese Befehle anwenden:

```
USE post INDEX namen
FIND Müller
COUNT WHILE nachname="Müller"
```

Jetzt wird die gleiche Arbeit in nur 3,20 Sekunden erledigt, was ja wohl eine beachtliche Zeitersparnis bedeutet.

Angenommen, Sie wollen das Feld BETRAG ausschließlich für alle Müllers zusammenzählen? Sie könnten dazu diese Befehle verwenden:

```
USE post
SUM betrag FOR nachname="Müller"
```

Die Bearbeitungszeit für diese Methode beträgt 58 Sekunden. Hier ist eine Alternative:

```
USE post
FIND Müller
SUM betrag WHILE nachname="Müller"
```

So formuliert, verringert sich der Zeitaufwand auf ein Zehntel, was noch ganze 5,28 Sekunden bedeutet.

Schnellere Berichte

Den so hilfreichen Befehl WHILE können Sie auch mit dem REPORT-Befehl zusammen einsetzen. Nehmen Sie beispielsweise an, Sie haben gerade einen formatierten Bericht namens POSTLIST erstellt und dazu den Befehl REPORT benutzt. Um alle Müllers in dem Bericht erscheinen zu lassen, könnten Sie diese Befehle verwenden:

```
USE post
REPORT FORM postlist FOR nachname="Müller"
```

Auf diese Weise brauchen Sie 135,3 Sekunden, um die Müllers auf dem Bildschirm anzuzeigen und zum dBASE-Punkt zurückzukommen. Wir können aber auch eine schnellere Möglichkeit finden, wenn wir diese Befehle nehmen:

```
USE post INDEX namen
FIND Müller
REPORT FORM postlist WHILE nachname="Müller"
```

Diese Befehle bewirken dasselbe, aber in nur 14,03 Sekunden.

Schneller kopieren

Um Teile aus der Datenbank POST in eine Datenbank namens TEMP zu kopieren, benötigen die Befehle:

```
USE post INDEX namen
COPY TO temp FOR nachname="Müller"
```

200,6 Sekunden, was über drei Minuten bedeutet. Sie können das gleiche mit diesen Befehlen erreichen:

```
USE post INDEX namen
FIND Müller
COPY TO temp WHILE nachname="Müller"
```

Diese Befehle verkürzen die Arbeitszeit erfreulicherweise auf 18,7 Sekunden.

Schneller editieren

Wenn Sie Datenbanken indizieren, kann das auch den Änderungsprozeß beschleunigen. Angenommen, Sie möchten mit dem Befehl BROWSE

durch die Datenbank rollen, um die Daten eines Müller zu ändern. Sie könnten diese Befehle verwenden:

```
USE post
BROWSE
```

und bekommen die typische Bildschirmausgabe für BROWSE, in der die Namen und Adressen in ihrer originalen Reihenfolge erscheinen (siehe Abb. 2.1). Sie werden eine Reihe von ^C-Eingaben benötigen, um den gesuchten Müller zu finden. Es kann lange dauern, bis Sie den richtigen Müller gefunden haben, weil die Einträge mit diesem Namen über die gesamte Datenbank verstreut sind.

NACHNAME------	VORNAME--------	STRASSE----------
Bond	James	Aktionstr. 007
Kenney	David	Ottostr. 123
Neumann	Josef	Baumallee 341
Mohr	Richard	Westweg 350
Rohr	Brigitte	Seefeld 999
Wallace	Edgar	Krimiweg 345
Müller	Lieschen	Hauptstr. 601

Abb. 2.1: Ein BROWSE-Bildschirm ohne Index

Andererseits können Sie diese Befehle auch verwenden, um in der Datenbank mit BROWSE zu blättern:

```
USE post INDEX namen
FIND Müller
BROWSE
```

Der BRWOSE-Bildschirm wird jetzt den ersten Müller in der Datenbank zeigen und alle anderen Müllers direkt dahinter, wie Sie es in der Abbildung 2.2 sehen. Sie haben es jetzt nicht mehr nötig, seitenweise durch die Datenbank zu blättern, um den richtigen Müller für Ihre Änderung zu finden, weil alle zehn gleichzeitig und hintereinander auf einem Bildschirm stehen.

Genauso gehen Sie vor, wenn Sie die Daten eines bestimmten Müller mit dem Befehl EDIT ändern wollen. Sie verwenden die Befehle

```
USE post INDEX namen
FIND Müller
EDIT WHILE nachname="Müller"
```

Der erste gesuchte Name erscheint blitzschnell auf dem EDIT-Bildschirm. Mit jedem ^C gelangen Sie direkt zum nächsten Müller in der Datenbank. Also noch einmal: Sie sparen eine Menge Zeit, weil Sie nicht nach jedem Eintrag "Müller" gesondert suchen müssen.

Schneller sortieren

In einigen Fällen ist es besser, eine Datenbank physikalisch zu sortieren, statt mit einem Index zu arbeiten. Beispielsweise arbeitet der Befehl UPDATE am besten, wenn die Datei, von der aus aktualisiert wird, bereits physikalisch vorsortiert ist.

NACHNAME------	VORNAME--------	STRASSE----------
Müller	Lieschen	Hauptstr. 601
Müller	Franz	Baumweg 123
Müller	Ernst	Seestr. 347
Müller	Klaus	Am Flughafen
Müller	Beate	Prinzenstr.1
Müller	Dr. Horst	Universitätsstr. 23
Müller	Prof. Karl	Schulweg 78
Müller	Egon	Postf. 12345
Müller	Julie	
Müller	Johannes	Eichenallee 77

Abb. 2.2: BROWSE-Bildschirm mit indizierter Datenbank

Sie arbeiten mit dem SORT-Befehl, um beispielsweise die sortierte Datenbank TEMP aus der Datenbank POST zu erstellen, in folgender Weise:

```
USE post
SORT ON nachname TO temp
```

Diese Aufgabe bewältigt dBASE in 940 Sekunden (etwa 15 Minuten). Existiert die Index-Datei NAMEN.NDX, bekommen Sie das gleiche Ergebnis mit den Befehlen

```
USE post INDEX namen
COPY TO temp
```

Das Kopieren der indizierten Datei dauert dabei nur 326 Sekunden (ungefähr ein Drittel der vorherigen Zeit). Wenn Sie eine indizierte Datenbank in eine andere Datenbank kopieren, stehen die Datensätze in der neuen Datenbank in physikalischer Reihenfolge sortiert zur Verfügung.

Die Leistungssteigerung schätzen

Geschwindigkeit ist ist ein wichtiger Teil eines benutzerfreundlichen dBASE II-Software-Systems. Grundsätzlich erhöht sich die Verarbeitungszeit entsprechend der Anzahl der Datensätze in einer Datenbank. Daher kann das Anzeigen aller Müllers aus einer Datenbank mit 5000 Einträgen gut und gerne 740 Sekunden oder etwa zwölf Minuten dauern, wenn Sie die LIST FOR-Methode wählen. Verwenden Sie die Befehle FIND und LIST WHILE für dieselbe Aufgabe, schaffen Sie es in 45 Sekunden (weniger als eine Minute). Bei 10 000 Datensätzen brauchen Sie mit LIST FOR 24 Minuten (fast eine halbe Stunde), während mit FIND und LIST WHILE alles nach 88 Sekunden (weniger als 1 1/2 Minuten) vergessen ist.

Was aber ist mit der Zeit, die man bei diesen Beispielen zum Indizieren mit dem Befehl INDEX ON nachname TO namen braucht? Insgesamt dauert es 529,06 Sekunden, um eine Datenbank mit 1000 Einträgen zu indizieren. Sie können diesen Zeitverbrauch allerdings in kleine Stückchen aufteilen, wenn Sie die Datenbank direkt nach ihrer Erstellung indizieren.

Wenn sie die Datenbank POST.DBF erstellen und dann sofort den Befehl INDEX ON nachname TO namen geben, geht die dafür aufgewendete Zeit gegen Null. Die Zeit, die durchschnittlich nötig ist, um eine leere Datenbank zu indizieren, beträgt nur ein paar Sekunden. Wenn Sie dann vor einer Änderung oder der Neueingabe von Daten (oder sonst irgendeiner Bearbeitung) USE post INDEX ON nachname TO namen eingeben, wird die Index-Datei ziemlich flott aktualisiert. Sie brauchen also nicht nach jeder Neueingabe oder Änderung den Befehl INDEX ON nachname TO namen einzugeben. Auf diese Weise verringert sich die Indizierungszeit auf ein paar Sekunden oder weniger.

Mit mehreren Index-Dateien arbeiten

In den vorhergehenden Beispielen haben wir die Verarbeitungszeiten beim Einsatz einer Index-Datei, NAMEN genannt, verglichen. Diese Index-Datei ist nur nach dem Feld NACHNAME geordnet. In der Praxis wird eine Versandliste bestimmt zwei unterschiedliche Sortierungen benötigen:

1. Eine Sortierung nach Vor- und Nachnamen, um ein Inhaltsverzeichnis drucken zu können,
2. Eine Sortierung nach Postleitzahlen, für Massendrucksachen

Werden regelmäßig zwei Sortierungsarten benötigt, können Sie zwei Index-Dateien anlegen, um die beiden unterschiedlichen Sortierungen zu

handhaben. Die eine Index-Datei, die wir NAMEN nennen werden, beinhaltet die Datensätze der Versandliste, nach Nach- und Vornamen geordnet. Die zweite Index-Datei, die wir PLZ taufen werden, hat die Daten nach Postleitzahlen geordnet, um Massensendungen zu ermöglichen. Sie erstellen diese beiden Index-Dateien, indem Sie zuerst die Datenbank POST.DBF mit dem Befehl CREATE aufbauen und dann die beiden Index-Dateien mit diesen Befehlen erstellen:

```
USE post
INDEX ON nachname + vorname TO namen
INDEX ON plz to plz
```

Die Datei POST.DBF hat jetzt zwei sie begleitende Index-Dateien: NAMEN.NDX und PLZ.NDX. Sie können beide auf dem neusten Stand halten, wenn Sie diese Befehle eingeben:

```
USE post INDEX namen,plz
```

Indem Sie die beiden Index-Dateien so benennen, bekommen diese alle Änderungen, die Sie nach den Befehlen APPEND, EDIT, BROWSE, READ oder REPLACE eingeben, mit und bleiben automatisch immer auf dem neusten Stand.

Listen Sie die Datensätze der Datenbank auf oder zeigen Sie sie auf dem Bildschirm an, erscheinen sie immer nach Nachnamen sortiert, weil die Index-Datei NAMEN als erste in der oben gezeigten Anweisung steht. Deshalb brauchen Sie auch nicht die INDEX ON plz TO plz-Prozedur noch einmal einzugeben, wenn Sie die Datensätze der Datenbank nach Postleitzahlen geordnet anzeigen wollen. Statt dessen geben Sie einfach ein:

```
SET INDEX TO plz
```

Weil diese Arbeitsweise die Zeit zum Neuindizieren der Datenbank einspart, wird Ihr dBASE-Programm erheblich schneller laufen.

Nachteile der Leistungssteigerung

Sie werden einige Nachteile in Kauf nehmen müssen, wenn sie mit mehrfachen Index-Dateien arbeiten wollen. Grundsätzlich verlangsamen die APPEND-, EDIT- oder REPLACE-Prozeduren, je mehr Index-Dateien Sie zu einem bestimmten Zeitpunkt gleichzeitig geöffnet haben. Wenn Sie beispielsweise neue Datensätze in die Datei POST.DBF einfügen möchten und dabei keinen Index benutzen, können Sie die Befehle

```
USE post
APPEND
```

verwenden. Sie schreiben dann die Neueinträge in die Maske auf dem Bildschirm, und dBASE nimmt diese direkt an, speichert ab und läßt Sie weiter eingeben. Erstellen Sie aber vier Index-Dateien und halten sie alle gleichzeitig offen, etwa so:

```
USE post INDEX namen,plz,ort,land
```

werden Sie eine Verzögerung bemerken, bis dBASE wieder die Eingabemaske auf den Bildschirm bringt. Bei einer großen Datenbank mit über 1000 Datensätzen kann das bis zu 20 Sekunden dauern, je nachdem, wie viele Felder in jeder Index-Datei stehen.

Im allgemeinen aber reichen zwei Index-Dateien pro Datenbank völlig aus, wie Sie noch an den folgenden Beispielen in diesem Buch sehen werden. Die Verzögerungen durch eine oder zwei Indizes fallen kaum ins Gewicht und werden reichlich wiedergutgemacht durch die Zeitvorteile der Befehle FIND und WHILE sowie die eingesparte Zeit für das Neuindizieren.

Ein anderer Nachteil der Index-Dateien stellt sich beim globalen Ersetzen in einer Datenbank heraus. Sie können beispielsweise folgende Befehle benutzen, um in unserer angenommenen Datenbank POST.DBF alle BETRAG-Felder auf Null zurückzusetzen:

```
USE post INDEX namen,plz
REPLACE ALL betrag WITH 0
```

Bei einer Datenbank mit 1000 Datensätzen kann dieser Vorgang 45 Minuten dauern. Die hier beschriebenen Befehle verschwenden unnötig viel Zeit, weil sie die Index-Datei auch aktualisieren.

Erinnern Sie sich, daß die Datei NAMEN.NDX die Felder NACHNAME und VORNAME enthält und PLZ.NDX das Feld PLZ für die Postleitzahl. Das Feld BETRAG hat keinen direkten Einfluß auf irgendeinen dieser Indizes. Wenn Ihnen solch ein Fall begegnet, können Sie die Option NOUPDATE (verfügbar in der dBASE II-Version 2.4 und höher) benutzen, um die Ersetzung durchführen zu lassen. Diese Befehle hier erledigen die Arbeit in etwa sechs Minuten:

```
USE post INDEX namen,plz
REPLACE NOUPDATE ALL betrag WITH 0
```

Die NOUPDATE-Option sagt dBASE, daß es die Index-Dateien während des Ersetzens nicht ändern soll. Sie müssen sich an einige wichtige Punkte erinnern, wenn Sie mit den Befehlen FIND und INDEX arbeiten wollen. Erstens arbeitet der FIND-Befehl nur mit einer indizierten Datenbank zusammen. Wird eine Datenbank mit mehreren Index-Dateien eröffnet, arbeitet FIND nur mit dem zuerst aufgeführten Index

zusammen. Wenn Sie beispielsweise die Datenbank POST mit den unten aufgeführten Index-Dateien eröffnen:

```
USE post INDEX plz,namen
```

kann der FIND-Befehl nur zum Auffinden einer Postleitzahl genutzt werden.

Sind zweitens die in der Datenbank zu suchenden Daten in einer Variablen gespeichert, dann muß der Variablenname für die Verwendung mit FIND als Makro gekennzeichnet sein (&), wie Sie hier sehen:

```
ACCEPT "Wen suchen? " TO suchen
FIND &suchen
```

Außerdem unterstützt FIND keine Funktionen oder Operatoren. Sie können also nicht befehlen: „FIND Müller .OR. Schmidt" oder „FIND nachname > &suchen".

Wenn Sie zwei Index-Dateien erstellen, aber später beim Ändern, Hinzufügen oder Löschen nur eine oder keine der beiden Dateien aktivieren, werden die Indizes zerstört, und Sie erhalten zur Strafe die Meldung: DATENSATZ AUSSERHALB DES MÖGLICHEN BEREICHES. In diesem Falle müssen Sie die beiden Index-Dateien neu erstellen, indem Sie diese Befehle eintippen:

```
USE post
INDEX ON nachname + vorname TO namen
INDEX ON plz TO plz
```

Noch einmal: Sie können die Zeit für die Neuindizierung einsparen, wenn Sie immer beide Index-Dateien aktivieren, mit dem Befehl USE post INDEX namen,plz, sobald Sie mit der Datenbank arbeiten.

Auf Leistung programmieren

Die Art und Weise, wie Sie ihr Programm strukturieren, hat ebenfalls einen Einfluß auf die Ablaufgeschwindigkeit Ihres gesamten Programmpaketes. Zum Beispiel können Sie das folgende einfache Programm verwenden, um nach Postleitzahlen geordnete Etiketten zu drucken:

```
********************************** LABELS.CMD
***************** Etiketten von POST.DBF drucken
USE post INDEX plz
DO WHILE .NOT. EOF
   DO mformat
   SKIP
ENDDO
```

Beachten Sie, daß dieses Programm LABELS.CMD für jedes Etikett zu einem anderen Programm mit dem Namen MFORMAT verzweigt. Das Programm MFORMAT sieht so aus:

```
********************************** MFORMAT.CMD
? TRIM(vorname),nachname
? strasse
? plz + " "+TRIM(ort)
? land
?
?
RETURN
```

Das LABELS-Programm braucht 20 Minuten, um 1000 nach Postleitzahlen sortierte Etiketten zu drucken, wenn Sie die Index-Datei PLZ erstellt haben.

Sie können die Verarbeitungszeit auf die Hälfte reduzieren, wenn Sie dBASE nicht zwingen, für jedes gedruckte Etikett zum Programm MFORMAT zu verzweigen. Benutzen Sie dazu die folgenden Befehle:

```
********************************** LABELS.CMD
******************** Etiketten drucken von POST.DBF
USE post INDEX plz
DO WHILE .NOT. EOF
  ? TRIM(vorname),nachname
  ? strasse
  ? plz +" "+ TRIM(ort)
  ? land
  ?
  ?
  ?
  SKIP
ENDDO
```

Dieses Programm wird 1000 Etiketten nach Postleitzahlen geordnet in etwa zehn Minuten drucken. Da dBASE nicht bei jedem Schleifendurchgang herausfinden muß, was alles in der Datei MFORMAT steht, kann es die gleiche Datenmenge in der halben Zeit verarbeiten.

Grundsätzlich erzielen Sie einen gewaltigen Zeitgewinn, wenn Sie vermeiden, andere Programme innerhalb einer DO WHILE ... ENDDO-Schleife aufzurufen. Alle Programmbeispiele in diesem Buch folgen dieser wichtigen Regel.

Sobald Sie einmal größere Softwaresysteme entwickeln, werden Sie feststellen, daß Sie nicht immer nur die schnellste Möglichkeit für jeden

Arbeitsabschnitt verwenden können. Da die meisten Datenbanken viele Felder enthalten, ist es unpraktisch, für jedes Feld eine Index-Datei aufzubauen. Aber ein wenig Planung kann die Arbeit schon merklich beschleunigen, wie Sie an dem Beispiel des nächsten Kapitels sehen werden.

Zusammenfassung

In diesem Kapitel haben wir eine ganze Reihe von Techniken kennengelernt, mit denen man die Arbeitsgeschwindigkeit von dBASE II-Programmen und Programmsystemen erhöhen kann. Die Verwendung von Index-Dateien kann die Leistungsfähigkeit eines dBASE II-Programmsystems beträchtlich erhöhen, weil die Datenbank nicht mehr nach jeder Neueingabe, Änderung oder Löschung sortiert werden muß und weil der FIND-Befehl das schnelle Auffinden von Daten in einem indizierten Feld ermöglicht. Zusätzlich können Sie mit WHILE statt mit FOR arbeiten, wenn Sie den Bereich der zu durchsuchenden Datensätze angeben (z.B.: WHILE nachname = "Müller", statt FOR nachname = "Müller).

Mehrfache Index-Dateien ermöglichen es einem dBASE-Programmsystem, mehrere Sortierungen gleichzeitig vorzunehmen. Ist eine Index-Datei einmal mit dem Befehl INDEX ON erstellt, kann man sie jederzeit aktivieren, indem man sie in den USE-Befehl mit aufnimmt. Um beispielsweise die beiden Index-Dateien NAMEN und PLZ mit der Datenbank POST zu aktivieren, geben Sie ein:

```
USE post INDEX namen,plz
```

Auf diesem Wege werden beide Index-Dateien automatisch aktualisiert, sobald in der Datenbank irgendeine Änderung vorgenommen wird.

Arbeitet eine Datenbank mit mehreren Indizes, ist der zuerst aufgeführte Index ausschlaggebend (in dem obigen Beispiel NAMEN). Werden Datensätze mit Hilfe der Befehle LIST, DISPLAY, REPORT, COPY, BROWSE oder einem anderen Befehl angezeigt, erscheinen sie immer in der Sortierung, die die erste Index-Datei vorgibt. Auch der FIND-Befehl arbeitet nur mit dem ersten Index zusammen. Eine einzelne Datenbank kann bis zu sieben Index-Dateien gleichzeitig verwalten.

Man kann dBASE-Programme auch dadurch schneller machen, daß man keine DO-Befehle innerhalb von DO WHILE-Schleifen einbaut. Indem das eine Programm immer wieder aufgerufen wird, erzeugt man eine Unmenge von Diskettenzugriffen und verdoppelt die Verarbeitungszeit für den jeweiligen Vorgang.

Programmentwurf und Programmentwicklung

3

Ein Programmpaket zu erstellen, ähnelt in der Vorgehensweise ziemlich dem Erstellen eines Unterrichtsplans. Die erste Frage ist: „Wo fange ich an?". Genau wie bei einem Unterrichtsplan kann ein Programmpaket eine chaotische Ansammlung von Informationen sein, wenn Sie nicht Ihr Ziel klar vor Augen haben und einen Plan, wie Sie dahin kommen wollen. In diesem Kapitel möchte ich mit Ihnen die Techniken diskutieren, mit deren Hilfe Sie das Ziel eines Programmpakets festlegen und auch erreichen. So nebenbei werden wir eine ganze Anzahl von Programmiertechniken kennenlernen, die in vielen anderen Anwendungen hilfreich einzusetzen sind. Diese Techniken beinhalten

1. das Erstellen von komfortablen Masken zum Eingeben und Ändern von Daten,
2. den Einsatz des Maskengenerators ZIP, zum Erstellen von Masken,
3. das Erstellen von menügesteuerten Such- und Sortieroperationen,
4. das Arbeiten mit mehreren Index-Dateien,
5. das Erstellen von komfortablen Berichten, die auf dem Bildschirm stehen bleiben und über einen Seitenumbruch verfügen,
6. das Prüfen auf doppelte Datensätze in einer Datenbank.

Software-Entwurf und Entwicklung

Die zum Entwurf und zur Entwicklung der Software notwendigen Schritte können wie folgt zusammengefaßt werden:

1. Bestimmen Sie das Ziel des Projekts.
2. Entwerfen Sie die Datenbankstruktur.
3. Entwerfen Sie die Programmstruktur.
4. Entwerfen und entwickeln Sie jede Befehlsdatei des Systems nach dem folgenden Muster:
   - Entwerfen Sie die Eingabemasken und Berichte.
   - Entwerfen Sie das Programm in einem Pseudocode.
   - Schreiben, testen und korrigieren Sie jedes Programm des Systems für sich allein.

Der beste Weg, jeden dieser Schritte zu untersuchen, ist, ihn auszuführen. Also, lassen Sie uns mit dem Entwerfen und Entwickeln eines dBASE-Programmsystems beginnen. Wir fangen mit dem ersten Schritt an und bestimmen das Ziel des Projekts.

Das Ziel des Projekts bestimmen

Als erstes Programmbeispiel entwerfen und entwickeln wir ein vernünftiges Adressenprogramm, das benutzerfreundlich, schnell und flexibel ist. Ein Adressenprogramm ist ein gutes Anfangsbeispiel, weil es eine leicht zu erlernende Anwendung darstellt.Das Ziel des Projekts zu entwickeln, kann einfach bedeuten, daß Sie Sie sich Gedanken darüber machen, welche Aufgaben das Programm erledigen soll, und diese Gedanken dann zu Papier bringen. Hier ist eine einfache Erklärung unseres Programms.

Das Adressenprogramm wird es dem Benutzer ermöglichen, Namen, Adressen und Schlüsselwörter in einer Datenbank zu speichern. Wenn er das Programm aufruft, sieht er zuerst das Hauptmenü mit den wichtigsten Wahlmöglichkeiten. Diese Optionen sollen so aussehen:

```
Adressen-Verwaltung

1.  Neue Namen und Adressen eingeben
2.  Berichte drucken
3.  Schnellsuchen
4.  Namen und Adressen ändern
```

```
5.  Namen und Adressen löschen
6.  Dubletten suchen
7.  ENDE
```

Option 1 ermöglicht dem Benutzer die Eingabe von neuen Namen und Adressen unter Zuhilfenahme einer komfortablen Eingabemaske, die außerdem noch Informationen über die Möglichkeiten der Cursor-Steuerung enthält.

Die 2. Option bietet die Möglichkeit, Etiketten oder ein Inhaltsverzeichnis auszudrucken oder eine MailMerge-Datei zu erstellen, um mit der WordStar-Option Serienbriefe zu schreiben. Bevor ein Bericht ausgedruckt wird, kann der Benutzer aus einem einfachen Menü die Art der Sortierung (nach Name, Postleitzahl, Schlüssel, Ort oder Land) auswählen. Außerdem soll der Benutzer die Möglichkeit haben, aus einem einfachen Menü bestimmte Suchbedingungen einzugeben, wie „nur suchen nach Personen mit dem Namen Schmidt" oder „alle Personen mit dem Kennwort GEKÜNDIGT 31/10/84". Wahlweise kann sich der Benutzer auch dazu entschließen, alle Namen und Adressen auszudrucken oder in eine MailMerge-Datei zu schicken.

Beim Etikettendruck möchte er vielleicht mehrere Etiketten für jede Adresse haben. Außerdem sollte das Etikettenprogramm dem Benutzer die Möglichkeit anbieten, den richtigen Sitz der Etiketten im Drucker zu prüfen und Korrekturen noch vor dem endgültigen Ausdruck zuzulassen.

Die Option 3 gestattet, schnell eine Adresse oder eine Telefonnumer nachzuschlagen. Sollen die Daten eines Eintrags in der Datenbank geändert werden, wählt er die Option Nummer 4 und gibt einfach nur den Namen der Person ein, deren Daten geändert werden sollen. Der Löschvorgang unter Option 5 funktioniert ganz ähnlich: Es wird einfach der zu löschende Name eingegeben. Diese Option macht zusammen mit dem Benutzer eine doppelte Kontrolle, bevor der Eintrag endgültig gelöscht wird.

Mit der Option 6 erhält der Benutzer eine Liste aller Namen, die möglicherweise in der Datenbank doppelt vorhanden sind.

Die letzte Wahlmöglichkeit führt wieder zum Betriebssystem (A>) zurück.

Die obige Beschreibung bietet ein ganzes Bündel von Zielen, auf die wir hinarbeiten werden. Wir bieten eine ganze Reihe Optionen an und haben sichergestellt, daß auch ein Anfänger das System leicht bedienen kann. Beispielsweise verlangt dBASE immer die Datensatznummer des Datensatzes, der geändert oder gelöscht werden soll. Wir machen dem Benutzer das Leben leichter, indem wir gestatten, statt der Datensatznummer ein-

fach den gewünschten Namen einzugeben. Ebenso kann der Benutzer Such- und Sortierkriterien angeben, ohne die entsprechenden dBASE-Befehle oder die Syntax für eine solche Aufgabe zu kennen.

Mit dem nächsten Schritt erstellen wir eine Datenbank, die die benötigten Informationen aufnimmt.

Die Datenbankstruktur erstellen

Entgegen einer weit verbreiteten Meinung bezieht sich die Bezeichnung Datenbank auf alle Datenbank- und Index-Dateien eines Programmsystems statt nur auf eine einzelne DBF-Datei von dBASE II. Für unser System brauchen wir zwar nur eine einfache Datenbank mit Platz für die Namen und Adressen, aber dazu eine stattliche Anzahl von Index-Dateien.

Die Namen und Adressen werden auf der Diskette im Laufwerk B in einer POST.DBF getauften Datei abgelegt. Die Datenbankstruktur sehen Sie in Abb. 3.1.

Das Feld SWORT gibt dem Benutzer zusätzliche Flexibiliät beim Erstellen von Suchbedingungen. Beispielsweise könnte diese Datenbank für die Versandliste eines Clubs benutzt werden. Der Programmanwender könnte in der Datenbank Vermerke über das Austrittsdatum einzelner Mitglieder eingeben und damit verhindern, daß jemand nach seinem Austritt noch die Clubnachricht zugeschickt bekommt. Stehen in diesem Feld Berufsbezeichnungen, können Briefe an eine bestimmte Berufsgruppe, wie Anwälte oder Lehrer geschickt werden.

STRUKTUR DER DATEI: B:POST.DBF

FELDNR	NAME	TYP	WEITE	DEZIMALSTELLEN
001	NACHNAME	C	020	
002	VORNAME	C	020	
003	FIRMA	C	020	
004	STRASSE	C	025	
005	ORT	C	020	
006	LAND	C	005	
007	PLZ	C	004	
008	TELEFON	C	013	
009	SWORT	C	020	
** TOTAL **			00147	

Abb. 3.1: Datenbankstruktur für die Adressen-Verwaltung

Die Struktur hat auch Platz für Telefonnummern, da es die Schnellsuch-Funktion, die wir mit anbieten wollen, dem Benutzer erlaubt, nach einer Telefonnummer zu suchen.

Als nächstes müssen wir uns überlegen, welche Art von Index-Dateien der Anwender brauchen wird. Da er sicherlich Etiketten für seine Serienbriefe drucken möchte, sollten wir einen Index für die Postleitzahlen erstellen.

Wir haben auch gesagt, daß Ändern, Löschen und eine Schnellsuche auf der Basis der vorhandenen Namen möglich sein soll. Aus diesem Grunde erstellen wir einen Index für die Namen, so daß das Programm den FIND-Befehl bei der Suche verwenden kann (dieser Index wird auch bei der Aufstellung und dem Druck des Inhaltsverzeichnisses hilfreich sein, das wahrscheinlich in alphabetischer Reihenfolge der Namen erscheinen soll). Bestimmt gibt es mindestens zehn Müllers und fünfzehn Schmidts in der Datenbank, so daß der Index dafür sorgen wird, daß alle Personen mit einem so verbreiteten Namen anhand ihres Vornamens sortiert, auf der Liste erscheinen.

Wir können dem System auch beibringen, alle Probleme, die der Benutzer bei einer Suche in der Datenbank mit der Groß- und Kleinschreibung haben sollte, zu lösen. Wir erstellen einfach den Index für die Namen in Großbuchstaben. Jedesmal wenn ein Suchname eingegeben wird, wird das Programm den Namen erst in Großbuchstaben umwandeln und dann mit der Suche beginnen. Wir nehmen also dem Benutzer die Probleme mit der Schreibweise ab und bürden sie dem Programm auf.

Wenn Sie sich entschieden haben, was in der Datenbank gespeichert werden soll und welche Index-Dateien Sie brauchen, sind Sie soweit, diese in das Programm einzutragen. Für dieses Beispiel, laden Sie jetzt dBASE und schalten mit SET DEFAULT TO B auf das Laufwerk B (oder auf das von Ihnen benutzte Laufwerk) um. Dann bauen Sie die Datenbank mit CREATE POST so auf, wie Sie es in der Abb. 3.1 gesehen haben.

Dann erstellen Sie mit der leeren Datenbank den Index für die Postleitzahlen mit folgenden Befehlen:

```
USE post
INDEX ON plz TO plz
```

Um den Index für die Namen in Großbuchstaben zu erstellen, geben Sie ein:

```
USE post
INDEX ON !(nachname) +!(vorname) TO namen
```

Jetzt besteht die vollständige Datenbank aus drei Dateien:

```
POST.DBF
PLZ.NDX
NAMEN.NDX
```

Sie sind alle leer und aufnahmebereit für die Daten. Nachdem Sie die Datenbankstruktur festgelegt haben, sollten Sie als nächsten Schritt einen Plan für die Erstellung der Programmdateien machen.

Die Programmstruktur erstellen

Die Ziele, die wir uns weiter oben in diesem Kapitel gestellt haben, werden ziemlich viel Software nötig machen. Es wäre sehr schwierig, eine gigantische Programmdatei zu schreiben, die alle Aufgaben allein erledigt. Deshalb lassen Sie uns die Gesamtaufgabe in kleinere Teilaufgaben unterteilen und für jede dieser Teilaufgaben ein eigenes Programm

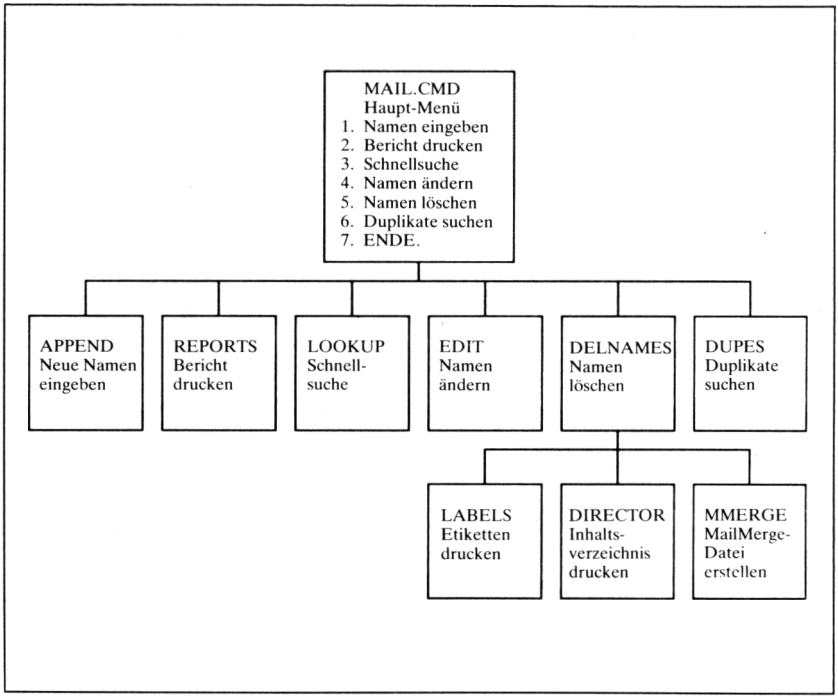


Abb. 3.2: Die Software-Struktur des Adressen-Verwaltungs-Systems

schreiben, das wir dann auch einzeln entwickeln, schreiben und austesten können. Auf Dauer gesehen wird unsere Entwicklungsarbeit dadurch erheblich einfacher. Jeder professionelle Programmierer wird Ihnen bestätigen, daß der Schlüssel für die Entwicklung großer Programmsysteme darin liegt, die scheinbar riesige und unüberwindbare Aufgabe in kleine Stücke zu zerteilen, mit denen man dann leicht arbeiten kann. Abb. 3.2 zeigt die Programmstruktur unseres Adressen-Verwaltungs-Systems.

Beachten Sie dabei die hierarchische Struktur der Programmdateien. Das Hauptmenü steht an der Spitze der Hierarchie, weil es seine Aufage ist festzustellen, welche Arbeit der Benutzer erledigt haben möchte. Hat der Benutzer eine Auswahl getroffen, verzweigt es zu dem entsprechenden Programm. Das REPORT-Programm, untergeordnet unter das Hauptmenü, hat auch verschiedene Aufgaben zu bewältigen. Es muß herausfinden:

1. den Berichtstyp (Etiketten, Inhaltsverzeichnis oder MailMerge-Datei),
2. die verlangte Sortierung,
3. die Suchbedingungen.

Nachdem das REPORT-Programm alle Informationen beisammen hat, verzweigt es zu dem entsprechenden Unterprogramm (entweder LABELS, DIRECTOR oder MMERGE). Sobald ein Unterprogramm seine Arbeit beendet hat, gibt es die Kontrolle wieder an das aufrufende Programm (direkt über ihm in der Hierarchie) zurück.

Ein hierarchisches Softwaresystem wie dieses erlaubt es uns, das übergroße Gesamtprogramm in kleine, handliche Stücke zu unterteilen, die relativ leicht zu schreiben und zu testen sind.

Die einzelnen Befehlsdateien entwerfen und entwickeln

Der erste Schritt beim Entwurf jeder Datei ist festzulegen, welche Ein- oder Ausgaben auf dem Bildschirm oder dem Drucker erscheinen sollen. Als erstes Beispiel werden wir das Hauptmenü für die Adressenverwaltung entwickeln. Für den Benutzer des Programms bringt diese Datei das Hauptmenü so auf den Bildschirm, wie Sie es in der Abb. 3.3 sehen.

Als nächstes müssen wir dann den logischen Ablauf dieses Programms festlegen. Das geht am besten so, daß Sie das Programm erst einmal in einem sogenannten Pseudocode erstellen, der in deutschen Wörtern beschreibt, was das Programm wann macht. Die Abb. 3.4 zeigt den Pseudocode für das Hauptmenüprogramm.

Nachdem der Programmablauf im Pseudocode festgelegt ist, können wir uns daran machen und ihn in dBASE II-Befehle übersetzen. Wir fangen an, indem wir dem Programm einen Namen geben, die dBASE-Meldungen mit SET TALK OFF abschalten und das Laufwerk B als Standardlaufwerk anmelden.

```
************************************** MAIL.CMD
*********************** Adressenverwaltung Hauptmenü
SET TALK OFF
SET DEFA TO B
```

```
HAUPTMENÜ ADRESSENVERWALTUNG

1. Neue Namen und Adressen eingeben
2. Reports drucken
3. Schnellsuchen
4. Namen und Adressen ändern
5. Namen und Adressen löschen
6. Duplikate suchen

7. ENDE

Geben Sie Ihre Wahl (1-7) ein: :
```

Abb. 3.3: Das Hauptmenü der Adressenverwaltung

Dann bestimmen wir POST.DBF als zu benutzende Datenbank mit den Indexdateien NAMEN und PLZ.

```
*********** Datenbank und Index-Dateien einrichten
USE post INDEX namen,plz
```

Wir haben den Index NAMEN vor den Index PLZ geschrieben, damit NAMEN als erster Index benutzt wird. So kann das Programm den FIND-Befehl benutzen, um Namen in der Datenbank schnell zu finden. Den PLZ-Index haben wir mit aufgeführt, damit er bei allen Änderungen automatisch auch mit verändert wird. Auf diesem Wege sorgen wir dafür, daß der PLZ-Index nicht jedesmal mit REINDEX auf den neusten Stand gebracht werden muß, wenn der Benutzer die Daten nach Postleitzahlen sortiert ausdrucken will.

Jetzt beginnen wir eine Schleife, um das Hauptmenü anzuzeigen:

```
***************** Schleife zur Anzeige des Menüs
STORE 0 TO wahl
DO WHILE wahl # 7
   ERASE
   TEXT
       HAUPTMENÜ ADRESSENVERWALTUNG

1.  Neue Namen und Adressen eingeben
2.  Reports drucken
3.  Schnellsuchen
4.  Namen und Adressen ändern
5.  Namen und Adressen löschen
6.  Duplikate suchen

7.  ENDE
   ENDTEXT

************************ Benutzerwahl erfragen
   §12,12 SAY 'Geben Sie Ihre Wahl (1-7) ein';
   GET wahl PICTURE "9"
   READ
```

MAIL.CMD Hauptmenü für die Adressenverwaltung

dBASE-Meldungen ausschalten
Auf Laufwerk B schalten

Datenbank POST.DBF mit den beiden Indexdateien NAMEN
und PLZ benutzen

Schleife für das Hauptmenü beginnen
Solange der Benutzer nicht aufhören will,
Bildschirm löschen
Hauptmenü anzeigen

1. Neue Namen und Adressen eingeben
2. Reports drucken
3. Schnellsuchen
4. Namen und Adressen ändern

Abb. 3.4: Pseudocode für das Hauptmenü der Adressenverwaltung

```
        5. Namen und Adressen löschen
        6. Duplikate suchen

        7. ENDE

    Antwort des Benutzers abwarten

    Wenn Option 1 gewählt
    Verzweige zu APPEND

    Wenn Option 2 gewählt
    Verzweige zu REPORTS

    Wenn Option 3 gewählt
    Verzweige zu LOOKUP

    Wenn Option 4 gewählt
    Verzweige zu EDIT

    Wenn Option 5 gewählt
    Verzweige zu DELNAMES

    Wenn Option 6 gewählt
    Verzweige zu DUPES

    Wenn der Benutzer nicht aufhören will, Hauptmenü weiter
    anzeigen

    Andernfalls QUIT
```

Abb. 3.4: Pseudocode für das Hauptmenü der Adressenverwaltung (Forts.)

Wir haben eine Variabale mit dem Namen CHOICE erstellt und den Wert 0 darin gespeichert. Der nächste Befehl DO WHILE choice 7 stellt sicher, daß das Hauptmenü so lange angezeigt wird, wie der Benutzer nicht 7 eingibt, um das Programm zu verlassen. Innerhalb der Schleife löscht das Programm zuerst den Bildschirm (ERASE) und zeigt dann das Hauptmenü, das im Programmtext zwischen den Befehlen TEXT und ENDTEXT steht.

Um herauszufinden, was der Benutzer machen möchte, haben wir die Meldung „Geben Sie Ihre Wahl (1–7) ein" unter das Menü in die Zeile

12, Spalte 12 plaziert. Die GET- und READ-Befehle warten auf die Eingabe einer einzelnen Ziffer. Mit Hilfe des PICTURE-Befehls stellen wir sicher, daß nur eine Ziffer als gültige Eingabe akzeptiert wird.

Nachdem der Benutzer eine Antwort eingegeben hat, verzweigt das Programm zu dem entsprechenden Unterprogramm, um die verlangte Arbeit zu erledigen. Die nächste Programmzeilen erledigen das:

```
********************** Verzweige zu dem entsprechen-
                       den Programm
DO CASE

   CASE wahl = 1
     DO append

   CASE wahl = 2
     DO reports

   CASE wahl = 3
     DO lookup

   CASE wahl = 4
     DO edit

   CASE wahl = 5
     DO delnames

   CASE wahl = 6
     DO dupes

ENDCASE
```

Dann müssen wir die DO WHILE-Schleife wieder schließen und dem Benutzer die Möglichkeit zum Verlassen von dBASE geben. Die nächsten beiden Zeilen erledigen das:

```
ENDDO (while wahl#7)

********************** Wenn wahl = 7, Schleife been-
den und QUIT
QUIT
```

Die Abb. 3.5 zeigt das gesamte Programm MAIL.CMD im Zusammenhang. Wenn Sie beim Lesen die Beispiele gleich eintippen, vergessen Sie

nicht die Datenbank POST.DBF und die Indexdateien NAMEN und PLZ zu erstellen. Die Beschreibung dazu finden Sie weiter vorn im Buch. Andernfalls geben Sie das MAIL-Programm ein (mit der Kennung CMD für CP/M-Rechner und der Kennung PRG für MS-DOS-Rechner). Zum Schreiben des Programms können Sie entweder MODIFY COMMAND oder ein Textprogramm benutzen.

```
************************************* MAIL.CMD
********************** Adressenverwaltung Hauptmenü
SET TALK OFF
SET DEFA TO B

********************** Datenbank und Index-Dateien einrichten
USE post INDEX namen,plz

********************** Schleife zur Anzeige des Menüs
STORE 0 TO wahl
DO WHILE wahl  #7
   ERASE
   TEXT
       HAUPTMENÜ ADRESSENVERWALTUNG

     1.  Neue Namen und Adressen eingeben
     2.  Reports drucken
     3.  Schnellsuchen
     4.  Namen und Adressen ändern
     5.  Namen und Adressen löschen
     6.  Duplikate suchen

     7.  ENDE
   ENDTEXT

************************ Benutzerwahl erfragen
   § 12,12 SAY 'Geben Sie Ihre Wahl (1-7) ein';
   GET wahl PICTURE "9"
   READ

********************** Verzweige zu dem entsprechenden Programm
DO CASE

   CASE wahl = 1
    DO append

   CASE wahl = 2
    DO reports
```

Abb. 3.5: Das Hauptmenü- Programm der Adressenverwaltung (MAIL.CMD)

```
    CASE wahl = 3
      DO lookup

    CASE wahl = 4
      DO edit

    CASE wahl = 5
      DO delnames

    CASE wahl = 6
      DO dupes

  ENDCASE

ENDDO (while wahl#7)

********************** Wenn wahl = 7, Schleife beenden und QUIT
QUIT
```

Abb. 3.5: Das Hauptmenü- Programm der Adressenverwaltung (MAIL.CMD) (Forts.)

Zusammenfassung

In diesem Kapitel haben wir Techniken erforscht, um ein Software-Projekt von der Stufe der ersten Idee in eine programmierbare Form zu bringen. Die Schritte zum Entwerfen und Entwickeln des Projektes sind:

1. das Ziel des Projektes definieren
2. die Struktur der Datenbank festlegen
3. die Struktur des Programm-Systems festlegen
4. jedes einzelne Programm entwerfen und entwickeln

Bisher haben wir ein Programm zur Adressenverwaltung entworfen, die Struktur der Datenbank und des Programm-Systems festgelegt und das Hauptmenü-Programm geschrieben.

Wir werden die restlichen Programme für unsere Adressenverwaltung in späteren Kapiteln entwickeln. Dabei beginnen wir mit einer Diskussion über bedienerfreundliche Masken für die Eingabe von neuen Namen und zur Veränderung bereits gespeicherter.

Benutzerfreundliche Dateneingabemasken

4

Bei einfachen Programmen erlauben die Befehle APPEND und EDIT, Daten in der Datenbank zu verändern oder neue einzugeben. Sobald Sie APPEND eintippen, zeigt Ihnen dBASE die Feldnamen der Datenbank zusammen mit den entsprechenden Feldgrößen und läßt Sie Daten eingeben, wie in Abb. 4.1 zu sehen.

Diese Art der Darstellung auf dem Bildschirm ist allerdings für einen Anfänger nicht leicht zu durchschauen. Wenn Sie Ihre eigenen Dateneingabemasken aufbauen, können Sie verständlichere Meldungen auf den Bildschirm bringen, wie zum Beispiel „NACHNAME" anstatt „NNAME". Sie können zusätzlich auch Informationen über die Befehle

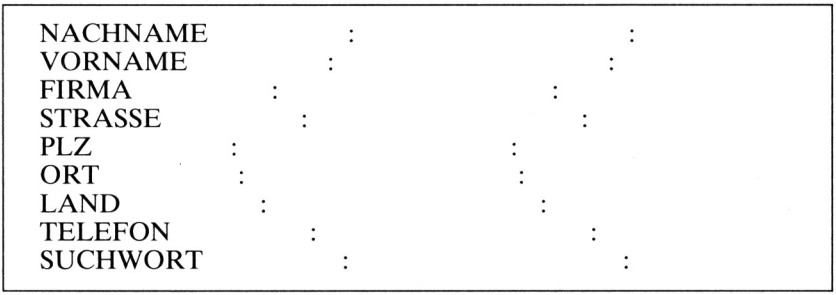


Abb. 4.1: Ein typischer APPEND-Bildschirm

zur Cursor-Steuerung anzeigen und noch einiges mehr. Lassen Sie uns einmal erarbeiten, wie die Befehle §, SAY, GET, READ und PICTURE angewendet werden, um eine Maske nach Ihren konkreten Vorstellungen zu gestalten.

Formatierte Masken erstellen

Für den Anfänger oder einen Gelegenheitsbenutzer sollte eine Maske zur Eingabe und zur Änderung von Daten einige kurze Angaben zur Handhabung des Cursors beinhalten. Die Abb. 4.2 zeigt die in unserem Programm verwendete Maske zum Ändern und Eingeben von Daten.

Es gibt drei grundsätzliche Möglichkeiten, Masken zu erstellen. Bei der ersten Methode benutzen Sie MODI COMM um die Datei zu eröffnen, schreiben dann die §-Zeichen und die Koordinaten (Zeilen- und Spaltennummern) für jede Bildschirmanzeige, die notwendigen GET-Befehle und den Namen des Feldes. Beachten Sie: Wenn Sie mit MODI COMM die Maske erstellen, müssen Sie unbedingt die Dateikennung FMT beim Eröffnen der Datei mit angeben. Beispiel: Sie erstellen eine Maske mit dem Namen AMASK1. Die Datei eröffnen Sie mit dem Befehl MODI COMM B:AMASK1.FMT, geben die notwendigen Programmzeilen ein (siehe Abb. 4.3) und sichern Ihre Arbeit mit ^W.

Als Alternative zu dieser ganzen Tipperei bietet ASHTON-TATE das Programm ZIP für das Betriebssystem CP/M. Für 16-Bit-Computer mit

```
Cursor-Kontrolle: (Das Symbol ^ bedeutet 'CTRL-Taste drücken...')
^E hoch : ^X runter : ^D rechts : ^S links : ^G loeschen : ^V einfuegen

Nachname :                    : Vorname :                :
Firma    :                 :
Strasse  :                    :
PLZ      :    :               Ort   :       : Land  :    :
Telefon  :         :
Suchwort :                    :
```

Abb. 4.2: Eingabemaske

```
§  1, 1 SAY "Cursor Kontrolle: (Das Symbol ^ bedeutet 'CTRL-Taste
                                                    drücken...')"
§  3, 1 SAY "^E hoch  :  ^X runter  : ^D rechts  : ^S links  : ^G
                                          loeschen : ^V einfuegen"
§  4, 1 SAY "================================================="
§  6, 1 SAY "Nachname"
§  6,12 GET nachname
§  6,37 SAY "Vorname"
§  6,46 GET vorname
§  8, 1 SAY "Firma"
§  8,12 GET firma
§ 10, 1 SAY "Strasse"
§ 10,12 GET strasse
§ 12, 1 SAY "PLZ"
§ 12,12 GET plz
§ 12,37 SAY "Ort"
§ 12,41 GET ort
§ 12,62 SAY "Land"
§ 12,67 GET land
§ 14, 1 SAY "Telefon"
§ 14,12 GET telefon
§ 16, 1 SAY "Suchwort"
§ 16,12 GET swort
```

Abb. 4.3: Das ZIP-Hauptmenü

dem Betriebssystem MS-DOS gibt es von der gleichen Firma ein ähnliches Programm mit dem Namen SED, allerdings nur in den USA. ZIP ermöglicht es Ihnen, Ihre Masken so auf den Bildschirm zu malen, wie sie später zur Eingabe von Änderungen oder neuen Daten erscheinen sollen. Dann schreibt das Programm automatisch die entsprechenden §-,SAY- und GET-Befehle.

Arbeiten mit ZIP

Um mit dem Programm ZIP eine Maske zu erstellen, stecken Sie die ZIP-Diskette in das Laufwerk A und Ihre Arbeitsdiskette in Laufwerk B. Hinter dem CP/M-Prompt A> geben Sie ZIP ein und starten mit RETURN. Danach sehen Sie ein Menü mit verschiedenen Optionen, wie in der Abb. 4.4 gezeigt.

Die Arbeit an der Maske beginnen Sie, indem Sie einfach RETURN eingeben. ZIP fragt dann:

```
<Neue> oder <Alte> Datei (Q für Ende)?
```

```
ZIP <tm>                    <c> 1982 Hal Pawluk
§ Wert anzeigen                 # Wert eingeben
() eingebettete Befehle
/<Tab>: Rand                    /C: Text zentrieren
/K: Lösche Zeile/Spalte         /A: zufügen Zeile/Spalte
/H: Horizontale Linie           /V: Vertikale Linie
/T: Oben /B: Unten              /M: Mitte
/D: Löschen    /I: Einfügen     /P: Vorherige Maske
/F: Erste      /L: Letzte       /N: Nächste Maske
/S: SICHERN    /E: Bildsch.     /Q: ENDE ohne sichern
                   löschen
T>abs:    5    V>ert:           H>oriz:
               M> Rand:   0

Geben Sie L zum Ändern, oder RETURN ein
```

Abb. 4.4: Das ZIP-Hauptmenü

Da es sich um eine neue Datei handelt, geben Sie N(eue) ein. Das Programm fragt dann:

```
Dateiname (Laufwerk optional):
```

Geben Sie hier einen Dateinamen samt der Laufwerksbezeichnung ein. Um die Maske AMASK1 auf dem Laufwerk B zu erstellen, geben Sie den Dateinamen so ein: B:AMASK1 und drücken dann RETURN. ZIP zeigt Ihnen nun einen leeren Bildschirm als Arbeitsfläche und am unteren Rand eine Hilfsanzeige, die Ihnen die Cursor-Position anzeigt:

```
Zeile 0, Spalte 0
```

Konstruieren Sie jetzt die Maske auf dem Bildschirm genauso, wie sie der Benutzer später sehen soll. An den Stellen, wo dBASE einen Feldinhalt oder den Inhalt einer Speichervariablen anzeigen soll, schreiben Sie das #-Symbol und den Namen der zu benutzenden Variablen oder des Feldes. Wenn Sie eine Maske wie in Abbildung 4.3 erstellen wollen, benutzen Sie Abb. 4.5 als Vorlage (mit /H können Sie mit einem Tastendruck eine horizontale Linie ziehen lassen).

Wissen Sie nicht mehr weiter beim Zeichnen Ihrer Maske, tippen Sie einfach zwei Schrägstriche ein (//) und rufen damit das ZIP-Hauptmenü auf. Mit RETURN kommen Sie wieder zu Ihrer Maske zurück.

Sind Sie mit der Maske fertig, geben Sie /S ein, um sie abzuspeichern. ZIP fragt dann:

```
Datei B:AMASK1 als (C)MD oder (F)MT-Datei sichern?
                                              (S=STOP)
```

```
Cursor-Kontrolle: (^ bedeutet: 'CTRL-Taste drücken...')
^E hoch : ^X runter : ^D rechts : ^S links : ^G loeschen : ^V einfuegen

Nachname   #nachname        Vorname   #vorname
Firma      #firma
Strasse    #strasse
PLZ        #plz Ort         Ort #ort          Land #land
Telefon    #telefon
Suchwort   #swort
```

Abb. 4.5: Ein ZIP-Bildschirm

Sie müssen hier angeben, ob ZIP eine CMD-Datei oder eine FMT-Datei schreiben soll (oder Sie geben S zum Aufhören ein). Für eine Eingabedatei wählen Sie F für die FMT-Datei. ZIP fragt Sie dann:

```
B:AMASK1: Wollen Sie den Namen ändern? (J/N)
```

Antworten Sie mit N (nein). Die letzte Frage zu der Maske lautet:

```
B:AMASK1: Druckformat erstellen? (J/N)
```

Wir haben die Maske entworfen, um neue Daten einzugeben, und nicht, um Daten auszudrucken. Geben Sie deshalb N (nein) ein. ZIP schreibt dann für Sie sowohl die Formatdatei AMASK1.FMT auf die Diskette im Laufwerk B wie auch die Dateien AMASK1.ZPR und AMASK1.ZIP, die Sie für spätere Änderungen wieder gebrauchen können.

Nachdem ZIP seine Arbeit erledigt hat, landen Sie wieder bei Ihrer Maske. Die Arbeit mit ZIP beenden Sie, indem Sie /Q eingeben. Das Programm vergewissert sich mit:

```
Zurück zum Betriebssystem (Ja oder Nein)?
```

Antworten Sie mit J (ja). Sie kommen dann wieder zum CP/M-Prompt A>.

Wenn Sie nun das von ZIP erstellte Formatprogramm begutachten möchten, laden Sie dBASE, gehen mit SET DEFAULT TO B auf das Lauf-

```
§  1, 1 SAY "Cursor Kontrolle:(^ bedeutet 'CTRL-Taste drücken...')"
§  3, 1 SAY "^E hoch  :  ^X runter  : ^D rechts  : ^S links  : ^G
                .                          loeschen : ^V einfuegen"
§  4, 1 SAY "================================================="
§  6, 1 SAY "Nachname"
§  6,12 GET nachname
§  6,37 SAY "Vorname"
§  6,46 GET vorname
§  8, 1 SAY "Firma"
§  8,12 GET firma
§ 10, 1 SAY "Strasse"
§ 10,12 GET strasse
§ 12, 1 SAY "PLZ"
§ 12,12 GET plz
§ 12,37 SAY "Ort"
§ 12,41 GET ort
§ 12,62 SAY "Land"
§ 12,67 GET land
§ 14, 1 SAY "Telefon"
§ 14,12 GET telefon
§ 16, 1 SAY "Suchwort"
§ 16,12 GET swort
```

Abb. 4.6: Die Formatdatei AMASK1.FMT, von ZIP erstellt

werk B und geben dann ein MODI COMM AMASK1.FMT. Sie sollten dann die Formatdatei sehen, wie sie auch in Abb. 4.6 gezeigt wird.

Mit ZIP kann man leicht arbeiten, man braucht nur etwas Übung. Denken Sie an die beiden Schrägstriche, wenn Sie einmal einen ZIP-Befehl vergessen haben sollten. Ausführlichere Informationen bietet Ihnen der Abschnitt über ZIP in Ihrem dBASE-Handbuch (wenn Ihre dBASE-Version ZIP enthält).

FMT-Dateien benutzen

Haben Sie erst einmal eine Formatdatei (FMT) erstellt, ist ihr Gebrauch ganz einfach. In die bereits erstellte Datenbankdatei POST.DBF geben Sie neue Daten ein, indem Sie dBASE ganz normal laden und Laufwerk B anmelden (oder das Laufwerk, mit dem Sie arbeiten wollen). Wenn der dBASE-Prompt erscheint, achten Sie darauf, daß Sie eingeben:

```
USE post INDEX namen,plz
```

Wieder ist es sehr wichtig, beide Index-Dateien mitanzugeben, weil wir

nämlich neue Daten eingeben werden und beide Indizes auf dem neuesten Stand bleiben sollen. Um ein paar neue Daten einzugeben, tippen Sie ein:

```
SET FORMAT TO amask1
```

und dann den üblichen Befehl:

```
APPEND
```

Die Eingabe funktioniert genau wie mit APPEND, nur daß jetzt die in Abb. 4.7 gezeigte Maske benutzt wird.

Zusätzlich können Sie SET FORMAT TO amask1 und dann EDIT oder READ eingeben, um bereits vorhandene Daten mit Hilfe der Maske zu verändern.

Nachdem Sie die Arbeit mit der Maske AMASK1 beendet haben, sollten Sie dBASE wieder in seinen normalen Zustand zurückversetzen, was mit folgendem Befehl geschieht:

```
SET FORMAT TO SCREEN
```

Andernfalls versucht dBASE vielleicht fälschlicherweise, die Maske zusammen mit zukünftigen APPEND-, READ- oder EDIT-Befehlen zu benutzen. Das Kommando SET FORMAT TO SCREEN bringt dBASE wieder in den normalen Bildschirmmodus zurück.

```
Cursor Kontrolle: (Das Symbol ^ bedeutet 'CTRL-Taste drücken...')

^E hoch : ^X runter : ^D rechts : ^S links : ^G loeschen : ^V einfuegen

Nachname :                    : Vorname :                    :

Firma    :                    :

Strasse  :                      :

PLZ      :     :                Ort   :          : Land  :     :

Telefon  :           :

Suchwort :                     :
```

Abb. 4.7: Maske für Eingabe nach SET FORMAT TO amask1

Eingabeformate festlegen

Wir können die eben erstellte Maske noch verbessern, wenn wir zum Beispiel für das Feld Telefon ein Eingabeformat festlegen. In der Art wie dieses Feld innerhalb der Maske jetzt definiert ist, erscheint es dem Benutzer so:

```
Telefon :              :
```

wobei wir davon ausgehen, daß er die dreizehn möglichen Ziffern in einem selbstdefinierten Format, etwa 069-555-1212, eingibt. Wir können ihm Zeit sparen helfen, wenn wir die Striche oder etwas anderes bereits im Feld vorgeben. Angenommen, ein Verein hat nur Mitglieder aus der Stadt Frankfurt (Telefonvorwahl 069), und wir wollen die Maske so ändern, daß die Telefonnummer bequemer eingegeben werden kann. Wir lassen das Feld so in der Maske erscheinen:

```
Telefon :(    )          :
```

Diese Art der Eingabebeeinflussung sichert uns schon einmal teilweise ein einheitliches Format der Telefonnummern. Um ganz sicher zu gehen, wollen wir aber dafür sorgen, daß nur Ziffern und nichts anderes eingege-

```
§  1, 1 SAY "Cursor Kontrolle:(^ bedeutet 'CTRL-Taste drücken...')"
§  3, 1 SAY "^E hoch  :  ^X runter  : ^D rechts  : ^S links  : ^G
                                          loeschen : ^V einfuegen"
§  4, 1 SAY "================================================="
§  6, 1 SAY "Nachname"
§  6,12 GET nachname
§  6,37 SAY "Vorname"
§  6,46 GET vorname
§  8, 1 SAY "Firma"
§  8,12 GET firma
§ 10, 1 SAY "Strasse"
§ 10,12 GET strasse
§ 12, 1 SAY "PLZ"
§ 12,12 GET plz
§ 12,37 SAY "Ort"
§ 12,41 GET ort
§ 12,62 SAY "Land"
§ 12,67 GET land
§ 14, 1 SAY "Telefon"
§ 14,12 GET telefon   PICTURE "(999)99999999
§ 16, 1 SAY "Suchwort"
§ 16,12 GET swort
```

Abb. 4.8: Das von ZIP erstellte Programm mit PICTURE-Befehl

ben werden können. Dafür werden wir die Datei AMASK1.FMT ändern. Keine Angst, das ist nicht schwierig. Sie können entweder MODI COMM B:AMASK1.FMT oder WordStar mit seiner N-Option verwenden, um die Datei anzupassen. Im Programmtext schreiben Sie hinter GET telefon das Wort PICTURE sowie die Klammern und die zehn 9er, wie Sie es in der Abb. 4.8 sehen.

Die Klammern erscheinen nun automatisch auf dem Bildschirm und werden auch in der Datenbank gespeichert, ohne daß der Benutzer sie eingeben muß. Die Reihe der 9er in der PICTURE-Anweisung sorgt dafür, daß nur Ziffern (0–9) als Eingabe in dieses Feld angenommen werden, was die Fehlerwahrscheinlichkeit deutlich vermindert.

Die Programmdatei für die Eingabemaske

Der Aufbau unseres Adressensystems verlangt eine Programmdatei mit dem Namen APPEND, um den Benutzer über die Maske neue Namen und Adressen eingeben zu lassen. Nachdem Sie die Datei AMASK1.FMT fertig haben, müssen Sie im nächsten Schritt die Programmdatei APPEND erstellen. Das ist wirklich ganz einfach. Die komplette Befehlsdatei sehen Sie in Abb. 4.9.

```
************************************ APPEND.CMD
**** Neue Daten über die Maske AMASK1.FMT eingeben

SET FORMAT TO amask1
APPEND
SET FORMAT TO SCREEN

RETURN
```

Abb. 4.9: Das Programm APPEND für die Adressenverwaltung

Sie können das Programm (nachdem Sie die Formatdatei AMASK1.FMT erstellt haben) mit dem Befehl MODI COMM B:APPEND oder mit Ihrem Textsystem schreiben. Zum Ausprobieren sagen Sie dBASE aber nicht DO APPEND, sondern DO POST. Sobald das Hauptmenü der Adressenverwaltung auf dem Bildschirm erscheint, wählen Sie die Option 1, um neue Daten einzugeben. Sie sollten danach Ihre Eingabemaske so sehen, wie sie in der Abb. 4.10 gezeigt wird.

Wenn Sie bei unseren Diskussionen die jeweiligen Dateien immer gleich mitaufbauen, können Sie das Programm jetzt testen, indem Sie einige

```
Cursor-Kontrolle: (^ bedeutet: 'CTRL-Taste drücken...')

^E hoch : ^X runter : ^D rechts : ^S links : ^G loeschen : ^V einfuegen

Nachname :                    : Vorname :                :

Firma    :                 :

Strasse  :                    :

PLZ      :    :               Ort   :       : Land  :    :

Telefon  :         :

Suchwort :                    :
```

Abb. 4.10: Maske für Eingabe neuer Namen und Adressen

neue Namen und Adressen in die Datenbank eingeben. Sind Sie fertig, geben Sie RETURN anstelle des Nachnamens ein. Der SET FORMAT TO SCREEN-Befehl im APPEND-Programm wird dBASE wieder in den normalen Bildschirmmodus zurückversetzen, und das RETURN übergibt die Kontrolle wieder an das Programm POST.CMD, das Ihnen dann das Hauptmenü zeigt. Dort angelangt, wählen Sie die Option 7 für ENDE, damit wir die fehlenden Programme noch entwickeln können.

Zusammenfassung

In diesem Kapitel haben wir die Techniken zur Maskenerstellung in dBASE II diskutiert. Die §-, SAY-, GET- und PICTURE-Befehle können dazu benutzt werden, den Bildschirm in beliebiger Weise zu gestalten. Masken können sehr schnell mit dem Zusatzprogramm ZIP (für CP/M) erstellt werden. Ist die Maske fertig, wird sie mit SET FORMAT TO gegenüber dBASE benannt und die Befehle READ, APPEND und EDIT können benutzt werden, um sie auf den Bildschirm zu bringen. Der Befehl SET FORMAT TO SCREEN löscht die Maske und stellt den normalen dBASE-Bildschirmmodus wieder her.

Wir haben auch das Programm APPEND geschrieben, um über das Hauptmenü der Adressenverwaltung neue Daten in die Datenbank eingeben zu können.

Menügeführtes Suchen und Sortieren

5

dBASE II bietet zwar großartige Möglichkeiten zum Sortieren und Suchen innerhalb einer Datenbank an, aber diese erfordern doch ausführliche Kenntnisse über die Syntax der dBASE II-Sprache. Da wir davon ausgehen, daß die Anwender unseres Systems unerfahrene dBASE II-Benutzer sind, vereinfachen wir die Sortier- und Suchprozeduren, indem wir sie zu Optionen im Menü machen. In unserem Adressenverwaltungssystem kümmert sich das Programm REPORTS um diese Aufgaben und fragt nach, welchen Report der Benutzer wünscht (Etiketten, Inhaltsverzeichnis oder eine MailMerge-Datei).

Das REPORTS-Programm

Das Programm REPORTS erledigt vier Hauptaufgaben:

1. Frage, welcher Report gewünscht wird
2. Frage, welche Sortierung gewünscht wird
3. Frage, welche Suchkriterien gewünscht werden
4. Verzweige zu dem entsprechenden Reportprogramm und drucke die Daten zu den Sortier- und Suchkriterien aus

Entscheidet sich der Benutzer für die Option 2 des Hauptmenüs (Reports drucken), sieht er zuerst ein Menü mit Optionen zu den verschieden Reportformen:

```
        REPORTS DRUCKEN

1.      Etiketten
2.      Inhaltsverzeichnis
3.      MailMerge-Datei

        Geben Sie den Reporttyp an (1-3) : :
```

Dann entscheidet er sich für eine dieser Optionen, indem er die entsprechende Ziffer eintippt. Der Bildschirm zeigt daraufhin ein anderes Menü, in dem er sich die gewünschte Sortierung aussuchen kann:

```
        GEWÜNSCHTE SORTIERUNG

1.      Alphabetisch (nach Nachnamen)
2.      Für Serienbriefe (nach Postleitzahlen)
3.      Nach Orten
4.      Nach Ländern
5.      Tatsächliche Reihenfolge

        Welche Sortierung wünschen Sie (1-5) : :
```

Wählt der Benutzer die Option 1, bekommt er die Daten in alphabetischer Reihenfolge, nach Nachnamen und zusätzlich nach Vornamen, präsentiert. Bei der 2. Option erscheinen die Daten in der aufsteigenden Reihenfolge der Postleitzahlen usw.

Aus dem nächsten Menü soll sich der Benutzer dann die Suchkriterien heraussuchen:

```
        SUCHKRITERIEN

1.      Nachname
2.      Postleitzahl
3.      Suchwort
4.      Alle Namen und Adressen drucken

        Geben Sie Ihr Kriterium an (1-4) : :
```

Sollte er sich für die Option 1 entscheiden (suche nach Namen), wird er gefragt:

```
Wen suchen?
```

An dieser Stelle kann er jetzt einen Namen eingeben, nach dem gesucht werden soll. Gibt er zum Beispiel Schmidt ein, werden nur Personen mit

diesem Nachnamen zum Report geschickt. Entscheidet sich der Benutzer für die Option 3, wird er gefragt:

```
Nach welchem Suchwort suchen?
```

Er kann dann sein Suchwort eingeben. Gibt er zum Beispiel GEKÜNDIGT 10/84 ein, erscheinen nur Namen und Adressen mit dem Suchwort GEKÜNDIGT 10/84 im Report. Mit der Option 4 schickt er alle Namen und Adressen der Datenbank in den Report.

Diese schlichte Reihe von Menüs ermöglicht dem Benutzer eine große Flexibilität. Möchte er beispielsweise einen Serienbrief an alle Mitglieder schicken, deren Mitgliedschaft in 10/84 endet, kann er die MailMerge-Datei als Reportoption, die Postleitzahl als Sortieroption (für Serienbriefe) und GEKÜNDIGT 10/84 als Suchwort eingeben. Das Programm erstellt dann eine nach Postleizahlen sortierte MailMerge-Datei aller Personen, deren Mitgliedschaft in 10/84 endet. Die dazugehörenden Etiketten werden gedruckt, indem er Etiketten als Reportoption eingibt, Postleitzahl als Sortierung und wieder GEKÜNDIGT 10/84 als Suchkriterium. So kommen die Briefe und die Etiketten in derselben Reihenfolge aus dem Drucker, was die Zusammenführung der beiden spürbar erleichtert.

Den Pseudocode für das Programm REPORTS sehen Sie in Abb. 5.1.

Lassen Sie uns jetzt mit dem Schreiben des Programms beginnen. Die erste Aufgabe ist, ein Menü mit den Reportoptionen zu zeigen und auf eine Antwort zu warten. Die üblichen Kommentare, den Programmnamen und die Programmaufgaben betreffend, stehen im Programmkopf wie hier:

```
********************************** REPORTS.CMD
*************** Benutzer nach Reporttyp, Sortierung
*************** und Suchkriterien fragen, dann zu
*************** dem entsprechenden Report verzweigen

ERASE
STORE " " TO rwahl
TEXT
                    REPORTS DRUCKEN

               1.   Etiketten
               2.   Inhaltsverzeichnis
               3.   MailMerge-Datei
ENDTEXT
§  8,18 SAY "Geben Sie den Reporttyp an (1-3) " GET
                                                rwahl
READ
```

Dieses Menü ist geradeheraus, ohne Schnörkel.

```
Bildschirm löschen
Menü mit Reportoptionen zeigen
REPORTS DRUCKEN

    1. Etiketten
    2. Inhaltsverzeichnis
    3. MailMerge-Datei

Nach gewünschtem Report fragen, Antwort in RWAHL speichern

Dann Menü mit Sortieroptionen zeigen

    GEWÜNSCHTE SORTIERUNG

    1. Alphabetisch (nach Nachnamen)
    2. Für Serienbriefe (nach Postleitzahlen)
    3. Nach Orten
    4. Nach Ländern
    5. Tatsächliche Reihenfolge

Sortierung erfragen und Antwort in SORTIER speichern

Sortierung entsprechend der Eingabe einleiten

    Wenn die 1 gewählt wurde
    Post mit Index Namen benutzen

    Wenn die 2 gewählt wurde
    Post mit Index PLZ benutzen

    Wenn die 3 gewählt wurde
    Zwischenindex Ort erstellen

    Wenn die 4 gewählt wurde
    Zwischenindex Land erstellen

    Andernfalls
    Post ohne Index benutzen
```

Abb. 5.1: Pseudocode für das Programm REPORTS

```
Menü der Suchoptionen zeigen

    SUCHKRITERIEN

    1. Nachname
    2. Postleitzahl
    3. Suchwort
    4. Alle Namen und Adressen drucken

Auf Antwort warten und in SUCHEN speichern

Suchkriterien nach Benutzerangaben formulieren

    Wenn die 1 gewählt wurde
    Mache Nachname zum Suchfeld
    Frage, nach wem zu suchen ist
    Mache die Antwort zur Suchbedingung

    Wenn die 2 gewählt wurde
    Mache PLZ zum Suchfeld
    Frage nach der zu suchenden Postleitzahl
    Mache die Antwort zur Suchbedingung

    Wenn die 3 gewählt wurde
    Erzeuge eine „immer richtig"-Bedingung
    so werden alle Datensätze gedruckt

Dann verzweige zu dem gewünschten Programm
entsprechend den Benutzerangaben (RWAHL)

    Wenn Reportoption 1
    Drucke Etiketten

    Wenn Reportoption 2
    Drucke Inhaltsverzeichnis

    Wenn Reportoption 3
    Erzeuge MailMerge-Datei

Sobald der Report fertig ist

Nachsehen, ob Zwischenindex erstellt wurde
    Wenn ja, löschen
```

Abb. 5.1: Pseudocode für das Programm REPORTS (Forts.)

Indexdateien Namen und PLZ vor Rückkehr
zum Hauptmenü aktivieren

Speichervariablen löschen

Zurück zum Hauptmenü

Abb. 5.1: Pseudocode für das Programm REPORTS (Forts.)

Menügesteuertes Sortieren

Als nächstes werden Sie eine Routine schreiben, die ein Menü der Sortierungsmöglichkeiten anbietet. Wie das geht, sehen Sie hier:

```
********************** Nach der Sortierung fragen
ERASE
STORE " " TO sortier
TEXT

          GEWÜNSCHTE SORTIERUNG

     1.   Alphabetisch (nach Nachnamen)
     2.   Für Serienbriefe (nach Postleitzahlen)
     3.   Nach Orten
     4.   Nach Ländern
     5.   Tatsächliche Reihenfolge

ENDTEXT
§ 10,11 SAY "Welche Sortierung wünschen Sie (1-5)"
                                          GET sortier
READ
```

Das war doch wieder einfach, oder? Ihre nächste Aufgabe besteht darin, die Sortierung nach den Eingaben des Benutzers einzuleiten. Die folgende DO CASE-Anweisung erledigt das:

```
****** Sortierung nach Benutzerwunsch einleiten
ERASE
DO CASE

   CASE sortier = "1"
     SET INDEX TO namen

   CASE sortier = "2"
     SET INDEX TO plz
```

```
    CASE sortier = "3"
      ? "Bitte warten ... ich sortiere"
      INDEX ON ort TO temp

    CASE sortier = "4"
      ? "Bitte warten ... ich sortiere"
      INDEX ON land TO temp

    OTHERWISE
      USE post
ENDCASE
```

Da die Indexdateien NAMEN und PLZ in diesem System immer auf dem neuesten Stand sind, benutzt das Programm den Befehl SET INDEX TO namen, wenn der Benutzer eine alphabetische Reihenfolge wünscht. Soll es in der Reihenfolge der Postleitzahlen sein, benutzt es den Befehl SET INDEX TO plz. Auf diesem Wege muß der Benutzer nicht erst lange auf eine Neusortierung warten, weil die Daten bereits sortiert vorliegen.

Wir haben dem Benutzer auch zugestanden, entweder nach ORT oder nach LAND sortieren zu lassen. Wir haben dafür keine Indexdateien eingerichtet, also müssen Sie bei Bedarf erstellt werden. Da es einige Zeit dauert, Indexdateien aufzubauen, erscheint

```
Bitte warten ... ich sortiere
```

auf dem Bildschirm. Wenn die ursprüngliche Reihenfolge beibehalten werden soll, werden natürlich die POST-Daten ohne Index-Datei verwendet.

Beachten Sie die Gestaltung des hier verwendeten Programms. Da dies eine Adressenverwaltung ist, können wir ziemlich sicher sein, daß der Benutzer die Sortierungen nach Namen und Postleitzahlen ziemlich regelmäßig brauchen wird. Aus diesem Grund führt das System zwei Index-Dateien für diese Anforderungen und vermeidet damit, daß nach jedem Zugriff auf die Datenbank neu sortiert werden muß. Die Chance ist ziemlich gering, daß eine Sortierung nach ORT oder LAND verlangt werden wird. Wir bieten die Möglichkeiten zwar an, lassen aber das Programm die Index-Dateien erst dann erstellen, wenn sie gebraucht werden.

Menügesteuertes Suchen

Als nächstes muß das REPORTS-Programm die Suchkriterien festlegen. In den hier folgenden Zeilen präsentiert das Programm ein Menü der

Suchkriterien und wartet auf den Benutzer, bis er seine Wahl eingegeben hat.

```
********************** Nach Suchkriterien fragen
ERASE
STORE " " TO suchen
TEXT

                    SUCHKRITERIEN

                1.  Nachname
                2.  Postleitzahl
                3.  Suchwort
                4.  Alle Namen und Adressen drucken
ENDTEXT

§  8,13 SAY "Geben Sie Ihr Kriterium an (1-4)" GET
                                                  suchen
READ
```

Nachdem klar ist, nach welchem Feld gesucht werden soll, braucht das Programm noch eine Eingabe über die zu suchende Information. Das bedeutet: Entscheidet sich der Benutzer, nach einem bestimmten Suchwort zu suchen, braucht das Programm dieses Suchwort, um suchen zu können. Es speichert dann diese Antworten in Speichervariablen, die SUCHEN und BEDINGUNG heißen. Die hier folgende CASE-Anweisung kümmert sich um diese Aufgabe:

```
********************** Suchbedingungen aufstellen
DO CASE

   CASE suchen = "1"
     STORE "!(Nachname)" TO suchen
     ACCEPT "Nach wem suchen?" TO bedingung
     STORE !(bedingung) TO bedingung

   CASE suchen = "2"
     STORE "PLZ" TO suchen
     ACCEPT "Nach welcher Postleitzahl suchen?" TO
                                          bedingung

   CASE suchen = "3"
     STORE "!(swort)" TO suchen
     ACCEPT "Nach welchem Suchwort suchen?" TO
                                       bedingung
     STORE !(bedingung) TO bedingung
```

```
    CASE suchen = "4"
      STORE "X" TO suchen
      STORE 1 TO x,bedingung

ENDCASE
```

Lassen Sie uns ein Beispiel dieses Konzepts einmal näher betrachten. Wählt der Benutzer die Option 3 (Suchen nach Suchwort), wird der Ausdruck „!(swort)" in die Speichervariable SUCHEN gespeichert. Dann erscheint die Frage auf dem Bildschirm:

```
Nach welchem Suchwort suchen?
```

und die Antwort darauf kommt in die Variable BEDINGUNG. Anschließend wird die Benutzerantwort in Großbuchstaben in BEDINGUNG abgelegt. Wenn also der Benutzer nach einem Suchwort suchen möchte und dazu „gekündigt 10/84" eingibt, werden zwei Speichervariablen mit diesem Inhalt erstellt:

```
Speichervariable              Inhalt

SUCHEN                  =     "!(swort)"
BEDINGUNG               =     "GEKÜNDIGT 10/84"
```

Wenn später der Report gedruckt wird, entscheidet das Programm, ob es einen Datensatz druckt oder nicht, mit der Anweisung:

```
IF &suchen = bedingung
```

Nachdem dBASE die Variablen durch die tatsächlichen Begriffe ersetzt hat, liest sich die Entscheidungsbedingung so:

```
IF !(swort) = GEKÜNDIGT 10/84
```

Deshalb werden von dem IF-Ausdruck nur Datensätze gewählt, in denen nach der Umwandlung in Großbuchstaben GEKÜNDIGT 10/84 steht.

Eine Schwierigkeit besteht noch. Sollen alle Namen und Adressen angezeigt werden, darf die IF-Konstruktion keinen ausschließen. Das heißt, die IF-Bedingung muß immer wahr sein. Die Menüoption 4 bearbeitet diesen Auftrag. Möchte der Benutzer alle Personen in der Datei sehen, erstellt das Programm diese Speichervariablen:

```
Speichervariable              Inhalt

SUCHEN                  =     "X"
BEDINGUNG               =      1
X                       =      1
```

Die IF-Bedingung für die Entscheidung lautet: IF &suchen = bedingung. Sobald dBASE II das Makro ersetzt, sieht die IF-Bedingung so aus:

```
IF X = 1
```

Weiter oben können wir sehen, daß in der Variablen X eine 1 gespeichert ist. So wird aus der IF-Bedingung

```
IF 1 = 1
```

Da bekanntlich 1 = 1 immer richtig ist, läßt die IF-Anweisung keinen Datensatz aus. Damit landen alle Einträge der Datenbank im Report.

Im nächsten Kapitel werden wir die Suchbedingungen im Reportprogramm deutlicher erkennen. Für jetzt merken Sie sich, daß das REPORTS-Programm die richtige Sortierung für die Datenbank erstellt und veränderliche Informationen in den Speichervariablen SUCHEN und BEDINGUNG bereithält. SUCHEN enthält den zu suchenden Feldnamen, und BEDINGUNG enthält den zu suchenden Eintrag.

Die letzte Aufgabe des REPOTRS-Programms besteht noch in der Verzweigung zu dem entsprechenden Programm, um den vom Benutzer gewünschten Report zu drucken. Erinnern Sie sich, daß die Wahl des Benutzers in der Speichervariablen RWAHL gespeichert wurde. Die folgende DO CASE-Anweisung übernimmt die Verzweigung:

```
****************** Verzweige zu dem entsprechenden
****************** Programm, abhängig von der Wahl
****************** des Benutzers (RWAHL)

DO CASE
   CASE rwahl = "1"
     *************** Drucke Etiketten
     DO LABELS

   CASE rwahl = "2"
     *************** Drucke Inhaltsverzeichnis
     DO DIRECTOR

   CASE rwahl = "3"
     *************** Erstelle MailMerge-Datei
     DO MMERGE

ENDCASE
```

Das Programm LABELS druckt die Etiketten, DIRECTOR sorgt für das Inhaltsverzeichnis und MMERGE erstellt die MailMerge-Datei. Diese Programme werden alle im nächsten Kapitel beschrieben.

Jedes dieser Programme übergibt nach Beendigung seiner Arbeit die Kontrolle wieder an das REPORTS-Programm, das nachsieht, ob ein Zwischenindex erstellt wurde. Wenn ja, löscht es diesen. Diese Zeilen werden die Zwischendatei löschen:

```
********************* Lösche TEMP.NDX, wenn vorhanden
IF FILE("temp.ndx")
   DELE FILE temp.ndx
ENDIF (temp.ndx vorhanden)
```

Anschließend werden noch die in diesem Programm verwendeten Speichervariablen gelöscht:

```
************************ Speichervariablen löschen
RELEASE rwahl,sortier,suchen,bedingung
```

Dann aktiviert das Programm wieder die beiden Index-Dateien NAMEN und PLZ, damit alle zukünftigen Änderungen oder Neueinträge in die Datei POST in beiden Index-Dateien sofort vermerkt werden. Diese Zeilen kümmern sich darum und übergeben die Kontrolle dann wieder dem Hauptmenü:

```
*************** Index-Dateien NAMEN und PLZ wieder
*************** aktivieren, bevor es zum Hauptmenü
*************** zurück geht

USE post INDEX namen,plz
RETURN
```

Das kurze REPORTS-Programm finden Sie in Abb. 5.2.

```
********************************* REPORTS.CMD
********************** Benutzer nach Reporttyp, Sortierung
********************** und Suchkriterien fragen, dann zu
********************** dem entsprechenden Report verzweigen

ERASE
STORE " " TO rwahl
TEXT
                      REPORTS DRUCKEN

               1.   Etiketten
               2.   Inhaltsverzeichnis
               3.   MailMerge-Datei
```

Abb. 5.2: Die Programmdatei REPORTS.CMD

```
ENDTEXT
§  8,18 SAY "Geben Sie den Reporttyp an (1-3) " GET rwahl
READ

************************** Nach der Sortierung fragen
ERASE
STORE " " TO sortier
TEXT


          GEWÜNSCHTE SORTIERUNG

     1.   Alphabetisch (nach Nachnamen)
     2.   Für Serienbriefe (nach Postleitzahlen)
     3.   Nach Orten
     4.   Nach Ländern
     5.   Tatsächliche Reihenfolge

ENDTEXT
§ 10,11 SAY "Welche Sortierung wünschen Sie (1-5)" GET sortier
READ
****** Sortierung nach Benutzerwunsch einleiten
ERASE
DO CASE

   CASE sortier = "1"
     SET INDEX TO namen

   CASE sortier = "2"
     SET INDEX TO plz

   CASE sortier = "3"
     ? "Bitte warten ... ich sortiere"
     INDEX ON ort TO temp

   CASE sortier = "4"
     ? "Bitte warten ... ich sortiere"
     INDEX ON land TO temp

   OTHERWISE
     USE post
ENDCASE

************************ Nach Suchkriterien fragen
ERASE
STORE " " TO suchen
TEXT
```

Abb. 5.2: Die Programmdatei REPORTS.CMD (Forts.)

```
                    SUCHKRITERIEN

                1.   Nachname
                2.   Postleitzahl
                3.   Suchwort
                4.   Alle Namen und Adressen drucken
ENDTEXT
§  8,13 SAY "Geben Sie Ihr Kriterium an (1-4)" GET suchen
READ


**************************** Suchbedingungen aufstellen
DO CASE

   CASE suchen = "1"
     STORE "!(Nachname)" TO suchen
     ACCEPT "Nach wem suchen?" TO bedingung
     STORE !(bedingung) TO bedingung

   CASE suchen = "2"
     STORE "PLZ" TO suchen
     ACCEPT "Nach welcher Postleitzahl suchen?" TO bedingung

   CASE suchen = "3"
     STORE "!(swort)" TO suchen
     ACCEPT "Nach welchem Suchwort suchen?" TO bedingung
     STORE !(bedingung) TO bedingung

   CASE suchen = "4"
     STORE "X" TO suchen
     STORE 1 TO x,bedingung

ENDCASE


************************** Verzweige zu dem entsprechenden
************************** Programm, abhängig von der Wahl
************************** des Benutzers (RWAHL)

DO CASE
   CASE rwahl = "1"
     *************** Drucke Etiketten
     DO LABELS

   CASE rwahl = "2"
     *************** Drucke Inhaltsverzeichnis
     DO DIRECTOR
```

Abb. 5.2: Die Programmdatei REPORTS.CMD (Forts.)

```
   CASE rwahl = "3"
     ********** Erstelle MailMerge-Datei
     DO MMERGE

ENDCASE

************************** Lösche TEMP.NDX, wenn vorhanden
USE
IF FILE("temp.ndx")
   DELE FILE temp.ndx
ENDIF (temp.ndx vorhanden)

********************************** Speichervariablen löschen
RELEASE rwahl,sortier,suchen,bedingung

****************** Index-Dateien NAMEN und PLZ wieder
****************** aktivieren, bevor's zum Hauptmenü
****************** zurück geht

USE post INDEX namen,plz
RETURN
```

Abb. 5.2: Die Programmdatei REPORTS.CMD (Forts.)

Zusammenfassung

In diesem Kapitel haben Sie einige Techniken gesehen, wie man Such- und Sortieroperationen aus einem Programm heraus bewerkstelligt. Das Ziel ist hier, es einem unerfahrenen Benutzer bei der Eingabe von Sortier- oder Suchkriterien ganz leicht zu machen. Dazu muß er keinen einzigen dBASE-Befehl kennen. Im nächsten Kapitel werden wir uns über benutzerfreundliche Reports unterhalten, die ihre Daten in der vom Benutzer angegebenen Sortierung ausgeben und die angezeigten Daten auf die beschränken, die bestimmte Suchkriterien erfüllen.

Berichte

6

Der dBASE II-Befehl REPORT FORM ermöglicht die Erstellung von Berichten. Allerdings, in vielen Situationen wird der REPORT FORM-Befehl nicht ausreichen, besonders wenn die Datenbank mit mehrfachen oder sehr langen Feldern arbeitet. In diesen Fällen müssen wir ein spezielles Programm schreiben, das uns den Bericht ausdruckt. Dieses Programm muß sich um die Seitennummern, den Seitenvorschub, die Wartepause auf dem Bildschirm usw. kümmern. Alle diese Notwendigkeiten werden in diesem Kapitel besprochen.

Bildschirm- und Druckformatierung

Die Datenbank unseres POST-Programms verbraucht insgesamt 154 Zeichen. Benutzen wir den REPORT FORM-Befehl, um einen Bericht im 80-Spalten-Format zu drucken, müssen wir viele Felder auf zwei oder drei Stücke aufteilen, damit sie auf das Papier passen. Als Alternative können wir ein Programm schreiben, das den gesamten Datensatz jedes Individuums in zwei einzelne Zeilen aufteilt. Das Programm DIRECTOR druckt das Inhaltsverzeichnis des Systems in dem in Abb. 6.1 gezeigten Format.

Wünscht der Benutzer den Bericht ausgedruckt, erscheint dieser seitenweise mit Überschrift und Seitenzahl im Kopf des Berichts, wie in Abb. 6.1 zu sehen. Soll der Bericht nur auf dem Bildschirm angezeigt werden,

	Inhaltsverzeichnis	Seite 1
Adam, Andy 5000 Köln	ABC GmbH 0221-12345	Oxfordstr. 123 GEKÜNDIGT 01/85
Anderson, Ruth 8000 München	XYZ & Co 089-987654	Baumallee 17 GEKÜNDIGT 03/86
Bouvier, Karl 4000 Düsseldorf	Buch AG 0211-765234	Buchstr. 100 PROSPEKT
Carlson, Clara 5205 St. Augustin 1	Wunder GmbH & Co 02241-339087	Hangweg 21 PROSPEKT

Abb. 6.1: Inhaltsverzeichnis mit dem Programm DIRECTOR

kommt immer nur ein Bildschirm voll zur Anzeige. Ist der Bildschirm voll, erscheint diese Nachricht am Fuße des Bildschirms:

```
Weiter mit beliebiger Taste (mit R zum Menü zurück)
```

Diese Methode bietet zwei Vorteile. Erstens hat der Betrachter genügend Zeit, um den Bildschirminhalt zu lesen, bevor irgendwelche Namen am oberen Rand ungelesen verschwinden. Zweitens kann er auf die Taste R drücken, wenn er das Gesuchte gefunden hat, und das Inhaltsverzeichnis verlassen. Auf diese Art und Weise muß er nicht jeden Namen der Datei ansehen (oder jeden mit einem bestimmten Suchwort).

Denken Sie daran, daß das Programm DIRECTOR von dem Programm REPORTS aufgerufen wird. Der zum REPORTS-Programm gehörende Index sorgt für ein Inhaltsverzeichnis mit sortierten Daten. Das

Bildschirm löschen

Variablen für Titel, Seitenzahl und Zeilenzähler einrichten

Ausdruck erwünscht ? (Antwort in Variable LP speichern)

Wenn Ausdruck erwünscht, Benutzer den Drucker einschalten lassen
dann SET PRINT ON

Abb. 6.2: Pseudocode für das DIRECTOR-Programm

```
Bildschirm löschen
Titel und Seitennummer drucken

Am Anfang der Datenbank beginnen
Durchlaufe jeden Datensatz bis zum letzten oder bis der Benutzer
    den Abbruch wünscht

    Überprüfe, ob Datensatz = BEDINGUNG
    Wenn Datensatz = BEDINGUNG
        drucke NAME, FIRMA, ADRESSE
        drucke PLZ, ORT, TELEFON, SWORT
        drucke Leerzeile
        erhöhe Zeilenzähler um 3

    Wenn Bericht zum Drucker geht und über 55 Zeilen
        gedruckt sind:
        beginne neue Seite
        erhöhe Zeilenzähler um 1
        drucke Titel und Seitennummer
        setze Zeilenzähler auf 0

    Wenn Bericht nicht gedruckt wird und mehr als 18 Zeilen
        angezeigt sind:
        Frage nach nächster Seite (oder Ende)
        warte auf Antwort
        Wenn Benutzer nicht aufhören will
        erhöhe Seitenzähler
        drucke Titel und Seitennummer
        setzte Zeilenzähler auf 0

Weiter zum nächsten Datensatz in der Datenbank
Schleife fortsetzen (wenn Datenbankende nicht erreicht und Ende
    nicht befohlen)

Wenn Report beendet
Drucker ausschalten
Wenn nicht ausgedruckt wird und Ende nicht befohlen, Pause machen

Speichervariablen löschen
Zurück zum Programm REPORTS
```

Abb. 6.2: Pseudocode für das DIRECTOR-Programm (Forts.)

REPORTS-Programm setzt auch die Bedingungen für das Inhaltsverzeichnis DIRECTOR und speichert die Suchbedingungen in die Speichervariablen SUCHEN und BEDINGUNG. Wenn Sie also das Programm DIRECTOR testen wollen, sollten Sie mit dem Hauptmenü beginnen (DO POST) und dann die Option 2 (Berichte drucken) wählen, um sicherzustellen, daß Ihr DIRECTOR-Programm auch richtig mit den Programmen POST und REPORTS zusammenarbeitet.

Der Pseudocode für das Programm DIRECTOR ist in Abb. 6.2 zu sehen.

Lassen Sie uns schrittweise den Pseudocode in dBASE II-Code übersetzen. Zuerst schreiben wir die üblichen Befehle und löschen den Bildschirm:

```
********************************* DIRECTOR.CMD
************************* Druckt Namen und Adressen
************************* als Inhaltsverzeichnis
ERASE
```

Als nächstes erstellen wir die Speichervariablen, die das Programm braucht:

```
********************** Titel und Zähler erstellen
STORE "                Inhaltsverzeichnis" TO titel
STORE 1 TO seite
STORE 2 TO lf
STORE " " TO fertig
```

TITEL ist der Titel des Inhaltsverzeichnisses und SEITE enthält die Seitennummer, während in LF die Zahl der Zeilen festgehalten wird. Wir brauchen diese Informationen, um zu wissen, wann eine Ausgabepause auf dem Bildschirm oder eine neue Seite auf dem Drucker notwendig ist. Die Variable FERTIG wird zum Aufhören benötigt, falls der Benutzer es befiehlt, bevor alle Daten gedruckt sind.

Die nächste Routine fragt, ob das Inhaltsverzeichnis zum Drucker gehen soll. Die Antwort wird in der Variablen LP gespeichert. Lautet die Antwort ja, schaltet das Programm den Drucker ein.

```
************************** Nach Ausdruck fragen
STORE " " TO lp
§  2,2 SAY "Inhaltsverzeichnis drucken (J/N)?" GET lp
READ

IF !(lp) = "J"
    ?
```

```
    ? "Drucker bereitmachen und beliebige Taste
                                          drücken"
    WAIT
    SET PRINT ON
ENDIF (jn=j)
```

Lassen Sie uns jetzt die Routine zum Drucken des Berichts schreiben. Zuerst bringen wir das Programm dazu, den Bildschirm zu löschen, den Titel und die Seitennummer zu drucken und am Anfang der Datenbank zu beginnen. Diese Zeilen kümmern sich darum:

```
*************************** Report drucken
ERASE
TITEL+"           Seite"+STR(seite,2)
?
GO TOP
```

Jetzt müssen wir eine Schleife konstruieren, so daß das Programm immer weiter Datensätze aus der Datenbank POST sucht und druckt, bis das Dateiende (EOF) erreicht ist oder der Benutzer die Rückkehr zum Menü (FERTIG=R) befiehlt. Die Befehlszeile für diese Schleife sehen Sie hier:

```
DO WHILE .NOT. EOF .AND. !(fertig) # "R"
```

Nun sollte das Programm dafür sorgen, daß der zum Drucken vorgesehene Datensatz auch die im Programm REPORTS aufgestellten Suchkriterien erfüllt. Wenn das zutrifft, werden die Daten in zwei Zeilen gedruckt, dann folgt eine Leerzeile, und anschließend wird der Zeilenzähler (LF) um drei erhöht. Benutzen Sie diese Befehlszeilen, um die Arbeit zu erledigen:

```
***** Überprüfe Übereinstimmung mit Suchkriterien
IF &suchen=bedingung
   ? TRIM(nachname)+", "+vorname,firma,strasse
   ? " ",plz,TRIM(ort),land,telefon,swort
   ?
   STORE lf+3 TO lf
```

Sobald das Programm die Daten eines Satzes ausdruckt, muß es neue Seitenanfänge oder eine Pause für den Bildschirmausdruck bewerkstelligen. Die erste Routine kümmert sich um neue Seitenanfänge auf dem Drukker:

```
********************** Wenn gedruckt wird, beginne
********************** alle 55 Zeilen neue Seite
IF !(lp)="J" .AND. lf>=55
   EJECT
```

```
    STORE seite + 1 TO seite
    ? titel+"           Seite"+STR(seite,2)
    ?
    STORE 2 TO lf
ENDIF (55 Zeilen gedruckt)
```

Beachten Sie die IF-Konstruktion in dem Programm. Der Ausdruck !(lp)="J" bedeutet, daß der Bediener auf die Frage „Inhaltsverzeichnis ausdrucken?" mit ja geantwortet hat. Mit lf>=55 wird festgestellt, daß bereits 55 Zeilen gedruckt wurden. Sind beide Bedingungen zutreffend, sorgt das Programm für einen Seitenvorschub am Drucker (EJECT) und erhöht den Seitenzähler um 1 (STORE seite+1 TO seite). Dann druckt es den TITEL und die Seitennummer, wobei die Seitennummer zu einem String umgewandelt wird, damit sie sich besser formatieren läßt. Dann druckt es eine Leerzeile unter dem Titel und setzt den Zeilenzähler auf 2 zurück (STORE 2 TO lf).

Wird der Bericht nicht ausgedruckt, muß das Programm die Bildschirmanzeige nach jeweils 18 Zeilen anhalten, damit der Benutzer Gelegenheit hat, die Informationen zu lesen oder zu notieren. Diese Routine kümmert sich um die Bildschirm-Stops:

```
**************** Wenn nicht gedruckt wird, halte den
**************** Bildschirm nach 18 Zeilen an
IF !(lp)#"J" .AND. lf>=18
    ?
    ? "Weiter mit beliebiger Taste "
    ?? "(mit R zum Menü zurück)"
    WAIT TO fertig
    IF !(fertig)#"R"
        STORE seite+1 TO seite
        ERASE
        ? titel+"           Seite"+STR(seite,2)
        ?
        STORE 2 TO lf
    ENDIF (fertig#R)
ENDIF (18 Zeilen auf Bildschirm)
```

Hier heißt die IF-Bedingung !(lp)#"J" (Benutzer will nicht drucken) und lf>=18 (bereits 18 Zeilen angezeigt). Wenn dies zutrifft, erscheint auf dem Bildschirm die Meldung:

```
Weiter mit beliebiger Taste (mit R zum Menü zurück)
```

Drückt der Benutzer nicht auf R, wird die Seitennummer um 1 erhöht, der Bildschirm gelöscht, der Titel und die Seitennummer gedruckt und

der Zeilenzähler wieder auf 2 zurückgesetzt. Drückt der Benutzer aber R, wird keine dieser Operationen ausgeführt.

Die nächsten Zeilen schließen die Haupt-IF-Bedingung (IF &suchen= bedingung) ab, gehen weiter zum nächsten Datensatz (SKIP) und setzen die Schleife fort (solange das Ende der Datenbank nicht erreicht ist und der Benutzer in einer Bildschirmpause kein R eingetippt hat, um zum Menü zurückzukehren):

```
   ENDIF (suchen=bedingung)
SKIP
ENDDO (while not eof und fertig # R)
```

Ist das Inhaltsverzeichnis gedruckt, kann das Programm wieder zu REPORTS zurückkehren. Allerdings, wollte der Benutzer das Verzeichnis nicht ausdrucken und auch nicht zwischendurch abbrechen, sollten Sie die Bildschirmausgabe noch einmal anhalten, damit die letzte Anzeige nicht zu schnell verschwindet. Diese Zeilen sorgen dafür:

```
************************ Geht das Verzeichnis nicht
************************ zum Drucker, vor Rückkehr
************************ Bildschirm anhalten

SET PRINT OFF
IF !(lp) # "J" .AND. !(fertig) # "R"
   ?
   ?
   ? "Mit beliebiger Taste zum Menü zurück"
   WAIT
ENDIF (jn#j)
```

Dann muß das Programm die Speichervariablen löschen und zum REPORTS-Programm zurückfinden. Fügen Sie diese letzten Zeilen noch an:

```
RELEASE titel,lp,lf,seite,fertig
RETURN
```

Die vollständige Programmliste ist in Abb. 6.3 zu sehen.

```
****************************** DIRECTOR.CMD
************************ Druckt Namen und Adressen
************************ als Inhaltsverzeichnis
ERASE
```

Abb. 6.3: Die Programmliste zu DIRECTOR.CMD

```
********************** Titel und Zähler erstellen
STORE "                Inhaltsverzeichnis" TO titel
STORE 1 TO seite
STORE 2 TO lf
STORE " " TO fertig

************************** Nach Ausdruck fragen
STORE " " TO lp
§  2,2 SAY "Inhaltsverzeichnis drucken (J/N)?" GET lp
READ

IF !(lp) = "J"
    ?
    ? "Drucker bereitmachen und beliebige Taste drücken"
    WAIT
    SET PRINT ON
ENDIF (jn=j)

************************** Report drucken
ERASE
? TITEL+"           Seite"+STR(seite,2)
?
GO TOP
DO WHILE .NOT. EOF .AND. !(fertig)  #"R"

   ********* Überprüfe Übereinstimmung mit Suchkriterien
   IF &suchen=bedingung
      ? TRIM(nachname)+", "+vorname,firma,strasse
      ? " ",plz,TRIM(ort),land,telefon,swort
      ?
      STORE lf+3 TO lf

      ************************ Wenn gedruckt wird, beginne
      ************************ alle 55 Zeilen neue Seite
      IF !(lp)="J" .AND. lf>=55
         EJECT
         STORE seite + 1 TO seite
         ? titel+"           Seite"+STR(seite,2)
         ?
         STORE 2 TO lf
      ENDIF (55 Zeilen gedruckt)

      ************************ Wenn nicht gedruckt wird, halte den
      ************************ Bildschirm nach 18 Zeilen an
      IF !(lp)#"J" .AND. lf>=18
         ?
         ? "Weiter mit beliebiger Taste "
```

Abb. 6.3: Die Programmliste zu DIRECTOR.CMD (Forts.)

```
         ?? "(mit R zum Menü zurück)"
         WAIT TO fertig
         IF !(fertig)#"R"
         STORE seite+1 TO seite
         ERASE
         ? titel+"           Seite"+STR(seite,2)
         ?
         STORE 2 TO lf
      ENDIF (fertig#R)
   ENDIF (18 Zeilen auf Bildschirm)
 ENDIF (suchen=bedingung)
SKIP
ENDDO (while not eof und fertig  #R)

********************************** Geht das Verzeichnis nicht
********************************** zum Drucker, vor Rückkehr
********************************** Bildschirm anhalten
SET PRINT OFF
IF !(lp) # "J" .AND. !(fertig) # "R"
   ?
   ?
   ? "Mit beliebiger Taste zum Menü zurück"
   WAIT
ENDIF (jn#j)

RELEASE titel,lp,lf,seite,fertig
RETURN
```

Abb. 6.3: Die Programmliste zu DIRECTOR.CMD (Forts.)

Dies ist ein Programm für viele Gelegenheiten. Es kann in vielen Programmsituationen verwendet und leicht an andere Datenbanken angepaßt werden. Wenn Sie beispielsweise mit einer Datenbank arbeiten, die Verweise auf Bücher und Zeitschriften enthält und folgende Felder besitzt: AUTOR, TITEL, VERLAG, SEITEN, DATUM, THEMA und ABSTRAKT, können Sie sich ein gutes Stück Arbeit sparen, indem Sie einfach eine Kopie der DIRECTOR-Liste machen und ein paar Zeilen ändern. Genaugenommen müssen Sie die Zeile

```
STORE "           Inhaltverzeichnis" TO titel
```

in eine bedeutungsvollere ändern wie zum Beispiel

```
STORE "           Bibliographie" TO titel
```

Natürlich müssen Sie auch die dazugehörenden Felder ausdrucken. Um das zu erreichen, ändern Sie diese Zeilen:

```
? TRIM(nachname)+", "+vorname,firma,strasse
  ? "      ",plz,TRIM(ort),land,telefon,swort
  ?
  STORE lf+3 TO lf
```

in die für das neue Programm genaueren Zeilen:

```
  ? autor,titel
  ? verlag,seiten,datum
  ? thema
  ? abstrakt
  ?
  STORE lf+5 TO lf
```

Beachten Sie, daß der Zeilenzähler (lf) jetzt jedesmal um 5 erhöht wird, weil jeder Datensatz fünf Zeilen für den Ausdruck umfaßt.

Um das Suchelement aus dem Programm zu entfernen, streichen Sie einfach diese Zeilen:

```
IF &suchen=bedingung
```

und

```
ENDIF (suchen=bedingung)
```

Mehrfache Etikettenkopien mit Justierung

In meinem Einführungsbuch „Arbeiten mit dBASE II" (SYBEX Verlag, 1984) stellte ich ein einfaches Etikettenprogramm und ein etwas aufwendigeres Programm zum Drucken von mehrbahnigen Etiketten vor. In diesem Buch werden wir ein Etikettenprogramm entwickeln, das einige zusätzliche Funktionen hat.

Eine der Unannehmlichkeiten jedes Etikettenprogramms ist ganz einfach, daß Sie die Etiketten in den Drucker einspannen müssen, wenn Sie welche drucken wollen. Einige schlaue Firmen drucken deshalb gleich einige Kopien von jedem Etikett. So werden die Etiketten einmal in den Drucker geladen, dann werden vielleicht zehn Kopien von jedem gedruckt und diese anschließend aufgehoben. So kann bei Bedarf ein Etikett einfach aus dem „Lager" geholt werden, ohne daß man wieder den Computer und den Drucker bemühen muß.

Unser Programm LABELS ermöglicht den Mehrfachdruck einfach durch die Frage an den Benutzer:

```
Wieviel Kopien von jedem Etikett?
```

Hier kann eine Ziffer zwischen 1 und 99 eingegeben werden, und das Programm wird entsprechend viele Etiketten drucken.

Es gibt aber noch eine andere Unannehmlichkeit beim Etikettendruck: Manchmal liegen sie nicht richtig im Drucker. Wird das zu spät bemerkt, muß der Bediener den Druck unterbrechen, sie neu ausrichten und erneut starten. Unser Etikettenprogramm LABELS hilft hier mit der Frage:

```
Wünschen Sie zuerst einen Probedruck?
```

Stimmt der Benutzer zu, druckt das Programm zwei Versuchsetiketten, die so aussehen:

```
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
PPPP    OOOOOOOOOOOOOOOOOOOOOOOOOOOO

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
PPPP    OOOOOOOOOOOOOOOOOOOOOOOOOOOO
```

So sieht der Benutzer auf einen Blick, ob die Etiketten richtig eingespannt sind oder nicht. Dann erscheint die Frage auf dem Bildschirm:

```
Noch ein Probedruck?
```

Hat es beim ersten Mal nicht geklappt, kann er jetzt mit J (ja) antworten und das Programm druckt noch einmal zwei „Schwindeletiketten". Bestehen keine weiteren Schwierigkeiten, antwortet er mit N (nein), und das Programm beginnt mit dem eigentlichen Druck der Etiketten.

Schließlich kommt es auch vor, daß die Etiketten während des Druckes ihre Ausrichtung verlieren. Dieser Fehler kann unter Umständen den kompletten Neudruck von Anfang an erfordern. Unser Programm kommt auch mit dieser Situation gut zurecht. Sobald es mit dem Drucken beginnt, erscheint diese Meldung auf dem Bildschirm:

```
Drücken Sie <ESC>, um den Druck anzuhalten. Danach
  geben Sie DO LABELS und einmal RETURN ein.
```

Wenn also irgend etwas mit dem Druck nicht stimmt, kann dieser mit der ESCAPE-Taste gestoppt werden, worauf das dBASE II-Prompt erscheint (das LABELS-Programm wird hinterher da weitermachen, wo es unterbrochen wurde). Dann gibt man DO LABELS ein. Das Programm ermöglicht zuerst das Ausrichten und Testen der Etikettenposition und macht dann mit dem Etikettendruck, ein Etikett vor dem zuletzt gedruckten, weiter (weil das sicherlich ruiniert ist).

Einen kleinen Schönheitsfehler hat diese Methode allerdings. Das Programm schaltet während des Druckens den Bildschirm aus (SET CONSOLE OFF), um die Etiketten nicht anzuzeigen. Wenn dann mit ESC das Programm unterbrochen und anschließend wieder DO LABELS eingegeben wird, erscheint dieser Befehl nicht auf dem Bildschirm. Trotzdem wird natürlich das Programm mit dem Etikettendruck weitermachen, wenn der Befehl richtig eingetippt wurde. Es gibt außer dem Weglassen des Befehls SET CONSOLE OFF keinen eleganteren Weg, diese Situation zu beherrschen.

Da das LABELS-Programm vom Programm REPORTS aufgerufen wird, erscheinen die Etiketten in der vom Benutzer angegebenen Reihenfolge. Außerdem werden nur Etiketten für jene Datensätze gedruckt, die den Suchkriterien entsprechen. Das Programm ist auf einbahnige Etiketten, drei Zoll breit und ein Zoll hoch, ausgerichtet.

Der Pseudocode für das LABELS-Programm steht in Abb. 6.4. Das LABELS-Programm selbst, beginnt so:

```
********************************** LABELS.CMD
*************** Mehrfache Etikettenkopien drucken
SET CONSOLE ON
SET PRINT OFF
```

Die üblichen Kommentare stehen wieder ganz oben. Direkt darunter schaltet das Programm den Bildschirm ein und den Drucker aus. Das ist notwendig, falls der Benutzer den Druck mit ESC unterbrochen und der Drucker immer noch ein- und/oder der Bildschirm immer noch ausgeschaltet sein sollte.

Dann muß sich das Programm entscheiden, ob es vom Beginn der Datenbank oder von einem Punkt aus starten soll, wo es vorher den Druck unterbrochen hat. Diese Zeilen ermöglichen die Entscheidung:

```
*** Wenn nicht unterbrochen, oben anfangen
IF TYPE(start)="U"
   STORE "TOP" TO start
ENDIF (type=u)
```

Die IF-Anweisung überprüft, ob die Speichervariable START existiert. Wenn nicht, ist TYPE(start) (U)ndefiniert, was bedeutet, daß mit dem Druck der Etiketten am Anfang der Datenbank begonnen werden soll. Das Wort „TOP" gehört also zu der Speichervariablen START. Später benutzen wir den Befehl GOTO&start, um den Datensatzzeiger auf das erste zu druckende Etikett zu setzen.

```
Schalte den Drucker aus und den Bildschirm ein

Wenn der Etikettendruck gerade beginnt
   starte am Anfang der Datenbank

Lösche Bildschirm

Frage nach der Anzahl der Kopien, speichere in NR:LABELS

Nachfragen, ob Probedruck gewünscht

Solange Probedruck gewünscht
   zwei Probeetiketten drucken
   fragen, ob weiter probedrucken

Fertigmachen, um richtige Etiketten zu drucken

Gehe zur entsprechenden Datensatznummer (START)
Zeige Informationen auf dem Bildschirm für den Fall,
   daß Druckprobleme auftauchen

Schalte Drucker ein

Beginne Schleife durch die Datenbank (bis EOF)
   Wenn Etikett = Suchkriterium
      Drucke die geforderte Zahl Kopien (NR:LABELS)

   Speichere die Nummer des zuletzt gedruckten Etiketts
Weiter zum nächsten Datensatz und in der Schleife (da nicht EOF)

Wenn fertig, Drucker ausschalten
Speichervariablen löschen
RETURN zum REPORTS-Programm
```

Abb. 6.4: Pseudocode für das LABELS-Programm

Gibt es die Variable START bereits, bedeutet das, daß der Etikettendruck bereits begonnen hatte, dann aber unterbrochen wurde. In diesem Fall enthält TYPE(start) Daten im (C)haracter-Format, die angeben, wo mit dem Drucken begonnen werden soll, wie wir gleich sehen werden.

In der folgenden Routine fragt das Programm nach der Anzahl der Kopien für jedes Etikett.

```
*** Zuerst nach der Zahl der Kopien fragen
ERASE
STORE 1 TO nr:labels
§  2,2 SAY "Wie viele Kopien von jedem Etikett?";
      GET nr:labels PICTURE "99"
READ
```

Beachten Sie, daß die Antwort in der Variablen NR:LABELS gespeichert wird.

Der nächste Schritt ist zu fragen, ob der Sitz der Etiketten im Drucker überprüft werden soll. Diese Zeilen sorgen dafür:

```
******** Dann frage den Benutzer, ob er den Sitz
******** der Etiketten
******** im Drucker überprüfen will
STORE " " TO sitz
§  4,4 SAY "Wünschen Sie zuerst einen Probedruck?";
      GET sitz
READ
```

Solange der Benutzer den Probedruck wünscht (die Variable SITZ ist = „J"), druckt das Programm zwei Probeetiketten und fragt wieder, ob jetzt alles in Ordnung ist. Die DO WHILE-Schleife hier erledigt den Probedruck:

```
********** Wenn Probedruck gewünscht, zwei
**********Musteretiketten drucken
DO WHILE !(sitz) = "J"
   SET PRINT ON
   SET CONSOLE OFF
   STORE 1 TO zaehler
   DO WHILE zaehler < 3
      ? "NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN"
      ? "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF"
      ? "SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS"
```

```
      ? "PPPP    OOOOOOOOOOOOOOOOOOOOOOOOOOOO"
      ?
      ?
      STORE zaehler+1 TO zaehler
   ENDDO (while zaehler<3)

   ********** Erlaubnis zum Weitermachen einholen
   SET CONSOLE ON
   SET PRINT OFF
   STORE " " TO sitz
   §     6,5 SAY "Noch ein Probedruck?" GET sitz
   READ

ENDDO (while sitz=J)
```

Wenn die Etiketten richtig im Drucker sitzen, kann das Programm mit dem Druck beginnen. Zuerst muß das Programm den Datensatzzeiger in Startposition bringen (gespeichert in der Variablen START). Diese Zeilen helfen dabei:

```
********************* Datensatzzeiger positionieren
GOTO &start
```

Dann wird der Bildschirm gelöscht, und die Informationen über den möglichen Programmabbruch werden angezeigt.

```
***************************** Informationen anzeigen
ERASE
? "Drücken Sie <ESC>, wenn Sie den Druck anhalten
                                                möchten,"
? "mit DO LABELS und einmal RETURN weitermachen."
```

Dann muß das Programm den Drucker einschalten und die Schleife durch die Datenbank mit diesen Zeilen beginnen:

```
SET PRINT ON
DO WHILE .NOT. EOF
```

Danach überprüft das Programm, ob das Etikett mit den im REPORTS-Programm genannten Suchbedingungen übereinstimmt:

```
*************** Sicherstellen, daß Etikett mit
************** Suchbedingungen übereinstimmt
IF &suchen=bedingung
```

Wenn ja, druckt das Programm so viele Etiketten, wie in der Variablen NR:LABELS angegeben, mit diesen Zeilen:

```
      STORE 1 TO zaehler
      ********** Drucke Mehrfachkopien dieses Etiketts
      DO WHILE zaehler <= nr:labels
        SET CONSOLE OFF
        ? TRIM(vorname),nachname
        ? firma
        ? strasse
        ? plz+"  "+TRIM(ort)+",  "+land
        ?
        ?
        SET CONSOLE ON
        STORE zaehler+1 TO zaehler
      ENDDO (zaehler < nr:labels)
  ENDIF (suchen=bedingung)
```

Das Programm schreibt dann die Nummer des gerade gedruckten Etiketts in die Speichervariable START. Wie das geht, sehen Sie in diesen Zeilen:

```
  ********** Das zuletzt gedruckte Etikett speichern
  STORE STR(#,4) TO start
```

Dann springt das Programm zum nächsten Datensatz und schließt damit die DO WHILE-Schleife (do while .not. eof):

```
  SKIP
  ENDDO (while not eof)
```

Sind die Etiketten fertig gedruckt, werden alle Speichervariablen gelöscht, der Drucker wird ausgeschaltet und die Kontrolle wieder an das Programm REPORTS zurückgegeben.

```
  ************************** Fertig mit Drucken
  SET PRINT OFF

  RELE nr:labels,sitz,zaehler,start
  RETURN
```

Das vollständige LABELS-Programm finden Sie in der Abb. 6.5.

```
********************************** LABELS.CMD
*************************** Mehrfache Etikettenkopien drucken
SET CONSOLE ON
SET PRINT OFF

*** Wenn nicht unterbrochen, oben anfangen
IF TYPE(start)="U"
   STORE "TOP" TO start
ENDIF (type=u)

*** Zuerst nach der Zahl der Kopien fragen
ERASE
STORE 1 TO nr:labels
§  2,2 SAY "Wie viele Kopien von jedem Etikett?";
       GET nr:labels PICTURE "99"
READ

******** Dann frage den Benutzer, ob er den Sitz der Etiketten
******** im Drucker überprüfen will
STORE " " TO sitz
§  4,4 SAY "Wünschen Sie zuerst einen Probedruck?";
       GET sitz
READ

************ Wenn Probedruck gewünscht, zwei Musteretiketten
************ drucken
DO WHILE !(sitz) = "J"
   SET PRINT ON
   SET CONSOLE OFF
   STORE 1 TO zaehler
   DO WHILE zaehler < 3
      ? "NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN"
      ? "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF"
      ? "SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS"
      ? "PPPP    OOOOOOOOOOOOOOOOOOOOOOOOOOOO"
      ?
      ?
      STORE zaehler+1 TO zaehler
   ENDDO (while zaehler<3)

   ********** Erlaubnis zum Weitermachen einholen
   SET CONSOLE ON
   SET PRINT OFF
   STORE " " TO sitz
   §  6,5 SAY "Noch ein Probedruck?" GET sitz
   READ
```

Abb. 6.5: Das Programm LABELS.CMD

```
ENDDO (while sitz=J)
**** Datensatzanzeiger positionieren
GOTO &start
***************************** Informationen anzeigen
ERASE
? "Drücken Sie <ESC>, wenn Sie den Druck anhalten möchten,"
? "mit DO LABELS und einmal RETURN weitermachen."
SET PRINT ON
DO WHILE .NOT. EOF

   ***************************** Sicherstellen, daß Etikett mit
   ***************************** Suchbedingungen übereinstimmt
   IF &suchen=bedingung
      STORE 1 TO zaehler

      ********************** Drucke Mehrfachkopien dieses Etiketts
      DO WHILE zaehler <= nr:labels
         SET CONSOLE OFF
         ? TRIM(vorname),nachname
         ? firma
         ? strasse
         ?
         ? plz+"  "+TRIM(ort)+",  "+land
         ?
         ?
         SET CONSOLE ON
         STORE zaehler+1 TO zaehler
      ENDDO (zaehler < nr:labels)
   ENDIF (suchen=bedingung)

   ****************** Das zuletzt gedruckte Etikett speichern
   STORE STR(#,4) TO start
   SKIP
ENDDO (while not eof)

************************** Fertig mit Drucken
SET PRINT OFF

RELE nr:labels,sitz,zaehler,start
RETURN
```

Abb. 6.5: Das Programm LABELS.CMD (Forts.)

Eine MailMerge-Datei erstellen

Wenn wir die MailMerge-Option von WordStar benutzen, können wir die Daten unserer Adressendatei dazu verwenden, Serienbriefe zu drucken. Die Hauptaufgabe besteht hier darin, die Daten der Datenbank

POST.DBF in eine andere Datei so zu transportieren, daß MailMerge sie lesen und verarbeiten kann. In MailMerge-Dateien muß jedes Feld ohne Leerzeichen in Anführungszeichen stehen, durch Kommas vom nächsten getrennt, wie hier zu sehen ist:

```
"feld 1","feld 2","feld 3","feld 4"
"feld 1","feld 2","feld 3","feld 4"
```

Die Option DELIMITED zusammen mit dem COPY-Befehl erstellt uns eine solche Datei.

Wir werden ein Programm mit dem Namen MMERGE.CMD schreiben, um die Daten von POST.DBF in eine MailMerge-Datei zu schicken. Der Benutzer kann einen Namen für die MailMerge-Datei eingeben. Die Daten stehen in der im Programm REPORTS angegebenen Sortierung, und es werden nur die Daten übertragen, die den Suchbedingungen entsprechen. Der Pseudocode für dieses Programm steht in der Abb. 6.6.

Die ersten Zeilen des Programms enthalten die üblichen Kommentare und das Löschen des Bildschirms:

```
******** MMERGE.CMD
******** Eine WordStar-MailMerge-Datei erstellen
ERASE
```

Dann sollte das Programm nach einem Namen für die MailMerge-Datei fragen. Es speichert die Antwort in einer Variablen mit dem Namen MDATEN:

```
************ Den Namen der MailMerge-Datei erfragen
? "Wie soll die MailMerge-Datei heißen?"
ACCEPT "(Mit Laufwerksbezeichnung eingeben, z.B.
                              B:MM.DAT)" TO MDATEN
```

Lösche Bildschirm

Nach dem MailMerge-Dateinamen fragen

Kopiere Datensätze, die den Suchbedingungen entsprechen, im MailMerge-Format in die neue Datei

Lösche Speichervariablen
Zurück zum Hauptmenü

Abb. 6.6: Pseudocode für die Datei MDATEN.CMD

Die folgende Kurzmitteilung erscheint auf dem Schirm, während die Daten kopiert werden:

```
ERASE
? "Erstelle MailMerge-Datei &mdaten"
```

Die nächsten Programmzeilen schicken die Daten aus der Datei POST.DBF in die MailMerge-Datei. Der Befehl DELIMITED WITH " sorgt dafür, daß die Felder mit Anführungszeichen versehen und durch Kommas getrennt werden. Die Bedingung IF &suchen=bedingung stellt sicher, daß das Programm nur Daten kopiert, die den Suchbedingungen entsprechen. Da das Programm REPORTS immer noch mit einem Index arbeitet, werden die Daten sortiert in die MailMerge-Datei übertragen:

```
COPY TO &mdaten DELIMITED WITH " FOR &suchen=
                                        bedingung
```

Sobald die Übertragung beendet ist, gibt das Programm eine kurze Meldung auf dem Bildschirm aus, löscht die Speichervariablen und kehrt zum Hauptmenü zurück:

```
********** Fertig. Zurück zum Menü 'Drucken'
?
```

```
********** MMERGE.CMD
********** Eine WordStar-MailMerge-Datei erstellen
ERASE

********** Den Namen der MailMerge-Datei erfragen
? "Wie soll die MailMerge-Datei heißen?"
ACCEPT "(Mit Laufwerksbezeichnung eingeben, z.B. B:MM.DAT)" TO
                                                    MDATEN
ERASE
? "Erstelle MailMerge-Datei &mdaten"
COPY TO &mdaten DELIMITED WITH " FOR &suchen=bedingung

********** Fertig. Zurück zum Menü 'Drucken'
?
?
? "MailMerge-Datei &mdaten erfogreich erstellt ..."
?
RELEASE mdaten
RETURN
```

Abb. 6.7: Das Programm MDATEN.CMD

```
?
? "MailMerge-Datei &mdaten erfogreich erstellt..."
?
RELEASE mdaten
RETURN
```

Die Abb. 6.7 zeigt das kurze MDATEN-Programm.

In Abb. 6.8 stelle ich Ihnen einmal eine WordStar-Dokumentdatei vor, mit deren Hilfe Sie die Serienbriefe mit unseren Adressen aus MDATEN drucken können. Die Dokumentendatei geht davon aus, daß Sie Ihre MailMerge-Datei MM.DAT nennen und diese auf der Diskette in Laufwerk B steht (.DF B:MM.DAT).

```
.OP
.DF B:MM.DAT
.RV NACHNAME,VORNAME,FIRMA,STRASSE,PLZ,ORT,
.RV LAND,TELEFON,SWORT

&VORNAME& &NACHNAME&
&FIRMA&
&STRASSE&
&PLZ& &ORT& &LAND&

Lieber Clubfreund &VORNAME&,

in unseren Unterlagen steht, daß Deine Mitgliedschaft im
nächsten Monat endet. Wenn Du doch noch in unserem wunder-
vollen Club bleiben möchtest, überweise bitte 10,- DM, und wir
werden Dich nicht aus unserer Datenbank löschen.

Mit kameradschaftlichem Gruß


Karl Kelle
1. Vorsitzender


.PA
```

Abb. 6.8: Die WordStar-Datei zum Drucken der Serienbriefe

Um eine solche WordStar-Datei zu erstellen, nehmen Sie die Diskette, auf der sich WordStar und MailMerge befinden und rufen WordStar auf, indem Sie WS und anschließend RETURN eintippen. Erstellen Sie die Datei (mit der D-Option) und speichern Sie diese auf die Diskette in Laufwerk B mit einem gültigen Namen (z.B. SERIE.BRF) ab. Dann geben Sie die Punktbefehle genau so ein, wie sie in der Abb. 6.8 stehen (sorgen Sie dafür, daß Sie den richtigen Dateinamen im .DV-Befehl verwenden).

Wenn Sie fertig sind, speichern Sie die Datei mit dem Befehl ^KD, und aus dem dann auftauchenden WordStar-Menü wählen Sie die Option M, um die MailMerge-Datei zu drucken. Sobald WordStar nach dem Namen der zu druckenden Datei fragt, geben Sie den Namen der Briefdatei ein (z.B. SERIE.BRF). Schalten Sie den Drucker ein, und schon kann es losgehen. WordStar wird für jeden Datensatz in der MailMerge-Datei einen Brief drucken. In der Abb. 6.9 sehen Sie ein Beispiel für solch einen Serienbrief.

Herrmann Zahlnicht
NIX GmbH
Am Goldmorgen 18

5000 Köln

Lieber Clubfreund Herrmann,

in unseren Unterlagen steht, daß Deine Mitgliedschaft im nächsten Monat endet. Wenn Du doch noch in unserem wundervollen Club bleiben möchtest, überweise bitte 10,– DM, und wir werden Dich nicht aus unserer Datenbank löschen.

Mit kameradschaftlichem Gruß

Karl Kelle
1. Vorsitzender

Abb. 6.9: Beispiel für einen Serienbrief

Zusammenfassung

In diesem Kapitel haben wir verschiedene Techniken zum Erstellen von nützlichen Berichten kennengelernt. Sie haben gelernt, wie grundlegende Reports für den formatierten Ausdruck über den Drucker oder den Bildschirm programmiert werden, außerdem wie Sie ein Programm zum Drucken von Etiketten schreiben und WordStar Serienbriefe mit MailMerge drucken lassen. Im nächsten Kapitel diskutieren wir Programme, mit denen Daten in einer Datenbank verändert und gelöscht werden können.

Programme zum Ändern und Löschen

7

Zum Ändern, Löschen und Beseitigen von Daten bietet dBASE II die Befehle EDIT, DELETE und PACK. Diese verlangen allerdings vom Anwender, daß er sich mit der Datensuche nach Datensatznummern und einige Controll-Kommandos auskennt. In einem menügesteuerten Programmsystem sollten diese Datenbankarbeiten ohne spezielle Kenntnisse über die Befehle und die Sprachsyntax möglich sein. Außerdem können Informationen über die verschiedenen Cursor-Befehle angezeigt werden, so daß der unerfahrene Benutzer sich daran nicht erst erinnern muß, wenn er die Datenbank verändern will.

Automatisches Ändern der Datenbank

Kann ein Benutzer lesbare Informationen zum Suchen eines zu ändernden Datensatzes benutzen, zum Beispiel einen Namen aus einer Versandliste, wird die Änderungsprozedur recht einfach. Wir können ein Programm schreiben, das nach solch einfachen Informationen wie einem Namen fragt:

```
Welchen ändern? (Nachname oder <RET> für ENDE):
```

Der Benutzer dieses Systems muß nur den Nachnamen Schmidt eingeben und RETURN drücken. Gibt es in der Datenbank keinen Schmidt, zeigt

der Bildschirm folgende Meldung:

```
Kein Schmidt in dieser Datenbank

        Weiter mit beliebiger Taste ...
```

Gibt es dagegen mehrere Schmidts in der Datenbank, werden alle auf dem Bildschirm angezeigt:

```
00005   Schmidt Andreas   ABC GmbH     Astr.123
00096   Schmidt Corinna   XYZ & Co     Bweg 23
00198   Schmidt Ralf      Datacomp     Callee 987
01234   Schmidt Zeppo     WMVB         Postf. 12345

      Welchen wählen Sie ? (Nummer eingeben) :
```

In diesem Fall kann der Benutzer den richtigen Schmidt für seine Änderungen aussuchen, indem er die links stehende Nummer eintippt.

Findet das System nur einen Schmidt in der Datenbank, wird dieser gleich in einer Änderungsmaske mit sämtlichen Informationen (s. Abb. 7.1) angezeigt.

Sind alle Änderungen gemacht, wird der Bildschirm gelöscht, und diese Frage erscheint wieder:

```
Welchen ändern? (Nachname oder <RET> für ENDE):
```

```
Cursor-Kontrolle: (Das Symbol ^ bedeutet 'CTRL-Taste drücken...')
^E hoch : ^X runter : ^D rechts : ^S links : ^G loeschen : ^V einfuegen
-------------------------------------------------------------------
Nachname:Schmidt          : Vorname  :Andreas    :
Firma    :ABC GmbH        :
Strasse  :Astr. 123       :
PLZ      :5000:           Ort  :Köln   : Land  :BRD :
Telefon  :0221-1234567    :
Suchwort :GEKÜNDIGT 10/84 :
```

Abb. 7.1: Änderungsmaske für die Adressenverwaltung

Will er einen anderen Namen zum Ändern aufrufen, kann der Benutzer hier den entsprechenden eingeben; um mit dieser Arbeit aufzuhören, tippt er einfach RETURN ein.

In unserem Beispiel einer Adressenverwaltung werden Änderungen vom Programm EDIT und der Formatdatei AMASK1.FMT, die wir in Kapitel 4 entwickelt haben, ermöglicht. Die logische Struktur des EDIT-Programms sehen Sie im Pseudocode in der Abb. 7.2.

Wir sind jetzt soweit, daß wir auf der Grundlage des Pseudocodes das EDIT-Programm entwickeln können. Dabei können wir auch die Index-

```
Schleifenvariablen erstellen

Solange der Benutzer nicht aufhören will

    Bildschirm löschen
    Benutzer nach zu änderndem Namen fragen
    Name in Großbuchstaben umwandeln
    Person finden
    Zählen, wieviel Personen in der Datenbank
        diesen Namen tragen

    Wenn kein Name zum Ändern angegeben
    Bildschirm löschen und zurück zum Hauptmenü

    Wenn der gefragte Name nicht in der Datenbank
    Benutzer darüber informieren und
        weiteren Versuch ermöglichen

    Wenn eine Person des Namens vorhanden
    Änderungsmaske und Datensatz zeigen

    Wenn mehrere Personen mit diesem Namen vorhanden
    Anzeigen und weitere Informationen einholen
    Dann Maske zeigen und Ändern ermöglichen

    Wiederhole den Änderungsprozeß (solange Benutzer nicht auf-
        hören will)

    Ist der Benutzer fertig
        Speichervariablen löschen
        zurück zum Hauptmenü
```

Abb. 7.2: Pseudocode für das EDIT-Programm

Datei NAMEN einsetzen und vor allem den WHILE-Befehl, um die Suche nach einem gewünschten Datensatz zu beschleunigen.

Der erste Schritt ist natürlich wieder, die Kommentare mit dem Namen und der Funktion des Programms zu zeigen:

```
********************************** EDIT.CMD
********** Daten der Datei POST mit Maske
                          AMASK1.FMT ändern
```

Dann erstellen wir eine Schleife, die nach einem Namen fragt, solange der Benutzer kein RETURN eingegeben hat. Hier sind die Aufgaben, die das Programm innerhalb der Schleife zu erledigen hat: Bildschirm löschen, nach dem zu ändernden Namen fragen und auf Antwort warten, dann wieder den Bildschirm löschen. Das erledigen diese Programmzeilen:

```
********** Zuerst den zu ändernden Namen erfragen
STORE "X" TO suchen
DO WHILE suchen # " "
   ERASE
   ACCEPT "Welchen ändern? (Nachname oder <RET> für
                                    ENDE)" TO suchen
   ERASE
```

Beachten Sie, die Bedingung für die Schleife heißt: DO WHILE suchen # " ". Um diese Schleife zu starten, müssen wir etwas speichern, was nicht LEER (" ") ist. Das kann sein:

1. etwas anderes als ein Leerzeichen,

2. ein Buchstabe des Alphabets,

3. irgendein Zeichen, außer dem Leerzeichen.

Oben haben wir einfach zufällig den Buchstaben X gewählt und in die Variable SUCHEN geladen, weil der ungleich einem Leerzeichen ist.

Nun übersetzt das Programm den eingegebenen Namen in Großbuchstaben, damit der Befehl FIND mit der Index-Datei richtig arbeiten kann (erinnern Sie sich, daß wir den Index mit !(nachname) und !(vorname) gebildet haben. Danach zählt das Programm die Häufigkeit des eingegebenen Namens, damit es sich für die richtige Anzeigeform entscheiden kann. Diese Zeilen zeigen den Weg:

```
*************** Übersetze in Großbuchstaben, dann
*************** zähle, wie
*************** oft der Name vorkommt
```

```
STORE!(suchen) TO suchen
FIND &suchen
COUNT WHILE !(nachname) = suchen TO wie viele
```

Abhängig von der Anzahl der gefundenen Namen, muß sich das Programm für die richtige Darstellungsform entscheiden. Wir fügen eine DO CASE-Anweisung hier noch an, um die Entscheidung zu ermöglichen:

```
******* Mache weiter, in Abhängigkeit von der Anzahl
******* der gefundenen Namen
DO CASE
```

Gibt der Benutzer keinen Namen ein, macht das Programm nichts anderes, als den Bildschirm zu löschen und zum Hauptmenü zurückzukehren. Die erste CASE-Anweisung trifft zu (suchen=" "), wenn kein Name eingegeben wird. Deshalb werden die weiteren CASE-Möglichkeiten übersprungen, die ENDDO-Schleife wird nicht wiederholt und das Programm verzweigt zurück zum Hauptmenü. Die erste CASE-Anweisung erscheint hier:

```
**** Wenn kein Name eingegeben, Bildschirm löschen
   CASE suchen=" "
     ERASE
```

Stellen die FIND- und COUNT-Befehle fest, daß niemand in der Datenbank die gesuchten Namen trägt, sollte das Programm den Benutzer darüber informieren und ihm eine weitere Möglichkeit zur Eingabe geben. Die zweite CASE-Anweisung ermöglicht das:

```
******** Wenn niemand mit dem Namen gefunden wird,
******** Benutzer informieren
CASE wie viele = 0
   ? "Kein &suchen in dieser Datenbank"
   ?
   ? "Weiter mit beliebiger Taste ... "
   WAIT
```

Gibt es in der Datenbank einen Eintrag mit dem gesuchten Nachnamen, soll das Programm einfach den Datensatz in der Änderungsmaske auf den Bildschirm bringen. Beachten Sie, daß die Formatdatei AMASK1.FMT zusammen mit dem READ-Befehl benutzt wird, um die Daten zum Ändern anzuzeigen. Bei der nächsten CASE-Anweisung sehen Sie, wie so etwas geht:

```
**** Wenn eine Person den richtigen Nachnamen hat,
**** AMASK1.FMT aufrufen und ändern lassen
```

```
CASE wie viele = 1
   FIND &suchen
   SET FORMAT TO amask1
   READ
   SET FORMAT TO SCREEN
```

Diese Case-Option setzt den Datensatzzeiger auf den zu ändernden Datensatz (FIND &suchen), ruft die Änderungsmaske AMASK1.FMT auf (SET FORMAT TO amask1), erlaubt die notwendigen Änderungen (READ) und schaltet dann wieder auf den normalen dBASE II-Bildschirmmodus um (SET FORMAT TO SCREEN).

Wenn natürlich mehrere Personen den gesuchten Namen haben, braucht das Programm zusätzliche Informationen. Deshalb wird es alle gefundenen Namen und die dazugehörenden Datensatznummern anzeigen und dann den Benutzer nach der zu ändernden Datensatznummer fragen. Diese weiteren Zeilen des Programmtextes erledigen die Aufgabe:

```
**** Wenn mehrere Personen den gesuchten Namen haben
**** zeige sie an und hole weitere Informationen
CASE wie viele > 1
  FIND &suchen
  LIST WHILE !(nachname) = suchen;
       TRIM(vorname),nachname,firma,strasse
  ?
  ACCEPT "Welchen wählen Sie ? (Nummer eingeben)"
TO satznr


  ***** Wenn der Benutzer eine Ziffer eingibt, ändere
  ***** diesen Datensatz und benutze AMASK1.FMT
  IF VAL(satznr)>0
     GOTO &satznr
     SET FORMAT TO amask1
     READ
     SET FORMAT TO SCREEN
  ENDIF (satznr>0)
```

Als Vorsichtsmaßnahme überprüft das Programm erst einmal, daß der Benutzer auch eine vernünftige Datensatznummer eingegeben hat (IF VAL(satznr)>0), bevor es sich entschließt, zu dieser Satznummer zu gehen. Ist die eingetippte Satznummer größer als Null, geht das Programm dorthin, ruft die Formatdatei AMASK1.FMT auf und gestattet die Änderungen. Danach setzt das Programm den Bildschirm wieder in den normalen dBASE II-Modus zurück.

Die DO CASE-Anweisung ist jetzt komplett, weil alle möglichen Eingaben geprüft werden. Deshalb schließen die nächsten Zeilen die DO CASE-Anweisung und die Schleife DO WHILE suchen # " ". Das geht so:

```
    ENDCASE
ENDDO (while suchen # " ")
```

Die DO WHILE-Schleife fragt nicht mehr „Welchen ändern?", sobald ein RETURN eingegeben wurde. Deshalb löschen die restlichen Programmzeilen nur noch die Variablen und führen zum Hauptmenü zurück:

```
********* Wenn kein Name zum Ändern eingegeben,
********* Variablen löschen und zurück zum Hauptmenü
RELEASE suchen, wie viele
RETURN
```

Die Abb. 7.3 zeigt das komplette Listing des EDIT-Programms. Beachten Sie den Gebrauch der Befehle FIND, COUNT WHILE und LIST WHILE, die für eine schnelle Verarbeitung sorgen. Das Hauptmenü-

```
********************************** EDIT.CMD
********** Daten der Datei POST mit Maske AMASK1.FMT ändern

*************** Zuerst den zu ändernden Namen erfragen
STORE "X" TO suchen
DO WHILE suchen # " "
   ERASE
   ACCEPT "Welchen ändern? (Nachname oder <RET> für ENDE)"
                                                    TO suchen
   ERASE

   *************** Übersetze in Großbuchstaben, dann zähle, wie
   *************** oft der Name vorkommt
   STORE!(suchen) TO suchen
   FIND &suchen
   COUNT WHILE !(nachname) = suchen TO wieviele

   ************** Mache weiter, in Abhängigkeit von der Anzahl
   ************** der gefundenen Namen
   DO CASE

      ***************** Wenn kein Name eingegeben, Bildschirm
      ***************** löschen
```

Abb. 7.3: Das Programm EDIT.CMD

```
      CASE suchen=" "
          ERASE

      ********** Wenn niemand mit dem Namen gefunden wird,
      ********** Benutzer informieren
      CASE wieviele = 0
          ? "Kein &suchen in dieser Datenbank"
          ?
          ? "Weiter mit beliebiger Taste ... "
          WAIT

      ***************** Wenn eine Person den richtigen Nachnamen
      ***************** hat, AMASK1.FMT aufrufen und ändern lassen
      CASE wieviele = 1
      FIND &suchen
          SET FORMAT TO amask1
          READ
          SET FORMAT TO SCREEN

      ********** Wenn mehrern Personen den gesuchten Namen
      ********** haben, zeige sie an und hole weitere
      ********** Informationen
      CASE wieviele > 1
      FIND &suchen
      LIST WHILE !(nachname) = suchen;
           TRIM(vorname),nachname,firma,strasse
           ?
           ACCEPT "Welchen wählen Sie ? (Nummer eingeben)"
                                                   TO satznr

           ********** Wenn der Benutzer eine Ziffer eingibt,
           ********** ändere diesen Datensatz und benutze
           ********** AMASK1.FMT
           IF VAL(satznr)>0
              GOTO &satznr
              SET FORMAT TO amask1
              READ
              SET FORMAT TO SCREEN
           ENDIF (satznr>0)

      ENDCASE

ENDDO (while suchen # " ")

**************** Wenn kein Name zum Ändern eingegeben,
**************** Variablen löschen und zurück zum Hauptmenü
RELEASE suchen,wieviele
RETURN
```

Abb. 7.3: Das Programm EDIT.CMD (Forts.)

Programm POST arbeitet immer noch mit der Datei NAMEN.NDX als erste Indexdatei, so daß der FIND-Befehl hier eingesetzt werden kann. Um das EDIT-Programm nach dem Eintippen zu testen, fangen Sie mit DO POST an und wählen die Option „4. Namen und Adressen ändern". Auf diese Weise kann das POST-Programm erst einmal den Befehl USE POST INDEX namen,plz ausführen und die richtige Datenbank für die Änderungen wählen.

Datensätze aus einer Datenbank löschen

Das Programm zum Löschen von Daten ist fast gleich dem Programm zum Ändern. Es fragt den Benutzer zuerst:

```
Wen löschen? (Nachname oder <ET> für ENDE):
```

erlaubt ihm, den Namen einzugeben oder einfach mit RETURN aufzuhören. Wie beim Programm EDIT, warnt das Programm nach der Eingabe eines Namens, der nicht in der Datenbank steht und gibt die Möglichkeit, es noch einmal zu versuchen. Gibt der Benutzer einen Namen, den dBASE kennt, und kommt dieser nur einmal vor, bringt das Programm diese Meldung auf den Bildschirm:

```
Zur Löschung markiert (nach Ihren Angaben)
```

Dann können noch weitere Namen zum Löschen eingegeben werden.

Stehen mehrere Personen mit demselben Nachnamen in der Datenbank, zeigt das Programm alle an und fragt nach weiteren Anweisungen.

Als Vorsichtsmaßnahme zeigt das Programm alle zum endgültigen Löschen markierten Namen noch einmal an, bevor es zum Hauptmenü zurückkehrt. Hat der Benutzer beispielsweise vier Personen zum Löschen markiert, wird das Programm alle auf dem Bildschirm mit den Datensatznummern links anzeigen und um Erlaubnis bitten, diese alle endgültig löschen zu dürfen, wie hier zu sehen ist:

```
00999   *Zach Anton       IBM Co.          Astr. 123
02345   *Muriel Duval     AAA Computers    Westweg 12
00055   *Rick Rosellio    Pizza & Co       Flutgraben 1
01234   *Anna Tumeo       Logic GmbH       Postfach 7777

Alle löschen? (J/N) :
```

So kann der Benutzer noch einmal genau prüfen, wer gelöscht werden soll und wenn er jemand entdeckt, der besser in der Datenbank bleibt, kann

er mit N (nein) auf die oben gestellte Frage antworten. Wenn das der Fall ist, fragt das Programm:

```
Welchen nicht löschen? (links stehende Ziffer
                                    eingeben)
```

Der Benutzer kann dann die leicht lesbare Datensatznummer der Person eingeben, die nicht gelöscht werden soll. Beispielsweise entscheidet sich der Benutzer, Muriel Duval nicht zu löschen. Er gibt dazu die entsprechende Datensatznummer (in diesem Fall 02345) als Antwort auf die oben gestellte Frage ein. Das Programm entfernt die Löschmarkierung von Muriels Datensatz und zeigt die übriggebliebenen Namen und Nummern auf dem Bildschirm, wie hier gezeigt, an:

```
00999  *Zach Anton       IBM Co.         Astr. 123
00055  *Rick Rosellio    Pizza & Co      Flutgraben 1
01234  *Anna Tumeo       Logic GmbH      Postfach 7777

Die alle löschen? (J/N) :
```

Wieder kann der Benutzer die Liste überprüfen und noch mehr Leute vor der Löschung retten, wenn er will. Möchte er weitermachen und die angezeigten Einträge löschen, gibt er ein J (ja) ein und auf dem Bildschirm erscheint:

```
Lösche die angegebenen Namen endgültig ...
```

Dabei werden die Einträge mit dem Befehl PACK endgültig von der Datenbank (und der Index-Datei) gelöscht.

Das Programm DELNAMES.CMD bewerkstelligt diese verschiedenen Möglichkeiten beim Löschen aus einer Datenbank. Der Pseudocode ist fast mit dem des Programms EDIT identisch, bis auf die Prüfroutine vor dem endgültigen Löschen der Daten. Die Abb. 7.4 zeigt den Pseudocode des Programms DELNAMES.

Die Programmzeilen mit der Frage nach den zu löschenden Daten ist fast indentisch mit denen, die nach dem zu ändernden Namen fragen. Deshalb zeige ich die Zeilen hier im Überblick:

```
********************************** DELNAMES.CMD
********** Namen aus der Datenbank löschen
********** Zuerst den zu löschenden Namen erfragen
STORE "X" TO suchen
DO WHILE suchen # " "
```

```
Schleifenvariable installieren

Solange der Benutzer nicht aufhören möchte

    Bildschirm löschen
    Benutzer nach dem zu löschenden Namen Fragen
    Namen in Großbuchstaben übertragen
    Namen finden
    Anzahl der gleichen Namen feststellen

    Wenn kein Name zum Löschen angegeben
        Zweimal prüfen, bevor endgültig gelöscht wird

    Wenn der Name nicht zu finden ist
        Benutzer informieren, erneuten Versuch ermöglichen

    Wenn Name einmal gefunden
        Eintrag löschen, Benutzer darüber informieren

    Wenn mehrere Einträge gefunden
        Namen anzeigen, weitere Anweisungen holen
        Dann angegebene Namen löschen

Weiter fragen, bis Benutzer aufhören möchte

Wenn alle zu löschenden Namen eingegeben sind

    Anzeigen, welche Namen gelöscht werden sollen
    Löscherlaubnis beim Benutzer einholen
    Wenn Erlaubnis nicht erteilt wird
        Fragen, welcher nicht gelöscht werden soll
        Den angegebenen Namen zurückholen (recall)
        Restliche Namen anzeigen

    Wenn keine Namen mehr zurückgeholt werden sollen
        Die restlichen löschen

Variablen löschen
Zurück zum Hauptmenü
```

Abb. 7.4: Pseudocode für DELNAMES.CMD

```
ERASE
ACCEPT "Wen löschen? (Nachname oder <RET> für
                              ENDE)" TO suchen
ERASE

******** Übersetze in Großbuchstaben, dann zähle,
******** wie oft der Name vorkommt
STORE!(suchen) TO suchen
FIND &suchen
COUNT WHILE !(nachname) = suchen TO wie viele

***** Mache weiter in Abhängigkeit von der Anzahl
***** der gefundenen Namen
DO CASE

   ****************** Wenn kein Name eingegeben,
   ****************** Bildschirm löschen
   CASE suchen=" "
       ERASE

   **** Wenn niemand mit dem Namen gefunden wird,
   **** Benutzer informieren
   CASE wie viele = 0
       ? "Kein &suchen in dieser Datenbank"
       ?
       ? "Weiter mit beliebiger Taste ... "
       WAIT

   ****************** Wenn eine Person den
   ****************** richtigen Nachnamen hat,
   ****************** AMASK1.FMT aufrufen und
   ****************** ändern lassen
   CASE wie viele = 1
       FIND &suchen
       ? "Zur Löschung markiert (nach Ihren
       Angaben)",TRIMM(vorname),nachname DELETE

   ************ Wenn mehrere Personen den
   ************ gesuchten Namen haben,
   ************ zeige sie an und hole
   ************ weitere Informationen
   CASE wieviel > 1
   FIND &suche
   LIST WHILE !(nachname) = suchen;
```

```
            TRIM(vorname),nachname,firma,strasse
            ?
            ACCEPT "Welchen wählen Sie ? (Nummer
                              eingeben)" TO satznr
            IF VAL(satznr)>0
               GOTO &satznr
               DELETE
            ENDIF (satznr>0)

      ENDCASE

ENDDO (while suchen # " ")
```

Die restlichen Zeilen des DELNAMES-Programms überprüfen, ob die Erlaubnis zum endgültigen Löschen der angezeigten Namen vorliegt, bevor diese wirklich gelöscht werden. Der erste Schritt ist hier, festzustellen, wie viele Einträge zu löschen sind. Der Befehl COUNT FOR * erledigt diese Aufgabe:

```
****** Vor dem endgültigen Löschen und der Rückkehr
****** zum Hauptmenü Löscherlaubnis einholen
COUNT FOR * TO nr:weg
```

Als nächstes vermutet das Programm, daß einige Datensätze vielleicht zurückgeholt werden sollen (durch die Speicherung eines N in der Variablen WEGDAMIT). Es zeigt deshalb den Nachnamen, die Firma und die Straße zu jedem Datensatz, der gelöscht werden soll und fragt nach, ob jemand zurückzuholen ist, wie Sie in den Zeilen hier sehen:

```
STORE "N" TO wegdamit
DO WHILE !(wegdamit) # "J" .AND. nr:weg >0
   ERASE
   DISPLAY ALL FOR * TRIM(vorname),nachname,firma,
                                          strasse
   ?
   ACCEPT "Die alle löschen? (J/N)" TO wegdamit
```

Entschließt sich der Benutzer dazu, einen Namen zurückzuholen, fragt das Programm, wen es zurückholen soll, und subtrahiert 1 von der Anzahl der zu löschenden Personen (NR:WEG):

```
****************** Wenn nicht alle löschen,
****************** herausfinden, wen nicht
IF !(wegdamit) # "J"
```

```
      ?
      ACCEPT "Welchen nicht löschen? (links stehende
                        Ziffer eingeben)" TO sichern
      IF VAL(sichern)>0
         RECALL RECORD &sichern
         STORE nr:weg - 1 TO nr:weg
      ENDIF (val(sichern>0))
   ENDIF (wegdamit # "J")
   ENDDO (while wegdamit # J)
```

Sobald der Benutzer die Genehmigung zum Löschen erteilt, werden alle angegebenen Namen endgültig gelöscht. Da beide Index-Dateien NAMEN und PLZ zu dieser Zeit aktiviert sind, sind die gelöschten Namen darin auch dort nicht mehr enthalten. Diese Zeilen sorgen dafür:

```
   ********* Wenn Löschen erlaubt und Namen vorhanden,
   ********* lösche
   ERASE
   IF nr:weg > 0
      ? "Lösche die angegebenen Namen endgültig ..."
      PACK
   ENDIF (Zahl der Löschungen >0)
```

Jetzt gilt es nur noch, die Speichervariablen zu löschen und zum Hauptmenü zurückzukehren, was mit den folgenden Zeilen geschieht:

```
   RELEASE wegdamit,suchen,wie viele,nr:weg
    ?
   ? "Bin auf dem Weg zum Hauptmenü ..."
   RETURN
```

Das gesamte Programm DELNAMES.CMD ist in der Abb. 7.5 zu sehen.

```
********************************** DELNAMES.CMD
********** Namen aus der Datenbank löschen

*************** Zuerst den zu löschenden Namen erfragen
STORE "X" TO suchen
DO WHILE suchen # " "
   ERASE
   ACCEPT "Wen löschen? (Nachname oder <RET> für ENDE)" TO suchen
   ERASE
```

Abb. 7.5: Das Programm DELNAMES.CMD

```
*************** Übersetze in Großbuchstaben, dann zähle,
*************** wie oft der Name vorkommt
STORE!(suchen) TO suchen
FIND &suchen
COUNT WHILE !(nachname) = suchen TO wieviele

************** Mache weiter in Abhängigkeit von der Anzahl
************** der gefundenen Namen
DO CASE

   ***************** Wenn kein Name eingegeben, Bildschirm
   ***************** löschen
   CASE suchen=" "
       ERASE

   *********** Wenn niemand mit dem Namen gefunden wird,
   *********** Benutzer informieren
   CASE wieviele = 0
       ? "Kein &suchen in dieser Datenbank"
       ?
       ? "Weiter mit beliebiger Taste ... "
       WAIT

   ***************** Wenn eine Person den richtigen Nachnamen
   ***************** hat, AMASK1.FMT aufrufen und ändern
   ***************** lassen
   CASE wieviele = 1
   FIND &suchen
       ? "Zur Löschung markiert (nach Ihren Angaben)",
                                TRIM(vorname),nachname
       DELETE

   ************ Wenn mehrere Personen den gesuchten Namen haben
   ************ zeige sie an und hole weitere Informationen
   CASE wieviele > 1
       FIND &suchen
       LIST WHILE !(nachname) = suchen;
            TRIM(vorname),nachname,firma,strasse
            ?
            ACCEPT "Welchen wählen Sie ? (Nummer eingeben)"
                                                    TO satznr
            IF VAL(satznr)>0
               GOTO &satznr
               DELETE
            ENDIF (satznr>0)

   ENDCASE
```

Abb. 7.5: Das Programm DELNAMES.CMD (Forts.)

```
ENDDO (while suchen # " ")

**************** Vor dem endgültigen Löschen und der Rückkehr
**************** zum Hauptmenü Löscherlaubnis einholen
COUNT FOR * TO nr:weg
STORE "N" TO wegdamit
DO WHILE !(wegdamit) # "J" .AND. nr:weg >0
   ERASE
   DISPLAY ALL FOR * TRIM(vorname),nachname,firma,strasse
   ?
   ACCEPT "Die alle löschen? (J/N)" TO wegdamit
   ************************** Wenn nicht alle löschen,
   ************************** herausfinden, wen nicht
   IF !(wegdamit) # "J"
      ?
      ACCEPT "Welchen nicht löschen? (links stehende Ziffer
                                              eingeben)" TO sichern
      IF VAL(sichern)>0
         RECALL RECORD &sichern
         STORE nr:weg - 1 TO nr:weg
      ENDIF (val(sichern>0))
   ENDIF (wegdamit # "J")
ENDDO (while wegdamit # J)
**************** Wenn Löschen erlaubt und Namen vorhanden,
**************** lösche
ERASE
IF nr:weg > 0
   ? "Lösche die angegebenen Namen endgültig ..."
   PACK
ENDIF (Zahl der Löschungen >0)
RELEASE wegdamit,suchen,wieviele,nr:weg
?
? "Bin auf dem Weg zum Hauptmenü ..."
RETURN
```

Abb. 7.5: Das Programm DELNAMES.CMD (Forts.)

Zusammenfassung

In diesem Kapitel haben Sie benutzerfreundliche Techniken zum Ändern und Löschen von Daten aus einer Datenbank kennengelernt. Wenn der Benutzer die Datenbank verändern möchte, tippt er nur den zu ändernden Nachnamen ein. Steht dieser Name nicht in der Datenbank, sagt das Programm Bescheid und gibt ihm die Möglichkeit, es noch einmal zu versuchen. Steht der gesuchte Name nur einmal in der Datenbank, kann der Benutzer diesen Datensatz gleich in einer Maske bearbeiten, die der Eingabemaske für Neueinträge sehr ähnlich ist.

Wird ein Name eingegeben, der häufiger in der Datenbank vorkommt, (Schmidt zum Beispiel), zeigt das Programm alle Schmidts und die dazugehörenden Datensatznummern an. Der Benutzer kann dann aussuchen, welchen Schmidt er ändern will, indem er die angezeigte Nummer eingibt. Das befreit den Benutzer davon, in der Datenbank nach Datensatznummern suchen zu müssen, und erlaubt es ihm, Änderungen sozusagen in natürlicher Weise vornehmen zu können: auf der Basis von Namen statt Nummern.

Die Prozedur zum Löschen ist der Änderungsprozedur sehr ähnlich. Der Benutzer gibt nur den zu löschenden Namen ein, und das Programm regelt den Rest von hier ab. Als Zusatzsicherung zeigt das Löschprogramm die zu löschenden Namen noch einmal an, bevor sie endgültig gelöscht werden. Das gibt dem Bediener die Möglichkeit, seine Meinung zu ändern, bevor alles endgültig gelöscht worden ist.

Im nächsten Kapitel werden wir Möglichkeiten diskutieren, die es dem Benutzer gestatten, anhand von Nachname und Vorname Informationen zu einer einzelnen Person zu bekommen. Wir werden auch ein Programm entwickeln, das beim Finden von Dublikaten bei den Namen und den Adressen hilft.

Zusätzliche Optionen

8

Es gibt noch zwei Optionen, die wir zwar in das Adressenverwaltungs-System eingebaut haben, von denen uns aber noch die Programme fehlen. Die eine Option dient zum schnellen Suchen und die andere zum Überprüfen der Datenbank auf Duplikate.

Schnellsuche nach einem einzelnen Eintrag

Die Schnellsuchfunktion ist so etwas wie ein Telefonbuch mit einem ausgefeilten Index. Der Benutzer wählt diesen Menüpunkt an, gibt den Nachnamen und den Vornamen oder den ersten Buchstaben davon ein, und das Programm zeigt die Informationen zu dem Gesuchten auf dem Bildschirm oder druckt sie in der folgenden Form aus:

```
Nachname  :  Cusey
Vorname   :  Julie
Firma     :  Ferien AG
Strasse   :  Sonnenuhr 24
PLZ       :  9212
Ort       :  Urlaubshausen
Land      :  Deutschland
Telefon   :  08999-55555
Swort     :  Ferien, Flugreisen
```

Die Schnellsuchfunktion ist schneller, wenn wir den FIND-Befehl benutzen, um in der Index-Datei NAMEN.NDX den Namen zu suchen. Erinnern Sie sich daran, daß der Index mit diesem Befehl erstellt wurde:

```
INDEX ON !(nachname) + !(vorname) TO namen
```

Aus diesem Grunde enthält die Datei NAMEN.NDX den Vor- und Nachnamen jeder Person in der Datenbank, alle in Großbuchstaben und sortiert (Indizes sind immer sortiert). Also sehen die Daten in der Index-Datei etwa so aus:

```
ADAMS           ANDY
BLOMGREN        LYNEA
BREWER          ADOLPH
CAPUSCHNIAK     IGOR
DANIELS         KARIN
DANIELS         EVA
EDWARDS         EDWARD
```

Beachten Sie, daß alle Nachnamen noch Leerzeichen hinter sich haben. Da das NACHNAME-Feld eine Größe von 20 Zeichen bei der Erstellung zugeteilt bekam, ist es auch in der Index-Datei 20 Zeichen groß. Wenn nun ein Benutzer nach Vor- und Nachnamen suchen will, muß das Programm dafür sorgen, daß die Variable, in die der Suchname gespeichert wird, ebenfalls Platz für 20 Zeichen hat.

Hier sehen Sie eine Schnellsuche aus der Anwendersicht. Zuerst wählt er die Option „3. Schnellsuchen". Der Bildschirm zeigt dann:

```
Nachnamen eingeben :                    :
Vornamen eingeben  :                    :

Daten ausdrucken?  : :
```

Gibt der Benutzer den Vor- und Nachnamen ein und ist die gesuchte Person in der Datenbank verzeichnet, zeigt der Bildschirm (oder der Drukker) sehr schnell alle Informationen über diese Person, etwa in dieser Form:

```
Nachname  :  Daniels
Vorname   :  Karin
Firma     :  Bleitreu GmbH & Co KG
Strasse   :  Leihpfad 7
PLZ       :  1234
Ort       :  Borgberg
```

```
Land       :  Deutschland
Telefon    :  01223
Swort      :  Pfandhaus

Mit beliebiger Taste zurück zum Hauptmenü ...
```

Gibt es den Namen in der Datenbank nicht, erscheint eine Meldung wie diese:

```
Karin Daniels steht nicht in dieser Datenbank!
```

An dieser Stelle kann der Benutzer versuchen, einen anderen Namen zu finden. Den Pseudocode für das LOOKUP-Programm finden Sie in der Abbildung 8.1.

Wir fangen das Programm wieder damit an, daß wir die üblichen Kommentare an den Kopf des Programms schreiben. Dann wird das Programm die Variablen NACHSUCH (für den Nachnamen), VORSUCH

```
Erstelle Speichervariablen mit Leerzeichen
Bildschirm löschen

Benutzer nach dem gesuchten Vor- und Nachnamen fragen
Benutzer fragen, ob Daten ausgedruckt werden sollen

Speichere Vor- und Nachname in Großbuchstaben in Suchvariable

Finde diese Person

Wenn Person gefunden
     Daten anzeigen (Bildschirm oder Drucker)

Wenn nicht gefunden
     Fehlermeldung bringen

Pause, damit Benutzer Daten lesen kann

Speichervariablen löschen

Zurück zum Hauptmenü
```

Abb. 8.1: Pseudocode für das Programm LOOKUP.CMD

(für den Vornamen) und DRUCK (um zu entscheiden, ob die Informationen an den Drucker gehen sollen oder nicht) erstellen. Diese Zeilen kümmern sich um die Arbeit:

```
********************************** LOOKUP.CMD
********** Schnellsuche mittels Vor- und Nachname

******* Zuerst 20 Leerzeichen in NACHSUCH und
******* VORSUCH speichern
******* und ein Leerzeichen in DRUCK
STORE "                    " TO nachsuch,vorsuch
STORE " " TO druck
```

Beachten Sie bitte, daß genau 20 Leerzeichen zwischen den Doppelpunkten in der Zeile

```
STORE "                    " TO nachsuch,vorsuch
```

stehen müssen. Diese Angabe muß mit der Angabe zu dem entsprechenden Feld bei der Erstellung der Datenbank genau übereinstimmen.

Der nächste Befehl fragt den Benutzer, nach wem er suchen möchte und ob die Daten ausgedruckt werden sollen oder nicht:

```
********* Herausfinden, wer gesucht wird
ERASE
§ 2,2 SAY "Nachnamen eingeben :       :" GET nachsuch
§ 3,2 SAY "Vornamen eingeben  :       :" GET vorsuch
§ 7,2 SAY "Daten ausdrucken?  : :" GET druck
READ
```

Dann erstellt das Programm eine Variable, die es mit dem FIND-Befehl benutzt, um die Suche durchführen zu können. Die Speichervariable heißt SUCHEN und enthält außer dem Nachnamen, nach dem der Benutzer sucht, auch den Vornamen, dessen Leerstellen beseitigt wurden. Die Großschreib-Funktion (!) stellt sicher, daß alle Zeichen als Großbuchstaben dastehen und so mit dem Inhalt der Index-Datei verglichen werden können.

```
*************** Setze Vor- und Nachname zusammen;
*************** wandle alles
*************** in Großbuchstaben um
STORE !(nachsuch)+!(TRIM(vorsuch)) TO suchen
```

Die TRIM-Funktion gibt dem Benutzer mehr Flexibilität beim Suchen. Wenn er beispielsweise Frau Daniels sucht und diesen Namen eingibt, für

den Vornamen aber nur ein K statt des vollen Vornamens, findet das Programm trotzdem Karin Daniels.

Als nächstes muß sich das Programm anstrengen, den angegebenen Namen mit Hilfe des FIND-Befehls und der SUCHEN-Variablen zu finden, wie die Zeilen hier belegen:

```
****************** Person finden
FIND &suchen
ERASE
```

Findet das Programm die gesuchte Person, zeigt es die Daten auf dem Bildschirm an. Weiterhin schaltet es zuerst den Drucker ein, wenn der Benutzer die Daten ausdrucken möchte. Wird der gesuchte Name nicht gefunden, erscheint eine Fehlermeldung. Die DO CASE-Anweisung hier ist dafür verantwortlich:

```
******* Wenn gefunden, anzeigen. Sonst Fehlermeldung
DO CASE
   CASE #>0
   IF !(druck) = "J"
      SET PRINT ON
   ENDIF (druck = J)
?"Nachname  :",nachname
?"Vorname   :",vorname
?"Firma     :",firma
?"Strasse   :",strasse
?"PLZ       :",plz
?"Ort       :",ort
?"Land      :",land
?"Telefon   :",telefon
?"Swort     :",swort
   ?
   EJECT
   SET PRINT OFF

CASE # = 0
   ?
   ? TRIM(vorsuch),TRIM(nachsuch),"gibt es nicht!"

ENDCASE
```

Beachten Sie dabei, daß der FIND-Befehl, wenn er die gesuchten Daten findet, den Datensatzzeiger (#) auf die Datensatznummer des entsprechenden Satzes setzt. Andernfalls setzt er den Datensatzzeiger auf 0. Sie

erkennen das Prinzip in den beiden DO CASE-Anweisungen: 1) CASE # >0 (was bedeutet, der Name wurde gefunden) und 2) CASE # = 0 (der Name wurde nicht gefunden).

Dann sollte das Programm anhalten, um dem Benutzer Zeit zum Lesen der Daten zu lassen. Die nächsten Programmzeilen erledigen das:

```
********** Anhalten, bevor es zum Menü zurückgeht
?
?
?"Mit beliebiger Taste zurück zum Hauptmenü ..."
?
WAIT
```

Dann löscht das Programm die Speichervariablen wieder und kehrt zum Hauptmenü mit diesen Zeilen zurück:

```
********************** Variablen löschen
RELEASE nachsuch,vorsuch,druck,suchen
RETURN
```

Das gesamte LOOKUP-Programm finden Sie in der Abb. 8.2. Wie üblich testen Sie mit DO POST (dem Hauptmenü-Programm) und wählen dann die Option 3, das LOOKUP-Programm.

```
****************************************** LOOKUP.CMD
********** Schnellsuche mittels Vor- und Nachname

******* Zuerst 20 Leerzeichen in NACHSUCH und VORSUCH speichern
******* und ein Leerzeichen in DRUCK
STORE "                    " TO nachsuch,vorsuch
STORE " " TO druck

********* Herausfinden, wer gesucht wird
ERASE
§ 2,2 SAY "Nachnamen eingeben " GET nachsuch
§ 3,2 SAY "Vornamen eingeben  " GET vorsuch
§ 7,2 SAY "Daten ausdrucken?  " GET druck
READ

*************** Setze Vor- und Nachname zusammen; wandle alles
*************** in Großbuchstaben um
STORE !(nachsuch)+!(TRIM(vorsuch)) TO suchen
```

Abb. 8.2: Das Programm LOOKUP.CMD

```
****************** Person finden
FIND &suchen
ERASE

**************** Wenn gefunden, anzeigen. Sonst Fehlermeldung
DO CASE
   CASE #>0
   IF !(druck) = "J"
      SET PRINT ON
   ENDIF (druck = J)
?"Nachname  :",nachname
?"Vorname   :",vorname
?"Firma     :",firma
?"Strasse   :",strasse
?"PLZ       :",plz
?"Ort       :",ort
?"Land      :",land
?"Telefon   :",telefon
?"Swort     :",swort
   ?
   EJECT
   SET PRINT OFF

CASE # = 0
   ?
   ? TRIM(vorsuch),TRIM(nachsuch),"gibt es nicht!"

ENDCASE

******************** Anhalten, bevor es zum Menü zurück geht
?
?
?"Mit beliebiger Taste zurück zum Hauptmenü ..."
?
WAIT

********************** Variablen löschen
RELEASE nachsuch,vorsuch,druck,suchen
RETURN
```

Abb. 8.2: Das Programm LOOKUP.CMD (Forts.)

Wünschen Sie sich das Schnellsuch-Programm so, daß es mit einem Nachnamen oder einem Vor- und einem Nachnamen arbeitet, sollten Sie das Programm leicht überarbeiten. Die Zeile, die die beiden Variablen NACHSUCH und VORSUCH zusammenbringt:

```
STORE !(nachsuch)+!(TRIM(vorsuch)) TO suchen
```

sollte so modifiziert werden, daß der Vorname nur dann zum Nachnamen gestellt wird, wenn der Benutzer nach beiden gleichzeitig suchen möchte. Ersetzen Sie die beschriebene Zeile durch diese Routine:

```
STORE !(nachsuch) TO suchen
********* Vornamen dazu, wenn einer eingegeben wurde
IF vorsuch # " "
   STORE suchen + !(TRIM(vorsuch)) TO suchen
ENDIF (vorsuch # " ")
```

Bei einer großen Datenbank möchten Sie möglicherweise sicherstellen, daß alle gleichen Namen angezeigt werden, wenn verschiedene Personen den gleichen Vor- und Nachnamen haben. Zur Zeit wird die erste in der Datenbank gefundene Person mit dieser Routine angezeigt:

```
   CASE #>0
   IF !(druck) = "J"
      SET PRINT ON
   ENDIF (druck = J)
?"Nachname   :",nachname
?"Vorname    :",vorname
?"Firma      :",firma
?"Strasse    :",strasse
?"PLZ        :",plz
?"Ort        :",ort
?"Land       :",land
?"Telefon    :",telefon
?"Swort      :",swort
?
```

Um das Programm mehrere Personen mit gleichen Namen anzeigen zu lassen, können Sie diese Routine in eine Schleife packen, wie ich es Ihnen hier zeige:

```
CASE #>0
  **** Zeige alle Personen mit dem gesuchten Namen
  DO WHILE !(nachname)+!(TRIM(vorname)) = suchen
                                  .AND. .NOT. EOF
     ERASE
     IF !(druck) = "J"
        SET PRINT ON
     ENDIF (druck = J)
     ?"Nachname   :",nachname
     ?"Vorname    :",vorname
```

```
      ?"Firma      :",firma
      ?"Strasse    :",strasse
      ?"PLZ        :",plz
      ?"Ort        :",ort
      ?"Land       :",land
      ?"Telefon    :",telefon
      ?"Swort      :",swort
      ?
      ?
      EJECT
      SKIP
      SET PRINT OFF

      ********* Wenn der nächste Name auch paßt,
      ********* frage Benutzer
      ********* was zu tun ist
      IF !(nachname)+!(TRIM(vorname) = suchen
         ? "Mit beliebiger Taste zum nächsten Namen
                                    oder X für ENDE"
         WAIT TO mehr
         ********** Will Benutzer aufhören,
         ********** verlasse die Schleife
         IF !(mehr)="X"
            GO BOTT
         ENDIF (mehr=X)
      ENDIF (nachname=suchen)
ENDDO (name=suchen und nicht eof)
```

Duplikate schnell finden

In dem Buch „Arbeiten mit dBASE II" lieferte ich ein Programm, mit dem man schnell nach doppelten Einträgen in einer Datenbank suchen und diese anzeigen lassen kann. Das Adressenverwaltungs-Programm, daß wir gerade entwickeln, wird die gleiche Aufgabe in sehr ähnlicher Weise erledigen, aber den Vorteil der Index-Datei NAMEN dabei ausnutzen. Dadurch müssen Sie für die Suche nach „dem doppelten Lottchen" die Datenbank nicht neu indizieren und es geht spürbar schneller.

Die grundsätzliche Logik beim Auffinden doppelter Einträge besteht darin, alle Namen und Adressen zu sortieren und die Vor- und Nachnamen in eine alphabetische Reihenfolge zu bringen. Die Index-Datei NAMEN ist bereits in dieser Weise vorbereitet. Dann muß das Programm benachbarte Namenpaare lesen. Sind zwei Namen gleich, werden sie logischerweise in der Sortierung nebeneinander stehen. Entdeckt das Programm

zwei gleiche Namen, kann es weiter prüfen, ob vielleicht auch die Straße und der Ort gleich sind. Trifft das zu, kann es den Benutzer darüber informieren. Andernfalls geht es zum nächsten Namenspaar und prüft dort weiter, bis alle Namen und Adressen in der Datenbank verglichen worden sind.

Unser Prüfprogramm erlaubt auch, die möglichen Doppelgänger auf dem Drucker statt auf dem Bildschirm aufzulisten. Es zeigt seinen Bericht in dieser Form:

```
Satz # 123  ist Schmidt Arnold Astr. 123   Köln
Satz # 932  ist Schmidt Arnold Astr. 123   Köln

Satz # 39  ist Müller Franz  Tauwehr 12    Emden
Satz # 321 ist Müller Franz  Tauwehr 12    Emden
```

Das Programm löscht keinen Datensatz. Es informiert den Benutzer nur über das Vorhandensein von möglichen Dubletten. Das ermöglicht diesem, weitere Nachforschungen über die vermeintlichen oder tatsächlichen Doppelgänger anzustellen, bevor er sie über die entsprechende Option im Hauptmenü löscht.

Findet das Programm keine Übereinstimmungen, erscheint diese Meldung auf dem Bildschirm:

```
Keine Duplikate gefunden!
Weiter mit beliebiger Taste ...
```

Auf den Druck einer beliebigen Taste hin, kehrt das Programm dann wieder zum Hauptmenü zurück.

Das Programm DUPES übernimmt die Prüfung auf doppelte Eintragungen. Der logische Ablauf des Programms ist in Abb. 8.3 zu sehen.

Das POST-Programm hat schon die Index-Datei NAMEN als ersten Index aufgerufen, weshalb das DUPES-Programm keinen neuen Index erstellen muß. Statt dessen wird es den Bildschirm löschen und fragen, ob die möglichen Duplikate auf dem Bildschirm angezeigt werden sollen. Es speichert die Benutzerantwort in der Variablen JN (ja/nein). So sehen die ersten Zeilen des Programms aus:

```
************************* DUPES.CMD
********************** Prüfen auf doppelte Einträge
ERASE

***************************** Fragen, ob drucken
STORE " " TO jn
```

```
§  2,2 SAY "Sollen die Duplikate ausgedruckt
                                        werden?";
           GET jn
READ
```

Bildschirm löschen

Fragen, ob Daten ausdrucken

Wenn ja, Drucker einschalten

Bildschirm löschen
Zähler für Doppeleintragungen auf 0

Reportüberschrift und zwei Leerzeilen drucken

Datenbank durchsuchen und Paare vergleichen, beginnend mit 1
    Name,Straße und Ort in Variable 1
    Einen Satz weitergehen
    Wenn nicht Ende der Datenbank
    Name,Straße und Ort in Variable 2
        Wenn beide identisch
        Einen Satz zurück
        Diese Daten anzeigen
        Einen Satz vor
        Diese Daten auch anzeigen
        Zwei Leerzeilen drucken
        Zähler für Duplikate um 1 erhöhen
    Speichervariablen mit Leerzeichen füllen
Weitermachen mit der Überprüfung (bis Dateiende)

Drucker ausschalten

Wenn keine Doppeleintragungen gefunden, Bescheid geben
Wenn Duplikate nicht ausgedruckt werden, Bildschirm anhalten

Variablen löschen
Zurück zum Hauptmenü

Abb. 8.3: Pseudocode für das DUPES.CMD-Programm

Gibt der Benutzer als Antwort J (oder j) ein, schaltet das Programm den Drucker ein, wie Sie in den folgenden Programmzeilen sehen können:

```
********************* Wenn ja, Drucker einschalten
IF !(jn)="J"
   SET PRINT ON
ENDIF (jn=j)
```

Als nächstes wird das Programm den Bildschirm löschen und die Überschrift des Reports, gefolgt von zwei Leerzeichen, drucken. Es speichert auch eine 0 in die Variable DUPZAEHL, worin festgehalten wird, wie viele Doppelgänger entdeckt wurden. Den Programmtext schreiben Sie so:

```
************************** Dubletten-Report drucken
ERASE
STORE 0 TO dupzaehl

? "Möglicherweise Doppeleintragungen ...."
?
?
```

Das Programm muß jetzt sicher sein können, daß es am Anfang der Datenbank steht. Dann beginnt eine Schleife durch alle Datensätze.

```
************** Datenbank durchlaufen und vergleichen
GO TOP
DO WHILE .NOT. EOF
```

Für jeden Datensatz in der Datenbank speichert das Programm jetzt Daten in eine AD1 genannte Variable. Das geschieht in dieser Zeile:

```
STORE nachname + strasse + ort TO ad1
```

Dann geht das Programm weiter in die nächste Zeile und macht dort einen Vergleich. Damit das Programm aber richtig arbeiten kann muß auch sichergestellt sein, daß es nicht über das Ende der Datenbank hinauskommt. Wenn es nicht am Ende der Datenbank angelangt ist, speichert es den Namen, die Straße und den Ort mit der Postleitzahl in die Variable AD2.

```
SKIP 1
IF .NOT. EOF
   STORE nachname+strasse+ort TO ad2
ENDIF (not eof)
```

Jetzt prüft es nach, ob die beiden entsprechenden Datensätze in den Feldern Name, Straße, und Ort (für höhere Genauigkeit werden Wörter in Großbuchstaben verglichen) gleich sind. Wenn ja, muß das Programm ei-

nen Datensatz zurückgehen, die Daten für diesen Datensatz anzeigen, dann wieder einen Datensatz weiter nach vorn und ebenfalls die Daten dort anzeigen. Dann sollte es einige Leerzeilen drucken und den Dublettenzähler (DUPZAEHL) um 1 erhöhen. Diese Aufgabe wird von den Programmzeilen innerhalb der IF-Bedingung erledigt:

```
********************* Bericht über die Dubletten
IF !(ad1)=!(ad2)
   SKIP -1
   ? "Satz # " +STR(#,3)+"ist ",TRIM(nachname),;
             TRIM(vorname),TRIM(strasse),ort
   SKIP 1
   ? "Satz # " +STR(#,3)+"ist ",TRIM(nachname),;
             TRIM(vorname),TRIM(strasse),ort
   ?
   ?
   STORE dupzaehl+1 TO dupzaehl
 ENDIF (ad1=ad2)
```

In der nächsten Zeile löscht das Programm erst einmal den Inhalt der Speicher-Variablen AD1 und AD2, bevor der nächste Vergleich ausgeführt wird.

```
STORE " " TO ad1,ad2
```

Dann geht die Schleife einfach weiter (bis das Ende der Datenbank erreicht ist) und sucht nach doppelten Einträgen. Wir müssen jetzt nur noch eine Zeile anhängen, mit der die Schleife geschlossen wird:

```
ENDDO (while not eof)
```

Nachdem alle Datensätze auf doppelte Einträge durchsucht sind, schaltet das Programm den Drucker mit diesen Befehlen wieder aus:

```
***************************** Ende des Programms
SET PRINT OFF
```

Wenn keine doppelten Einträge gefunden wurden, soll das Programm dem Benutzer Bescheid sagen, damit er weiß, daß das Programm gearbeitet hat. Da in der Variablen DUPZAEHL jede doppelte Eintragung festgehalten wird, kann das Programm leicht feststellen, ob Dubletten vorhanden sind oder nicht, indem es DUPZAEHL auf den Inhalt Null (0) hin überprüft. Die nächsten Programmzeilen informieren den Benutzer darüber, daß keine doppelten Eintragungen gefunden wurden und lassen ihn den Bildschirm lesen, bevor es zum Hauptmenü zurückgeht.

```
******** Wenn nichts gefunden, Benutzer informieren
IF dupzaehl = 0
   ERASE
   ? "Keine doppelten Einträge gefunden!"
   ? "Mit beliebiger Taste zurück zum Hauptmenü..."
   ?
   WAIT
ENDIF (dupzaehl=0)
```

Wurden doppelte Einträge gefunden und angezeigt, aber nicht zum Drucker geschickt, soll das Programm anhalten. Der Benutzer kann dann die Daten auf dem Bildschirm lesen, bevor das Programm zum Hauptmenü zurückkehrt. Die nächsten Zeilen erledigen das:

```
********** Wenn Duplikate nicht ausgedruckt werden,
********** pausieren
IF !(jn)#"J" .AND. dupzaehl > 0
    ?
    ?
    ? "Mit beliebiger Taste zurück zum Hauptmenü..."
    ?
    WAIT
ENDIF (jn#J und dupzaehl > 0)
```

Jetzt bleibt uns nur noch die Variablen zu löschen und zum Hauptmenü zurückzukehren, wie in den folgenden Zeilen gezeigt:

```
******************** Zurück zum Hauptmenü
RELE ad1,ad2,jn,dupzaehl
RETURN
```

In der Abb. 8.4 finden Sie das gesamte DUPES-Programm. Wenn Sie es ausprobieren wollen, geben Sie DO MAIL ein und wählen die Option „6. Duplikate suchen" aus dem Hauptmenü.

```
********************************** DUPES.CMD
********************************** Prüfen auf doppelte Einträge
ERASE

********************************** Fragen, ob drucken
STORE " " TO jn
§  2,2 SAY "Sollen die Duplikate ausgedruckt werden?";
           GET jn
READ
```

Abb. 8.4: Das Programm DUPES.CMD

```
************************ Wenn ja, Drucker einschalten
IF !(jn)="J"
   SET PRINT ON
ENDIF (jn=j)

************************* Dubletten-Report drucken
ERASE
STORE 0 TO dupzaehl

? "Möglicherweise Doppeleintragungen ..."
?
?

************************ Datenbank durchlaufen und vergleichen
GO TOP
DO WHILE .NOT. EOF
   STORE nachname + strasse + ort TO ad1
   SKIP 1
   IF .NOT. EOF
      STORE nachname+strasse+ort TO ad2
   ENDIF (not eof)

   ********************* Bericht über die Dubletten
   IF !(ad1)=!(ad2)
      SKIP -1
      ? "Satz # " +STR(#,3)+"ist ",TRIM(nachname),;
                TRIM(vorname),TRIM(strasse),ort
      SKIP 1
      ? "Satz # " +STR(#,3)+"ist ",TRIM(nachname),;
                TRIM(vorname),TRIM(strasse),ort
      ?
      ?
      STORE dupzaehl+1 TO dupzaehl
    ENDIF (ad1=ad2)

    STORE " " TO ad1,ad2

ENDDO (while not eof)

***************************** Ende des Programms
SET PRINT OFF

******************* Wenn nichts gefunden, Benutzer informieren
IF dupzaehl = 0
   ERASE
   ? "Keine doppelten Einträge gefunden!"
   ? "Mit beliebiger Taste zurück zum Hauptmenü ..."
```

Abb. 8.4: Das Programm DUPES.CMD (Forts.)

```
   ?
   WAIT
ENDIF (dupzaehl=0)

********** Wenn Duplikate nicht ausgedruckt werden, pausieren
IF !(jn)#"J" .AND. dupzaehl > 0
    ?
    ?
    ? "Mit beliebiger Taste zurück zum Hauptmenü..."
    ?
    WAIT
ENDIF (jn#J und dupzaehl > 0)

 ******************** Zurück zum Hauptmenü
RELE ad1,ad2,jn,dupzaehl
RETURN
```

Abb. 8.4: Das Programm DUPES.CMD (Forts.)

Zusammenfassung

Sie haben jetzt gelernt, wie man ein gutes Adressenverwaltungs-Programm gestaltet und entwickelt. Mit dem nächsten Kapitel beginnend werden wir ein anderes Programm-System gestalten und entwickeln: Ein Inventarsystem, das mit mehreren Datenbanken und mehreren Index-Dateien arbeitet. Damit bekommen wir natürlich schwierigere Aufgaben zu lösen. Aber wir werden viele der bisher gelernten und erarbeiteten Techniken in unser Inventarsystem übernehmen können, wodurch die Arbeit dann doch etwas leichter wird.

Ein Inventarsystem

9

In diesem Kapitel beginnen wir, ein Inventarsystem zu entwickeln. Während der Entwicklung werden wir einige neue Programmiertechniken diskutieren, von denen einige sind:

1. Überprüfen der richtigen Eingabe,
2. Überprüfen auf doppelte Einträge,
3. Aktualisieren einer Hauptdatei mit Bewegungsdateien,
4. Daten automatisch berechnen und das Ergebnis anzeigen,
5. bestimmte Felder einer Datenbank automatisch füllen,
6. Suchen nach Daten in einem bestimmten Bereich.

Wir beginnen mit der Diskussion der Gestaltung von Inventarsystemen.

Grundsätzlich verlangen Inventarsysteme das Arbeiten mit mehreren Datenbanken. Eine Datenbank, normalerweise Hauptdatei genannt, enthält die Menge jedes Teils, das zur Zeit auf Lager liegt. Zusätzlich kann sie auch die Kosten dafür, die Mindestmenge, die Bestellmenge, den Lagerort im Warenhaus sowie Namen und Adresse des Lieferanten enthalten. Auf diese Weise kann der Benutzer ein Teil sehr schnell finden, eine Liste der notwendigen Nachbestellungen anfertigen und eine Bestellung automatisch ausdrucken.

Die meisten Geschäftsbereiche möchten auch eine Übersicht über jeden Geschäftsvorgang innerhalb des Geschäftes. Einzelne Geschäftsvorgänge werden grundsätzlich mit eigenen Datenbanken behandelt, die dann

Datenbanken für Geschäftsvorgänge heißen. Eine Datei ist vielleicht für Verkäufe (z.B.: an wen etwas verkauft wurde, wann, wie teuer etc.) und die entsprechenden Bestell- und Rechnungsnummern sinnvoll. Eingehende Waren könnten auch in einer eigenen Datei registriert werden, um eine fortlaufende Geschichte aller erhaltenen Waren zu bekommen. Die Beziehungen zwischen der Haupt- und den beiden Geschäftsvorgänge-Dateien können wir so aufzeigen, wie sie in der Abb. 9.1 zu sehen sind.

Die Dateien für die Vorgänge ermöglichen die „Geschichtsschreibung" sämtlicher Einzelvorgänge, einschließlich der Verkäufe und des Warenerhalts, während die Hauptdatei zu jeder Zeit den Status der Waren im

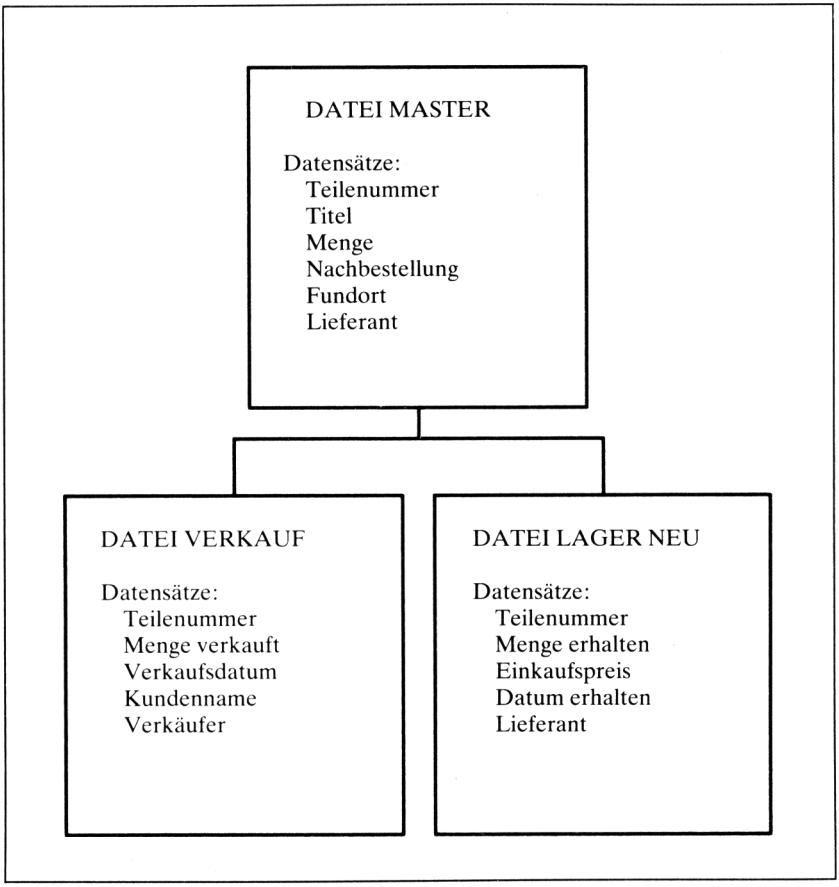


Abb. 9.1: Die Beziehungen der Datenbanken des Inventarsystems

Lager wiederspiegelt. Die Hauptdatei bleibt durch die Eingaben von den beiden anderen Dateien ständig auf dem neuesten Stand. In gewissem Sinne ist die Hauptdatei eine fortlaufende Zusammenfassung aller Ereignisse, die in den beiden anderen Dateien festgehalten wurden.

Wann immer Sie ein System mit mehrfach verknüpften Datenbanken wie dieses hier erstellen, ist es sehr wichtig, ein Schlüsselfeld zu definieren, über welches die Informationen in den unterschiedlichen Datenbanken verknüpft werden können. Ein Schlüsselfeld besitzt folgende Eigenarten:

1. Es ist in jedem Datensatz der Hauptdatei gleich;
2. es existiert in der Haupt- wie in den Nebendateien mit gleichem Feldnamen, Typ und gleicher Größe.

Das Schlüsselfeld in den meisten Software-Systemen ist meist ein Erkennungscode. In einem Inventarsystem wie diesem wird normalerweise eine Teilenummer als Schlüsselfeld benutzt. Ein Unternehmen verwaltet jedes Teil im Lager, indem es dieses über die Teilenummer identifiziert. Etwa so, wie die Sozialversicherung jedes Mitglied unter seiner Sozialversicherungsnummer speichert. Ausweisnummern, Kontonummern und Versicherungsnummern dienen überall als Schlüsselfelder in den verschiedenen Computern.

Ohne Schlüsselfelder sind Daten fast überhaupt nicht zu verarbeiten. Würde zum Beispiel das Finanzamt ohne Schlüsselnummern arbeiten, müßten die Mitarbeiter dort meine Steuerschulden dadurch ermitteln, daß sie die Steuerbescheide aller Bürger mit dem Namen Allen Simpson durchsuchen. Indem sie aber einfach meine Steuernummer eintippen, sparen sie viel Zeit und teure Irrtümer. Indem also jedem Steuerzahler eine eigene Nummer zugeteilt und diese als Schlüsselfeld benutzt wird, spart das Finanzamt erhebliche Mittel, und die Arbeit wird einfacher.

Selbst in einem so kleinen System wie unserem Inventarprogramm, sind Schlüsselfelder notwendig, wie wir bei der Entwicklungsarbeit sehen werden.

Die Ziele des Inventarsystems

Das Inventarsystem soll es einem kleinen Geschäft ermöglichen, einen Überblick über die Teile im Lager und die Bestellungen zu bekommen. Wann immer das Unternehmen sich entschließt, einen neuen Artikel in den Verkauf mit aufzunehmen, gibt es dem neuen Teil eine eigene Nummer und speichert diese in der Inventar-Hauptdatei. Das System ermöglicht es dem Benutzer auch, Berichte über alle Teile im Lager, Nachbestellteile und Teile, die bereits bestellt sind, auszudrucken. Bestellungen können auch automatisch abgewickelt werden.

Das System gibt dem Benutzer auch die Möglichkeit, jeden einzelnen Verkauf zu verfolgen: die verkauften Teile, den Käufer, den Verkäufer, das Verkaufsdatum und die Rechnungsnummer. Genauso kann der Benutzer alle eingehenden Waren nachhalten, wenn sein Lager aufgefüllt wird. Die einzelnen Vorgänge aus den Ver- und Einkäufen werden benutzt, um den Status des Lagers immer auf dem laufenden zu halten.

Die Datenbanken des Inventarsystems

Drei Datenbanken werden benutzt, um das Inventar zu verwalten. Die erste heißt MASTER.DBF. Ihre Struktur und eine kurze Beschreibung jedes Feldes zeige ich Ihnen hier:

```
STRUKTUR DER DATEI: B:MASTER.DBF

FELDNR  NAME       TYP     WEITE   DEZ   Inhalt
001     TEIL:NR     C      005           Teilenummer=
                                         (Schlüsselfeld)
002     TITEL       C      020           Teilebeschreibung
003     MENGE       N      004           Menge im Lager
004     KOSTEN      N      009     002   Einkaufspreis
005     NACHBEST    N      004           Nachbestellung
006     BESTELL     N      004           Bestellmenge
007     FUNDORT     C      005           Lagerort
008     LIEFERANT   C      025           Lieferant
009     LIEF:ADR    C      025             "  Adresse
010     LIEF:PO     C      025             " PLZ,Ort
011     DATUM       C      008           Datum letzte
                                         Aktualisierung
012     BEST:DAT    C      008           Datum letzte Best.
013     NEUBEST     N      004           Bestellmenge
```

Sie können diese Datenbank nach den hier aufgeführten Informationen mit dem dBASE-Befehl CREATE aufbauen, indem Sie die Feldbezeichnungen, Typen, Weiten und Dezimalstellen einfach übernehmen (nicht die Kommentare).

Um einmal die Datenbank nach Teilenummern sortiert zu bekommen und zweitens die Geschwindigkeit beim Aktualisieren und Suchen zu erhöhen, erstellen wir jetzt noch die Index-Datei MASTER.NDX. Nachdem Sie die Datenbank MASTER.DBF fertig haben, können Sie den Index so erstellen:

```
USE master
INDEX ON teil:nr TO master
```

Denken Sie daran, mit dem Befehl SET DEFAULT TO das entsprechende Laufwerk anzuwählen, bevor Sie die Datenbank und den Index erstellen (bei den Beispielen in diesem Buch ist das immer Laufwerk B).

Jetzt wollen wir eine Datei für die einzelnen Verkäufe erstellen. Beachten Sie bitte, daß in der eben erstellten Datenbank MASTER.DBF das Feld TEIL:NR das Schlüsselfeld darstellt. Es ist vom Typ (C) und bietet fünf Zeichen Platz. Da dies das Schlüsselfeld zum Aktualisieren der Hauptdatei sein wird, muß die Verkäufe-Datei ein gleiches Feld enthalten. Die Datenbank für die Verkäufe heißt SALES.DBF und ist so strukturiert:

```
STRUKTUR DER DATEI: B:SALES.DBF

FELDNR  NAME         TYP     WEITE   DEZ   Inhalt
001     TEIL:NR       C      005           Teilenummer
                                           (Schlüsselfeld)
002     RECHNG:NR     N      006           Rechnungsnummer
003     VERKAEUFER    C      012           Verkäufer
004     KUNDE         C      012           Kunde
005     MENGE         N      004           Verkaufte Menge
006     PREIS         N      009     002   Verkaufspreis
007     DATUM         C      008           Verkaufsdatum
008     UEBERTR       L      001           Zur Hauptdatei
                                           geschickt?
```

Das Feld Nummer 008, UEBERTR, ist ein spezielles Feld, in dem festgehalten wird, ob ein bestimmter Vorgang bereits in die Hauptdatei MASTER.DBF übertragen wurde oder nicht. Diese Vorgehensweise ist notwendig, um zu verhindern, daß ein Vorgang zweimal übertragen wird. Beachten Sie, daß dies ein logisches Feld ist (L). Ein solches Feld kann entweder (T)rue = wahr oder (F)alse = unwahr sein. Wir werden unser System so gestalten, daß nach jedem Übertrag das Feld UEBERTR für jeden übertragenen Datensatz auf (T)rue gesetzt wird.

Um die Verarbeitungsgeschwindigkeit zu erhöhen, werden wir auch für die SALES-Datenbank einen Index über die Teilenummer erstellen (SALES.NDX). Sie können jetzt die neue SALES-Datenbank auf derselben Diskette erstellen wie die Datei MASTER.DBF, indem Sie die oben vorgegebenen Informationen benutzen. Dann erstellen Sie mit

```
USE sales
INDEX ON teil:nr TO sales
```

die Index-Datei.

In der dritten Datei stehen eingegangene Waren, mit denen das Lager aufgefüllt wurde. Wieder sollte auch diese Datei das Schlüsselfeld

TEIL:NR haben, da die Hauptdatei Informationen aus der Datenbank für neue Teile bezieht. Die dritte Datenbank heißt NEWSTOCK.DBF und bekommt diese Struktur:

```
STRUKTUR DER DATEI: B:NEWSTOCK.DBF

FELDNR     NAME       TYP    WEITE    DEZ    Inhalt
001        TEIL:NR     C     005             Teilenummer
                                             (Schlüsselfeld)
002        MENGE       N     004             Erhaltene Menge
003        KOSTEN      N     009      002    Einkaufspreis
004        DATUM       C     008             Eingangsdatum
005        LIEFERANT   C     025             Lieferant
006        UEBERTR     L     001             Zur Hauptdatei
                                             geschickt?
```

Wieder bauen Sie die Datenbank mit dem Befehl CREATE nach dem hier gezeigten Muster auf und indizieren über das Feld TEIL:NR in die Index-Datei NEWSTOCK.NDX mit den bekannten Befehlen:

```
USE newstock
INDEX ON teil:nr TO newstock
```

Die übergreifende Datenbank für unser Inventarsystem besteht jetzt aus sechs Dateien: drei DBF-Dateien und drei NDX-Dateien.

Datendateien	*Index-Dateien*
MASTER.DBF	MASTER.NDX
SALES.DBF	SALES.NDX
NEWSTOCK.DBF	NEWSTOCK.NDX

Nachdem nun die Datenbanken erstellt sind, müssen wir uns um die Gestaltung der Software-Struktur kümmern. Offensichtlich haben wir es mit einem etwas größeren System zu tun, und etwas Vorplanung ist angeraten.

Gestaltung der Inventarprogramme

Die Struktur des Programmsystems erlaubt es, daß verschiedene Personen die unterschiedlichen Arbeiten am System erledigen. Das heißt, der Geschäftsführer wird sich um die Hauptdatei kümmern, Bestellungen aufgeben, den Lagerstatus abfragen usw. Verkäufer werden einzelne Verkäufe eingeben, und ein Lagerchef wird sich um die hereinkommenden Waren kümmern.

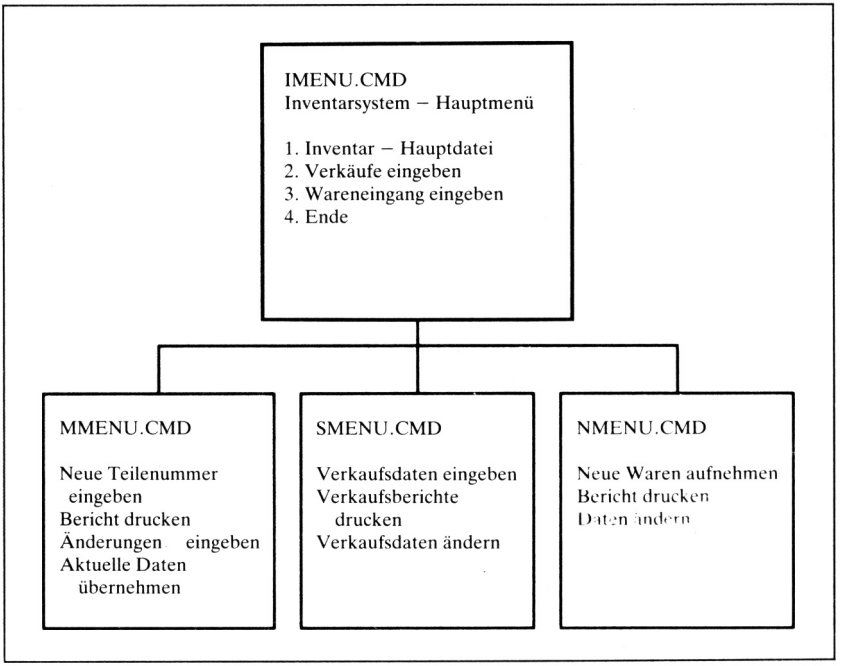


Abb. 9.2: Die Software-Struktur des Inventarsystems

In einem gewissen Sinne entwickeln wir drei selbstständige, aber verbundene Systeme. Die übergreifende Struktur dieser Systeme sehen Sie in Abb. 9.2.

Das Programm IMENU.CMD ist das oberste Inventarmenü. Je nach Wahl des Benutzers verzweigt es entweder zu

1. MMENU.CMD, dem Hauptmenü zur Bearbeitung der Hauptdatei
2. SMENU.CMD, dem Menü für die einzelnen Verkäufe, oder zu
3. NMENU.CMD, dem Menü für den Wareneingang.

Jedes der Untermenüs in Abb. 9.2 arbeitet mit anderen Programmen zusammen, aber wir kümmern uns erst darum, wenn es soweit ist.

Das Inventar-Hauptmenü

Lassen Sie uns bei der Entwicklung des Inventarsystems mit dem übergeordneten Hauptmenü beginnen. Wir werden alle Untermenüprogramme in späteren Kapitel entwickeln.

Das Hauptmenü des Inventarprogramms fragt den Benutzer nach zwei Informationen. Zuerst erfragt es das Datum (wenn notwendig) und benutzt dieses bei verschiedenen Prozeduren weiter. Dann fragt es den Benutzer, mit welcher Datei er arbeiten möchte: 1. der Hauptdatei, 2. der Verkaufsdatei oder 3. der Wareneingangsdatei. Dann verzweigt es zu dem ausgewählten Programm. Die Abb. 9.3 zeigt Ihnen den Pseudocode für das Hauptmenü des Inventarsystems.

```
Parameter setzen
Bildschirm löschen
Wenn kein Datum vorhanden
    Benutzer nach dem heutigen Datum fragen
    Das heutige Datum zum dBASE-Systemdatum machen

Schleife für Hauptmenü beginnen
    Bildschirm löschen
    Hauptmenü anzeigen:
        1. Inventar-Hauptdatei
        2. Verkäufe eingeben
        3. Wareneingang
        4. ENDE
    Auf Auswahl warten
    Zu dem entsprechenden Programm verzweigen

Weiter mit Schleife, bis Benutzer aufhören möchte
dBASE mit QUIT verlassen
```

Abb. 9.3: Pseudocode für das Programm IMENU.CMD

Wir können jetzt das Programm erstellen. Unsere erste Aufgabe ist wie immer, ein paar erklärende Informationen in den Kopf des Programms zu schreiben und die notwendigen dBASE-Parameter zu installieren, wie Sie es hier sehen:

```
***************************** IMENU.CMD
***************** Hauptmenü für das Inventarsystem
SET TALK OFF
SET DEFA TO B
```

Als nächstes wird das Programm den Bildschirm löschen, nachsehen, ob es nach dem Datum fragen soll und nötigenfalls danach fragen. Fügen Sie diese Zeilen an das Programm an:

```
********* Systemdatum aufnehmen (wenn notwendig)
ERASE
STORE DATE() TO t:datum
IF VAL(t:datum)<=1
   §  5,5 SAY "Geben Sie das heutige Datum ein";
              GET t:datum PICTURE "99/99/99"
   READ
   SET DATE TO &t:datum
ENDIF (t:datum=0)
```

Wie Sie sehen, speichert das Programm zuerst das dBASE-Datum <DATE()> in die Variable T:DATUM. Dann überprüft es, ob der Wert (VAL) dieses Datums kleiner oder gleich 1 ist (weil einige Computer automatisch ein Datum mit 01/01 eingeben, wenn der Benutzer nichts eingibt, überprüft man am besten, ob der Monat kleiner oder größer 1 ist). Das Programm fragt mit einer Vorgabemaske (//) nach dem Tagesdatum und setzt dann das dBASE-Datum auf diesen Wert.

Jetzt müssen wir eine Schleife aufbauen, die den Bildschirm löscht, das Menü anzeigt und auf eine Eingabe des Benutzers wartet. Wir haben so etwas schon in anderen Programmen geschrieben. Für das Inventarsystem lautet der Programmcode:

```
****************** Schleife für Hauptmenü beginnen
STORE " " TO iwahl
DO WHILE iwahl # "4"
ERASE
TEXT
                    Inventarsystem - Hauptmenü
               1.   Inventar - Hauptdatei
               2.   Verkäufe eingeben
               3.   Wareneingang eingeben
               4.   ENDE

ENDTEXT

********************** Hole Antwort vom Benutzer
§ 10,18 SAY "Bitte wählen Sie ";
         GET iwahl PICTURE "9"
READ
```

Dann verzweigt das Programm zu dem entsprechenden Unterprogramm, getreu der Wahl des Benutzers. Fügen Sie die DO CASE-Anweisungen an Ihr Programm an:

```
DO CASE

   CASE iwahl = "1"
        DO mmenu

   CASE iwahl = "2"
        DO smenu

   CASE iwahl ="3"
        DO nmenu

ENDCASE
```

Nun müssen wir noch die DO WHILE-Schleife schließen und dBASE beenden, wenn der Benutzer ENDE gewählt hat. Lassen Sie uns noch diese beiden Zeilen anhängen:

```
ENDDO (while iwahl #4)
QUIT
```

Das gesamte Programm ist in Abb. 9.4 zu sehen.

```
***************************** IMENU.CMD
***************** Hauptmenü für das Inventarsystem
SET TALK OFF
SET DEFA TO B

*************** Systemdatum aufnehmen (wenn notwendig)
ERASE
STORE DATE() TO t:datum
IF VAL(t:datum)<=1
   §  5,5 SAY "Geben Sie das heutige Datum ein";
              GET t:datum PICTURE "99/99/99"
   READ
   SET DATE TO &t:datum
ENDIF (t:datum=0)

***************************** Schleife für Hauptmenü beginnen
STORE " " TO iwahl
```

Abb. 9.4: Das Programm IMENU.CMD

```
DO WHILE iwahl # "4"
   ERASE
   TEXT
                    Inventarsystem - Hauptmenü

                1.  Inventar - Hauptdatei

                2.  Verkäufe eingeben

                3.  Wareneingang eingeben


                4.  ENDE

   ENDTEXT

   ********************** Hole Antwort vom Benutzer
   § 12,20 SAY "Bitte wählen Sie ";
          GET iwahl PICTURE "9"
   READ


   DO CASE

      CASE iwahl = "1"
           DO mmenu

      CASE iwahl = "2"
           DO smenu

      CASE iwahl ="3"
           DO nmenu

   ENDCASE

ENDDO (while iwahl #4)

QUIT
```

Abb. 9.4: Das Programm IMENU.CMD (Forts.)

Zusammenfassung

In diesem Kapitel haben wir das Inventarsystem entworfen, das wir in den folgenden Kapiteln entwickeln werden. Drei Datenbanken haben wir erstellt, um Daten über den gegenwärtigen Lagerstatus (MASTER.DBF), die einzelnen Verkäufe (SALES.DBF) und den Wareneingang (NEW-

STOCK.DBF) zu erhalten und zu verarbeiten. Das Feld TEIL:NR wird als Schlüsselfeld benutzt, um den aktuellen Lagerbestand aufgrund der Änderungen in den Dateien SALES und NEWSTOCK zu erhalten.

Wir haben auch das übergreifende Hauptmenü des gesamten Inventarsystems IMENU.CMD entwickelt. Im nächsten Kapitel werden wir die Programme entwerfen und entwickeln, mit denen man Informationen der Hauptdatei verarbeitet oder von ihr enthält.

Die Haupt-Inventar-Datei

10

Wie schon im vorherigen Kapitel beschrieben, enthält die Hauptdatei des Inventarsystems den gegenwärtigen Status des Lagers. Die Datenbank heißt MASTER.DBF und besitzt diese Struktur:

```
STRUKTUR DER DATEI: B:MASTER.DBF

FELDNR     NAME        TYP      WEITE     DEZ
001        TEIL:NR      C       005
002        TITEL        C       020
003        MENGE        N       004
004        KOSTEN       N       009       002
005        NACHBEST     N       004
006        BESTELL      N       004
007        FUNDORT      C       005
008        LIEFERANT    C       025
009        LIEF:ADR     C       025
010        LIEF:PO      C       025
011        DATUM        C       008
012        BEST:DAT     C       008
013        NEUBEST      N       004
```

Weiterhin haben wir diese Datei bereits nach der TEIL:NR indiziert und damit die Datei MASTER.NDX erstellt. In diesem Kapitel werden wir Programme schreiben, um Daten in der Hauptdatei zu verarbeiten.

Die Gestaltung der Hauptdatei

Erinnern Sie sich, daß der Benutzer in das Inventarsystem durch das IMENU-Programm einsteigt, das folgende Möglichkeiten bietet:

```
Inventarsystem - Hauptmenü

1.  Inventar - Hauptdatei

2.  Verkäufe eingeben

3.  Wareneingang eingeben

4.  ENDE
```

Wenn der Benutzer die Option 1 wählt, verzweigt IMENU.CMD zu dem Programm MMENU.CMD, das das Untermenü zum Bearbeiten der Hauptdatei ist. Das MMENU wiederum bietet dem Benutzer diese Auswahlmöglichkeiten an:

```
Inventarsystem - Hauptdatei

1.  Neue Teilenummer eingeben

2.  Report drucken

3.  Änderungen eingeben

4.  Aktuelle Daten übernehmen

5.  Zurück zum Hauptmenü
```

Die Struktur der Programme, die für das Bearbeiten der Hauptdatei notwendig sind, sehen Sie in Abb. 10.1.

Die Hauptdatei

Lassen Sie uns mit der Entwicklung des Programms MMENU anfangen. Dies ist ein Menüprogramm, das wir in seinen grundlegenden Strukturen schon früher entwickelt haben, so daß wir den Pseudocode und den erklärenden Teil der Beschreibung weglassen können. Das Programm sehen Sie in der Abb. 10.2. Wenn Sie sich die Umrisse des Programms betrachten, werden Sie feststellen, wie ähnlich es den früheren Programmen ist.

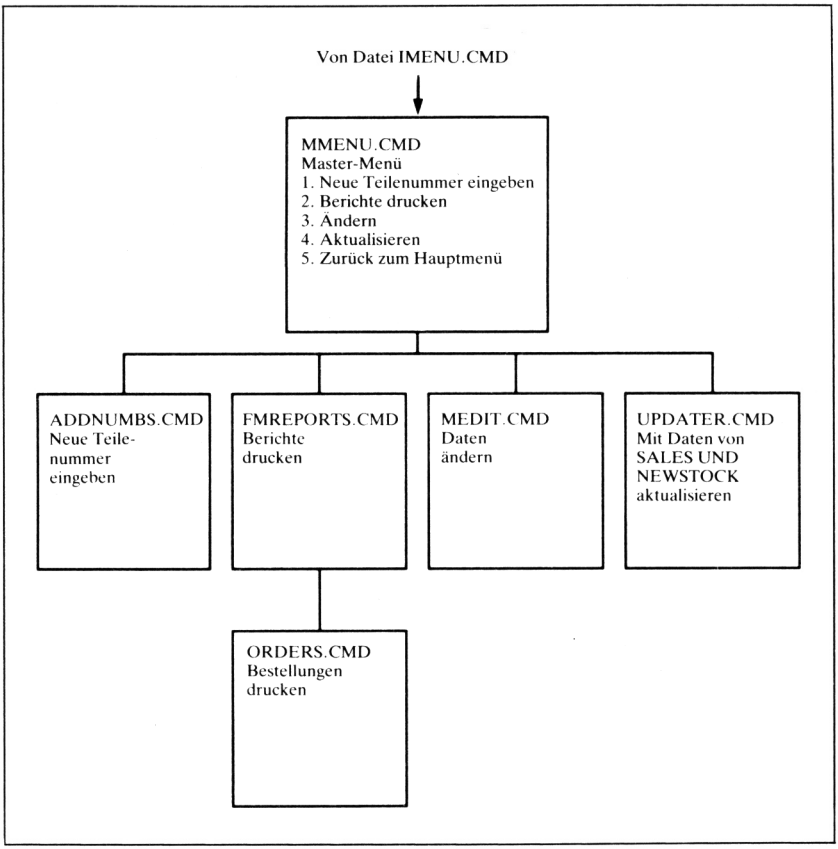


Abb. 10.1: Die Software-Struktur zum Bearbeiten der Hauptdatei

Neue Teilenummer eingeben

Die erste Option des Hauptdateimenüs ermöglicht die Eingabe neuer Teilenummern. Diese Aufgabe wird von dem Programm ADDNUMBS.CMD erledigt. Wenn der Benutzer diese Wahl trifft, fordert das Programm auf dem Bildschirm:

```
Geben Sie die Teilenummer ein :  :
```

Denken Sie daran, daß jede Teilenummer nur einmal vorkommen darf, weil dies ein Schlüsselfeld ist. Das ADDNUMBS-Programm wird dann

```
****************************************** MMENU.CMD
*********************** Inventarsystem - Hauptdateimenü
STORE " " TO mwahl
DO WHILE mwahl # "5"
   ERASE
   TEXT
                  Inventarsystem - Hauptdatei

              1.  Neue Teilenummer eingeben

              2.  Report drucken

              3.  Änderungen eingeben

              4.  Hauptdatei aktualisieren


              5.  Zurück zum Hauptmenü

    ENDTEXT

    § 18,10 SAY "Bitte wählen Sie (1-5)";
            GET mwahl PICTURE "9"
    READ

    DO CASE

       CASE mwahl = "1"
          DO addnumbs

       CASE mwahl = "2"
          DO mreports

       CASE mwahl = "3"
          DO medit

       CASE mwahl = "4"
          DO updater

    ENDCASE
ENDDO (while mwahl # 5)

RELEASE mwahl
RETURN
```

Abb. 10.2: Das Programm MMENU.CMD

die Teilenummer daraufhin überprüfen, ob sie wirklich neu ist. Wenn nicht, erscheint dies:

```
X-999 existiert bereits!
```

```
Cursor-Kontrolle : ^E hoch: ^X runter: ^D rechts: ^S links : ^G loeschen

========================================

Teilenummer    :                 Datum :

Titel          :

Menge im Lager :                 EK-Preis :

Bestellmenge   :                 Nachbest. bei :

Lagerort       :

Lieferant      :

Adresse        :

PLZ, Ort       :
```

Abb. 10.3: Die Eingabemaske der Hauptdatei MASTER

```
Cursor-Kontrolle : ^E hoch: ^X runter: ^D rechts: ^S links : ^G loeschen

========================================
========================================
Teilenummer :#teil:nr Datum :#t:datum

Titel :#titel

Menge im Lager:#menge EK-Preis :#kosten

Bestellmenge :#bestell Nachbest. bei :#nachbest

Lagerort :#fundort

Lieferant :#lieferant

Adresse :#lief:adr

PLZ, Ort :#lief:po
```

Abb. 10.4: Der ZIP-Bildschirm für ISCREEN1.FMT

(wobei x-999 die bereits existierende Nummer ist), und dazu warnt noch ein Piepston. Dann kann der Benutzer eine andere Nummer versuchen. Gibt er eine wirkliche neue Nummer an, kann er die dazugehörenden Daten über die Erfassungsmaske, die Sie in Abbildung 10.3 sehen, in die Hauptdatei schreiben.
Wie schon beschrieben, ist der einfachste Weg zu solchen Masken der Gebrauch von ZIP. Die Maske bekommt den Namen ISCREEN1.FMT.

In der Abb. 10.5 sehen Sie das dBASE-Programm für die Datei ISCREEN1.FMT.

Nachdem nun die Maske fertig ist, können wir uns an das Schreiben des Programms ADDNUMBS.CMD machen. Der dazugehörende Pseudocode steht in der Abb. 10.6.

Jetzt lassen Sie uns aber das Programm schreiben. Der erste Schritt ist, die einführenden Kommentare hinzuschreiben und die Datenbank

```
* ISCREEN1.FMT
§  1, 1 SAY "Cursor Control : 'E hoch: 'X runter: 'D rechts: 'S
          links : 'G loeschen"
§  3, 1 SAY "=================================================="
§  5, 1 SAY "Teilenummer   "
§  5,16 GET teil:nr
§  5,42 SAY "Datum "
§  5,49 GET t:datum
§  7, 1 SAY "Titel         "
§  7,16 GET titel
§  9, 1 SAY "Menge im Lager"
§  9,16 GET menge
§  9,42 SAY "EK-Preis "
§  9,52 GET kosten
§ 11, 1 SAY "Bestellmenge  "
§ 11,16 GET bestell
§ 11,42 SAY "Nachbest. bei "
§ 11,57 GET nachbest
§ 13, 1 SAY "Lagerort      "
§ 13,16 GET fundort
§ 15, 1 SAY "Lieferant     "
§ 15,16 GET lieferant
§ 17, 1 SAY "Adresse       "
§ 17,16 GET lief:adr
§ 19, 1 SAY "PLZ, Ort      "
§ 19,16 GET lief:po
```

Abb. 10.5: Die Formatdatei ISCREEN1.FMT

MASTER.DBF mit der Index-Datei MASTER.NDX als aktuelle Datei anzumelden (USE):

```
****************************** ADDNUMBS.CMD
********************** Neue Teile in die Hauptdatei
********************** eintragen
USE master INDEX master
```

Dann öffnen wir eine Schleife, damit das Programm immer weiter nach Teilenummern fragt (bis der Benutzer aufhören will). Um es ihm einfach zu machen, erlauben wir ihm, das Programm mit einem einfachen RETURN zu verlassen. Um die Schleife überhaupt zum Laufen zu bekommen, speichern wir ein zufällig gewähltes Zeichen in die Variable TEIL:NR, in diesem Fall den Buchstaben X. Dann benutzen wir die üblichen §-SAY- und READ-Befehle, um die Teilenummern zu bekommen, wie Sie hier sehen:

```
************** Schleife für Teilenummer beginnen
STORE "X" TO teil
DO WHILE teil# " "
```

Benutze die Datei MASTER mit dem Index MASTER.NDX

Beginne Schleife für Eingabe neuer Teilenummern
  Bildschirm löschen
  Benutzer nach neuer Teilenummer fragen
  Überprüfen, ob Teilenummer bereits vorhanden

  Wenn keine Nummer eingegeben
  Bildschirm löschen und zurück zum Menü

  Wenn Teilenummer existiert, Benutzer warnen
  und weiteren Versuch ermöglichen

  Wenn Teilenummer nicht vorhanden, restliche
  Daten eingeben lassen

Weiter mit Schleife zum Eingeben der Teilenummern
(bis der Benutzer RETURN eingibt)

Variablen löschen
Zurück zum Hauptdateimenü

Abb. 10.6: Pseudocode für das ADDNUMBS.CMD Programm

```
********** Hole vorgeschlagene Teilenummer
ERASE
STORE " " TO teil
§ 5,5 SAY "Teilenummer eingeben " GET teil
READ
```

Jetzt kommt der Teil, der sicherstellt, daß die eingegebene Teilenummer nicht schon existiert. Jeden eingegebenen Buchstaben des Feldes TEIL:NR wandeln wir in Großbuchstaben um. Da die Datei MASTER über das Feld TEIL:NR indiziert ist, können wir einfach den FIND-Befehl mit einem Makro benutzen:

```
******************** Nachsehen, ob die Teilenummer
******************** bereits existiert
STORE !(teil) TO teil
FIND &teil
```

Jetzt bestimmt die Benutzereingabe, was das Programm als nächstes tun wird. Gibt er keine Teilenummer ein, sollte es die Kontrolle wieder an das Hauptdateimenü übergeben (MMENU.CMD). Die erste Anweisung in der DO CASE-Schleife kümmert sich darum:

```
**************** Entscheide, was jetzt zu tun ist
DO CASE

   ***** Wenn keine Teilenummer eingegeben,
   ***** Bildschirm löschen und zurück zum Menü
   CASE teil = " "
      ERASE
```

Gab der Benutzer eine Nummer ein, die bereits existiert, soll er eine Warnung bekommen und es noch einmal versuchen dürfen. Das zweite CASE sorgt dafür:

```
**************** Wenn Teilenummer existiert, Benut-
**************** zer über das
**************** Problem informieren
  CASE #>0
     § 10,10 SAY teil + "existiert bereits"
     ? CHR(7)
     WAIT
```

Die Anweisung ? CHR(7) läßt die meisten Computer piepsen, wodurch der Bediener eine zusätzliche akustische Warnung erhält.

Hat schließlich der Benutzer eine akzeptable Nummer eingegeben, sollte ihm das Programm ermöglichen, die restlichen Informationen zu diesem Teil über die Maske ISCREEN1 einzugeben. Diese Zeilen sorgen dafür:

```
******** Wenn es die Teilenummer noch nicht gibt,
******** soll der Benutzer alles eingeben
CASE # = 0
   APPEND BLANK
   REPLACE teil:nr WITH teil
   REPLACE datum WITH t:datum
   SET FORMAT TO iscreen1
   READ

ENDCASE
```

Lassen Sie uns die gemachten Schritte hier noch einmal wiederholen. Zuerst überprüft die CASE-Anweisung, ob die Datensatznummer nach dem FIND-Befehl gleich 0 ist (CASE #=0). Das zeigt an, daß die eingegebene neue Teilenummer nicht gefunden wurde und deshalb als neu anerkannt wird. Wenn das zutrifft, fügt das Programm der Datenbank einen neuen Datensatz hinzu, wobei die dort stehenden Felder alle leer sind (APPEND BLANK). Da der Benutzer bereits eine Teilenummer (TEIL) und ein Datum (T:DATUM) eingegeben hat, werden diese beiden Felder automatisch gefüllt, und er braucht sie nicht noch einmal einzugeben. Dann ruft das Programm die Formatdatei ISCREEN1 auf, nimmt die Daten für den neuen Datensatz entgegen und schaltet dann wieder auf den normalen dBASE-Bildschirmmodus um.

Dann macht das Programm wieder eine Schleife, weil es davon ausgeht, daß der Benutzer noch nicht das Ende des Programms befohlen hat. Wünscht er aber aufzuhören, löscht das Programm eine Speichervariable und kehrt zum Programm MMENU zurück, wie Sie hier sehen können:

```
ENDDO (teil # " ")

RELEASE teil
RETURN
```

Das gesamte Programm ist in der Abb. 10.7 zu sehen.

Hauptdatei-Reports

Wenn der Benutzer die Option 2 des Inventar-Hauptdateimenüs anwählt, um Berichte zu drucken, zweigt das Programm (MMENU.CMD) zu dem

```
********************************************** ADDNUMBS.CMD
********************** Neue Teile in die Hauptdatei eintragen
USE master INDEX master

************************ Schleife für Teilenummer beginnen
STORE "X" TO teil
DO WHILE teil# " "

   ************************ Hole vorgeschlagene Teilenummer
   ERASE
   STORE "      " TO teil
   @  5,5 SAY "Teilenummer eingeben " GET teil
   READ

   ******************** Nachsehen, ob die Teilenummer
   ******************** bereits existiert
   STORE !(teil) TO teil
   FIND &teil

   ************************* Entscheide, was jetzt zu tun ist
   DO CASE

   *************** Wenn keine Teilenummer eingegeben,
   *************** Bildschirm löschen und zurück zum Menü
   CASE teil = " "
      ERASE

   ***************** Wenn Teilenummer existiert, Benutzer über das
   ***************** Problem informieren
   CASE #>0
      @ 10,10 SAY teil + "existiert bereits"
      ? CHR(7)
      WAIT

   ************** Wenn es die Teilenummer noch nicht gibt,
   ************** soll der Benutzer alles eingeben
   CASE # = 0
      APPEND BLANK
      REPLACE teil:nr WITH teil
      REPLACE datum WITH t:datum
      SET FORMAT TO iscreen1
      READ
      SET FORMAT TO SCREEN

ENDCASE

ENDDO (teil  #" ")

RELEASE teil
RETURN
```

Abb. 10.7: Das Programm ADDNUMBS.CMD

Programm MREPORTS ab, das dieses Menü von möglichen Berichten zeigt:

```
Inventar-Hauptdatei

Report Optionen

1.  Gesamtinventar
2.  Nachbestellungen
3.  Bestellte Teile
4.  Bestellung schreiben


5.  Zurück zum Menü
```

Beispielausdrucke und die ersten drei Report-Dateien werden im folgenden beschrieben.

Bericht über den aktuellen Lagerbestand

Wenn der Benutzer die Option 1 wählt, wird ein Bericht, wie in Abb. 10.8 beschrieben, gedruckt. Der dBASE-Befehl REPORT wird benutzt, um den Bericht auszudrucken. Sie erstellen die Report-Datei, indem Sie diese Zeilen eingeben:

```
USE master
REPORT FORM allmast
```

SEITE NR. 00001
30/10/84

Aktueller Lagerbestand

Teil #	Beschreibung	auf Lager	EK	Nachbest. bei	Fund-ort	Aktualisiert am
A-111	Schneeschuhe	1	34.00	10	S-111	1/01/80
A-113	Tennisschläger	6	44.00	30	T-11	01/01/80
B-232	Hundekuchen	30	2.50	50	DB-99	01/01/80
B-232	Hundeknochen	30	2.50	50	DB-99	01/01/80
B-232	Hundedecken	30	2.50	50	DB-99	01/01/80
B-232	Halsband	30	2.50	50	DB-99	01/01/80
Z-999	Bärenfell	10	100.00	5	B-502	02/05/84

Abb. 10.8: Inventar-Gesamtbericht für das Inventarsystem

Beantworten Sie die auftauchenden Fragen, wie hier vorgegeben:

```
ENTER OPTIONS, M=LEFT MARGIN,L=LINES/PAGE,
W=PAGE WIDTH M=1,W=70
PAGE HEADING? (Y/N) Y
ENTER PAGE HEADING: Lagerbestand
DOUBLE SPACE REPORT? (Y/N) N
ARE TOTALS REQUIRED? (Y/N) N
COL     WIDTH,CONTENTS
001     05,TEIL:NR
ENTER HEADING: Teil #
002     20,TITEL
ENTER HEADING: Beschreibung
003     5,MENGE
ENTER HEADING: auf;Lager
004     9,KOSTEN
ENTER HEADING: EK
005     7,NACHBEST
ENTER HEADING: Nachbst.;bei
006     7,FUNDORT
ENTER HEADING: Fundort
007     8,DATUM
ENTER HEADING: Aktual.;am
008
```

Bericht über Nachbestellungen

Wählt der Benutzer die zweite Option, erhält er den Bericht über die Nachbestellungen, wie es in Abb. 10.9 zu sehen ist. Benutzen Sie den Befehl REPORT FORM reorder direkt hinter dem dBASE-Prompt, um die Reportdatei zu erstellen, und geben Sie die hier aufgeführten Antworten ein:

```
ENTER OPTIONS, M=LEFT MARGIN,L=LINES/PAGE,
W=PAGE WIDTH M=1,W=70
PAGE HEADING? (Y/N) Y
ENTER PAGE HEADING: Nachbestellungen
DOUBLE SPACE REPORT? (Y/N) N
ARE TOTALS REQUIRED? (Y/N) N
COL     WIDTH,CONTENTS
001     5,TEIL:NR
ENTER HEADING: Teil;Nr
002     20,TITEL
ENTER HEADING: Beschreibung
```

SEITE NR. 00001
30/10/84

Nachbestellungen

Teil Nr.	Beschreibung	auf Lager	best.	Min.	Lieferant
A-111	Schneeschuhe	1	10	30	Eis & Schuh GmbH
A-113	Tennisschläger	6	10	30	Zappos PowerPlay
Z-999	Bärenfell	10	10	30	Bearskin Canada Co.

Abb. 10.9: Inventarbericht Nachbestellungen

```
003       5,MENGE
ENTER HEADING: auf;Lager
004       5,BESTELL
ENTER HEADING: best.
005       5,NACHBEST
ENTER HEADING: Min
006       20,LIEFERANT
ENTER HEADING: Lieferant
007
```

Bericht der bestellten Teile

Mit der Option 3 wählt der Benutzer einen Bericht, der ihm zeigt, welche Teile bereits bestellt wurden. Dieser Bericht kommt so aus dem Drucker, wie in Abb. 10.10 dargestellt ist.

Sie benutzen den Befehl REPORT FORM onorder, um diese Berichtdatei zu erstellen. Geben Sie die Antworten wie hier angegeben ein:

```
ENTER OPTIONS, M=LEFT MARGIN,L=LINES/PAGE,
W=PAGE WIDTH M=1,W=80
PAGE HEADING? (Y/N) Y
ENTER PAGE HEADING: Bestellte Teile
DOUBLE SPACE REPORT? (Y/N) N
ARE TOTALS REQUIRED? (Y/N) Y
SUBTOTALS IN REPORT? (Y/N) N
COL       WIDTH,CONTENTS
001       8,BEST:DAT
ENTER HEADING: Bestell;Datum
```

SEITE NR. 00001
30/10/84

Bestellte Teile

Best. Datum	Teil #	Beschreibung	Bestellt	EK	Summe	Lieferant
01/10/84	A-111	Schneeschuhe	10	34.00	340.00	Eis & Schuh Gmb
01/10/84	A-113	Tennisschläger	10	44.00	440.00	Zappos PowerPLA
15/10/84	B-232	Hundekuchen	10	2.50	25.00	Tierfreund KG
15/10/84	B-232	Hundeknochen	10	2.50	25.00	Tierfreund KG
15/10/84	B-232	Hundedecken	10	2.50	25.00	Tierfreund KG
15/10/84	B-232	Halsband	10	2.50	25.00	Tierfreund KG
01/10/84	Z-999	Bärenfell	10	100.00	1000.00	BearskinCanada
** TOTAL **					1880.00	

Abb. 10.10: Bericht über bestellte Teile

```
002      5,TEIL:NR
ENTER HEADING: Teile;Nr
003      20,TITEL
ENTER HEADING: Bezeichnung
004      5,BESTELL
ENTER HEADING: Best.;Menge
ARE TOTALS REQUIRED? (Y/N) N
005      9,KOSTEN
ENTER HEADING: EK
ARE TOTALS REQUIRED? (Y/N) N
006      9,KOSTEN*BESTELL
ENTER HEADING: Gesamtkosten
ARE TOTALS REQUIRED? (Y/N) Y
007      15,$(LIEFERANT,1,15)
ENTER HEADING: Lieferant
008
```

Teile bestellen

Mit der vierten Option hat der Benutzer die Möglichkeit, Bestellungen zu schreiben und diese gleich als Bestellzettel auszudrucken. Sobald er diese Option wählt, erscheinen auf dem Bildschirm die wichtigsten Informationen zu jedem Teil des Lagers, dessen vorhandene Stückzahl unter der Nachbestellgrenze liegt. Jedes Teil wird einzeln angezeigt, damit der Benutzer direkt die Bestellung fertigmachen kann. Hier ist ein Beispiel:

```
Teilenummer                    : A-111
Bezeichnung                    : Schneeschuhe
Vorhanden                      : 10
Bestellt                       : 10
Nachbest. bei                  : 30
Einkaufspreis                  : 12.11

Wieviel bestellen? :_    :
```

Der Benutzer sieht auf einen Blick, wieviel Teile noch auf Lager liegen, bestellt sind und wie hoch die Nachbestellmenge ist. Um ein Teil zu bestellen, gibt er einfach die gewünschte Menge ein. Danach erscheint das nächste Teil, das unterhalb der Nachbestellgrenze liegt, auf dem Bildschirm. Das geht so weiter, bis alle Teile angezeigt worden sind. Nachdem alle Teile angezeigt wurden, druckt das Programm eine Bestellung. Sie sehen ein Beispiel in der Abb. 10.11.

Die Report-Optionen auf den Bildschirm bringen

Das Programm MREPORTS.CMD dient dazu, die Report-Optionen dem Benutzer zu präsentieren und dann den entsprechenden Report zu

Tierfreund KG
Gänsepütz 21

5000 Köln 90

Hiermit bestellen wir folgende Teile...

10 Hundekuchen	2.50	25.00
10 Hundeknochen	2.50	25.00
10 Hundedecken	2.50	25.00

Gesamtauftragssumme: 75.00

Bitte liefern an: Meine Firma GmbH
Hauptstr. 150

9764 Irgendwo

Abb. 10.11: Vom Inventarsystem erstellte Bestellung

drucken. Obwohl MREPORTS eigentlich ein ganz normales Menüprogramm ist, hat es einige Feinheiten, die wir hier diskutieren sollten. Es beginnt auch wieder mit den üblichen Kommentaren, wählt die notwendige Datenbank aus und beginnt mit einer Schleife, die das Menü auf den Bildschirm bringt. Sie sehen die Programmzeilen hier:

```
****************************** MREPORTS.CMD
********** Präsentiere die Report-Optionen der
********** Hauptdatei

USE master INDEX master
STORE " " TO repwahl
DO WHILE repwahl # "5"
   ERASE
   TEXT


              Inventar-Hauptdatei

                Report Optionen

            1.  Gesamtinventar

            2.  Nachbestellungen

            3.  Bestellte Teile

            4.  Bestellung schreiben


            5.  Zurück zum Menü

ENDTEXT
§12,10 SAY "Bitte wählen Sie (1-5)";
       GET repwahl PICTURE "9"
READ
```

Wir fragen den Benutzer zuerst, ob er einen Ausdruck des Reports haben möchte (außer wenn er Bestellungen eingibt, dann können wir davon ausgehen, daß der Drucker benutzt wird). Die nächsten paar Zeilen fragen nach dem Drucker:

```
********** Wenn nicht Bestellungen gewählt, nach
********** Drucker fragen
ERASE
```

```
STORE " " TO jn,drucker
  IF repwahl<"4"
§ 4,4 SAY "Report ausdrucken?";
      GET jn PICTURE "!"
  READ
  ERASE
```

Wünscht der Bediener den Ausdruck des Berichts, speichert das Programm den Befehl TO PRINT in der Variablen DRUCKER. Später wird die Variable als Makro mit dem Befehl REPORT FORM zusammen benutzt werden.

```
************************** Drucker einschalten
   IF jn="J"
     STORE "TO PRINT" TO drucker
  ENDIF (jn=j)
  ENDIF (repwahl <4)
```

Danach druckt das Programm den gewünschten Report. Werden Berichte über das Gesamtinventar, Nachbestellungen oder bestellte Teile gedruckt, wählt das Programm einfach die entsprechenden Report-Dateien. Der &DRUCKER-Makro entscheidet, ob ein Report zum Drucker geschickt wird oder nicht. Die CASE-Anweisung kümmert sich um die ersten drei Reports:

```
DO CASE

   CASE repwahl = "1"
        REPORT FORM allmast &drucker

   CASE repwahl = "2"
        REPORT FORM reorder FOR;
        (menge+bestell)<=reorder &drucker

   CASE repwahl = "3"
        REPORT FORM onorder FOR bestell >0;
        &drucker
```

Beachten Sie, daß die erste CASE-Anweisung alle Daten der Hauptdatei druckt. Die zweite druckt nur die Datensätze mit der Report-Datei REORDER.FRM, bei denen die vorhandene MENGE und die BESTELL-Menge zusammen weniger ergeben als die NACHBESTell-menge. Die dritte CASE-Anweisung zeigt nur die Teile an, deren Feld BESTELL einen Betrag gößer als Null enthält.

Die vierte Möglichkeit „Bestellung schreiben" wird von einem eigenen Programm bearbeitet, das ORDERS heißt und gleich besprochen wird. Das vierte CASE ruft das Programm ORDERS auf. Fügen Sie das CASE noch hinzu, und schließen Sie die DO CASE-Schleife:

```
CASE repwahl = "4"
     DO orders
ENDCASE
```

Dann hält das Programm an (falls der Bericht nicht zum Drucker geht und der Benutzer nicht aufhören will), um dem Benutzer Zeit zum Lesen des Berichts zu geben. Fügen Sie dazu noch diese Zeile ein:

```
****************** Wenn Bericht nicht zum Drucker
****************** und nicht Programmende, Pause
IF jn#"J" .AND. repwahl # "5"
   ?
   ?
   ? "Weiter mit beliebiger Taste ..."
   WAIT
ENDIF (jn#j)
```

Dann fügen Sie noch diese Zeilen an, um die Speichervariablen zu löschen, die Schleife zu beenden und gegebenenfalls zum Programm MMENU zurückzukehren:

```
ENDDO (repwahl#"5")

RELEASE repwahl,jn,drucker
RETURN
```

Die Abb. 10.12 zeigt das ganze MREPORTS.CMD-Programm.

```
****************************** MREPORTS.CMD
********************** Präsentiere die Report-Optionen der
********************** Hauptdatei

USE master INDEX master
STORE " " TO repwahl
DO WHILE repwahl # "5"
   ERASE
   TEXT
```

Abb. 10.12: Das Programm MREPORTS.CMD

```
            Inventar-Hauptdatei

              Report Optionen

          1.  Gesamtinventar

          2.  Nachbestellungen

          3.  Bestellte Teile

          4.  Bestellung schreiben


          5.  Zurück zum Menü

 ENDTEXT
 § 18,10 SAY "Bitte wählen Sie (1-5)";
         GET repwahl PICTURE "9"
 READ

 ****** Wenn nicht Bestellungen gewählt, nach Drucker
 ****** ERASE
 STORE " " TO jn,drucker
 IF repwahl<"4"
    §  4,4 SAY "Report ausdrucken?";
           GET jn PICTURE "!"
    READ
    ERASE

    ********************************** Drucker einschalten
    IF jn="J"
       STORE "TO PRINT" TO drucker
    ENDIF (jn=j)
 ENDIF (repwahl <4)

DO CASE

   CASE repwahl = "1"
        REPORT FORM allmast &drucker

   CASE repwahl = "2"
        REPORT FORM reorder FOR;
        (menge+bestell)<=nachbest &drucker

   CASE repwahl = "3"
        REPORT FORM onorder FOR bestell >0;
        &drucker
```

Abb. 10.12: Das Programm MREPORTS.CMD (Forts.)

```
      CASE repwahl = "4"
           DO orders
   ENDCASE

   ****************** Wenn Bericht nicht zum Drucker
   ****************** und nicht Programmende, Pause
   IF jn#"J" .AND. repwahl # "5"
      ?
      ?
      ? "Weiter mit beliebiger Taste ..."
      WAIT
   ENDIF (jn#j)
   ENDDO (repwahl#"5")

RELEASE repwahl,jn,drucker
RETURN
```

Abb. 10.12: Das Programm MREPORTS.CMD (Forts.)

Damit der Benutzer seine Bestellungen aufgeben und ausdrucken kann, verzweigt das Programm MREPORTS.CMD zu dem Programm ORDERS.CMD. Dieses muß verschiedene Aufgabe erfüllen. Zuerst arbeitet es sich durch jeden Datensatz der Hauptdatei und stellt fest, welche Teile nachbestellt werden müssen, indem es die vorhandene Menge zu der bestellten Menge addiert und mit der Mindestmenge vergleicht. Jedesmal wenn es ein Teil findet, das nachbestellt werden muß, soll es den Benutzer fragen, wieviel Stück von diesem Teil bestellt werden sollen. Nachdem alle Bestellungen gemacht sind, druckt das Programm die Bestelliste aus und aktualisiert das Feld BESTELL in der Hauptdatei. Der Pseudocode für das Programm ORDERS steht in der Abb. 10.13.

Wir können das ORDERS-Programm wie folgt schreiben. Zuerst schreiben wir die Eingangszeilen:

```
************************** ORDERS.CMD
****** Bestellschein für Nachbestellung schreiben
ERASE
USE master INDEX master
```

Als nächstes muß das Programm die Datenbank daraufhin durchsuchen, bei welchem Teil die Summe von MENGE und BESTELL geringer ist als der Wert im Feld NACHBEST. Für jedes dieser Teile muß es dann den Benutzer fragen, wieviel er nachbestellen möchte. Schreiben Sie die untenstehende DO WHILE-Schleife zu Ihrem Programm dazu, damit es die Bestellungen bearbeiten kann.

```
****** Durchsuche die Hauptdatei nach Teilen, die
****** nachbestellt werden müssen, und zeige sie an.
****** Dann frage, wie viele bestellt werden sollen
GO TOP
DO WHILE .NOT. EOF
   IF (menge+bestell)<=nachbest
      ERASE
      §  5,5 SAY "Teilenummer" + teil:nr
      §  5,30 SAY titel
      §  6,5 SAY "Vorhanden" + STR(menge,4)
      §  7,5 SAY "Bestellt" + STR(bestell,4)
      §  8,5 SAY "Nachbest. bei" + STR(nachbest,3)
      §  9,5 SAY "Einkaufspreis" + STR(kosten,9,2)
      § 12,5 SAY "Wieviel bestellen? ";
             GET neubest PICTURE "9999"
      REPLACE best:dat WITH T:datum
      READ
   ENDIF (menge+bestell <neubest)
   SKIP
ENDDO (while not eof)
```

Beachten Sie, daß die Bestellmenge in dem Feld NEUBEST gespeichert und das Bestelldatum (T:DATUM) automatisch auf das Tagesdatum gesetzt wird. Sind dann alle Teile eingegeben, kopiert das Programm die zu bestellenden Teile in eine gesonderte Datenbank mit dem Namen TEMP:

```
********** Erstelle eine Zwischendatei mit den
********** Bestellteilen
ERASE
? "Die Datei wird bearbeitet .... bitte warten"
? "(Sie können schon mal den Drucker einschalten)"
COPY TO TEMP FOR neubest > 0
```

Da jetzt die Bestellteile in einer eigenen Datenbank stehen, müssen wir die Hauptdatei etwas verändern. Zuerst muß das Feld BESTELL die neue Anzahl der bestellten Teile mitgeteilt bekommen (die bereits gespeicherte Bestellmenge plus die gerade bestellte Menge). Dann kann das Feld NEUBEST auf Null gesetzt werden. Die jetzt folgenden Zeilen besorgen diese Arbeit:

```
******* Die neuen Bestellungen zum Feld BESTELL,
******* dann NEUBEST wieder auf Null
```

```
REPLACE NOUPDATE ALL bestell WITH;
        bestell+neubest
REPLACE NOUPDATE ALL neubest WITH 0
```

Die hier gezeigte dBASE-Option NOUPDATE hält dBASE davon ab, den Index neu zu sortieren, da weder BESTELL noch NEUBEST mit der Indexdatei etwas zu tun haben.

Um die Bestellung zu drucken, muß das Programm die Datei TEMP benutzen und nach Lieferanten sortieren. Wie das geht, sehen Sie hier:

```
**************** Datei TEMP benutzen, nach Lieferan-
**************** ten sortieren
USE temp
INDEX ON !(lieferant) TO temp
```

Ist die TEMP-Datei bereit, stellen die folgenden Zeilen sicher, daß der Benutzer zum Einschalten des Druckers aufgefordert wird:

```
**************** Datei fertig, Benutzer informieren
ERASE
? CHR(7)
? "Drucker einschalten und beliebige Taste drücken"
WAIT
```

Um nun Bestellungen für jeden einzelnen Lieferanten zu drucken, sollte das Programm durch die ganze TEMP-Datei suchen. Für jeden Lieferanten wird es dann eine Bestellung drucken.

Das erfordert zwei verschachtelte Schleifen. Die erste geht Datensatz für Datensatz durch die Zwischendatei:

```
********* Durchlaufe die Zwischendatei und drucke
********* Bestellungen für jeden Lieferanten
GO TOP
SET PRINT ON
DO WHILE .NOT.EOF
```

Innerhalb dieser Schleife speichert das Programm den Namen des gerade bearbeiteten Lieferanten in eine Variable mit dem Namen LSCHLEIFE. Für die große Abrechnung am Schluß speichert es eine Null in die Variable GESAMT. Dann druckt es den Namen und die Adresse des Lieferanten in den Kopf der Bestellung. Die folgenden Zeilen dienen der Erstellung des Bestellungskopfes:

```
****************************** Drucke 1 Bestellung
STORE lieferant TO lschleife
```

```
Bildschirm löschen
Benutze die Datei MASTER mit Index MASTER

Beginne Schleife durch die Datei MASTER
    Wenn vorhandene Menge + bestellte Menge weniger ist
        als die Nachbestellmenge
        Bildschirm löschen
        Zeige die Daten des zu bestellenden Teils
        Frage Benutzer, wieviel bestellen
        Aktualisiere NEUBEST- und BEST:DAT-Feld
    Weiter zum nächsten Datensatz
Weiter mit der Schleife durch die Datei

Wenn alle Bestellungen fertig
Bildschirm löschen
Erstelle Zwischendatei für Bestellteile
Aktualisiere BESTELL-Feld in der Hauptdatei
Setze NEUBEST-Feld in der Hauptdatei wieder auf 0

Benutze Zwischendatei (da stehen die Bestellungen drin)
Indiziere nach Lieferanten

Schalte Drucker ein
Beginne Schleife durch die Zwischendatei
    Für jeden Lieferanten in der Zwischendatei
        Name und Adresse drucken
    Für jedes Teil von diesem Lieferanten
        Bestellmenge, Bezeichnung, und Kosten drucken

    Wenn mit diesem Lieferanten fertig
    Gesamtkosten der Bestellung drucken

    Zieladresse drucken
    Neue Seite beginnen

Weiter zum nächsten Lieferanten
Weiter mit Schleife durch die Zwischendatei

Wenn alles erledigt, Variablen löschen, Drucker ausschalten
und zurück zu MREPORTS
```

Abb. 10.13: Pseudocode für das ORDERS-Programm

```
STORE 0 TO gesamt
? lieferant
? liefer:adr
? liefer:po
?
? "Hiermit bestellen wir folgende Teile ..."
?
```

Solange nun der Name des Lieferanten nicht wechselt (LIEFERANT= LSCHLEIFE), druckt das Programm die Bestellmenge (NEUBEST), die Bezeichnung (TITEL) und die Gesamtsumme (NEUBEST*KOSTEN). Zusätzlich wird die Gesamtsumme aller Bestellungen laufend mitgeführt. Diese Schleife kann das:

```
DO WHILE lieferant = lschleife .AND. .NOT. EOF
   ? neubest,titel,kosten,neubest*kosten
   STORE neubest*kosten + gesamt TO gesamt
   SKIP
ENDDO (while lieferant = lschleife)
```

Sobald das Programm mit diesem Lieferanten fertig ist, druckt es die Gesamtkosten der Bestellung und die Adresse des Auftraggebers aus. Dann geht es eine Seite weiter, wie Sie in dem folgenden Programmtext sehen:

```
?
? "Gesamtauftragssumme:                          ",gesamt
?
? "Bitte liefern an:  Meine Firma GmbH"
? "                   Hauptstr. 150"
?
? "                   9764 Irgendwo"
EJECT
```

Die Hauptschleife durch die gesamte Datei muß weitergehen, bis die Bestellungen für alle Lieferanten geschrieben sind. Deshalb schließen wir jetzt die Schleife mit

```
ENDDO (while not eof)
```

Sind alle Bestellungen gedruckt, löscht das Programm die Variablen und kehrt zum MREPORTS.CMD-Programm zurück. Außerdem wird mit der Speicherung von J in die Variable JN, am Ende der Datei MREPORTS, die Bildschirmpause verhindert, weil der Bericht ausgedruckt

wurde. Dann werden wieder die Dateien MASTER.DBF und MASTER.NDX in Gebrauch genommen und die Datei TEMP gelöscht:

```
***************** Fertig. Lösche Variable, schalte
***************** Drucker aus, speichere J in JN und
***************** kehre zum Menü zurück
SET PRINT OFF
RELEASE lschleife,gesamt
STORE "J" TO jn
USE master INDEX master
IF FILE("temp.dbf")
   DELETE FILE temp
ENDIF (datei temp existiert)

RETURN
```

Das ORDERS-Programm ist ein wenig abstrakt. Sie sollten es sorgfältig studieren, um es wirklich zu verstehen. Ich verspreche Ihnen, daß Ihre Zeit gut angelegt ist, da es eine Menge Programmiertechniken enthält, die Sie immer wieder bei vielen Programmierproblemen einsetzen können. Die Abb. 10.14 zeigt Ihnen das komplette Programm ORDERS.CMD.

```
*************************** ORDERS.CMD
********** Bestellschein für Nachbestellung schreiben
ERASE
USE master INDEX master

****** Durchsuche die Hauptdatei nach Teilen, die nachbestellt
****** werden müssen, und zeige sie an. Dann frage, wie viele
****** bestellt werden sollen
GO TOP
DO WHILE .NOT. EOF
   IF (menge+bestell)<=nachbest
      ERASE
      § 5,5 SAY "Teilenummer" + teil:nr
      § 5,30 SAY titel
      § 6,5 SAY "Vorhanden" + STR(menge,4)
      § 7,5 SAY "Bestellt" + STR(bestell,4)
      § 8,5 SAY "Nachbest. bei" + STR(nachbest,3)
      § 9,5 SAY "Einkaufspreis" + STR(kosten,9,2)
      § 12,5 SAY "Wieviel bestellen? ";
             GET neubest PICTURE "9999"
```

Abb. 10.14: Das Programm ORDERS.CMD

```
      REPLACE best:dat WITH T:datum
      READ
   ENDIF (menge+bestell <neubest)
   SKIP
ENDDO (while not eof)
********** Erstelle eine Zwischendatei mit den Bestellteilen
ERASE
? "Die Datei wird bearbeitet ... bitte warten"
? "(Sie können schon mal den Drucker einschalten)"
COPY TO TEMP FOR neubest > 0

******************** Die neuen Bestellungen zum Feld BESTELL,
************** dann NEUBEST wieder auf Null
REPLACE NOUPDATE ALL bestell WITH;
        bestell+neubest
        REPLACE NOUPDATE ALL neubest WITH 0

**************** Datei TEMP benutzen, nach Lieferanten
**************** sortieren
USE temp
INDEX ON !(lieferant) TO temp

**************** Datei fertig, Benutzer informieren
ERASE
? CHR(7)
? "Drucker einschalten und beliebige Taste drücken"
WAIT

****************** Durchlaufe die Zwischendatei und drucke
****************** Bestellungen für jeden Lieferanten
GO TOP
SET PRINT ON
DO WHILE .NOT.EOF

   ****************************** Drucke 1 Bestellung
   STORE lieferant TO lschleife
   STORE 0 TO gesamt
   ? lieferant
   ? lief:adr
   ? lief:po
   ?
   ? "Hiermit bestellen wir folgende Teile ..."
   ?

   DO WHILE lieferant = lschleife .AND. .NOT. EOF
      ? neubest,titel,kosten,neubest*kosten
      STORE neubest*kosten + gesamt TO gesamt
      SKIP
```

Abb. 10.14: Das Programm ORDERS.CMD (Forts.)

```
    ENDDO (while lieferant = lschleife)
    ?
    ? "Gesamtauftragssumme:                              ",gesamt
    ?
    ? "Bitte liefern an:  Meine Firma GmbH"
    ? "                   Hauptstr. 150"
    ?
    ? "                   9764 Irgendwo"
    EJECT

ENDDO (while not eof)

**************************** Fertig. Lösche Variable, schalte
**************************** Drucker aus, speichere J in JN und
**************************** kehre zum Menü zurück
SET PRINT OFF
RELEASE lschleife,gesamt
STORE "J" TO jn
USE master INDEX master
IF FILE("temp.dbf")
   DELETE FILE temp
ENDIF (datei temp existiert)

RETURN
```

Abb. 10.14: Das Programm ORDERS.CMD (Forts.)

Änderungen in der Hauptdatei

Die Option 3 aus dem Hauptmenü ermöglicht es dem Benutzer, die Hauptdatei zu verändern. Das dazu notwendige Programm heißt MEDIT.CMD, und dessen Pseudocode finden Sie in Abb. 10.15.

Da wir die Formatdatei AMASK1.FMT bereits erstellt haben, können wir sie auch gleich zum Ändern von Daten benutzen. Das macht außerdem die Entwicklung des Programms MEDIT recht einfach. Der erste Schritt sind wieder die Kommentare im Kopf des Programms und die Verwendung der Datenbank MASTER mit ihrem Index:

```
********************** MEDIT.CMD
******************** Ändern der Inventar-Hauptdatei
USE master INDEX master
```

Dann öffnen wir eine Schleife, die es dem Benutzer erlaubt, mehrere Änderungen hintereinander durchzuführen oder durch Eingabe von

RETURN statt einer Teilenummer die Schleife zu verlassen:

```
STORE "X" TO teilnr
DO WHILE teilnr#" "
```

Jetzt brauchen wir ein paar Zeilen, die den Benutzer nach der zu ändernden Teilenummer fragen:

```
******************** Die zu ändernde Teilenummer
******************** herausfinden
ERASE
STORE "      " TO teilnr
§ 5,5 SAY "Welche Teilenummer ändern?";
      GET teilnr
READ
```

Die nächste Zeile läßt dBASE die Teilenummer suchen:

```
*************** Versuche die Teilenummer zu finden
STORE !(teilnr) TO teilnr
FIND &teilnr
```

Benutze die Datei MASTER und die Index-Datei MASTER

Beginne mit der Schleife für Änderungen

  Stelle fest, welche Teilenummer ändern

  Versuche die Teilenummer zu finden

  Wenn keine Teilenummer eingegeben, zurück

  Wenn Nummer vorhanden, mit der Maske AMASK1 ändern lassen

  Anderfalls Benutzer warnen und weiteren Versuch zulassen

Weiter mit Ändern, bis Abbruch gewünscht

Wenn Ändern beendet, Variablen löschen und zurück

Abb. 10.15: Pseudocode für das Programm MEDIT.CMD

Jetzt muß das Programm entscheiden, was als nächstes zu tun ist, abhängig von der Eingabe des Benutzers. Gibt er keine Nummer ein, löscht es nur den Bildschirm und überspringt die anderen CASE-Anweisungen. Wurde eine gültige Teilenummer eingegeben, ermöglicht es Änderungen und benutzt dabei die AMASK1-Datei. Hat der Benutzer eine ungültige Nummer eingetippt, bekommt er eine Warnung angezeigt. Die DO CASE-Konstruktion hier kümmert sich um die drei Möglichkeiten:

```
********************* Wenn gefunden, mit AMASK1.FMT
********************* ändern, sonst Benutzer warnen
DO CASE

   *** Wenn keine Nummer eingegeben, einfach zurück
   CASE teilnr=" "
     ERASE

   *** Wenn Nummer gefunden, Ändern ermöglichen
   CASE #>0
     SET FORMAT TO amask1
     READ
     SET FORMAT TO SCREEN

   *** Wenn Nummer nicht gefunden, Benutzer warnen
   CASE teilnr # = 0
     § 10,5 SAY "Kein Teil mit der Nummer" + teilnr
     § 12,5 SAY "Weiter mit beliebiger Taste ..."
     WAIT

ENDCASE
```

Die DO WHILE-Schleife wird von einem ENDDO geschlossen, so daß der Benutzer immer weiter nach der nächsten Teilenummer gefragt wird:

```
ENDDO (while teilnr # " ")
```

Sind alle Änderungen erledigt, löscht das Programm die Variablen und kehrt zum Programm MMENU.CMD zurück:

```
*************** Fertig. Variablen löschen und zurück
RELEASE teilnr
RETURN
```

Die Abb. 10.16 zeigt das gesamte MEDIT.CMD-Programm.

```
********************************** MEDIT.CMD
********************************** Ändern der Inventar-Hauptdatei
USE master INDEX master
STORE "X" TO teilnr
DO WHILE teilnr#" "

   ************************ Die zu ändernde Teilenummer
   ************************ herausfinden
   ERASE
   STORE "      " TO teilnr
   §  5,5 SAY "Welche Teilenummer ändern?";
       GET teilnr
   READ

   ********************** Versuche die Teilenummer zu finden
   STORE !(teilnr) TO teilnr
   FIND &teilnr

   ************************** Wenn gefunden, mit AMASK1.FMT
   ************************** ändern, sonst Benutzer warnen
   DO CASE

      **** Wenn keine Nummer eingegeben, einfach zurück
      CASE teilnr=" "
        ERASE

      *** Wenn Nummer gefunden, Ändern ermöglichen
      CASE #>0
        SET FORMAT TO iscreen1
        READ
        SET FORMAT TO SCREEN

      *** Wenn Nummer nicht gefunden, Benutzer warnen
      CASE # = 0
        § 10,5 SAY "Kein Teil mit der Nummer" + teilnr
        § 12,5 SAY "Weiter mit beliebiger Taste ..."
        WAIT

   ENDCASE

ENDDO (while teilnr # " ")

***************** Fertig. Variablen löschen und zurück
RELEASE teilnr
RETURN
```

Abb. 10.16: Das Programm MEDIT.CMD

Die letzte Option im Menü des Programms MMENU.CMD gestattet es dem Benutzer, die Hauptdatei mit Informationen aus den Dateien SALES und NEWSTOCK zu aktualisieren. Allerdings können wir diesen Vorgang nicht testen, solange diese Dateien leer sind. Deshalb werden wir die Aktualisierungsprogramme erst dann entwickeln, wenn die Programme für SALES und NEWSTOCK fertig sind, was wir nach den nächsten beiden Kapiteln geschafft haben werden.

Zusammenfassung

Wir haben in diesem Kapitel verschiedene neue Programmiertechniken erforscht und einige andere benutzt, die wir bei unserer Adressenverwaltung kennengelernt hatten.

Wenn neue Teile in die Hauptdatei MASTER eingegeben werden, sorgt das Programm ADDNUMBS.CMD automatisch dafür, daß jede Nummer nur einmal gespeichert wird. In jedem System, das Schlüsselfelder für den Bezug zwischen Datenbanken benutzt, ist es wichtig, daß die Hauptdatei einmalige Daten im Schlüsselfeld jedes Datensatzes enthält (so, wie jede Kontonummer in einer Bank nur einmal vorkommen darf). Ein Programm mit eingebauter Prüfroutine, wie ADDNUMBS.CMD, sorgt für die Verwendbarkeit der Hauptdatei.

Wir haben auch einige Report-Dateien erstellt, um den gegenwärtigen Lagerbestand, die Nachbestellungen und die bestellten Teile aufzulisten. Dazu haben wir die Möglichkeiten des eingebauten Reportgenerators benutzt und in dem Programm MREPORTS.CMD ein Menü mit Auswahlmöglichkeiten anzubieten.

Das fortgeschrittenste Programm in diesem Kapitel war ORDERS.CMD, mit dem Bestellungen gemacht und hinterher automatisch ausgedruckt werden. Das Programm läuft durch alle Datensätze und zeigt alle an, deren Bestand unter die Nachbestellmenge gefallen ist. Der Benutzer kann dann eine Bestellung fertigmachen, indem er die zu bestellende Stückzahl eingibt. Sind alle Bestellungen eingegeben, druckt das Programm die Bestellzettel versandfertig aus. Die wichtigste Einzeltechnik in diesem Programm ist möglicherweise die Fähigkeit, für jeden Lieferanten eine Bestellung zu drucken, egal ob von ihm nur ein Teil oder Dutzende gebraucht werden. Das wurde durch den Einsatz von verschachtelten DO WHILE-Schleifen möglich (DO WHILE .NOT. EOF und DO WHILE lieferant = lschleife .AND. .NOT. EOF).

Das MEDIT.CMD-Programm erlaubt dem Benutzer Änderungen in der Hauptdatei und benutzt dazu sehr ähnliche Techniken wie das Programm zum Ändern von Adreßeinträgen in der Adressenverwaltung.

Inventar Verkaufssystem

11

In diesem Kapitel werden wir Programme entwickeln, die Daten in die SALES-Datenbank schreiben und Berichte entsprechend der Rechnungsnummer oder dem Datum drucken. Das Dateneingabeprogramm wird dem Verkäufer hilfreich zur Seite stehen. Die zum Eintragen von Verkäufen benutzte Bildschirmmaske wird genauso aussehen wie die Seite eines Verkaufsbuches und das Programm wird automatisch die Teilebezeichnung hineinschreiben, sobald der Verkäufer die Teilenummer eingibt. Tippt er eine nicht existierende Nummer ein, bekommt er eine entsprechende Warnung und kann es noch einmal versuchen. Das Programm wird außerdem automatisch Zwischensummen und Gesamtsummen berechnen und anzeigen, sobald Daten eingegeben werden. Rechnungsnummern werden ebenfalls automatisch erstellt und gespeichert.

Der Verkaufsmodul des Inventarsystems läßt einzelne Verkäufe als Geschäftsvorgänge zu. Die Daten dafür werden in der Datenbank SALES.DBF gespeichert, die wir schon vorher mit dieser Struktur erstellt haben:

```
STRUKTUR DER DATEI: B:SALES.DBF

FELDNR    NAME        TYP       WEITE      DEZ    Inhalt
001       TEIL:NR      C         005               Teilenummer
                                                   (Schlüsselfeld)
002       RECHNG:NR    N         006               Rechnungsnummer
003       VERKAEUFER   C         012               Verkäufer
```

```
004     KUNDE       C     012          Kunde
005     MENGE       N     004          Verkaufte Menge
006     PREIS       N     009     002  Verkaufspreis
007     DATUM       C     008          Verkaufsdatum
008     UEBERTR     L     001          Zur Hauptdatei
                                       geschickt?
```

Die Struktur des Verkaufssystems

Die Programme des Verkaufssystems werden an das Programm SMENU.CMD angehängt. Es gibt für jede Aufgabe ein Programm:

1. Neue Daten eingeben.

2. Berichte aus der Verkaufsdatenbank drucken.

3. Die Datei SALES ändern.

In diesem Kapitel werden wir die Programme SMENU, POS und SALREPS entwickeln. Das Programm SALEDIT schreiben wir erst im Kapitel 13, da die Änderungsprozeduren vom Aktualisieren beeinflußt werden.

Das Menü des Verkaufssystems

Entschließt sich der Benutzer im Hauptmenü die Option „2. Verkäufe eingeben" zu wählen, erscheint dieses Menü für die Verkäufe:

```
        Menü Verkauf

1.  Verkaufsdaten eingeben

2.  Verkaufsberichte drucken

3.  Verkaufsdaten ändern

4.  Zurück zum Hauptmenü
```

Nach der Anzeige dieses Menüs verzweigt SMENU.CMD zu dem vom Benutzer gewählten Unterprogramm.

Da dies auch wieder ein ganz einfaches Menü-Programm ist, übergehe ich den Pseudocode dafür und zeige das gesamte Programm in der Abb. 11.2.

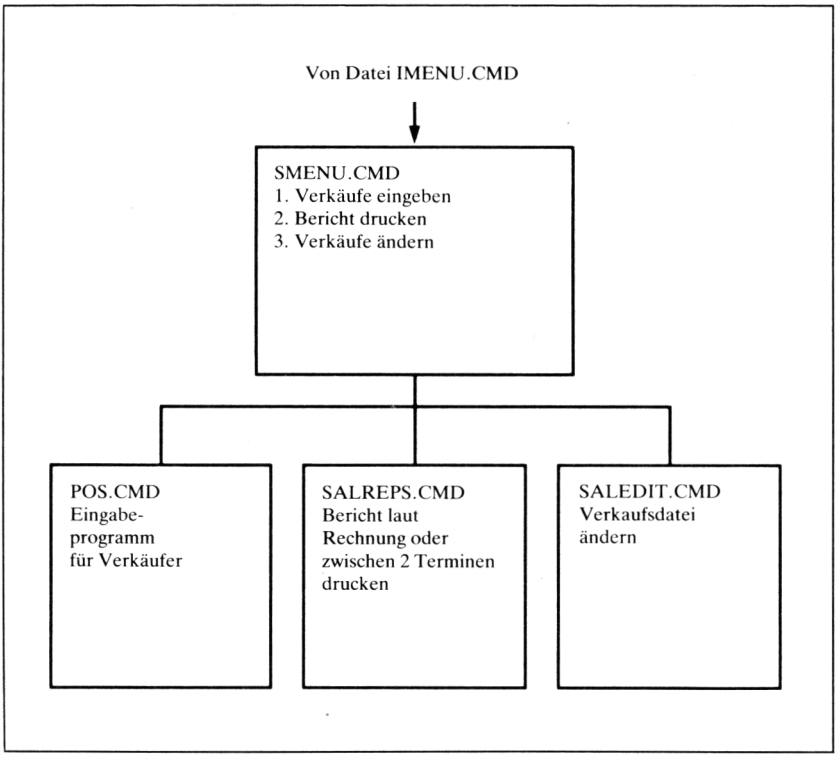


Abb. 11.1: Die Programmstruktur des Verkaufssystems

Eingabe der Verkaufsdaten

Das Programm zur Eingabe neuer Daten in die SALES-Datenbank arbeitet wie folgt: Sobald es angewählt wird, erscheint die Kopfleiste des Verkaufsblattes auf dem Bildschirm:

```
10/10/84                                  Rechnung Nr: 23456
Verkäufer:_                               : Kunde:            :
-------------------------------------------------------------
Teil #          Name            Menge     Preis       Gesamt
```

Das Programm füllt automatisch die Rubriken Datum und Rechnungsnummer aus, und der Verkäufer gibt dann seinen Namen und den des Kunden ein. Dann gibt er Verkäufe folgendermaßen ein:

```
****************************************** SMENU.CMD
**** Menu für den Abschnitt Verkauf des Inventarsystems

******** Beginne Schleife für das Menü
STORE " " TO swahl
DO WHILE SWAHL  #"4"
   ERASE
   TEXT

               Menü Verkauf

          1.  Verkaufsdaten eingeben

          2.  Verkaufsberichte drucken

          3.  Verkaufsdaten ändern

          4.  Zurück zum Hauptmenü

    ENDTEXT
    § 14,15 SAY "Wahl eingeben (1-4) " GET swahl PICTURE "9"
    READ

    ********** Verzweige zum entsprechenden Programm
    DO CASE

       CASE swahl = "1"
            DO pos

       CASE swahl = "2"
            DO salreps

       CASE swahl = "3"
            DO saledit

     ENDCASE

ENDDO (while swahl  #4)

****************** Wenn fertig, Variablen löschen
****************** und zurück zum Hauptmenü
RELEASE swahl
RETURN
```

Abb. 11.2: Das Programm SMENU.CMD

Zuerst erscheint ein Eingabefeld für die Teilenummer auf dem Bildschirm, wie hier gezeigt:

```
10/10/84                                    Rechnung Nr: 23456
Verkäufer:ACS                               : Kunde:    Jane Doe:
-------------------------------------------------------------
Teil #          Name            Menge      Preis        Gesamt
:_     :
```

Dann gibt der Verkäufer eine Teilenummer ein. Existiert diese in der Datenbank nicht, erscheint auf dem Bildschirm eine Warnung:

```
10/10/84                                    Rechnung Nr: 23456
Verkäufer:ACS                               : Kunde:    Jane Doe:
-------------------------------------------------------------
Teil #          Name            Menge      Preis        Gesamt
:300-Z: ** Teilenummer existiert nicht **
```

An dieser Stelle kann er noch einmal eine andere Nummer versuchen. Gibt er eine gültige Nummer ein, erscheinen der Name des Teils sowie weitere Eingabefelder für die Menge und den Verkaufspreis automatisch:

```
10/10/84                                    Rechnung Nr: 23456
Verkäufer:ACS                               : Kunde:    Jane Doe:
-------------------------------------------------------------
Teil #          Name            Menge      Preis        Gesamt
:A-111:    Schneeschuhe         :_    :    :          :
```

Gibt der Verkäufer zum Beispiel für Menge 5 ein und 10.00 DM als Verkaufspreis dazu, berechnet das Programm den Gesamtpreis, zeigt ihn an und wartet auf die nächste Teilenummer:

```
10/10/84                                    Rechnung Nr: 23456
Verkäufer:ACS                               : Kunde:    Jane Doe:
-------------------------------------------------------------
Teil #          Name            Menge      Preis        Gesamt
:A-111:    Schneeschuhe         :   5:    :    10.00: :    50.00:
:_     :
```

Werden hier wieder eine gültige Nummer sowie die verkaufte Menge und der Preis eingegeben, erscheint wieder der Gesamtpreis auf dem Bildschirm:

```
10/10/84                                    Rechnung Nr: 23456
Verkäufer:ACS                               : Kunde:    Jane Doe:
-------------------------------------------------------------
Teil #          Name            Menge      Preis        Gesamt
:A-111:    Schneeschuhe         :   5:    :    10.00: :    50.00:
:Z-999:    Bärenfell            :   1:    :   100.00: :   100.00:
:_     :
```

Waren das alle Verkäufe, gibt der Verkäufer einfach RETURN anstelle einer neuen Teilenummer ein. Die Gesamtsumme der Rechnung sowie die Frage nach einer weiteren Rechnungseingabe werden so angezeigt:

```
10/10/84                          Rechnung Nr: 23456
Verkäufer:ACS                     : Kunde:    Jane Doe:
-------------------------------------------------------
Teil #          Name       Menge  Preis       Gesamt
:A-111:     Schneeschuhe  :  5:  :    10.00: :    50.00:
:Z-999:     Bärenfell     :  1:  :   100.00: :   100.00:
:     :

GESAMTSUMME                                    150.00 DM

Neue Rechnung eingeben? (J/N) :_:
```

Der Verkäufer kann jetzt eine neue Rechnung eingeben, indem er einfach J eintippt. Möchte er nicht weitermachen, gibt er einfach N auf die oben gestellte Frage ein. Daraufhin erscheint dann wieder das Verkauf-Menü.

Das Programm zum Eingeben der Verkäufe heißt POS.CMD und der Pseudocode steht in der Abb. 11.3.

Die erste Aufgabe des POS-Programms ist, die zuletzt verwendete Rechnungsnummer herauszufinden. Das erledigen wir einfach dadurch, daß wir die Rechnungsnummer der letzten Rechnung gleich zu Beginn des Programms in die Variable LRECHNG speichern:

```
****************************************** POS.CMD
**************** Eingabeprogramm für Verkäufe

********** Hole die nächste Rechnungsnummer aus
********** der Datenbank
********** SALES
SET DEFA TO B
USE sales
GO BOTT
STORE rechng:nr TO lrechng
```

Das POS-Programm benutzt zwei Datenbanken: die Hauptdatei MASTER, um nach gültigen Teilenummern zu suchen und die Bezeichnung der Teile anzeigen zu können sowie die Datenbank SALES, um Verkäufe zu speichern. Die nächsten Zeilen stellen diese Beziehung her:

```
********** Die Datenbanken als „erste" und „zweite"
********** eröffnen
SELECT PRIMARY
USE master INDEX master
SELECT SECONDARY
USE sales INDEX sales
```

Die nächste Aufgabe des Programms besteht darin, eine Schleife zu beginnen, die das fortlaufende Eingeben von Rechnungen ermöglicht. Die Variable WEITER wird diese Schleife kontrollieren.

```
Hole die nächste Rechnungsnummer aus SALES.DBF

Arbeite mit der ersten (MASTER) und der zweiten (SALES) Datei

Beginne Schleife zum Anzeigen des Verkaufsformulars
    Zeige Kopf des Formulars auf dem Bildschirm
    Erhöhe Rechnungsnummer um 1
    Eröffne die Variablen
    Hole den oberen Teil der Rechnung
    Beginne Schleife für jedes Teil auf der Rechnung

        Beginne Schleife für einzelne Zeilen auf der Rechnung

        Beginne Schleife zum Prüfen der Teilenummer

            Frage nach Teilenummer

            Stelle sicher, daß Teil existiert

            Entscheide, was als nächstes zu tun,
            abhängig von der Existenz einer Teilenummer

            Fall 1: Keine Teilenummer eingegeben
            Programmende

            Fall 2: Nummer existiert nicht
            Benutzer warnen, neuer Versuch

            Fall 3: Teil existiert
            Beschreibung des Teils anzeigen
```

Abb. 11.3: Pseudocode für das POS.CMD-Programm

Nach Menge und Preis fragen
Summe berechnen und anzeigen
Gesamtsumme berechnen
Leeren Datensatz an SALES anhängen,
Feldinhalte übernehmen

Weiter mit Schleife (für gültige Nummer)

Weiter mit Schleife (für einzelne Zeilen)

Gesamtsumme zeigen und Bildschirm vor nächster Rechnung anhalten

Weiter mit Schleife für Rechnungen, bis Benutzer Ende wünscht

Speichervariablen löschen
Zurück zum Menü Verkauf

Abb. 11.3: Pseudocode für das POS.CMD-Programm (Forts.)

Jede Rechnung soll eine einmalige Rechnungsnummer erhalten, weshalb die Variable LRECHNG nach jeder Rechnung um 1 erhöht wird. Außerdem müssen die Variablen MVERKAEUF und MKUNDE angelegt werden, um die Namen des Verkäufers und des Kunden darin zu speichern. Die Gesamtsumme der Rechnung wird in der Variablen GESSUM gespeichert, die für jede Rechnung erst einmal eine 0 enthalten muß. Dann soll der Kopf des Verkaufsblattes auf dem Bildschirm erscheinen, damit es der Verkäufer ausfüllen kann. Die folgenden Zeilen erledigen die anfallenden Arbeiten:

```
********** Beginne Schleife zur Rechnungserstellung
STORE "J" TO weiter
DO WHILE weiter = "J"
   ************* Kopf des Verkaufsblattes aufbauen
   ERASE
   STORE lrechng+1 TO lrechng
   STORE "             " TO mverkauef,mkunde
   STORE 0 TO gessum
   § 1,2 SAY t:datum
   § 1,30 SAY "Rechnung Nr: "+STR(lrechng,5)
   § 2,2 SAY "Verkäufer " GET mverkaeuf
   § 2,35 SAY "Kunde " GET mkunde
```

```
   § 3,1 SAY  "------------------"
   § 4,1 SAY "Teil #       Name         Menge
Preis     Gesamt"
   READ
```

Als nächstes muß das Programm eine Schleife für jedes einzelne Teil der Rechnung beginnen. Die Schleife nimmt Eintragungen ab Zeile sechs des Bildschirms entgegen, bis der Bildschirm voll ist (ZEILE = 20) oder der Benutzer keine Teilenummer mehr eingibt (TEILNR = " "). Die nächsten Zeilen eröffnen die Variablen und beginnen die Schleife:

```
******** Beginne Schleife für jedes einzelne Teil
STORE 6 TO zeile
STORE "X" TO teilnr

DO WHILE zeile < 20 .AND. teilnr # " "
   STORE " " TO teilnr
```

Dann müssen wir noch eine Schleife aufbauen, um die Richtigkeit der Teilenummern zu überprüfen. Die Variable OK bekommt den Wert wahr (T), sobald eine gültige Teilenummer (oder der Befehl zum Aufhören) eingegeben wurde. Um sicherzustellen, daß die Schleife wenigstens einmal durchlaufen wird, speichern wir den Wert unwahr (F) in OK. Diese Zeilen eröffnen die Schleife:

```
STORE F TO ok
********** Beginne Schleife zum Prüfen der Nummern
DO WHILE .NOT. ok
```

Die Menge und der Verkaufspreis der verkauften Teile werden in den Variablen STUECK und VK gespeichert, weshalb diese erst einmal mit dem Wert Null eröffnet werden müssen, wie Sie hier sehen:

```
**************** Verkaufsblatt-Variablen eröffnen
STORE 0 TO stueck
STORE 0.00 TO vk
```

Jetzt kann das Programm nach der Teilenummer des verkauften Teils fragen und die Antwort in der Variablen TEILNR ablegen:

```
************** Nach Teilenummer fragen
STORE " " TO teilnr
§ zeile,2 GET teilnr
READ
```

Dann muß das Programm, wie Sie sehen, überprüfen, ob die Teilenummer überhaupt exisiert:

```
********* Sicherstellen, daß die Nummer existiert
STORE !(TRIM(teilnr)) TO teilnr
SELE PRIM
FIND &teilnr
```

Für die nächsten drei Entscheidungsmöglichkeiten brauchen wir eine DO CASE-Anweisung, um festzustellen, ob der Benutzer aufhören will, eine ungültige oder eine gültige Teilenummer eingegeben hat. Gibt der Benutzer keine Teilenummer ein, wird die Variable OK wahr, da dies eine gültige Eingabe ist und die OK-Variable die Schleife DO WHILE .NOT. ok kontrolliert. Das Programm überspringt dann die restlichen CASE-Anweisungen, da der Benutzer aufhören will:

```
********** Entscheiden, was als nächstes zu tun ist,
********** in Abhängigkeit von der Benutzereingabe
DO CASE

    *** Fall 1: Keine Teilenummer eingegeben
    CASE teilnr=" "
      STORE T TO ok
```

Wurde eine ungültige Teilenummer eingegeben, muß das Programm den Benutzer warnen und eine neue Eingabe ermöglichen. Das Programm erledigt diese Aufgabe, indem es die Variable OK auf den Wert unwahr (F) setzt, wie Sie in der CASE-Anweisung hier sehen:

```
    *** Fall 2: Teil existiert nicht
    CASE # = 0
      § zeile,10 SAY "*** Teilenummer existiert;
                                      nicht ***"
      STORE F TO ok
```

Gibt der Benutzer eine richtige Teilenummer ein, soll das Programm weitermachen und die restlichen Daten des Vorgangs holen. Zuerst zeigt es den Namen des Teils an (TITEL), fragt dann nach der Menge (STUECK) und dem Verkaufspreis (VK). Mit diesen Zeilen wird das möglich:

```
    *** Fall 3: Teil existiert
      CASE # > 0
        § zeile,10 SAY titel
        § zeile,25 GET stueck PICT "999"
```

```
        § zeile,30 GET vk PICT "999.99"
        READ
```

Das Programm soll dann die Summe für den Verkauf anzeigen und den Betrag zu einer Gesamtsumme für die Rechnung in der Variablen GESSUM addieren. Außerdem soll die Variable OK auf wahr (T) gesetzt werden, da eine gültige Teilenummer eingegeben wurde. Die folgenden Zeilen bewerkstelligen dies:

```
********** Berechne und zeige Zwischensumme
§ zeile,40 SAY stueck*vk;
           USING "DM ####.##"
STORE gessum+stueck*vk TO;
      gessum
STORE T TO ok
```

Nachdem die Daten für den Geschäftsvorgang eingegeben sind, sollten sie in der Datenbank SALES (secondary) gespeichert werden. Ein APPEND BLANK-Befehl, gefolgt von einer Reihe REPLACE-Befehlen, erledigt das:

```
********** Einen Datensatz wegspeichern
SELECT SECONDARY
APPEND BLANK
REPLACE datum WITH t:datum
REPLACE verkaeufer WITH mverkaeuf
REPLACE rechng:nr WITH lrechng
REPLACE kunde WITH mkunde
REPLACE teil:nr WITH teilnr
REPLACE menge WITH stueck
REPLACE preis WITH vk
REPLACE uebertr WITH F
```

Das beendet die DO CASE-Anweisung und auch die Prüfschleife für die Gültigkeit einer Teilenummer. Fügen Sie diese Zeilen noch ein:

```
  ENDCASE
ENDDO (while teilenr not ok)
```

Jetzt kann der Benutzer den nächsten Verkauf eingeben. Die Variable ZEILE wird um 1 erhöht, da diese Angabe bestimmt, in welche Zeile des Bildschirms Daten eingegeben werden können:

```
 STORE zeile + 1 TO zeile
ENDDO (while verkäufe eingeben)
```

Nachdem alle Bestandteile einer Rechnung eingetragen sind, zeigt das Programm die Gesamtsumme und fragt an, ob eine weitere Rechnung eingegeben werden soll oder nicht und macht dann mit der Schleife weiter (wenn der Benutzer es so will). Die folgenden Zeilen sorgen dafür:

```
*************** Gesamtsumme der Rechnung zeigen,
*************** Bildschirm anhalten vor der nächsten
*************** Rechnung
§ zeile,40 SAY gessum USING "DM ####.##"
§ zeile+2,2 SAY "Neue Rechnung eingeben? (J/N)";
            GET weiter PICTURE "!"
READ

ENDDO (while weiter=j)
```

Sobald der Verkäufer mit der Eingabe aller Rechnungen fertig ist, kann das Programm die Variablen löschen, die erste und die zweite Datenbank schließen und zum Programm SMENU zurückkehren. Gesteuert wird diese Aktion mit den folgenden Zeilen:

```
******** Variablen löschen, Datenbanken schließen
******** und zurück zu SMENU.CMD
RELEASE lrechng,mkunde,stueck,vk
RELEASE weiter,zeile,teilnr,mverkaeuf,gessum,ok
SELE SECO
USE
SELE PRIM
USE
RETURN
```

Das Programm POS.CMD ist zwar recht komplex, bietet dem Anwender aber ein hohes Maß an Bequemlichkeit. Es verringert auch die Fehlerhäufigkeit, weil alle Teilenummern auf Gültigkeit überprüft werden, und übernimmt alle Berechnungen. Es ist ein sehr allgemein gehaltenes Programm für die „Verkaufsfront" und muß möglicherweise für den Einsatz in einem bestimmten Fall überarbeitet werden. Aber die grundsätzlichen Prozeduren zum Überprüfen von Daten in der Hauptdatei, der automatischen Berechnung und Eingabe sowie dem Speichern in der Verkäufedatei, werden in den meisten Fällen ihre Gültigkeit haben. Die Abb. 11.4 zeigt das gesamte POS.CMD-Programm.

Verkaufsberichte

Die zweite Option des Verkaufsmenüs gestattet dem Benutzer, Berichte aus der SALES-Datenbank zu drucken. Dabei können die Berichte ent-

```
********************************************** POS.CMD
*************** Eingabeprogramm für Verkäufe

*********** Hole die nächste Rechnungsnummer aus der Datenbank
*********** SALES
SET DEFA TO B
USE sales
GO BOTT
STORE rechng:nr TO lrechng

********** Die Datenbanken als ,,erste" und ,,zweite" eröffnen
SELECT PRIMARY
USE master INDEX master
SELECT SECONDARY
USE sales INDEX sales

************** Beginne Schleife zur Rechnungserstellung
STORE "J" TO weiter
DO WHILE weiter = "J"
   ******************** Kopf des Verkaufsblattes aufbauen
   ERASE
   STORE lrechng+1 TO lrechng
   STORE "             " TO mverkaeuf,mkunde
   STORE 0 TO gessum
   § 1,2 SAY t:datum
   § 1,30 SAY "Rechnung Nr: "+STR(lrechng,5)
   § 2,2 SAY "Verkäufer " GET mverkaeuf
   § 2,35 SAY "Kunde " GET mkunde
   § 3,1 SAY  "------------------"
   § 4,1 SAY "Teil # Name       Menge   Preis   Gesamt"
   READ

   ********************** Beginne Schleife für jedes einzelne Teil
   STORE 6 TO zeile
   STORE "X" TO teilnr

   DO WHILE zeile < 20 .AND. teilnr  #" "
      STORE "     " TO teilnr
      STORE F TO ok

      ********************** Beginne Schleife zum Prüfen der
      ********************** Nummern
      DO WHILE .NOT. ok

         ********************** Verkaufsblatt-Variablen eröffnen
         STORE 0 TO stueck
         STORE 0.00 TO vk
```

Abb. 11.4: Das Programm POS.CMD

```
************** Nach Teilenummer fragen
STORE "      " TO teilnr
§ zeile,2 GET teilnr
READ

************* Sicherstellen, daß die Nummer existiert
STORE !(TRIM(teilnr)) TO teilnr
SELE PRIM
FIND &teilnr

***************** Entscheiden, was als nächstes zu tun
***************** ist, in Abhängigkeit von der
***************** Benutzereingabe
DO CASE

   *** Fall 1: Keine Teilenummer eingegeben
   CASE teilnr=" "
     STORE T TO ok

   *** Fall 2: Teil existiert nicht
   CASE # = 0
     § zeile,10 SAY "*** Teilenummer existiert nicht ***"
     STORE F TO ok

   *** Fall 3: Teil existiert
   CASE # > 0
     § zeile,10 SAY titel
     § zeile,25 GET stueck PICT "999"
     § zeile,30 GET vk PICT "999.99"
     READ

     ************** Berechne und zeige Zwischensumme
     § zeile,40 SAY stueck*vk;
                USING "DM ####.##"
     STORE gessum+stueck*vk TO;
           gessum
     STORE T TO ok

     ************ Einen Datensatz wegspeichern
     SELECT SECONDARY
     APPEND BLANK
     REPLACE datum WITH t:datum
     REPLACE verkaeufer WITH mverkaeuf
     REPLACE rechng:nr WITH lrechng
     REPLACE kunde WITH mkunde
     REPLACE teil:nr WITH teilnr
     REPLACE menge WITH stueck
```

Abb. 11.4: Das Programm POS.CMD (Forts.)

```
            REPLACE preis WITH vk
            REPLACE uebertr WITH F

         ENDCASE
      ENDDO (while teilenr not ok)

      STORE zeile + 1 TO zeile
   ENDDO (while verkäufe eingeben)

   ************************** Gesamtsumme der Rechnung zeigen,
   ************************** Bildschirm anhalten vor der
   ************************** nächsten Rechnung
   § zeile,40 SAY gessum USING "DM ####.##"
   § zeile+2,2 SAY "Neue Rechnung eingeben? (J/N)";
               GET weiter PICTURE "!"
   READ

ENDDO (while weiter=j)

********************* Variablen löschen, Datenbanken schließen
********************* und zurück zu SMENU.CMD
RELEASE lrechng,mkunde,stueck,vk
RELEASE weiter,zeile,teilnr,mverkaeuf,gessum,ok
SELE SECO
USE
SELE PRIM
USE
RETURN
```

Abb. 11.4: Das Programm POS.CMD (Forts.)

weder auf einer einzelnen Rechnungsnummer oder auf den Vorgängen eines bestimmten Zeitraumes beruhen (zum Beispiel alle Verkäufe zwischen dem 01/01/04 und dem 31/03/84). Sobald die Option 2 gewählt wird, erscheint dies Menü:

```
          Mögliche Verkaufsberichte

    1.    Nach Rechnungsnummer

    2.    Nach Datum

    3.    Zurück zum Menü Verkauf
```

Wählt der Bediener die Option 1, fragt das Programm:

```
Nach welcher Rechnungsnummer suchen? :_
```

Daraufhin gibt er die Rechnungsnummer ein, und der Bildschirm (oder der Drucker) listet die Daten zu dieser Rechnungsnummer so auf:

```
Rechnungsnummer: 12345                          Datum 04/05/84
Verkäufer: ACS                                  Kunde:D. Jones

Teil #                  Menge          Preis          Summe
Z-999                      2          12.00          24.00
A-113                      3          15.00          45.00
A-111                      1          90.00          90.00
```

Wählt der Benutzer die Option 2, um Daten innerhalb eines bestimmten Zeitraumes anzuzeigen, wird er gefragt:

```
Anfangsdatum :  /  /  :
Enddatum     :  /  /  :
```

Hier kann er dann die gewünschten Datumsgrenzen eingeben. Danach wird dann der Bildschirm (oder Drucker) alle Verkäufe für die gewünschte Periode auflisten, so wie Sie es hier sehen:

```
Seite Nr. 00001          Zeitraum vom 05/05/84 bis 05/06/84

                                Verkäufe

Datum      Rechng.  Verkäufer Kunde      Teil #   Menge    VK-
Preis
05/05/84     123       SHS    D. Jones    A-111      2      20.00
05/05/84     123       SHS    D. Jones    A-113      2      14.00
05/05/84     123       SHS    D. Jones    Z-999      4      50.00
05/06/84     124       ACS    A. Schmidt  Z-999      3      25.00
05/06/84     124       ACS    A. Schmidt  A-111      5      10.00
05/06/84     125       CGG    Z. Zeppo    Z-999      3      30.00
```

Sie können zur Erstellung des zusammenfassenden Berichts den in dBASE eingebauten Reportgenerator benutzen, den Sie mit dem Befehl REPORT aufrufen.

Hinter den dBASE-Punkt schreiben Sie:

```
USE sales
REPORT FORM sales
```

und antworten auf die Fragen, wie hier abgebildet:

```
ENTER  OPTIONS,M=LEFT MARGIN,L=LINES/PAGE,
W=PAGE WIDTH M=1,W=65
PAGE HEADING? (Y/N) Y
ENTER PAGE HEADING: Verkäufe
DOUBLE SPACE REPORT? (Y/N) N
ARE TOTALS REQUIRED? (Y/N) N
COL     WIDTH,CONTENTS
001     9,DATUM
ENTER HEADING: Datum
002     7,RECHNG:NR
ENTER HEADING: Rechng.
003     14,VERKAEUFER
ENTER HEADING: Verkaeufer
004     14,KUNDE
ENTER HEADING: Kunde
005     6,TEIL:NR
ENTER HEADING: Teile-;nummer
006     5,MENGE
ENTER HEADING: Menge
007     10,PREIS
ENTER HEADING: Verkaufs-;preis
008
```

Das Programm, das den Benutzer nach dem zu druckenden Report fragt und diesen Report dann auch anzeigt, heißt SALREPS.CMD. Der dazugehörende Pseudocode steht in der Abb. 11.5.

Das Programm beginnt, indem es die Datei SALES ohne Index aufruft. Sie erinnern sich, daß der Index für die Bestellungen nach der Teilenummer notwendig war. Für diese Berichte jetzt möchten wir die Verkäufe nach Datum oder Rechnungsnummer gruppiert sehen. Da unser Verkaufprogramm automatisch Rechnungsnummern erstellt und das Datum in das Formblatt überträgt, werden die Datensätze nach dem Datum sortiert sein, wenn kein Index benutzt wird. Die ersten Zeilen des SALREPS-Programms öffnen die SALES-Datenbank ohne Index-Datei:

```
****************************************** SALREPS.CMD
****************** Berichte aus SALES drucken
USE sales
```

Dann müssen wir eine Schleife für das Menü beginnen. Außerdem müssen wir die Menüwahl des Benutzers in der Variablen REPWAHL und seine Entscheidung, ob gedruckt werden soll oder nicht, in der Variablen JN speichern.

```
Benutze die Datenbank SALES
Beginne eine Schleife für das Menü
Biete Wahlmöglichkeiten an

    1. Nach Rechnungsnummer
    2. Nach Datum
    3. Zurück zum Menü Verkauf
Die Wahl des Benutzers entgegennehmen

Wenn nicht Ende, fragen ob Daten ausgedruckt werden sollen

Den entsprechenden Report ausdrucken

    CASE 1: Suchen nach Rechnungsnummer
    Benutzer nach der zu druckenden Nummer fragen
    Ersten Vorgang mit dieser Nummer finden
    Wenn drucken gewünscht, Drucker einschalten
        Informationen aus dem Formularkopf des ersten
        Datensatzes mit dieser Nummer drucken
        Daten der restlichen Datensätze mit dieser Nummer
        drucken
        Drucker ausschalten

    CASE 2: Suchen nach Datum
    Leerzeichen in Such-Daten speichern
    Nach Startdatum fragen
    Nach Enddatum fragen
    Bildschirm löschen
    Makro für die Datumsuche aufbauen
    Makro für den Drucker aufbauen, wenn nötig
    Datumzeile für den Report erstellen
    Report mit Daten aus SALES.DBF drucken

Wenn nicht gedruckt werden soll und nicht Ende, Bildschirm
anhalten
Weiter mit der Menüschleife (solange Benutzer nicht
aufhören möchte
Wenn fertig, Variablen löschen und zurück zu SMENU
```

Abb. 11.5: Pseudocode für das Programm SALREPS.CMD

Diese Zeilen eröffnen die Variablen und die Schleife, zeigen dann das Menü und warten auf eine Eingabe:

```
STORE " " TO repwahl,jn
DO WHILE repwahl # "3"
   ERASE
   TEXT
                          Mögliche Verkaufsberichte

                     1.   Nach Rechnungsnummer

                     2.   Nach Datum

                     3.   Zurück zum Menü Verkauf
    ENDTEXT
    § 8,12 SAY " Ihre Wahl (1-3)?";
           GET repwahl PICT "9"
    READ
```

Dann fragen Sie den Benutzer, ob der Bericht ausgedruckt werden soll. Das geht so:

```
************************ Nach Ausdruck fragen
    ERASE
    STORE " " TO jn,drucker
    IF repwahl # "3"
       § 5,5 SAY "Bericht ausdrucken? ";
             GET jn PICT "!"
       READ
    ENDIF (repwahl # 3)
```

Wie Sie sehen, fragt das Programm nur dann nach dem Drucker, wenn die Option 3 (Zurück zum Hauptmenü) nicht angewählt wurde.

Wir müssen noch eine DO CASE-Anweisung einfügen, damit der Benutzer den entsprechenden Bericht drucken kann. Will er nach Rechnungsnummern ausdrucken, muß das Programm zuerst nach der Rechnungsnummer fragen, die Kopfzeilen des Formblattes (Nummer, Datum, Verkäufer und Kunde) anzeigen und dann die Daten der vorhandenen Vorgänge mit dieser Rechnungsnummer auflisten. Diese Zeilen erledigen das:

```
********************** Drucke entsprechenden Report
ERASE
DO CASE

   ******* Fall 1: Suchen nach Rechnungsnummern
   CASE repwahl = "1"
```

```
INPUT "Nach welcher Rechnungsnummer suchen? ";
      TO suchnr
ERASE
LOCATE FOR rechng:nr=suchnr

******* Wenn gefunden, drucke Rechnung
IF .NOT. EOF
   IF jn="J"
      SET PRINT ON
   ENDIF (jn=j)
   ? "Rechnungsnummer ",rechng:nr
   ?? "Datum",datum
   ? "Verkäufer",verkaeufer,"Kunde",kunde
   ?
   ? "Teil # Menge Preis Summe"
   LIST OFF WHILE rechng:nr=suchnr;
        teil:nr,menge,preis,(menge*preis)
   EJECT
   SET PRINT OFF
ENDIF (not eof)
```

Wählt der Benutzer die Option 2, muß das Programm nach dem Anfangs- und Enddatum fragen, und dann die Datei REPORT FORM SALES.FRM zum Ausdruck des Berichts für alle Datensätze, die innerhalb der eingegebenen Daten liegen, benutzen. Das zweite CASE sieht deshalb so aus:

```
********** Fall 2:Suchen nach Datum
CASE repwahl = "2"
  STORE "        " TO start,ende
  § 5,3 SAY "Startdatum ";
        GET start PICT "99/99/99"
  § 7,3 SAY "Enddatum   ";
        GET ende PICT "99/99/99"
  READ
  ERASE

  ******* Makro für Datumsuche erstellen
  STORE "FOR datum>= start .AND. ;
         datum <= ende" TO makro
  ******* Makro für den Drucker erstellen
  IF jn = "J"
     STORE "TO PRINT" TO drucker
  ELSE
```

```
      STORE " " TO drucker
   ENDIF (jn=j)

   ******* Überschrift erstellen
   SET HEAD TO Zeitraum vom &start bis &ende

   ******* REPORT FORM sales &makro &drucker

ENDCASE
```

Beachten Sie, wie hier oben Makros verwendet wurden. Der Benutzer gibt das Anfangs- und Enddatum der zu druckenden Daten ein, und das Programm speichert diese in den Variablen START und ENDE. Dann erstellt die Zeile

```
STORE "FOR datum>=start .AND. datum<=ende" TO makro
```

eine Variable mit dem Namen MAKRO. Zusätzlich speichert eine IF-Abfrage die Wörter "TO PRINT" in die Variable DRUCKER, wenn der Bericht ausgedruckt werden soll.

Stellen Sie sich vor, der Benutzer gibt 01/01/84 als Anfangsdatum und 31/03/84 als Enddatum ein und drückt zusätzlich noch J als Anwort auf die Frage nach dem Drucker. Das Programm enthält diese Zeile:

```
REPORT FORM sales &makro &drucker
```

Wenn das Programm läuft, ersetzt dBASE die beiden Makros, und die Zeile sieht dann so aus:

```
REPORT  FORM  sales  FOR datum>=start  .AND.  datum
                                              <=ende;
      TO PRINT
```

Das stellt sicher, daß nur die Daten, die in den angeforderten Zeitraum fallen, auch ausgedruckt werden.

Die nächste Routine des SALREPS-Programms hält einfach die Bildschirmanzeige an, wenn der Bericht weder ausgedruckt werden soll, noch der Abbruch des Programms verlangt wurde:

```
***** Wenn nicht zum Drucker oder Ende des Programms
***** gewünscht, Bildschirm anhalten
IF jn # "J" .AND. repwahl # "3"
   ?
   ?
   WAIT
ENDIF (jn#j)
```

Die restlichen Zeilen schließen die DO WHILE-Schleife und löschen die Variablen, bevor es zum Programm SMENU zurückgeht:

```
ENDDO (while wahl #3)

******* Wenn fertig, Variablen löschen und zurück
******* zu SMENU
RELE repwahl,jn,drucker,start,ende,makro,suchnr
RETURN
```

Die Abb. 11.6 zeigt das komplette SALREPS-Programm.

```
********************************************** SALREPS.CMD
************************* Berichte aus SALES drucken
USE sales

STORE " " TO repwahl,jn
DO WHILE repwahl  #"3"
   ERASE
   TEXT
                        Mögliche Verkaufsberichte

                    1.  Nach Rechnungsnummer

                    2.  Nach Datum

                    3.  Zurück zum Menü Verkauf
    ENDTEXT
    § 10,12 SAY " Ihre Wahl (1-3)?";
           GET repwahl PICT "9"
    READ

************************ Nach Ausdruck fragen
    ERASE
    STORE " " TO jn,drucker
    IF repwahl # "3"
       § 5,5 SAY "Bericht ausdrucken? ";
             GET jn PICT "!"
       READ
    ENDIF (repwahl # 3)

********************* Drucke entsprechenden Report
ERASE
DO CASE
```

Abb. 11.6: Das Programm SALREPS.CMD

```
    ******* Fall 1: Suchen nach Rechnungsnummern
    CASE repwahl = "1"
      INPUT "Nach welcher Rechnungsnummer suchen? ";
          TO suchnr
      ERASE
      LOCATE FOR rechng:nr=suchnr

      ******* Wenn gefunden, drucke Rechnung
      IF .NOT. EOF
         IF jn="J"
            SET PRINT ON
         ENDIF (jn=j)
         ? "Rechnungsnummer ",rechng:nr
         ?? "Datum",datum
         ? "Verkäufer",verkaeufer,"Kunde",kunde
         ?
         ? "Teil  # Menge    Preis        Summe"
         LIST OFF WHILE rechng:nr=suchnr;
              teil:nr,menge,preis,(menge*preis)
         EJECT
         SET PRINT OFF
      ENDIF (not eof)

************ Fall 2: Suchen nach Datum
CASE repwahl = "2"
  STORE "        " TO start,ende
  § 5,3 SAY "Startdatum ";
        GET start PICT "99/99/99"
  § 7,3 SAY "Enddatum   ";
        GET ende PICT "99/99/99"
  READ
  ERASE

  ******* Makro für Datumsuche erstellen
  STORE "FOR datum>= start .AND. ;
         datum <= ende" TO makro
  ******* Makro für den Drucker erstellen
     IF jn = "J"
     STORE "TO PRINT" TO drucker
  ELSE
     STORE " " TO drucker
  ENDIF (jn=j)

  ******* Überschrift erstellen
  SET HEAD TO Zeitraum vom &start bis &ende

  ******* Drucke den Bericht
  REPORT FORM sales &makro &drucker
```

Abb. 11.6: Das Programm SALREPS.CMD (Forts.)

```
ENDCASE
**************** Wenn nicht zum Drucker oder Ende des Programms
**************** gewünscht, Bildschirm anhalten
IF jn  #"J" .AND. repwahl  #"3"
   ?
   ?
   WAIT
ENDIF (jn#j)

ENDDO (while wahl #3)

******* Wenn fertig, Variablen löschen und zurück zu SMENU
RELE repwahl,jn,drucker,start,ende,makro,suchnr
RETURN
```

Abb. 11.6: Das Programm SALREPS.CMD (Forts.)

Eine Bemerkung zum Datum

Mit der Methode, mit der wir nach Verkäufen innerhalb eines bestimmten Bereichs von Kalenderdaten gesucht haben, können wir nur innerhalb eines Kalenderjahres vernünftige Aussagen bekommen. Reicht der angegebene Zeitraum über einen Jahreswechsel hinaus, werden einige Änderungen notwendig.

Um dieses Problem zu verstehen, versuchen Sie einmal, wie ein Computer zu denken. Angenommen, eine Datenbank enthält Datumeinträge von 1984 und 1985. Der Benutzer möchte alle Datensätze sehen, die zwischen dem 01/01/85 und dem 31/01/85 liegen.

Das Programm benutzt dazu den folgenden Suchbefehl, um die Datensätze innerhalb des geforderten Zeitraums auszusuchen:

```
FOR datum >="01/01/85" .AND. datum <="31/01/85
```

Bedauerlicherweise werden eine Reihe von 1984er Daten mit in der Liste erscheinen. Zum Beispiel auch das Datum 15/01/84. Wenn Sie die Schrägstriche aus den Datumangaben entfernen, sehen Sie auch warum das so ist: 150184 ist größer als 010185 und kleiner als 310185. Also schiebt dBASE den 15.1.84 zwischen den 1.1.85 und den 31.1.85. Was können wir dagegen tun?

Das Problem liegt in der Form (TT/MM/JJ), in der wir unser Datum eingeben. Richtige Vergleiche sind nur möglich, wenn die wichtigste Stelle des Datums, die Jahresangabe, an erster Stelle steht und so zuerst verglichen wird. Wenn dBASE zum Beispiel alle Daten auflisten soll, die zwi-

schen 85/01/01 und 85/31/01 liegen, würde das Datum 84/15/01 nicht dazugehören, weil es weniger als 85/01/01 ist (das heißt, 841501 liegt nicht zwischen 850101 und 853101).

Deshalb müssen wir unsere Suchroutine so modifizieren, daß sie bei der Suche zuerst die Jahreszahlen vergleicht. Im Programm SALREPS werden die Such-Daten in den Variablen START und ENDE im Format TT/MM/JJ gespeichert. Die folgenden Zeilen Programmtext werden dieses Format in das benötigte (YY/MM/TT) umformen:

```
STORE $(start,7,2) + $(start,4,2)+ $(start,1,2) TO start
STORE $(ende,7,2) + $(ende,4,2) + $(ende,1,2) TO ende
```

Als nächste Aktion muß natürlich auch das Datum im Datenbankfeld DATUM in YY/MM/TT umgedreht werden, damit es sich vergleichen läßt. Das geht mit der FOR-Anweisung so:

```
FOR $(datum,7,2)+$(datum,4,2)+$(datum,1,2)>=start .AND.;
    $(datum,7,2)+$(datum,4,2)+$(datum,1,2)<= ende
```

Lassen Sie uns die beiden Routinen näher durchleuchten. Die beiden STORE-Befehle bringen die Jahreszahl an den Anfang und die Monatszahl in die Mitte des Datums in den beiden Variablen START und ENDE. Dann vergleicht die FOR-Anweisung das (ebenfalls auf JJ/MM/TT geänderte) Datum in der Datenbank mit dem geänderten Datum in den Variablen.

Das Datum sollte auch in dieser Weise verändert werden, wenn eine Datenbank sortiert wird. Wollen Sie beispielsweise die Datenbank SALES nach dem Datumfeld sortieren, benutzen Sie am besten diesen Befehl:

```
INDEX ON $(datum,7,2)+$(datum,4,2)+$(datum,1,2) TO
                                                saldatum
```

Um ein bestimmtes Datum zu finden oder alle Datensätze eines bestimmten Datums aufzulisten, benutzen Sie eine Routine wie diese:

```
************** Hole das Suchdatum (Format TT/MM/JJ)
STORE " " TO suchdat
§ 5,5 SAY "Geben Sie das Suchdatum ein";
      GET suchdat PICT "99/99/99"
READ

************** Datum auf das Format JJ/MM/TT bringen
STORE $(datum,7,2)+$(datum,4,2)+$(datum,1,2) TO;
                                               suchdat
```

```
********** Finde ersten Datensatz mit diesem Datum
FIND &suchdat

********** Drucke alle Datensätze mit diesem Datum
LIST WHILE $(datum,7,2)+$(datum,4,2)+$(datum,1,2);
                                        <= suchdat
```

Es gibt eine Alternative zu dieser Methode: Der Benutzer könnte das Datum gleich in der ungewöhnlichen JJ/MM/TT-Form eingeben. Das würde es uns ersparen, die gesamte Datumangabe umdrehen zu müssen. Allerdings sind die Leute es gewohnt, das Datum in der Form TT/MM/JJ zu schreiben. Damit ist es für den Benutzer einfacher, wenn wir die kleine Datum-Routine anwenden. Sie sollten sich nur daran erinnern, daß bei jeder Suche nach Daten innerhalb eines bestimmten Zeitraums das Datum umgewandelt werden muß.

Zusammenfassung

In diesem Kapitel haben wir den Teil des Inventarsystems entwickelt, mit dem Datensätze erfaßt und verwaltet werden können. Die Datenbank SALES.DBF speichert die Daten aller Verkäufe. Das Programm SMENU.CMD bietet die Möglichkeiten der Datenverwaltung in einem Menü an.

Das Programm POS.CMD ermöglicht die Eingabe von Verkaufsdaten. Eine Reihe von Programmiertechniken wurden in diesem Programm besprochen:

- Prüfen der Teilenummer auf Richtigkeit und automatisches Einfügen der Teilebezeichnung,
- die Möglichkeit, mehrere Einzelverkäufe auf einer Rechnung unterzubringen,
- Zwischensummen und die Gesamtsumme der Rechnung automatisch erstellen.

Während der Entwicklung des Programms SALREPS.CMD haben wir gesehen, wie Daten aufgelistet werden, die in einen bestimmten Zeitraum fallen. Wir haben auch gelernt, wie man mit Daten umgeht, deren Datum im Format TT/MM/JJ gespeichert ist.

Die dritte Option des Verkauf-Menüs heißt: Verkaufsdaten ändern. Diese Möglichkeit wird in Kapitel 13 unter „Aktualisieren der Daten" diskutiert. Im nächsten Kapitel werden wir den Teil des Inventarsystems entwickeln, der zum Verwalten der Lagereingänge notwendig ist.

Inventar Wareneingangsystem

12

Der Wareneingangsteil des Inventarsystems ermöglicht es dem Benutzer, Waren zu verzeichnen, die neu in das Lager kommen. Diese Daten werden in der Datenbank NEWSTOCK.DBF gespeichert, die wir schon früher mit dieser Struktur erstellt haben:

```
STRUKTUR DER DATEI: B:NEWSTOCK.DBF

FELDNR    NAME       TYP    WEITE    DEZ    Inhalt
001       TEIL:NR     C     005             Teilenummer
                                            (Schlüsselfeld)
002       MENGE       N     004             Erhaltene Menge
003       KOSTEN      N     009      002    Einkaufspreis
004       DATUM       C     008             Eingangsdatum
005       LIEFERANT   C     025             Lieferant
006       UEBERTR     L     001             Zur Hauptdatei
                                            geschickt?
```

Wir haben den Index für diese Datenbank aufgebaut, indem wir den Befehl INDEX ON teil:nr TO newstock gaben. An dieser Stelle kann der Benutzer die Option 3 aus dem Hauptmenü des Inventarsystems wählen. Daraufhin erscheint dieses Menü auf dem Bildschirm:

```
Menü Wareneingang

1.  Neue Waren aufnehmen
```

```
2.  Report der eingegangenen Waren
        drucken
3.  Daten verändern
4.  Zurück zum Hauptmenü
```

Sie werden sicherlich gleich bemerken, daß dieses Menü dem Menü Verkauf stark ähnelt. Beide Menüs erlauben dem Benutzer folgendes: Geschäftsvorgänge eingeben, Berichte drucken, Daten verändern oder die Rückkehr zum Hauptmenü des Inventarsystems.

Wareneingangsystem Programmstruktur

Lassen Sie uns die Programmstruktur des Wareneingangsystems näher betrachten. Es besteht aus den vier hier aufgeführten Programmen:

1. NMENU, das das Menü präsentiert.
2. NEWSTOCK, ermöglicht Neueingaben.
3. NEWREPS, druckt die Berichte.
4. NEWEDIT, ermöglicht Änderungen.

Die Abb. 12.1 zeigt die Software-Struktur für diesen Teil des Inventarsystems. In diesem Schaubild sehen Sie, wie jedes der vier Programme mit den anderen zusammenarbeitet.

Menü Wareneingang

Das Menü des Wareneingangsystems wird vom Programm NMENU.CMD zur Verfügung gestellt. Da es ein ganz einfaches Menüprogramm ist, zeige ich es in der Abb. 12.2. Wir werden die Programme NEWSTOCK.CMD und NEWREPS.CMD in diesem Kapitel entwickeln und das Programm NEWEDIT.CMD im nächsten.

Das Eingabeprogramm

Nachdem der Benutzer die Option 1 vom Menü des NEWSTOCK-Programms gewählt hat, wird der Bildschirm gelöscht und folgende Meldung erscheint:

```
Geben Sie die Daten der erhaltenen Waren ein
Teilenummer:_     :
```

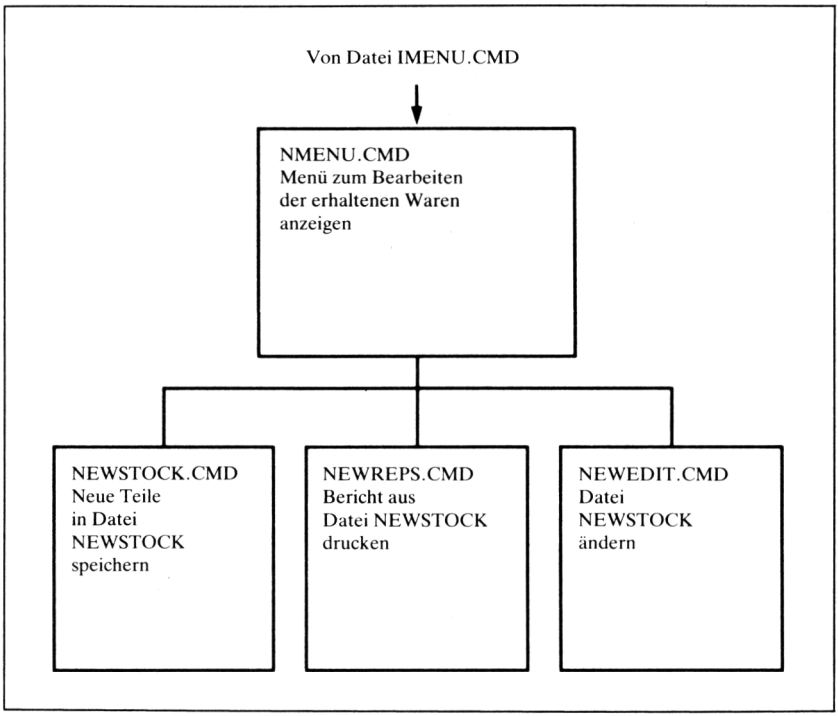


Abb. 12.1: Die Struktur des Wareneingangssystems

Gibt er hier eine ungültige Teilenummer ein, ertönt ein Warnton und auf dem Bildschirm erscheint die Nachricht:

```
Geben Sie die Daten der erhaltenen Waren ein
Teilenummer:X-999: ** Teil nicht vorhanden **
```

An dieser Stelle bekommt der Benutzer die Möglichkeit, einen neuen Versuch zu machen. Gibt er dann eine gültige Nummer ein, erscheinen auf dem Bildschirm die Bezeichnung des Teils, das Datum des Vorgangs und der Lieferant (der bei Bedarf auch geändert werden kann). Dann soll er die erhaltene Menge und den Einkaufspreis eingeben:

```
Geben Sie die Daten der erhaltenen Waren ein
Teilenummer:A-111:     Schneeschuhe
Menge :    :        Preis:      .  :
Datum :10/10/84     Lieferant:Eis & Schuh GmbH :
```

Dann gibt der Benutzer die erhaltene Menge und den Einkaufspreis ein, und eine neue Maske fragt nach dem nächsten Vorgang. Ist er mit der Arbeit fertig, tippt er einfach RETURN anstatt einer Teilenummer ein. Das bringt ihn zum Menü Wareneingang zurück.

```
****************************************** NMENU.CMD
**** Menü für den Abschnitt Wareneingang des Inventarsystems

******** Beginne Schleife für das Menü
STORE " " TO nwahl
DO WHILE nwahl # "4"
   ERASE
   TEXT
                Menü Wareneingang

           1.   Neue Waren aufnehmen

           2.   Report der eingegangenen Waren drucken

           3.   Daten verändern


           4.   Zurück zum Hauptmenü

    ENDTEXT
    § 14,15 SAY "Wahl eingeben (1-4) " GET nwahl PICTURE "9"
    READ

    ********** Verzweige zum entsprechenden Programm
    DO CASE

       CASE nwahl = "1"
            DO newstock

       CASE nwahl = "2"
            DO newreps

       CASE nwahl = "3"
            DO newedit
    ENDCASE

ENDDO (while nwahl # 4)

****************** Wenn fertig, Variablen löschen
****************** und zurück zum Hauptmenü
RELEASE nwahl
RETURN
```

Abb. 12.2: Das Programm NMENU.CMD

Der Pseudocode für das Programm NEWSTOCK steht in Abb. 12.3.

Das NEWSTOCK-Programm muß die Hauptdatei zum Überprüfen der Teilenummern benutzen und die Datei NEWSTOCK.DBF, um die einzelnen Vorgänge aufzuzeichnen. Die ersten Programmzeilen eröffnen die Datenbanken als erste und zweite Datenbank:

```
************************************* NEWSTOCK.CMD
***************** Aufzeichnung der erhaltenen Waren
SELECT PRIMARY
USE master INDEX master
SELECT SECONDARY
USE newstock INDEX newstock
SELECT PRIMARY
```

```
Bestimme die erste (MASTER) und die zweite
(NEWSTOCK) Datenbank

Beginne Schleife zum Speichern der erhaltenen Waren
    Bildschirm löschen
    Hole Teilenummer für nächsten Vorgang

    Wenn Teilenummer eingegeben wurde
    Finde Teilenummer in der Hauptdatei

    Wenn nicht zu finden, Warnung an Benutzer
    und weiteren Versuch ermöglichen

    Wenn Teilenummer existiert
    Zeige Bezeichnung des Teils
    Neuen Datensatz an NEWSTOCK.DBF anhängen
    Teilenummer eintragen
    Datum eintragen
    Lieferant eintragen
    Hole Menge und Einkaufspreis der neuen Waren

Weiter mit der Schleife (bis Benutzer aufhören möchte)

Lösche Variablen

Zurück zum Menü Wareneingang
```

Abb. 12.3: Pseudocode für das Programm NEWSTOCK.CMD

Dann muß das Programm mit einer Schleife zum Aufnehmen der neuen Vorgänge beginnen, was mit folgenden Zeilen geschieht:

```
********** Beginne Schleife für eingehende Waren
STORE "X" TO teilnr
DO WHILE teilnr # " "
   ERASE
   STORE " " TO teilnr
   § 2,2 SAY " Geben Sie die Daten der erhaltenen;
                                          Waren ein"
   § 4,4 SAY "Teilenummer " GET teilnr PICT "!!!!!"
   READ
```

Wünscht der Benutzer, an dieser Stelle nicht aufzuhören, versucht das Programm mit dem Befehl FIND die entsprechende Teilenummer in der Hauptdatei zu finden:

```
IF teilnr # " "
     FIND &teilnr
```

Als nächstes muß das Programm auf die Eingaben des Benutzers reagieren. Sollte er eine ungültige Teilenummer eingegeben haben, bekommt er die Möglichkeit, es noch einmal zu versuchen. Das erste CASE in der DO CASE-Anweisung kümmert sich um ungültige Teilenummern:

```
DO CASE

     CASE # = 0
       ********* Wenn Teil nicht gefunden, Benutzer;
       ********* warnen
       § 4,25 SAY "** Teil nicht vorhanden **"
       § CHR(7)
```

Wenn der Benutzer eine gültige Teilenummer eingibt, soll die dazugehörende Bezeichnung auf dem Bildschirm erscheinen, ein neuer Datensatz an die Datei NEWSTOCK.DBF (SECONDARY) angehängt und die vorhandenen Daten zu dem Teil sollten angezeigt werden. Die Felder TEIL:NR, DATUM und LIEFERANT werden automatisch vom Programm mit bereits vorhandenen Daten ausgefüllt, und der Benutzer muß deshalb nur noch die Menge (MENGE) und den Einkaufspreis (KOSTEN) eingeben. Diese Zeilen bewerkstelligen das:

```
CASE # >0
     ****** Wenn gefunden, die restlichen Daten holen
     § 4,25 SAY titel
```

```
SELECT SECONDARY
APPEND BLANK
REPLACE teil:nr WITH teilnr
REPLACE S.datum WITH t:datum
REPLACE S.lieferant WITH P.lieferant
§ 6,2 SAY "Menge " GET S.menge
§ 6,22 SAY "Preis " GET S.kosten;
       PICT "99999.99"
§ 8,2 SAY "Datum" GET datum PICT "99/99/99"
§ 8,22 SAY "Lieferant " GET S.lieferant
READ
SELECT PRIMARY
```

Beachten Sie die Feldnamen, die in den beiden Datenbanken MASTER (PRIMARY) und NEWSTOCK (SECONDARY) gleich sind: MENGE, KOSTEN UND LIEFERANT. Diese werden in Zukunft entweder mit einem P. für PRIMARY davor oder einem S. für SECONDARY davor aufgerufen (z. B. P.lieferant, S.kosten). Das stellt sicher, daß vom Programm die richtigen Felder aus der richtigen Datenbank verwendet werden.

Der Rest des Programms schließt einfach nur noch die DO CASE-, IF- und DO WHILE-Schleifen. Wählt der Benutzer den Programmabbruch, wird die Variable TEILNR gelöscht, die beiden Datenbanken werden geschlossen, und das Programm kehrt zum NMENU zurück:

```
ENDCASE

ENDIF (teilnr#" ")
ENDDO (while teilnr#" ")

RELEASE teilnr
SELECT SECONDARY
USE
SELECT PRIMARY
USE
RETURN
```

Die Abb. 12.4 zeigt das gesamte NEWSTOCK.CMD-Programm.

Wareneingang-Berichte

Das Wareneingangsystem erlaubt es dem Benutzer, den Lagerbestand nach Teilenummern oder dem Eingang innerhalb eines bestimmten Zeitraums aufzulisten. Diese Berichte können bei Unstimmigkeiten mit

```
************************************ NEWSTOCK.CMD
****************************** Aufzeichnung der erhaltenen Waren
SELECT PRIMARY
USE master INDEX master
SELECT SECONDARY
USE newstock INDEX newstock
SELECT PRIMARY

***************** Beginne Schleife für eingehende Waren
STORE "X" TO teilnr
DO WHILE teilnr # " "
   ERASE
   STORE "      " TO teilnr
   § 2,2 SAY " Geben Sie die Daten der erhaltenen Waren ein"
   § 4,4 SAY "Teilenummer " GET teilnr PICT "!!!!!"
   READ

   IF teilnr # " "
     FIND &teilnr

     DO CASE

        CASE # = 0
          ********* Wenn Teil nicht gefunden, Benutzer warnen
          § 4,25 SAY "** Teil nicht vorhanden **"
          § CHR(7)

        CASE # >0
          *********** Wenn gefunden, die restlichen Daten holen
          § 4,25 SAY titel

          SELECT SECONDARY
          APPEND BLANK
          REPLACE teil:nr WITH teilnr
          REPLACE S.datum WITH t:datum
          REPLACE S.lieferant WITH P.lieferant
          § 6,2 SAY "Menge " GET S.menge
          § 6,22 SAY "Preis " GET S.kosten;
                        PICT "99999.99"
          § 8,2 SAY "Datum " GET datum PICT "99/99/99"
          § 8,22 SAY "Lieferant " GET S.lieferant
          READ
          SELECT PRIMARY

     ENDCASE

   ENDIF (teilnr#" ")
ENDDO (while teilnr#" ")
```

Abb. 12.4: Das Programm NEWSTOCK.CMD

```
RELEASE teilnr
SELECT SECONDARY
USE
SELECT PRIMARY
USE
RETURN
```

Abb. 12.4: Das Programm NEWSTOCK.CMD (Forts.)

einem Lieferanten oder Unterschieden zwischen dem Computerbericht und dem tatsächlich vorhandenen Lagerbestand helfen. Zusätzlich können diese Daten einen Überblick über die Umschlagsgeschwindigkeit eines bestimmten Teils in einem bestimmten Zeitabschnitt liefern und dadurch bei der Festlegung neuer Nachbestellgrenzen hilfreich sein.

Sobald der Benutzer Berichte zu drucken wünscht, erscheint ein Menü mit diesen Optionen:

```
Berichte Wareneingang

1. Nach Teilenummer

2. Nach Datum

3. Zurück zum Menü Wareneingang
```

Wählt der Benutzer die Option 1, wird er gefragt:

```
Nach welcher Teilenummer suchen?:_    :
```

Er gibt dann eine Nummer ein, und alle Vorgänge für Teile mit dieser Nummer werden in einem Bericht aufgelistet, der wie dieser hier aussieht:

```
Seite Nr: 00001
01/01/84
                        Eingegangene Teile

Teil        Menge     EK-Preis        Datum          Lieferant
Nr.
A-111       15        10.00           28/02/84       Eis & Schuh GmbH
A-111       15        10.00           15/04/84       Eis & Schuh GmbH
A-111       15        10.00           10/05/84       Eis & Schuh GmbH
A-111       15        10.00           10/06/84       Eis & Schuh GmbH
A-111       15        10.00           10/09/84       Eis & Schuh GmbH
A-111       15        10.00           10/10/84       Eis & Schuh GmbH
```

Der Benutzer kann sich auch alle Vorgänge ansehen, die zwischen zwei bestimmten Terminen liegen. Wenn er die Option 2 wählt, erscheint auf dem Bildschirm:

```
Anfangsdatum:  /  /  :
  Enddatum     :  /  /  :
```

Werden hier die entsprechenden Termine eingegeben, zeigt das Programm alle Vorgänge innerhalb dieses Zeitraums, einschließlich der angegebenen Daten, in solch einem Bericht:

```
Seite Nr: 00001      Zeitraum vom 01/10/84 bis 30/10/84

                          Eingegangene Teile

Teil Nr.   Menge   EK-Preis      Datum        Lieferant
A-111      15      10.00        01/10/84     Eis & Schuh GmbH
Z-999      10      10.00        15/10/84     Bearskin Canada
A-112      12      24.00        20/10/84     Tierfreund KG
B-123      23      11.55        10/10/84     Buchshop
```

Beide Berichte benutzen eine REPORT-Datei mit dem Namen NEWSTOCK.FRM. Diese wurde erstellt mit den Befehlen

```
USE newstock
REPORT FORM newstock
```

die direkt hinter das dBASE-Prompt eingegeben wurden. Dann muß das Report-Formular nur noch so ausgefüllt werden:

```
ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE,
W=PAGE WIDTH M=1,W=55
PAGE HEADING? (Y/N) Y
ENTER PAGE HEADING: Eingegangene Teile
DOUBLE SPACE REPORT? (Y/N) N
ARE TOTALS REQUIRED? (Y/N) N
COL WIDTH,CONTENTS
001   5,TEIL:NR
ENTER HEADING: Teil;Nr
002   4,MENGE
```

```
ENTER HEADING: Stk.
003   9,KOSTEN
ENTER HEADING: EK-Preis
004   8,DATUM
ENTER HEADING: Datum
005   25,LIEFERANT
ENTER HEADING: Lieferant
006
```

Die Logik des Programms NEWREPS.CMD ist größtenteils identisch mit der des SALREPS.CMD-Programms aus dem letzten Kapitel.

Wie die meisten Menüprogramme, beginnt auch dieses mit dem Eröffnen einer Schleife, dem Zeigen eines Menüs und dem Warten auf die Benutzereingabe. Die entsprechende Antwort wird in der Variablen REPWAHL gespeichert. Die folgenden Zeilen zeigen das Menü und fragen nach der gewählten Option:

```
************************************ NEWREPS.CMD
************************* Berichte aus der Datei
************************* NEWSTOCK drucken
STORE " " TO repwahl,jn
DO WHILE repwahl # "3"
   ERASE
   TEXT

         Berichte Wareneingang

     1.  Nach Teilenummer

     2.  Nach Datum

     3. Zurück zum Menü Wareneingang
   ENDTEXT
   § 8,12 SAY "Ihre Wahl (1-3)";
          GET repwahl PICT "9"
   READ
```

Wenn der Benutzer jetzt nicht aufhören will, sollte das Programm nach dem Drucker fragen. In den folgenden Zeilen passiert das:

```
*************************** Nach dem Drucker fragen
ERASE
STORE " " TO jn,drucker
```

```
IF repwahl # "3"
   § 5,5 SAY "Daten drucken?";
         GET jn PICT "!"
   READ
ENDIF (repwahl #3)
```

Bejaht der Benutzer die Frage nach dem Ausdruck der Daten, speichert das Programm die Wörter TO PRINT in die Variable DRUCKER. Verneint er die Frage, bleibt die Variable leer. Die untenstehenden Zeilen eröffnen die DRUCKER-Variable, die später als Makro mit dem REPORT FORM-Befehl benutzt werden wird, um festzustellen, ob ein Report auf den Bildschirm oder zum Drucker geschickt werden soll:

```
*************** Makro für den Drucker erstellen
IF jn = "J"
   STORE "TO PRINT" TO drucker
ELSE
   STORE " " TO drucker
ENDIF (jn=j)
```

Dann muß das Programm den Bericht anzeigen. Sollte der Bericht auf der Basis einer Teilenummer erstellt werden, fragt das Programm nach der zu suchenden Nummer. Um die Verarbeitungsgeschwindigkeit beim Suchen zu erhöhen, benutzt das Programm die Index-Datei NEWSTOCK.NDX (indiziert nach Teilenummern) und arbeitet dann mit den Befehlen FIND und WHILE. Das erste CASE in den hier abgedruckten Zeilen zeigt den Bericht nach Teilenummer:

```
***************** Drucke den entsprechenden Report
ERASE
DO CASE

   ******** Fall 1: Suche nach Teilenummer
   CASE repwahl = "1"
   USE newstock INDEX newstock
   STORE " " TO stocksuch
   § 5,5 SAY "Nach welcher Teilenummer suchen?";
         GET stocksuch PICT "!!!!!"
   READ
   ERASE
   FIND &stocksuch
   REPORT FORM newstock WHILE;
       teil:nr=stocksuch &drucker
```

Wünscht der Benutzer alle Vorgänge anzuzeigen, die zwischen zwei Terminen liegen, fragt das Programm zuerst nach dem entsprechenden Zeitraum. Dann, wie im früher erstellten Programm SALREPS.CMD, werden die Variablen START und ENDE als Makro in dem REPORT FORM-Befehl benutzt, um den Bericht anzuzeigen. Die CASE-Anweisung hier zeigt den Bericht für eine bestimmte Zeitspanne:

```
******** Fall 2: Suchen nach Terminen
CASE repwahl = "2"
  USE newstock
  STORE "        " TO start,ende
  § 5,3 SAY "Anfangsdatum :";
        GET start PICT "99/99/99"
  § 7,3 SAY "Enddatum     :";
        GET ende PICT "99/99/99"
  READ
  ERASE

  ********** Makro für die Terminsuche aufbauen
  STORE "FOR datum >= start .AND.;
         datum <= ende" TO makro

  ********** Überschrift
  SET HEAD TO Zeitraum vom &start bis &ende

  ********** Report drucken
  REPORT FORM newstock &makro &drucker
  SET HEADING TO

ENDCASE
```

Schließlich hält das Programm die Bildschirmausgabe an, wenn der Benutzer N gedrückt hat, und setzt dann entweder die Schleife fort oder löscht die Speichervariablen und kehrt zum Programm NMENU.CMD zurück. Diese Aufage wird von den folgenden Zeilen bewältigt:

```
****** Wenn nicht gedruckt wird, Bildschirm anhalten
IF jn # "J" .AND. repwahl # "3"
   ?
   ?
   WAIT
ENDIF (jn#j)

ENDDO (while repwahl #3)
```

```
********************* Wenn fertig, Variablen löschen
********************* und zurück zu NMENU.CMD
USE
RELEASE repwahl,jn,drucker,start,ende,makro,;
                                        stocksuch
RETURN
```

In der Abb. 12.5 sehen Sie das gesamte Programm NEWREPS.CMD.

```
************************************ NEWREPS.CMD
************************ Berichte aus der Datei NEWSTOCK drucken
STORE " " TO repwahl,jn
DO WHILE repwahl # "3"
   ERASE
   TEXT

         Berichte Wareneingang

      1. Nach Teilenummer

      2. Nach Datum


      3. Zurück zum Menü Wareneingang
    ENDTEXT
    § 10,12 SAY "Ihre Wahl (1-3)";
          GET repwahl PICT "9"
    READ
    ************************** Nach dem Drucker fragen
    ERASE
    STORE " " TO jn,drucker
    IF repwahl # "3"
    § 5,5 SAY "Daten drucken?";
         GET jn PICT "!"
    READ
    ENDIF (repwahl #3)

    ************** Makro für den Drucker erstellen
    IF jn = "J"
       STORE "TO PRINT" TO drucker
    ELSE
       STORE " " TO drucker
    ENDIF (jn=j)
```

Abb. 12.5: Das Programm NEWREPS.CMD

```
************************ Drucke den entsprechenden Report
ERASE
DO CASE

   ******** Fall 1: Suche nach Teilenummer
   CASE repwahl = "1"
     USE newstock INDEX newstock
     STORE "      " TO stocksuch
     § 5,5 SAY "Nach welcher Teilenummer suchen?";
           GET stocksuch PICT "!!!!!"
     READ
     ERASE
     FIND &stocksuch
     REPORT FORM newstock WHILE;
            teil:nr=stocksuch &drucker

   ******** Fall 2: Suchen nach Terminen
   CASE repwahl = "2"
     USE newstock
     STORE "         " TO start,ende
     § 5,3 SAY "Anfangsdatum :";
           GET start PICT "99/99/99"
     § 7,3 SAY "Enddatum     :";
           GET ende PICT "99/99/99"
     READ
     ERASE

     ********** Makro für die Terminsuche aufbauen
     STORE "FOR datum >= start .AND.;
            datum <= ende" TO makro

     ********** Überschrift
     SET HEAD TO Zeitraum vom &start bis &ende

     ********** Report drucken
     REPORT FORM newstock &makro &drucker
     SET HEADING TO

ENDCASE

********** Wenn nicht gedruckt wird, Bildschirm anhalten
IF jn  #"J" .AND. repwahl # "3"
   ?
   ?
   WAIT
ENDIF (jn#j)
```

Abb. 12.5: Das Programm NEWREPS.CMD (Forts.)

```
ENDDO (while repwahl #3)

********************* Wenn fertig, Variablen löschen und zurück
********************* zu NMENU.CMD
USE
RELEASE repwahl,jn,drucker,start,ende,makro,stocksuch
RETURN
```

Abb. 12.5: Das Programm NEWREPS.CMD (Forts.)

Zusammenfassung

Jetzt haben wir die drei Hauptbestandteile des Inventarsystems entwikkelt. Allerdings müssen wir noch den etwas kritischen letzten Schritt tun. Wir müssen ein Programm schreiben, das Daten aus der SALES-Datenbank und der NEWSTOCK-Datenbank lesen kann und damit den Status in der Hauptdatei für die entsprechenden Teile aktualisiert. Zusätzlich müssen wir auch noch Programme erstellen, mit denen der Benutzer die SALES- und NEWSTOCK-Dateien verändern kann, nachdem die Aktualisierung schon durchgeführt wurde. Wir werden das im nächsten Kapitel schon schaffen.

Aktualisierung des Inventarsystems

13

Nachdem wir nun bereits Programme zum Bearbeiten der Hauptdatei des Inventarsystems und der Verkaufs- sowie Wareneingangsdateien geschrieben haben, müssen wir noch ein Programm entwickeln, mit dem die Hauptadtei aktualisiert werden kann. Dieses Programm muß die Mengen der SALES-Datei von denen der Hauptdatei abziehen, da diese Teile verkauft wurden. Dann muß es die Mengen aus der NEWSTOCK-Datei zu dem Bestand in der Hauptdatei dazuzählen, da diese jetzt den aktuellen Bestand ausmachen.

Weiterhin muß das Programm die neu eingelieferten Mengen von den Bestellmengen der Hauptdatei abziehen, da diese ja jetzt geliefert worden sind. Schließlich ersetzt das Programm noch den Einkaufspreis in der MASTER-Hauptdatei durch den der Wareneingangsdatei, da dies der aktuellere Preis ist. Weiterhin ersetzt das Programm die Einträge im Datum-Feld der Hauptdatei durch die der beiden anderen Datenbanken, um festzuhalten, wann die letzte Aktualisierung stattgefunden hat.

Aktualisierung der Hauptdatei

Der Benutzer aktualisiert die Hauptdatei einfach dadurch, daß er die Option 4 im Hauptmenü wählt, die wir im Programm MMENU.CMD mit angeboten haben. Das Programm für die Aktualisierung heißt UPDATER.CMD. Der entsprechende Pseudocode steht in der Abb. 13.1.

```
Bildschirm löschen
Zeige eine Nachricht, damit der Benutzer weiß, daß der Computer
arbeitet
Benutze die Datei SALES, indiziert nach Teilenummern

Kopiere alle nichtaktualisierten Datensätze in TEMP

Stelle sicher, daß Daten in TEMP stehen
Wenn Daten vorhanden ...
   Benutze Datei TEMP
   Mache alle MENGE-Felder negativ, um zu subtrahieren

   Benutze Hauptdatei und Index für Aktualisierung
   Aktualisiere mit den TEMP-Daten durch Addieren der
   MENGEn und Übernahme der DATUM-Einträge

   Benutze die Datei SALES
   Und schreibe (T)rue in die Felder UEBERTR

Aktualisiere die Hauptdatei mit Daten aus NEWSTOCK
Kopiere alle nichtaktualisierten Daten in TEMP

Stelle sicher, daß Daten in TEMP stehen
Wenn Daten vorhanden ...
   Benutze Hauptdatei und Index zum Aktualisieren

   Aktualisiere von der Datei TEMP aus, addiere die
   Mengen und ersetze KOSTEN und DATUM

   Mache alle UEBERTR-Felder (T)rue in NEWSTOCK.DBF

   Aktualisiere in der MASTER-Datei das Feld BESTELL mit
      dem negativen Wert des Feldes MENGE in der NEWSTOCK-
      Datei
   Mache TEMP zur PRIMARY-Datenbank
   Mache MASTER plus Index-Datei zur SECONDARY-Daten-
   bank
   Beginne Schleife durch die TEMP-Datenbank
   Für jeden Datensatz in TEMP
```

Abb. 13.1: Pseudocode für UPDATER.CMD

Finde identische Teilenummer in MASTER.DBF
Subtrahiere neuen Lagerbestand von dem Feld
BESTELL der Hauptdatei

Lösche Variablen
Schließe Datenbanken
RETURN zu MMENU.CMD

Abb. 13.1: Pseudocode für UPDATER.CMD (Forts.)

Das UPDATER-Programm löscht den Bildschirm und gibt während eines möglicherweise ziemlich langen Prozesses die Meldung aus: „Aktualisiere Hauptdatei mit Daten der SALES-Datenbank ...". Danach benutzt das Programm die Datei SALES mit dem entsprechenden Index und kopiert alle noch nicht aktualisierten Datensätze in eine Zwischendatei (TEMP), wie es diese Zeilen zeigen:

```
********************************** UPDATER.CMD
**** Aktualisieren der Hauptdatei mit
**** Daten von SALES und NEWSTOCK
ERASE
? "Aktualisiere Hauptdatei mit Daten der SALES-
                                    Datenbank..."

******** Benutze die SALES-Datenbank, indiziert
******** nach Teilenummern
USE sales INDEX sales

******** Kopiere alle nicht aktualisierten
******** Datensätze in die
******** Zwischendatei TEMP
COPY TO TEMP FOR .NOT. uebertr
```

Zu diesem Zeitpunkt stehen alle Daten, die für die Aktualisierung der Hauptdatei gebraucht werden, in der Zwischendatei TEMP. Allerdings stehen die Mengen in positiven Ziffern in der Zwischendatei. Da es aber Verkäufe waren, müssen sie subtrahiert werden. Der dBASE Befehl UPDATE kann keine Werte subtrahieren, weshalb wir alle MENGE-Felder in der Zwischendatei negativ machen müssen, bevor wir die Aktualisierung durchführen können. Bevor wir aber irgend etwas machen, soll das Programm nachsehen, ob in der Zwischendatei über-

haupt Daten stehen. Die folgenden Zeilen überprüfen die Zwischendatei und machen alle MENGE-Felder negativ, wenn Daten vorhanden sind:

```
********************** Sicherstellen, daß Daten in
**********************TEMP stehen
USE TEMP
GO BOTT
IF #>0

    ***** Mache alle MENGE-Felder negativ
    REPLACE ALL menge WITH -1*menge
```

Als nächstes wird die MASTER-Datei mit der Index-Datei MASTER.NDX in Gebrauch genommen, da diese Datei ja aktualisiert werden soll:

```
********** Benutze die Hauptdatei zum Aktualisieren
USE master INDEX master
```

Dann wird aktualisiert, die Mengen werden addiert (die in TEMP negativ sind) und das Datum wird ersetzt:

```
********** Aktualisieren aus der SALES-Zwischendatei
UPDATE ON teil:nr FROM TEMP ADD menge REPLACE datum
```

Dann müssen die Datensätze in der Verkaufsdatei markiert werden, damit sie nicht fälschlicherweise in Zukunft noch einmal zur Aktualisierung benutzt werden. Das Feld UEBERTR dient dabei als Merker, der den logischen Wert wahr (T) besitzt, wenn dieser Datensatz bereits übertragen wurde (zur Aktualisierung der Hauptdatei):

```
************** Zurück zur Original SALES-Datenbank
USE sales INDEX sales

************** und mache alle UEBERTR-Felder wahr
************** (T)rue
REPLACE NOUPDATE ALL uebertr WITH T
```

Jetzt können wir die IF-Schleife zum Überprüfen der TEMP-Datei auf Inhalt schließen:

```
ENDIF (# > 0)
```

Das Programm muß die gleiche Aktualisierung durch die NEWSTOCK-Datei durchführen. Die Datei und der Index werden benutzt, und alle

Datensätze, die bis jetzt noch nicht übertragen wurden, werden in eine Zwischendatei kopiert:

```
******** Jetzt aktualisiere die MASTER-Datei aus
******** der NEWSTOCK-Datei
?"Aktualisiere MASTER aus NEWSTOCK ..."
USE newstock INDEX newstock

************** Kopiere nichtaktualisierte
************** Datensätze nach TEMP
COPY TO TEMP FOR .NOT. uebertr
```

Um Zeit zu sparen und Fehler zu vermeiden, prüft das Programm wieder, ob auch Daten in TEMP stehen, bevor aktualisiert wird:

```
********** Prüfe die Anzahl der Datensätze in TEMP
USE TEMP
GO BOTT
IF * >0
```

Dann werden die Haupt- und die Indexdatei wieder in Gebrauch genommen:

```
*********** Benutze Hauptdatei zum Aktualisieren
USE master INDEX master
```

Das Programm aktualisiert dann mit den Daten der Zwischendatei, addiert die Mengen und ersetzt die Felder KOSTEN und DATUM:

```
***** Aktualisieren mit der NEWSTOCK-Zwischendatei
UPDATE ON teil:nr FROM TEMP ADD menge REPLACE
                                        datum,kosten
```

Dann müssen alle übertragenen Felder in der NEWSTOCK-Datei auf „wahr" gesetzt werden, damit sie nicht fälschlicherweise in Zukunft noch einmal übertragen werden:

```
******** Alle UEBERTR-Felder in NEWSTOCK wahr
******** (T)rue machen
USE newstock INDEX newstock
REPLACE NOUPDATE ALL uebertr WITH T
```

Dann muß in der Hauptdatei das Feld BESTELL aktualisiert werden, indem alle eingegangenen Waren von dem dort stehenden BESTELL-

Wert subtrahiert werden. Da die Feldnamen unterschiedlich sind (BESTELL und MENGE), können wir den UPDATE-Befehl nicht verwenden. Die beiden Datenbanken müssen gleichzeitig geöffnet sein, weshalb wir sie als PRIMARY und SECONDARY öffnen:

```
****** Aktualisiere das Feld BESTELL der Hauptdatei
****** mit dem negativen Wert des Feldes MENGE
****** aus der NEWSTOCK-Datei
SELECT PRIMARY
USE temp
SELECT SECONDARY
USE master INDEX master
```

Jetzt müssen wir eine Schleife durch jeden Datensatz der TEMP-Datei beginnen:

```
SELECT PRIMARY
DO WHILE .NOT. EOF
```

Für jeden Datensatz in der TEMP-Datei muß das Programm die entsprechende Teilenummer in der MASTER-Datei finden und dann die MENGE der TEMP-Datei (P.MENGE) vom BESTELL-Feld der Hauptdatei abziehen. Diese Zeilen sorgen dafür:

```
STORE P.teil:nr TO suchen
SELECT SECONDARY
FIND &suchen
REPLACE bestell WITH bestell-P.menge
```

Dann muß die Schleife zum nächsten Datensatz in der TEMP-Datei (PRIMARY) weitergehen.

```
SELECT PRIMARY
  SKIP
ENDDO (while not eof)
```

Um den Aktualisierungsprozeß abzuschließen, können wir die IF-Anweisung beenden, die darüber entschied, ob Daten in TEMP standen oder nicht:

```
ENDIF (#>0)
```

Alles, was noch zu tun bleibt, ist das Löschen der Variablen, das Schließen der Datenbanken und danach die Rückkehr zum Programm

MMENU.CMD. Hier sind die korrekten Zeilen für diese Prozedur:

```
****************** Zurück zu MMENU.CMD
RELEASE suchen
SELE SECO
USE
SELE PRIM
USE
RETURN
```

In Abb. 13.2 können Sie sich das Programm als Ganzes betrachten.

```
****************************************** UPDATER.CMD
**** Aktualisieren der Hauptdatei mit
**** Daten von SALES und NEWSTOCK
ERASE
? "Aktualisiere Hauptdatei mit Daten der SALES-Datenbank ..."

******** Benutze die SALES-Datenbank, indiziert nach Teilenummern
USE sales INDEX sales

******** Kopiere alle nicht aktualisierten Datensätze in die
******** Zwischendatei TEMP
COPY TO TEMP FOR .NOT. uebertr

********************** Sicherstellen, daß Daten in TEMP stehen
USE TEMP
GO BOTT
IF #>0

   ***** Mache alle MENGE-Felder negativ
   REPLACE ALL menge WITH -1*menge
   USE master INDEX master

   ********** Aktualisieren aus der SALES-Zwischendatei
   UPDATE ON teil:nr FROM TEMP ADD menge REPLACE datum

   ************** Zurück zur Original SALES-Datenbank
   USE sales INDEX sales

   ************** und mache alle UEBERTR-Felder wahr (T)rue
   REPLACE NOUPDATE ALL uebertr WITH T

ENDIF (# > 0)

*************** Aktualisiere MASTER aus NEWSTOCK
? "Aktualisiere Hauptdatei mit Daten der NEWSTOCK-Datenbank ..."
```

Abb. 13.2: Das Programm UPDATER.CMD

```
USE newstock INDEX newstock

********* Kopiere nichtaktualisierte Datensätze nach TEMP
COPY TO TEMP FOR .NOT. uebertr

************** Prüfe die Anzahl der Datensätze in TEMP
USE TEMP
GO BOTT
IF # >0

   ************ Benutze Hauptdatei zum Aktualisieren
   USE master INDEX master

   ************** Aktualisieren mit der NEWSTOCK-Zwischendatei
   UPDATE ON teil:nr FROM TEMP ADD menge REPLACE datum,kosten

   ******** Alle UEBERTR-Felder in NEWSTOCK wahr (T)rue machen
   USE newstock INDEX newstock
   REPLACE NOUPDATE ALL uebertr WITH T

   ******** Aktualisiere das Feld BESTELL der Hauptdatei
   ******** mit dem negativen Wert des Feldes MENGE aus der
   ******** NEWSTOCK-Datei
   SELECT PRIMARY
   USE temp
   SELECT SECONDARY
   USE master INDEX master
   SELECT PRIMARY
   DO WHILE .NOT. EOF

      STORE P.teil:nr TO suchen
      SELECT SECONDARY
      FIND &suchen
      REPLACE bestell WITH bestell-P.menge
      SELECT PRIMARY
      SKIP
   ENDDO (while .not. eof)

ENDIF (#>0)

**************** Zurück zu MMENU.CMD
RELEASE suchen
SELE SECO
USE
SELE PRIM
USE
RETURN
```

Abb. 13.2: Das Programm UPDATER.CMD (Forts.)

Ändern in der Datei Verkauf

Jetzt haben wir ein kleines Problem. Stellen Sie sich vor, der Benutzer ändert einen Datensatz in der SALES-Datei, nachdem diese bereits aktualisiert wurde. Beispielsweise könnte es ja sein, daß er 100 Teile mit der Nummer A-111 als verkauft eingibt, hinterher aber feststellt, es waren nur 10. Es ist für ihn kein Problem, die SALES-Datei zu ändern, aber in der Hauptdatei fehlen 90 Teile. Genausogut könnte es passieren, daß er den Verkauf eines Teiles mit der Nummer A-111 eingibt, hinterher aber feststellt, daß die Nummer A-112 lauten müßte. Wieder kann er den Fehler zwar in der SALES-Datei ändern, aber die Hauptdatei enthält falsche Werte.

Es gibt für dieses Problem eine einfache Lösung: Schreiben Sie eine Routine, die es dem Benutzer gestattet, in der SALES-Datei jede beliebige Änderung vorzunehmen. Dann macht dieses Programm einfach die gleiche Berichtigung auch in der Hauptdatei. Die Programmlogik für solch eine Routine sieht so aus:

1. Erlaube dem Benutzer, die Daten in SALES zu ändern.
2. Wird eine Teilenummer geändert, addiere die Menge zur alten Teilenummer und subtrahiere das von der neuen.
3. Hat der Benutzer eine MENGE in der SALES-Datei geändert, ermittle die Differenz zwischen der alten und der neuen MENGE und subtrahiere diese MENGE von der MENGE in der Hauptdatei.

Sollte der Benutzer das Datum ändern, können wir das auch in der Hauptdatei ändern, was aber nicht nötig ist. Das Datum wird weiterhin den Zeitpunkt der letzten Aktualisierung zeigen, statt den Zeitpunkt der Änderung.

Wählt der Benutzer die Option zum Ändern der SALES-Datei, wird der Bildschirm gelöscht und folgende Meldung erscheint:

```
Geben Sie die zu ändernde Rechnungsnummer ein (0
                                        wenn keine):_
```

Er gibt daraufhin eine Nummer ein, und wenn es mehrere Teilenummern zu dieser Rechnung geben sollte, zeigt das Programm die einzelnen Vorgänge wie folgt:

```
00099    A-111     10   10.50   ACS   J. Schmidt   12/12/84
00100    Z-999      3   12.99   ACS   J. Schmidt   12/12/84
00100    B-232     14   11.99   ACS   J. Schmidt   12/12/84

   Welche Zeile? (links stehende Ziffer):_
```

Sobald der Benutzer einen Vorgang zum Ändern oder Löschen angegeben hat, werden dessen Daten so auf dem Bildschirm angezeigt:

```
Verkäufe ändern ...

Rechnungsnummer:  123:          Teilenummer :A-111:

Verkäufer  :ACS       :         Kunde  :J. Schmidt      :

Menge :    10:   VK-Preis :    10.50:   Datum :12/12/84:

Datensatz löschen? (J/N) : :
```

In dieser Maske kann der Benutzer den Cursor beliebig bewegen, um Einträge zu verändern. Oder er kann einfach J auf die Frage „Datensatz löschen? (J/N)" eingeben, um einen Eintrag zu löschen. Das Programm wird dann alles weitere übernehmen, um die Hauptdatei zu verändern, wenn nötig, und dem Benutzer weitere Änderungsmöglichkeiten in der SALES-Datei einräumen.

Das dazu notwendige Programm heißt SALEDIT.CMD, und dessen Pseudocode finden Sie in der Abb. 13.3.

Jetzt können wir das Programm SALEDIT entwickeln. Lassen Sie uns mit der Eröffnung der Variablen zum Zählen der gelöschten Datensätze (NRLOESCH) anfangen. Danach erstellen wir noch die zum Suchen nach Rechnungsnummern benötigte Variable (SUCH) in den ersten Programmzeilen:

```
************************************ SALEDIT.CMD
************ Datei SALES editieren und Master
************ aktualisieren

******************** Variablen eröffnen
STORE 0 TO nrloesch
STORE 1 TO such
```

Weiter müssen wir dem Programm sagen, daß es die Datenbank SALES mit der Index-Datei SALES verwenden und den Benutzer nach der zu ändernden Rechnungsnummer fragen soll:

```
******** Hole Teilenummer für den Änderungsdatensatz
DO WHILE such <> 0
   USE sales INDEX sales
   ERASE
   INPUT "Geben Sie die zu ändernde Rechnungsnummer;
                                ein (0 wenn keine)";
       TO such
```

```
Eröffne Variablen
Beginne Schleife für Rechnungsnummer

    Benutze die Datei SALES mit dem Index SALES
    Hole Rechnungsnummer mit zu ändernden Daten
    Wenn Benutzer nicht aufhören will ...

    Zähle die Datensätze mit dieser Rechnungsnummer

    Wenn Rechnung nicht gefunden, Benutzer warnen und weiteren
        Versuch ermöglichen

    Wenn mehrere Datensätze mit dieser Nummer gefunden ...
        Alle Vorgänge mit dieser Nummer anzeigen
        Datensatznummer vom Benutzer holen

    Finde Datensatz und speichere Originaldaten in Variable

    Zeige Daten in Änderungsmaske und erlaube Änderungen

    Wenn Änderung fertig, korrigiere MASTER-Datei, wenn
        Datensatz schon übertragen wurde und Teilenummer oder
        Menge geändert wurde

    Wenn Datensatz bereits übertragen, aber jetzt gelöscht
        Datensatz zum Löschen markieren
        Finde Teilenummer in MASTER-Datei
        Verändere Lagerbestand entsprechend der Menge in der
        SALES-Datei

    Wenn Datensatz bereits übertragen und Benutzer ändert
    Teilenummer...
        Finde alte Teilenummer in der Hauptdatei
        Alte Menge in MASTER speichern
        Finde die neue Teilenummer in MASTER
        Subtrahiere die Menge von MASTER

    Wenn Datensatz übertragen und Menge geändert
        Berechne die Differenz zwischen alter und neuer Menge
        Finde Teilenummer in der Hauptdatei
        Subtrahiere Differenz von MASTER-Datei
```

Abb. 13.3: Pseudocode für das SALEDIT.CMD-Programm

Weiter mit Schleife für mehrere Eintragungen

Wenn Datensätze gelöscht SALES-Datei PACKen

Lösche Variablen

RETURN to SMENU.CMD

Abb. 13.3: Pseudocode für das SALEDIT.CMD-Programm (Forts.)

Wünscht der Benutzer nicht aufzuhören, soll das Programm die in Frage kommenden Datensätze zählen und diesen Wert in der Speichervariablen WIEVIELE ablegen:

```
*************** Wenn nicht Ende, Datensätze zählen
IF such > 0
   *** Zähle Datensätze mit dieser Rechnungsnummer
   COUNT FOR rechng:nr=such TO wieviele
```

Gibt es keine Vorgänge mit dieser Nummer, soll auf dem Bildschirm eine Warnung erscheinen. Diese Zeilen sorgen dafür:

```
DO CASE

      **** Wenn keine Nummer gefunden, Warnung
      **** ausgeben
      CASE wieviele = 0
           ? "    ** Rechnungsnummer nicht
                          vorhanden **"
           ? CHR(7)
```

Gibt es mehrere Datensätze mit dieser Rechnungsnummer, soll jeder Vorgang auf dem Bildschirm angezeigt und nach dem gewünschten Datensatz gefragt werden. Hier sind die richtigen Zeilen für diese Aufgabe:

```
******* Wenn Rechnungsnummer gefunden, weitermachen
CASE wieviele > 0

****** Wenn mehrere Verkäufe diese Rechnungsnummer
****** haben, hole mehr Informationen ...
IF wieviele > 1
   ERASE
```

```
LIST FOR rechng:nr = such teil:nr,menge;
   preis,verkaeufer,kunde,datum
?
INPUT "Welche Zeile? (linksstehende Ziffer)";
   TO satznr
GOTO satznr
```

Gibt es nur einen Datensatz mit der erfragten Nummer, kann das Programm direkt zu dem Datensatz gehen:

```
********** Andernfalls gehe zu dem Datensatz
ELSE
   LOCATE FOR rechng:nr = such
   STORE # TO satznr
ENDIF (wieviele >1)
```

Dann speichert das Programm die Originaldaten für TEIL:NR und MENGE in die Variablen ALT:TEIL und ALT:MENGE, damit es später feststellen kann, ob diese Felder geändert wurden oder nicht:

```
******** Speichere Originalfeldinhalte in
******** Variablen ab
STORE teil:nr TO alt:teil
STORE menge TO alt:menge
```

Jetzt zeigt das Programm eine Maske, damit der Benutzer die Daten verändern kann:

```
******************** Zeige Änderungsmaske und
******************** ermögliche Änderungen
ERASE
STORE " " TO loesch
§ 1,1 SAY "Verkaufseinträge ändern ..."
§ 3,1 SAY "Rechnungsnummer " GET rechng:nr
§ 3,33 SAY "Teilenummer " GET teil:nr;
     PICT "!!!!!"
§ 5,1 SAY "Verkäufer " GET verkaeufer
§ 5,40 SAY"Kunde " GET kunde
§ 7,1 SAY "Menge " GET menge
§ 7,20 SAY "Verkaufspreis " GET preis
§ 7,45 SAY "Datum ";
       GET datum PICT "99/99/99"
§ 9,2 SAY "Datensatz löschen? (J/N)";
      GET loesch PICT "!"
READ
```

Sobald der Benutzer einen Datensatz verändert hat, sieht das Programm nach, ob die Hauptdatei aktualisiert werden muß. Löscht der Benutzer einen Datensatz, holt das Programm die Menge aus der SALES-Datenbank wieder zurück in die MASTER-Datenbank und erhöht den Wert der Variablen NRLOESCH, in der die als gelöscht markierten Datensätze gezählt werden:

```
********** Wenn Ändern fertig, korrigiere MASTER-
********** Datei, falls notwendig
DO CASE

   ****** Wenn Verkäufe gelöscht wurden, bringe
   ****** diese Menge
   ****** wieder zurück in die MASTER-Datenbank
   CASE uebertr .AND. loesch = "J"
      DELETE
      STORE nrloesch +1 TO nrloesch
      USE master INDEX master
      FIND &alt:teil
      REPL menge WITH menge+alt:menge
```

Wenn der Benutzer die Teilenummer geändert hat, addiert das Programm die Menge aus der SALES-Datei zurück in die alte Teilenummer der Hauptdatei (bevor es sie tatsächlich subtrahiert). Dann subtrahiert es die verkaufte Menge von der neuen Teilenummer in der MASTER-Datei:

```
******** Wenn Teilenummer geändert, addiere MENGE
******** zu der
******** alten Teilenummer und subtrahiere von der
******** neuen Teilenummer
    CASE uebertr .AND. teil:nr # alt:teil
      STORE menge TO neu:menge
      STORE teil:nr TO neu:teilnr
      USE master INDEX master
      FIND &alt:teil
      REPLACE menge WITH menge+alt:menge
      FIND &neu:teil
      REPLACE menge WITH menge-neu:menge
```

Wurde nur die Menge der verkauften Teile geändert, ändert das Programm die Menge entsprechend in der Hauptdatei:

```
********* Wenn nur die Menge geändert,
********* Menge in Hauptdatei korrigieren
CASE uebertr .AND. menge # alt:menge
```

```
      STORE menge-alt:menge TO diff
      USE master INDEX master
      FIND &alt:teil
      REPLACE menge WITH menge - diff

   ENDCASE
   ENDCASE
   ENDIF (such>0)

   ENDDO (while such > 0)
```

Schließlich löscht das Programm, wenn nötig, die zum Löschen markierten Datensätze mit dem Befehl PACK:

```
   ************* Wenn Datensätze als gelöscht markiert,
   ************* die Datenbank SALES PACKen
   IF nrloesch >0
      ERASE
      ? "Die Daten aus SALES werden endgültig
      gelöscht ..."
      USE sales INDEX sales
      PACK
   ENDIF
```

Dann löscht es die Variablen und kehrt zum Programm SMENU.CMD zurück:

```
   ************************** Lösche Speichervariablen
   RELEASE alt:menge,alt:teil,neu:teil,nrloesch
   RELEASE such,wieviele,diff,loesch
   USE
   RETURN
```

Die Abb. 13.4 zeigt das gesamte Programm SALEDIT.CMD.

```
************************************ SALEDIT.CMD
************ Datei SALES editieren und MASTER aktualisieren

******************** Variablen eröffnen
STORE 0 TO nrloesch
STORE 1 TO such
*************** Hole Teilenummer für den Änderungsdatensatz
DO WHILE such <> 0
   USE sales INDEX sales
```

Abb. 13.4: Das Programm SALEDIT.CMD

```
ERASE
INPUT "Geben Sie die zu ändernde Rechnungsnummer ein (0 wenn
                                                    keine)" ;
TO such

****************** Wenn nicht Ende, Datensätze zählen
IF such > 0
   ****** Zähle Datensätze mit dieser Rechnungsnummer
   COUNT FOR rechng:nr=such TO wieviele
   DO CASE

      **** Wenn keine Nummer gefunden, Warnung ausgeben
      CASE wieviele = 0
         ? "    ** Nummer nicht vorhanden **"
         ? chr(7)
         ******* Wenn Rechnungsnummer gefunden, weitermachen
      CASE wieviele > 0

         ********* Wenn mehrere Verkäufe diese Rechnungsnummer
         ********* haben, hole mehr Informationen ...
         IF wieviele > 1
            ERASE
            LIST FOR rechng:nr = such teil:nr,menge;
            preis,verkaeufer,kunde,datum
            ?
            INPUT "Welche Zeile? (linksstehende Ziffer)" ;
            TO satznr
            GOTO satznr

            ************ Andernfalls gehe zu dem Datensatz
         ELSE
            LOCATE FOR rechng:nr = such
            STORE  #TO satznr
         ENDIF (wieviele >1)

         ******** Speichere Originalfeldinhalte in
         ******** Variablen ab
         STORE teil:nr TO alt:teil
         STORE menge TO alt:menge

         ******************** Zeige Änderungsmaske und
         ******************** ermögliche Änderungen
         ERASE
         STORE " " TO loesch
         § 1,1 SAY "Verkaufseinträge ändern ..."
         § 3,1 SAY "Rechnungsnummer " GET rechng:nr
         § 3,33 SAY "Teilenummer " GET teil:nr;
```

Abb. 13.4: Das Programm SALEDIT.CMD (Forts.)

```
          PICTURE "!!!!!"
          § 5,1 SAY "Verkäufer " GET verkaeufer
          § 5,40 SAY "Kunde " GET kunde
          § 7,1 SAY "Menge " GET menge
          § 7,20 SAY "Verkaufspreis " GET preis
          § 7,45 SAY "Datum " ;
          GET datum PICTURE "99/99/99"
          § 9,2 SAY "Datensatz löschen? (J/N)" ;
          GET loesch PICTURE "!"
          READ

          ************ Wenn Ändern fertig, korrigiere
          ************ Masterdatei falls notwendig
          DO CASE

               ****** Wenn Verkäufe gelöscht wurden, bringe
               ****** diese Menge wieder zurück in die
               ****** MASTER-Datenbank
             CASE uebertr .AND. loesch = "J"
               DELETE
               STORE nrloesch +1 TO nrloesch
               USE master INDEX master
               FIND &alt:teil
               REPLACE menge WITH menge+alt:menge

               ******** Wenn Teilenummer geändert, addiere
               ******** Menge zu  der alten Teilenummer und
               ******** subtrahiere von der neuen Teilenummer
             CASE uebertr .AND. teil:nr  #alt:teil
               STORE menge TO neu:menge
               STORE teil:nr TO neu:teilnr
               USE master INDEX master
               FIND &alt:teil
               REPLACE menge WITH menge+alt:menge
               FIND &neu:teil
               REPLACE menge WITH menge-neu:menge

               ********* Wenn nur die MENGE geändert,
               ********* MENGE in Hauptdatei korrigieren
             CASE uebertr .AND. menge  #alt:menge
               STORE menge-alt:menge TO diff
               USE master INDEX master
               FIND &alt:teil
               REPLACE menge WITH menge - diff

        ENDCASE
     ENDCASE
```

Abb. 13.4: Das Programm SALEDIT.CMD (Forts.)

```
   ENDIF (such>0)

ENDDO (WHILE such >0)

********************** Wenn Datensätze als gelöscht markiert,
********************** die Datenbank SALES PACKen
IF nrloesch >0
   ERASE
   ? "Die Daten aus SALES werden endgültig gelöscht ..."
   USE sales INDEX sales
   PACK
ENDIF nrloesch >0

***************************** Lösche Speichervariablen
RELEASE alt:menge,alt:teil,neu:teil,nrloesch
RELEASE such,wieviele,diff,loesch
USE
RETURN
```

Abb. 13.4: Das Programm SALEDIT.CMD (Forts.)

Änderungen in der Datei Wareneingang

Die Prozedur zum Ändern der Wareneingangsdatei und entsprechenden Änderungen in der Hauptdatei ist dem Programm SALEDIT.CMD sehr ähnlich. Allerdings muß sich das Änderungsprogramm für die NEWSTOCK-Datei um die Felder KOSTEN und BESTELL kümmern, da die Hauptdatei mit Daten aus der Datenbank NEWSTOCK direkt versorgt wird. Ein anderer Unterschied ist, daß wir Änderungen nur auf derBasis der Teilenummer machen können, da es keine Rechnungsnummer wie in der Verkaufsdatei gibt.

Das Programm NEWEDIT.CMD verarbeitet Änderungen in der NEWSTOCK-Datenbank. Am Anfang sieht es dem SALEDIT-Programm sehr ähnlich, mit der Ausnahme, daß es den Benutzer nach einer Teilenummer statt nach einer Rechnungsnummer fragt. Sie sehen die Eröffnungszeilen hier:

```
************************************ NEWEDIT.CMD
************ Datei NEWSTOCK editieren und MASTER
************ aktualisieren

******************** Variablen eröffnen
STORE 0 TO nrloesch
```

```
STORE "1" TO such
****** Hole Teilenummer für den Änderungsdatensatz
DO WHILE such <> "0"
   USE newstock INDEX newstock
   ERASE
   ACCEPT "Geben Sie die zu ändernde Teilenummer
                              ein (0 wenn keine)" ;
   TO such
   STORE !(such) TO such
```

Dann zählt das Programm, wie viele Datensätze in der NEWSTOCK-Datei die gesuchte Teilenummer besitzen:

```
************ Wenn nicht Ende, Datensätze zählen
   IF such # "0"
      ****** Zähle Datensätze mit dieser
      ******Rechnungsnummer
      FIND &such
      COUNT WHILE teil:nr= !(such) TO wieviele
```

Existiert die gesuchte Teilenummer nicht, wird der Benutzer gewarnt:

```
      DO CASE

         **** Wenn keine Nummer gefunden,
         **** Warnung ausgeben
      CASE wieviele = 0
         ? "    ** Teilenummer nicht vorhanden
                                              **"
         ? chr(7)
```

Gibt es in der Datei mehrere Datensätze mit dieser Teilenummer, soll der Benutzer die richtige anhand der Datensatznummer angeben:

```
******* Wenn Teilenummer gefunden,
******* weitermachen
       CASE wieviele > 0

          ********* Wenn mehrere Wareneingänge
          ********* diese Teilenummer haben,
          ********* hole mehr Informationen ...
          IF wieviele > 1
             ERASE
             FIND &such
```

```
        LIST WHILE teil:nr = !(such) teil:nr,
        menge; kosten,datum,lieferant
        ?
        INPUT "Welche Zeile? (linksstehende
                                    Ziffer)" ;
        TO satznr
      GOTO satznr
```

Besitzt nur ein Datensatz in der Datei NEWSTOCK die erfragte Teilenummer, wird dieser Eintrag gefunden und seine Satznummer in der Variablen SATZNR gespeichert:

```
********** Andernfalls gehe zu dem Datensatz
      ELSE
        FIND &such
        STORE # TO satznr
      ENDIF (wieviele >1)
```

Weiterhin speichert das Programm einige der benötigten Originaldaten für den zu ändernden Datensatz in Speichervariablen. Dies versetzt wieder das Programm in die Lage zu entscheiden, ob die Hauptdatei nach der Änderung aktualisiert werden muß oder nicht:

```
    ******** Speichere Originalfeldinhalte in
      ******** Variablen ab
      STORE teil:nr TO alt:teil
      STORE menge TO alt:menge
      STORE kosten TO alt:kosten
```

Dann bringt das Programm den Datensatz aus der NEWSTOCK-Datenbank auf den Bildschirm, damit der Benutzer etwas ändern oder den Satz löschen kann:

```
******************** Zeige Änderungsmaske
******************** und ermögliche  Änderungen
ERASE
STORE " " TO loesch
§ 1,1 SAY "Wareneingänge ändern ..."
§ 3,1 SAY "Teilenummer " GET teil:nr
PICTURE "!!!!!"
§ 5,1 SAY "Menge " GET menge
§ 5,20 SAY "Einkaufspreis " GET kosten
```

```
§ 7,1 SAY "Datum " GET datum PICT "99/99/99"
§ 7,15 SAY "Lieferant " GET lieferant
§ 9,2 SAY "Datensatz löschen? (J/N)" ;
GET loesch PICTURE "!"
            READ
```

Jetzt muß sich das Programm entscheiden, welche Änderungen es in der Hauptdatei durchführen soll. Wurden Preise in der NEWSTOCK-Datei geändert, müssen sie auch in der Hauptdatei geändert werden:

```
************ Wenn Ändern fertig,
************ korrigiere MASTER-Datei,
************ falls notwendig

************ Bearbeite zuerst Änderungen
************ beim Preis falls notwendig
IF kosten # alt:kosten
    STORE teil:nr TO neu:teil
    STORE kosten TO neu:kosten
    USE master INDEX master
    FIND &neu:teil
    REPLACE kosten WITH neu:kosten
    USE newstock INDEX newstock
    GOTO satznr
    ENDIF (kosten # alt:kosten)
```

Sollte der Datensatz nach der Aktualisierung gelöscht worden sein, hat der dazugehörende Datensatz in der Hauptdatei zu viele Teile in seinem MENGE-Feld und zu wenig Teile in seinem Feld für Bestellungen (BESTELL). Die erste CASE-Anweisung kümmert sich darum:

```
DO CASE

      ****** Wenn Wareneingänge gelöscht wurden,
      ****** subtrahiere diese MENGE von der
      ****** Hauptdatei und addiere sie
      ****** wieder zurück in das Feld BESTELL
    CASE uebertr .AND. loesch = "J"
      DELETE
      STORE nrloesch +1 TO nrloesch
      USE master INDEX master
      FIND &alt:teil
      REPLACE menge WITH menge-alt:menge
      REPLACE bestell WITH bestell+alt:menge
```

Wenn der Benutzer die Teilenummer eines bereits aktualisierten Datensatzes in der NEWSTOCK-Datei geändert hat, stehen in dem entsprechenden Feld der Hauptdatei zu viele Teile für die alte Teilenummer und zu wenig Teile für die neue. Zusätzlich enthält das Feld für Bestellungen zu wenig Teile für die alte Teilenummer und zu viele für die neue. Die zweite CASE-Anweisung regeneriert die Hauptdatei in dieser Situation:

```
******** Wenn Teilenummer geändert, subtrahiere MENGE
******** von der alten Teilenummer und addiere
******** zu der neuen. Mache das Gegenteil für
******** das BESTELL-Feld
CASE uebertr .AND. teil:nr # alt:teil
  STORE menge TO neu:menge
  STORE teil:nr TO neu:teilnr
  USE master INDEX master
  FIND &alt:teil
  REPLACE menge WITH menge-alt:menge
  REPLACE bestell WITH bestell+alt:menge
  FIND &neu:teil
  REPLACE menge WITH menge+neu:menge
  REPLACE bestell WITH bestell-neu:menge
```

Hat der Benutzer allerdings nur die Menge geändert, dann müssen die Felder MENGE und BESTELL mit der Differenz zwischen der ursprünglich bei der Aktualisierung verwendeten Menge und der neuen, korrekten Menge in der Hauptdatei korrigiert werden. Die dritte CASE-Anweisung zeigt, wie das geht:

```
        ********* Wenn nur die MENGE geändert,
        ********* MENGE in Hauptdatei korrigieren
        CASE uebertr .AND. menge # alt:menge
          STORE menge-alt:menge TO diff
          USE master INDEX master
          FIND &alt:teil
          REPLACE menge WITH menge + diff
          REPLACE bestell WITH bestell - diff
        ENDCASE
    ENDCASE
 ENDIF (such#0)

ENDDO (WHILE such # 0)
```

Die oben gezeigten Zeilen schließen auch die DO CASE-, IF- und DO WHILE-Schleifen.

Sobald der Benutzer zum Beenden des Programms bereit ist, werden zuerst alle als gelöscht markierten Datensätze mit dem Befehl PACK endgültig beseitigt:

```
********** Wenn Datensätze als gelöscht markiert,
********** die Datenbank NEWSTOCK PACKen
IF nrloesch >0
   ERASE
   ? "Die Daten aus NEWSTOCK werden endgültig
                                  gelöscht ..."
   USE newstock INDEX newstock
   PACK
ENDIF (nrloesch >0 )
```

Danach kann das Programm die Variablen löschen und zum Programm NMENU.CMD zurückkehren, das es auch aufgerufen hatte:

```
************************** Lösche Speichervariablen
RELEASE alt:menge,alt:teil,neu:teil,nrloesch,
                                      alt:kosten
RELEASE such,wieviele,diff,loesch,neu:kosten,
                                       neu:menge
USE
RETURN
```

Das vollständige Programms sehen Sie in Abb. 13.5.

```
********************************** NEWEDIT.CMD
********** Datei NEWSTOCK editieren und MASTER aktualisieren

******************** Variablen eröffnen
STORE 0 TO nrloesch

STORE "1" TO such

*************** Hole Teilenummer für den Änderungsdatensatz
DO WHILE such <> "0"
   USE newstock INDEX newstock
   ERASE
   ACCEPT "Geben Sie die zu ändernde Teilenummer ein (0 wenn
                                                  keine)" ;
   TO such
   STORE !(such) TO such
```

Abb. 13.5: Das Programm NEWEDIT.CMD

```
******************** Wenn nicht Ende, Datensätze zählen
IF such # "0"
   ****** Zähle Datensätze mit dieser Rechnungsnummer
   FIND &such
   COUNT WHILE teil:nr= !(such) TO wieviele
   DO CASE

      **** Wenn keine Nummer gefunden, Warnung ausgeben
      CASE wieviele = 0
         ? "** Nummer nicht vorhanden **"
         ? chr(7)

      ******* Wenn Teilenummer gefunden, weitermachen
      CASE wieviele > 0

         ********* Wenn mehrere Wareneingänge diese
         ********* Teilenummer haben,
         ********* hole mehr Informationen ...
         IF wieviele > 1
            ERASE
            FIND &such
            LIST WHILE teil:nr = !(such) teil:nr,menge;
            kosten,datum,lieferant
            ?
            INPUT "Welche Zeile? (linksstehende Ziffer)" ;
            TO satznr
            GOTO satznr

            ************ Andernfalls gehe zu dem Datensatz
          ELSE
            FIND &such
            STORE  #TO satznr
          ENDIF (wieviele >1)

          ******** Speichere Originalfeldinhalte in
          ******** Variablen ab
          STORE teil:nr TO alt:teil
          STORE menge TO alt:menge
          STORE kosten TO alt:kosten

          ********************* Zeige Änderungsmaske und
          ********************* ermögliche Änderungen
          ERASE
          STORE " " TO loesch
          § 1,1 SAY "Wareneingänge ändern ..."
          § 3,1 SAY "Teilenummer " GET teil:nr;
          PICTURE "!!!!!"
```

Abb. 13.5: Das Programm NEWEDIT.CMD (Forts.)

```
§ 5,1 SAY "Menge " GET menge
§ 5,20 SAY "Einkaufspreis " GET kosten
§ 7,1 SAY "Datum " GET datum PICT "99/99/99"
§ 7,15 SAY "Lieferant " GET lieferant
§ 9,2 SAY "Datensatz löschen? (J/N)" ;
GET loesch PICTURE "!"
READ

********** Wenn Ändern fertig, korrigiere
********** MASTER-Datei, falls notwendig

********** Bearbeite zuerst Änderungen beim Preis,
********** falls notwendig
IF kosten # alt:kosten
   STORE teil:nr TO neu:teil
   STORE kosten TO neu:kosten
   USE master INDEX master
   FIND &neu:teil
   REPLACE kosten WITH neu:kosten
   USE newstock INDEX newstock
   GOTO satznr
ENDIF (kosten # alt:kosten)
DO CASE

     ****** Wenn Wareneingänge gelöscht wurden,
     ****** subtrahiere diese MENGE von der
     ****** Hauptdatei und addiere sie
     ****** wieder zurück in das Feld BESTELL
   CASE uebertr .AND. loesch = "J"
     DELETE
     STORE nrloesch +1 TO nrloesch
     USE master INDEX master
     FIND &alt:teil
     REPLACE menge WITH menge-alt:menge
     REPLACE bestell WITH bestell+alt:menge

     ******** Wenn Teilenummer geändert, subtrahiere
     ******** Menge von der alten Teilenummer und
     ******** addiere zu der neuen. Mache das
     ******** Gegenteil für das das BESTELL-Feld
   CASE uebertr .AND. teil:nr # alt:teil
     STORE menge TO neu:menge
     STORE teil:nr TO neu:teilnr
     USE master INDEX master
     FIND &alt:teil
     REPLACE menge WITH menge-alt:menge
     REPLACE bestell WITH bestell+alt:menge
```

Abb. 13.5: Das Programm NEWEDIT.CMD (Forts.)

```
                FIND &neu:teil
                REPLACE menge WITH menge+neu:menge
                REPLACE bestell WITH bestell-neu:menge

                ********* Wenn nur die MENGE geändert,
                ********* MENGE in Hauptdatei korrigieren
              CASE uebertr .AND. menge # alt:menge
                STORE menge-alt:menge TO diff
                USE master INDEX master
                FIND &alt:teil
                REPLACE menge WITH menge + diff
                REPLACE bestell WITH bestell - diff
            ENDCASE
      ENDCASE
   ENDIF (such#0)

ENDDO (WHILE such # 0)

********************** Wenn Datensätze als gelöscht markiert,
********************** die Datenbank NEWSTOCK PACKen
IF nrloesch >0
   ERASE
   ? "Die Daten aus NEWSTOCK werden endgültig gelöscht ..."
   USE newstock INDEX newstock
   PACK
ENDIF (nrloesch >0 )

****************************** Lösche Speichervariablen
RELEASE alt:menge,alt:teil,neu:teil,nrloesch,alt:kosten
RELEASE such,wieviele,diff,loesch,neu:kosten,neu:menge
USE
RETURN
```

Abb. 13.5: Das Programm NEWEDIT.CMD (Forts.)

Zusammenfassung

Das Inventarsystem, das wir jetzt erstellt haben, ist eine komplexe Aufgabe für einen Programmierer, aber leicht zu bedienen für Anfänger. Das Programm ist sehr leistungsfähig bei der Speicherung von Verkaufs- und Wareneingangsdaten und bietet zusätzlich nützliche Berichte. Es erlaubt dem Lagerverwalter auch, die Hauptdatei mit den Daten der Verkaufs- und Wareneingangsdatei zu aktualisieren, indem er einfach eine Option des Hauptmenüs auswählt. Entdeckt der Benutzer einen Fehler in einer der beiden Dateien, kann er den Vorgang einfach korrigieren. Das System übernimmt die Aufgabe, die entsprechenden Korrekturen automatisch auch in der Hauptdatei durchzuführen.

Im nächsten Kapitel werden wir uns mit dem Programmentwurf und verschiedenen Techniken befassen, die zum Erstellen eines Buchhaltungssystems nötig sind. In vielen Bereichen sind Buchhaltungssysteme und Inventarsysteme gleich: Es gibt Dateien für einzelne Geschäftsvorgänge und eine Hauptdatei, die den Status der verschiedenen Zahlungen enthält. Viele der von Ihnen bereits in früheren Kapiteln gelernten Techniken, können auch bei der Entwicklung des Buchhaltungssystems verwendet werden.

Ein Buchhaltungssystem

14

Unser nächstes Projekt wird ein anwenderfreundliches Buchhaltungssystem sein. Wie die vorhergehenden Systeme auch wird die Buchhaltung menügesteuert und für den ungeübten Benutzer leicht zu bedienen sein. Hier einige der neuen Techniken, die wir während der Entwicklung kennenlernen werden:

1. eine Möglichkeit für den Benutzer, das System für sich zu installieren,
2. eine chronologische Fortschreibung der Buchungen,
3. Bilanzen berechnen,
4. Ziffern in Wörter umsetzen, um Schecks zu schreiben,
5. mehrere Felder aktualisieren.

Das Buchhaltungssystem wird außerdem viele der Techniken benötigen, die wir bei der Adressenverwaltung und dem Inventarsystem kennengelernt haben.

Die Ziele des Buchhaltungssystems

Das Buchhaltungssystem soll es dem Benutzer ermöglichen, eine Kontenliste aufzustellen, mit der er seine Buchhaltung machen kann. Das System verlangt, daß jedes Konto eine eigene Nummer und Bezeichnung hat.

Dann kann der Benutzer seine Ein- und Ausgaben während des Jahres festhalten, indem er einfach Zahlungsvorgänge eingibt. Das System akzeptiert nur gültige Zahlungsnummern, wobei es gleichzeitig die Korrektheit der Daten noch überprüft. Weiterhin erhält der Benutzer die Möglichkeit Zahlungsvorgänge zu korrigieren, um Fehler zu berichtigen, wobei gleichzeitig eine Chronologie des gesamten Jahres erstellt wird.

Zu jeder Zeit innerhalb des Jahres, kann der Benutzer die Zahlungsvorgänge wieder durchsehen, entweder für den aktuellen Monat oder für einen beliebigen Zeitraum. Er kann die Kontenübersicht auch ausdrukken, wobei monatliche, vierteljährliche und jährliche Bilanzen der Konten genauso möglich sind wie Gesamtsummen. Das System druckt Schecks automatisch aus, führt ein Scheckregister und erlaubt dem Benutzer, die Bilanz des Scheckkontos schnell einmal nachzusehen.

Als Zugabe erlaubt das Buchhaltungssystem auch, Unterkonten für Zahlungen einzurichten. Damit läßt sich feststellen, wie sich verdiente oder bezahlte Beträge zusammensetzen. Zum Beispiel: Der Benutzer schreibt einen Scheck über 100 DM aus, um einen Kredit abzutragen. Obwohl er nur einen Scheck ausstellt, möchte er vielleicht aber doch festhalten, wie hoch der Anteil der Zinsen und der Abtragung an dieser Summe ist. In solch einem Fall kann er drei Konten so eingeben:

Zahlungsnummer	Beschreibung
500.00	Geschäftskredit
500.10	Schuldabtragung Geschäftskredit
500.20	Zinsen für Geschäftskredit

Wenn er dann den Betrag von 100 Mark für das Konto 500.00 eingibt, 80 Mark für das Konto 500.10 und 20 Mark für das Konto 500.20, wird nur ein Scheck für das Konto 500.00 gedruckt und der Betrag im Kontofeld 500.00 ist der einzige, der bei der Erstellung einer Bilanz berücksichtigt wird. Die Beträge in den Unterkonten werden zwar auch abgespeichert, aber nur zu seiner Information. Diese kann wieder sehr hilfreich sein, wenn es an die Steuererklärung geht, weil die Zinsen für den Geschäftskredit absetzbare Kosten darstellen.

Entwurf der Datenbanken des Buchhaltungssystems

Das Buchhaltungssystem besteht aus vier Datenbanken:

1. GENINFO.DBF: Datei für generelle Informationen. Enthält grundsätzliche Information für das System, einschließlich des Firmennamens, des letzen Aktualisierungsdatums, der Kontenübersicht und der zuletzt benutzten Scheckummer.

2. COA.DBF: Kontenübersicht. Hier stehen die Kontennummern, die Bezeichnungen und die Bilanz für monatliche, vierteljährliche und jährliche Zahlungen.

3. TRANS.DBF: Vorgänge-Datei. Hier steht jede einzelne Zahlung, ob eingenommen oder ausgegeben, die der Benutzer eingibt. Für jeden Vorgang kümmert sie sich um die Kontennummer, den Grund der Ausgabe oder die Einnahmequelle, den Betrag und das Datum. Wird ein Scheck benötigt, stehen hier die Schecknummer und an wen er geschickt wurde (oder werden soll). Außerdem gibt es hier noch mehrere Felder zum Überwachen der Aktualisierungen.

4. REGISTER.DBF: Das Scheckregister. Hier wird fortlaufend notiert, welche Zahlungen eingegegangen oder wofür Schecks geschrieben wurden. Weiterhin stehen hier die Schecknummer, die Beschreibung, an wen gezahlt, der Betrag und das Datum.

Die Beziehungen zwischen den Datenbanken sehen Sie in der Abb. 14.1.

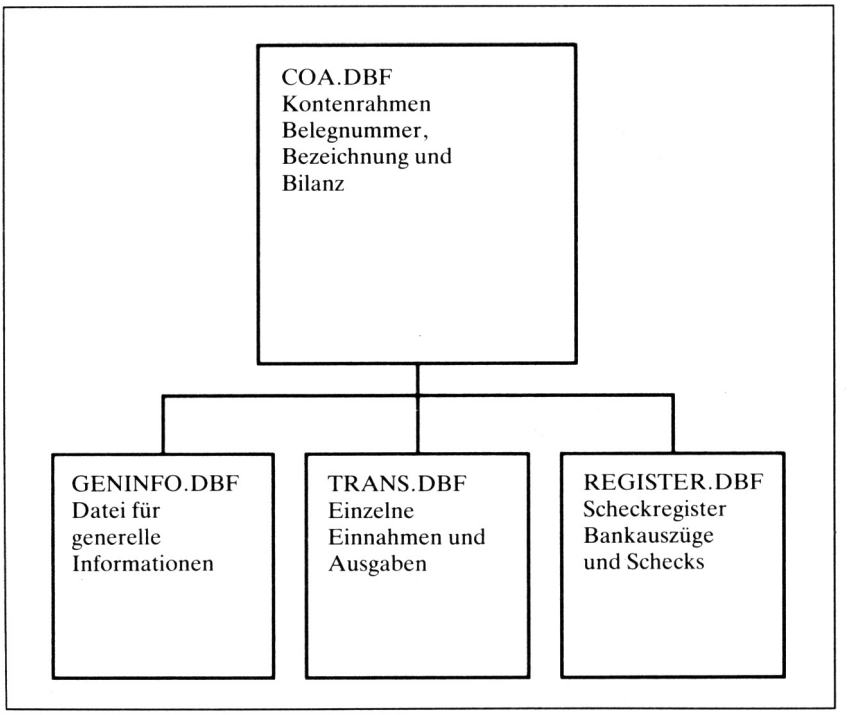


Abb. 14.1: Die Datenbankstruktur des Buchhaltungssystems

Die Datenbank GENINFO ist unabhängig. Die Datei COA aktualisiert ihre Bilanzen nach Daten aus der TRANS-Datei. Das Scheckregister bekommt auch seine Daten von TRANS.DBF, wird von dort aber nicht aktualisiert, sondern steht in einem Abhängigkeitsverhältnis, das wir während der Programmentwicklung noch kennenlernen werden.

Die Datei GENINFO enthält nur einen Datensatz mit dieser Struktur:

```
STRUKTUR FÜR DATEI: GENINFO.DBF

Feld  Name       Typ   Weite  Dez     Bezeichnung
001   FIRMA       C     025           Firmenname
002   LE:AKTU     C     008           Datum der letzten
                                      Aktualisierung
003   L:SCHECK    N     004           Letzte Schecknummer
```

Achtung: Ich habe zwar die Beschreibung hinter jedes Feld geschrieben, aber wenn Sie die Datenbank erstellen, geben Sie nur FELD, NAME, TYP, WEITE und DEZ ein.

Die TRANS-Datenbank ist für einzelne Einnahmen und Ausgaben da. Ihre Struktur sieht so aus:

```
STRUKTUR FÜR DATEI: TRANS.DBF
Feld  Name       Typ   Weite  Dez     Bezeichnung
001   KONTO       N     006    002     Zahlungsnummer
                                       (Schlüsselfeld)
002   GRUND       C     020           Grund oder Quelle d. Zahlng.
003   BETRAG      N     012    002     Betrag der Zahlung
004   TYP         N     001           Typ: 1=Scheck, 2=anderes
005   SCHECK:NR   N     004           Schecknummer
006   AN:WEN      C     020           an wen Scheck gezahlt wurde
007   DATUM       C     008           Zahlungsdatum
008   UEBERTR     L     001           An die Datei COA übertragen?
009   GESCHR      L     001           Scheck schon geschrieben?
010   REGISTER    L     001           Zum REGISTER schicken?
011   MERKER      C     001           markiert Unterkonten
```

Beachten Sie bitte, daß diese Datenbank eine Reihe von logischen Feldern enthält. Diese werden als Markierung benutzt, um anzuzeigen, welche Datensätze bereits in die Kontenübersicht übertragen wurden (UEBERTR), für die die Schecks bereits geschrieben wurden (GESCHR) und welche bereits in das Scheckregister (REGISTER) übertragen wurden. Das Feld MERKER markiert Datensätze, die Unterkonten sind.

Da der Benutzer Zahlungen entweder in der Reihenfolge der Kontennummern oder nach Terminen sortiert anzeigen kann, ist die Datei

TRANS nach dem Feld KONTO indiziert und eine zweite Index-Datei hält die Einträge des Feldes DATUM. Nachdem Sie die TRANS-Datei erstellt haben, benutzen Sie diese Befehle, um die Index-Dateien zu erstellen:

```
USE trans
INDEX ON konto TO konto
INDEX ON $(datum,7,2)+$(datum,1,6)) TO datum
```

Da es sein kann, daß ein Geschäftsjahr des Benutzers über zwei Jahre geht (zum Beispiel vom 1. Juni bis zum 31. Mai), wird die Jahreszahl im Index vorangestellt. Das Feld KONTO (die Kontennummer) ist sowohl in der COA-Datenbank (Kontenübersicht) wie auch in der TRANS-Datenbank (Vorgänge) vorhanden und wird als Schlüsselfeld für Aktualisierungen benutzt.

Die Kontenübersicht wird in der Datenbank COA.DBF gespeichert. Darin stehen Zahlungsnummer, Beschreibung, monatliche, vierteljährliche und jährliche Bilanzen.

```
STURKTUR FÜR DATEI: COA.DBF

Feld Name Typ Weite Dez Bezeichnung
001 KONTO N 006 002 Kontonummer
002 TITEL C 020 Zahlungsbeschreibung
003 BETRAG N 012 002 monatliche Bilanz
004 VTJ N 012 002 vierteljährliche Bilanz
005 JRL N 012 002 jährliche Bilanz
006 MERKER C 001 markiert Unterkonten
```

Beachten Sie, daß das Feld KONTO dem in der Datei TRANS.DBF gleicht. Der Grund dafür ist, daß die Kontonummer als Schlüsselfeld für die Aktualisierung der Datei COA mit Daten aus TRANS dient. Das Feld BETRAG ist die gerade gültige Bilanz des Kontos. Die COA-Datei hat ebenfalls ein MERKER-Feld, mit dem Unterkonten markiert werden.

Die Kontenübersicht wird immer in der Reihenfolge der Kontonummern angezeigt. Um die Richtigkeit der in die TRANS-Datei eingegebenen Kontonummern zu überprüfen, müssen wir diese Nummern in der COA-Datei nachsehen. Deshalb wird die COA-Datei nach dem KONTO-Feld indiziert. Nachdem Sie die COA-Datenbank, wie oben gezeigt, erstellt haben, sollten Sie die Index-Datei COA.NDX mit diesen Befehlen auch noch erstellen:

```
USE coa
INDEX ON konto TO coa
```

Die Datei REGISTER enthält das Scheckregister einschließlich der geschriebenen Schecks (die direkt aus der TRANS-Datei gezogen werden) und Einzahlungen auf das Scheckkonto (die der Benutzer einzeln eintippt, da eine Einzahlung auf ein Scheckkonto weder eine Ausgabe noch eine Einnahme ist). Die Datenbankstruktur und kurze Erklärungen dazu sehen Sie hier:

```
STRUKTUR FÜR DATEI: REGISTER.DBF

Feld  Name       Typ   Weite  Dez     Bezeichnung
001   SCHECK:NR  N     004            Schecknummer
002   AN:WEN     C     020            an wen gezahlt
003   GRUND      C     020            Grund der Zahlung
004   BETRAG     N     012    002     Scheckbetrag
005   DATUM      C     008            Scheckdatum
006   EINZAHLNG  L     001            Ist es eine Einzahlung?
```

Die Datenbank REGISTER hält ein logisches Feld (true-false, wahr-unwahr) mit dem Namen EINZAHLNG. Dieses Feld bekommt den Wert „wahr", wenn es sich um eine Einzahlung handelt, und den Wert „unwahr", wenn es sich um einen Scheck handelt. Dies erleichtert es einem Benutzer, Bilanzen auszurechnen, da ein Programm ganz leicht die Beträge für Einzahlungen (EINZAHLNG ist „wahr") summieren kann, dann die Beträge für die Schecks (EINZAHLNG ist „unwahr") zusammenzählt und die Summe der Schecks von der Summe der Einzahlungen abzieht. Das Ergebnis ist die Bilanz des Scheckkontos.
Die Datenbank REGISTER hat einige Felder mit der TRANS-Datei gemeinsam (SCHECK:NR, AN:WEN, GRUND, BETRAG und DATUM), weil sie alle ihre Informationen über Schecks direkt von der TRANS-Datei bekommt. Die Namensgleichheit der Felder vereinfacht die Übertragung der Daten direkt in die REGISTER-Datenbank.

Die Programmstruktur des Buchhaltungssystems

Wie unser vorheriges Programmsystem hat auch die Buchhaltung ein Hauptmenü und verschiedene Programme zum Ändern, Neueintragen und Ausdrucken von Daten aus den verschiedenen Datenbanken. Außerdem gibt es ein Programm zum Aktualisieren der Kontenübersicht aus der Datenbank TRANS, wo die einzelnen Einnahmen und Ausgaben stehen. Ein einmal zu benutzendes Installationsprogramm wird es dem Benutzer erlauben, vor der Arbeit mit dem Buchhaltungssystem eine Kontenübersicht zu erstellen. Dieses Programm wird eine Reihe von direkten Hilfen auf dem Bildschirm bieten, um bei der Erstellung der Kontenübersicht zu helfen.

Die Programmstruktur des Buchhaltungssystems ist in Abb. 14.2 dargestellt. Das Installationsprogramm heißt BINSTALL.CMD. Es ist ein eigenständiges Programm, das keine Verbindung zu den anderen Programmen des Systems hat. Es wird nur einmal benutzt, um die Erstellung der Kontenübersicht überhaupt zu ermöglichen und einige grundsätzliche Informationen zu erfragen: ihre Kontenübersicht und die Nummer des ersten zu schreibenden Schecks. Das Menüprogramm der Buchhaltung heißt BOOKS.CMD und die dazugehörenden Programme werden mit dem Befehl DO und dem entsprechenden Namen von hier aus aufgerufen.

Die Programme BREPORTS und BEDIT in Abb. 14.2 verzweigen weiter zu anderen Programmen. Wir werden diese gestalten und entwickeln, wenn die Zeit dafür gekommen ist.

Im nächsten Kapitel werden wir mit dem Entwurf und der Entwicklung des Programms für das Buchhaltungssystem beginnen, indem wir mit dem Installationsprogramm (BINSTALL) und dem Menüprogramm (BOOKS) anfangen.

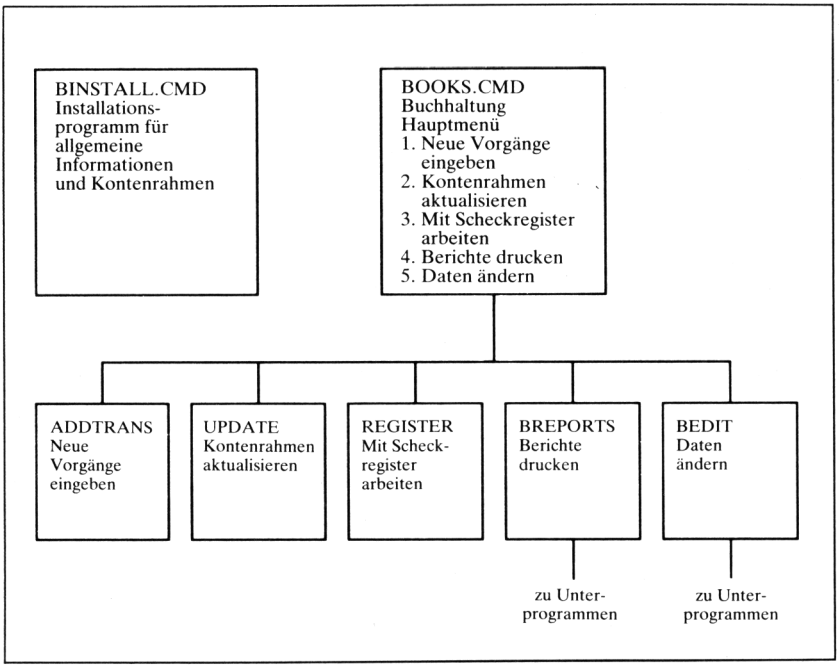


Abb. 14.2: Ein Teil der Programmstruktur des Buchhaltungssystems

Zusammenfassung

In diesem Kapitel haben wir ein menügesteuertes Buchhaltungssystem entworfen, das einfach zu bedienen ist. Wir haben verschiedene Möglichkeiten eingebaut, die auch in anderen Buchhaltungssystemen üblich sind, wie zum Beispiel Möglichkeiten zur Rechnungsprüfung, monatliche, vierteljährliche und jährliche Bilanzen für verschiedene Konten, das automatische Schreiben von Schecks und ein Scheckregister.

Das System besteht aus vier Datenbanken:

1. GENINFO.DBF: enthält allgemeine Informationen.

2. COA.DBF: bearbeitet die Kontenübersicht sowie die Bilanzen der unterschiedlichen Zeiträume.

3. TRANS.DBF: speichert einzelne Einzahlungen und Ausgaben.

Um diese Daten zu verarbeiten, ist eine beachtliche Zahl von Programmen nötig. Wir beginnen mit deren Erstellung im nächsten Kapitel.

Installations- und Menüprogramm für das Buchhaltungssystem

15

Um das Buchhaltungssystem zu benutzen, sollte der Anwender zuerst eine Kontenübersicht erstellen, den Firmennamen (wird bei Berichten als Überschrift gebraucht) und die Nummer des ersten zu druckenden Schecks eingeben. Diese Prozedur führt der Benutzer nur einmal durch. Wenn er sich später entschließt, ein Konto hinzuzufügen, zu ändern oder zu löschen, kann er dazu eine einfache Änderungsroutine aus dem Hauptmenü verwenden.

Wenn wir ein Buchhaltungssystem entwickeln, müssen wir uns erst überlegen, welches Numerierungsschema wir zur Unterscheidung von Einnahmen und Ausgaben bei den Konten benutzen wollen. Ein hochentwickeltes Numerierungssystem (wie zum Beispiel bei der DATEV und großen Kontenrahmen verwendet) enthält auch Möglichkeiten für Abschreibungen, Kredite, Herstellungskosten verkaufter Waren und andere Kontenarten. Bei unserem System teilen wir die Konten nur in Einnahmen und Ausgaben auf.

Kontennummern werden in dem Format XXX.XX gespeichert, womit Kontonummern von 100.00 bis 999.999 möglich sind. Da es meist mehr Ausgabe- als Einnahmekonten gibt, teilen wir die Bereiche der Kontennummern so auf, daß die Nummern von 100.00 bis 299.99 für Einnahmen und ab 300.00 bis 999.99 für Ausgaben verwendet werden:

Kontonummer	Art des Kontos
100.00-299.99	Einnahme-Konten
300.00-999.99	Ausgabe-Konten

Erinnern Sie sich bitte: Wir haben festgelegt, daß der Benutzer auch Unterkonten anlegen kann, um darin zusätzliche Informationen über die jeweiligen Vorgänge zu speichern. Diese Unterkonten sollen aber beim Erstellen der Bilanzen und beim Scheckschreiben nicht berücksichtigt werden. Wir müssen uns also etwas ausdenken, wie wir die Unterkonten von den anderen unterscheiden können, damit unsere Programme sie richtig behandeln. Da unsere Kontennummern zwei Dezimalstellen zulassen, stellen wir die einfache Regel auf, daß jedes Konto, dessen Nachkommastellen etwas anderes als den Wert 00 enthalten, ein Unterkonto ist. Nehmen wir beispielsweise an, unser Benutzer möchte ein Konto für seine Energieversorgung einrichten und nur einen Scheck dafür ausdrucken lassen. Zusätzlich möchte er aber festhalten, wieviel er für Gas und wieviel für Strom ausgegeben hat. Er kann dann drei Konten anlegen, die so aussehen:

400.00	Energiekosten
400.10	Gas
400.20	Strom

Energiekosten ist damit ein ordentliches Konto, weil die Kontonummer größer als 299.99 ist und die Dezimalstellen mit zwei Nullen besetzt sind. Gas und Strom sind Unterkonten, weil ihre Dezimalstellen nicht Null sind. Das ist eine wichtige Regel, an die Sie sich beim Entwickeln des Systems erinnern sollten.

Nachdem wir nun ein paar Regeln zum Erkennen der Kontentypen aufgestellt haben, lassen Sie uns jetzt das Buchhaltungssystem entwickeln.

Buchhaltungs-Installationsprogramm

Das Installationsprogramm bekommt den Namen BINSTALL.CMD. Es wird vom dBASE-Prompt aus aufgerufen mit dem Befehl DO BINSTALL (oder DO B:BINSTALL) und zeigt folgende Information auf dem Bildschirm:

```
Dieses Programm ermöglicht Ihnen die Eingabe
grundsätzlicher Informationen und eines Konten-
rahmens für das Buchhaltungsystem. Sie müssen
dieses Programm nur einmal benutzen. Die Option
"Daten ändern" des Hauptmenüs im Programm BOOKS
erlaubt Ihnen noch spätere Änderungen.

'Geben Sie den Firmennamen oder den Benutzernamen
ein:                                              :

'Geben Sie die erste Schecknummer ein :           :
```

Nachdem der Benutzer den Namen und die erste Schecknummer eingegeben hat, erscheint die nächste Mitteilung auf dem Bildschirm:

```
Jetzt sollen Sie den Kontenrahmen eingeben.
Kontennummern besitzen dieses Format XXX.XXX
(z.B.: 100.00).
Kontennummern zwischen 100.00 und 299.99 sind
nur für EINNAHMEN, Kontennummern im Bereich
zwischen 300.00 und 999.99 sind für AUSGABEN
vorgesehen.

Weiter mit beliebiger Taste ...
```

Der Benutzer drückt auf irgendeine Taste, und die nächste Mitteilung erscheint auf dem Bildschirm:

```
Konten, deren Dezimalstellen nicht Null sind,
gelten als Unterkonten und werden bei Berechnungen
nicht berücksichtigt. Wenn Sie möchten, können
Sie trotzdem Gesamtzahlungen auf mehrere
Unterkonten aufteilen. Sie schreiben zum
Beispiel nur einen Scheck für Energiekosten.
Trotzdem können Sie den Teilbetrag für das
Gas und den für Strom einzeln speichern.
Um das zu erreichen, richten Sie folgende
drei Konten ein:

                Konto
                Nummer          Bezeichnung

                400.00          Energiekosten
                400.10          Gas
                400.20          Strom

In diesem Beispiel wird nur ein Scheck
geschrieben, nämlich für Konto 400.00.
Die Unterkonten 400.10 und 400.20 werden
zwar gespeichert, aber Schecks werden nicht
geschrieben.

Weiter mit RETURN, aufhören mit ESC ...
```

Als nächstes bekommt der Benutzer die Möglichkeit, seinen Kontenrahmen einzugeben, immer ein Konto nach dem anderen. Die Maske zum

Eingeben der Kontonummer, der Bezeichnung und des Anfangsbetrags sehen Sie hier:

```
Geben Sie Kontonummer, Bezeichnung und Anfangsbe-
                                          trag ein
Aufhören mit Ctrl-W

Kontonummer :        .  :
Bezeichnung :                 :
monatlich   :        .  :
1/4-jährl.  :        .  :
jährlich    :        .  :
```

Der Benutzer kann hier fortlaufend Vorgänge eingeben und mit Control-W aufhören. Danach kommt diese Meldung zusammen mit dem dBASE-Prompt auf den Bildschirm:

```
Zurück zu dBASE. Um das Buchhaltungsprogramm zu
benutzen, geben Sie DO BOOKS ein, sobald der Punkt
erscheint.
```

Der Pseudocode für das BINSTALL-Programm steht in der Abb. 15.1.

Das BINSTALL-Programm ist wirklich einfach. Zuerst stellt es die benötigten dBASE-Parameter ein:

```
********************************** BINSTALL.CMD
****************************** START
SET TALK OFF
SET DEFA TO B
```

Dann benutzt es die Datenbank GENINFO, löscht alle Einträge, hängt einen neuen Datensatz an, um die neuen Benutzereingaben zu speichern:

```
********************************** Hole allgemeine
********************************** Informationen
ERASE
USE geninfo
DELE ALL
PACK
APPEND BLANK
```

Danach zeigt es die erste Mitteilung auf dem Bildschirm und wartet, bis der Benutzer den Firmennamen und die Anfangsschecknummer eingegeben hat:

```
Setze dBASE-Parameter

Lösche den Bildschirm
Benutze Datenbank für allgemeine Informationen (GENINFO)
Lösche dort alle existierenden Daten
Füge einen leeren Datensatz hinzu

Zeige erste Seite der Mitteilungen für das Installationsprogramm

Hole Firmenname vom Benutzer
Hole die erste Schecknummer
Speichere beides in die GENINFO-Datenbank

Zeige Informationen für die Kontonummern

Bereite die Kontennummern-Datenbank vor und lösche dafür
alle existierenden Eintragungen

Beginne Schleife für Eingabe der Kontonummern
   Zeige Anweisungen
   Hole Kontonummer, Bezeichnung und Anfangswert für
   jedes Konto

Weiter mit Schleife (bis Benutzer Ende wünscht)

Zeige Schlußinformationen

Stelle sicher, daß kein Konto mit der Nummer 0 eingegeben wurde

Lösche Variablen
Zurück zum dBASE-Prompt
```

Abb. 15.1: Pseudocode für das Programm BINSTALL.CMD

```
TEXT
  Dieses Programm ermöglicht Ihnen die Eingabe
  grundsätzlicher Informationen und eines
  Kontenrahmens für das Buchhaltungssystem.
  Sie müssen dieses Programm nur einmal
  benutzen. Die Option "Daten ändern" des
  Hauptmenüs im Programm BOOKS erlaubt
  Ihnen noch spätere Änderungen.

 'Geben Sie den Firmennamen oder den Benutzernamen
 ein:                                              :
```

```
 'Geben Sie die erste Schecknummer ein :      :

ENDTEXT

§ 11,1 SAY 'Geben Sie den Firmen- oder Benutzernamen;
                                                ein'
§ 11,41 GET firma
§ 13,1 SAY 'Geben Sie die erste Schecknummer ein'
§ 13,50 GET 1:scheck
READ
```

Nachdem der Benutzer seine Anfangsinformationen eingetippt hat, zeigt das Programm Hinweise, wie den Einnahmen und Ausgaben Kontennummern zugeordnet werden. Das Programm wartet auf einen Tastendruck, bevor es weitergeht:

```
********************* Jetzt zeige Anweisungen für
********************* die Kontennummern
ERASE

TEXT
   Jetzt sollen Sie den Kontenrahmen eingeben.
   Kontennummern besitzen das Format XXX.XXX
   (z.B.: 100.00). Kontennummern zwischen 100.00
   und 299.99 sind nur für EINNAHMEN,
   Kontennummern im Bereich zwischen
   300.00 und 999.99 sind für AUSGABEN vorgesehen.

   Weiter mit beliebiger Taste ...
ENDTEXT
WAIT
```

Dann zeigt das Programm die Informationen zur Erkennung der Unterkonten und wartet wieder auf einen Tastendruck:

```
********************* Dann zeige Informationen zum
********************* Erkennen der Unterkonten
ERASE

TEXT

   Konten, deren Dezimalstellen nicht Null sind,
   gelten als Unterkonten und werden bei
   Berechnungen nicht berücksichtigt. Wenn Sie
   möchten, können Sie trotzdem Gesamtzahlungen auf
   mehrere Unterkonten aufteilen. Sie schreiben zum
```

```
      Beispiel nur einen Scheck für Energiekosten.
      Trotzdem können Sie den Teilbetrag für das Gas
      und den für Strom einzeln speichern. Um das zu
      erreichen, richten Sie folgende drei Konten ein:

                  Konto
                  Nummer          Bezeichnung

                  400.00          Energiekosten
                  400.10          Gas
                  400.20          Strom

      In diesem Beispiel wird nur ein Scheck
      geschrieben, nämlich für Konto 400.00. Die
      Unterkonten 400.10 und 400.20 werden zwar
      gespeichert, aber Schecks werden nicht geschrie-
ben.

      Weiter mit RETURN, aufhören mit ESC ...

ENDTEXT
WAIT
```

Dann nimmt das Programm die Datenbank COA.DBF (Datenbank für die Kontonummern) zusammen mit der entsprechenden Index-Datei (COA.NDX) in Gebrauch (USE) und löscht alle vorhandenen Datensätze:

```
********************** Bereite die COA-Datei für Auf-
********************** nahme der Kontennummern vor
ERASE
USE coa INDEX coa
DELE ALL
PACK
```

Das Programm beginnt anschließend eine Schleife, bietet einige Grundinformationen und gestattet es dem Benutzer, Kontonummer, Bezeichnung und Anfangswerte einzugeben:

```
********************** Ermögliche die Eingabe eines
********************** Kontenrahmens
STORE T TO ktoein
DO WHILE ktoein
APPEND BLANK
ERASE
§ 1, 0 SAY 'Kontonummer, Bezeichnung'
§ 1,32 SAY 'und Anfangswert eingeben'
```

```
§ 2, 0 SAY 'Mit Ctrl-W aufhören'
§ 4, 0 SAY 'Kontonummer'
§ 4,16 GET konto
§ 5, 0 SAY 'Bez.'
§ 5,16 GET titel
§ 6, 0 SAY 'Monat'
§ 6,16 GET betrag
§ 7, 0 SAY 'Quartal'
§ 7,16 GET vtj
§ 8, 0 SAY 'Jahr'
§ 8,16 GET jrl
READ

********************** Prüfe, ob 0 als Kontonummer
IF konto = 0
   STORE f TO ktoein
ENDIF
ENDDO (while ktoein)
```

Sobald der Benutzer mit der Eingabe der Kontonummern fertig ist, zeigt das Programm die Abschlußmeldung auf dem Bildschirm:

```
********************** Fertig mit Eingabe
ERASE
TEXT
   Zurück zu dBASE. Um das Buchhaltungsprogramm
   zu benutzen, geben Sie DO BOOKS ein, sobald der
   Punkt erscheint.

ENDTEXT
```

Dann löscht das Programm alle Konten mit der Kontonummer Null.

```
*************** Stelle sicher, daß kein Konto mit der
*************** Nummer Null im Kontenrahmen steht
DELE ALL FOR konto= 0
PACK
```

Am Ende löscht das Programm nur noch die Variablen, schließt die COA.DBF-Datenbank und kehrt zum dBASE-Prompt zurück:

```
********************** Beende das BINSTALL-Programm
RELEASE ALL
USE
```

Das komplette Installationsprogramm BINSTALL.CMD ist in der Abb. 15.2 zu sehen.

```
********************************** BINSTALL.CMD
********************************** START
SET TALK OFF
SET DEFA TO B

********************************** Hole allgemeine
********************************** Informationen
ERASE
USE geninfo
DELE ALL
PACK
APPEND BLANK

TEXT

  Dieses Programm ermöglicht Ihnen die Eingabe
  grundsätzlicher Informationen und eines Kontenrahmens
  für das Buchhaltungsystem. Sie müssen dieses Programm
  nur einmal benutzen. Die Option "Daten ändern" des Hauptmenüs
  im Programm BOOKS erlaubt Ihnen noch spätere Änderungen.

ENDTEXT

§ 11,1 SAY 'Geben Sie Firma oder Benutzernamen ein'
§ 11,41 GET firma
§ 13,1 SAY 'Geben Sie die erste Schecknummer ein'
§ 13,50 GET 1:scheck
READ

********************************** Jetzt zeige Anweisungen für
********************************** die Kontennummern
ERASE

TEXT
   Jetzt sollen Sie den Kontenrahmen eingeben.
   Kontennummern besitzen das Format XXX.XX (z.B.: 100.00).
   Kontennummern zwischen 100.00 und 299.99 sind nur für
   EINNAHMEN, Kontennummern im Bereich zwischen
   300.00 und 999.99 sind nur für AUSGABEN vorgesehen.

   Weiter mit beliebiger Taste ...

ENDTEXT
WAIT

***************************** Dann zeige Informationen zum
***************************** Erkennen der Unterkonten
ERASE
```

Abb. 15.2: Das Programm BINSTALL.CMD

```
TEXT

    Konten, deren Dezimalstellen nicht Null sind, gelten als
    Unterkonten und werden bei Berechnungen nicht berücksichtigt.
    Wenn Sie möchten, können Sie trotzdem Gesamtzahlungen auf
    mehrere Unterkonten aufteilen. Sie schreiben zum Beispiel
    nur einen Scheck für Energiekosten. Trotzdem können Sie den
    Teilbetrag für das Gas und den für Strom einzeln speichern.
    Um das zu erreichen, richten Sie folgende drei Konten ein:

                Konto
                Nummer          Bezeichnung

                400.00          Energiekosten
                400.10          Gas
                400.20          Strom

    In diesem Beispiel wird nur ein Scheck geschrieben, nämlich
    für Konto 400.00. Die Unterkonten 400.10 und 400.20 werden
    zwar gespeichert, aber Schecks werden nicht geschrieben.

    Weiter mit RETURN, aufhören mit ESC ...

ENDTEXT
WAIT

********************** Bereite die COA-Datei für Aufnahme der
********************** Kontennummern vor
ERASE
USE coa INDEX coa
DELE ALL
PACK

********************** Ermögliche die Eingabe eines
********************** Kontenrahmens
STORE T TO ktoein
DO WHILE ktoein
APPEND BLANK
ERASE
§  1, 0 SAY 'Kontonummer, Bezeichnung'
§  1,32 SAY ' und Anfangswert eingeben'
§  2, 0 SAY 'Mit Ctrl-W aufhören'
§  4, 0 SAY 'Kontonummer'
§  4,16 GET konto
§  5, 0 SAY 'Bez.'
§  5,16 GET titel
§  6, 0 SAY 'Monat'
```

Abb. 15.2: Das Programm BINSTALL.CMD (Forts.)

```
§  6,16 GET betrag
§  7, 0 SAY 'Quartal'
§  7,16 GET vtj
§  8, 0 SAY 'Jahr'
§  8,16 GET jrl
READ
********************* Prüfe, ob 0 als Kontonummer
IF konto = 0
   STORE f TO ktoein
ENDIF
ENDDO (while ktoein)

********************* Fertig mit Eingabe
ERASE
TEXT
     Zurück zu dBase. Um das Buchhaltungsprogramm zu benutzen,
     geben Sie DO BOOKS ein, sobald der Punkt erscheint.

ENDTEXT

********************* Stelle sicher, daß kein Konto mit der
********************* Nummer Null im Kontenrahmen steht
DELE ALL FOR konto= 0
PACK

********************* Beende das BINSTALL-Programm
RELEASE ALL
USE
```

Abb. 15.2: Das Programm BINSTALL.CMD (Forts.)

Bevor Sie irgendwelche anderen Programme entwickeln, sollten Sie BINSTALL laufen lassen und einige Kontonummern eingeben, damit schon einige Daten vorhanden sind, wenn Sie das Programm testen. Sie können einen kleinen persönlichen Kontenrahmen zum Beispiel so aufbauen:

Konto Nummer	Bezeichnung	Stand Monat	Quartal	Jahr
100.00	Einkommen	0.00	0.00	0.00
110.00	Lizenzeinn.	0.00	0.00	0.00
120.00	Zinseinn.	0.00	0.00	0.00
300.00	Lohnsteuern	0.00	0.00	0.00
310.00	Umsatzsteuer	0.00	0.00	0.00
400.00	Hausfinanzierung	0.00	0.00	0.00
400.10	Kapitalabtrag	0.00	0.00	0.00
400.20	Zinsen	0.00	0.00	0.00

500.00	Autokredit	0.00	0.00	0.00
510.00	Versicherung	0.00	0.00	0.00
520.00	Benzin	0.00	0.00	0.00
530.00	Reparaturen	0.00	0.00	0.00
600.00	Krankheitskosten	0.00	0.00	0.00
610.00	Medizin	0.00	0.00	0.00
620.00	Zahnarzt	0.00	0.00	0.00
630.00	Krankenvers.	0.00	0.00	0.00
700.00	Bewirtung (Gesch.)	0.00	0.00	0.00
710.00	Bewirtung (Priv.)	0.00	0.00	0.00

Buchhaltungssystem Hauptmenü

Das Hauptmenü des Buchhaltungssystems ist in dem Programm BOOKS.CMD gespeichert. Es zeigt zuerst die Systemzeit des Computers (DATE()) und gestattet dem Benutzer diese zu ändern. Die Variable T:DATUM wird dann im gesamten System benutzt. Das Programm zeigt anschließend das Hauptmenü so, wie Sie es hier sehen:

```
        Buchhaltung Hauptmenü

1.   Neue Vorgänge eingeben
2.   Kontenrahmen aktualisieren
3.   Mit Scheckregister arbeiten
4.   Berichte drucken
5.   Daten ändern

6.   Ende
```

Da das Programm BOOKS sich von den bisher entwickelten Menüprogrammen nicht unterscheidet, zeige ich das Gesamtprogramm in der Abb. 15.3.

```
********************************* BOOKS.CMD
****************** Buchhaltungssystem Hauptmenü
SET TALK OFF
SET DEFAULT TO B
SET BELL OFF
STORE 0 TO wahl
```

Abb. 15.3: Das Programm BOOKS.CMD

```
******************** Hole heutiges Datum
ERASE
STORE DATE() TO t:datum
§ 5, 1 SAY 'Geben Sie das Datum ein';
GET t:datum PICTURE '99/99/99'
READ
******************** Zeige Hauptmenü
DO WHILE wahl <> 6
   ERASE
   TEXT
                Buchhaltung Hauptmenü

          1.    Neue Vorgänge eingeben
          2.    Kontenrahmen aktualisieren
          3.    Mit Scheckregister arbeiten
          4.    Berichte drucken
          5.    Daten ändern

          6.    Ende

   ENDTEXT

   § 12,10 SAY 'Ihre Wahl (1-6) '
   § 12,45 GET wahl PICTURE '9'
   READ

   ********************* Verzweige entsprechend
   DO CASE

      CASE wahl = 1
         DO addtrans

      CASE wahl = 2
         DO UPDATE

      CASE wahl = 3
         DO register

      CASE wahl = 4
         DO breports

      CASE wahl = 5
         DO bedit

   ENDCASE
ENDDO (WHILE wahl <> 6)
QUIT
```

Abb. 15.3: Das Programm BOOKS.CMD (Forts.)

Zusammenfassung

In diesem Kapitel haben wir die ersten beiden Programme des Buchhaltungssystems entwickelt. Das sind:

1. BINSTALL.CMD, das die Regeln zum Erstellen des Kontenrahmens beschreibt und dem Benutzer die Eingabe eines Kontenrahmens erlaubt. Außerdem kann er den Namen seiner Firma eingeben, der auf Berichten als Überschrift erscheint, und die Anfangsschecknummer zum Ausdrucken von Schecks.
2. BOOKS.CMD, das Programm mit dem Hauptmenü des Buchhaltungssystems.

Im nächsten Kapitel werden wir das Programm ADDTRANS.CMD entwickeln, das die Eingabe von Vorgängen in die Datenbank TRANS ermöglicht.

Vorgänge eingeben und aktualisieren

16

In diesem Kapitel werden wir das Programm entwickeln, mit dem Sie neue Vorgänge in die Datenbank TRANS.DBF eingeben und den Kontenrahmen (Datenbank COA.DBF) aus der TRANS-Datei aktualisieren können.

Buchhaltungsvorgänge speichern

Das Programm zum Eingeben von Geschäftsvorgängen in das Buchhaltungssystem dient der Bequemlichkeit und der Sicherheit des Benutzers. Wenn der Benutzer eine Kontonummer eingibt, die nicht existiert, erhält er eine Warnung und kann es noch einmal versuchen. Gibt er aber eine gültige Nummer ein, erscheint deren Bezeichnung auf dem Bildschirm, und er kann mit der Eingabe weitermachen.

Um Geschäftsvorgänge einzugeben, wählt der Benutzer die Option 1 des Hauptmenüs. Daraufhin erscheint folgende Meldung auf dem Bildschirm:

```
-Neue Einträge eingeben (0 wenn fertig)-
 Kontonummer:_       :
```

Wenn der Benutzer jetzt eine ungültige Nummer eingibt, piepst der Computer, zeigt eine Warnung und fragt nach einer neuen Kontonummer, wie hier gezeigt:

```
-Neue Einträge eingeben (0 wenn fertig)-
 Kontonummer:_       :** Konto nicht vorhanden**
```

Gibt er aber eine gültige Nummer ein, fragt der Bildschirm nach weiteren Informationen:

```
-Neue Einträge eingeben (0 wenn fertig)-
 Kontonummer:_       :

 Beschreibung:_                 :
 Betrag:          .   :
 Datum :10/10/84:
```

Der Benutzer kann entweder eine Beschreibung eingeben oder einfach RETURN drücken, woraufhin die Kontobeschreibung als Beschreibung eingeblendet wird. Hätte der Benutzer zum Beispiel die Kontonummer 710 eingegeben, hätte der Bildschirm die Kontobeschreibung (Bewirtung (Gesch.)) gezeigt und auf eine neue Beschreibung gewartet. Die Beschreibung hier könnte heißen „Essen mit Elmar A. Weiler", die der Benutzer einfach eintippt. Gibt er allerdings eine Kontonummer wie 100 ein, ist die Kontobeschreibung „Einkommen" ausreichend, und er kann einfach RETURN drücken, worauf die Beschreibung automatisch in das Feld Beschreibung in der Maske übernommen wird, wie Sie hier sehen:

```
-Neue Einträge eingeben (0 wenn fertig)-
 Kontonummer:100.00:

 Beschreibung:Einkommen         :
 Betrag:_         .   :
 Datum :10/10/84:
```

Er kann dann den Betrag seines Einkommens eingeben und dann entweder das eingeblendete Datum übernehmen oder ein anderes eingeben.

Da die obige Eingabe für ein Einnahmefeld ist, wird nicht nach weiteren Informationen gefragt. Statt dessen erscheint die Frage nach der nächsten Kontonummer. Angenommen, der Benutzer gibt die Nummer 300 für Lohnsteuer ein. Wieder kann er einfach RETURN drücken, um die Kontobezeichnung als Beschreibung zu übernehmen, den Betrag eingeben und das vorgegebene Datum verwenden. Da dies aber eine Ausgabe ist, braucht das Programm zusätzliche Informationen, die es sich so besorgt:

```
-Neue Einträge eingeben (0 wenn fertig)-
 Kontonummer:300.00:
```

```
Beschreibung:Lohnsteuer      :
Betrag:      398.76:
Datum :10/10/84:
Typ (1=Scheck, 2= anderes) :_:
```

Das Programm will wissen, welche Art von Ausgabe das war. Wenn ein Scheck benutzt wird, sollte der Benutzer eine 1 eingeben. Da dies aber eine steuerliche Ausgabe war, gibt er eine 2 ein. Danach fragt das Programm nach der nächsten Kontonummer.

Bei der nächsten Eingabe gibt der Benutzer die Nummer 710 ein und füllt das Beschreibungsfeld aus, statt einfach nur RETURN zu drücken. Dann trägt er den Betrag und das Datum ein und tippt eine 1, um anzuzeigen, daß er per Scheck bezahlt hat. Das Programm fragt dann nach weiteren Einzelheiten:

```
-Neue Einträge eingeben (0 wenn fertig)-
 Kontonummer:710.00:

 Beschreibung:Essen mit Elmar :
 Betrag:       12.99:
 Datum :10/10/84:
 Typ (1=Scheck, 2= anderes) :1:
 Schecknummer: 234:
 Gezahlt an:                          :
 Scheck drucken? (J/N): :
```

Das System schlägt automatisch Schecknummern vor. Der Benutzer kann diese Nummer einfach übernehmen oder eine andere eingeben. Dann muß er noch angeben, an wen er den Scheck ausgestellt hat (oder noch auszustellen hat), und die Frage „Scheck drucken" mit J oder N beantworten. Logischerweise braucht er keinen Scheck zu drucken, wenn er ihn bereits ausgestellt hat. Wenn er allerdings den Scheck zu drucken wünscht, kann er mit Ja antworten, und das System druckt den Scheck.

Das Programm für diese Kunststücke heißt ADDTRANS.CMD. Der Pseudocode steht in der Abb. 16.1.

Also, los geht's. Wir fangen mit den üblichen Kommentaren an und holen die letzte Schecknummer aus der GENINFO-Datei. Beachten Sie, daß die Variable SCHNR die Schecknummer enthält:

```
************************************ ADDTRANS.CMD
************************* Neue Einträge eingeben
```

```
Hole letzte Schecknummer von der Datei GENINFO

Lege erste und zweite Datenbank fest

Beginne Schleife zum Eingeben von Geschäftsvorgängen
   Neuen Datensatz an Datei Vorgänge (TRANS) anhängen
   Fülle Datumfeld mit Tagesdatum
   Zeige Anweisungen

   Beginne Schleife zum Zurückweisen ungültiger Nummern
      Frage nach Kontonummer
      Wenn Benutzer Nummer eingibt (nicht Null)
         Stelle sicher, daß Konto in COA existiert
         Wenn Konto existiert, hole Kontobeschreibung
         Wenn Konto nicht existiert, Warnung ausgeben
         piepsen und zurück zur Frage nach Kontonummer

   Wenn keine Kontonummer eingegeben, Ende

      Hole Rest des Vorgangs, wenn nicht Ende
      Hole Grund für Eingabe
      Wenn kein Grund angegeben, verwende Kontobeschreibung
      Hole Betrag und Datum

      Wenn Ausgabe, frage nach dem Typ
      (Scheck oder nicht?)

      Wenn Scheck drucken, frage nach Scheckinformationen
         Zeige an und frage nach Schecknummer
         Frage, an wen gezahlt
         Scheck drucken?

         Wenn Scheck nicht drucken, markiere als gedruckt

         Erhöhe Scheckzähler um 1

Weiter mit Schleife für Eingabe (bis Ende verlangt)

Wenn fertig mit Eingabe, Dateien schließen, leeren Datensatz
beseitigen und die Datei GENINFO aktualisieren

Speichervariablen löschen
Zurück zum Hauptmenü
```

Abb. 16.1: Pseudocode für das Programm ADDTRANS

```
*************** Zuerst letzte Schecknummer holen
USE geninfo
STORE 1:scheck TO schnr
```

Als nächstes muß das Programm die beiden Datenbanken TRANS.DBF (für die Neueintragung) und COA.DBF (zum Überprüfen der Kontonummer und Heraussuchen der Bezeichnung) eröffnen:

```
SELECT PRIMARY
USE trans INDEX konto,datum
SELECT SECONDARY
USE coa INDEX coa
```

Jetzt müssen wir die Schleife für die Eingabe beginnen:

```
******************** Beginne Schleife für Eingabe
STORE T TO transein
DO WHILE transein
```

Sobald die Schleife beginnt, muß das Programm einen leeren Datensatz an die Datei TRANS anhängen und das Feld DATUM mit dem aktuellen Datum füllen. Dann löscht es den Bildschirm und zeigt kurze Informationen:

```
APPEND BLANK
REPLACE DATE WITH t:datum
ERASE
§ 1, 0 SAY 'Neue Einträge eingeben (0 wenn fertig)'
```

Jetzt brauchen wir eine Schleife, um die Kontonummern zu überprüfen. Sollte der Benutzer eine ungültige Nummer eingeben, muß das Programm zurückgehen und noch einmal nach der Nummer fragen. Die Variable OK kontrolliert diese Schleife. Dann fragt das Programm nach der Kontonummer:

```
********** Frage nach Kontonummer der Eingabe
************** und beginne Schleife zum Zurückweisen
************** ungültiger Nummern
   STORE N TO ok
   DO WHILE .NOT. ok
     § 3, 0 SAY 'Kontonummer'
     § 3,15 GET konto
      READ
```

Hat der Benutzer eine Kontonummer eingegeben (größer als Null), muß das Programm die Nummer in der Datei COA.DBF nachsehen:

```
IF konto> 0
  ***************** Sicherstellen, daß Nummer in
  ***************** COA existiert
  STORE STR(konto,6,2) TO pruef
  SELE SECO
  FIND &pruef
```

Wenn die Kontonummer bereits existiert, zeigt das Programm die Beschreibung auf dem Bildschirm und speichert sie in der Variablen MTITEL. Wenn die Kontonummer nicht vorhanden ist, produziert das Programm einen Piepston und warnt den Benutzer:

```
IF # > 0
   **** Wenn gefunden, hole Beschreibung
   STORE T TO ok
   § 3,26 SAY titel
   STORE titel TO mtitel
ELSE
   *** Wenn nicht gefunden, Benutzer warnen
   § 3,26 SAY '** Konto nicht vorhanden **'
   ? chr(7)
ENDIF (Kontonummer existiert)
```

Hat der Benutzer überhaupt keine Kontonummer eingegeben, verändert das Programm die Schleifenvariablen (TRANSEIN und OK) und „fällt" dann zum ENDDO-Befehl (LOOP). Die gegenwärtigen Werte in den beiden Variablen sorgen dafür, daß das Programm beendet wird:

```
   ELSE
      STORE F TO transein
      STORE T TO ok
      LOOP
   ENDIF (konto>0)
   SELECT PRIM
ENDDO (WHILE .NOT. ok)
```

Hat der Benutzer keinen Befehl zum Aufhören gegeben, fragt das Programm nach einer Beschreibung für den Vorgang:

```
************************ Hole restliche Informatio-
************************ nen, wenn nicht Ende
   IF transein
      ************ Hole Beschreibung des Vorgangs
      § 5, 0 SAY 'Beschreibung'
      § 5,12 GET grund
      READ
```

Wenn der Benutzer keine Beschreibung eingibt, füllt das Programm automatisch das Feld mit der Kontobeschreibung:

```
********** Wenn keine Beschreibung eingegeben,
********** verwende Kontobeschreibung
IF grund = ' '
   REPLACE grund WITH titel
   § 5,13 SAY grund
ENDIF (grund = ' ' )
```

Als nächstes fragt es nach dem Betrag und dem Datum des Vorgangs:

```
********************** Hole Betrag und Datum
§ 7, 0 SAY 'Betrag'
§ 7, 7 GET betrag
§ 9, 0 SAY 'Datum'
§ 9, 5 GET datum
READ
```

Wenn jetzt der eingegebene Vorgang eine Ausgabe ist (und kein Unterkonto), fragt das Programm nach dem Typ (1=Scheck, 2=anderes). Da wir festgelegt haben, daß alle Unterkonten einen höheren Wert als 00 in ihren Dezimalstellen nach dem Komma haben müssen, können wir leicht feststellen, ob ein eingegebenes Konto ein Unterkonto ist oder nicht. Wir machen das, indem wir den ganzzahligen Wert (INT) der Nummer mit der Kontonummer vergleichen. Das bedeutet: Wenn die Kontonummer zum Beispiel 100.00 lautet, ist ihr ganzzahliger Wert 100 und die beiden sind gleich. Der ganzzahlige Wert des Unterkontos 123.45 lautet aber 123 und ist nicht gleich der Kontonummer (123.45). Diese Routine trifft die Entscheidung und fragt nach zusätzlichen Informationen, wenn notwendig:

```
***************** Frage nach dem Typ, wenn Ausgabe
***************** und kein Unterkonto
IF konto > 299.99 .AND. INT(konto)=konto
  § 11, 0 SAY 'Typ (1=Scheck, 2=anders)'
  § 11,24 GET typ
          READ
```

Wenn jetzt als Typ Scheck (1) angegeben wird, zeigt das Programm eine Schecknummer (veränderbar), fragt nach dem Empfänger und ob ein Scheck gedruckt werden soll. Diese Angaben werden in der Datenbank TRANS in den Feldern SCHECK:NR, AN:WEN und GESCHR gespeichert:

```
**************** Wenn Typ = Scheck, frage nach den
**************** Scheckinformationen
```

```
IF typ = 1
   REPLACE scheck:nr WITH schnr
   STORE ' ' TO jn
   § 13, 0 SAY 'Schecknummer'
   § 13,13 GET scheck:nr
   § 15, 0 SAY 'Gezahlt an?'
   § 15,13 GET an:wen
   § 17, 0 SAY 'Scheck drucken?  (J/N)'
   § 17,22 GET jn PICTURE '!'
   READ
```

Wenn der Scheck nicht gedruckt werden soll, setzt das Programm das Feld GESCHR auf „wahr" (T), was bedeutet: der Scheck wurde bereits geschrieben:

```
***************** Wenn kein Scheck zu schreiben ist,
***************** Feld GESCHR auf T setzen
IF jn = 'N'
   REPLACE geschr WITH T
ENDIF (jn = 'N')
```

Dann erhöht das Programm den Schecknummerzähler (SCHNR) um 1 und schließt die DO WHILE- und die IF-Schleifen:

```
            ************ Schecknummer eins höher
            STORE scheck:nr TO schnr
            STORE schnr +1 TO schnr
         ENDIF (typ=1)
      ENDIF (konto>299.99)

   ENDIF (IF transein)
ENDDO (WHILE transein)
```

Wenn die Eingaben beendet sind, löscht das Programm den zuletzt angehängten, leeren Datensatz, schließt alle Datenbanken, löscht die Variablen, aktualisiert die Datei GENINFO mit der letzten Schecknummer und kehrt zum Hauptmenü zurück:

```
*********** Wenn fertig mit Eingabe, schließe
*********** Dateien, lösche letzten Datensatz und
*********** aktualisiere die Datenbank GENINFO
SELE SECO
USE
SELE PRIM
DELETE ALL FOR konto=0
```

```
PACK
USE
USE geninfo
REPLACE l:scheck WITH schnr
USE
RELEASE schnr,transein,ok,pruef,mtitel,jn
RETURN
```

Das komplette Programm ADDTRANS.CMD finden Sie in der Abb. 16.2.

```
********************************** ADDTRANS.CMD
*********************** Neue Einträge eingeben

***************************** Zuerst letzte Schecknummer holen
USE geninfo
STORE l:scheck TO schnr

SET BELL OFF
SELE PRIM
USE trans INDEX konto,datum
SELE SECO
USE coa INDEX coa

SELE PRIM

************************ Beginne Schleife für Eingabe
STORE T TO transein
DO WHILE transein
   APPEND BLANK
   REPLACE datum WITH t:datum
   ERASE
   § 1, 0 SAY '-Neue Einträge eingeben (0 wenn fertig)-'

   ************************ Frage nach Kontonummer der Eingabe
   ************************ und beginne Schleife zum
   ************************ Zurückweisen ungültiger Nummern
   STORE N TO ok
   DO WHILE .NOT. ok
      § 3, 0 SAY 'Kontonummer'
      § 3,15 GET konto
      READ

      IF konto> 0
         ************************ Sicherstellen, daß Nummer in
         ************************ COA existiert
```

Abb. 16.2: Das Programm ADDTRANS.CMD

```
        STORE STR(konto,6,2) TO pruef
        SELE SECO
        FIND &pruef

        IF # > 0
           **** Wenn gefunden, hole Beschreibung
           STORE T TO ok
           § 3,26 SAY titel
           STORE titel TO mtitel
        ELSE
           *** Wenn nicht gefunden, Benutzer warnen
           § 3,26 SAY '** Konto nicht vorhanden **'
           ? chr(7)
        ENDIF (Kontonummer existiert)

     ELSE
        STORE F TO transein
        STORE T TO ok
        LOOP
     ENDIF (konto>0)
     SELE PRIM
  ENDDO (WHILE .NOT. ok)

  ************************ Hole restliche Informationen, wenn
  ************************ nicht Ende
  IF transein
     ************************ Hole Beschreibung des Vorgangs
     § 5, 0 SAY 'Beschreibung'
     § 5,12 GET grund
     READ

     ************************ Wenn keine Beschreibung eingegeben,
     ************************ verwende Kontobeschreibung
     IF grund = ' '
        REPLACE grund WITH titel
        § 5,13 SAY grund
     ENDIF (grund = ' ' )

     ************************ Hole Betrag und Datum
     § 7, 0 SAY 'Betrag'
     § 7,7 GET betrag
     § 9, 0 SAY 'Datum'
     § 9,5 GET datum
     READ

     ************************ Frage nach dem Typ, wenn Ausgabe
     ************************ und kein Unterkonto
```

Abb. 16.2: Das Programm ADDTRANS.CMD (Forts.)

```
      IF konto > 299.99 .AND. INT(konto)=konto
         § 11, 0 SAY 'Typ (1=Scheck, 2=anders)'
         § 11,24 GET typ
         READ

      ************************ Wenn Typ = Scheck, frage nach den
      ************************ Scheckinformationen
         IF typ = 1
            REPLACE scheck:nr WITH schnr
            STORE ' ' TO jn
            § 13, 0 SAY 'Schecknummer'
            § 13,13 GET scheck:nr
            § 15, 0 SAY 'Gezahlt an?'
            § 15,13 GET an:wen
            § 17, 0 SAY 'Scheck drucken?  (J/N)'
            § 17,22 GET jn PICTURE '!'
            READ

            ************************ Wenn kein Scheck zu schreiben
            ************************ ist, Feld GESCHR auf T setzen
            IF jn = 'N'
               REPLACE geschr WITH T
            ENDIF (jn = 'N')

            ************************ Schecknummer eins höher
            STORE scheck:nr TO schnr
            STORE schnr +1 TO schnr
         ENDIF (typ=1)
      ENDIF (konto>299.99)

   ENDIF (IF transein)
ENDDO (WHILE transein)

************************ Wenn fertig mit Eingabe, schließe
************************ Dateien, lösche letzten Datensatz und
************************ aktualisiere die Datenbank GENINFO
SELE SECO
USE
SELE PRIM
DELETE ALL FOR konto=0
PACK
USE
USE geninfo
REPLACE 1:scheck WITH schnr
USE
RELEASE schnr,transein,ok,pruef,mtitel,jn
RETURN
```

Abb. 16.2: Das Programm ADDTRANS.CMD (Forts.)

Die Kontenaufstellung aktualisieren

Die Option 2 des Hauptmenüs ermöglicht es dem Benutzer, die Kontenaufstellung nach der Eingabe von Vorgängen zu aktualisieren. Aktualisieren ist jederzeit möglich, wenn man dem Programm mitteilt, ob es sich dabei um den Beginn eines neuen Monats, Quartals oder Jahres handelt.

Aktualisiert der Benutzer in der Monatsmitte, werden die Beträge aus der TRANS-Datenbank einfach zu den Monats-, Quartals- und Jahresbeträgen addiert. Kennzeichnet die Aktualisierung den Beginn eines neuen Monats, dann müssen die vorhandenen Monatssummen auf Null gesetzt werden, bevor die Beträge der Eingaben dazukommen. Das gleiche gilt auch für die Quartals- und Jahreswerte.

Sobald der Benutzer die Option 2 vom Hauptmenü aus aufruft, beginnt die Aktualisierung. Zuerst zeigt das Programm diese Meldung:

```
Dieses Programm sollte benutzt werden, um die
Kontenaufstellung zu aktualisieren

Wollen Sie das jetzt machen? : :
```

Vorausgesetzt, der Benutzer ist nicht zufällig in dieser Option gelandet, kann er jetzt J eingeben und mit der Aktualisierung weitermachen. Auf dem Bildschirm erscheint dann:

```
Die Kontenaufstellung wurde zuletzt aktualisiert am
10/10/84
  Beginnt jetzt ein neuer Monat? : :
```

Wenn der Benutzer jetzt mit J antwortet, zeigt das Programm auch noch die Fragen nach dem Beginn eines neuen Quartals und eines neuen Jahres:

```
Die Kontenaufstellung wurde zuletzt aktualisiert am
10/10/84
  Beginnt jetzt ein neuer Monat? : :
  Beginnt jetzt ein neues Quartal?: :
  Beginnt jetzt ein neues Jahr? : :
  OK? (J/N) : :
```

Der Benutzer beantwortet dann die restlichen Fragen, und die OK-Eingabe hält die Antwort auf dem Bildschirm, damit er sich die Sache noch einmal überlegen kann. Ein Fehler an dieser Stelle kann teuer werden, aber den einzigen Schutz, den wir im Moment anbieten können ist, den

Benutzer seine Entscheidung doppelt überprüfen zu lassen, so daß er auch zweimal darüber nachdenken muß. Entdeckt er einen Fehler in seinen Angaben oberhalb der OK-Frage, kann er ein N eingeben und die Fragen werden noch einmal gestellt.

Hat der Benutzer einmal Ja gesagt, erledigt das Programm die Aktualisierung und kehrt zum Hauptmenü zurück. Das Datum dieser Aktualisierung wird auch in der Datei GENINFO gespeichert.

Die Aktualisierung des Buchhaltungssystems wird von dem Programm UPDATE.CMD durchgeführt. Der Pseudocode steht in der Abb. 16.3.

Wir werden jetzt das UPDATE-Programm entwickeln. Zuerst die üblichen Kommentare und die Möglichkeit, sofort wieder zum Hauptmenü zurückkehren zu können:

```
********************************** UPDATE.CMD
******************** Die Kontendatenbank (COA) mit
******************** den Daten TRANS aktualisieren

ERASE
? 'Dieses Programm sollte dazu benutzt werden,'
? 'die Bilanz der Kontenübersicht zu aktualisieren'
?
STORE ' ' TO jn
§ 5, 2 SAY 'Wollen Sie das jetzt machen? ' GET jn
READ

************** Zurück zum Hauptmenü, wenn gewünscht
IF !(jn) <> 'J'
   RETURN
ENDIF !(jn) <> 'J'
```

Als nächstes soll das Programm das Datum der letzten Aktualisierung aus der Datei GENINFO holen und zur Bequemlichkeit des Benutzers sowie zur Erinnerung auf dem Bildschirm anzeigen:

```
************ Hole das letzte Aktualisierungsdatum
************************ aus GENINFO
USE geninfo
STORE le:aktu  TO le:datum
```

Jetzt soll das Programm das Datum der letzten Aktualisierung anzeigen und fragen, ob heute ein neuer Monat beginnt. Trifft das zu, fragt es auch noch, ob ein neues Quartal oder ein neues Jahr beginnt. Die Abfragen

```
Zeige Eröffnungsmeldung

Wenn der Benutzer nicht aktualisieren will, zurück zum Hauptmenü

Hole Datum der letzten Aktualisierung aus GENINFO

Lösche Bildschirm
Zeige Datum der letzten Aktualisierung

Beginne Schleife mit der Frage, wie aktualisiert werden soll
   Frage, ob dies ein neuer Monat ist
   Wenn neuer Monat ...
      Frage, ob neues Quartal beginnt
      Frage, ob neues Jahr beginnt
   Wenn nicht OK, zurück und die Optionen noch einmal zeigen

Beginne Aktualisierung
Kopiere alle Datensätze, die nicht UEBETRagen sind in
Zwischendatei, und markiere diese als UEBERTR

Eröffne COA-Datei
Wenn neuer Monat, setzte alle monatlichen Werte auf Null

Wenn neues Quartal, setzte alle VTJ-Werte auf Null

Wenn neues Jahr, setzte alle JRL-Werte auf Null

Eröffne erste (COA) und zweite (TEMP) Datei für Aktualisierung

Durchlaufe jeden Datensatz der TEMP-Datei
   Finde TEMP-Kontonummer in der COA-Datei
   Erhöhe COA-Monatsbetrag um TEMP-Betrag
   Erhöhe COA-Quartalsbetrag um TEMP-Betrag
   Erhöhe COA-Jahresbetrag um TEMP-Betrag
   Weiter zum nächsten Datensatz in TEMP
Weiter mit Schleife bis Ende der TEMP-Datei

Aktualisiere GENINFO mit dem neuen Aktualisierungsdatum

Schließe Datenbanken
Lösche Variablen
Zurück zum Hauptmenü
```

Abb. 16.3: Pseudocode für das UPDATE-Programm

sind in einer Schleife plaziert, so daß der Benutzer Korrekturen vornehmen kann, wenn er etwas falsch gemacht hat. Diese Zeilen erledigen das:

```
*** Frage, wie aktualisiert werden soll, und prüfe
*** die Anwort zur Sicherheit zweimal
STORE F TO ok
DO WHILE .NOT. ok
   ERASE
   STORE ' ' TO neu:vtj,neu:mon,neu:jahr
   § 5, 2 SAY 'Kontenübersicht wurde zuletzt;
                                        aktualisiert'
   § 5,36 SAY 'am ' +le:datum
   § 8, 2 SAY 'Beginnt ein neuer Monat ?'
   § 8,38 GET neu:mon PICTURE '!'
   READ
   IF new:month= 'J'
      § 10, 2 SAY 'Beginnt ein neues Quartal ? '
      § 10,38 GET neu:vtj PICTURE '!'
      § 12, 2 SAY 'Beginnt ein neues Jahr ? '
      § 12,38 GET neu:jahr PICTURE '!'
      READ
   ENDIF (neu:mon = j)
   § 14,20 SAY 'OK ? (J/N) ' GET ok PICTURE '!'
   READ
ENDDO (WHILE NOT ok)
```

Sobald der Benutzer soweit ist, nimmt das Programm alle nicht aktualisierten Vorgänge aus der TRANS-Datenbank und speichert sie in einer Zwischendatei (TEMP). Dann markiert es alle Datensätze in der TRANS-Datei als übertragen (UEBERTR = T):

```
********** Kopiere alle nicht aktualisierten
********** Datensätze in eine Zwischendatei und
********** markiere diese Sätze als UEBETRagen
ERASE
? 'Jetzt wird aktualisiert ...'
USE trans INDEX konto
COPY TO temp FOR .NOT. uebertr
REPLACE ALL uebertr WITH T NOUPDATE
```

Wenn diese Aktualisierung den Beginn eines neuen Monats signalisiert, müssen alle monatlichen Werte in der Kontenübersicht vor der Aktualisierung auf Null gesetzt werden. Diese Zeilen bewerkstelligen das:

```
************** Setze monatliche Felder(BETRAG)
************** auf NUll, wenn neuer Monat beginnt
USE coa INDEX coa
IF neu:mon = 'J'
   REPLACE ALL betrag WITH 0 NOUPDATE
ENDIF (neu:mon = J)
```

Wenn mit dieser Aktualisierung ein neues Quartal oder ein neues Jahr beginnt, müssen die Werte in den Feldern VTJ und JRL auch auf Null gesetzt werden:

```
************** Wenn neues Quartal oder Jahr, ersetze
************** auch die Quartalsbeträge durch Null
IF neu:vtj= 'J'
   REPLACE ALL vtj WITH 0 NOUPDATE
ENDIF (neu:vtj=j)


******************** Wenn neues Jahr, ersetze alle
******************** jährlichen Beträge durch Null

IF neu:jahr= 'J'
   REPLACE ALL jrl WITH 0 NOUPDATE
ENDIF (neu:jahr=j)
```

Um diese Aktualisierung durchzuführen, läuft das Programm durch die Zwischendatei, liest jeden Datensatz und aktualisiert den entsprechenden Datensatz in der Kontenübersicht-Datenbank. Der Befehl UPDATE ist für diesen Trick nicht zu gebrauchen, weil wir die Kontrolle darüber haben müssen, wie die Werte der Felder VTJ und JRL in der Kontenübersicht verändert werden. Der UPDATE-Befehl kann das BETRAG-Feld in der COA-Datei mit dem Feld BETRAG aus der Zwischendatei aktualisieren. Dann wird es aber unmöglich, nur die entsprechenden VTJ- und JRL-Felder zu verändern.

Wir müssen beide Dateien eröffnen, um die Aktualisierung zu schaffen, weshalb diese Zeilen hier den Dateien ihren Status zuweisen:

```
******************** Datenbanken auf Aktualisierung
******************** vorbereiten
SELE PRIM
USE coa INDEX coa
SELE SECO
USE temp
```

Jetzt wird die Aktualisierung dadurch erledigt, daß jeder Datensatz in der Zwischendatei gelesen, der entsprechende Satz mit derselben Kontonummer in der COA-Datei gefunden wird und die Felder BETRAG, VTJ und JRL um die Beträge der Zwischendatei erhöht werden. Die untenstehende Schleife sorgt für die Aktualisierung:

```
********************** Aktualisiere!
DO WHILE .NOT. EOF
   STORE STR(konto,6,2) TO suchen
   SELE PRIM
   FIND &suchen
   REPLACE P.betrag WITH P.betrag+S.betrag
   REPLACE vtj WITH vtj +S.betrag
   REPLACE jrl WITH jrl + S.betrag
   SELE SECO
   SKIP
ENDDO (WHILE NOT end of SECONDARY FILE)
```

Sobald die Schleife zu Ende ist, ist auch die Aktualisierung beendet. Wir müssen nur noch die Datenbanken schließen:

```
***************** Wenn fertig, schließe Datenbanken
SELE SECO
USE
SELE PRIM
USE
```

Mit den nächsten Zeilen speichert das Programm das Datum dieser Aktualisierung in der Datei GENINFO:

```
***************** Speichere Aktualisierungsdatum in
***************** Datei GENINFO
USE geninfo
REPLACE le:aktu WITH t:datum
```

Jetzt können wir die GENINFO-Datei schließen, die Speichervariablen löschen und zum Hauptmenü zurückkehren:

```
USE
RELEASE le:datum,neu:mon,neu:vtj,neu:jahr
RELEASE suchen,jn,ok
RETURN
```

Die Abb. 16.4 zeigt das komplette UPDATE-Programm.

```
********************************** UPDATE.CMD
*********************** Die Kontendatenbank (COA) mit den Daten
*********************** TRANS aktualisieren
ERASE
? 'Dieses Programm sollte dazu benutzt werden,'
? 'die Bilanz der Kontenübersicht zu aktualisieren'
?
STORE ' ' TO jn
§ 5, 2 SAY 'Wollen Sie das jetzt machen? ' GET jn
READ

*********************** Zurück zum Hauptmenü, wenn gewünscht
IF !(jn) <> 'J'
   RETURN
ENDIF !(jn) <> 'J'

*********************** Hole das letzte Aktualisierungsdatum
*********************** aus GENINFO
USE geninfo
STORE le:aktu  TO le:datum

*** Frage, wie aktualisiert werden soll, und prüfe
*** die Anwort zur Sicherheit zweimal
STORE F TO ok
DO WHILE .NOT. ok
   ERASE
   STORE ' ' TO neu:vtj,neu:mon,neu:jahr
   § 5, 2 SAY 'Kontenübersicht wurde zuletzt aktualisiert'
   § 5,36 SAY 'am ' +le:datum
   § 8, 2 SAY 'Beginnt ein neuer Monat ?'
   § 8,38 GET neu:mon PICTURE '!'
   READ
   IF neu:mon= 'J'
      § 10, 2 SAY 'Beginnt ein neues Quartal ? '
      § 10,38 GET neu:vtj PICTURE '!'
      § 12, 2 SAY 'Beginnt ein neues Jahr ? '
      § 12,38 GET neu:jahr PICTURE '!'
      READ
   ENDIF (neu:mon = j)
   § 14,20 SAY 'OK ? (J/N) ' GET ok PICTURE '!'
   READ
ENDDO (WHILE NOT ok)

*********************** Kopiere alle nicht aktualisierten
*********************** Datensätze in eine Zwischendatei und
*********************** markiere diese Sätze als UEBETRagen
ERASE
? 'Jetzt wird aktualisiert...'
```

Abb. 16.4: Das Programm UPDATE.CMD

```
USE trans INDEX konto
COPY TO temp FOR .NOT. uebertr
REPLACE ALL uebertr WITH T NOUPDATE

********************** Setze monatliche Felder(BETRAG)
********************** auf NUll, wenn neuer Monat beginnt
USE coa INDEX coa
IF neu:mon = 'J'
   REPLACE ALL betrag WITH 0 NOUPDATE
ENDIF (neu:mon = J)

********************** Wenn neues Quartal oder Jahr, ersetze
********************** auch die Quartalsbeträge durch Null
IF neu:vtj= 'J'
   REPLACE ALL vtj WITH 0 NOUPDATE
ENDIF (neu:vtj=j)

********************** Wenn neues Jahr, ersetzte alle jährlichen
********************** Beträge durch Null
IF neu:jahr= 'J'
   REPLACE ALL jrl WITH 0 NOUPDATE
ENDIF (neu:jahr=j)

********************** Datenbanken auf Aktualisierung
********************** vorbereiten
SELE PRIM
USE coa INDEX coa
SELE SECO
USE temp

********************** Aktualisiere!
DO WHILE .NOT. EOF
   STORE STR(konto,6,2) TO suchen
   SELE PRIM
   FIND &suchen
   REPLACE P.betrag WITH P.betrag+S.betrag
   REPLACE vtj WITH vtj +S.betrag
   REPLACE jrl WITH jrl + S.betrag
   SELE SECO
   SKIP
ENDDO (WHILE NOT end of SECONDARY FILE)

********************** Wenn fertig, schließ Datenbanken
SELE SECO
USE
SELE PRIM
USE
```

Abb. 16.4: Das Programm UPDATE.CMD (Forts.)

```
********************** Speichere Aktualisierungsdatum in
********************** Datei GENINFO
USE geninfo
REPLACE le:aktu WITH t:datum

USE

RELEASE le:datum,neu:mon,neu:vtj,neu:jahr
RELEASE suchen,jn,ok
RETURN
```

Abb. 16.4: Das Programm UPDATE.CMD (Forts.)

Zusammenfassung

In diesem Kapitel haben wir zwei weitere Programme für das Buchhaltungssystem entwickelt. Nämlich:

1. ADDTRANS.CMD, womit der Benutzer neue Vorgänge in das Buchhaltungssystem eingeben kann. Dieses Programm demonstriert verschiedene Techniken, die in allen Buchhaltungsprogrammen nützlich sind:
   - Überprüfen der gültigen Kontonummer
   - Automatische Anzeige des Kontonamens
   - Möglichkeiten des Scheckdrucks
   - Automatisches Numerieren der Schecks
2. UPDATE.CMD, das automatisch alle Bilanzen aller Konten entsprechend den neuen Eingaben aktualisiert. Das Programm fragt den Benutzer auch, ob die Aktualisierung den Beginn eines neuen Monats, Quartals oder Jahres markiert, und berichtigt die Kontobilanzen entsprechend.

Im nächsten Kapitel werden wir Programme entwickeln, die das Scheckregister bearbeiten.

Das Scheckregister bearbeiten

17

Das Buchhaltungssystem erlaubt es dem Benutzer, bei der Eingabe von Vorgängen anzugeben, ob ein Scheck für eine Ausgabe geschrieben wurde. Zusätzlich kann er verlangen, daß der Scheck gedruckt wird. Weiterhin bekommt der Benutzer ein Scheckregister, er kann Einzahlungen speichern, den gegenwärtigen Kontostand und das Scheckregister für einen beliebigen Zeitraum ausdrucken lassen. In diesem Kapitel werden wir die Programme entwickeln, die diese drei Scheckregister-Möglichkeiten anbieten.

Das erste Programm heißt REGISTER.CMD, speichert die Einnahmen und zeigt die Bilanz an. Die Daten für das Scheckregister werden in der Datenbank REGISTER gespeichert, die wir schon früher mit dieser Struktur erstellt haben:

```
STRUKTUR FÜR DATEI: REGISTER.DBF

Feld  Name       Typ   Weite  Dez     Bezeichnung
001   SCHECK:NR  N     004            Schecknummer
002   AN:WEN     C     020            an wen gezahlt
003   GRUND      C     020            Grund der Zahlung
004   BETRAG     N     012    002     Scheckbetrag
005   DATUM      C     008            Scheckdatum
006   EINZAHLNG  L     001            Ist es eine Einzahlung?
```

Um das Scheckregister zu bearbeiten, wählt der Benutzer die Option 3 im

Hauptmenü des Buchhaltungssystems. Daraufhin sieht er folgendes Untermenü:

```
                Scheckregister Optionen

          1.    Einnahmen eingeben
          2.    Bilanz

          3.    Zurück zum Hauptmenü
```

Wählt der Benutzer dann die Option 1, wird der Bildschirm gelöscht und frühere Eingaben erscheinen, so wie hier:

```
01/01/84      2500.00
01/02/84      5000.00
01/03/84      2500.55
05/04/84      2450.55

Neue Einnahme eingeben? (J/N)
```

Mit Hilfe dieser Anzeige kann er sehen, ob er die Einnahme vielleicht schon eingegeben hat. Wünscht er eine Eingabe zu machen, gibt er einfach J ein, worauf der Bildschirm gelöscht und diese Maske zum Eingeben der Daten gezeigt wird:

```
Datum:  /  /  :
Betrag:      .  :
Scheck von :                        :
```

Der Benutzer gibt dann die gewünschten Informationen ein, und die Einnahme wird in der Scheckregister-Datenbank gespeichert.

Wenn er die 2. Option (Bilanz) aus dem Untermenü wählt, wird der Bildschirm gelöscht und diese kurze Meldung erscheint:

```
Bilanz wird berechnet ...
```

Anschließend erscheint die Scheckbilanz auf dem Bildschirm:

```
Derzeitige Bilanz: 1234.67
```

Dann kann er mit der Option 3 des Untermenüs wieder zum Hauptmenü zurückkehren.

Einnahmen speichern, Bilanzen errechnen

Der Pseudocode für das Programm REGISTER.CMD steht in der Abb. 17.1. Es beginnt mit einer Schleife für das Menü und dem Warten auf eine Anwort:

```
************************ REGISTER.CMD
******************** Einnahmen verbuchen, Bilanz für
******************** Scheckregister erstellen
STORE ' ' TO rwahl
DO WHILE rwahl # '3'
   ERASE
   TEXT

                Scheckregister-Optionen

           1.   Einnahmen eingeben
           2.   Bilanz

           3.   Zurück zum Hauptmenü

ENDTEXT
§ 8,15 SAY 'Ihre Wahl (1-3) ' ;
GET rwahl
READ
```

Sobald der Benutzer seine Wahl eingegeben hat, fängt eine DO CASE-Anweisung mit ihrer Arbeit an. Wenn er die Option 1 gewählt hat, öffnet die erste CASE-Anweisung die Datenbank REGISTER.DBF und listet alle vorhandenen Einnahmen auf (das sind die Datensätze mit einem T in dem logischen Feld EINZAHLNG):

```
************************** Reagiere auf Eingabe
  DO CASE

     CASE rwahl = '1'
        *************** Einnahme eingeben
        ERASE
        USE register
        *************** Zeige vorhandene Einnahmen
        ? 'Gespeicherte Einnahmen ...'
        ?
        LIST OFF FOR einzahlng datum,betrag
        ?
```

```
Beginne Schleife zum Anzeigen des Menüs
    Bildschirm löschen
    Menü anzeigen
    Auf Eingabe warten

    Entsprechende Arbeit ausführen

    Wenn Benutzer eine Einzahlung eingeben möchte ...
        Bildschirm löschen
        Benutze Datenbank REGISTER
        Zeige vorhandene Einzahlungen
        Beginne Schleife zum Eingeben von Einzahlungen
            Hole Daten der Einzahlung vom Benutzer
            Frage, ob noch mehr Einzahlungen eingegeben werden
        Weiter mit der Schleife, wenn Benutzer mehr
        Einzahlungen eingibt

    Wenn der Benutzer die Bilanz sehen möchte
        Summiere die Beträge der Einzahlungen
        Summiere die Beträge der Schecks
        Zeige Bilanz (Einzahlungen-Schecks)
        Bildschirm anhalten

Weiter mit der Menü-Schleife (bis Ende gewünscht)

Lösche die Variablen

Zurück zum Hauptmenü
```

Abb. 17.1: Pseudocode für das Programm REGISTER.CMD

Als nächstes beginnt das Programm eine Schleife zur Eingabe von Einnahmen. Es fragt immer weiter nach den Daten der Einnahmen, bis der Benutzer aufhören möchte. Mit diesen Zeilen ist das möglich:

```
STORE j TO regein
  DO WHILE regein

     ? 'Weiter Einnahme eingeben? (J/N)'
     WAIT TO jn
```

Sobald der Benutzer eine Einnahme eingeben will, fügt das Programm

einen leeren Datensatz an die Datenbank REGISTER.DBF an, schreibt ein T in das Feld EINZAHLNG und „Einnahme" in das Feld AN:WEN, bevor es nach weiteren Angaben fragt. Die restlichen Zeilen schließen die IF- und DO WHILE-Schleifen in der CASE-Anweisung:

```
IF !(jn)= 'J'
     APPEND BLANK
     REPLACE einzahlng WITH T
     REPLACE an:wen WITH 'Einnahme'
     ERASE
     § 5, 2 SAY 'Datum ' ;
            GET datum PICTURE '99/99/99'
     § 7, 2 SAY 'Betrag ' GET betrag
     § 9, 2 SAY 'Scheck von ' ;
            GET grund
      READ
   ELSE
      STORE F TO regein
   ENDIF (WHILE regein)
ENDDO (WHILE regein)
```

Wenn der Benutzer die Bilanz zu sehen wünscht, summiert das Programm alle Beträgefelder für die Einnahmen in die Variable EINSUM. Dann summiert es alle Scheckbeträge (.NOT. EINZAHLNG) in die Variable SCHECKEIN und zeigt die Differenz auf dem Bildschirm. Bevor es dann wieder zum Untermenü zurückkehrt, wartet es, bis der Benutzer eine Taste gedrückt hat:

```
CASE rwahl = '2'
***************** berechne und zeige die Bilanz
      ERASE
      USE register
      ? 'Bilanz wird berechnet....'
      SUM betrag FOR einzahlng TO einsum
      SUM betrag FOR .NOT. einzahlng TO scheckein
      § 10,10 SAY 'Derzeitige Bilanz ' +;
              STR(einsum-scheckein,9,2)
      ?
      ?
      ? 'Weiter mit beliebiger Taste ...'
      WAIT

ENDCASE
```

```
********************************** REGISTER.CMD
****************************** Einnahmen verbuchen, Bilanz für
****************************** Scheckregister erstellen
STORE ' ' TO rwahl
DO WHILE rwahl # '3'
   ERASE
   TEXT

               Scheckregister-Optionen

          1.   Einnahmen eingeben
          2.   Bilanz

          3.   Zurück zum Hauptmenü

   ENDTEXT
   § 8,15 SAY 'Ihre Wahl (1-3) ' ;
          GET rwahl
   READ

 ************************ Reagiere auf Eingabe
   DO CASE

      CASE rwahl = '1'
         ************************ Einnahme eingeben
         ERASE
         USE register
         ************************ Zeige vorhandene Einnahmen
         ? 'Gespeicherte Einnahmen ...'
         ?
         LIST OFF FOR einzahlng datum,betrag
         ?
         STORE 'J' TO regein
         DO WHILE regein

            ? 'Weiter Einnahme eingeben? (J/N)'
            WAIT TO jn

            IF !(jn)= 'J'
               APPEND BLANK
               REPLACE einzahlng WITH T
               REPLACE an:wen WITH 'Einnahme'
               ERASE
               § 5, 2 SAY 'Datum ' ;
                      GET datum PICTURE '99/99/99'
               § 7, 2 SAY 'Betrag ' GET betrag
               § 9, 2 SAY 'Scheck von ' ;
                      GET grund
```

Abb. 17.2: Das Programm REGISTER.CMD

```
            READ
         ELSE
            STORE F TO regein
         ENDIF (WHILE regein)
      ENDDO (WHILE regein)

   CASE rwahl = '2'
      ************************ Berechne und zeige die Bilanz
      ERASE
      USE register
      ? 'Bilanz wird berechnet ...'
      SUM betrag FOR einzahlng TO einsum
      SUM betrag FOR .NOT. einzahlng TO scheckein
      § 10,10 SAY 'Derzeitige Bilanz ' +;
              STR(einsum-scheckein,9,2)
      ?
      ?
      ? 'Weiter mit beliebiger Taste ...'
      WAIT

ENDCASE

ENDDO (WHILE rwahl  #3)0    71:

RELEASE rwahl,regein,jn,einsum,scheckein
RETURN
```

Abb. 17.2: Das Programm REGISTER.CMD (Forts.)

Will der Benutzer dann zum Hauptmenü zurück, ist die Menüschleife beendet. Darum werden dann die Variablen gelöscht, und das Programm übergibt die Kontrolle wieder dem Hauptmenü:

```
ENDDO (WHILE rwahl # 3)

RELEASE rwahl,regein,jn,einsum,scheckein
RETURN
```

Das vollständige Programm REGISTER.CMD ist in der Abb. 17.2 zu sehen.

Das Scheckregister drucken

Die vierte Option des Buchhaltungssystems gestattet das Drucken von Berichten. Angewählt, bringt sie dieses Untermenü:

```
Buchhaltung Reportoptionen

1.   Aktuelle Vorgänge
2.   Kontenliste
3.   Schecks drucken
4.   Scheckregister

5.   Zurück zum Hauptmenü
```

Wenn der Benutzer die Option 4 wählt, zeigt der Bildschirm kurz diese Nachricht:

```
Registerdatei wird aktualisiert ...
```

und bringt darauf die Frage nach dem Zeitraum für das auszudruckende Register und fragt, ob der Bericht ausgedruckt werden soll. Die Fragen sehen so aus:

```
Anfangsdatum:  /  /  :
Enddatum    :  /  /  :
Daten ausdrucken? (J/N): :
```

Der Benutzer kann nun einen Zeitraum eingeben und sich den Bericht ausdrucken lassen. Das Programm druckt dann das Register und die Bilanz für Einnahmen und Schecks innerhalb des gewählten Zeitraums:

```
Seite Nr. 0001
31/03/84

                    Scheckregister

Scheck        Gezahlt an       Grund         Betrag      Datum
 Nr.
100           Deutsche Bank    Leasing        999.99     15/01/84
101           KFZ-Bank         Auto           223.13     15/01/84
102           KFZ-Vers.        Haftpflicht    222.22     15/01/84
103           ABC              Sportwagen    6543.21     01/02/84
0             Einnahme         Zinsen         123.00     15/02/84
0             Einnahme         Gehalt        5000.00     15/02/84
0             Einnahme         Kreditrückz.   234.56     01/03/84
104           Deutsche Bank    Hypothek       600.00     01/03/84
0             Einnahme         Gehalt        5555.55     05/03/84

              Bilanz                         2324.56
```

Dieser Bericht wurde von einer REPORT FORM-Datei mit dem Namen REGISTER.FRM erstellt, die mit den Befehlen

```
USE register
REPORT FORM register
```

hinter dem dBASE-Prompt eröffnet wurde und diesen Inhalt hat:

```
ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE,
W=PAGE WIDTH M=1
PAGE HEADING? (Y/N) Y
ENTER PAGE HEADING:  Scheckregister
DOUBLE SPACE REPORT? (Y/N) N
ARE TOTALS REQUIRED? (Y/N) N
COL      WIDTH,CONTENTS
001      6,SCHECK:NR
ENTER HEADING: >Scheck;nr
002      20,AN:WEN
ENTER HEADING: <Gezahlt an
003      20,GRUND
ENTER HEADING: <Grund
004      12,BETRAG
ENTER HEADING: <Betrag
005      8,DATUM
ENTER HEADING: <Datum
006
```

Das REGREPT.CMD-Programm benutzt die Formatdatei REGISTER.FRM, um den Bericht zu drucken. Der Pseudocode für das Programm steht in der Abb. 17.3.

Das REGREPT-Programm beginnt seine Arbeit mit dem Löschen des Bildschirms und der Meldung „Registerdatei wird aktualisiert", da die Aktualisierung mit Hilfe der TRANS-Datenbank einige Zeit in Anspruch nehmen kann:

```
********************************** REGREPT.CMD
********************** Aktualisieren und Drucken
********************** des Scheckregisters
ERASE
? 'Registerdatei wird aktualisiert ...'
```

Bevor das Scheckregister jetzt gedruckt werden kann, müssen alle Datensätze in der TRANS-Datei, die geschriebene, aber noch nicht an REGISTER geschickte Schecks enthalten, in die Datenbank REGISTER addiert werden. Wir können diesen Vorgang einleiten, indem wir alle nicht registrierten, geschriebenen Scheckzahlungen aus der TRANS-Datei in eine Zwischendatei kopieren. Dann markieren wir sie als registriert. Diese Zeilen bewerkstelligen das:

Bildschirm löschen
Datenbank TRANS benutzen
Schicke alle geschriebenen, aber nicht registrierten Scheck-
    Vorgänge aus TRANS in eine Zwischendatei (TEMP)
Markiere alle registrierten Schecks in TRANS als registriert

Aktualisiere REGISTER mit den Daten aus TEMP

Summiere Schecks und Einnahmen

Frage nach Datumsgrenzen und nach Drucker

Wenn gewünscht, Drucker einschalten

Bericht drucken, REGISTER.FRM benutzen
Bilanz ausdrucken

Drucker ausschalten
Variablen löschen
Zurück zum Hauptmenü

Abb. 17.3: Pseudocode für das Programm REGREPT.CMD

```
****************** Schicke alle geschriebenen, aber
****************** nicht registrierten Schecks aus
****************** der TRANS-Datei in eine
******************Zwischendatei (TEMP)
USE trans
COPY TO temp FOR geschr .AND. .NOT. ;
                                  register .AND. typ=1
REPLACE ALL register WITH T FOR geschr .AND. .NOT. ;
                                  register .AND. typ=1
```

Nun kann das Programm die REGISTER-Datei benutzen und die Datensätze aus der Zwischendatei anhängen. Dabei werden nur Feldnamen angehängt, die in beiden Datenbanken identisch sind (SCHECK:NR, AN:WEN, GRUND, BETRAG und DATUM). Deshalb müssen wir nur die REGISTER-Datenbank in Gebrauch nehmen (USE) und mit dem Befehl APPEND FROM TEMP die Daten übernehmen:

```
***************************** Aktualisiere REGISTER
***************************** mit Daten von TEMP
USE register
APPEND FROM temp
```

Als nächstes muß das Programm die Beträge für die Schecks und die Einnahmen in der REGISTER-Datenbank summieren. Die Summe der Schecks wird in der Variablen AUSGABE und die der Einnahmen in EINNAHME gespeichert:

```
******************** Summiere Schecks und Einnahmen
? 'Berechne Bilanz ...'
SUM betrag FOR .NOT. einzahlng TO ausgabe
SUM betrag FOR einzahlng TO einnahme
```

Jetzt kann das Programm den Benutzer nach dem Zeitraum für den Ausdruck fragen und ob der Bericht gedruckt werden soll oder nicht:

```
******************** Frage nach Zeitraum und Druck
ERASE
STORE '         ' TO start,ende
STORE ' ' TO jn,beding

§ 3, 2 SAY 'Anfangsdatum ' GET start PICTURE
                                          '99/99/99'
§ 4, 2 SAY 'Enddatum ' GET ende PICTURE '99/99/99'
§ 6, 2 SAY 'Bericht drucken? (J/N) ' GET jn
                                          PICTURE '!'
READ
```

Wenn der Benutzer gern drucken lassen möchte, schaltet das Programm den Drucker ein:

```
***************************** Schalte Drucker ein,
***************************** wenn gewünscht
IF jn= 'J'
   SET PRINT ON
ENDIF jn= 'J'
SET EJECT OFF
```

Jetzt druckt das Programm den Bericht und benutzt dazu die Formatdatei REGISTER.FRM. Dabei werden nur die Datensätze angezeigt, deren Datum größer oder gleich dem eingegebenen Anfangsdatum und kleiner oder gleich dem Enddatum sind:

```
****************************** Drucke Bericht,
****************************** benutze REGISTER.FRM
ERASE
REPO FORM register FOR datum >= start .AND.;
                                    datum <= ende
```

Dann druckt es eine Reihe von Leerzeilen, und die Bilanz und schaltet den Drucker aus:

```
?
?
? '   Bilanz          ' ,einnahme-ausgabe
SET PRINT OFF
```

Geht der Bericht nicht auf den Drucker, sollte der Bildschirm anhalten, damit der Benutzer die Daten lesen kann:

```
************************** Wenn nicht zum Drucker,
************************** dann anhalten
IF jn # 'J'
   ?
   ?
   ? 'Weiter mit beliebiger Taste ...'
   WAIT
ENDIF (jn # j)
```

Schließlich löscht das Programm die Variablen und kehrt zum Hauptmenü zurück:

```
****************************** Lösche Variablen und
****************************** zurück zum Hauptmenü
RELEASE ende,einnahme,ausgabe,jn,start
RETURN
```

Das fertige Programm REGREPT.CMD sehen Sie in der Abb. 17.4.

```
********************************* REGREPT.CMD
***************************** Aktualisieren und drucken
***************************** des Scheckregisters
ERASE
? 'Registerdatei wird aktualisiert ...'

***************************** Schicke alle geschriebenen, aber
****************************** nicht registrierten Schecks aus
```

Abb. 17.4: Das Programm REGREPT.CMD

```
****************************** der TRANS-Datei in eine
****************************** Zwischendatei (TEMP)
USE trans
COPY TO temp FOR geschr .AND. .NOT. ;
          register .AND. typ=1
REPLACE ALL register WITH T FOR geschr .AND. .NOT. ;
          register .AND. typ=1
****************************** Aktualisiere REGISTER mit Daten
****************************** von TEMP
USE register
APPEND FROM temp

****************************** Summiere Schecks und Einnahmen
? 'Berechne Bilanz ...'
SUM betrag FOR .NOT. einzahlng TO ausgabe
SUM betrag FOR einzahlng TO einnahme

****************************** Frage nach Zeitraum und Druck
ERASE
STORE '         ' TO start,ende
STORE ' ' TO jn,beding

§ 3, 2 SAY 'Anfangsdatum ' GET start PICTURE '99/99/99'
§ 4, 2 SAY 'Enddatum ' GET ende PICTURE '99/99/99'
§ 6, 2 SAY 'Bericht drucken? (J/N) ' GET jn PICTURE '!'
READ

****************************** Schalte Drucker ein, wenn
****************************** gewünscht
IF jn= 'J'
   SET PRINT ON
ENDIF jn= 'J'
SET EJECT OFF

****************************** Drucke Bericht,
****************************** benutze REGISTER.FRM
ERASE
REPO FORM register FOR datum >= start .AND.;
          datum <= ende

?
?
? '   Bilanz          ' ,einnahme-ausgabe
SET PRINT OFF

****************************** Wenn nicht zum Drucker, dann
****************************** anhalten
```

Abb. 17.4: Das Programm REGREPT.CMD (Forts.)

```
IF jn # 'J'
   ?
   ?
   ? 'Weiter mit beliebiger Taste ...'
   WAIT
ENDIF (jn # j)

*************************** Lösche Variablen und zurück
*************************** zum Hauptmenü
RELEASE ende,einnahme,ausgabe,jn,start
RETURN
```

Abb. 17.4: Das Programm REGREPT.CMD (Forts.)

Zusammenfassung

In diesem Kapitel haben wir die Programme zum Bearbeiten des Scheckregisters entwickelt:

1. REGISTER.CMD, das dem Benutzer die Eingabe von Einnahmen und die Ausgabe der aktuellen Bilanz ermöglicht.
2. REGREPT.CMD, womit die Scheckregisterdatei mit den Einnahme- und Ausgabedaten der TRANS-Datei aktualisiert wird. Darüber hinaus ermöglicht es dem Benutzer, das Scheckregister und die aktuelle Bilanz anzuzeigen. Dieser kann dabei angeben, ob nur Schecks eines bestimmten Zeitraums angezeigt werden sollen.

Im nächsten Kapitel werden wir Programme zur Anzeige weiterer Buchhaltungsberichte, einschließlich der Einnahmenliste und der Kontenliste mit Kontensummen, entwickeln. Wir entwickeln außerdem ein Programm, das Schecks ausdruckt.

Die Berichte des Buchhaltungssystems

18

In diesem Kapitel werden wir folgende Programme entwickeln:

1. zum Schreiben von Schecks,

2. zum Ausdrucken des aktuellen Einnahmen-/Ausgaben-Reports,

3. zum Ausdrucken des Kontenrahmens mit Bilanzen für jeden Monat, jedes Vierteljahr und jedes Jahr.

Diese Programme, einschließlich des im letzten Kapitel entwickelten REGREPT.CMD, werden von dem Programm BREPORTS.CMD aufgerufen. BREPORTS wiederum wird von dem Hauptmenüprogramm BOOKS.CMD aufgerufen. Die Beziehungen zwischen den Programmen sind in Abb. 18.1 dargestellt.

Buchhaltungssystem Berichtsmenü

Wenn der Benutzer die Option 4 (Berichte drucken) im Hauptmenü wählt, präsentiert das Programm BREPORTS das folgende Untermenü für die Druckoptionen:

```
        Buchhaltung-Reportoptionen

1.      Einnahmen-/Ausgaben-Übersicht
2.      Kontenübersicht
```

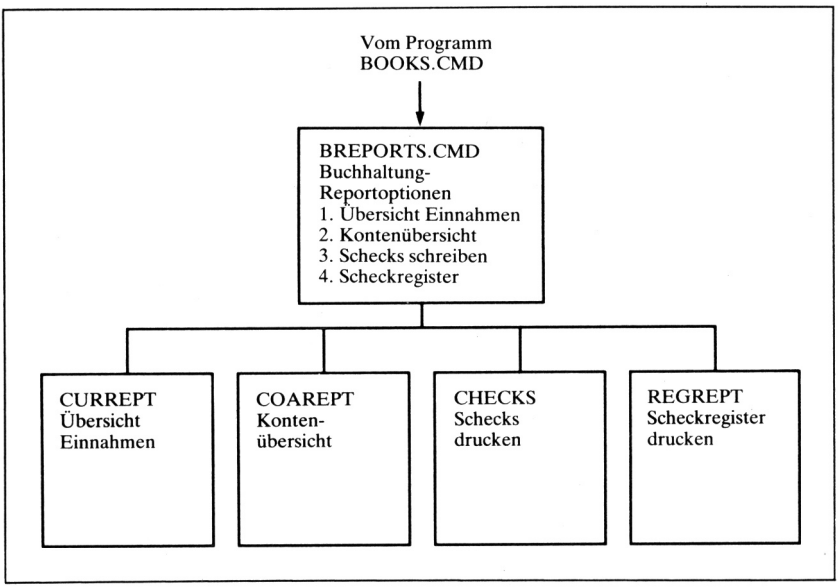


Abb. 18.1: Die Programmstruktur der Berichtsprogramme

```
3.   Schecks schreiben
4.   Scheckregister

5.   Zurück zum Hauptmenü
```

BREPORTS ist ein einfaches Menüprogramm; Sie können es in Abb. 18.2 sehen.

Einnahmen-/Ausgaben-Übersicht

Die Übersicht über Einnahmen und Ausgaben ermöglicht es dem Benutzer, die Vorgänge des aktuellen Monats oder eines beliebigen Zeitraums zu drucken. Zusätzlich kann er sich aussuchen, ob die Aufstellung in der Reihenfolge der Kontonummern oder der Termine erfolgen soll. Sobald sich der Benutzer entschließt, diesen Report zu drucken, erscheint auf dem Bildschirm:

```
Nur die Vorgänge dieses Monats? : :
```

Stimmt er hier mit Ja zu, werden nur die Vorgänge des einen Monats

```
****************************************** BREPORTS.CMD
******************** Frage, welchen Bericht der Benutzer
******************** wünscht, und verzweige entsprechend
STORE 0 TO repwahl
DO WHILE repwahl <> 5
   ERASE
   TEXT
               Buchhaltung Reportoptionen

          1.   Einnahmen/Ausgaben Übersicht
          2.   Kontenübersicht
          3.   Schecks schreiben
          4.   Scheckregister

          5.   Zurück zum Hauptmenü

   ENDTEXT
   § 12,10 SAY "Ihre Wahl (1-5)";
           GET repwahl PICT "9"
   READ

   DO CASE

      CASE repwahl=1
           DO currept

      CASE repwahl=2
           DO coarept

      CASE repwahl=3
           DO checks

      CASE repwahl=4
           DO regrept

   ENDCASE
ENDDO (while repwahl <> 5)
RETURN
```

Abb. 18.2: Das Programm BREPORTS.CMD

gedruckt. Antwortet er mit Nein, fragt der Bildschirm nach einem Zeitraum, für den Daten ausgedruckt werden sollen:

```
Anfangsdatum:  /  /  :
Enddatum    :  /  /  :
```

Sobald der Benutzer einen Zeitraum eingegeben hat, erscheint ein Menü zur Auswahl der Sortierung:

```
Sortierung

1. Nach Kontonummern
2. Nach Datum

Ihre Wahl (1-2): :
```

Der Benutzer kann sich die Art der Sortierung aussuchen. Soll es nach Kontonummern gehen, wird der Bericht in dieser Sortierung, mit den Kontonummern ganz links stehend, ausgedruckt. Wenn er lieber einen bestimmten Zeitraum ausdrucken lassen möchte, werden die Einträge in chronologischer Reihenfolge ausgedruckt, wobei das Datum jeweils ganz links steht, wie in Abb. 18.3 gezeigt.

Zwei Formatdateien werden für den Ausdruck des Berichts gebraucht. KONTO.FRM druckt den Bericht mit den Kontonummern links und wurde mit Eingabe dieser Befehle erstellt:

```
USE trans
REPORT FORM konto
```

Die gestellten Fragen wurden so beantwortet:

```
ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE, W=PAGE
WIDTH M=1
PAGE HEADING? (Y/N) N
DOUBLE SPACE REPORT? (Y/N) N
ARE TOTALS REQUIRED? (Y/N) N
COL      WIDTH,CONTENTS
001      8,DATUM
ENTER HEADING: Datum
002      6,KONTO
ENTER HEADING: Konto
003      20,GRUND
ENTER HEADING: Bezeichnung
004      10,BETRAG
ENTER HEADING: >Betrag
005      1,MERKER
ENTER HEADING:
006
```

DATUM.FRM ist ziemlich ähnlich aufgebaut, nur stehen die Termine in der linken Spalte. Um den Bericht zu erstellen, geben Sie den Befehl REPORT FORM datum hinter den dBASE-Prompt ein und beantworten die Fragen so:

Einnahmen/Ausgaben für Elmar A. Weiler

Für Buchungen zwischen dem 01/01/84 und dem 28/02/84

EINNAHMEN:

Datum	Konto	Beschreibung	Betrag
15/01/84	100.00	Gehalt	5555.55
15/01/84	120.00	Zinsen	500.00
01/02/84	100.00	Gehalt	1500.00
Einnahmen	TOTAL		7555.55

AUSGABEN:

Datum	Konto	Beschreibung	Betrag
15/01/84	300.00	Lohnsteuer	999.99
15/01/84	300.10	Hauszinsen	400.00*
15/01/84	300.20	Hypothekabtrag	500.00*
15/01/84	400.00	Autokredit	223.13
15/01/84	500.00	Versicherung	123.45
15/01/84	500.00	Versicherung	222.22
15/02/84	600.00	Essen bei Pierre	17.50
15/02/84	600.00	Essen mit Pierre	32.12
28/02/84	300.00	Hausfinanzierung	600.00
Ausgaben	TOTAL		2218.41

Einnahmen	TOTAL	7555.55
Ausgaben	TOTAL	2218.41

Bilanz		5537.14

ACHTUNG: Unterkonten sind mit * markiert und in den Gesamtsummen nicht enthalten!

Abb. 18.3: Beispiel für eine Einnahmen-/Ausgaben-Übersicht

```
ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE, W=PAGE
WIDTH M=1
PAGE HEADING? (Y/N) N
```

```
DOUBLE SPACE REPORT? (Y/N) N
ARE TOTALS REQUIRED? (Y/N) N
COL     WIDTH,CONTENTS
001     8,DATUM
ENTER HEADING: Datum
002     6,KONTO
ENTER HEADING: Konto
003     20,GRUND
ENTER HEADING: Bezeichnung
004     10,BETRAG
ENTER HEADING: >Betrag
005     1,MERKER
ENTER HEADING:
006
```

Das Programm CURREPT benutzt eine dieser Formatdateien, um die Übersicht nach Daten der TRANS.DBF-Datenbank in der Reihenfolge der Kontonummern oder der Termine zu drucken, je nachdem, welche Version der Benutzer wünscht. Der Pseudocode für das CURREPT-Programm steht in der Abb. 18.4.

Wir beginnen mit der Entwicklung des CURREPT-Programms, indem wir uns den Firmennamen aus der Datei GENINFO holen und ihn in eine Variable namens FIRMA speichern:

```
********************************** CURREPT.CMD
************ Drucke Bericht der aktuellen Buchungen

************ Hole Firmenname aus GENINFO
USE geninfo
STORE firma TO firma
```

Dann schreiben wir die nächsten Zeilen, mit denen das Programm den Benutzer fragt, ob es nur die aktuellen Buchungen ausdrucken soll:

```
**************** Hole Informationen über Termine und
**************** speichere in Variable (beding)
ERASE
STORE ' ' TO jn
§ 5, 2 SAY 'Nur die Buchungen für diesen Monat ? ' ;
       GET jn
READ
```

Wenn der Benutzer mit J auf diese Frage antwortet, speichert das Programm die Wörter „datum = " und den Monat (die ersten beiden Zeichen

Hole den Firmennamen aus der Hauptdatei als Titel

Frage, ob nur Vorgänge aus diesem Monat gedruckt werden sollen
Wenn nicht, frage nach dem Zeitraum, für den Vorgänge gedruckt werden sollen
Eröffne ein Makro für die Suche

Zeige die Sortiermöglichkeiten und hole die gewünschte Sortierung
Wähle die gewünschte Sortierung und den entsprechenden Report

Frage, ob Daten gedruckt werden sollen, und schalte eventuell Drucker ein

Kennzeichne Untersummen für die Anzeige im Report

Berechne die Summe der Einnahmen ohne Untersummen
Berechne die Summe der Ausgaben ohne Untersummen

Drucke den Titel und den entsprechenden Untertitel
Drucke die Einnahmen und die Gesamtsummen
Drucke die Ausgaben und die Gesamtsummen
Zeige die Bilanz am Ende des Blattes
Drucke die Meldung über die Untersummen

Blattvorschub auf dem Drucker
Drucker ausschalten

Wenn der Bericht nicht auf den Drucker geht, Datenausgabe anhalten

Variablen löschen
Zurück zu BREPORTS

Abb. 18.4: Pseudocode für das Programm CURREPT

in der Variablen T:DATUM) in die Variable BEDING und ein „unwahr" (F) in die Variable ABSCHNITT:

```
**************** Formuliere Suchbedingung
**************** entsprechend den Benutzereingaben
IF !(jn)= 'J'
   STORE "datum = '" +$(t:datum,1,2)+ "'" TO beding
   STORE F TO abschnitt
```

Möchte der Benutzer den Bericht für einen bestimmten Zeitabschnitt drucken, fragt das Programm nach dem Anfangsdatum (START) und dem Enddatum (ENDE) des ABSCHNITTes. Dann speichert es die Wörter "datum>= start .AND. datum <=ende" in die BEDING-Variable und ein „wahr" in die Variable ABSCHNITT, weil jetzt für einen bestimmten Zeitabschnitt gedruckt wird. Die folgenden Zeilen übernehmen das:

```
ELSE
   STORE '          ' TO start,ende
   § 7,2 SAY 'Anfangsdatum ' ;
         GET start PICTURE '99/99/99'
   § 9,2 SAY 'Enddatum ' ;
         GET ende PICTURE '99/99/99'
   READ
   STORE 'datum >= start .and. datum <= ende' TO
                                                  beding
   STORE T TO abschnitt
ENDIF (jn=j)
```

Die Variable BEDING wird als Makro benutzt, um Gesamtsummen zu errechnen und zu entscheiden, welche Buchungen gedruckt werden sollen. Die Variable ABSCHNITT sucht die richtige Überschrift des Berichts aus.

Als nächstes fragt das Programm nach der Sortierung, um die Buchungen anzeigen zu können:

```
************************ Dann hole Sortieranweisung
ERASE
STORE 0 TO swahl
§ 3,10 SAY 'Sortierung'
§ 5,8 SAY ' 1. Nach Kontonummer'
§ 6,8 SAY ' 2. Nach Datum'
§ 8,8 SAY ' Ihre Wahl (1-2) ' ;
       GET swahl PICTURE '9'
READ
```

Wählt der Benutzer die Option 2, benutzt das Programm die Datenbank TRANS mit dem Index DATUM und speichert den REPORT-Namen „DATUM" in die Variable ROPTION. Möchte der Benutzer den Bericht in der Reihenfolge der Kontonummern haben, benutzt das Programm die TRANS-Datei mit dem Index KONTO und speichert „KONTO" in die Variable ROPTION. Später wird ROPTION als Makro benutzt, um den richtigen Reportnamen zu finden:

```
*************** Bestimme Sortierung und Reporttyp
IF swahl = 2
   USE trans INDEX datum
   STORE 'datum' TO roption
ELSE
   USE trans INDEX konto
   STORE 'konto' TO roption
ENDIF (swahl = 2)
```

Dann fragt das Programm nach dem Drucker und schaltet ihn gegebenenfalls ein:

```
************************ Nach Drucker fragen
ERASE
STORE ' ' TO jn
§ 5,2 SAY 'Bericht ausdrucken ? ' GET jn
READ

************************ und einschalten wenn nötig
IF !(jn)= 'J'
   SET PRINT ON
ENDIF (jn=j)
```

Um alle Unterkonten mit einem Stern zu versehen, muß das Programm in jedes MERKER-Feld der Unterkonten in der TRANS-Datei einen Stern schreiben. Um ganz sicherzugehen, werden erst alle existierenden Sternchen gelöscht, und dann wird in jedes Unterkontofeld (das sind die mit einem anderen Wert als Null in den Dezimalstellen) ein Stern geschrieben:

```
************** Markiere Unterkonten für den Bericht
REPLACE NOUPDATE ALL merker WITH ' '
REPLACE NOUPDATE ALL merker WITH '*' ;
FOR konto <> int(konto)
```

Dann errechnet das Programm die Gesamtsummen der Buchungen für den angegebenen Zeitraum unter Ausschluß der Unterkontenbeträge. Es berechnet die Einnahmen und Ausgaben einzeln, wie hier zu sehen ist:

```
***************** Berechne Gesamtsummen,
***************** ohne Unterkontenbeträge
SUM betrag FOR konto<300 .AND. &beding .AND.;
int(konto)=konto TO einnahme
SUM betrag FOR konto>=300 .AND. &beding .AND.;
int(konto)=konto TO ausgabe
```

Jetzt kann es den Bericht drucken. Bevor es aber anfängt, zeigt es die Überschrift mit dem Firmennamen:

```
********************* Drucke den Bericht, benutze
********************* vordefinierte Report-Dateien
SET EJECT OFF
ERASE
? '                Einnahmen/Ausgaben-Übersicht für;
                                                &firma'
?
```

Wird der Report für einen bestimmten Zeitraum gedruckt (ABSCHNITT ist wahr (T)), druckt es das Anfangs- und Enddatum im Untertitel, andernfalls wird das aktuelle Datum (T:DATUM) gedruckt:

```
************** Drucke den entsprechenden Untertitel
IF abschnitt
   ? 'Für Buchungen zwischen dem &start und  &ende'
ELSE
   ? 'für den &t:datum'
ENDIF (abschnitt)
```

Das Programm druckt dann die Einnahmebuchungen mit Hilfe der entsprechenden Formatdatei (gespeichert in der Variablen ROPTION) und Suchbedingungen (gespeichert in BEDING). Wenn es fertig ist, zeigt es die Gesamtsumme der Einnahmen:

```
************************ Drucke Einnahmebuchungen
?
? 'EINNAHMEN:'
REPORT FORM &roption FOR konto<300 .AND. &beding;
                                                PLAIN
? 'Einnahmen TOTAL                              ' +;
    STR(einnahme,10,2)
?
```

Dann benutzt es eine identische Prozedur, um die Ausgaben zu drucken:

```
************************ Drucke Ausgabebuchungen
? 'AUSGABEN:'
REPORT FORM &roption FOR konto>= 300 .AND. &beding;
                                                PLAIN
? 'Ausgaben TOTAL                               ' +;
    STR(ausgabe,10,2)
***************** Zeige Bilanz am Fuße des Blattes
?
```

Am Ende des Berichts druckt es die Bilanz und eine Bemerkung zu den Unterkonten:

```
? 'Einnahmen TOTAL                                    ' +;
    STR(einnahme,10,2)
? 'Ausgaben TOTAL                                     ' +;
    STR(ausgaben,10,2)
? '                                                 ---------
'
? 'Bilanz                                             ' +;
    STR(einnahme-ausgabe,10,2)
?
? 'ACHTUNG: Unterkonten sind mit einem * markiert'
?? 'und in den Gesamtsummen nicht enthalten!'
EJECT
SET PRINT OFF
```

Wurde der Bericht nur auf dem Bildschirm angezeigt, hält das Programm die Ausgabe an, damit der Benutzer die Daten lesen kann. Dann löscht es die Speichervariablen und kehrt zum Programm BREPORTS zurück:

```
**************** Wenn Bericht nicht gedruckt wurde,
**************** Bildschirm anhalten
IF !(jn) # 'J'
   ?
   ?
   ? 'Weiter mit beliebiger Taste ...'
   WAIT
ENDIF (jn=j)

RELEASE firma,jn,beding,start,ende,abschnitt
RELEASE swahl,roption,einnahme,ausgabe
RETURN
```

Die Abb. 18.5 zeigt das CURREPT-Programm.

Bericht Kontenübersicht

Die Option 2 des Berichtsmenüs druckt die Kontenübersicht. Wenn diese Option gewählt wurde, erscheint auf dem Bildschirm:

```
Bericht ausdrucken? (J/N): :
```

```
********************************** CURREPT.CMD
************************ Drucke Bericht der aktuellen Buchungen

************************ Hole Firmenname aus GENINFO
USE geninfo
STORE firma TO firma

************************ Hole Informationen über Termine und
************************ speichere in Variable (beding)
ERASE
STORE ' ' TO jn
§ 5, 2 SAY 'Nur die Buchungen für diesen Monat ? ' ;
       GET jn
READ

************************ Formuliere Suchbedingung entsprechend
************************ den Benutzereingaben
IF !(jn)= 'J'
   STORE "Datum = '" +$(t:datum,1,2)+ "'" TO beding
   STORE F TO abschnitt
ELSE
   STORE '         ' TO start,ende
   § 7,2 SAY 'Anfangsdatum ' ;
         GET start PICTURE '99/99/99'
   § 9,2 SAY 'Enddatum ' ;
         GET ende PICTURE '99/99/99'
   READ
   STORE 'datum >= start .AND. datum <= ende' TO beding
   STORE T TO abschnitt
ENDIF (jn=j)

************************ Dann hole Sortieranweisung
ERASE
STORE 0 TO swahl
§ 3,10 SAY 'Sortierung'
§ 5,8 SAY ' 1. Nach Kontonummer'
§ 6,8 SAY ' 2. Nach Datum'
§ 8,8 SAY ' Ihre Wahl (1-2) ' ;
      GET swahl PICTURE '9'
READ

************************ Bestimme Sortierung und Reporttyp
IF swahl = 2
   USE trans INDEX datum
   STORE 'datum' TO roption
ELSE
```

Abb. 18.5: Das Programm CURREPT.CMD

```
   USE trans INDEX konto
   STORE 'konto' TO roption
ENDIF swahl = 2

************************ Nach Drucker fragen
ERASE
STORE ' ' TO jn
§ 5,2 SAY 'Bericht ausdrucken ? ' GET jn
READ

************************ und einschalten wenn nötig
IF !(jn)= 'J'
   SET PRINT ON
ENDIF (jn=j)

************************ Markiere Unterkonten für den Bericht
REPLACE NOUPDATE ALL merker WITH ' '
REPLACE NOUPDATE ALL merker WITH '*' ;
FOR konto <> int(konto)

***************** Berechne Gesamtsummen, ohne Unterkontenbeträge
SUM betrag FOR konto<300 .AND. &beding .AND.;
int(konto)=konto TO einnahme
SUM betrag FOR konto>=300 .AND. &beding .AND.;
int(konto)=konto TO ausgabe

************************ Drucke den Bericht, benutze
************************ vordefinierte Report-Dateien
SET EJECT OFF
ERASE
? '                Einnahmen/Ausgaben-Übersicht für &firma'
?

********************** Drucke den entsprechenden Untertitel
IF abschnitt
   ? 'Für Buchungen zwischen dem &start und &ende'
ELSE
   ? 'für den &t:datum'
ENDIF (abschnitt)

************************ Drucke Einnahmebuchungen
?
? 'EINNAHMEN:'
REPORT FORM &roption FOR konto<300 .AND. &beding PLAIN
? 'Einnahmen TOTAL                          ' +;
    STR(einnahme,10,2)
?
```

Abb. 18.5: Das Programm CURREPT.CMD (Forts.)

```
************************ Drucke Ausgabebuchungen
? 'AUSGABEN:'
REPORT FORM &roption FOR konto>= 300 .AND. &beding PLAIN
? 'Ausgaben TOTAL                          ' +;
    STR(ausgabe,10,2)

************************ Zeige Bilanz am Fuße des Blattes
?

? 'Einnahmen TOTAL                         ' +;
    STR(einnahme,10,2)
? 'Ausgaben TOTAL                          ' +;
    STR(ausgabe,10,2)
? '                                         ------------'
? 'Bilanz                                  ' +;
    STR(einnahme-ausgabe,10,2)
?
? 'ACHTUNG: Unterkonten sind mit "*" markiert'
?? 'und in Gesamtsummen nicht enthalten!'
EJECT
SET PRINT OFF

************************ Wenn Bericht nicht gedruckt wurde,
************************ Bildschirm anhalten
IF !(jn) # J'
   ?
   ?
   ? 'Weiter mit beliebiger Taste ...'
   WAIT
ENDIF (jn=j)

RELEASE firma,jn,beding,start,ende,abschnitt
RELEASE swahl,roption,einnahme,ausgabe
RETURN
```

Abb. 18.5: Das Programm CURREPT.CMD (Forts.)

Wird die Frage mit J beantwortet, erscheint die kurze Meldung:

```
Die Gesamtsummen werden berechnet ...
```

Danach druckt das Programm die Kontenübersicht einschließlich der Bilanzen, wie in Abb. 18.6 zu sehen ist.

Zum Erstellen des Berichts wird die Formatdatei COA.FRM benutzt. Sie wurde mit diesen Befehlen erstellt:

```
USE coa
REPORT FORM coa
```

Kontenübersicht für Elmar Weiler

Aktualisiert am 01/02/84

EINNAHMEN:

Konto Nr	Bezeichnung	Monat	Quartal	Jahr
100.00	Einkommen	22166.10	38776.65	38776.65
110.00	Lizenzeinn.	0.00	0.00	0.00
120.00	Zinseinn.	500.00	1000.00	1000.00
Einkommen	TOTAL	22666.10	39776.65	39776.65

AUSGABEN:

Konto Nr	Bezeichnung	Monat	Quartal	Jahr
300.00	Lohnsteuer	1999.99	3999.98	3999.98
310.00	Umsatzsteuer	0.00	0.00	0.00
400.00	Hausfinanzierung	7266.34	14532.68	14532.68
400.10	Kapitalabtrag	200.00	400.00	400.00*
400.20	Zinsen	300.00	600.00	600.00*
500.00	Autokredit	345.67	691.34	691.34
510.00	Versicherung	0.00	0.00	0.00
520.00	Benzin	0.00	0.00	0.00
530.00	Reparaturen	0.00	0.00	0.00
600.00	Krankheitskosten	172.62	345.24	345.24
610.00	Medizin	12.00	24.00	24.00
620.00	Zahnarzt	0.00	0.00	0.00
630.00	Krankenvers.	0.00	0.00	0.00
700.00	Bewirtung (Gesch.)	0.00	0.00	0.00
710.00	Bewirtung (Priv.)	17.50	35.00	35.00
Ausgaben	TOTAL	9814.12	19628.24	19628.24
Einnahmen	TOTAL	22666.10	39776.65	39776.65
Ausgaben	TOTAL	9814.12	19628.24	19628.24
Bilanz		12851.98	20148.41	20148.41

ACHTUNG: Unterkonten sind mit einem * markiert und in den Gesamtsummen nicht berücksichtigt!

Abb. 18.6: Ausdruck einer Kontenübersicht

Die Fragen beantworten Sie so:

```
ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE,
W=PAGE WIDTH M=1
PAGE HEADING? (Y/N) N
DOUBLE SPACE REPORT? (Y/N) N
ARE TOTALS REQUIRED? (Y/N) N
COL     WIDTH,CONTENTS
001     6,KONTO
ENTER HEADING: Konto;Nr
002     20,TITEL
ENTER HEADING: >Bezeichnung
003     12,BETRAG
ENTER HEADING: Monat
004     12,VTJ
ENTER HEADING: Quartal
005     12,JRL
ENTER HEADING: Jahr
006     1,MERKER
ENTER HEADING:
 007
```

Das Programm COAREPT.CMD benutzt die Formatdatei COA.FRM dazu, um mit der Datenbank COA.DBF die Kontenübersicht zu drukken. Der Pseudocode für das COAREPT.CMD-Porgramm steht in Abb. 18.7.

Das COAREPT.CMD-Programm beginnt, indem es das Datum der letzten Aktualisierung und den Firmennamen aus der Datei GENINFO holt, wie die ersten Zeilen des Programms zeigen:

```
********************************** COAREPT.CMD
************ Drucke Kontenübersicht und Bilanz

************ Hole Datum der letzten Aktualisierung

************ und Firmenname
USE geninfo
STORE le:aktu TO letzte
STORE firma TO firma
```

Als nächstes benutzt es die Dateien COA.DBF und COA.NDX:

```
********************** Benutze COA.DBF und COA.NDX
USE coa INDEX coa
```

Hole Datum der letzten Aktualisierung und Firmenname aus GENINFO

Benutze Daten aus COA.DBF und der Indexdatei

Frage, ob ausdrucken gewünscht

Berechne Gesamtsummen für BETRAG, VTJ und JRL für Einnahmen und Ausgaben, lasse Unterkonten weg

Markiere Unterkonten mit * für den Ausdruck des Berichts

Wenn Ausdruck gewünscht, schalte Drucker ein
Drucke Reportüberschrift
Drucke Unterzeile

Drucke Einnahmekonten
Drucke Einnahme Gesamtsumme
Drucke Ausgabekonten
Drucke Ausgabe Gesamtsumme
Zeige Bilanz am Ende des Berichts
Drucke Bemerkung zu Unterkonten
Schalte Drucker aus

Wenn nicht gedruckt wird, halte Bildschirm an

Lösche Variablen
Zurück zum Reportmenü

Abb. 18.7: Pseudocode für das Programm COAREPT.CMD

Dann fragt das Programm nach dem Drucker und speichert die Antwort in der Variablen JN:

```
********************* Frage, ob Ausdruck gewünscht
ERASE
STORE ' ' TO jn
§ 5,2 SAY 'Bericht drucken ? ' GET jn PICTURE '!'
READ
```

Das Programm berechnet dann die Gesamtsummen der Einnahmen und Ausgaben für die Bilanz des Monatsfeldes (BETRAG), des VTJ- und des

JRL-Feldes und läßt dabei die Unterkonten unberücksichtigt:

```
********** Berechne die Summen für Einnahmen und
********** Ausgaben für Monat, Quartal und Jahr,
********** berücksichtige Unterkonten nicht
ERASE
? 'Gesamtsummen werden berechnet ...'

SUM betrag FOR konto<300 .AND. int(konto)=konto ;
TO sum:mon1
SUM betrag FOR konto>=300 .AND. int(konto)=konto ;
TO sum:mon2
SUM vtj FOR konto<300 .AND. int(konto)=konto ;
TO sum:qurt1
SUM vtj FOR konto>=300 .AND. int(konto)=konto ;
TO sum:qurt2
SUM jrl FOR konto<300 .AND. int(konto)=konto ;
TO sum:jahr1
SUM jrl FOR konto>=300 .AND. int(konto)=konto ;
TO sum:jahr2
```

Jetzt müssen wir noch eine Markierung an alle Unterkonten anbringen, die dann im Bericht gedruckt wird:

```
************************* Markiere Unterkonten mit *
REPLACE NOUPDATE ALL merker WITH ' '
REPLACE NOUPDATE ALL merker WITH '*' ;
FOR int(konto) # konto
```

Wenn der Benutzer den Bericht ausdrucken will, schaltet das Programm den Drucker ein:

```
************************* Wenn Ausdruck verlangt,
************************* Drucker einschalten
ERASE
IF jn= 'J'
   SET PRINT ON
ENDIF (jn=j)
SET EJECT OFF
```

Nun kann das Programm die Berichtüberschrift und die Unterzeile drukken:

```
************************* Drucke Überschrift
? '                  Kontenübersicht für &firma'
```

```
?
************************ Drucke Unterzeile
? 'Aktualisiert am: &letzte'
```

Dann druckt es alle Einnahmen und benutzt dazu die Formatdatei COA.FRM:

```
****************** Drucke Einnahmen und Gesamtsumme
?
? 'EINNAHME:'
```

Dann druckt es die Gesamtsummen der Einnahmekonten (SUM:MON1, SUM:QURT1, SUM:JAHR1). Die String-Funktion (STR) wurde benutzt, um das Ausrichten der Summen in den Spalten etwas einfacher zu machen:

```
REPO FORM coa FOR konto<300 PLAIN
? 'Einnahmen TOTAL              ' +STR(sum:mon1,
                                              10,2)+;
'    ' +STR(sum:qurt1,10,2)+ '     ' +STR(sum:jahr1,;
                                               10,2)
?
```

Als nächstes druckt es die Ausgaben-Konten und ihre Summen (SUM:MON2, SUM:QURT2, SUM:JAHR2):

```
****************** Drucke Ausgabekonten und Summen
? 'AUSGABEN:'
REPORT FORM coa FOR konto>=300 PLAIN
? 'Ausgabe TOTAL                ' +STR(sum:mon2,;
                                              10,2)+;
'    ' +STR(sum:qurt2,10,2)+ '     ' +STR(sum:jahr2,;
                                               10,2)
```

Jetzt kann das Programm die Einnahmen- und Ausgabensummen am Ende des Berichts drucken und dazu die Bilanz (das bedeutet: Einnahmen TOTAL – Ausgaben TOTAL) für jede Zeiteinheit:

```
***************** Zeige Bilanz am Ende des Berichts
?
? ? 'Einnahmen TOTAL              ' +STR(sum:mon1,;
                                              10,2)+;
'    ' +STR(sum:qurt1,10,2)+ '     ' +STR(sum:jahr1,;
                                               10,2)
```

```
? 'Ausgaben TOTAL                    ' +STR(sum:mon2,
                                                10,2)+;
'    ' +STR(sum:qurt2,10,2)+ '      ' +STR(sum:jahr2,
                                                 10,2)
?
? 'Bilanz                          ' ,;
STR(sum:mon1-sum:mon2,10,2)+ '    ' +;
STR(sum:qurt1-sum:qurt2,10,2)+ '    ' +;
STR(sum:jahr1-sum:jahr2,10,2)
```

Dann druckt es ein paar leere Zeilen und zeigt die Unterkontomitteilung am Ende des Berichts:

```
?
?
?
? 'ACHTUNG: Unterkonten sind mit einem "*" markiert'
?? 'und in den Gesamtsummen nicht berücksichtigt'
```

Das Programm läßt den Drucker – um Ihnen entgegenzukommen – ein neues Blatt beginnen und schaltet dann den Drucker aus:

```
EJECT
SET PRINT OFF
```

Wenn der Bericht nicht an den Drucker ging, hält das Programm den Bildschirm an:

```
********************** Wenn nicht gedruckt wurde,
********************** Bildschirm anhalten
IF jn # 'J'
   ?
   ?
   ? 'Weiter mit beliebiger Taste ...'
   WAIT
ENDIF (jn # j)
```

Danach löscht es die Speichervariablen und kehrt zum BREPORTS-Menü zurück:

```
RELEASE firma,letzte,jn,sum:mon1,sum:mon2
RELEASE sum:qurt1,sum:qurt2,sum:jahr1,sum:jahr2
RETURN
```

Die Abb. 18.8 zeigt das komplette Programm COAREPT.CMD.

```
********************************** COAREPT.CMD
********************** Drucke Kontenübersicht und Bilanz

********************** Hole Datum der letzten Aktualisierung
********************** und Firmenname
USE geninfo
STORE le:aktu TO letzte
STORE firma TO firma

********************** Benutze COA.DBF und COA.NDX
USE coa INDEX coa

************************ Frage, ob Ausdruck gewünscht
ERASE
STORE ' ' TO jn
§ 5,2 SAY 'Bericht drucken ? ' GET jn PICTURE '!'
READ

***************** Berechne die Summen für Einnahmen und
***************** Ausgaben für Monat, Quartal und Jahr,
***************** berücksichtige Unterkonten nicht
ERASE
? 'Gesamtsummen werden berechnet ...'

SUM betrag FOR konto<300 .AND. int(konto)=konto ;
TO sum:mon1
SUM betrag FOR konto>=300 .AND. int(konto)=konto ;
TO sum:mon2
SUM vtj FOR konto<300 .AND. int(konto)=konto ;
TO sum:qurt1
SUM vtj FOR konto>=300 .AND. int(konto)=konto ;
TO sum:qurt2
SUM jrl FOR konto<300 .AND. int(konto)=konto ;
TO sum:jahr1
SUM jrl FOR konto>=300 .AND. int(konto)=konto ;
TO sum:jahr2

************************ Markiere Unterkonten mit *
REPLACE NOUPDATE ALL merker WITH ' '
REPLACE NOUPDATE ALL merker WITH '*' ;
FOR int(konto) # konto

************************ Wenn Ausdruck verlangt, Drucker
************************ einschalten
ERASE
IF jn= 'J'
   SET PRINT ON
ENDIF (jn=j)
SET EJECT OFF
```

Abb. 18.8: Das Programm COAREPT.CMD

```
************************* Drucke Überschrift
? '                  Kontenübersicht für &firma'
?
************************* Drucke Unterzeile
? 'Aktualisiert am: &letzte'

************************* Drucke Einnahmen und Gesamtsumme
?
? 'EINNAHME:'
REPO FORM coa FOR konto<300 PLAIN
? 'Einnahmen TOTAL                ' +STR(sum:mon1,10,2)+;
'    ' +STR(sum:qurt1,10,2)+ '    ' +STR(sum:jahr1,10,2)
?

************************* Drucke Ausgabekonten und Summen
? 'AUSGABEN:'
REPORT FORM coa FOR konto>=300 PLAIN
? 'Ausgaben TOTAL                 ' +STR(sum:mon2,10,2)+;
'    ' +STR(sum:qurt2,10,2)+ '    ' +STR(sum:jahr2,10,2)

************************* Zeige Bilanz am Ende des Berichts
?
? 'Einnahmen TOTAL                ' +STR(sum:mon1,10,2)+;
'    ' +STR(sum:qurt1,10,2)+ '    ' +STR(sum:jahr1,10,2)

? 'Ausgaben TOTAL                 ' +STR(sum:mon2,10,2)+;
'    ' +STR(sum:qurt2,10,2)+ '    ' +STR(sum:jahr2,10,2)
?
? 'Bilanz                        ' ,;
STR(sum:mon1-sum:mon2,10,2)+ '    ' +;
STR(sum:qurt1-sum:qurt2,10,2)+ '    ' +;
STR(sum:jahr1-sum:jahr2,10,2)

?
?
?
? 'ACHTUNG: Unterkonten sind mit "*" markiert'
?? 'und in Gesamtsummen nicht enthalten'

EJECT
SET PRINT OFF

************************** Wenn nicht gedruckt wurde,
************************** Bildschirm anhalten
IF jn # 'J'
   ?
   ?
   ? 'Weiter mit beliebiger Taste ...'
```

Abb. 18.8: Das Programm COAREPT.CMD (Forts.)

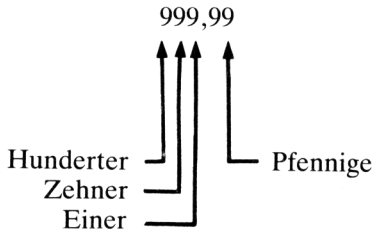
```
   WAIT
ENDIF (jn # j)

RELEASE firma,letzte,jn,sum:mon1,sum:mon2
RELEASE sum:qurt1,sum:qurt2,sum:jahr1,sum:jahr2
RETURN
```

Abb. 18.8: Das Programm COAREPT.CMD (Forts.)

Schecks schreiben

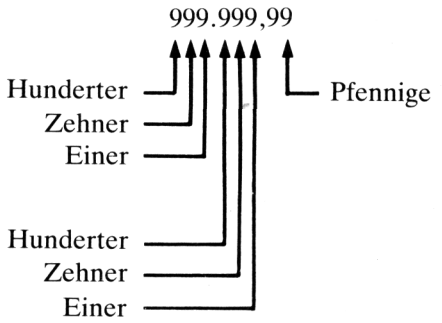
Wenn der Computer Schecks schreiben soll, muß er die numerischen Beträge (wie DM 1234.56) ins Deutsche übersetzen (eintausendzweihundertvierunddreißig 56/100).

Wir können die Sache vereinfachen, wenn wir eine Regel finden, nach der alle Zahlen aufgebaut werden. Wenn wir uns zum Beispiel eine Zahl unterhalb von eintausend ansehen, ist sie so aufgebaut:



Die Pfennige stehen immer rechts neben dem Komma, die Einer eine Stelle vor dem Komma, die Zehner zwei und die Hunderter drei.

Eine größere Zahl ist genauso aufgebaut:



Unsere Aufgabe ist es nun, zwei dreistellige Ziffern, jede im Hunderter-Zehner-Einer-Format, in einen ausgeschriebenen Betrag zu übersetzen. Die Pfennige schreiben wir als Bruch (z.B. 99/100).

Lassen Sie uns noch eine andere Besonderheit der Zahlen betrachten. Die Zahlen von eins bis neunzehn haben alle einen eigenen Namen (eins, zwei drei, elf, neunzehn usw.). Nach der Neunzehn haben nur noch die Zahlen einen eigenen Namen, die durch zehn teilbar sind (zwanzig, dreißig, neunzig usw.). Nach neunzig besitzen die Zahlen Einhundert und Eintausend wieder eigene Namen. Wir können diese Namen alle in Variablen speichern.

Wir können diese Wörter auf eine Art und Weise speichern, die uns eine Übersetzung sehr einfach macht. dBASE-Variablen müssen mit einem Buchstaben beginnen, dürfen aber Ziffern enthalten. Deshalb können wir die Variablen ihrem Inhalt entsprechend benennen:

```
STORE "eins" TO U1
STORE "zwei" TO U2
STORE "drei" TO U3
STORE "zwanzig" TO U20
```

Nachdem Sie die Zahlen in dieser Weise gespeichert haben, sehen Sie die deutsche Übersetzung einer Zahl, indem Sie dBASE fragen:

```
?U1
```

und dBASE antwortet:

```
eins
```

Geben Sie jetzt 23 ein, um die deutsche Übersetzung zu sehen:

```
U20,U3
```

antwortet dBASE:

```
zwanzig drei
```

Hier bemerken Sie, daß die deutsche Zählweise nach der Zahl Zwanzig die strenge Logik verläßt und Zahlen umgedreht aufbaut. Wir müssen also dem Programm beibringen, bei Zahlen über zwanzig die Einer voranzustellen und ein „und" einzufügen, damit wir das richtige Ergebnis „dreiundzwanzig" erhalten.

Obwohl das alles noch recht einfach ist, wird es schwieriger, wenn wir eine Million übersetzen wollen.

Bevor wir das Scheckprogramm schreiben können, müssen wir alle deutschen Zahlwörter, die wir brauchen, in Variablen speichern. Wir machen

das und speichern sie alle anschließend in eine MEMORY-Datei, die die Kennung MEM trägt. Damit wir die Variablen nicht jedesmal neu belegen müssen, lassen wir ein Programm diese Arbeit erledigen. Das Programm nennen wir MAKEMEMS.CMD, und seine Aufgabe wird es sein, alle Zahlwörter in die Speicherdatei ZAHLWORT.MEM zu speichern. Dieses Programm ist in Abb. 18.9 zu sehen.

```
********************************** MAKEMEMS.CMD
*************************** ERÖFFNEN EINER MEM-DATEI
*************************** FÜR ZAHLWÖRTER IN DEUTSCH
SET DEFAULT TO B
SET TALK OFF
RELEASE ALL

STORE ' ' TO u,deutsch
STORE 'eins' TO u1
STORE 'zwei' TO u2
STORE 'drei' TO u3
STORE 'vier' TO u4
STORE 'fünf' TO u5
STORE 'sechs' TO u6
STORE 'sieben' TO u7
STORE 'acht' TO u8
STORE 'neun' TO u9
STORE 'zehn' TO u10
STORE 'elf' TO u11
STORE 'zwölf' TO u12
STORE 'dreizehn' TO u13
STORE 'vierzehn' TO u14
STORE 'fünfzehn' TO u15
STORE 'sechzehn' TO u16
STORE 'siebzehn' TO u17
STORE 'achtzehn' TO u18
STORE 'neunzehn' TO u19
STORE 'zwanzig' TO u20
STORE 'dreißig' TO u30
STORE 'vierzig' TO u40
STORE 'fünfzig' TO u50
STORE 'sechzig' TO u60
STORE 'siebzig' TO u70
STORE 'achtzig' TO u80
STORE 'neunzig' TO u90

SAVE TO zahlwort
DISPLAY MEMORY
RETURN
```

Abb. 18.9: Das Programm MAKEMEMS.CMD

Beachten Sie, daß dieses Programm alle Speichervariablen löscht, alle Zahlwörter mit eigenem Namen in Variablen speichert, die eine entsprechende Benennung haben, und diese dann in der Datei ZAHLWORT speichert. Bevor Sie das Scheckprogramm schreiben, müssen Sie das ZAHLWORT-Programm schreiben und laufen lassen.

Das Programm für den Scheckausdruck heißt CHECKS.CMD. Der Pseudocode dazu steht in Abb. 18.10.

```
Zeige Eröffnungsmeldung auf dem Bildschirm, lasse Benutzer
Schecks in den Drucker laden

Speichere vorhandene Variablen in eine MEM-Datei
Hole Variablen mit den deutschen Zahlwörtern aus der Datei
ZAHLWORT.MEM

Erstelle aus der TRANS-Datenbank eine Zwischendatei der Schecks,
die noch zu schreiben sind

Indiziere die Zwischendatei nach Schecknummern

Warte, bis der Benutzer die Schecks im Drucker hat

Beginne Schleife durch die Zwischendatei mit den Schecks
   Speichere drei Stellen der Nummer in eine Zeichenkette
   Beginne Schleife für zweimaliges Durchlaufen der Nummern
      Hole Hunderter, Zehner und Einer einzeln heraus
      Übersetze die Hunderter in Zahlwörter

      Bearbeite die nächsten beiden Stellen
      Wenn sie Zehner (11–19) oder gerade Zehner (20, 30) sind,
      übersetze in deutsche Zahlwörter

      Wenn die nächsten zwei Stellen größer als 10,
      aber nicht durch 10 glatt teilbar sind,
      übersetze die beiden Stellen unabhängig voneinander

      Wenn die beiden Stellen kleiner als Zehn sind,
      übersetze die einzelne Stelle

      Wenn im „Tausender"-Bereich der Zahl und der Betrag ist
      größer als 999.99, füge „tausend" an
```

Abb. 18.10: Pseudocode für das Programm CHECKS.CMD

```
    Wiederhole Schleife für die nächsten drei Stellen der Zahl

    Füge die Pfennigbeträge an
    Drucke den Scheck

    Weiter zum nächsten Scheck in der Zwischendatei

Weiter mit der Schleife in der Zwischendatei

Wenn alle Schecks geschrieben
Markiere alle Schecks als geschrieben in der Datei TRANS

Beseitige die ZAHLWORT-Variablen und hole die ursprünglichen
Variablen wieder

Zurück zum Menüprogramm
```

Abb. 18.10: Pseudocode für das Programm CHECKS.CMD (Forts.)

Lassen Sie uns jetzt das CHECKS-Programm entwickeln. Wir beginnen mit dem Schreiben einiger Zeilen, die eine Eröffnungsmeldung auf den Bildschirm bringen:

```
********************************** CHECKS.CMD
***************************** Schecks schreiben
ERASE
SET TALK OFF
? 'Bereite die Scheckdatei vor ...'
?
? 'Bitte Schecks in Drucker laden'
```

Um innerhalb des Speichers für die ZAHLWORT-Variablen Platz zu schaffen, fügen wir noch ein paar Zeilen an, die es dem Programm ermöglichen, alle existierenden Variablen in eine Memory-Datei namens IMSINN zu speichern und dann die ZAHLWORT-Variablen einzulesen:

```
************** Speichere Variablen in IMSINN.MEM
************** und lade die Variablen aus ZAHLWORT
SAVE TO imsinn
RESTORE FROM zahlwort
```

Als nächstes nimmt unser Programm alle ungeschriebenen Schecks aus der Datei TRANS und speichert sie in der Zwischendatei TEMP:

```
********** Erstelle Zwischendatei für
********** zu schreibende Schecks
USE trans INDEX konto
COPY TO TEMP FOR scheck:nr>0 .AND. .NOT. geschr
```

Dann sortiert es die Zwischendatei nach Schecknummern:

```
USE TEMP
INDEX ON scheck:nr TO schecknrs
```

Nun beginnt das Programm am Anfang der Zwischendatei und sagt dem Benutzer, mit welcher Schecknummer angefangen wird. Dann gibt es ihm ein wenig Zeit, um die Schecks in den Drucker zu laden:

```
GO TOP
?
? 'Der erste Scheck hat die Nummer ' ,scheck:nr
?
?
? 'Mit beliebiger Taste den Ausdruck beginnen'
WAIT
```

Sobald der Benutzer zum Drucken bereit ist, löscht das Programm den Bildschirm, schaltet den Drucker ein und beginnt eine Schleife durch die Zwischendatei:

```
ERASE
SET PRINT ON
DO WHILE .NOT. EOF
```

Um die Übersetzung auszuführen, speichert das Programm zuerst den numerischen Betrag in eine Zeichenkette mit vorgegebener Größe (neun Stellen mit zwei Dezimalstellen). Auf diese Weise kann es den Wert jeder Stelle feststellen. Zum Beispiel haben die Zahlen

```
"     1.23"
"   123.00"
" 11234.40"
"999999.99"
```

innerhalb der Zeichenkette ihre Hunderter, Zehner, Einer und Pfennige genau an der gleichen Stelle (der Wert für eine Mark beispielsweise steht immer an der vierten Stelle von rechts). Da das Programm zwei Durchläufe durch die Zeichenkette machen wird (einen für jeweils drei Zahlen), brauchen wir auch einen Schleifenzähler und eine Startposition für jeden Dreiersatz von Zahlen. Diese Zeilen eröffnen die Variablen und spei-

chern den numerischen Scheckbetrag in eine Zeichenvariable mit dem Namen STRING:

```
****************** Übersetze den Betrag in Wörter
STORE 1 TO zaehler,start
STORE STR(betrag,9,2) TO string
```

Jetzt beginnen wir die Schleife, die zweimal durch die zu übersetzende Zahl geht:

```
*********** Schleife durch Tausender und Hunderter
  DO WHILE zaehler < 3
```

Nachdem das Programm mit der Position START beginnt, speichert es drei Stellen der Ziffer in die Variable TEIL und holt daraus dann die Hunderter, Zehner und Einer:

```
******** Hunderter, Zehner und Einer herausholen
STORE $(string,start,3) TO teil
STORE $(teil,1,1) TO hun
STORE $(teil,2,2) TO zehn
STORE $(teil,3,1) TO ein
```

Lassen Sie uns ein Beispiel mit diesem Programm versuchen. Wenn der Scheckbetrag für den zu schreibenden Scheck 123456.78 ist, haben die folgenden Variablen diese Werte und alle sind vom Typ „C":

Speichervariable	Inhalt
STRING	"123456.78"
TEIL	"123"
HUN	"1"
ZEHN	"23"
EIN	"3"

Die nächste Aufgabe besteht darin, die deutsche Übersetzung der Hunderter und das Wort „hundert" in eine Variable mit Namen DEUTSCH zu speichern, falls TEIL größer als 99 ist. Die deutsche Unlogik beim Zählen zwingt uns dazu, die Variable U1 noch zu verändern. Da „einshundert" in unserem Sprachgebrauch nicht vorkommt, speichern wir „ein" in die Variable U1:

```
******** Hunderter bearbeiten
     IF val(teil) > 99
        STORE 'ein' TO u1
        STORE deutsch + u&hun + 'hundert' TO deutsch
     ENDIF (teil > 99)
```

Benutzen wir wieder die Zahl aus unserem Beispiel, haben wir U&HUN (U1, in diesem Fall „ein") und das Wort „hundert" in die Variable DEUTSCH gespeichert, so daß sie jetzt „einhundert" enthält.

Dann übersetzt das Programm einfach die nächsten drei Stellen in TEIL. Zuerst speichert es den numerischen Wert von ZEHN in die Variable X. Dann stellt es sicher, daß X größer als null ist. Mit den Zehnern haben wir das gleiche Problem wie mit den Hundertern, wie Sie an STORE "ein" TO U1 sehen können:

```
******** Bearbeite die Zehner
STORE val(zehn) TO x
STORE 'ein' TO u1
IF x > 0
```

Wenn diese Bedingung zutrifft, müssen wir uns um drei Möglichkeiten kümmern. Zunächst kann es sein, daß die ersten beiden Stellen Zehner sind (11–19) oder glatt durch zehn geteilt werden können (20, 30 usw.). Bei diesen Zahlen übernimmt das Programm das deutsche Wort für die Variable ZEHN und speichert es in DEUTSCH ab:

```
DO CASE

  ********* Fall 1: Zehner (11-19) oder durch zehn
  ********* teilbar (20, 30 usw.)
  CASE (int(x/10.0)=x/10.0) .OR. (x > 9 .AND. x< 20)
  STORE deutsch + u&zehn TO deutsch
```

Die zweite Möglichkeit ist, daß die zwei Ziffern größer als neun, aber nicht durch zehn teilbar sind (21–29, 31–39 usw.). In diesem Fall speichert das Programm die deutsche Übersetzung für beide Teile der Zahl, die Zehner und die Einer, in die Variable DEUTSCH. Da wir beim Zählen die Einer und Zehner mit einem „und" verbinden (einundzwanzig), schiebt das Programm noch ein „und" dazwischen, nachdem es die Einer vorangestellt hat:

```
******* Fall 2: Größer als zehn, aber nicht
******* durch zehn teilbar
 CASE x > 9 .AND. (int(x/10.0) # x/10.0)
    STORE $(zehn,1,1) + '0' TO zehn
    STORE deutsch + u&ein + 'und' + u&zehn TO;
                                          deutsch
```

Unsere Beispielzahl 123456.78 fällt unter diese Kategorie. Deshalb speichert die Routine zuerst das erste Zeichen von ZEHN (das ist die 2, weil

ZEHN 23 enthält) und eine Null (0) in die Variable ZEHN, die jetzt 20 enthält. Als nächstes bringt die Zeile „STORE deutsch + u&ein + 'und' + u&zehn TO deutsch" die Einer (U&ein) nach vorn (worin jetzt U3, das Wort „drei", enthalten ist) in die Variable DEUTSCH. Weiter speichert das Programm den Zusatz „und" sowie die Variable U&ZEHN (worin jetzt U20, das Wort „zwanzig", steht) in die Variable DEUTSCH ab. An dieser Stelle enthält die Variable DEUTSCH „einhundertdreiundzwanzig".

Die dritte Möglichkeit ist, daß die letzten beiden Stellen weniger als zehn enthalten (01, 02, 09). In diesem Fall fügt das Programm nur die Übersetzung des „ein"-Stückes in den DEUTSCH-String ein:

```
********** Fall 3: Weniger als 10
  CASE x < 10
       STORE deutsch + u&ein TO deutsch
  ENDCASE
ENDIF (x > 0)
```

Wenn jetzt das Programm die dreistellige Zahl im Tausenderteil der Gesamtzahl in Wörter übersetzt hat und der Betrag größer als 999.99 ist, fügt es einfach noch das Wort „tausend" an. Diese Zeilen sorgen dafür:

```
***************** "tausend" anhängen, wenn nötig
  IF betrag > 999.99 .AND. zaehler = 1
     STORE deutsch+ 'tausend' TO deutsch
  ENDIF (tausend anfügen)
```

Wenn wir unsere Beispielzahl noch einmal betrachten, steht in DEUTSCH jetzt „einhundertdreiundzwanzigtausend".

Jetzt muß das Programm die gleiche Aufgabe für die nächsten drei Stellen erledigen. Diese beginnen an der vierten Stelle in STRING. Deshalb ist die Startposition für die nächste Übersetzung die 4 und der Schleifenzähler ist bei 2:

```
******************* Fertigmachen zum Durchlaufen
******************* der Hunderter
STORE 4 TO start
STORE zaehler+1 TO zaehler

ENDDO (WHILE zaehler <3)
```

Benutzen wir unsere Beispielzahl in der zweiten Schleife, sehen die Speichervariablen so aus:

Speichervariable	Inhalt
STRING	"123456.78"
TEIL	"456"
HUN	"4"
ZEHN	"56"
EIN	"6"

Da wir dieselbe Prozedur wie vorher auch beim zweiten Schleifendurchlauf verwenden, sieht die Variable DEUTSCH jetzt so aus: „einhundertdreiundzwanzigtausendvierhundertsechsundfünzig". An dieser Stelle muß das Programm nur noch die Pfennige, die letzten beiden Stellen der Zahl, in die Variable STRING speichern. Natürlich müssen wir noch das Wort „UND" (z.B. einhundertdreiundzwanzig UND 56/100 Pfennig) für Schecks über 1,00 DM anfügen. Diese kleine Routine erledigt das:

```
*************** Pfennige anfügen
  IF int(betrag) > 0
     STORE deutsch + 'und' TO deutsch
  ENDIF (int(betrag)>0))
  STORE deutsch + $(string,8,2)+ "/100" TO deutsch
```

An dieser Stelle sieht die Variable DEUTSCH unseres Beispiels so aus „einhundertdreiundzwanzigtausendvierhundertsechsundfünzig und 78/100", und die Übersetzung ist vollständig (vorausgesetzt, der Ausdruck DM steht auf dem Scheck).

Da die Übersetzung jetzt fertig ist, kann das Programm das Datum, den Empfänger, den Betrag und die Übersetzung des Betrags auf den Scheck schreiben. Sie werden diesen Teil des Programms vielleicht umschreiben müssen, um ihn an Ihre vorgedruckten Scheckformulare anzupassen. Sie verwenden am besten Fotokopien Ihrer Schecks zum Ausprobieren, da dies doch meist einige Versuche nötig macht. Diese Programmzeilen des CHECKS-Programms drucken den Scheck:

```
************ Scheck drucken
  ? '                                    ' ,datum
  ?
  ? '      ' ,an:wen, '          ' , 'DM' ,betrag
  ?
  ? '      ' ,deutsch
  ?
  ?
  ?
```

Sobald der Scheck geschrieben ist, sollte das Programm zum nächsten Datensatz gehen und weiter durch die Zwischendatei laufen. Vorher muß

es aber noch DEUTSCH wieder leeren. Diese Zeilen bewerkstelligen das:

```
    SKIP
    STORE ' ' TO deutsch

ENDDO (while not eof)
```

Das Ende dieser Schleife bedeutet, daß alle Schecks geschrieben wurden. Wir können den Drucker ausschalten, zur TRANS-Datei zurückgehen und alle Schecks als geschrieben markieren, indem wir in das Feld GESCHR ein T eingeben:

```
SET PRINT OFF

********** Markiere Schecks als geschrieben
********** in TRANS.DBF
USE trans
REPLACE ALL geschr  WITH T FOR typ = 1
USE
```

Jetzt können wir das Programm alle Variablen aus ZAHLWORT löschen lassen und die alten Variablen aus IMSINN.MEM wieder einladen. Dann kann das Programm zum Report-Menü zurückkehren. Mit diesen Zeilen geht das:

```
********** Lösche ZAHLWORT-Variablen
********** und hole Originalvariablen zurück
RELEASE ALL
RESTORE FROM imsinn
RETURN
```

Das gesamte CHECKS.CMD-Programm ist in Abb. 18.11 zu sehen. Es ist zwar ein ziemlich langes Programm, lohnt aber das genaue Studium, da es ein schwieriges Problem mit einem minimalen Programmieraufwand löst. Außerdem läuft es auch recht schnell.

```
********** CHECKS.CMD
********** Schecks schreiben
ERASE
SET TALK OFF
? 'Bereite die Scheckdatei vor ...'
?
? 'Bitte Schecks in Drucker laden'
```

Abb. 18.11: Das Programm CHECKS.CMD

```
************** Speichere Variablen in IMSINN.MEM
************** und lade die Variablen aus ZAHLWORT
SAVE TO imsinn
RESTORE FROM zahlwort

************ Erstelle Zwischendatei für zu schreibende Schecks
USE trans INDEX konto
COPY TO TEMP FOR scheck:nr>0 .AND. .NOT. geschr
USE TEMP
INDEX ON scheck:nr TO schecknrs
GO TOP
?
? 'Der erste Scheck hat die Nummer ' ,scheck:nr
?
?
? 'Mit beliebiger Taste den Ausdruck beginnen'
WAIT

ERASE
SET PRINT ON
DO WHILE .NOT. EOF

   ************************* Übersetze den Betrag in Wörter
   STORE 1 TO zaehler,start
   STORE STR(betrag,9,2) TO string

   ******************** Schleife durch Tausender und Hunderter
   DO WHILE zaehler < 3

      ******** Hunderter, Zehner und Einer herausholen
      STORE $(string,start,3) TO teil
      STORE $(teil,1,1) TO hun
      STORE $(teil,2,2) TO zehn
      STORE $(teil,3,1) TO ein

      ******** Hunderter bearbeiten
      IF val(teil) > 99
         STORE 'ein' TO u1
         STORE deutsch + u&hun + 'hundert' TO deutsch
      ENDIF (teil > 99)

      ******** Bearbeite die Zehner
      STORE val(zehn) TO x
      STORE 'ein' TO u1
      IF x > 0

        DO CASE
```

Abb. 18.11: Das Programm CHECKS.CMD (Forts.)

```
          ********* Fall 1: Zehner (11-19) oder durch zehn
          ********* teilbar (20, 30 usw.)
          CASE (int(x/10.0)=x/10.0) .OR. (x > 9 .AND. x < 20)
            STORE deutsch + u&zehn TO deutsch

          ******* Fall 2: Größer als zehn, aber nicht durch
          ******* zehn teilbar
          CASE x > 9 .AND. (int(x/10.0)  #x/10.0)
              STORE $(zehn,1,1) + '0' TO zehn
              STORE deutsch + u&ein + 'und' + u&zehn TO deutsch

          ********** Fall 3: Weniger als 10
          CASE x < 10
              STORE deutsch + u&ein TO deutsch
       ENDCASE
    ENDIF (x > 0)

    ***************** "tausend" anhängen, wenn nötig
    IF betrag > 999.99 .AND. zaehler = 1
       STORE deutsch+ 'tausend' TO deutsch
    ENDIF (tausend anfügen)

    ***************** Fertigmachen zum Durchlaufen der
    ***************** Hunderter
    STORE 4 TO start
    STORE zaehler+1 TO zaehler

ENDDO (WHILE zaehler <3)

*************** Pfennige anfügen
IF int(betrag) > 0
   STORE deutsch + ' und ' TO deutsch
ENDIF (int(betrag)>0))
STORE deutsch + $(string,8,2)+ "/100" TO deutsch

********** Scheck drucken
? '                                ' ,datum
?
? '      ' ,an:wen, '         ' , 'DM' ,betrag
?
? '      ' ,deutsch
?
?
?

SKIP
STORE ' ' TO deutsch
```

Abb. 18.11: Das Programm CHECKS.CMD (Forts.)

```
ENDDO (while not eof)

SET PRINT OFF

**************** Markiere Schecks als geschrieben in TRANS.DBF
USE trans
REPLACE ALL geschr  WITH T FOR typ = 1
USE

********** Lösche ZAHLWORT-Variablen
********** und hole Originalvariablen zurück
RELEASE ALL
RESTORE FROM imsinn
RETURN
```

Abb. 18.11: Das Programm CHECKS.CMD (Forts.)

Zusammenfassung

In diesem Kapitel haben wir die Entwicklung der Programme zum Anzeigen der Buchhaltungsdaten in formatierten Berichten abgeschlossen. Diese Programme sind:

1. BREPORTS.CMD, das ein Menü mit Report-Option zeigt.
2. CURREPT.CMD, das Die Einnahmen/Ausgaben-Übersicht nach Daten aus der TRANS-Datei zeigt. Es ermöglicht dem Benutzer auch, die Buchungen des aktuellen Monats oder für einen bestimmten Zeitraum zu drucken, wobei er die Sortierung aus einem Menü wählen kann.
3. COAREPT.CMD, das die Kontenübersicht druckt, mit Monats-, Quartals- und Jahresbilanzen. Es berechnet und zeigt auch alle Gesamtsummen und demonstriert die Techniken zum Berechnen komplexer Summen und Buchhaltungsdaten.
4. CHECKS.CMD, das Schecks ausdruckt und die notwendigen Techniken zum Übersetzen von Zahlen in Zahlwörter zeigt.

Daten der Buchhaltung ändern

19

Alle Buchhalter wissen, daß sie im Laufe eines Jahres Änderungen machen müssen. In diesem Kapitel werden wir Programme entwickeln, die es dem Benutzer erlauben, die notwendigen Änderungen und Berichtigungen zu machen, ohne dabei die chronologische Fortschreibung der Daten zu stören. Über diese Änderungen wird in allen Bilanzen und Gesamtsummen Rechenschaft abgelegt werden, wenn die Buchhaltungsberichte ausgedruckt werden.

Der Benutzer muß folgende Änderungen durchführen können:

1. Ändern und Löschen von Daten in der Kontenübersicht und in den Dateien TRANS und REGISTER,
2. neue Konten zur Kontenübersicht hinzufügen,
3. möglicherweise Daten in das Scheckregister einfügen,
4. die Daten in der Informations-Datei GENINFO ändern.

Um im Buchhaltungssystem Änderungen vorzunehmen, wählt der Benutzer die Option 5 im Hauptmenü. Diese Wahl ruft folgendes Menü auf den Bildschirm:

```
Buchhaltung-Änderungen

1.   Kontenrahmen
2.   Buchungsdatei
```

```
3.   Scheckregister
4.   Allgemeine Informationen

5.   Zurück zum Hauptmenü
```

Wenn der Benutzer die Option 1 wählt, erscheinen auf dem Bildschirm die Fragen:

```
(E)ingeben, (V)erändern oder (L)öschen?
```

Will er die Daten eines Kontos ändern, wählt er V und auf dem Bildschirm erscheint:

```
Welche Kontonummer? :     .  :
```

Er gibt dann die Kontonummer ein, und der Bildschirm zeigt die Daten dieser Kontonummer und kurze Hinweise über die Änderungsmöglichkeiten, wie hier gezeigt:

```
Cursor Control : ^E hoch: ^X runter: ^D rechts: ^S
links : ^G loeschen

Screen Control : ^C nächstes Konto : ^R vorheriges
Konto : ^W sichern
```

```
Kontonummer :100.00:     Titel :Einkommen :

Monat     : 2500.00:

Quartal   : 5000.00:

Jahr      : 5000.00:

ACHTUNG: Verändern der Bilanzen verhindert chronologi-
sche Fortschreibung! Benutzen Sie Berichtigungs-Buchun-
gen.
```

An dieser Stelle kann der Benutzer Änderungen vornehmen, zum vorherigen oder nächsten Konto blättern und/oder seine Änderungen abspeichern. Die Warnung am unteren Rand der Maske soll ihn daran erinnern, daß er die Bilanzen nicht ohne Folgen ändern kann. In Buchhaltungssystemen werden Änderungen normalerweise über Korrekturbuchungen eingegeben, über die wir später sprechen werden.

Die Änderungsmasken der Buchhaltung

Der Benutzer kann alle Änderungen und Einträge über vordefinierte Masken vornehmen. Die Maske zum Ändern und Eingeben von Daten in die Kontenübersicht wurde zuerst mit dem Programm ZIP so gezeichnet:

```
Cursor Control : ^E hoch: ^X runter: ^D rechts: ^S
links : ^G loeschen

Screen Control : ^C nächstes Konto : ^R vorheriges
Konto : ^W sichern
```

```
Kontonummer #konto        Titel #titel

Monat      #betrag

Quartal    #vtj

Jahr       #jrl

ACHTUNG: Verändern der Bilanzen verhindert chronologi-
sche Fortschreibung! Benutzen Sie Berichtigungs-Buchun-
gen.
```

Das ZIP-Programm hat nach der Fertigstellung dieses Programm geschrieben:

```
* COA.FMT
§  1, 1 SAY "Cursor Control : ^E hoch: ^X runter: ^D rechts: ^S
        links : ^G loeschen"
§  3, 1 SAY "Screen Control : ^C nächstes Konto : ^R vorheriges
        Konto : ^W sichern"
§  4, 1 SAY "============================================
             ================================"
§  6, 1 SAY "Kontonummer"
§  6,13 GET konto
§  6,25 SAY "Titel"
§  6,31 GET titel
§  8, 1 SAY "Monat"
§  8,10 GET betrag
§ 10, 1 SAY "Quartal"
§ 10,10 GET vtj
§ 12, 1 SAY "Jahr"
§ 12,10 GET jrl
```

```
§ 16, 1 SAY "ACHTUNG: Verändern der Bilanzen verhindert chrono-
logische Fortschreibung!"
§ 17,10 SAY "Benutzen Sie Berichtigungs-Buchungen."
§ 20, 1 SAY "=========================================
             ================================="
```

Wenn ZIP Sie nach dem Typ des zu erstellenden Programms fragt, geben Sie FMT statt CMD ein.

Sie können die anderen drei Masken in derselben Weise erstellen. Die Maske TRANS.FMT sieht so aus:

```
Cursor Control : ^E hoch: ^X runter: ^D rechts: ^S
links : ^G loeschen
Screen Control : ^C nächste Buchung: ^R vorherige
Buchung: ^W sichern
```

```
 Kontonummer #konto                          Beschreibung
#grund

 Betrag    #betrag                           Datum #datum

 Typ (1=Scheck, 2=anderes)   #typ

 Schecknummer #scheck:nr

 Gezahlt an    #an:wen
```

ZIP schreibt Ihnen dann dieses Programm:

```
* TRANS.FMT
§  1, 1 SAY "Cursor Control : ^E hoch: ^X runter: ^D rechts:
        ^S links : ^G loeschen"
§  3, 1 SAY "Screen Control : ^C nächste Buchung: ^R vorherige
        Buchung: ^W sichern"
§  4, 1 SAY "=========================================
             ================================="
§  6, 1 SAY "Kontonummer"
§  6,13 GET konto
§  6,38 SAY "Beschreibung"
§  6,51 GET grund
§  8, 1 SAY "Betrag"
§  8,10 GET betrag
§  8,38 SAY "Datum"
```

```
§  8,44 GET datum PICT "99/99/99"
§ 10, 1 SAY "Typ (1=Scheck, 2=anderes)"
§ 10,28 GET typ
§ 12, 1 SAY "Schecknummer"
§ 12,14 GET scheck:nr
§ 14, 1 SAY "Gezahlt an"
§ 14,14 GET an:wen
§ 20, 1 SAY "============================================
             ================================="
```

Eine Schwierigkeit müssen Sie beim Arbeiten mit ZIP beachten. Es ist nicht möglich, PICTURE-Statements direkt auf den Bildschirm zu schreiben und von ZIP mit erstellen zu lassen. Aus diesem Grund müssen Sie nach der Erstellung der Maske TRANS.FMT noch einmal in dieses Programm hinein gehen und die Datum-Zeile so ändern:

```
§ 8,44 GET datum PICT "99/99/99"
```

Sie erstellen die GENINFO-Maske mit ZIP einfach folgendermaßen:

```
Cursor Control : ^E hoch: ^X runter: ^D rechts: ^S
links : ^G loeschen
```

```
Firma   #firma

Letzte Schecknummer #1:scheck
```

Das dazugehörige Programm erstellt ZIP so:

```
* GENINFO.FMT
§  1, 1 SAY "Cursor Control : ^E hoch: ^X runter: ^D rechts:
        ^S links : ^G loeschen"
§  3, 1 SAY "============================================
             ================================="
§  5, 1 SAY "Firma"
§  5, 9 GET firma
§  7, 1 SAY "Letzte Schecknummer"
§  7,21 GET 1:scheck
```

Die Maske zum Ändern des Scheckregisters sieht auf dem ZIP-Bildschirm so aus:

```
Cursor Control : ^E hoch: ^X runter: ^D rechts: ^S
links  :^G loeschen
```

```
Screen Control : ^C nächster Scheck: ^R vorheriger
Scheck :^W sichern

Schecknummer #scheck:nr        Gezahlt an #an:wen

Betrag    #betrag              Grund #grund

Datum #datum                   Ist das eine Einzah-
                               lung? (J N)#einzahlng
```

Und bei dem daraus entstehenden dBASE-Programm müssen Sie wieder die Änderung am DATUM-Feld vornehmen, weil ZIP keine PICTURE-Anweisung erstellen kann:

```
* REGISTER.FMT
§  1, 1 SAY "Cursor Control : ^E hoch: ^X runter: ^D rechts:
        ^S links  :^G loeschen"
§  3, 1 SAY "Screen Control : ^C nächster Scheck: ^R vorheriger
        Scheck :^W sichern"
§  4, 1 SAY
"=================================================
             ======================"
§  6, 1 SAY "Schecknummer"
§  6,14 GET scheck:nr
§  6,39 SAY "Gezahlt an"
§  6,50 GET an:wen
§  8, 1 SAY "Betrag"
§  8,10 GET betrag
§  8,39 SAY "Grund"
§  8,45 GET grund
§ 10, 1 SAY "Datum"
§ 10, 7 GET datum PICT "99/99/99"
§ 10,39 SAY "Ist das eine Einzahlung? (J/N)"
§ 10,69 GET einzahlng
§ 20, 1 SAY
"=================================================
             ========================"
```

Sollten Sie ein anderes Maskenerstellungsprogramm besitzen, können Sie das selbstverständlich auch zur Erstellung der Masken benutzen.

Buchhaltungsdaten eingeben, ändern oder löschen

Wir können uns viel Programmierarbeit ersparen, wenn wir ein Programm entwickeln, das die Änderungen an allen vier Datenbanken vornimmt. Um das zu erreichen, werden wir nur ein paar Makros erstellen.

```
Erstelle Makros, die Datenbanken und Format-Dateien an
bestimmten Menüpunkt binden

Speichere die Befehle APPEND und EDIT in Makros

Beginne Schleife für das Menü
   Lösche Bildschirm
   Zeige das Menü
   Warte auf Eingabe

   Stelle sicher, daß gültige Eingabe gemacht wurde
   Wenn nicht, zeige Fehlermeldung und wiederhole Vorgang

   Wenn nicht Ende verlangt ...
      Benutze entsprechende Datenbank

      Wenn COA bearbeitet wird, frage nach Befehl
      Wenn keine neue Kontonummer eingegeben wird ...
      Frage nach zu löschendem oder zu änderndem Konto
      Finde dieses Konto

      Wenn TRANS-Datei bearbeitet wird, frage nach Befehl
         Frage nach Kontonummer und Datum des zu ändernden
         Datensatzes
         Zähle, wie viele Datensätze diese Nummer und dieses
         Datum haben
         Wenn mehr als ein Satz mit diesen Kriterien,
         anzeigen und nach Satznummer fragen
         Wenn nur ein Satz mit dieser Kontonummer und diesem
         Datum, finde Datensatz

      Wenn Scheckregister bearbeitet wird, frage nach Befehl
      Frage nach der zu bearbeitenden Schecknummer

      Wenn die gefragte Nummer gefunden wurde ...
         Eingeben oder Ändern, ganz nach Benutzerwunsch

         Wenn Benutzer zu löschen wünscht ...
         Datensatz löschen
         Daten endgültig löschen (PACK)

         Wenn kein Satz gefunden, Benutzer warnen

         Weiter mit dem Menü (solange nicht Ende gewünscht)
```

Abb. 19.1: Pseudocode für das Programm BEDIT.CMD

Wir werden das Programm zum Ändern der Buchhaltungsdaten, BEDIT.CMD, entsprechend dem Pseudocode in Abb. 19.1 entwickeln.

Also lassen Sie uns mit der Entwicklung des BEDIT-Programms anfangen. Zuerst müssen wir Makros aufbauen, die Datenbanken und Masken direkt an bestimmte Menüpunkte anbinden. Wenn der Benutzer zum Beispiel die Kontenübersicht überarbeiten möchte, muß das Programm die Datenbank COA mit dem Index COA in Gebrauch nehmen und das Format auf COA.FMT setzen (SET FORMAT TO COA).

Wenn er die Option 2 wählt, um die Buchungsdatei zu verändern, muß das Programm befehlen: USE TRANS INDEX KONTO,DATUM und muß die Formatdatei so aufrufen: SET FORMAT TO TRANS (der TRANS.FMT-Formatdatei). Die ersten Zeilen des Programms erstellen die entsprechenden Makros, damit es sich je nach Benutzerwahl für die richtige Datenbank und die richtige Formatdatei entscheiden kann:

```
************************************ BEDIT.CMD
************** Änderungsprogramm für die Buchhaltung

************ Erstelle Makros, um Datenbanken und
************ Format Dateien einer Menüwahl
************ zuzuordnen

STORE "coa INDEX coa" TO datei1
STORE "coa" TO format1
STORE "trans INDEX konto,datum" TO datei2
STORE "trans" TO format2
STORE "register" TO datei3,format3
STORE "geninfo" TO datei4,format4
```

Der Benutzer hat auch die Möglichkeit, das (E)ingeben, (V)erändern oder (L)öschen von Daten zu verlangen. Wenn er E eingibt, sollte das Programm einen leeren Datensatz anhängen. Wählt er V, soll er die Möglichkeit zum Ändern bekommen. Die nächsten Zeilen erstellen die Befehlsmakros und speichern sie in Variablen mit den Namen BEFEHLE und BEFEHLV:

```
***************************** Speichere Befehle
STORE "APPEND" TO befehle
STORE "EDIT #" TO befehlv
```

Nachdem jetzt die Makros fertig sind, muß das Programm das Menü der Änderungsmöglichkeiten zeigen und auf eine Eingabe warten. Hier sind die Zeilen, die das möglich machen:

```
********** Schleife für Menü beginnen
STORE " " TO ewahl
DO WHILE ewahl # "5"
   ERASE
   TEXT
                    Buchhaltung Änderungen

              1.    Kontenrahmen
              2.    Buchungsdatei
              3.    Scheckregister
              4.    Allgemeine Informationen

              5.    Zurück zum Hauptmenü

    ENDTEXT
    § 11,12 SAY "Ihre Wahl (1-5)" GET ewahl
    READ
```

Weil das Programm die Eingabe des Benutzers (EWAHL) später als Makro verwenden wird, können hier nur gültige Eingaben geduldet werden. Die folgende Routine stellt sicher, daß die Benutzereingabe eine Zahl zwischen 1 und 5 war. Wenn nicht, erscheint ein Fehlermeldung, und das Menü erscheint wieder auf dem Bildschirm. Die folgenden Zeilen behandeln eine Falscheingabe:

```
********** Sicherstellen, daß gültige Eingabe
********** gemacht wurde
IF ewahl < "1" .OR. ewahl > "5"
   **** Wenn nicht, zeige Fehlermeldung und
   **** dann wieder das Menü
   § 13,12 SAY "** Ungültige Eingabe **"
   ? CHR(7)
   ********** Zähle bis 20, um Zeit zu verbrauchen
   STORE 1 TO dauer
   DO WHILE dauer < 21
      STORE dauer+1 TO dauer
   ENDDO (dauer<21)
   LOOP
ENDIF (falsche Eingabe)
```

Die DO WHILE-Schleife in der obenstehenden Routine dauert 20 Sekunden. Während dieser Zeit steht die Fehlermeldung „** Ungültige Eingabe **" auf dem Bildschirm. Dann wird der Bildschirm gelöscht und das Menü wieder angezeigt.

Will der Benutzer jetzt nicht aufhören, kann das Programm weitermachen. Die nächsten Zeilen erlauben ihm weiterzumachen:

```
********** Wenn nicht Ende, hole Informationen
********** um Datensatz zu finden
IF ewahl # "5"
   ERASE
```

Zuerst muß das Programm die richtige Datei benutzen. Wenn der Benutzer die Datei TRANS bearbeiten will, muß das Programm befehlen: USE trans INDEX konto,datum. Erinnern Sie sich daran, daß das Makro DATEI2 diese Information enthält? Deshalb muß das Programm sagen: USE &datei2. Die 2 entspricht zur Zeit der Eingabe des Benutzers vom Menü aus, die in der Variablen EWAHL gespeichert wurde. Deshalb sagt das Programm: USE &datei&ewahl, was dBASE zuerst mit USE &datei2 und dann mit USE trans INDEX konto,datum übersetzt. Die nächste Zeile wählt das entsprechende Programm:

```
USE &datei&ewahl
```

Je nachdem, welche Wahl der Benutzer getroffen hat, muß das Programm herausfinden, mit welchen der drei Möglichkeiten „(E)ingeben, (V)erändern oder (L)öschen" es weitermachen und welcher Datensatz bearbeitet werden soll. Die auszuführende Arbeit wird in der Variablen SUBBEFEHL als E, V oder L gespeichert. SUBBEFEHL wird ohne Inhalt eröffnet:

```
STORE " " TO subbefehl
```

Für den Fall, daß der Benutzer die Option 4 wählt (Allgemeine Information), überspringen wir die DO CASE-Anweisung, weil keine Suchroutine gebraucht wird. Das gelingt uns, indem wir eingeben:

```
IF ewahl = "4"
     STORE "V" TO subbefehl
ENDIF (ewahl = 4)
```

Jetzt brauchen wir eine CASE-Anweisung, um die verschiedenen Dateien bearbeiten zu können. Wenn der Benutzer die Option 1 wählt (Kontenrahmen), kann er ein Konto entweder eingeben, ändern oder löschen. Das erste CASE fragt ihn, was er machen möchte:

```
DO CASE
********** Wenn COA.DBF gewünscht, frage nach
******* Befehl, dann suche nach Kontonummer
```

```
CASE ewahl = "1"
  ERASE
  § 3,5 SAY "(E)ingeben, (V)erändern oder
  (L)öschen?"; GET subbefehl PICT "!"
  READ
```

Will er keine neue Kontonummer eingeben, muß das Programm fragen, welches Konto geändert oder gelöscht werden soll und dieses Konto in der Datenbank finden:

```
IF subbefehl # "E"
   STORE "      " TO finden
   § 5,5 SAY "Welche Kontonummer?";
         GET finden PICT "999.99"
   READ
   FIND &finden
ENDIF (befehl # e)
```

Will der Benutzer die TRANS-Datei (Menüoption 2) bearbeiten, wird die Sache schon etwas trickreicher. Zuallererst sollten wir es ihm nicht gestatten, mit diesem Programm Datensätze einzugeben. Es bietet nämlich längst nicht die Sicherheitsvorkehrungen, die das Programm ADDTRANS bietet. Deshalb zeigt das Programm nur Optionen zum Ändern oder Löschen an:

```
*************** Wenn Buchung bearbeitet wird, suche
*************** nach Konto und Datum
CASE ewahl = "2"
  § 3,5 SAY "(V)erändern oder (L)öschen?";
        GET finden PICT "999.99"
```

Dann muß das Programm den zu editierenden Datensatz finden. Die Kontonummer und das Datum bieten sich als Suchfelder an, und das Programm fragt danach:

```
STORE "    " TO finden
  STORE "     " TO finden2
  § 3,5 SAY "Welche Kontonummer?";
        GET finden PICT "999.99"
  § 7,5 SAY "Welches Datum?";
        GET finden2 PICT "99/99/99"
  READ
```

Jetzt muß das Programm den entsprechenden Datensatz finden. Zuerst aber zählt es ab, wie viele Datensätze in der TRANS-Datei diese Nummer und dieses Datum haben:

```
*************** Ermittle genau Anzahl der Sätze
COUNT FOR konto = &finden .AND. datum = finden2 TO
                                                wieviele
```

Besitzen mehrere Sätze diese Suchkriterien, zeigt das Programm alle auf dem Bildschirm an und holt sich die richtige Satznummer vom Benutzer:

```
IF wieviele > 1
   ERASE
   LIST FOR konto = &finden .AND. datum = finden2;
   konto, datum,grund,betrag
   ?
   INPUT "Welchen Datensatz?" TO satznr
   GOTO satznr
```

Wenn nur ein Datensatz die gesuchte Nummer und das gesuchte Datum hat, kann das Programm diesen direkt finden:

```
ELSE
    LOCATE FOR konto = &finden .AND. datum = finden2
ENDIF (wieviele>1)
```

Wenn der Benutzer das Scheckregister bearbeiten möchte, kann er Daten eingeben, verändern oder löschen. Diese Zeilen machen das möglich:

```
************* Wenn REGISTER, suche nach Schecknummer
CASE ewahl = "3"
  § SAY "(E)ingeben, (V)erändern oder (L)öschen?";
    GET subbefehl PICT "!"
READ
```

Wenn der Benutzer keinen Scheck eingibt, kann das Programm den zu verändernden Datensatz suchen, indem es nach der Schecknummer fragt und diesen Scheck dann in der Datenbank findet:

```
      IF subbefehl # "E"
         STORE "      " TO finden
          5,5 SAY "Welche Schecknummer?";
               GET finden PICT "9999"
         READ
         LOCATE FOR scheck:nr = &finden
      ENDIF (befehl # a)

ENDCASE
```

Will er die GENINFO-Datei bearbeiten, kann er nur Daten verändern, weil in dieser Datei nur ein Datensatz steht. Deshalb brauchen wir hier keine Suchroutine.

Sobald das Programm den Datensatz gefunden hat, kann es fortfahren:

```
****************** Wenn Datensatz gefunden, ändern
****************** oder löschen ermöglichen
IF # > 0 .AND. .NOT. EOF

   DO CASE
```

Will der Benutzer Daten einfügen oder löschen (SUBBEFEHL = "E" oder "V"), muß das Programm die entsprechende Maske aufrufen und den entsprechenden Befehl ausführen. Gibt der Benutzer beispielsweise ein Konto in den Kontenrahmen ein, wird er die Option 1 im Menü wählen. An dieser Stelle wird EWAHL = 1. Er wird zusätzlich E wählen, für (E)ingeben, und deshalb wird der SUBBEFEHL zu E. Also wird die Konstruktion SET FORMAT TO &format&ewahl zu SET FORMAT TO &format1 und nach dem Ersetzen des Makros zu dem Befehl SET FORMAT TO coa. Wir können dann den Befehl „&befehl&subbefehl" geben, woraus zuerst „&befehle" und nach der Ersetzung des Makros dann APPEND wird. Diese CASE-Anweisung erledigt das Einfügen und Ändern für alle Datenbanken, indem es EWAHL und SUBBEFEHL als Makros benutzt, um die Makros FORMAT und BEFEHL mit dem richtigen Inhalt zu versehen:

```
*** Wenn Eingabe oder Änderung, benutze Ändern-Maske
CASE subbefehl = "E" .OR. subbefehl = "V"
     SET FORMAT TO &format&ewahl
     &befehl&subbefehl
     SET FORMAT TO SCREEN
```

Wünscht der Benutzer zu löschen, markiert das Programm den Datensatz als gelöscht und beseitigt ihn mit der zweiten CASE-Anweisung (mit dem Befehl PACK) endgültig aus der Datenbank:

```
   ***** Wenn löschen, markiere als gelöscht und
   ***** dann PACK
   CASE subbefehl = "L"
        ? "Lösche Satz "+STR(#,4)
        DELETE
        PACK
ENDCASE
```

Wurde der gesuchte Datensatz überhaupt nicht gefunden, gibt das Programm eine Warnmeldung aus:

```
ELSE
     ****** Wenn kein Satz gefunden, warnen
```

```
        § 10,10 SAY "Nicht gefunden!"
        ? CHR(7)
```

Jetzt müssen wir nur noch die IF-Anweisung und die DO WHILE-Schleife zur Anzeige des Menüs beenden:

```
      ENDIF (#>0 and not eof)
    ENDIF (wahl # 5)
ENDDO (while ewahl # 5)
```

Sobald der Benutzer das Programm zu beenden wünscht, löscht es die Variablen und kehrt zum Hauptmenü zurück:

```
************** Lösche Variablen und kehre zum
***************** Hauptmenü zurück
RELEASE ALL LIKE datei?
RELEASE ALL LIKE format?
RELEASE ALL LIKE befehl?
RELEASE ewahl,finden,finden2,zaehl
RELEASE subbefehl,wieviele
RETURN
```

Die Abb. 19.2 zeigt das gesamte BEDIT.CMD-Programm.

```
************************************ BEDIT.CMD
************************ Änderungsprogramm für die Buchhaltung

**************** Erstelle Makros, um Datenbanken und Format-
**************** Dateien einer Menüwahl zuzuordnen
STORE "coa INDEX coa" TO datei1
STORE "coa" TO format1
STORE "trans INDEX konto,datum" TO datei2
STORE "trans" TO format2
STORE "register" TO datei3,format3
STORE "geninfo" TO datei4,format4
***************************** Speichere Befehle
STORE "APPEND" TO befehle
STORE "EDIT #" TO befehlv

**************** Schleife für Menü beginnen
STORE " " TO ewahl
DO WHILE ewahl  #"5"
   ERASE
   TEXT
```

Abb. 19.2: Das Programm BEDIT.CMD

```
                    Buchhaltung-Änderungen

                1.   Kontenrahmen
                2.   Buchungsdatei
                3.   Scheckregister
                4.   Allgemeine Informationen

                5.   Zurück zum Hauptmenü

ENDTEXT
§ 11,12 SAY "Ihre Wahl (1-5)" GET ewahl
READ

************** Sicherstellen, daß gültige Eingabe gemacht
***************wurde
IF ewahl < "1" .OR. ewahl > "5"
   **** Wenn nicht, zeige Fehlermeldung und dann wieder das
   **** Menü
   § 13,12 SAY "** Ungültige Eingabe **"
   ? CHR(7)
   *************** Zähle bis 20, um Zeit zu verbrauchen
   STORE 1 TO dauer
   DO WHILE dauer < 21
      STORE dauer+1 TO dauer
   ENDDO (dauer<21)
   LOOP
ENDIF (falsche Eingabe)

********************** Wenn nicht Ende, hole Informationen,
********************** um Datensatz zu finden
IF ewahl # "5"
   ERASE
   USE &datei&ewahl
   STORE " " TO subbefehl
   IF ewahl="4"
      STORE "V" TO subbefehl
   ENDIF (ewahl=4)
   DO CASE

      ***************** Wenn COA.DBF gewünscht, frage nach
      ************** Befehl, dann suche nach Kontonummer
      CASE ewahl = "1"
        ERASE
        § 3,5 SAY "(E)ingeben, (V)erändern oder (L)öschen?";
              GET subbefehl PICT "!"
        READ
        IF subbefehl # "E"
        STORE "      " TO finden
```

Abb. 19.2: Das Programm BEDIT.CMD (Forts.)

```
      § 5,5 SAY "Welche Kontonummer?";
            GET finden PICT "999.99"
      READ
      FIND &finden
    ENDIF (befehl # e)

  *************** Wenn Buchung bearbeitet wird, suche nach
  ***************** Konto und Datum
  CASE ewahl = "2"
    § 3,5 SAY "(V)erändern oder (L)öschen?";
          GET subbefehl PICT "!"
    STORE "       " TO finden
    STORE "         " TO finden2
    § 5,5 SAY "Welche Kontonummer ";
          GET finden PICT "999.99"
    § 7,5 SAY "Welches Datum?";
          GET finden2 PICT "99/99/99"
    READ

    *************** Ermittle genau Anzahl der Sätze
    COUNT FOR konto = &finden .AND. datum = finden2 TO
    wieviele

    IF wieviele > 1
       ERASE
       LIST FOR konto = &finden .AND. datum = finden2
                                                    konto,;
                datum,grund,betrag
       ?
       INPUT "Welchen Datensatz?" TO satznr
       GOTO satznr

    ELSE
      LOCATE FOR konto = &finden .AND. datum = finden2
    ENDIF (wieviele>1)

  ************* Wenn REGISTER, suche nach Schecknummer
  CASE ewahl = "3"
    § 3,5 SAY "(E)ingeben, (V)erändern oder (L)öschen?";
          GET subbefehl PICT "!"
    READ
    IF subbefehl  #"E"
       STORE "     " TO finden
       § 5,5 SAY "Welche Schecknummer?";
             GET finden PICT "9999"
       READ
       LOCATE FOR scheck:nr = &finden
    ENDIF (befehl # a)
```

Abb. 19.2: Das Programm BEDIT.CMD (Forts.)

```
        ENDCASE

        ********************** Wenn Datensatz gefunden, ändern oder
        ************************ löschen ermöglichen
        IF # > 0 .AND. .NOT. EOF

           DO CASE

              *** Wenn Eingabe oder Änderung, benutze Ändern-Maske
              CASE subbefehl = "E" .OR. subbefehl = "V"
                SET FORMAT TO &format&ewahl
                &befehl&subbefehl
                SET FORMAT TO SCREEN

                ***** Wenn löschen, markiere als gelöscht und dann
                ***** PACK
              CASE subbefehl = "L"
                ? "LÖsche Satz "+STR(#,4)
                DELETE
                PACK

           ENDCASE

        ELSE
          ****** Wenn kein Satz gefunden, warnen
          § 10,10 SAY "Nicht gefunden!"
          ? CHR(7)

        ENDIF (#>0 and not eof)
     ENDIF (wahl # 5)
ENDDO (while ewahl # 5)

************** Lösche Variablen und kehre zum
***************** Hauptmenü zurück
RELEASE ALL LIKE datei?
RELEASE ALL LIKE format?
RELEASE ALL LIKE befehl?
RELEASE ewahl,finden,finden2,zaehl
RELEASE subbefehl,wieviele,satznr,dauer
RETURN
```

Abb. 19.2: Das Programm BEDIT.CMD (Forts.)

Berichtigungsbuchungen

Da das Programm die Bilanzen der Kontenübersicht direkt mit den Daten der TRANS-Datei aktualisiert, spiegeln diese Beträge immer eine fortlaufende Zusammenfassung der einzelnen Buchungen wieder. Das

ermöglicht eine chronologische Aufstellung aller Bilanzen. Geht der Benutzer direkt in die Kontenübersicht und verändert dort eine der Bilanzen, zerstört er damit die chronologische Fortschreibung.

Um aber Änderungen vorzunehmen und trotzdem die Chronologie zu bewahren, kann er Berichtigungsbuchungen in der TRANS-Datei speichern. Wir brauchen dafür kein Programm zu schreiben, da diese Möglichkeit bereits in das System eingebaut ist.

Lassen Sie uns davon ausgehen, daß der Benutzer 500 DM auf das Konto 100 (Einkommen) bucht, später aber feststellt, daß dieser Betrag eigentlich auf das Konto 110 (Lizenzeinnahmen) gehört. Damit ist die Bilanz für Konto Nummer 100 um 500 DM zu hoch und für Konto 110 um den gleichen Betrag zu niedrig. Der Benutzer wählt nun die Option 1 des Hauptmenüs (Neue Buchungen eingeben) und fügt zwei Berichtigungsbuchungen folgendermaßen durch:

```
Kontonummer   :100.00:   Einkommen
Beschreibung  :Berichtigung      :
Betrag : -500.00:
Datum  :12/12/84:
```

Die zweite Buchung sieht so aus:

```
Kontonummer   :100.10:   Lizenzeinnahmen
Beschreibung  :Lizenzeinn.       :
Betrag :  500.00:
Datum  :12/12/84:
```

Wenn er das nächste Mal die Kontenübersicht aktualisiert, werden 500 Mark vom Konto 100 abgezogen (da dieser Betrag negativ eingegeben wurde) und 500 Mark werden zu dem Konto 110 addiert, so daß die Bilanzen wieder korrekt sind. Es gibt außerdem einen gespeicherten Eintrag in der Datenbank über diese Berichtigung.

Genauso verhält es sich, wenn der Benutzer unbeabsichtigt 10.000 Mark auf das Konto 100 gebucht hat, aber 1.000 Mark buchen wollte. Dann kann er eine einfache Berichtigungsbuchung zur Behebung des Fehlers machen. In diesem Fall würde er 9.000 Mark vom Konto 100 abziehen, indem er die Buchung so eingibt:

```
Kontonummer   :100.00:   Einkommen
Beschreibung  :Berichtigung      :
Betrag :-9000.00:
Datum  :12/12/84:
```

Wie in dem vorherigen Beispiel wird die Kontenübersicht korrigiert und in der TRANS-Datei steht ein gespeicherter Eintrag über die Berichtigung.

Erweiterungsmöglichkeiten der Buchhaltung

Unser Buchhaltungssystem ist ziemlich einfach gehalten, verglichen mit den Buchhaltungen, die in vielen Geschäftsbereichen eingesetzt werden. Aber auch unser System ist noch ausbaubar. Um zum Beispiel eine doppelte Buchführung zu ermöglichen, braucht die TRANS-Datenbank zwei Kontonummern für jede Buchung, ein Debitoren- und ein Kreditorenkonto. Bei einer Aktualisierung müssen dann beide Konten in der Konten-Datenbank (COA.DBF) aktualisiert werden.

Viele Buchhaltungssysteme erlauben es dem Benutzer, unter einer Kontonummer Gesamtausgaben für einen bestimmten Bereich anzuzeigen. Diese Konten geben automatisch alle Summen der entsprechenden Unterkonten in einem Betrag wieder. Zum Beispiel zeigt das Konto Verkaufsausgaben die Gesamtsumme der Unterkonten, die es enthält:

Verkaufsausgaben	30.700 DM
Gehälter	18.500 DM
Miete, Verkaufsfläche	8.100 DM
Werbung	700 DM
Lagerkosten	400 DM
Abschreibung, Lager	3.000 DM

Wenn Sie ein System entwickeln wollen, das diese automatische Zusammenstellung eines Kontos ermöglicht, müssen Sie zuerst festlegen, wie diese Gesamtsummenkonten zu erkennen sind. Sie können so vorgehen, daß Sie alle Kontennummern, die glatt durch 100 teilbar sind und den Wert null in den Dezimalstellen haben, für diese Aufgabe reservieren. Dann müssen Sie festlegen, wie die Unterkonten, die zu einem solchen Oberkonto gehören, zu identifizieren sind. Hier eine einfache Möglichkeit: Die Unterkonten haben die gleiche Kontonummer wie das Oberkonto, aber die Dezimalstellen sind größer als null. Die Konten in der Tabelle verdeutlichen dieses Schema:

100.00	Verkaufsausgaben	30.700 DM
100.10	Gehälter	18.500 DM
100.20	Miete,Verkaufsfläche	8.100 DM
100.30	Werbung	700 DM
100.40	Lagerkosten	400 DM
100.50	Abschreibung, Lager	3.000 DM

Dann müssen Sie eine Routine schreiben, die alle Summen der Unterkonten in der Kontenübersicht findet und die Gesamtsumme berechnet. Die hier vorgestellte Routine macht das mit dem Numerierungsystem, das ich Ihnen gerade vorgestellt habe:

```
USE trans INDEX konto
STORE 0 TO sum:konto
DO WHILE .NOT. EOF
   STORE 100.00+sum:konto TO sum:konto
   STORE STR(sum:konto,6,2) TO finden
   FIND &finden
   IF # > 0
      REPLACE betrag WITH 0
      SUM betrag WHILE INT(konto)=sum:konto;
                                          TO sub:konto
      FIND &finden
      REPLACE betrag WITH sub:konto
   ENDIF (#>0)
ENDDO (not eof)
```

Die Routine schreibt die Gesamtsummen in ein Oberkonto, indem es jede Kontonummer findet, die glatt durch 100 teilbar ist. Zuerst beseitigt es die alten Beträge in diesem Konto, indem es eine Null hineinschreibt. Dann summiert es die Beträge der Datensätze, deren Kontonummer vor dem Dezimalpunkt gleich der Kontonummer des Oberkontos ist. Die Gesamtsumme der Unterkonten wird in der Variablen SUM:KONTO gespeichert. Dann schreibt das Programm den Betrag von SUM:KONTO in das Feld Betrag des Oberkontos.

Zusammenfassung

In diesem Kapitel haben wir unser Buchhaltungssystem vervollständigt. Viele der benutzten Techniken gleichen denen, die wir für das Inventarsystem benutzt haben, speziell jene, die ein Aktualisieren der Hauptdatei (COA.DBF) mit den Daten derBuchungen-Datei (TRANS.DBF) nach sich ziehen. Wir haben auch ähnliche Techniken benutzt, um fehlerhafte Eingaben auszuschließen und nur korrekte Kontonummern zuzulassen.

Der Hauptunterschied zwischen dem Inventar- und dem Buchhaltungsprogramm liegt in den auftauchenden Problemen bei der Arbeit mit Einnahmen und Ausgaben sowie Unter- und Hauptkonten. Diese Probleme können aber gelöst werden, wenn alle Programme des Systems die unterschiedlichen Konten leicht erkennen können, ohne durcheinander zu geraten. Wenn Sie ein Kontenschema aufbauen, das mit dBASE-Befeh-

len leicht zu kontrollieren ist, können Sie Programme schreiben, die viele verschiedene Kontentypen bearbeiten, ohne lange DO CASE- und IF-Anweisungen zu verwenden.

In diesem Kapitel haben wir auch den fortgeschrittenen Einsatz von Makros kennengelernt. In dem Programm zum Schreiben von Schecks haben wir Makros zum Übersetzen von numerischen Werten in Zahlwörter eingesetzt. Im Änderungsmodul des Systems benutzen wir ein einziges Programm, um die Änderungen in vier verschiedenen Datenbanken durchzuführen. Wir haben auch ein allgemeines Änderungsprogramm entwickelt, das die entsprechende Datenbank und Formatdatei sowie die notwendigen Befehle aufgrund der Menüeingaben des Benutzers aussucht.

Ein Literatur-Verwaltungs-Programm

20

In diesem Kapitel werden wir ein Programm erstellen, mit dem man eine Bibliothek verwalten kann. Hier sehen Sie einige neue Programmier-Techniken, die wir näher betrachten werden:

1. Möglichkeiten für den Benutzer, mit einem einfachen Menü nach komplizierten Kriterien zu suchen;
2. Text in Flattersatz zu formatieren;
3. Berichte in zwei Spalten zu drucken;
4. das Aufbauen einer Stichwort-Datei, um die Suche zu erleichtern;
5. die Umwandlung eines Datums (von 20/04/85 in 20. April 1985).

Die Vorteile der Literatur-Verwaltung

Die Literatur-Verwaltung ist für jeden nützlich, der schnellen Zugriff auf gedruckte Information benötigt. Der Benutzer kann nach Einträgen suchen, wenn er entweder den Autor, den Titel, den Verlag, das Erscheinungsdatum oder eines der sechs Stichwörter kennt. Die Einträge können im Bibliotheks-Format ausgedruckt und direkt in ein Dokument übertragen werden.

Mit Hilfe von benutzerfreundlichen Menüs geben Sie neue Daten ein oder verändern bereits gespeicherte. Befehle, wie Sie den Cursor bewe-

gen oder andere Aktionen veranlassen, stehen ebenfalls in den Menüs. Sie können selbst komplizierte Suchoperationen einfach dadurch einleiten, daß Sie die Felder in den Menüs ausfüllen. Sie bestimmen die Suchfelder und Werte, ob Sie die logischen Operatoren (.AND.) für „und" oder (.OR.) für „oder" gebrauchen wollen und ob die Stichwörter und der Textauszug mitgedruckt werden sollen. Weiter wählen Sie aus, ob das Ergebnis Ihrer Suche ausgedruckt, auf dem Bildschirm angezeigt oder in eine Datei geschrieben werden soll. Den Inhalt der Datei können Sie dann hinterher leicht mit Ihrem Textprogramm in ein Dokument übernehmen.

Die von dem Literatur-Verwaltungs-Programm erstellten Berichte sehen wie von Hand geschrieben aus. Beispielsweise wird der Text mit einem rechten Flatterrand gedruckt, wie Sie ihn auch mit einer Schreibmaschine schreiben würden. Wenn Sie suchen oder sortieren wollen, geben Sie das Datum in der dBASE-Form "TT/MM/JJ" ein, aber im Report erscheint es in der ausgeschriebenen Form als "TT. Monat Jahr".

Die Struktur der Datenbank

Das Literatur-Verwaltungs-Programm besteht aus zwei Datenbanken. Die erste heißt LIBRARY.DBF und enthält die Daten von Zeitungsartikeln und Büchern. Hier die Struktur:

```
STRUKTUR FÜR DATEI: B:LIBRARY.DBF

FELD  NAME        TYP   WEITE   DEZ     BEDEUTUNG
001   AUTOR        C      030            Name des Autors
002   TITEL        C      040            Titel des Buches/Berichtes
003   VERLAG       C      030            Verlag/Zeitschrift
004   AUSGABE      C      020            Ausgabe
005   FUND         C      020            Fundstelle (Buch)
006   SEITEN       C      008            Seiten (Artikel)
007   DATUM        C      008            Datum (TT/MM/JJ)
008   SWORT1       C      025            Stichwort
009   SWORT2       C      025            Stichwort
010   SWORT3       C      025            Stichwort
011   SWORT4       C      025            Stichwort
012   SWORT5       C      025            Stichwort
013   SWORT6       C      025            Stichwort
014   ZUSAMMEN     C      244            Zusammenfassung
```

Für das Sortieren und Suchen wird zuerst nach den Autorennamen sortiert. Sie können aber auch die Beiträge in chronologischer Reihenfolge auflisten. Die Index-Dateien heißen AUTOR und DATUM und werden so erstellt:

```
USE library
INDEX ON !(autor) TO autor
INDEX ON $(datum,7,2)+$(datum,1,6) TO datum
```

Beachten Sie bitte, daß der Index nach Autorennamen in Großbuchstaben erstellt wird und das Indizieren des Datums mit dem Jahr beginnt, damit die Reihenfolge stimmt.

Das Programm erstellt auch eine Liste aller Stichwörter in alphabetischer Reihenfolge. Zu jedem Stichwort bekommen Sie die Information, wie oft es insgesamt in der Datenbank vorkommt. Die Struktur der Datenbank sieht so aus:

```
STRUKTUR FÜR DATEI: B:TOPICS.DBF

FELD  NAME       TYP   WEITE   DEZ    BEDEUTUNG
001   SUBJECT     C      025           Stichwort
002   ANZAHL      N      004           Häufigkeit
```

Diese beiden Datenbanken und die zwei Index-Dateien sollten Sie erstellen, noch bevor Sie mit dem Schreiben und Testen irgendeines Programms beginnen.

Die Dateien des Literatur-Verwaltungs-Systems

Die Literatur-Verwaltung besteht aus diesen Dateien:

1. LIBRARY.CMD, das Hauptmenü,
2. LIBEDIT.CMD, zum Ändern,
3. LIBDEL.CMD, zum Löschen,
4. LIBSEARC.CMD, zum Suchen und Drucken der Reports,
5. SUBJECTS.CMD, zum Erstellen der Stichwort-Liste.

Dann sind da noch die drei Menü-Dateien (FMT):

1. LIBADD.FMT, zum Eingeben neuer Daten,
2. LIBEDIT.FMT, zum Ändern,
3. LIBSEARC.FMT, zum einfachen Suchen.

Wie die Beziehungen der verschiedenen Dateien untereinander sind, sehen Sie in Abb. 20.1.

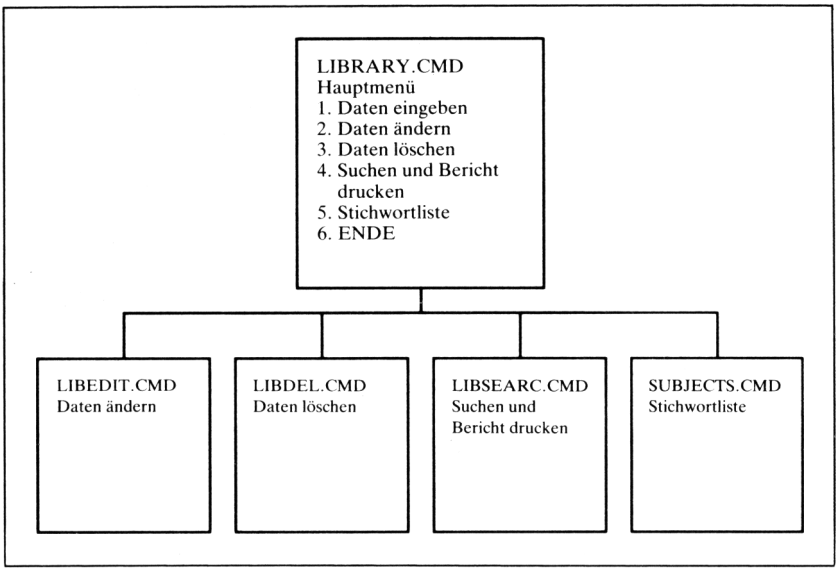


Abb. 20.1: Programmstruktur des Literatur-Verwaltungs-Systems

Das Hauptmenü der Literatur-Verwaltung

Das Hauptmenü zeigt dem Benutzer folgende Wahlmöglichkeiten:

```
>>>  Literatur-Verwaltungs-System  <<<

        1. Einträge neu eingeben
        2. Einträge ändern
        3. Einträge löschen
        4. Einträge suchen
        5. Stichwort-Liste drucken

        6. ENDE
```

Der Hauptmenüteil des Literatur-Programms ist fast identisch mit denen der anderen Programme, die wir bisher entwickelt haben. Ein Unterschied besteht darin, daß es nicht zu einem anderen Programm verzweigt, wenn der Benutzer neue Daten eingeben möchte. Statt dessen wird mit SET FORMAT TO die Datei LIBADD.FMT aufgerufen, die Daten werden an die Datenbank „angehängt", und das Programm kehrt zum Bildschirm zurück (beachten Sie den Programmteil ab CASE WAHL= '1' in der Abb. 20.2). Das LIBRARY.CMD-Programm sehen Sie in Abb. 20.2.

```
********************************** LIBRARY.CMD
************ Hauptmenü für das Literatur-Verwaltungs-System
SET TALK OFF
SET DEFAULT TO a
STORE ' ' TO wahl
DO WHILE wahl # '6'
   ERASE
   TEXT

              >>> Literatur-Verwaltungs-System <<<

                    1. Einträge neu eingeben
                    2. Einträge ändern
                    3. Einträge löschen
                    4. Einträge suchen
                    5. Stichwort-Liste drucken

                    6. ENDE

   ENDTEXT
   § 11,10 SAY 'Wählen Sie aus (1-6) ' GET wahl
   READ

   *** Entsprechend der Auswahl reagieren
   DO CASE
      CASE wahl = '1'
         USE library INDEX autor,datum
         SET FORMAT TO libadd
         APPEND
         SET FORMAT TO screen

      CASE wahl = '2'
         DO libedit

      CASE wahl = '3'
         DO libdel

      CASE wahl = '4'
         DO libsearc

      CASE wahl = '5'
         DO subjects

   ENDCASE

ENDDO (WHILE wahl # 6)
QUIT
```

Abb. 20.2: Das Programm LIBRARY.CMD

Neue Einträge aufnehmen

Sie müssen auch noch die Menü-Datei LIBADD.FMT erstellen, damit Sie neue Daten eingeben können. Dazu benutzen Sie am besten das Programm ZIP, mit dem sich Menüs spielend leicht erstellen lassen. Die Abb. 20.3 zeigt Ihnen, wie das Menü mit ZIP erstellt wird. Die Datei LIBADD.FMT erstellt das Programm dann automatisch.

Der Benutzer kann Einträge zu Zeitschriften-Beiträgen oder zu Büchern über dieses Menü machen. Grundsätzlich sollten Sie für ein Buch die Felder Autor, Titel, Verlag und Erscheinungsort eingeben. In der Abb. 20.5 sehen Sie solch einen Eintrag für ein Buch.

Für Beiträge aus Zeitschriften sollten Sie grundsätzlich den Autor, Titel, Zeitschriften-Name, Ausgabe, Seite und Datum festhalten. In Abb. 20.6 sehen Sie ein ausgefülltes Menü für einen Zeitschriften-Beitrag.

Einträge ändern

Die Datei LIBEDIT.CMD ermöglicht zusammen mit der Menü-Datei LIBEDIT.FMT die Änderung von Einträgen im Literatur-Verwaltungs-

```
                    >>> Ä n d e r n <<<
-------------------------------------------------------------
Cursor-Befehle: ^E hoch : ^X runter: ^D rechts : ^S links :
^V einfügen ^G löschen: ^C weiter : ^R zurück:
>> ^W SPEICHERN << : ^Q abbrechen
-------------------------------------------------------------
Autor #autor

Titel #titel

Verlag (Publ.) #verlag                Ausgabe #ausgabe

Datum #datum      Seiten#pages      Ersch.-ort#ort
Stichwörter:
----------
1. #swort1              2. #swort2
3. #swort3              4. #swort4
5. #swort5              6. #swort6

Zusammenfassung:
---------------
#zusammen
```

Abb. 20.3: Der ZIP-Bildschirm der LIBADD.FMT-Datei

Programm. Sie können das Menü wieder mit ZIP erstellen und finden in Abb. 20.7 ein Beispiel für den Aufbau.

Haben Sie die Datei erstellt, schreibt Ihnen ZIP wieder automatisch das entsprechende dBASE-Programm. Die Programm-Liste finden Sie in Abb. 20.8.

```
* LIBADD.FMT
§  1,23 SAY ">>>  N e u a u f n a h m e  <<<"
§  2, 1 SAY "=================================================
             ========================="
§  3, 1 SAY "Cursor Befehle: ^E hoch : ^X runter : ^D rechts :
            ^S links"
§  4,15 SAY ": ^G Zeichen löschen  : ^V Einfügen"
§  5, 1 SAY "=================================================
             ========================="
§  6, 2 SAY "Autor"
§  6, 9 GET autor
§  8, 2 SAY "Titel"
§  8, 9 GET titel
§ 10, 2 SAY "Verlag (Publ.)"
§ 10,17 GET verlag
§ 10,49 SAY "Ausgabe"
§ 10,56 GET ausgabe
§ 12, 2 SAY "Datum"
§ 12, 9 GET datum
§ 12,21 SAY "Seiten"
§ 12,27 GET seiten
§ 12,47 SAY "Ersch.-Ort"
§ 12,57 GET ort
§ 13, 2 SAY "Stichwörter:"
§ 14, 2 SAY "-----------"
§ 15, 1 SAY "1."
§ 15, 5 GET swort1
§ 15,37 SAY "2."
§ 15,41 GET swort2
§ 16, 1 SAY "3."
§ 16, 5 GET swort3
§ 16,37 SAY "4."
§ 16,41 GET swort4
§ 17, 1 SAY "5."
§ 17, 5 GET swort5
§ 17,37 SAY "6."
§ 17,41 GET swort6
§ 19, 1 SAY "Zusammenfassung:"
§ 20, 1 SAY "---------------"
§ 21, 2 GET zusammen
```

Abb. 20.4: Das Programm LIBADD.FMT

Die Befehls-Datei LIBEDIT.CMD fragt den Benutzer zuerst nach dem ersten Autoren-Namen des Datensatzes, der geändert werden soll:

```
Welchen ändern? (Ersten Autoren-Namen):
```

Geben Sie RETURN als Antwort ein, hält das Programm an und kehrt in das Hauptmenü zurück. Geben Sie einen Autoren-Namen ein, für den noch kein Datensatz erstellt wurde, erhalten Sie die Meldung:

```
Nicht gefunden
```

und können es noch einmal versuchen. Ist der Datensatz vorhanden, wird der erste Eintrag für diesen Autoren angezeigt und Sie können nach Herzenslust ändern. Zusätzlich besteht die Möglichkeit, mit Control-C den nächsten Datensatz zu holen oder mit Control-R den vorherigen. Da die Datenbank nach den Autoren-Namen indiziert ist, erscheinen die Einträge in alphabetischer Reihenfolge.

```
                    >>> N e u a u f n a h m e <<<

Cursor-Befehle: ^E hoch : ^X runter : ^D rechts : ^S links
                : ^G Zeichen löschen : ^V Einfügen

Autor  :Simpson                    :

Titel :  Arbeiten mit dBASE II                :

Verlag (Publ.) :SYBEX                  : Ausgabe:                :

Datum :00/03/84:  Seiten:      :     Ersch.-Ort:Düsseldorf      :
Stichwörter:
-----------
1. :dBASE II                   :  2. :Datenverwaltung          :
3. :Adressverwaltung           :  4. :Mehrere Datenbanken      :
5. :Programmieren              :  6. :                         :

Zusammenfassung:
---------------
:Eine Einführung in dBASE für den unerfahrenen Computer-       :
:Benutzer oder den dBASE-Anfänger. Enthält viele Beispiele     :
:über das Erstellen, Verändern, Modifizieren, Sortieren und    :
:Suchen mit und in einer Datenbank.                            :
```

Abb. 20.5: Eine ausgefüllte Buchreferenz

Der Pseudocode für die Datei LIBEDIT.CMD ist in Abb. 20.9 zu sehen.

Die Kommando-Datei LIBEDIT.CMD beginnt mit dem Aufruf der Datenbank LIBRARY zusammen mit den Index-Dateien AUTOR.NDX und DATUM.NDX. Das sieht so aus:

```
********************************** LIBEDIT.CMD
************************ Einträge verändern
USE library INDEX autor,datum
```

Danach beginnt eine DO WHILE-Schleife, damit Sie entweder mehrere Einträge ändern können oder durch ein einfaches RETURN, als Eingabe für den Autoren-Namen, wieder zum Hauptmenü gelangen:

```
*** Schleife zum Ändern beginnen
STORE 'x' TO suchen
DO WHILE suchen # ' '
```

```
                    >>> N e u a u f n a h m e <<<
-------------------------------------------------------------------
Cursor-Befehle: ^E hoch : ^X runter : ^D rechts : ^S links
                : ^G Zeichen löschen : ^V Einfügen
-------------------------------------------------------------------
Autor  :Schmidt,P. und Paul,J                    :

Titel :  ACCESS MANAGER 86:eine Beschreibung     :

Verlag (Publ.) :Microsystems     : Ausgabe:Ausg. 1, Nr. 4     :

Datum :00/03/84:  Seiten:12-17  :     Ersch.-Ort:            :
Stichwörter:
----------
1. :Menügestaltung          :  2. :Indizieren          :
3. :CB-86                   :  4. :Sortieren           :
5. :                        :  6. :                    :

Zusammenfassung:
---------------
:Eine Beschreibung des Programms ACCESS MANAGER,  :
:Programm zum Wiederherstellen beschädigter Daten und :
:Dateien. Dazu noch verschiedene Programme, geschrieben in :
:der kompilierbaren CB-86-Programmiersprache.     :
```

Abb. 20.6: Eine Zeitschriften-Referenz

```
                    >>> Ä n d e r n <<<

Cursor-Befehle: ^E hoch : ^X runter: ^D rechts : ^S links
: ^V einfügen ^G löschen: ^C weiter : ^R zurück
: ¯ ^W SPEICHERN << : ^Q abbrechen

Autor   #autor

Titel   #titel

Verlag (Publ.) #verlag                          Ausgabe#ausgabe

Datum #datum          Seiten#pages          Ersch.-Ort#ort
Stichwörter:
----------
1. #swort1                    2. #swort2
3. #swort3                    4. #swort4
5. #swort5                    6. #swort6

Zusammenfassung:
---------------
#zusammen
```

Abb. 20.7: ZIP-Bildschirm für die Formatdatei LIBEDIT.FMT

```
* LIBEDIT.FMT
§  0,23 SAY ">>>  Ä n d e r n  <<<"
§  1, 1 SAY "============================================
             ================================="
§  2, 1 SAY "Cursor Befehle: ^E hoch : ^X runter: ^D rechts :
            ^S links : ^V einfügen"
§  3, 3 SAY "^G löschen: ^C weiter : ^R zurück: >> ^W SPEICHERN
            <<  : ^Q abbrechen"
§  4, 1 SAY "============================================
             ================================="
§  5, 2 SAY "Autor"
§  5, 9 GET autor
§  7, 2 SAY "Titel"
§  7, 9 GET titel
§  9, 2 SAY "Verlag (Publ.)"
§  9,17 GET verlag
```

Abb. 20.8: Die Formatdatei LIBEDIT.FMT

```
§  9,50 SAY "Ausgabe"
§  9,57 GET ausgabe
§ 11, 2 SAY "Datum"
§ 11, 9 GET datum
§ 11,24 SAY "Seiten"
§ 11,30 GET pages
§ 11,50 SAY "Ersch.-Ort"
§ 11,60 GET ort
§ 12, 2 SAY "Stichwörter:"
§ 13, 2 SAY "-----------"
§ 14, 1 SAY "1."
§ 14, 5 GET swort1
§ 14,37 SAY "2."
§ 14,41 GET swort2
§ 15, 1 SAY "3."
§ 15, 5 GET swort3
§ 15,37 SAY "4."
§ 15,41 GET swort4
§ 16, 1 SAY "5."
§ 16, 5 GET swort5
§ 16,37 SAY "6."
§ 16,41 GET swort6
§ 18, 1 SAY "Zusammenfassung:"
§ 19, 1 SAY "---------------"
§ 20, 2 GET zusammen
```

Abb. 20.8: Die Formatdatei LIBEDIT.FMT (Forts.)

```
STORE '                                    ' TO suchen
ERASE
§ 5,5 SAY 'Welchen ändern (ersten Autornamen)?';
GET suchen
§ 6,10 SAY '<Mit RETURN zurück>'
READ
```

Die Befehlsdatei übersetzt dann den Autoren-Namen in Großbuchstaben, weil der Index noch in normaler Schreibweise ist. Dann beseitigt das Programm die Leerstellen, die der READ-Befehl übriggelassen hat. Dadurch sind Sie in der Lage, nur den Nachnamen des ersten Autors einzugeben, wenn sie nach einem Eintrag suchen. Geben Sie zum Beispiel Schmidt ein, sollte der FIND-Befehl ohne Schwierigkeiten einen Beitrag des Herrn Schmidt finden:

```
Schmidt,P. und Paul,J.
```

```
Die Datenbank LIBRARY und die Index-Dateien eröffnen
Editier-Schleife eröffnen
   Nach dem gesuchten Autor fragen

   Den ersten Artikel des Autors finden

      Wenn nichts geändert werden soll, Bildschirm löschen
      und zum Hauptmenü zurück

      Wenn Eintrag gefunden, Ändern ermöglichen und Menü-
      Datei LIBEDIT.FMT aufrufen

      Wenn Eintrag nicht gefunden, Benutzer warnen und
      noch einmal versuchen

Editier-Schleife weiterführen (bis Ende befohlen)

Memory-Variablen löschen
Zurück zum Hauptmenü
```

Abb. 20.9: Pseudocode für LIBEDIT.CMD

Die nächsten Zeilen übersetzen den Namen in Großbuchstaben, beseitigen die Leerzeichen und suchen nach dem erfragten Eintrag:

```
*** Ersten Artikel des Autors finden
STORE !(TRIM(suchen)) TO suchen
FIND &suchen
```

Jetzt müssen wir eine DO CASE-Schleife aufbauen, damit der Benutzer weitermachen kann. Gibt der Benutzer nichts ein, soll das Programm einfach über die CASE-Anweisungen hinweggehen und den Bildschirm löschen. Hier sind die Zeilen, die das ermöglichen:

```
DO CASE
     *** Wenn RETURN gedrückt, Bildschirm löschen
     CASE suchen = ' '
     ERASE
```

Wenn das Programm einen Eintrag findet, ruft es die Menüdatei LIBEDIT.FMT auf, ermöglicht alle Änderungen und schaltet dann wieder zurück, wie Sie hier sehen:

```
*** Wenn Eintrag gefunden, Änderungs-Modus anwählen
CASE # > 0
  SET FORMAT TO libedit
  EDIT #
  SET FORMAT TO screen
```

Wenn das Programm keinen Eintrag findet, bekommen Sie eine Warnung. Diese Programmzeilen sorgen dafür:

```
*** Wenn Eintrag nicht gefunden, Benutzer warnen
*** und nochmal versuchen
 CASE # = 0
    § 10,5 SAY 'nicht gefunden ...!'
    ?
    WAIT
ENDCASE
```

Dann brauchen wir nur noch Befehle, die die Editier-Schleife wieder schließen, die Variablen löschen und zum Hauptmenü zurückführen:

```
ENDDO (WHILE suchen # ' ' )
RELEASE suchen
RETURN
```

Das vollständige Programm LIBEDIT.CMD ist in Abb. 20.10 zu sehen.

```
********************************** LIBEDIT.CMD
************************ Einträge verändern
USE library INDEX autor,datum

*** Schleife zum Ändern beginnen
STORE 'x' TO suchen
DO WHILE suchen # ' '
   STORE '                                ' TO suchen
   ERASE
   § 5,5 SAY 'Welchen ändern (Autor)? ' ;
   GET suchen
   § 6,10 SAY '<Mit RETURN zurück>'
   READ

   *** Ersten Artikel des Autors finden
   STORE !(TRIM(suchen)) TO suchen
```

Abb. 20.10: Das Programm LIBEDIT.CMD

```
    FIND &suchen
    DO CASE
          *** Wenn RETURN gedrückt, Bildschirm löschen
       CASE suchen = ' '
          ERASE

          *** Wenn Eintrag gefunden, Änderungs-Modus anwählen
       CASE # > 0
          SET FORMAT TO libedit
          EDIT #
          SET FORMAT TO screen

          *** Wenn Eintrag nicht gefunden, Benutzer warnen und
          *** noch einmal versuchen
       CASE # = 0
           § 10,5 SAY 'nicht gefunden ...!'
          ?
          WAIT

  ENDCASE 0    36:

ENDDO (WHILE suchen  #' ' )
RELEASE suchen
RETURN
```

Abb. 20.10: Das Programm LIBEDIT.CMD (Forts.)

Einträge löschen

Um Einträge aus dem Literatur-Verwaltungs-System zu löschen, benutzen wir das gleiche Programm (DELNAMES) wie weiter vorne beim Löschen der Namen in der Adressen-Verwaltung. Sehen Sie für eine ausführliche Beschreibung bitte in Kapitel 7 nach. Hier zeige ich Ihnen einfach nur den Programmtext (Abb. 20.11). Das Programm heißt LIBDEL.CMD, und Sie erstellen und sichern es wie jedes andere Programm.

```
********************************* LIBDEL.CMD
************************** Einträge löschen
USE library INDEX autor,datum

*** Den Namen des zu löschenden Autors holen
STORE 'x' TO suchen
DO WHILE suchen # ' '
```

Abb. 20.11: Das Programm LIBDEL.CMD

```
    ERASE
    ACCEPT 'Welchen löschen? (Autoren-Name) ' ;
    TO suchen
    ERASE

    *** Übersetze in Großbuchstaben, dann zähle,
    *** wie oft der Name vorkommt
    STORE !(suchen) TO suchen
    FIND &suchen
    COUNT WHILE !(autor) = suchen TO wieviele

    *** Nächste Aktion hängt von der Zahl der gefundenen
    *** Autoren-Namen ab
    DO CASE
          *** Wenn kein Name eingegeben, Bildschirm löschen
       CASE suchen = ' '
          ERASE

          *** Wenn der Name nicht vorhanden ist, Benutzer warnen
       CASE wieviele = 0
          ? 'Es gibt keinen &suchen in dieser Datenbank'
          ?
          ? 'Zum Weitersuchen beliebige Taste drücken ...'
          WAIT

          *** Wenn der Name nur einmal vorkommt, löschen
       CASE wieviele=1
          FIND &suchen
          ? 'Gelöscht wird -->' ,TRIM(autor),TRIM(titel)
          DELETE

          *** Wenn der Name mehrmals vorkommt, alle anzeigen
          *** und weitere Informationen einholen
       CASE wieviele > 1
          FIND &suchen
          LIST WHILE !(autor)=suchen;
          TRIM(autor),TRIM(titel)
          ?
          ACCEPT 'Welchen löschen (Nr. eingeben)' TO nummer
          IF val(nummer)>0
             GOTO &nummer
             DELETE
          ENDIF (nummer > 0)
    ENDCASE

ENDDO (suchen # ' ' )
```

Abb. 20.11: Das Programm LIBDEL.FMT (Forts.)

```
*** Vor dem endgültigen Löschen der Daten und der Rückkehr
*** zum Hauptmenü Löscherlaubnis einholen
COUNT FOR '*' TO nr:loesch
STORE 'N' TO packem
DO WHILE !(packem)  #'J' .AND. nr:loesch >0
   ERASE
   DISPLAY ALL FOR * TRIM(autor),TRIM(titel)
   ?
   ACCEPT 'Alle löschen? (J/N) ' ;
   TO packem

   *** Wenn nicht alle gelöscht werden sollen,
   *** herausfinden, wer nicht
   IF !(packem) # 'J'
      ?
      ACCEPT 'Welchen zurückholen? (Nr. eingeben) ' ;
               TO sichern
      IF val(sichern)>0
         RECALL RECORD &sichern
         STORE nr:loesch-1 TO nr:loesch
      ENDIF (sichern>0)
    ENDIF (packem # 'J')
   ENDDO (WHILE packem # j)

   *** Wenn Namen vorhanden und PACKen gewünscht, ausführen
   ERASE
   IF nr:loesch > 0
      ? 'Die angegebenen Einträge werden jetzt gelöscht ...'
      PACK
   ENDIF (nr:loesch >0)

   *** Memory-Variablen löschen und zum Hauptmenü zurück
   RELEASE ALL except wahl
   ?
   ? 'Bin auf dem Weg zum Hauptmenü ...'
   RETURN
```

Abb. 20.11: Das Programm LIBDEL.FMT (Forts.)

Zusammenfassung

In diesem Kapitel haben wir ein System erstellt, mit dem man Einträge zu einem Buch oder einer Zeitschrift eingeben und wieder ausgeben lassen kann. Die Einträge besitzen bis zu sieben Stichwörter, nach denen gezielt gesucht werden kann. Zusätzlich können Sie bei jedem Eintrag eine Zusammenfassung mit maximal 254 Zeichen Länge eingeben. Die Daten werden in der Datenbank LIBRARY.DBF gespeichert.

Eine zweite Datenbank mit dem Namen TOPICS.DBF wird vom System benutzt, um eine alphabetische Liste der Stichwörter in der Datenbank und die Häufigkeit jedes Stichwortes anzuzeigen.

In diesem Kapitel haben wir folgende Programm- und Menü-Dateien für unser Literatur-Verwaltungs-System erstellt:

1. LIBRARY.CMD, das Hauptmenü;
2. LIBADD.FMT, das Menü zum Eingeben der Einträge in die Datenbank;
3. LIBEDIT.CMD, ermöglicht Änderungen in der Datenbank;
4. LIBEDIT.FMT, das Menü zum Ändern von Datensätzen;
5. LIBDEL.CMD, zum Löschen von Einträgen.

Literaturverwaltungs-Berichte

21

Die Literaturverwaltung bietet zwei unterschiedliche Berichte an. Der erste ist eine Bilbiographie mit oder ohne Suchwörter und Zusammenfassungen. Anders als die bisherigen Berichte in diesem Buch wird bei der Bibliographie jede Zeile des Berichts fortlaufend gedruckt statt in Spaltenblöcken. Die Abb. 21.1 zeigt einen solchen Ausdruck mit Suchwörtern und der Zusammenfassung für jede Referenz.

Eine Bibliographie ohne die Suchwörter und Zusammenfassung sehen Sie in der Abb. 21.2. Beide Berichte werden mit der „Suchen nach Referenzen"-Option des Hauptmenüs erstellt. Der Benutzer kann beliebige Suchkriterien eingeben und auch festlegen, ob die Daten in alphabetischer Reihenfolge oder chronologisch geordnet dargeboten werden sollen. In beiden möglichen Berichten erscheint das Datum ausgeschrieben (z.B. 10. Januar 1984), obwohl es im Standardformat TT/MM/JJ gespeichert wurde.

Das Programmsystem kann auch eine alphabetisierte Liste aller vorhandenen Stichwörter drucken. Die Anzahl der Beiträge, die zu jedem Stichwort gespeichert sind, wird ebenfalls in dem Bericht aufgeführt. Die Abb. 21.3 zeigt einen solchen Ausdruck der Stichwortliste.

Nach Referenzen suchen

Das Suchprogramm der Literaturverwaltung ist ein sehr nützliches Werkzeug. Es ermöglicht dem Benutzer ganz leicht komplizierte Suchvorgänge, die sonst schwierig zu formulieren wären.

Bibliographie

Levine, Joel H., Hard Disks for Portables, Portable Computer, Vol. 2, No. 3, 38-41, 10. März 1984.
STICHW: Hard Disks, Winchesters, Portable Computers.
TEXT : Beschreibt, wie mit Hard Disks bei transportablen Computern umzugehen ist. Einschließlich Temperatur-, Sturz- und Feuchtigkeitsproblemen.

Trost, S. und King, R., Doing Business with Multiplan, SYBEX, Berkeley, CA, 1984.
STICHW: Multiplan, Budget, Verkauf.
TEXT : Einführung in den Gebrauch von Multiplan für Geschäftszwecke (englisch).

Simpson, Alan, Data File Programming On Your IBM PC, SYBEX, Berkeley, CA, Mai 1984.
STICHW: BASIC, IBM PC, Data File Programming, Sortieren, Suchen, Datenbanken.
TEXT : Das Buch bespricht fortgeschrittene BASIC-Programmiertechniken. Beiträge über sequentielles und wahlfreies Suchen in BASIC, außerdem Sortiermethoden.

Abb. 21.1: Bibliographie mit Stichwörtern und Zusammenfassungen

Der Benutzer kann seine Suchkriterien in die Maske so eintippen, wie in der Abb. 21.4 gezeigt.

Angenommen, der Benutzer möchte alles sehen, was P. Schmidt zu dem Stichwort dBASE II geschrieben hat. Dazu muß er nur die Maske so ausfüllen, wie in der Abb. 21.5 gezeigt.

Beachten Sie, daß das Autorenfeld und ein Stichwortfeld ausgefüllt sind. Weiterhin wird bei der Suchlogik verlangt, daß beide Begriffe zutreffen müssen: „ALLE Kriterien :J:". Das heißt, es werden nur Referenzen gefunden, die für P. Schmidt *und* dBASE II zutreffen. Die Optionen „Stichwörter zeigen" und „Zusammenfassung" sind ebenfalls mit einem J markiert. Das bedeutet, daß beides für jede Referenz im Bericht gezeigt wird. Da auch das Feld „zum Drucker" ein J enthält, wird der Bericht auch ausgedruckt.

Lassen Sie uns jetzt einmal annehmen, daß der Benutzer Artikel sucht, die sich mit Datenverwaltung, dBASE II, Datenbank-Design und Programmiertechniken befassen. Er möchte weder die Zusammenfassungen

noch die Stichwörter anzeigen lassen, aber der Bericht soll sowohl in eine Datei geschrieben als auch ausgedruckt werden. In diesem Fall wird er die Maske wie in Abb. 21.6 gezeigt ausfüllen.

Bei der Suche nach Referenzen in der Abb. 21.6 sehen Sie, daß hinter „Best. Kriterien" ein J eingegeben wurde. Das stellt sicher, daß jede

Bibliographie

Levine, Joel H., Hard Disks for Portables, Portable Computer, Vol. 2, No. 3, 38-41, 10. März 1984.

Trost, S. und King, R., Doing Business with Multiplan, SYBEX, Berkeley, CA, 1984.

Simpson, Alan, Data File Programming On Your IBM PC, SYBEX, Berkeley, CA, Mai 1984.

Glass, Jim, Word Processor File Conversion, PC Tech Journal, Vol.1, NO. 2, 168-179, September 1984.

Simpson, Alan, Arbeiten mit dBASE II, SYBEX, Düsseldorf, 1984.

Zaks, Rodnay, CP/M Handbuch, SYBEX, Berkeley, CA, 1982.

Abb. 21.2: Bibliographie ohne Stichwörter und Zusammenfassungen

Stichwortliste

Subject	Nr.	Subject	Nr.
ASSEMBLER	11	BASIC	22
C	9	CP/M	42
dBASE II	21	Datenbanken	13
Hard Disks	12	IBM PC	22
Input/Output	7	Kaypro	19
Sortieren	51	Suchen	44
Winchesters	11	Textverarbeitung	12
WordStar	21		

Abb. 21.3: Stichwortliste

Referenz, die auch nur einen der Suchbegriffe enthält, angezeigt wird. Hätte der Benutzer statt dessen hinter „Alle Kriterien" ein J eingegeben, wären nur Referenzen angezeigt worden, die alle vier Bedingungen gleichzeitig erfüllen. Mit anderen Worten: wenn hinter „Alle Kriterien" ein J steht, wird mit dem logischen Operator AND gesucht, wenn hinter „Best. Kriterien" ein J steht, wird mit dem Operator OR gesucht. Verständlicherweise finden Sie mit der letzten Methode eher Einträge, weil die Suchbasis breiter ist.

Das Programm LIBSEARC.CMD führt die Suche durch und erstellt die Bibliographie, wie sie bereits in Abb. 21.1 zu sehen war. Es ist aus zwei Gründen ein recht komplexes Programm. Erstens muß es Suchbedingungen aus den Benutzereingaben in die Maske erstellen. Zweitens muß es dafür sorgen, daß eine lange Zeichenkette am rechten Rand niemals mitten im Wort getrennt wird.

Sie können die Suchmaske wieder mit dem ZIP-Programm erstellen. Die

```
                    >>>S u c h e n <<<
-------------------------------------------------------------
Cursor-Befehle: ^E hoch : ^X runter: ^D rechts : ^S links : ^V einfügen
-> Geben Sie bestimmte Kriterien ein (ohne Eingabe wird nach allen gesucht)
-------------------------------------------------------------

Autor :                                                :

Titel :                                                :

Verlag (Publ.) :                                       :

Datum :   /   /    :

Stichwörter:
1. :                          :    2. :                          :
3. :                          :    4. :                          :
5. :                          :    6. :                          :
-------------------------------------------------------------
Such-Logik (J/N):   Alle Kriterien :  :  Best. Kriterien     :  :
Format:          Stichwörter zeigen :  :  Zusammenfassung   :  :
Ausgabe:               zum Drucker :  :  in eine Datei      :  :
```

Abb. 21.4: Eine leere Suchmaske

Abb. 21.7 zeigt Ihnen, wie die Formatdatei LIBSEARC.FMT auf dem ZIP-Bildschirm aussieht.

Wenn Sie fertig sind, erstellt Ihnen ZIP wieder das entsprechende dBASE-Programm, so, wie Sie es in der Abb. 21.8 sehen.

In der Abbildung 21.9 sehen Sie den Pseudocode für die LIBSEARC. CMD-Programmdatei.

Wir werden jetzt mit der Entwicklung des LIBSEARC.CMD-Programms beginnen. Zuerst müssen wir die benötigten Variablen „ins Leben rufen":

```
********************************** LIBSEARC.CMD
************************** Einträge suchen
SET TALK OFF

********************* Memory-Variablen vorbereiten
STORE '          ' TO m:autor,m:verlag
```

```
                    >>>S u c h e n <<<

Cursor-Befehle: ^E hoch : ^X runter: ^D rechts : ^S links : ^V ein-
fügen
-> Geben Sie bestimmte Kriterien ein (ohne Eingabe wird nach
allen gesucht)

Autor :Schmidt,P                                    :

Titel :                                             :

Verlag (Publ.) :                                    :

Datum :   /   /    :

Stichwörter:
1. :dBASE II                :    2. :                    :
3. :                        :    4. :                    :
5. :                        :    6. :                    :

Such-Logik (J/N):    Alle Kriterien :J:  Best. Kriterien    : :
Format:              Stichwörter zeigen :J:  Zusammenfassung  :J:
Ausgabe:             zum Drucker :J:  in eine Datei         : :
```

Abb. 21.5: Eine Suche nach Beiträgen über dBASE II von P. Schmidt

```
STORE '
                    ' TO m:titel
STORE '          ' TO m:datum
STORE '             ' TO m:swort1,m:swort2,m:swort3
STORE '             ' TO m:swort4,m:swort5,m:swort6
STORE ' ' TO any,alle,zeig:swrts,zeig:zus,drucker,;
                                                  to:text
STORE '                ' TO text:datei
```

Alle Variablen, die mit einem „m" beginnen, sind die Gegenstücke zu den gleichlautenden Feldnamen in der Datenbank. Zum Beispiel ist M:AUTOR der zu suchende Autor, M:SWORT1 ist ein Stichwort, nach dem zu suchen ist und so weiter. Die Variablen ALLE, ANY, ZEIG:SWORTS, ZEIG:ZUS, DRUCKER und TO:TEXT werden alle als Merker benutzt, die anzeigen, ob bei der Suche nach ALLEn oder bestimmten Kriterien (ANY) gesucht werden soll, ob der Bericht die

```
                    >>>S u c h e n<<<

Cursor-Befehle: ^E hoch : ^X runter: ^D rechts : ^S links : ^V ein-
fügen
-> Geben Sie bestimmte Kriterien ein (ohne Eingabe wird nach
allen gesucht)

Autor :                                              :

Titel :                                              :

Verlag (Publ.) :                                     :

Datum :   /   /   :

Stichwörter:
1. :Datenverwaltung          :    2. :dBASE II              :
3. :      Datenbank-Design   :    4. :Programmiertechniken  :
5. :                         :    6. :                      :

Such-Logik (J/N):  Alle Kriterien :  :  Best. Kriterien     :J:
Format:            Stichwörter zeigen :J:  Zusammenfassung  :J:
Ausgabe:           zum Drucker :J:  in eine Datei           :J:
```

Abb. 21.6: Eine Suche nach Referenzen mit vier Stichwörtern

```
                    >>> S u c h e n <<<
-------------------------------------------------------------------
Cursor-Befehle: ^E hoch : ^X runter: ^D rechts : ^S links : ^V ein-
fügen
-> Geben Sie bestimmte Kriterien ein (ohne Eingabe wird nach
allen gesucht)
-------------------------------------------------------------------
Autor #m:autor

Titel #m:titel

Verlag (Publ.) #verlag

Datum #m:datum

Stichwörter:
1. #m:swort1              2. #m:swort2
3. #m:swort3              4. #m:swort4
5. #m:swort5              6. #m:swort6
-------------------------------------------------------------------
Such-Logik (J/N): Alle Kriterien#alle      Best. Kriterien #any
Format: Stichwörter zeigen #zeig:sworts Zusammenfassung #zeig:zus
Ausgabe:      zum Drucker #drucker      in eine Datei #to:text
```

Abb. 21.7: Die Suchmaske auf dem ZIP-Bildschirm

```
* LIBSEARC.FMT
§  1,22 SAY ">>>  S u c h e n  <<<"
§  2, 1 SAY "=========================================
             ================================="
§  3, 1 SAY "Cursor-Befehle: ^E hoch : ^X runter: ^D rechts :
            ^S links : ^V einfügen"
 §  4, 2 SAY "--> Geben Sie bestimmte Kriterien ein (ohne Eingabe
            wird nach allen gesucht)"
§  5, 1 SAY "=========================================
             ================================="
§  6, 2 SAY "Autor"
§  6, 9 GET m:autor
§  8, 2 SAY "Titel"
§  8, 9 GET m:titel
§ 10, 2 SAY "Verlag (Publ.)"
§ 10,17 GET verlag
```

Abb. 21.8: Die Formatdatei LIBSEARC.FMT

```
§ 12, 2 SAY "Datum"
§ 12, 9 GET m:datum
§ 14, 2 SAY "Stichwörter:"
§ 15, 1 SAY "1."
§ 15, 5 GET m:swort1
§ 15,37 SAY "2."
§ 15,41 GET m:swort2
§ 16, 1 SAY "3."
§ 16, 5 GET m:swort3
§ 16,37 SAY "4."
§ 16,41 GET m:swort4
§ 17, 1 SAY "5."
§ 17, 5 GET m:swort5
§ 17,37 SAY "6."
§ 17,41 GET m:swort6
§ 18, 0 SAY "=========================================
            ================================
§ 19, 1 SAY "Such-Logik (J/N): Alle Kriterien"
§ 19,33 GET alle
§ 19,47 SAY "Best. Kriterien"
§ 19,63 GET any
§ 20, 1 SAY "Format:     Stichwörter zeigen"
§ 20,33 GET zeig:swort
§ 20,44 SAY "s  Zusammenfassung"
§ 20,63 GET zeig:zus
§ 21, 1 SAY "Ausgabe:           zum Drucker"
§ 21,33 GET drucker
§ 21,47 SAY "in eine Datei"
§ 21,63 GET to:text
```

Abb. 21.8: Die Formatdatei LIBSEARC.FMT (Forts.)

Stichwörter (ZEIG:SWORTS) oder die Zusammenfassung (ZEIG:ZUS) zeigen soll und ob der Bericht zum Drucker (DRUCKER) oder in eine Datei (TO:TEXT) geht.

Wenn der Benutzer die gefundenen Daten in einer Datei speichern möchte, wird der Name für diese Datei in der Variablen TEXT:DATEI gespeichert.

Als nächstes muß das Programm die Monatsnamen erstellen, um die numerischen Werte des Datums in Wörter übersetzen zu können. Beachten Sie, daß die letzten beiden Zeichen jeder Variablen dem Monat entsprechen, den sie repräsentieren. Januar ist also MO01, und Dezember heißt MO12. Das erleichtert später die Übersetzung:

```
**************************** Monatsnamen speichern
STORE ' ' TO mo00
```

```
Initialisiere die Variablen

Speichere Monatsnamen zur Übersetzung des Datums in Wörter

Hole Suchkriterien vom Benutzer, benutze LIBSEARC.FMT

Wenn der Bericht in eine Datei geht, hole Name für Datei

Frage Benutzer nach gewünschter Sortierung

Benutze entsprechende Index-Datei, je nach Benutzereingabe

Erstelle Makro für die Suche mit AND oder OR

Erstelle eine Suchkette für AUTOR, TITEL, VERLAG und
DATUM

Füge Stichwörter an Suchbedingung an

Wenn nötig, lösche das letzte AND oder OR aus dem Suchmakro

Wenn keine Suchbedingung eingegeben, mache sie immer wahr (true)

Wenn Drucker gewünscht, schalte Drucker ein

Wenn Datei gewünscht, eröffne Datei

Drucke Titel und beginne Zeilenzähler

Beginne mit der Suche am Anfang der Datenbank

Stelle Stichwörter für Suche und Druck zusammen

Prüfe auf Übereinstimmung

Wenn Datensatz den Suchbedingungen entspricht

    Erstelle Bibliographie-Zeile
    Übersetze das Datum in Wörter
    Drucke die Zeile richtig formatiert
```

Abb. 21.9: Pseudocode für das Programm LIBSEARC.CMD

```
Wenn Benutzer Stichwörter sehen will
   Beseitige zuerst das führende Komma, füge dann Titel und
   Schlußpunkt hinzu
   Zeige Stichwörter ordentlich formatiert

Wenn Benutzer Zusammenfassung sehen möchte
   Füge Titel der Zusammenfassung hinzu
   Zeige alles ordentlich formatiert

Drucke eine Leerzeile zwischen jeder Referenz
   Erhöhe Zeilenzähler
   Zähle gedruckte Zeilen und mache Blattvorschub oder halte
   Bildschirm an, wenn notwendig
   Weiter zum nächsten Datensatz in der Datenbank

Weiter mit der Schleife (bis Ende der Datenbank)

Schließe und beende Textdatei, wenn notwendig

Halte Bildschirm an, wenn nicht gedruckt wird

Lösche alle Variablen, bis auf WAHL

Zurück zum Hauptmenü
```

Abb. 21.9: Pseudocode für das Programm LIBSEARC.CMD (Forts.)

```
STORE 'Januar' TO mo01
STORE 'Februar' TO mo02
STORE 'März' TO mo03
STORE 'April' TO mo04
STORE 'Mai' TO mo05
STORE 'Juni' TO mo06
STORE 'Juli' TO mo07
STORE 'August' TO mo08
STORE 'September' TO mo09
STORE 'Oktober' TO mo10
STORE 'November' TO mo11
STORE 'Dezember' TO mo12
```

Dann wollen wir Zeilen dazuschreiben, um die Suchmaske mit den Eingabemöglichkeiten für die Suche und den Bericht zu zeigen. Das wird von

der Formatdatei LIBSEARC.FMT mit diesen Zeilen durchgeführt:

```
***************** Suchkriterien vom Benutzer holen
SET FORMAT TO libsearc
READ
SET FORMAT TO screen
```

Wünscht der Benutzer die Ausgabe in eine Textdatei, muß das Programm nach einem Dateinamen fragen, wie Sie hier sehen:

```
***************** Wenn Ausgabe in Datei gewünscht,
***************** Dateinamen verlangen
IF !(to:text)= 'J'
   ERASE
   § 5,5 SAY 'In welche Datei sollen die Daten? ' ;
         GET text:datei
   READ
ENDIF (to:text=j)
```

Jetzt kann das Programm fragen, in welcher Sortierung die Daten angezeigt werden sollen:

```
************************ Wie soll sortiert werden
ERASE
STORE ' ' TO sort:m
TEXT
               Sortiermöglichkeiten

          1.   Alphabetisch nach Autoren
          2.   Nach Datum
          3.   Original-Reihenfolge
ENDTEXT
§ 8,10 SAY 'Gewünschte Sortierung eingeben' GET;
                                                sort:m
READ
```

Das Programm wählt die entsprechende Index-Datei oder benutzt die Datenbank ohne Index:

```
*************** Entsprechende Index-Datei benutzen
DO CASE
   CASE sort:m = '1'
      USE library INDEX autor

   CASE sort:m = '2'
```

```
      USE library INDEX datum

   OTHERWISE
      USE library

ENDCASE
```

Wenn der Benutzer angibt, daß die Übereinstimmung mit einem bestimmten Suchkriterium (ANY) für die Anzeige einer Referenz ausreicht, wird daraufhin mit der Bedingung OR gesucht. Andernfalls wird bei der Suche der logische Operator AND verwendet. Die folgenden Zeilen bereiten die Suche vor, indem sie den richtigen Operator in die Variable LOGIC speichern:

```
********** Macro für die Suche mit .AND. und .OR.
********** definieren
IF !(any) = 'J'
   STORE '.OR.' TO logic
ELSE
   STORE '.AND.' TO logic
ENDIF (any # ' ' )
```

Der nächste Hauptschritt in diesem Programm ist, eine Zeichenfolge zu entwickeln, die für die Suche als Makro verwendet werden kann. Wenn der Benutzer beispielsweise einen Beitrag von P. Schmidt mit dem Titel „Willkommen zu Hause" finden möchte, würde der Such-String so aussehen:

```
m:autor $!(autor) .AND. m:titel $!(titel)
```

so daß er mit einer IF-Bedingung zusammen, wie hier:

```
IF m:autor $!(autor) .AND. m:titel $!(titel)
```

nur die Referenzen anzeigt, die SCHMIDT, P in Großbuchstaben im Autoren-Feld und WILLKOMMEN ZU HAUSE in Großbuchstaben im Titel-Feld stehen haben.

Da der Benutzer so viele Möglichkeiten bei der Definition der Suchbedingungen hat, braucht das Programm mehrere Schritte, um den Suchstring zu formen. Die Variable BED enthält die Suchbedingungen. Sie wird erst einmal als leere Variable indiziert:

```
********** Such-String für Autor,Titel,Verlag und
********** Datum erstellen
STORE ' ' TO bed
```

Will der Benutzer nach dem Autor suchen, übersetzt das Programm den Namen in der Variablen M:AUTOR in Großbuchstaben und beseitigt die führenden Leerzeichen. Dann schreibt es den Ausdruck M:AUTOR $!(AUTOR) in die Variable BED und dazu, je nach Suchlogik, ein AND oder OR:

```
**************************** Nach dem Autor suchen
IF m:autor # ' '
   STORE !(TRIM(m:autor)) TO m:autor
   STORE 'm:autor $ ! (autor)' +logic TO bed
ENDIF (m:autor # ' ' )
```

Sucht der Benutzer nach einem Titel, passiert in etwa das gleiche mit dem Feld M:TITEL. Der Ausdruck M:TITEL $!(TITEL) wird in die BED-Variable gespeichert und wieder ein AND oder ein OR dazu:

```
************************* Nach dem Titel suchen
IF m:titel # ' '
   STORE !(TRIM(m:titel)) TO m:titel
   STORE bed+ 'm:titel $!(titel)' +logic TO bed
ENDIF (m:titel # ' ' )
```

Dieselbe Prozedur wird für die Felder VERLAG und DATUM verwendet, außer daß es hierbei nicht nötig ist, den Inhalt in Großbuchstaben zu übersetzen und die Leerzeichen des Datums zu löschen:

```
**************************** Nach dem Verlag suchen
IF m:verlag # ' '
   STORE !(TRIM(m:verlag)) TO m:verlag
   STORE bed+ 'm:verlag $!(verlag)' +logic TO bed
ENDIF (m:verlag # ' ' )
**************************** Nach dem Datum suchen
IF m:datum # ' '
   STORE bed+ 'm:datum = datum' + logic TO bed
ENDIF (m:datum # ' ' )
```

Zu diesem Zeitpunkt enthält die Variable BED die notwendige dBASE-Syntax, um nach jedem Feld, mit Ausnahme der Stichwörter, zu suchen. Da wir sechs Stichwörter haben, die alle eine Ziffer zum Variablennamen besitzen (M.SWORT1, M:SWORT2, M:SWORT3 ...), können wir alle sechs mit einer Schleife bearbeiten. Wir beginnen die Schleife bei 1 und machen weiter, bis der Zähler bei sechs ist:

```
**** Aus Suchwörtern und Suchbedingungen einen
**** Such-String machen
STORE 1 TO zaehler
DO WHILE zaehler < 6
```

Dann speichern wir mit Hilfe der STRING-Funktion den Inhalt des ZÄHLERs in eine Variable mit Namen SUB, die wir dann an die Stichwortvariablennamen als Makro anhängen können. Das bedeutet: Beim ersten Schleifedurchgang bekommen wir M:SWORT&SUB (M:SWORT1). Der zweite Schleifedurchgang bringt uns dann M:SWORT&SUB, was M:SWORT2 bedeutet. So geht es weiter durch alle sechs Schleifendurchgänge:

```
STORE STR(zaehler,1) TO sub
```

Jede der sechs M:SWORT-Variablen, außer leer gebliebenen, übergibt ihre Suchbedingung in den BED-String, und zwar in dem Format M:SWORT1 $!(SCHLUESSEL). Um die Suche zu beschleunigen, werden alle sechs Stichwörter der Datenbank in eine lange Zeichenkette, SCHLUESSEL genannt, in diesem Format gespeichert:

```
Datenverwaltung,dBASE II,Datenbank-Design,Program-
miertechniken
```

Um ein Höchstmaß an Genauigkeit zu bekommen, erhält das M:SWORT, nach dem gesucht wird, eine Leerstelle vorn und ein Komma hinten, wie hier:

```
" Datenbanken,"
```

Auf diese Weise beachtet die Substring-Funktion ($), wenn sie in SCHLUESSEL nach einem Stichwort sucht, keine Wortteile innerhalb der SCHLUESSEL-Liste.

Die folgende Routine stellt sicher, daß das M:SWORT Stichwort nicht leer ist. Wenn das zutrifft, wird der entsprechende Ausdruck in die Variable BED gespeichert und der richtige Operator, AND oder OR, dazu:

```
IF m:swort&sub # ' '
   STORE ' ' +!(TRIM(m:swort&sub))+ ',' TO m:swort-
&sub
   STORE bed+ 'm:swort&sub $!(schluessel)' +logic;
   TO bed
ENDIF (swort&sub # ' ' )
```

Die ZAEHLER-Variable wird dann um 1 erhöht und die Schleife durch die sechs M:SWORT-Variablen geht weiter:

```
 STORE zaehler+1 TO zaehler
ENDDO (zaehler < 6)
```

An dieser Stelle sieht der BED-String, falls der Benutzer nach allen Refe-

renzen für das Stichwort SCHMIDT im Zusammenhang mit dBASE II verlangt hat, so aus:

```
M:AUTOR $!(AUTOR) .AND. M:SWORT1 $!SCHLUESSEL) .AND.
```

M:AUTOR und M:SWORT1 sind schon von Leerstellen befreit und für eine maximale Suchgenauigkeit in Großbuchstaben übersetzt. Natürlich kann BED auch viel länger und noch komplizierter werden, wenn der Benutzer noch mehr Suchkriterien in die Maske auf dem Bildschirm eingibt.

Es gibt nur einen kleinen Fehler in der oben gezeigten Variablen BED: Das letzte .AND. sollte beseitigt werden. Das können wir erreichen, indem wir alles, außer den letzten fünf Zeichen der Variablen BED , in die Variable BED zurückspeichern. Natürlich müssen nur die letzten vier Zeichen beseitigt werden, wenn der Benutzer eine „.OR."-Suche macht. Die nächste Routine übernimmt diese Aufgabe:

```
********** Das letzte .AND. oder .OR. beseitigen
IF !(any) = 'J'
   STORE $(bed,1,len(bed)-4) TO bed
ELSE
   STORE $(bed,1,len(bed)-5) TO bed
ENDIF (any # ' ' )
```

Der Benutzer hat auch die Möglichkeit, alle Suchkriterienfelder in der Maske einfach leer zu lassen und damit alle Referenzen der Datenbank aufzulisten. Wenn dieser Fall eintritt, bleibt die Variable BED einfach leer. Wir müssen deshalb ein Suchkriterium in das Programm einbauen, das immer „wahr" ist, etwa in der Art von 1=1, um diese Situation zu meistern. Schauen Sie sich dazu die nächste Routine an:

```
********** Wenn keine Suchbedingungen eingegeben,
********** ist es immer richtig (T)rue
IF len(bed) = 1
   STORE '1=1' TO bed
ENDIF (bed = ' ' )
```

So, jetzt ist die Suchbedingung in BED komplett. Das Programm kann nun den Drucker einschalten, wenn der Benutzer so gewählt hat:

```
*************** Wenn Drucker angewählt, einschalten
IF !(drucker)= 'J'
   SET PRINT ON
ENDIF
```

Hat der Benutzer den Wunsch, die auszugebenden Daten in eine Textdatei zu schreiben, muß das Programm diese Datei eröffnen (SET ALTERNATE TO) und den gewünschten Dateinamen dabei verwenden:

```
******************** Wenn Datei gewählt, erstellen
IF !(to:text)= 'J'
   SET ALTERNATE TO &text:datei
   SET ALTERNATE ON
ENDIF (to:text=j)
```

Jetzt kann das Programm den Bildschirm löschen, die Berichtüberschrift drucken, zum ersten Datensatz der Datenbank gehen, eine Variable zum Zählen der gedruckten Zeilen starten und eine Schleife bis zum Ende der Datei beginnen:

```
****************** Titel drucken und Zeilen zählen
ERASE
? '                          Bibliographie'
?
STORE 2 TO lf
******************* Jetzt mit der Suche beginnen und
******************* die Ausgabe formatieren
GO TOP
DO WHILE .NOT. EOF
```

Für jeden Datensatz der Datenbank müssen zuerst alle Stichwörter (SWORT1–SWORT6) in eine einzelne Variable, SCHLUESSEL genannt, gespeichert werden. Das Programm stellt automatisch hinter jedes Stichwort ein Komma, um damit den Ausdruck besser aussehen zu lassen und, wie oben schon besprochen, um genauer suchen zu können. Noch einmal benutzen wir eine Schleife (von 1–6), damit jedes Stichwort einmal an die Reihe kommt:

```
*** Schlüsselwörter zum Suchen und Drucken
***hintereinanderschreiben
STORE 1 TO zaehler
STORE ' ' TO schluessel
DO WHILE zaehler <= 6
   STORE STR(zaehler,1) TO sub
   IF swort&sub # ' '
      STORE schluessel+TRIM(swort&sub)+ ', ' TO;
                                          schluessel
   ENDIF (swort&sub # ' ' )
   STORE zaehler+1 TO zaehler
ENDDO (zaehler <= 6 )
```

Da die gesamte BED-Zeichenkette schon fertig zusammengestellt ist, kann das Programm ganz einfach überprüfen, ob die Daten des aktuellen Datensatzes mit den Suchkriterien übereinstimmen, indem es die BED-Variable in einer IF-Anweisung als Makro benutzt:

```
************************ Etwas gefunden ?
IF &bed
```

Und nun geht der Spaß los. Sollen die Daten des aktuellen Datensatzes gedruckt werden, müssen zuerst die Felder AUTOR, TITEL und VERLAG in eine lange Zeichenkette (String) mit Kommas und Leerzeichen dazwischen gespeichert werden. Das alles wird in der Variablen LANG abgelegt:

```
********************** Dann für Ausgabe formatieren
STORE TRIM(autor)+ ', ' +TRIM(titel)+ ', ' +;
   TRIM(verlag)+ ', ' TO lang
```

Wenn eine Ausgabenummer (AUSGABE), eine Fundstelle (ORT) oder eine Seitenzahl (SEITEN) in diese Referenz mitaufgenommen werden sollen, sollten diese auch in die Variable LANG gespeichert werden:

```
IF ausgabe # ' '
   STORE lang+TRIM(ausgabe)+ ', ' TO lang
ENDIF ausgabe # ' '
IF ort # ' '
   STORE lang+TRIM(ort) + ', ' TO lang
ENDIF ort # ' '
IF seiten # ' '
   STORE lang + TRIM(seiten)+ ', ' TO lang
ENDIF seiten # ' '
```

Natürlich gehört auch das Datum dazu. Da aber das Datum in Bibliographien normalerweise nicht im Format 10/10/84 verwendet wird, müssen wir es in Wörter übersetzen. Wir schaffen das, indem wir die ersten beiden Zeichen des Datums (TT) in die Variable TAG, die nächsten beiden (MM) in die Variable MONAT und die letzten beiden (JJ) in JAHR speichern. Zuerst teilt das Programm das Datum in die drei Variablen TAG, MONAT, JAHR auf:

```
******************************** Datum umformen
STORE $(datum,1,2) TO tag
STORE $(datum,4,2) TO monat
STORE $(datum,7,2) TO jahr
```

Wir haben bereits vorher die verschiedenen Monate eines Jahres in Variablen gespeichert, wie zum Beispiel MO01 (Januar) und MO06 (Juni), so daß ein einfaches Makro das Datum in Wörter übersetzen kann (z. B. MO&MONAT). Wir speichern das in eine Variable mit dem Namen E:DATUM:

```
STORE mo&monat TO e:datum
```

Zu diesem Zeitpunkt enthält E:DATUM den Monatsnamen in ausgeschriebener Form (z. B. März).

Wenn das Datum eine Angabe zum Tag besitzt (es also nicht so aussieht: 00/03/84), speichern wir den Tag mit einem Punkt und einem Leerzeichen dahinter:

```
IF val(tag)>0
   STORE tag + '. ' + e:datum TO e:datum
ENDIF val(tag)>0
```

Damit sieht das Datum jetzt so aus: 31. März. Wurde keine Angabe zum Tag des Monats (März) gemacht, steht nur der Monatsname in der Variablen. Für die Jahreszahl hängen wir einfach eine 19 plus die letzten beiden Ziffern (JAHR) des Datums an:

```
STORE e:datum+ ' 19' +jahr TO e:datum
```

Nun enthält E:DATUM das Datum in der Form 31. März 1985.

Nachdem das Datum jetzt übersetzt ist, können wir E:DATUM an die LANG-Variable anhängen:

```
STORE lang + e:datum+ '.' TO lang
```

So weit, so gut. Lassen Sie uns jetzt die Spalte 76 als den rechten Rand unseres Berichts festlegen. Ist die Variable LANG kürzer als 76 Zeichen oder gerade so lang, kann sie einfach ausgedruckt und der Zeilenzähler (LF) um 1 erhöht werden:

```
************************** Zeile drucken
STORE 76 TO rand
IF len(lang) <= rand
   ? lang
   STORE lf+1 TO lf
ELSE
```

Wenn LANG aber länger als 76 Zeichen ist, sollte sie an der Leerstelle geteilt werden, die dem rechten Rand am nächsten ist (solch eine Prozedur nennt man Wortumbruch). Um dieses Funktion zu bekommen,

beginnen wir eine Schleife, die von 76 ab rückwärts zählt und damit nicht aufhört, bis sie eine Leerstelle in dem String LANG findet:

```
DO WHILE #(lang,rand,1) # ' '
     STORE rand - 1 TO rand
ENDDO (zeichen ganz rechts kein leerzeichen)
```

Diese Schleife wird beendet, wenn sie das Leerzeichen entdeckt, das dem rechten Rand am nächsten steht. An dieser Stelle wird die Variable RAND der Position entsprechen, wo die Zeile getrennt werden muß. Deshalb können wir den ersten Teil von LANG bis zu RAND in einer Zeile drucken und den zweiten Teil von LANG ab der Position RAND (plus eins) in der zweiten Zeile. Der Zeilenzähler wird dann um 2 erhöht:

```
  ? $(lang,1,rand)
  ? $(lang,rand+1)
  STORE lf+2 TO lf
ENDIF (lang <= rand)
```

Wenn der Benutzer die Stichwörter nun ausgedruckt haben möchte, können wir dieselbe Formatierungsmöglichkeit mit der Variablen SCHLUESSEL benutzen. Aber zuerst müssen wir sie noch ein wenig bearbeiten. Die Variable SCHLUESSEL hat immer noch ein Komma am Ende stehen (wurde weiter oben für die Suche gebraucht), das wir löschen und durch einen Punkt ersetzen können. Und wir fügen noch die kurze Überschrift „STICHW:" am Anfang der Zeile hinzu, um dem Bericht ein klareres Erscheinungbild zu geben:

```
********** Wenn Stichwörter gewünscht, anzeigen
  IF !(zeig:swrts) = 'J'
     ************ Das führende Komma löschen und
     ************ einen Punkt am Ende anhängen
     STORE len(TRIM(schluessel)) TO length
     STORE $(schluessel,1,length-1) TO schluessel
     STORE 'STICHW:' +schluessel TO schluessel
     IF len(schluessel) > 10
        STORE schluessel+ '.' TO schluessel
     ENDIF len(schluessel) > 10
```

Dann können wir die gleiche Routine wie mit LANG benutzen, um den Wortumbruch am rechten Rand zu bewirken:

```
  STORE 76 TO rand
    IF len(schluessel) <= rand
       ? schluessel
```

```
      STORE lf+1 TO lf
   ELSE
      DO WHILE $(schluessel,rand,1) # ' '
         STORE rand - 1 TO rand
      ENDDO (zeichen ganz rechts nicht ' ' )
      ? $(schluessel,1,rand)
      ? '           ' +$(schluessel,rand+1)
      STORE lf+2 TO lf
   ENDIF (len(schluessel) <= rand)
ENDIF (!(zeig:swrts=j)
```

Wenn der Benutzer jetzt auch die Zusammenfassung ausdrucken möchte, schreibt das Programm die kurze Überschrift „TEXT:" dazu und speichert alles in der Variablen LANG ab:

```
*** Wenn Zusammenfassung gewählt, ausdrucken
IF !(zeig:zus) = 'J'
   STORE 'TEXT : ' +TRIM(zusammen) TO lang
```

Die Zusammenfassung zu formatieren ist schon ein wenig aufwendiger. Da die Maximallänge bei 254 Zeichen liegt (einschließlich der Überschrift TEXT:), kann sie im Bericht drei oder vier Zeilen beanspruchen. Und so geht es: Zuerst beginnen wir eine Schleife, die fortwährend die LANG-Zusammenfassung in kleinere Stücke zerteilt, die nicht länger als 76 Zeichen sind:

```
DO WHILE len(lang) > 76
```

Innerhalb der Schleife wird der Rand auf 76 Zeichen gesetzt, und eine weitere Schleife findet die Leerstelle, die dem rechten Rand (dem 76. Zeichen) am nächsten liegt:

```
STORE 76 TO rand
   DO WHILE $(lang,rand,1) # ' '
      STORE rand -1 TO rand
    ENDDO WHILE $(lang,rand,1) # ' '
```

Sobald die letzte Leerstelle vor dem rechten Rand gefunden wurde, werden die Zeichen innerhalb der Grenze gedruckt und der Zeilenzähler erhöht:

```
? $(lang,1,rand)
STORE lf+1 TO lf
```

Dann wird der übriggebliebene Teil von LANG in die Variable LANG gespeichert, und wenn der Rest immer noch länger als 76 Zeichen ist, geht

das Programm wieder durch die äußere Schleife und holt sich das nächste Stück von LANG. Das geht so weiter, bis in LANG nur noch ein Stück übrig ist, das weniger als 76 Zeichen enthält:

```
   STORE $(lang,rand+1) TO lang
ENDDO (len(lang>76))
```

Sobald ein Teilstück, das in die Zeile paßt, übrig ist, wird es auf dem Bildschirm ausgegeben, und der Zeilenzähler wird erhöht:

```
 ? lang
 STORE lf+2 TO lf
ENDIF (zeig:zus=j)
```

An diesem Punkt ist dann ein ganzer Eintrag gedruckt. Ob Sie es glauben oder nicht: Der Computer braucht nur etwa drei Sekunden, um die BED-Zeichenkette zusammenzustellen, und zwei oder drei Sekunden, um die ganze Formatierarbeit zu leisten, die wir gerade besprochen haben.

Damit die Texte auf dem Papier nicht zu dicht beieinander stehen, schieben wir eine Leerzeile ein, nachdem jeweils eine Referenz gedruckt wurde, und erhöhen wieder den Zeilenzähler:

```
************** Eine Leerzeile dazwischen drucken
?
STORE lf+1 TO lf
```

Will der Benutzer den Bericht ausdrucken und 55 Zeilen sind bereits gedruckt worden, soll das Programm eine Seite weitergehen, die Überschrift drucken und den Zeilenzähler zurücksetzen:

```
*** Gedruckte Zeilen zählen und das nächste Blatt
*** nehmen oder, wenn nötig, Pause machen
IF !(drucker)= 'J' .AND. lf >=55
   EJECT
   ? '                                Bibliographie'
   ?
   STORE 2 TO lf
ENDIF !(drucker)= 'J' .AND. lf <=55
```

Soll der Bericht nur auf dem Bildschirm erscheinen, soll das Programm jeweils nach 20 Zeilen anhalten und den Zeilenzähler zurücksetzen:

```
IF !(drucker) # 'J' .AND. !(to:text) # 'J' .AND.
                                                lf >=20
```

```
      ?
      ? 'Weiter mit beliebiger Taste ...'
      WAIT
      STORE 2 TO lf
      ERASE
   ENDIF (NOT drucker)
```

Selbstverständlich wird keine der Formatier- oder Druckroutinen gebraucht, wenn der Datensatz den Suchkriterien nicht entspricht. Wie auch immer, das Programm soll zum nächsten Datensatz weitergehen, bis das Ende der Datenbank erreicht ist:

```
      ENDIF (datensatz gefunden)
      SKIP
   ENDDO (WHILE .NOT. EOF)
   SET PRINT OFF
```

Sobald die DO WHILE .NOT. EOF-Schleife beendet ist, ist auch das Programm beendet. Wurde eine Textdatei eröffnet, sollte diese jetzt geschlossen werden:

```
   ******************** Wenn notwendig, Datei schließen
   IF !(to:text) = 'J'
      SET ALTERNATE OFF
      SET ALTERNATE TO
   ENDIF (to:text = j)
```

Wurden die Referenzen nur auf dem Bildschirm angezeigt, soll das Programm anhalten, bevor es den Bildschirm löscht und zum Hauptmenü zurückkehrt:

```
   ********** Wenn Daten nur auf dem Bildschirm
   ********** erscheinen und
   ********** nicht gedruckt werden, Pause einlegen
   IF !(drucker) # 'J'
      ?
      ? 'Weiter mit beliebiger Taste ...'
      WAIT
   ENDIF !(drucker) # 'J'
```

Jetzt kann das Programm alle Variablen (außer WAHL, die zum Hauptmenü gehört) löschen und zum Programm LIBRARY.CMD zurückkehren:

```
   ********* Alle Memory-Variablen außer WAHL löschen
   RELEASE ALL EXCEPT wahl
   RETURN
```

Das gesamte LIBSEARC.CMD-Programm finden Sie in der Abb. 21.10. Es ist zwar ein langes und komplexes Programm, aber es demonstriert die Leistungsfähigkeit der dBASE-Makros genauso wie die Mächtigkeit und Flexibilität der dBASE II-Programmiersprache. Für engagierte Programmierer ist der Zeitaufwand zum Studium gerade dieses Programms bestimmt keine Zeitverschwendung.

```
********************************** LIBSEARC.CMD
************************** Einträge suchen
SET TALK OFF

*** Memory-Variablen vorbereiten
STORE '                                ' TO m:autor,m:verlag
STORE '                                        ' TO m:titel
STORE '         ' TO m:datum
STORE '                           ' TO m:swort1,m:swort2,m:swort3
STORE '                           ' TO m:swort4,m:swort5,m:swort6
STORE ' ' TO any,alle,zeig:swrts,zeig:zus,drucker,to:text
STORE '               ' TO text:datei

*** Monatsnamen speichern
STORE ' ' TO mo00
STORE 'Januar' TO mo01
STORE 'Februar' TO mo02
STORE 'März' TO mo03
STORE 'April' TO mo04
STORE 'Mai' TO mo05
STORE 'Juni' TO mo06
STORE 'Juli' TO mo07
STORE 'August' TO mo08
STORE 'September' TO mo09
STORE 'Oktober' TO mo10
STORE 'November' TO mo11
STORE 'Dezember' TO mo12

*** Suchkriterien vom Benutzer holen
SET FORMAT TO libsearc
READ
SET FORMAT TO screen

*** Wenn Ausgabe in Datei gewünscht,
*** Dateinamen verlangen
IF !(to:text)= 'J'
   ERASE
   § 5,5 SAY 'In welche Datei sollen die Daten? ' ;
   GET text:datei
```

Abb. 21.10: Das Programm LIBSEARC.CMD

```
   READ
ENDIF (to:text=j)

*** Wie soll sortiert werden
ERASE
STORE ' ' TO sort:m
TEXT
     Sortier-Möglichkeiten
     1. Alphabetisch nach Autoren
     2. Nach Datum
     3. Original-Reihenfolge
ENDTEXT
§ 8,10 SAY 'Gewünschte Sortierung eingeben' GET sort:m

READ

*** Entsprechende Index-Datei benutzen
DO CASE
   CASE sort:m = '1'
      USE library INDEX autor

   CASE sort:m = '2'
      USE library INDEX datum

   OTHERWISE
      USE library

ENDCASE

*** Macro für die Suche mit .AND. und .OR definieren
IF !(any) = 'J'
   STORE '.OR.' TO logic
ELSE
   STORE '.AND.' TO logic
ENDIF (any # ' ' )

*** Such-String für Autor,Titel,Verlag und Datum erstellen
STORE ' ' TO bed

*** Nach dem Autor suchen
IF m:autor # ' '
   STORE !(TRIM(m:autor)) TO m:autor
   STORE 'm:autor $ ! (autor)' +logic TO bed
ENDIF (m:autor # ' ' )

*** Nach dem Titel suchen
IF m:titel # ' '
```

Abb. 21.10: Das Programm LIBSEARC.CMD (Forts.)

```
   STORE !(TRIM(m:titel)) TO m:titel
   STORE bed+ 'm:titel $ !(titel)' +logic TO bed
ENDIF (m:titel # ' ' )

*** Nach dem Verlag suchen
IF m:verlag # ' '
   STORE !(TRIM(m:verlag)) TO m:verlag
   STORE bed+ 'm:verlag $ !(verlag)' +logic TO bed
ENDIF (m:verlag # ' ' )

*** Nach dem Datum suchen
IF m:datum # ' '
   STORE bed+ 'm:datum = datum' + logic TO bed
ENDIF (m:datum # ' ' )

*** Aus Suchwörtern und Suchbedingungen einen Such-String machen
STORE 1 TO zaehler
DO WHILE zaehler < 6
   STORE STR(zaehler,1) TO sub
   IF m:swort&sub #  ' '
      STORE ' ' +!(TRIM(m:swort&sub))+ ',' TO m:swort&sub
      STORE bed+ 'm:swort&sub $ !(schluessel)' +logic;
      TO bed
   ENDIF (swort&sub # ' ' )
   STORE zaehler+1 TO zaehler
ENDDO (zaehler < 6)

*** Das letzte .AND. oder .OR. beseitigen
IF !(any) = 'J'
   STORE $(bed,1,len(bed)-4) TO bed
ELSE
   STORE $(bed,1,len(bed)-5) TO bed
ENDIF (any # ' ' )

*** Wenn keine Suchbedingungen eingegeben,
*** ist es immer richtig (T)rue
IF len(bed) = 1
   STORE '1=1' TO bed
ENDIF (bed = ' ' )

*** Wenn Drucker gewählt, einschalten
IF !(drucker)= 'J'
   SET PRINT ON
ENDIF

*** Wenn Datei gewählt, erstellen
IF !(to:text)= 'J'
```

Abb. 21.10: Das Programm LIBSEARC.CMD (Forts.)

```
   SET ALTERNATE TO &text:datei
   SET ALTERNATE ON
ENDIF (to:text=j)

*** Titel drucken und Zeilen zählen
ERASE
? '                        Bibliographie'
?
STORE 2 TO lf
*** Jetzt mit der Suche beginnen und
*** die Ausgabe formatieren
GO TOP
DO WHILE .NOT. EOF
   *** Schlüsselwörter zum Suchen und Drucken
   *** hintereinanderschreiben
   STORE 1 TO zaehler
   STORE ' ' TO schluessel
   DO WHILE zaehler <= 6
      STORE STR(zaehler,1) TO sub
      IF swort&sub # ' '
         STORE schluessel+TRIM(swort&sub)+ ', ' TO schluessel
      ENDIF (swort&sub # ' ' )
      STORE zaehler+1 TO zaehler
   ENDDO WHILE zaehler <= 6
   *** Etwas gefunden ?
   IF &bed
      *** Dann für Ausgabe formatieren
      STORE TRIM(autor)+ ', ' +TRIM(titel)+ ', ' +;
      TRIM(verlag)+ ', ' TO lang
      IF ausgabe # ' '
         STORE lang+TRIM(ausgabe)+ ', ' TO lang
      ENDIF ausgabe # ' '
      IF ort # ' '
         STORE lang+TRIM(ort) + ', ' TO lang
      ENDIF ort # ' '
      IF seiten # ' '
         STORE lang + TRIM(seiten)+ ', ' TO lang
      ENDIF seiten # ' '

      *** Datum umformen
      STORE $(datum,1,2) TO tag
      STORE $(datum,4,2) TO monat
      STORE $(datum,7,2) TO jahr
      STORE mo&monat TO e:datum
      IF val(tag) < 10
         STORE $(tag,2,1) TO tag
      ENDIF val(tag)
```

Abb. 21.10: Das Programm LIBSEARC.CMD (Forts.)

```
IF val(tag)>0
   STORE tag + '. ' + e:datum TO e:datum
ENDIF val(tag)>0
STORE e:datum+ ' 19' +jahr TO e:datum
STORE lang + e:datum+ '.' TO lang

*** Zeile drucken
STORE 76 TO rand
IF len(lang) <= rand
   ? lang
   STORE lf+1 TO lf
ELSE
   DO WHILE $(lang,rand,1) # ' '
      STORE rand - 1 TO rand
   ENDDO (zeichen ganz rechts kein leerzeichen)
   ? $(lang,1,rand)
   ? $(lang,rand+1)
   STORE lf+2 TO lf
ENDIF (lang <= rand)

*** Wenn Stichwörter gewünscht, anzeigen
IF !(zeig:swrts) = 'J'
   *** Das führende Komma löschen und
   *** einen Punkt am Ende anhängen
   STORE len(TRIM(schluessel)) TO length
   STORE $(schluessel,1,length-1) TO schluessel
   STORE 'STICHWÖRTER:' +schluessel TO schluessel
   IF len(schluessel) > 10
      STORE schluessel+ '.' TO schluessel
   ENDIF len(schluessel) > 10
   STORE 76 TO rand
   IF len(schluessel) <= rand
      ? schluessel
      STORE lf+1 TO lf
   ELSE
      DO WHILE $(schluessel,rand,1) # ' '
         STORE rand - 1 TO rand
      ENDDO (zeichen ganz rechts nicht ' ' )
      ? $(schluessel,1,rand)
      ? '           ' +$(schluessel,rand+1)
      STORE lf+2 TO lf
   ENDIF (len(schluessel) <= rand)
ENDIF (!(zeig:swrts=j)

*** Wenn Zusammenfassung gewählt, ausdrucken
IF !(zeig:zus) = 'J'
   STORE 'TEXT : ' +TRIM(zusammen) TO lang
```

Abb. 21.10: Das Programm LIBSEARC.CMD (Forts.)

```
         DO WHILE len(lang) > 76
            STORE 76 TO rand
            DO WHILE $(lang,rand,1) # ' '
               STORE rand -1 TO rand
            ENDDO WHILE $(lang,rand,1) # ' '
            ? $(lang,1,rand)
            STORE lf+1 TO lf
            STORE $(lang,rand+1) TO lang
         ENDDO (len(lang>76))
         ? lang
         STORE lf+2 TO lf
      ENDIF (zeig:zus=j)

      *** Eine Leerzeile dazwischendrucken
      ?
      STORE lf+1 TO lf

      *** Gedruckte Zeilen zählen und das nächste Blatt nehmen
      *** oder, wenn nötig, Pause machen
      IF !(drucker)= 'J' .AND. lf >=55
         EJECT
         ? '                                Bibliographie'
         ?
         STORE 2 TO lf
      ENDIF !(drucker)= 'J' .AND. lf >=55

      IF !(drucker) # 'J' .AND. !(to:text) # 'J' .AND. lf >=20
         ?
         ? 'Beliebige Taste drücken ...'
         WAIT
         STORE 2 TO lf
         ERASE
      ENDIF (NOT drucker)
   ENDIF (datensatz gefunden)
   SKIP
ENDDO (WHILE .NOT. EOF)

SET PRINT OFF

*** Wenn notwendig, Datei schließen
IF !(to:text) = 'J'
   SET ALTERNATE OFF
   SET ALTERNATE TO
ENDIF (to:text = j)

*** Wenn Daten nur auf dem Bildschirm erscheinen und
*** nicht gedruckt werden, Pause einlegen
```

Abb. 21.10: Das Programm LIBSEARC.CMD (Forts.)

```
IF !(drucker) # 'J'
   ?
   ? 'Beliebige Taste drücken ...'
   WAIT
ENDIF !(drucker) # 'J'

*** Alle Memory-Variablen außer "wahl" löschen
RELEASE ALL except wahl
RETURN
```

Abb. 21.10: Das Programm LIBSEARC.CMD (Forts.)

Unterstreichungen

Vielleicht möchten Sie die Literaturverwaltung oder ein anderes Programm weiter ausbauen und die Unterstreichung wichtiger Textstellen hinzufügen. Zum Beispiel könnten Sie die Bibliographien mit unterstrichenen Titeln ausdrucken:

Bibliographie

Glass, Jim, Word Processor File Conversion, PC Tech Journal, Vol. 1, No. 2, 168–179, September 1983

Levine, Joel H., Hard Disks for Portables, Portable Computer, Vol. 2, No. 3, 38–41, 10. März 1984

Trost, S. und King, R., Doing Business with Multiplan, SYBEX, Berkeley, CA, 1984

Es gibt zwei Möglichkeiten, gedruckte Daten zu unterstreichen. Die erste Möglichkeit verlangt, daß Sie spezielle Druckerbefehle vor und hinter die Daten plazieren, bevor Sie drucken. Sie können diese Befehle in Ihrem Druckerhandbuch finden oder über den Druckerhersteller erfahren.

Hier ein Beispiel: Der Juki-Drucker benutzt für den Beginn der Unterstreichung die Zeichenfolge Esc-E und Esc-R für deren Ende. Das Symbol Esc steht für die ESC-Taste und wird in dBASE als CHR(27) (ASCII-Zeichen 27) dargestellt. Sie können also im Kopf des LIBSEARC-Programms, wo die Speichervariablen erstellt werden, die Variablen für die Unterstreichung mit einfügen, wie hier gezeigt:

```
STORE CHR(27)+"E" TO unter:ein
STORE CHR(27)+"R" TO unter:aus
```

Sobald das Programm die Varibale UNTER:EIN entdeckt, beginnt es, die Daten unterstrichen auf dem Drucker auszudrucken. Mit der Variablen UNTER:AUS wird die Unterstreichung beendet.

Das LIBSEARC-Programm speichert für jede Referenz den Autor, den Titel und den Verlag in eine Variable namens LANG, wie Sie an diesen Programmzeilen sehen:

```
********************* Formatierung beginnen
STORE TRIM(autor)+","+TRIM(titel)+","+;
      TRIM(verlag)+"," TO lang
```

Um den Titel zu unterstreichen, umgeben Sie das TITEL-Feld mit den Druckercodes. Da wir diese Druckeranweisungen nur verwenden wollen, wenn der Bericht gedruckt wird, müssen wir die vorhandenen Zeilen durch diese ersetzen:

```
********************* Formatierung beginnen
STORE TRIM(autor)+"," TO lang

IF !(drucker)= 'J'
   STORE lang+unter:ein+TRIM(titel)+unter:aus+;
     "," TO lang
ELSE
   STORE lang+TRIM(titel)+"," TO lang
ENDIF (drucker=J)

STORE lang+TRIM(verlag)+"," TO lang
```

Jetzt umschließen die Variablen UNTER:EIN und UNTER:AUS nur das TITEL-Feld, wenn der Bericht ausgedruckt wird.

Wenn Sie das Programm so modifiziert haben, die Literaturverwaltung das nächste Mal laufen lassen und dabei den Ausdruck der Daten verlangen, werden die Titel unterstrichen ausgedruckt.

Die nächste Methode benutzt WordStar oder ein anderes Textprogramm, um Unterstreichungen zu drucken. In diesem Fall schickt das Programm die Referenzen in eine Textdatei und umrahmt den TITEL mit einem selten benutzten Zeichen. Bevor dann die Bibliographie ausgedruckt wird, kann der Benutzer mit der Suchen-und-Ersetzen-Funktion die selten benutzten Zeichen durch die entsprechenden Druckcodes für die Unterstreichung ersetzen.

Um die Textprogramm-Möglichkeit zu benutzen, müssen Sie das LIBSEARC-Programm so modifizieren, daß es das TITEL-Feld mit sel-

ten benutzten Zeichen (z.B. einem #) umgibt, wenn die Referenzen in die Textdatei geschrieben werden. Das LIBSEARC-Programm benutzt die Variable TO:TEXT, um zu entscheiden, ob eine Textdatei erstellt wird oder nicht. Deshalb brauchen wir LIBSEARC.CMD nur in diesen Zeilen zu verändern:

```
********************* Formatierung beginnen
STORE TRIM(autor)+","+TRIM(titel)+","+;
      TRIM(verlag)+"," TO lang
```

Wir schreiben eine Routine dazu, die das TITEL-Feld mit dem #-Zeichen umgibt, wenn die Daten in eine Textdatei geschrieben werden:

```
********************* Formatierung beginnen
STORE TRIM(autor)+"," TO lang

IF !(to:text)= 'J'
   STORE lang+ +TRIM(titel)+ +"," TO lang
ELSE
   STORE lang+TRIM(titel)+"," TO lang
ENDIF (to:text=J)

STORE lang+TRIM(verlag)+"," TO lang
```

Nachdem das Programm so verändert wurde, schreibt es #-Zeichen vor und hinter die Titel der Referenzen. Angenommen, der Benutzer möchte eine Textdatei mit diesem Programm erstellen lassen und nennt diese B:BIBLIO.TXT. Das Programm erstellt dann die Textdatei BIBLIO.TXT auf dem Laufwerk B. Um diese Datei nun auszudrucken, verläßt er dBASE (QUIT) und benutzt das Textprogramm. Wenn er mit WordStar arbeitet, legt er die Diskette mit dem Textprogramm in das Laufwerk A und gibt den Befehl WS B:BIBLIO.TXT. Die Referenzen werden auf dem Bildschirm erscheinen, und die Titel werden von #-Zeichen umgeben sein. Das sieht etwa so aus:

Bibliographie

Glass, Jim, #Word Processor File Conversion#, PC Tech Journal,Vol.1, No. 2, 168–179, September 1983

Levine, Joel H., #Hard Disks for Portables#, Portable Computer,Vol. 2, No. 3, 38–41, 10. März 1984

Trost, S. und King, R., #Doing Business with Multiplan#, SYBEX,Berkeley, CA, 1984

Bevor er die Datei jetzt ausdruckt, muß er alle # in den entsprechenden Unterstreichcode verwandeln. In unserem WordStar-Beispiel gibt er ^QA ein, um die Suchen-und-Ersetzen-Routine aufzurufen. WordStar fragt:

```
SUCHEN?_
```

Der Benutzer gibt # ein und dann RETURN. WordStar fragt dann weiter:

```
AUSTAUSCH MIT?_
```

Der Benutzer drückt jetzt die Control-Taste und gibt PS ein und noch einmal RETURN (der Unterstreichbefehl bei WordStar lautet ^PS). Dann fragt das Textprogramm:

```
OPTIONEN? (? FÜR INFO)_
```

Hier kann man nun GN (bedeutet: im gesamten Text, ohne Nachfragen) eingeben und RETURN drücken. WordStar ersetzt dann alle #-Zeichen durch ^PS, wie Sie es hier an unserem Beispiel sehen:

Bibliographie

Glass, Jim, ^PSWord Processor File Conversion^PS, PC Tech Journal, Vol.1, No. 2, 168-179, September 1983

Levine, Joel H.,^PSHard Disks for Portables^PS, Portable Computer, Vol. 2, No. 3, 38-41, 10. März 1984

Trost, S. und King, R., ^PSDoing Business with Multiplan^PS, SYBEX, Berkeley, CA, 1984

Nachdem der Austausch vollzogen ist, kann der Benutzer die Datei BIBLIO.TXT mit dem Befehl ^KD (Control-Taste drücken und KD eingeben) abspeichern. Dann wählt er die Option P aus dem WordStar-Hauptmenü (eine Datei drucken). Das Programm fragt:

```
DATEI ZUM DRUCKEN?_
```

der Benutzer gibt den Dateinamen an (hier BIBLIO.TXT), und die Referenzen werden mit unterstrichenen Titeln gedruckt.

Unser nächstes Programm, das eine Datei mit allen Stichwörtern der Literaturverwaltung erstellt und diese in einem zweispaltigen Bericht anzeigt, ist längst nicht so komplex wie das vorherige Programm.

Stichwörterbericht der Literaturverwaltung

Wie in diesem Kapitel schon früher gesagt, zeigt der Stichwörterbericht alle Stichwörter der Datenbank und die Anzahl der Referenzen dazu. Der Bericht wird erstellt, indem alle sechs Stichwörter jedes Datensatzes der LIBRARY-Datenbank gelesen und in einer eigenen Datenbank, TOPICS.DBF, gespeichert werden. Bevor es aber soweit ist, werden alle existierenden Einträge in dieser Datei gelöscht. Es könnte ja sein, daß seit dem letzten Ausdruck einige Stichwörter neu aufgenommen, geändert oder gelöscht worden sind.

Das Programm für die Stichwortübersicht heißt SUBJECTS.CMD, und der Pseudocode steht in der Abb. 21.11.

```
Frage, ob Bericht ausgedruckt werden soll

Zeige Nachricht, während die Dateien erstellt werden

Lösche alle existierenden Sätze aus der TOPICS-Datei
Indiziere die leere TOPICS-Datei

Nimm LIBRARY und TEMP-Datei in Gebrauch

Zähle die Eintragungen in LIBRARY und zeige Benutzer die
Anzahl der bisher kopierten Datensätze

Schleife durch die LIBRARY-Datei

    Zeige Nachricht, daß das Programm arbeitet
    Durchlaufe die sechs Stichwörter in dem LIBRARY-Datensatz
        Versuche, Stichwort in TOPICS zu finden
        Wenn gefunden, erhöhe die Anzahl-Variable

    Weiter mit Schleife durch die sechs Stichwörter
Gehe zum nächsten Datensatz in LIBRARY
Weiter mit Schleife durch LIBRARY (bis Ende der Datei)

Schließe die SECONDARY-Datei und mache SUBJECTS zur
PRIMARY

    Schalte Drucker ein, wenn gewünscht
```

Abb. 21.11: Pseudocode für das Programm SUBJECTS.CMD

```
    Beginne Schleife durch TOPICS-Datei

    Drucke Stichwort und Anzahl in der ersten Spalte
    Weiter zum nächsten Datensatz
    Wenn nicht Ende der Datei, drucke Stichwort und Anzahl in
    nächste Spalte

    Wenn Ausdruck gewünscht, besorge Seitenvorschub

    Wenn Bildschirmausgabe gewünscht, besorge Pausen

Weiter zum nächsten Datensatz in TOPICS
Weiter mit Schleife durch TOPICS

Wenn Report nicht ausgedruckt wird, anhalten, bevor es zum
Hauptemnü zurückgeht

Schalte Drucker aus, lösche Speichervariablen
und kehre zum Hauptmenü zurück
```

Abb. 21.11: Pseudocode für das Programm SUBJECTS.CMD (Forts.)

Der erste Schritt im Programm ist, den Benutzer zu fragen, ob er den Bericht ausdrucken möchte:

```
************************************ SUBJECTS.CMD
****************** Eine Stichwort-Datei erstellen

*** Fragen, ob drucken oder nicht
ERASE
STORE " " TO jn
§ 5,5 SAY 'Drucken? (J/N) ' GET jn PICT '!'
READ
```

Das Programm löscht dann alle existierenden Daten in der TOPICS-Datenbank. Da dies etwas Zeit in Anspruch nimmt, geben wir dem Benutzer eine Nachricht auf den Bildschirm, damit er sieht, daß der Computer arbeitet:

```
ERASE
? "Die Stichwort-Datei wird vorbereitet ..."
```

Es gibt zwei Möglichkeiten, alle Datensätze einer Datenbank zu löschen.

Bei der ersten werden die Befehle DELETE ALL und PACK verwendet. Mit einer großen Index-Datei kann das unangenehm lange dauern. Die andere Möglichkeit ist, die Struktur einer Datenbank zuerst in eine Zwischendatei und dann wieder zurück in die Originaldatei zu speichern. Das dauert nur ein paar Sekunden. Dann können Sie die leere Datei indizieren, was auch nur Sekunden dauert. Die unten vorgestellte Routine benutzt den COP STRUCTURE-Befehl, um die TOPICS-Datenbank zu leeren:

```
*** Zuerst alle Datensätze in der Datenbank
*** TOPICS.DBF löschen
USE topics
COPY STRUCTURE TO temp
USE temp
COPY STRUCTURE TO topics
USE topics
INDEX ON !(subject) TO topics
```

Als nächstes wollen wir ein Art Aktualisierung der TOPICS-Datei von der LIBRARY-Datei vornehmen, so daß beide Dateien gleichzeitig geöffnet sein müssen:

```
********** Öffne Dateien LIBRARY und TOPICS
SELE PRIM
USE library
SELE SECO
USE topics INDEX topics
```

Die Aktualisierung dauert ein paar Minuten. Wir halten den Benutzer über den Fortgang der Arbeit auf dem laufenden, indem wir anzeigen, wie viele Datensätze noch zu bearbeiten sind, und die zur Zeit bearbeitete Datensatznummer angeben:

```
Es sind insgesamt 174 Einträge ...
Jetzt wird Datensatz Nr. 1 bearbeitet
```

Während das Programm arbeitet, zählt der Zähler hinter „Jetzt wird Datensatz Nr." von 1 bis zu der angegebenen Gesamtzahl der Einträge (174 in diesem Beispiel). Auf diese Art und Weise kann der Benutzer abschätzen, wieviel Zeit noch gebraucht wird, und eine Tasse Kaffee trinken gehen (das verstehe ich unter benutzerfreundlich). Um diese Meldung anzeigen zu können, müssen wir erst feststellen, wie viele Datensätze in der LIBRARY-Datenbank stehen. Das Programm erledigt das, indem es die Datei LIBRARY benutzt, zum letzten Datensatz springt und die Datensatznummer in die Variable MAX:NR speichert:

```
** Ermittle die Zahl der Datensätze in der
** Datenbank LIBRARY.DBF,
** damit der Benutzer sieht, daß gearbeitet wird
SELE PRIM
GO BOTT
STORE # TO max:nr
```

Das Programm beginnt dann am Anfang der LIBRARY-Datei mit einer Schleife, die bis zum Ende der Datei gültig ist:

```
********* Durchlaufe LIBRARY.DBF und speichere
********* Stichwörter in die Datenbank TOPICS.DBF
GO TOP
ERASE
DO WHILE .NOT. EOF
```

Innerhalb dieser Schleife wird der Fortschritts-Bericht gezeigt:

```
§ 5,5 SAY 'Es sind insgesamt '+STR(max:nr,4)+;
                                        'Einträge ...'
§ 7,5 SAY 'Jetzt wird Datensatz Nr. '+STR(#,4);
                                      +' bearbeitet'
```

Dann wird eine kleine Schleife zum Analysieren jedes der sechs Stichwörter in jedem Datensatz der LIBRARY-Datei begonnen:

```
********** Durchlaufe die sechs Schlüsselwörter
STORE 1 TO zaehler
DO WHILE zaehler <= 6
STORE STR(zaehler,1) TO sub
```

Jedes Stichwort (das nicht leer ist) speichert das Programm in Großbuchstaben in die Variable FINDEN. Dann nimmt es die SECONDARY-Datei (TOPICS INDEX TOPICS) und versucht, das Stichwort dort zu finden:

```
IF swort&sub # ' '
   STORE !(swort&sub) TO finden
   SELE SECO
   FIND &finden
```

Gibt es das Stichwort schon in der TOPICS-Datei, wird der Anzahl-Zähler (ANZAHL) um 1 erhöht:

```
*** Wenn etwas gefunden, Zähler erhöhen
IF # > 0
   REPLACE anzahl WITH anzahl+1
```

Wenn das Stichwort noch nicht existiert, speichert es das Programm ab und setzt ANZAHL auf 1:

```
      *** sonst in Datei schreiben
      ELSE
         APPEND BLANK
         REPLACE subject WITH swort&sub
         REPLACE anzahl WITH 1
      ENDIF (#>0)
   ENDIF (swort&sub # ' ')
```

Das Programm erhöht dann ZAEHLER um 1, um die Schleife durch die Stichwörter fortzusetzen:

```
     STORE zaehler + 1 TO zaehler
   ENDDO (while zaehler <= 6)
```

Nachdem das Programm alle sechs Stichwörter eines Datensatzes analysiert hat, geht es zum nächsten innerhalb der LIBRARY-Datei und wiederholt den Vorgang:

```
     SELE PRIM
     SKIP
   ENDDO (not eof)
```

Sind alle Datensätze analysiert, kann das Programm die SECONDARY-Datei (LIBRARY.DBF) schließen und TOPICS.DBF zur PRIMARY-Datei machen:

```
*** Sekundäre Datei schließen und TOPICS.DBF zur
*** primären Datei machen
SELE SECO
USE
SELE PRIM
USE topics INDEX topics
```

Wollte der Benutzer die Daten ausdrucken, schaltet das Programm den Drucker ein:

```
***************** Drucker einschalten, wenn verlangt
IF jn = 'J'
   SET PRINT ON
ENDIF (jn = 'J')
```

Das Programm kann jetzt den Bericht drucken. Zuerst löscht es den Bildschirm, druckt Überschrift und Unterzeile, setzt den Zeilenzähler (LF)

auf 4 und beginnt mit der Schleife durch die TOPICS-Datei:

```
********** Den Report in zwei Spalten drucken
ERASE
? '                              Stichwort-Liste'
?
? ' Subject                          Nr.'
?? '   Subject                         Nr.'
STORE 4 TO lf
DO WHILE .NOT. EOF
```

Dann druckt es die Felder SUBJECT und ANZAHL, gefolgt von einigen Leerzeichen, aus der TOPICS-Datei und geht weiter zum nächsten Datensatz der Datenbank:

```
? subject,anzahl,'     '
SKIP
```

Solange das Ende der Datei noch nicht erreicht ist, druckt das Programm die nächsten SUBJECT- und ANZAHL-Felder in die zweite Spalte derselben Zeile, indem es das doppelte Fragezeichen (??)verwendet:

```
IF .NOT. EOF
   ?? subject,anzahl
ENDIF (not eof)
```

Dann erhöht das Programm pflichtbewußt den Zeilenzähler (LF) um 1 und kümmert sich um den Seitenvorschub und die Bildschirmpausen, wenn nötig:

```
STORE lf+1 to lf
*** Zum Druck formatieren
IF jn='J' .AND. lf>= 55
   EJECT
   ? '    Subject                          Nr.'
   ?? '         Subject                          Nr.'
   STORE 2 TO lf
ENDIF (jn= 'J' and lf >= 55)

*** Zur Anzeige auf dem Bildschirm formatieren
IF jn # 'J' .AND. lf >= 20
   ?
   ? 'Weiter mit beliebiger Taste ...'
   WAIT
   STORE 2 TO lf
ENDIF (jn # 'J' and lf >= 20)
```

Um weitere Daten zu drucken, geht das Programm dann zum nächsten Datensatz, bis das Ende der Datei erreicht ist:

```
   SKIP
ENDDO (while not eof)
```

Wurde der Bericht nur auf den Bildschirm ausgegeben, hält das Programm an, bevor es zum Hauptmenü zurückkehrt:

```
********** Wird der Report nicht gedruckt, eine
********** Pause vor der Rückkehr zum Menü
IF jn # 'J'
   ?
   ? 'Mit beliebiger Taste zurück zum Menü'
   WAIT
ENDIF (jn # j)
```

Dann schaltet es den Drucker aus, löscht die Speichervariablen und geht zum Hauptmenü zurück:

```
*** Drucker ausschalten, Memory-Variablen löschen
*** und zum Hauptmenü zurück
USE
SET PRINT OFF
RELEASE ALL EXCEPT wahl
RETURN
```

Die Abb. 21.12 zeigt das vollständige SUBJECTS.CMD-Programm. Damit ist unser Literaturverwaltungssystem komplett.

```
********************************** SUBJECTS.CMD
********************** Eine Stichwort-Datei erstellen

*** Fragen, ob drucken oder nicht
ERASE
STORE " " TO jn
§  5,5 SAY 'Drucken? (J/N) ' GET jn PICT '!'
READ

ERASE
? "Die Stichwortdatei wird vorbereitet ..."

*** Zuerst, alle Datensätze in der Datenbank
*** TOPICS.DBF löschen
```

Abb. 21.12: Das Programm SUBJECTS.CMD

```
USE topics
COPY STRUCTURE TO temp
USE temp
COPY STRUCTURE TO topics
USE topics
INDEX ON !(subject) TO topics

********** Öffne beide Dateien LIBRARY und TOPICS
SELE PRIM
USE library
SELE SECO
USE topics INDEX topics

** Ermittle die Zahl der Datensätze in der Datenbank LIBRARY.DBF,
** damit der Benutzer sieht, daß gearbeitet wird
SELE PRIM
GO BOTT
STORE # TO max:nr

************ Durchlaufe LIBRARY.DBF und speichere Stichwörter
************ in die Datenbank TOPICS.DBF
GO TOP
ERASE
DO WHILE .NOT. EOF

   §  5,5 SAY 'Es sind insgesamt '+STR(max:nr,4)+'Einträge ...'
   §  7,5 SAY 'Jetzt wird Datensatz Nr. '+STR(#,4)+' bearbeitet'

   ******************** Durchlaufe die sechs Schlüsselwörter
   STORE 1 TO zaehler
   DO WHILE zaehler <= 6
      STORE STR(zaehler,1) TO sub

      IF swort&sub # ' '
         STORE !(swort&sub) TO finden
         SELE SECO
         FIND &finden

         *** Wenn etwas gefunden, Zähler erhöhen
         IF # > 0
            REPLACE anzahl WITH anzahl+1

            *** sonst in Datei schreiben
         ELSE
            APPEND BLANK
            REPLACE subject WITH swort&sub
            REPLACE anzahl WITH 1
```

Abb. 21.12: Das Programm SUBJECTS.CMD (Forts.)

```
         ENDIF (#>0)
       ENDIF (swort&sub # ' ')

       STORE zaehler + 1 TO zaehler
   ENDDO (while zaehler <= 6)

   SELE PRIM
   SKIP
ENDDO (not eof)

*** Sekundäre Datei schließen und TOPICS.DBF zur primären
******** Datei machen
SELE SECO
USE
SELE PRIM
USE topics INDEX topics

***************** Drucker einschalten, wenn verlangt
IF jn = 'J'
   SET PRINT ON
ENDIF (jn = 'J')

********** Den Report in zwei Spalten drucken
ERASE
? '                          Stichwort-Liste'
?
? ' Subject                      Nr.'
?? '   Subject                     Nr.'
STORE 4 TO lf
DO WHILE .NOT. EOF
   ? subject,anzahl,'    '
   SKIP
   IF .NOT. EOF
      ?? subject,anzahl
   ENDIF (not eof)
   STORE lf+1 to lf

   *** Zum Druck formatieren
   IF jn='J' .AND. lf>= 55
      EJECT
      ? '    Subject                        Nr.'
      ?? '         Subject                        Nr.'
      STORE 2 TO lf
   ENDIF (jn= 'J' and lf >= 55)

   *** Zur Anzeige auf dem Bildschirm formatieren
   IF jn # 'J' .AND. lf >= 20
```

Abb. 21.12: Das Programm SUBJECTS.CMD (Forts.)

```
      ?
      ? 'Weiter mit beliebiger Taste ...'
      WAIT
      STORE 2 TO lf
   ENDIF (jn # 'J' and lf >= 20)
   SKIP
ENDDO (while not eof)

************************* Wird der Report nicht gedruckt, eine
************************* Pause vor der Rückkehr zum Menü
IF jn # 'J'
   ?
   ? 'Mit beliebiger Taste zurück zum Menü'
   WAIT
ENDIF (jn # j)

*** Drucker ausschalten, Memory-Variablen löschen
*** und zum Hauptmenü zurück
USE
SET PRINT OFF
RELEASE ALL EXCEPT wahl
RETURN
```

Abb. 21.12: Das Programm SUBJECTS.CMD (Forts.)

Zusammenfassung

In diesem letzten Kapitel haben wir Programme zum Anzeigen von Berichten in der Literaturverwaltung entwickelt und die Entwicklung des Literaturverwaltungssystems abgeschlossen. Wir haben einige sehr weit fortgeschrittene Datenbankverwaltungs-Techniken in nur zwei Programmen diskutiert:

1. LIBSEARC.CMD, das es dem Benutzer ermöglicht, komplexe Suchoperationen in der Stichwortdatei LIBRARY vorzunehmen, indem er einfach einige Felder einer einfachen Maske ausfüllt. Fortgeschrittene Techniken, die wir dabei verwendet haben sind:

   - Stichwort-Suche,
   - komplexe, menügesteuerte Suche,
   - Wortumbruch bei langen Zeichenketten,
   - Übersetzen des Datums in Wörter,
   - Unterstreichen und
   - das Erstellen einer Textdatei für einen Bericht.

2. SUBJECTS.CMD, das eine Liste aller einzigartigen Stichwörter in der LIBRARY-Datei erstellt. Einige der fortgeschrittenen Programmiertechniken dabei sind:

   - das Übertragen mehrerer Felder jedes Datensatzes in ein Feld einer anderen Datenbank,
   - das Erstellen einer Datenbank und Zählen, wie oft ein einzelnes Stichwort in der Datenbank steht, und
   - das Drucken von Daten aus einem Feld in zwei alphabetisch geordneten Spalten.

Obwohl einige der Techniken dem Benutzer etwas Mühe bereiten werden, lohnt es sich, sie zu studieren. Sie sind sehr wirkungsvoll und bieten manche Annehmlichkeit. Mehrfache Stichwörter zu jedem Datensatz in einer Datenbank bei einer Suche zu benutzen, kann in vielen Vergleichssituationen hilfreich sein. Und die Suche mit Hilfe einer Maske ist erheblich bequemer, als komplizierte Suchbefehle hinter den dBASE-Prompt zu schreiben.

Stichwortverzeichnis

! 43, 278
$ 203

ACCEPT 27
ACCT.FRM 308
ADDNUMBS.CMD 156
ADDTRANS.CMD 279
ALLMAST.FRM 157
AMASK1.FMT 58
APPEND 33, 53, 59
APPEND BLANK 155, 189
APPEND.CMD 61
Adressenliste 78
Adressenverwaltung 40
Aktualisieren 223, 282
Aktualisieren nach Änderung 229
Aktuelle Bilanz 315
Ausgabekonten 258, 313
Automatisches Numerieren 186
Ändern
  Buchhaltungsdaten 341
  Befehlsdateien 15
  komplex 229
  Inventar NEWSTOCK 238
  Literatur-Referenzen
  Versandliste-Daten 102
  Verkaufs-Vorgänge 223
Ändern, mehrere Dateien 229
Ändern, Wareneingang 238
Ändern, Verkäufe 229
Änderungsmasken, Buchhaltung,
  Installation 258
  Berichte 305
  Software 254
  System 250
  System-Menü 268
  Vorgänge 271

B: 15
BAK-Datei 17
BEDIT-CMD 354
BINSTALL.CMD 265
BOOKS.CMD 268
BREPORTS.CMD 307
BROWSE 31
Berechnungen auf dem Bildschirm 189
Bericht, Bestellungen 160
Bericht, Nachbestellung 158
Bericht, Verkäufe 193

Bericht, zweispaltig 383, 417
Berichte, ausdrucken 162, 199
Bestellungen, Formular 161
Bestellungen, plazieren 160
Bilanz, Quartal 286
Bilanz, aktualisieren 282
Bilanz, jährlich 285
Bilanz, monatlich 286
Bilanzberechnung 282
Blöcke bewegen 19
Buchhaltung Datenbanken 252

CASE 49
CHECKS.CMD 337
CHR 409
CHR(7) 154
COA.DBF 253
COA.FMT 342
COA.FRM 318
COA.NDX 253
COAREPT.CMD 325
COUNT 28, 105, 113
CP/M 17
CREATE 43
CURREPT.CMD 316

DELETE 101
DOS 16
DUPES.CMD 132
Datei schließen 190
Daten übertragen 183, 272
Datensätze löschen 109
Datensätze zurückholen 110
Druckerpause 87
Duplikate 127

EDIT 30, 33, 101
EDIT.CMD 107
EJECT 198
ENDCASE 22, 49
ENDDO 21
ENDTEXT 49
EOF 81
ERASE 17
Eingebundenes Suchen 393
Einnahmekonten 257
Einnahmen-/Ausgaben-Übersicht 305
Einrückungen 20
Etiketten 35, 86, 90
Etiketten ausrichten 87

FILE 73
FIND 27, 28, 35, 46, 154, 216
FIND (Nummer) 276
FIND mit zwei Feldern 123
FIND und WHILE 25–27, 105, 107
FMT-Datei 55, 152, 341
FOR, berechnet 163
Formatdateien 55
Formatieren langer Zeichenketten 397, 398, 400
Formbriefe 97

GENINFO.DBF 252
GENINFO.FMT 345
GET 49
Generelle Informationen-Datei 252
Großschreibung 43, 278

Hauptdateien 135, 147, 222
  ändern 173
  durchsuchen 183
  Berichte 157
  aktualisieren 223
Hierarchischer Aufbau 44

IF FILE( ) 73
IF, berechnet 167
IMENU.CMD 144
INDEX 26, 32, 33, 43
INPUT 198
INT 277
ISCREEN1.FMT 151
Index über mehrere Felder 43
Indexdateien, mehrfach 32
Inventar 135
  Datenbankaufbau 138, 139
  Hauptmenü 141
  Eingangslager 206
  Nachbestellungen 158
  Verkäufe 180
  Verkaufsbericht 190
  Software-Design 140
  System 137

KONTO.NDX 253
Konten, summiert 360
Kontenrahmen 251, 267
  Bilanz 319
  Bericht 318
  Aktualisierung 282

Löschen, Referenzen 375
LABELS.CMD 35, 93
LIBADD.FMT 369
LIBDEL.CMD 376
LIBEDIT.CMD 375
LIBEDIT.FMT 372
LIBRARY.CMD 367
LIBRARY.DBF 364
LIBSEARC.CMD 403
LIBSEARC.FMT 387
LIST WHILE 27
LOCATE 198
LOOKUP.CMD 124
Laufwerkangabe 15
Leistung verbessern 25
Leistungssteigerung, Nachteile 33
Literaturverwaltung 363, 368
  Datenbanken 364
  Berichte 381
  Programme 366
Logisches Feld 139
Logisches Suchen 391

MAIL.CMD 50
MAIL.DBF 25, 42
MAKEMEMS.CMD 329
MASTER.DBF 138, 147
MASTER.NDX 138
MEDIT.CMD 176
MEM-Datei 329
MMENU.CMD 150
MMERGE.CMD 96
MODIFY COMMAND 15
MREPORTS.CMD 164
MS-DOS 16
MailMerge 97
Makrobefehle 348
Makros 392, 397
Maximale Leistung erreichen 25–37
Memory-Datei 329
Menü 46
Monat suchen 310

NAMEN.NDX 43, 120
NDX-Datei 140
NEWEDIT.CMD 243
NEWREPS.CMD 218
NEWSTOCK.CMD 212
NEWSTOCK.DBF 140, 205
NEWSTOCK.FRM 214
NEWSTOCK.NDX 140, 205
NMENU.CMD 208
NOUPDATE 34, 168, 224
Numerierung 187

OFF 198
ONORDER.FRM 159
ORDERS.CMD 171
OTHERWISE 49

P. 226
PACK 101
PC-DOS 16
PICTURE 49, 60
POS.CMD 191
PRG-Datei 16
PRIMARY 185, 190
Programm installieren 258
Programm, Leistung 35
Programm-Kommentare 21
Programmfluß 45
Programmieren, strukturiert 44
Pseudocode 45

READ 49
REGISTER.CMD 296
REGISTER.DBF 254, 291
REGISTER.FMT 346
REGISTER.FRM 299
REGREPT.CMD 302
RELEASE 73
REORDER.FRM 158
REPLACE 34, 167, 225
REPORT 29, 77, 157, 158, 213
REPORT, Spalten 160
REPORT, zwei Spalten 383, 417
REPORTS.CMD 73
RESTORE 331

SALEDIT.CMD 235
SALES.DBF 139, 179
SALES.FRM 195
SALES.NDX 139
SALREPS.CMD 195
SAVE 331
SAY, berechnet 188
SCREEN 61
SECONDARY 226
SELECT 226
SET ALTERNATE 396
SET CONSOLE 88
SET DEFAULT 16
SET FORMAT 59, 62, 106
SET INDEX 33, 69
SET PRINT 81
SMENU.CMD 182
SORT 31
STR 167
SUBJECTS.CMD 419
SUM 28
Schablonen 60
Scheckregister 291, 292
Scheckregister-Bilanz 292, 296
Schecks drucken 327
Schlüsselfeld, 137
Schlüsselwort 383, 386, 396
Seitennumerierung 80
Seitenumbruch 81
Sichern, Dateien 16
Sichern, Variablen 331
Software, Design 40–46
Software, Struktur 46
Sortieren, menügesteuert 69
Sortieren, nach Datum 202
Suche, schnell 119
  eingebaut 382
  logisch 381
  nach Referenzen 383
  nach Stichwörtern 384
  mehrere Schlüsselwörter 395
  mehrere Stichwörter 386
  menügesteuert 69

TEXT 49
TOPICS.DBF 365
TRANS.DBF 252
TRANS.FMT 344
TYPE 88
Textdateien 391
Textprogramm 15

UPDATE 223, 224
UPDATE mit subtrahieren 224
UPDATE.CMD 283
UPDATER.CMD 227
USE 190
USING 190
Unterkonten 250, 309
Unterstreichen 409

VAL 106
Verarbeitungszeit 27
Vergleich 183, 186, 271

WHILE 26–29, 104–109, 216
WordStar 17, 94
Wortumbruch 398

ZAHLWORT.MEM 329
Zahlen in Wörter umwandeln 328
Zahlungsnummern 250
Zeichenketten, formatieren 398
Zweispaltige Berichte 383, 417
Zwischensummen 313

Die SYBEX-Bibliothek

Einführende Literatur

MEIN HEIMCOMPUTER
von N. Hesselmann – zeigt, was ein Heimcomputer ist und was man mit ihm anfangen kann, von den Chips bis zu Tips für den Kauf. 256 Seiten, 124 Abb., Best.-Nr. **3064** (1985)

CHIP UND SYSTEM: Einführung in die Mikroprozessoren-Technik
von Rodnay Zaks – eine sehr gut lesbare Einführung in die faszinierende Welt der Computer, vom Microprozessor bis hin zum vollständigen System. 2., überarbeitete und aktualisierte Ausgabe. 568 Seiten, 325 Abbildungen, Best.-Nr.: **3601** (1985)

COMPUTER TOTAL VERRÜCKT
von Daniel Le Noury – mit diesem Buch kommen Sie wieder zur Besinnung, nachdem Sie sich halbtot gelacht haben. Ca. 100 Cartoons rund um den Computer. 96 Seiten, Best.-Nr. **3042** (1984)

Sprachen

BASIC

GRUNDKURS IN BASIC
von U. Ströbel – die Einführung in die meistgenutzte Programmiersprache für Lehrer, Schüler und das Selbststudium (Reihe SYBEX Informatik). 208 Seiten, mit Abb., Best.-Nr. **3058** (1985), Lehrerbegleitheft Best.-Nr. **3091** (1985)

Pascal

GRUNDKURS IN PASCAL Bd. 1
von K.-H. Rollke – der sichere Einstieg in Pascal, speziell für Schule und Fortbildung (Reihe SYBEX Informatik). 224 Seiten, mit Abb., Format 17,5x25 cm, Best.-Nr. **3046** (1984), Lehrerbegleitheft Best.-Nr. **3059**

GRUNDKURS IN PASCAL BAND 2
von K.H. Rollke – Mit diesem Buch wird der Pascal-Grundkurs aus der Reihe SYBEX Informatik abgerundet. Für Lehrer, Schüler, Teilnehmer an Pascal-Kursen, Studenten und Autodidakten. 224 Seiten, mit Abb., Best.-Nr. **3061** (1985), Lehrerbegleitheft Best.-Nr. **3090**

DAS PASCAL HANDBUCH
von Jacques Tiberghien – ein Wörterbuch mit jeder Pascal-Anweisung und jedem Symbol, reservierten Wort, Bezeichner und Operator, für beinahe alle bekannten Pascal-Versionen incl. Turbo Pascal. 520 Seiten, 270 Abbildungen, Format 23 x 18 cm, Best.-Nr.: **3614** (1986)

Andere Programmiersprachen

GRUNDKURS IN LOGO
von Karin und Karl-Heinz Hauer – eine umfassende LOGO-Einführung aus der SYBEX Informatik Reihe für Lehrer und Schüler der Sekundarstufe II, Studenten, Hobbyprogrammierer und Autodidakten. 224 Seiten, zahlr. Abb., Best.-Nr. **3088** (1986), Lehrerbegleitheft Best.-Nr. **3089**

Commodore

COMMODORE 64 – GRAFIK + DESIGN
von Ch. Platt – Eine Schritt-für-Schritt-Einführung in die Grafik-Programmierung Ihres C 64. Tips, die Sie in keinem Handbuch finden. 280 S., 150 Abb., Best.-Nr. **3073** (1984)

COMMODORE 64 STARTEXTER
Textverarbeitungs-Kurs aus der Reihe MISTER MICRO – StarTexter ist ein Textverarbeitungskurs mit Doppelnutzen: das Buch führt Sie in die Textverarbeitung mit Ihrem C64 ein, die Diskette bietet Ihnen ein exzellentes Programm – komplett zu einem erstaunlichen Preis! 112 Seiten, Handbuch und Diskette, Best.-Nr.: **3419** (1985)

COMMODORE 64 STARDATEI
von Toni Schwaiger – Der universelle Karteikasten für den C 64 aus der Reihe MISTER MICRO, voll kompatibel zu StarTexter. Diskette und ausführliches Trainingsbuch (ca. 120 Seiten) Best.-Nr. **3413** (1985)

C 128 STARTEXTER
von Toni Schwaiger – Die Textverarbeitung der Spitzenklasse auch für professionelle Anwender mit dem Commodore C 128. Außergewöhnliche features, die den C 128 zum Textverarbeitungs-Star werden lassen – zum kleinen Preis. Diskette + Trainingsbuch (120 Seiten), Best.-Nr. **3415** (1986)

COMMODORE 64 STARTOOL
von Frank U. Müller – Ein weiteres Produkt aus der StarSoftware-Serie. Leistungsfähige Dienstprogramme: Editor/Assembler/Reassembler/Monitor mit außergewöhnlichen features und ausführlichem Handbuch. Diskette mit Trainingsbuch, Best.-Nr. **3417** (1986)

COMMODORE 64 STARPAINTER
von H. Hansen/E. Sonnenschein – Das Grafikprogramm (plus Trainingsbuch) der Spitzenklasse, mit dem Sie sogar professionell arbeiten können. Und das komfortabel, gut verständlich und zum kleinen Preis. Diskette mit ausführlichem Handbuch, Best.-Nr. **3421** (1986)

Schneider

SCHNEIDER CPC STARTEXTER
von Reinhold Krummscheid – Das Textverarbeitungs-Programm der Spitzenklasse auch für den Schneider CPC 464/664/6128. Mit Profi-Möglichkeiten zum kleinen Preis. Diskette + Trainingsbuch, Best.-Nr. **3416** (1986)

Anwendungssoftware

EINFÜHRUNG IN WORDSTAR
von Arthur Naiman – eine klar gegliederte Einführung, die aufzeigt, wie das Textbearbeitungsprogramm WORDSTAR funktioniert, was man damit tun kann und wie es eingesetzt wird. 240 Seiten, 36 Abbildungen, Best.-Nr.: **3019** (1983)

PRAKTISCHE WORDSTAR-ANWENDUNGEN
von J. A. Arca – das Buch für Einsteiger, um nach kurzer Zeit praktische Textverarbeitungs-Probleme zu lösen, eine programmierte Unterweisung zur Leistungsoptimierung mit WORDSTAR. 368 Seiten, 69 Abbildungen, Best.-Nr.: **3057** (1985)

ERFOLG MIT VisiCalc
von D. Hergert – umfassende Einführung in VisiCalc und seine Anwendung. Zeigt Ihnen u. a.: Aufstellung eines Verteilungsbogens, Benutzung von VisiCalc-Formeln, Verwendung der DIF-Datei-Funktion. 224 Seiten, 58 Abbildungen, Best.-Nr.: **3030** (1983)

ERFOLG MIT MULTIPLAN
von Th. Ritter – das Tabellenkalkulations-Programm Multiplan hilft Ihnen bei der Lösung kommerzieller, wissenschaftlicher und allgemeiner Probleme. Lernen Sie die Möglichkeiten kennen, Ihre Software optimal zu nutzen! 208 Seiten, 68 Abbildungen, Best.-Nr.: **3043** (1984)

LOTUS 1-2-3. DATENVERARBEITUNG OHNE VORKENNTNISSE
von S. Heine – für alle, die ohne DV-Kenntnisse das starke Software-Paket LOTUS 1-2-3 für berufliche oder private Anwendungen einsetzen möchten. 264 Seiten, 64 Abb., Best.-Nr. **3052** (1985)

ARBEITEN MIT LOTUS 1-2-3
von B. F. Kehlmann – die wichtigsten Anwendungsfunktionen von LOTUS 1-2-3 im Betrieb anhand praktischer Fallstudien. 232 Seiten, 115 Abb., Best.-Nr. **3078** (1985)

ARBEITEN MIT SYMPHONY
von Douglas Cobb – Ein umfassendes Standardwerk für professionelle Anwender. Für den Kenner von LOTUS 1-2-3 werden alle Erweiterungen von Symphony deutlich und schnell erkennbar herausgearbeitet. 832 Seiten, 420 Abb., Best.-Nr. **3095**(1986)

ARBEITEN MIT APPLEWORKS
von Elna Tymes – Anwender eines Apple IIc oder IIe lernen durch dieses systematisch aufgebaute Arbeitsbuch, mit dem Software-Paket AppleWorks schnell kommerzielle wie private Aufgabenstellungen zu lösen. 240 Seiten, ca. 60 Abbildungen, Best.-Nr.: **3604** (1986)

ARBEITEN MIT dBase II
von A. Simpson – Grundlagen und Programmiertechniken für die Datenbank-Verwaltung mit dBASE II. Zahlreiche praktische Tips. 264 Seiten, 50 Abb., Best.-Nr. **3070** (1984)

DATEIVERWALTUNG SELBST GEMACHT
von A. Simpson – vermittelt die wesentlichen Techniken der Dateiverwaltung, ihre Verwendung in BASIC-Programmen, sowie das Mischen, Sortieren, Aktualisieren und formatierte Ausdrucken von Dateien. 240 S., mit Abb., Best.-Nr. **3085** (1985)

SXBEX RATGEBER dBASE III
von Gerhard Renner – Das Nachschlagewerk zu dBase III für alle, die sich mit ihrer Software auskennen, aber ganz schnell Detail-Informationen benötigen. Übersichtlicher Nachschlageteil mit Piktogrammen und Querverweisen. 432 Seiten, Best.-Nr. **3300** (1986)

ARBEITEN MIT dBASE III
von Alan Simpson – zeigt Ihnen, wie Sie Ihre Datenbank-Verwaltung mit dBASE III noch effektiver machen; mit vielen nützlichen Beispielen. 288 Seiten, mit Abbildungen, Best.-Nr. **3609** (1986)

SYBEX RATGEBER dBASE II
von Gerhard Renner – Das sichere Nachschlagewerk mit kurzem Zugriff für PC-Anwender, die nicht die Zeit haben, Befehle und sonstige Detail-Informationen langwierig zu suchen. Mit vielen Querverweisen und Suchhilfen. Ca. 280 Seiten, Best.-Nr. **3305** (1986)

Fordern Sie ein Gesamtverzeichnis unserer Verlagsproduktion an:

SYBEX-VERLAG GmbH
Vogelsanger Weg 111
4000 Düsseldorf 30
Tel.: (02 11) 61 80 2-0
Telex: 8 588 163

SYBEX INC.
2344 Sixth Street
Berkeley, CA 94710, USA
Tel.: (415) 848-8233
Telex: 287 639 SYBEX UR

SYBEX
6–8, Impasse du Curé
75018 Paris
Tel.: 1/203-95-95
Telex: 211.801 f

SYBEX

Das dBASE II Profibuch

Ein unentbehrliches Arbeitsbuch, das Ihnen viel Zeit ersparen wird, wenn Sie vorhaben, geschäftliche Aufgaben mit dBASE II zu erledigen. Alan Simpson, Autor des SYBEX-Bestsellers „Arbeiten mit dBASE II", bringt Ihnen hier in sinnvoll strukturierter Form fortgeschrittene Programmiertechniken näher.

Aus dem Inhalt:

- **Schulung in Software-Entwicklung**
  So erkennen Sie das Anwendungsproblem. Was muß in der Datenbank gespeichert werden? Unterteilung von großen Projekten in überschaubare Abschnitte. Wie Programm-Routinen erstellt werden.
- **Techniken,** mit denen Sie eine maximale Leistung und Ablaufgeschwindigkeit von dBASE-Programmen erzielen.
- **Grundlegende Algorithmen**
  Sie lernen, selbst Routinen zu erstellen und zu benutzen, mit denen Sie häufig auftretende Probleme lösen können.
- **Programmsysteme für geschäftliche Anwendungen**
  Adreßverwaltung, Lagerhaltung, Buchhaltung u. a.

Über den Autor

Alan Simpson lehrt an der Hochschule über dBASE II und leitet eine dBASE II user group. Außerdem arbeitet er als Berater für dBASE II und andere Sprachen. Erfahrungen aus dieser Tätigkeit veröffentlichte er in zahlreichen Artikeln sowie in **Arbeiten mit dBASE II.**

ISBN 3-88745-610-6

DM 62,–
sFr 57,–
öS 484,–

3610

SYBEX

Das dBASE II Profibuch

Simpson