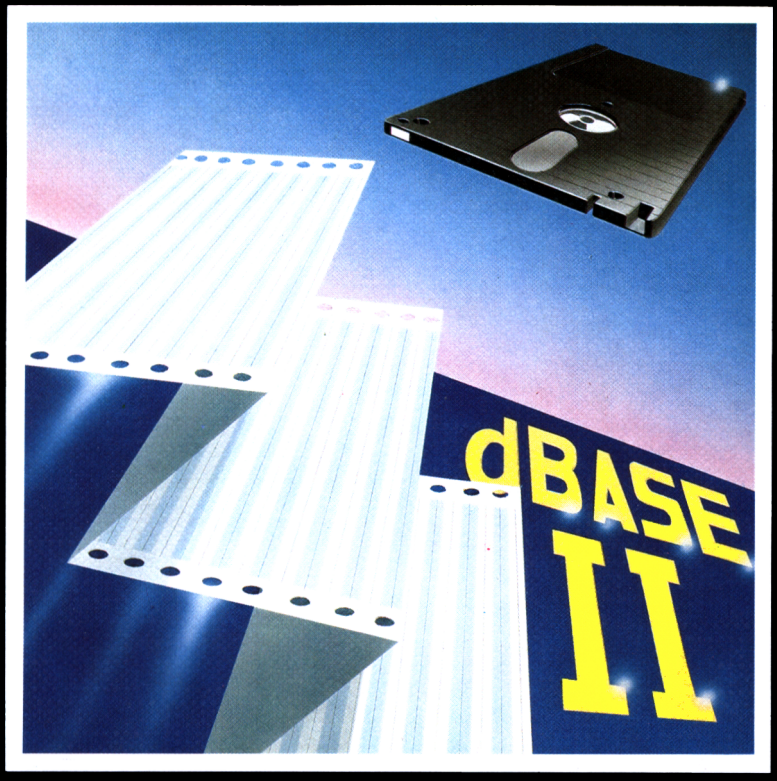
SYBEX

Schneider CPC

Arbeiten mit dBASE II



Michael A. Beisecker

Schneider CPC
Arbeiten mit dBASE II

Schneider CPC
Arbeiten mit dBASE II

Michael A. Beisecker

DÜSSELDORF · BERKELEY · PARIS

Satz: SYBEX-Verlag GmbH, Düsseldorf
Umschlaggestaltung: Daniel Boucherie/tgr
Gesamtherstellung: Boss-Druck und Verlag, Kleve

ISBN 3-88745-660-2
1. Auflage 1986

Inhaltsverzeichnis

Vorwort

dBASE II ist das leistungsfähigste Datenbankprogramm für Heim- und Personalcomputer unter dem Betriebssystem CP/M. Es besitzt Leistungsmerkmale, die noch vor wenigen Jahren nur von Programmen auf Großrechenanlagen geboten wurden. Mit einem einzigen Befehl kann beispielsweise eine Datei in jeder gewünschten Struktur angelegt werden. Sofort danach können Daten eingegeben werden.

Dazu wird eine spezielle Datenbanksprache verwendet, die entweder durch direkte Befehlseingabe oder in Form eines Programms – also einer Zusammenstellung von Befehlen, die hintereinander abgearbeitet werden – eingesetzt wird. Ein dBASE-Programm kann nicht nur zur Verwaltung von Daten, sondern auch zu komplexeren Aufgaben, wie einer Rechnungsschreibung oder gar einer Finanzbuchhaltung verwendet werden.

dBASE-Befehle haben eine größere Mächtigkeit als Befehle in herkömmlichen Programmiersprachen wie etwa BASIC oder Pascal. Daher sind dBASE-Programme auch wesentlich kürzer als Programme in einer dieser Sprachen. Für den dBASE-Anwender bringt dies den Vorteil, in verhältnismäßig kurzer Zeit mit wenigen Befehlen auch umfangreichere Problemstellungen lösen zu können. Allerdings muß der Anwender dazu die Wirkung jedes Befehls genau kennen, um falsche Ergebnisse oder Fehlermeldungen zu vermeiden.

Während die meisten dBASE-II-Bücher amerikanische Übersetzungen sind und somit amerikanische Versionen von dBASE II erläutern, handelt dieses Buch über dBASE II in der deutschen Version, und zwar speziell für den Schneider-Computer. Diese Version unterscheidet sich in vielerlei Hinsicht von dem amerikanischen Ursprungsprogramm.

Sowohl die Benutzerführung als auch die Wirkung einzelner Befehle ist bei den verschiedenen dBASE-Versionen teilweise stark unterschiedlich. Insbesondere für den Anfänger ist es wichtig, daß er Beispiele und Programme direkt am Schneider-Computer üben und die Ergebnisse mit denen im Buch vergleichen kann. Der Anfänger sollte zunächst die Technik der direkten Befehlseingabe perfekt beherrschen – diese wird in Kapitel III "Arbeiten mit dBASE II im interaktiven Befehlsmodus" behandelt –, bevor er mit dem Programmieren – in Kapitel IV "Programmieren mit dBASE II" – beginnt.

Der fortgeschrittene Anwender findet viele Tips und Tricks im Umgang mit dBASE II. Für diesen Leserkreis sind die Hinweise zur strukturierten und normierten Programmierung sowie die Programmier- und Maskenformulare gedacht. Der Verbund von Theorie und Praxis mit vielen Übungen und praxis-

nahen Programmen ermöglicht ein müheloses Erlernen der dBASE-II-Sprache. Mit den Übungen, die sich jeweils am Ende eines Kapitels befinden, kann jeder selbst überprüfen, ob er den behandelten Lehrstoff verstanden hat.

Die Beispiele wurden sowohl aus dem geschäftlichen als auch dem privaten Bereich gewählt. Es sind unter anderem komplette Programmpakete zur Verwaltung von Adressen, Artikeln und zur Rechnungsschreibung in diesem Buch enthalten. Das Ziel des Buches ist aber nicht, daß der Leser diese Programme lediglich in den Computer eingibt und anwendet, sondern daß er selbst gestellte Aufgaben mit Hilfe von dBASE II lösen kann.

Am erfolgreichsten ist die Arbeit mit diesem Buch, wenn alle Kapitel vom Anfang bis zum Ende nacheinander durchgearbeitet werden, da viele Kapitel auf dem Inhalt vorheriger Kapitel aufbauen. Die einzelnen Befehle und Beispiele sollten am Computer nachvollzogen werden. Dabei ist es leicht möglich, die vorgestellten Beispiele und Programme nach eigenen Ideen und Bedürfnissen abzuändern, so daß man während des Erlernens von dBASE II schon seine eigene Datenbank aufbauen kann.

Dieses Buch gliedert sich in vier Teile:

Der erste Teil behandelt die Installation von dBASE II auf dem Schneider-Computer. Dies ist insbesondere für alle diejenigen Leser wichtig, die sich noch nicht so gut mit dem Betriebssystem CP/M auskennen oder dBASE II mit nur einem Diskettenlaufwerk einsetzen wollen. Neben einer genauen Anleitung zur Installation wird dort erklärt, wie man eine Arbeitsdiskette mit einer kleinen dBASE-II-Version herstellen kann, auf der noch sehr viel Speicherplatz für Daten und Programme verbleibt. Außerdem werden die verschiedenen Tastatur- und Zeichensatzbelegungen vorgestellt, die das Arbeiten mit dBASE II stark beeinflussen.

Der zweite Teil enthält die Grundlagen einer relationalen Datenbank und die Leistungsmerkmale von dBASE II. Die zur Vermittlung der Theorie verwendeten Beispiele werden später in Programme umgesetzt, so daß das Studium dieses Kapitels für das Verständnis der nachfolgenden Kapitel wichtig ist.

Im dritten Teil lernt der Leser alle dBASE-II-Befehle kennen, die sich ohne Programm im direkten Befehlsmodus anwenden lassen. Es werden Dateien angelegt, Daten erfaßt, geändert, gelöscht und auf dem Bildschirm oder Drukker ausgegeben. Das Sortieren und Indizieren von Dateien sowie das Rechnen mit dBASE II werden ebenfalls behandelt.

Im vierten Teil wird der Leser in das Programmieren mit dBASE II eingeführt. Dazu werden zusätzliche Befehle und die zur Programmierung notwendigen Techniken vorgestellt. An anspruchsvollen Programmen (Menüaufbau,

Rahmen, Digitaluhr, Druckprogramme, Artikelverwaltung, Adressenverwaltung und Rechnungsschreibung) wird bei steigendem Schwierigkeitsgrad das Programmieren erlernt und die Programmiertechnik verfeinert.

Der Anhang enthält eine Anleitung zur Erstellung von dBASE-II-Programmen mit der Textverarbeitung WordStar, ein alphabetisches Befehlsverzeichnis mit kurzer Erklärung zu jedem Befehl, eine Übersicht der möglichen Tastaturbelegungen, den ASCII-Zeichensatz des Schneider-Computers, die Auflösung der Übungsaufgaben und Formularunterlagen zum Programm- und Maskenentwurf.

Das umfangreiche Stichwortverzeichnis ermöglicht es, gesuchte Textstellen schnellstens zu finden.

Ich möchte an dieser Stelle dem Lektorat des Verlages und insbesondere Herrn Matthias Grafen sowie meiner Ehefrau Marianne für die Durchsicht des Buches danken.

Oberhausen, im September 1986 Michael-Alexander Beisecker

Teil I

Installation von dBASE II

Zwischeninhaltsverzeichnis

Kapitel 1

Installation von dBASE II

In diesem Kapitel werden folgende Themen behandelt:

Erstellen der dBASE-II-Arbeitsdisketten
Wählen der Tastaturbelegung
Wählen des Zeichensatzes
Weitere Einstellmöglichkeiten
Starten von dBASE II
Beenden von dBASE II

Zu diesen Themen gehören die folgenden Befehle:

DBASC
dBASE
DBDIN
DIR
DISCKIT3
|*CPM
LANGUAGE
PIP
QUIT
SETKEYS
SHOW

Nur bei dem letzten Befehl "QUIT" handelt es sich um einen dBASE-II-Befehl, die anderen Befehle gehören zu dem Betriebssystem CP/M Plus.

Erstellen der dBASE-II-Arbeitsdisketten

Bevor wir die Arbeit mit dBASE-II-beginnen, sollten wir uns eine Arbeitsdiskette erstellen. Die dBASE-II-Originaldiskette verwenden wir nur dann, wenn unsere Arbeitsdiskette beschädigt wird. Wir sollten uns zur Sicherheit gleich mehrere Arbeitsdisketten anlegen, wenn wir mit nur einem Diskettenlaufwerk arbeiten, denn auf diesen Disketten sollen, außer den Systemprogrammen,

auch noch unsere selbstgeschriebenen Programme und die Daten Platz finden. Auf einer Seite einer 3-Zoll-Diskette können 180 KByte gespeichert werden. Da in einem KByte 1024 Zeichen Platz finden, lassen sich in 180 KByte etwa 180.000 Zeichen abspeichern. Die dBASE-II-Systemprogramme sind so umfangreich, daß sie nicht mehr alle auf eine Diskettenseite passen. Trotzdem können wir mit einem Diskettenlaufwerk arbeiten, indem mehrere verschiedene Arbeitsdisketten angelegt werden. Besitzen Sie einen Schneider-Computer mit zwei Laufwerken, dann legen Sie in Laufwerk A die dBASE-II-Arbeitsdiskette und in Laufwerk B eine Diskette, auf der nur Daten abgespeichert werden. Wir wollen jetzt gemeinsam die Arbeitsdisketten erstellen. Zunächst schalten wir den Computer ein und legen die CP/M-Plus-Systemdiskette in Laufwerk A.

Um in das CP/M-Betriebssystem zu gelangen, geben wir |cpm <RETURN> ein. Den senkrechten Strich erhalten wir durch gleichzeitiges Drücken der SHIFT- und @-Taste. Das <RETURN> bedeutet, daß die Taste RETURN gedrückt werden soll.

Im folgenden Text werden alle Tasten, die gedrückt werden sollen, auf diese Weise dargestellt. Die Eingaben können wahlweise in Groß- oder Kleinschreibung erfolgen. Einen Fehler können wir mit der DEL-Taste korrigieren. Diese Taste löscht ein links von dem Cursor – das ist der helle viereckige Fleck, der die momentane Bildschirmposition anzeigt – stehendes Zeichen.

Wenn Sie alles richtig gemacht haben, erscheint diese Meldung:

```
CP/M Plus   Amstrad Consumer Electronics plc
v 1.0, 61K TPA, 1 disc drive

A>
```

Das "A>" ist die Bereitschaftsanzeige von CP/M, die man auch als Prompt bezeichnet. Der Prompt ist außerordentlich wichtig, denn wir erkennen durch ihn, in welchem Programm wir uns befinden und mit welchem Diskettenlaufwerk wir arbeiten. Das "A" steht für das Diskettenlaufwerk A.

Besitzen wir zwei Laufwerke, so können wir durch die Eingabe von B: <RETURN> das zweite Laufwerk aktivieren und erhalten dann den Prompt "B>". Durch die Eingabe von A: <RETURN> kehren wir wieder in den Ausgangszustand zurück. Wir lassen uns jetzt das Inhaltsverzeichnis der CP/M-Diskette anzeigen.

Dazu geben wir DIR <RETURN> hinter dem Prompt ein. Alle Dateien, die auf "COM" enden, sind Programme. Das Programm DISCKIT3 werden wir

jetzt benutzen, um Disketten zu formatieren. Wir starten das Programm mit der Eingabe DISCKIT3 <RETURN> und erhalten folgendes Menü:

```
Copy                    7
Format                  4
Verify                  1
Exit from program       0
```

Die einzelnen Funktionen werden über die Funktionstasten angewählt. Zum Formatieren drücken wir <f4> und erhalten ein neues Menü:

```
System format           9
Data format             6
Vendor format           3
Exit menu               .
```

Wir wählen mit <f9> die Funktion "System format" aus. Nun erscheint folgende Meldung:

```
About to read reserved tracks
Insert a System disc
Press any key to continue
```

(Da reservierte Spuren gelesen werden müssen, legen Sie bitte die Betriebssystemdiskette ein und drücken eine beliebige Taste.)

Wir lassen die Systemdiskette im Laufwerk und drücken irgendeine Taste. Der Computer fordert uns jetzt auf, die Diskette aus dem Laufwerk zu nehmen:

```
Remove disc
Press any key to continue
```

Nachdem wir die CP/M-Diskette herausgenommen haben, drücken wir wieder eine Taste und beantworten die nächste Frage mit dem Drücken der Taste Y. Jetzt werden wir gebeten, die neue Diskette einzulegen, die wir formatieren möchten:

```
Insert disc to format
Press any key to continue
```

Das Formatieren einer bespielten Diskette löscht alle Dateien. Wenn Sie ganz sicher sind, daß eine leere Diskette – auf keinen Fall die CP/M- oder die dBASE-II-Originaldiskette – in dem Laufwerk liegt, drücken Sie eine Taste. Es vergehen jetzt etwa dreißig Sekunden, danach zeigt die folgende Meldung an, daß Sie die Diskette wieder herausnehmen können.

```
Format completed
Remove disc
Press any key to continue
```

Wir nehmen die Diskette aus dem Laufwerk und drücken eine Taste. Wenn wir weitere Disketten formatieren möchten, geben wir zur Bestätigung der

folgenden Frage die Taste Y ein

```
Y Format another as System
      Any other key to exit menu
```

Die 3-Zoll-Disketten lassen sich auch auf der Rückseite formatieren, so daß wir auf beiden Seiten jeweils 180 KByte speichern können. Wenn wir genügend Disketten formatiert haben, geben wir an dieser Stelle kein Y ein, sondern drücken eine andere Taste und gelangen wieder in das Ausgangsmenü:

```
Copy                    7
Format                  4
Verify                  1
Exit from program       0
```

Aus diesem Menü kehren wir mit <f0> in das CP/M-Betriebssystem zurück. Nachdem wir nun formatierte Disketten besitzen, können wir die benötigten Programme von der CP/M-Diskette und der dBASE-II-Diskette kopieren. Dazu nehmen wir das Programm PIP. Wir geben PIP <RETURN> ein und erhalten folgende Meldung:

```
CP/M 3 PIP VERSION 3.0
*
```

Bei dem Programm PIP ist das Sternchen der Prompt. Hinter dem Sternchen können wir Befehle eingeben. Die erste Datei, die wir kopieren wollen, heißt C10CPM3.EMS. Sie ermöglicht es, direkt von der Arbeitsdiskette aus das CP/M-Betriebssystem zu starten. Der Befehl zum Kopieren der Datei lautet:

```
B:=A:C10CPM3.EMS <RETURN>
```

Die Datei wird zunächst in den Arbeitsspeicher geladen. Dann werden wir aufgefordert, die Arbeitsdiskette in Laufwerk B einzulegen:

```
Please put the disc for B: into the drive then press any key
```

Diese Meldung erscheint auch, wenn unser Computer mit nur einem Laufwerk ausgestattet ist. Das Betriebssystem arbeitet in diesem Fall mit einem virtuellen Laufwerk im Hauptspeicher, das genau wie ein zweites Laufwerk B heißt. In diesem Fall wechseln wir die CP/M-Diskette in Laufwerk A durch die neue Diskette aus. Das Sternchen in der nächsten Zeile teilt uns mit, daß wir einen neuen Befehl eingeben können. Dazu muß die CP/M-Diskette in Laufwerk A liegen. Wir benötigen noch einige Dateien, die wir mit folgenden Befehlen kopieren:

```
B:=A:SETKEYS.COM <RETURN>
B:=A:LANGUAGE.COM <RETURN>
B:=A:SUBMIT.COM <RETURN>
```

Mit diesen Programmen können wir den deutschen Zeichensatz auf der Tastatur einstellen und die Funktionstasten belegen. Wenn alle Programme kopiert sind, legen wir anstelle der CP/M-Diskette die dBASE-II-Originaldiskette in das Laufwerk A und geben folgenden Befehl ein:

```
B:=A:*.*
```

Dieser Befehl kopiert alle Dateien von der dBASE II-Diskette auf die Arbeitsdiskette. Wenn nur ein Diskettenlaufwerk vorhanden ist, müssen wir auf die Meldungen achten und die Disketten im Laufwerk A entsprechend wechseln. Auf der Rückseite der dBASE-II-Diskette befinden sich Dateien zur Anpassung von dBASE II an unterschiedliche Rechnertypen sowie der Maskengenerator ZIP.

Diese Programme sollten wir auf eine andere unserer formatierten Disketten kopieren. Dazu legen wir die dBASE-Diskette mit dem Etikett nach unten in das Laufwerk A und wenden den oben stehenden Befehl noch einmal an. Die Installationsprogramme benötigen wir normalerweise nicht, denn diese dBASE-II-Version ist bereits für den Schneider-Computer angepaßt. Wir sollten jetzt überprüfen, wieviel Platz auf der Arbeitsdiskette für Daten und Programme geblieben ist. Dazu legen wir die CP/M-Diskette in Laufwerk A und die Arbeitsdiskette in Laufwerk B und geben SHOW B: <RETURN> hinter dem Prompt ein. Bei nur einem Diskettenlaufwerk müssen wir wieder die Disketten wechseln. Das Ergebnis lautet:

```
B: RW, Space:          8k
```

Von den anfänglichen 180 KByte Speicherplatz sind uns ganze 8 KByte geblieben. Daher müssen wir uns eine zweite, kleinere Arbeitsdiskette erstellen, um mit einem Diskettenlaufwerk arbeiten zu können. Sehen wir uns an, wieviel Platz die einzelnen Dateien auf unserer Arbeitsdiskette belegen und wozu sie gebraucht werden. Wir verwenden dazu wieder den DIR-Befehl und geben DIR [FULL] B: ein. Die CP/M-Diskette muß wieder im Laufwerk A liegen. Es erscheint folgende Meldung:

```
A>DIR [FULL] B:

Scanning Directory...

Sorting Directory...

Directory For Drive B:  User  0

Name      Bytes   Recs   Attributes      Name      Bytes   Recs   Attributes
--------- ------ ------ ------------ ------------ ------ ------ ----------
C10CPM3   EMS     25k     200 Dir RW    DBASC     KEY      2k      12 Dir RW
DBASC     SUB      1k       1 Dir RW    dBASE     COM     20k     158 Dir RW
dBASEMSG  TXT     60k     475 Dir RW    dBASEOVR  COM     41k     328 Dir RW
DBDIN     KEY      2k      12 Dir RW    DBDIN     SUB      1k       1 Dir RW
LANGUAGE  COM      1k       8 Dir RW    SETKEYS   COM      2k      16 Dir RW
```

```
SUBMIT   COM      6k      42 Dir RW

Total Bytes     =    161k  Total Records =    1253  Files Found =    11
Total 1k Blocks =    161   Used/Max Dir Entries For Drive B:   18/   64
```

Hilfstexte

Um mit dBASE II arbeiten zu können, benötigen wir nur die Programme dBASE.COM und dBASEOVR.COM, die zusammen 61 KByte belegen. Dazu gehört die Datei dBASEMSG.TXT, die alle dBASE-II-Hilfsmenüs enthält. Wir können auf die Hilfstexte verzichten und damit sehr viel Speicherplatz für Daten und Programme sparen. Dazu legen wir die CP/M-Diskette in Laufwerk A und rufen mit PIP <RETURN> das Kopierprogramm auf. Dann kopieren wir die beiden Programme mit dem Befehl:

```
B:=A:dBASE*.COM
```

Wählen des Zeichensatzes

Um in das dBASE-System zu gelangen, legen wir die vollständige Arbeitsdiskette in Laufwerk A und starten mit |CPM <RETURN> das Betriebssystem, wenn wir uns nicht schon darin befinden. Dann können wir mit dem Programm LANGUAGE zwischen dem DIN- und dem ASCII-Zeichensatz wählen. Der DIN-Zeichensatz ermöglicht die Darstellung von Umlauten und ß auf dem Bildschirm, dafür lassen sich allerdings einige Zeichen nicht mehr darstellen:

ASCII-Zeichen	DIN-Zeichen
@	@
[	Ä
]	Ü
{	ä
}	ü
\	Ö
≥	ö
~	ß

Unser Schneider-Computer ist beim Einschalten automatisch im ASCII-Zeichensatz. Die Umschaltung erfolgt durch die Eingabe von LANGUAGE 2 <RETURN> für den DIN- bzw. LANGUAGE 0 <RETURN> für den ASCII-Zeichensatz. Dazu muß die Tastaturbelegung mit dem SETKEYS-Programm passend eingestellt werden.

Wählen der Tastaturbelegung

Die DIN-Tastatur wird mit der Befehlszeile

```
SETKEYS DBDIN.KEY <RETURN>
```

eingestellt und die ASCII-Tastatur durch die Eingabe von

```
SETKEYS DBASC.KEY <RETURN>
```

installiert. Im Anhang sind die möglichen Tastaturbelegungen graphisch dargestellt. Diese Eingaben können auch durch den Aufruf DBDIN.SUB <RETURN> zur Einstellung von DIN-Tastatur und DIN-Zeichensatz sowie DBASC.SUB <RETURN> für die Kombination ASCII-Tastatur und ASCII-Zeichensatz ersetzt werden.

Starten von dBASE II

Nachdem wir dies ausgiebig geübt haben, starten wir dBASE II zum ersten Mal. Dazu gibt man

```
dBASE <RETURN>
```

ein. dBASE II fragt dann nach dem Tagesdatum:

```
Tagesdatum eingeben oder RETURN falls nicht benötigt
(DD/MM/YY) :
```

Im Gegensatz zu dem Betriebssystem CP/M gibt dBASE II deutsche Meldungen aus. Die Abkürzungen "DD", "MM" und "YY" stehen allerdings für die englischen Wörter "day", "month" und "year", die übersetzt "Tag", "Monat" und "Jahr" bedeuten. Wenn wir heute den 1. September 1986 hätten, würden wir also 01/09/86 <RETURN> eingeben.

Anstelle des Schrägstrichs können wir auch einen Punkt zur Trennung verwenden. Wir sollten immer das aktuelle Tagesdatum eingeben. Dies ist sehr hilfreich, wie wir später sehen werden. Nachdem wir das Datum eingegeben haben, erscheint die Copyright-Meldung von Ashton Tate auf dem Bildschirm, die wir nicht weiter beachten müssen. Viel wichtiger für uns ist der kleine Punkt in der letzten Zeile. Dieser Punkt ist die Bereitschaftsanzeige von dBASE II, also der Prompt, hinter dem wir dBASE-Befehle eingeben können. Fürs erste haben wir jetzt genug gearbeitet. Daher verlassen wir dBASE II.

Beenden von dBASE II

Dazu geben wir hinter dem Prompt QUIT <RETURN> ein. Wir sollten den Computer nicht einfach ausschalten, ohne dBASE ordnungsgemäß verlassen zu haben, da sonst Daten verloren gehen könnten. Bevor wir den Schneider-Computer ausschalten, nehmen wir die Diskette aus dem Laufwerk.

Übung 1:

1. Was ist ein Prompt?
2. Welche Prompts wurden in diesem Kapitel behandelt?
3. Zu welchen Programmen gehören sie?
4. Mit welchem Programm formatieren wir Disketten?
5. Welches Programm haben wir benutzt, um Dateien zu kopieren?
6. Wozu dient das Programm LANGUAGE.COM?
7. Wozu dient das Programm SETKEYS.COM?
8. Wie rufen wir dBASE II auf?
9. Mit welchem Befehl verlassen wir dBASE II?

Teil II

Das relationale Datenbankmodell

Zwischeninhaltsverzeichnis

Kapitel 1

Allgemeiner Aufbau einer relationalen Datenbank

Grundlagen

In diesem Kapitel werden die Grundlagen einer relationalen Datenbank dargestellt. Diese sind zwar nicht zwingend notwendig, um mit dBASE II zu arbeiten, die Kenntnis von den Vorgängen in einer relationalen Datenbank kann jedoch diese Arbeit effizienter gestalten.

Wir werden einigen Begriffen aus der EDV-Fachsprache (EDV=Elektronische Daten-Verarbeitung) begegnen, die zum Grundverständnis des Aufbaus und der Funktion einer Datenbank benötigt werden. Eine Datenbank ist ein Gebilde aus mehreren Dateien. Das Wort Datei ist eine Synthese aus den Wörtern Daten und Kartei. Daten können Informationen aller Art sein. Eine Datei ist eine Sammlung von Daten nach bestimmten Kriterien.

Um deren Aufbau und Funktion zu verstehen, sehen wir uns ein Beispiel an. Eine Datei läßt sich mit einer Kartei vergleichen. Stellen wir uns einen Karteikasten zur Verwaltung von Adressen vor:

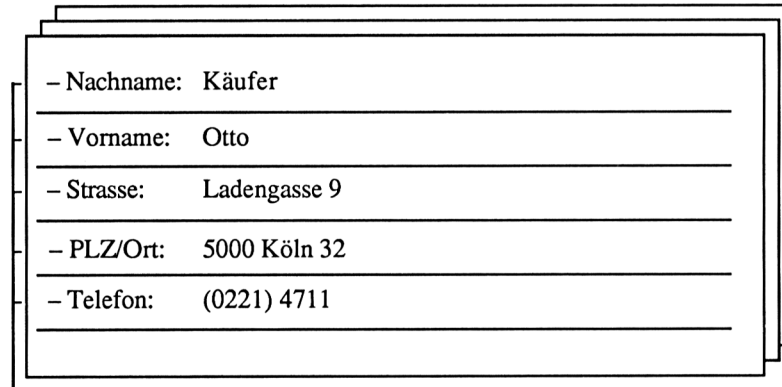
Nachname:	Käufer
Vorname:	Otto
Strasse:	Ladengasse 9
PLZ/Ort:	5000 Köln 32
Telefon:	(0221) 4711

Abb. 1.1: Beispiel einer Karteikarte

Eine Karteikarte enthält eine Anschrift, oder anders ausgedrückt die Daten einer Person. Diese Daten sind die Angaben von Name, Vorname, Strasse, Postleitzahl, Ort und Telefonnummer.

Zur Unterscheidung der verschiedenen Daten ordnet man diese in sogenannte Datenfelder ein.

In diesem Beispiel lauten die Datenfelder Nachname, Vorname, Strasse, Postleitzahl, Ort und Telefon. Diese Unterteilung wurde der Karteikarte entnommen.



Felder	Daten	Feldbeschreibung
– Nachname:	Käufer	alphanumerisch, max. 15
– Vorname:	Otto	alphanumerisch, max. 10
– Strasse:	Ladengasse 9	alphanumerisch, max. 15
– Postleitzahl:	5000	numerisch, max. 4
– Ort:	Köln 32	alphanumerisch, max. 12
– Telefon:	(0221) 4711	alphanumerisch, max. 12

Abb. 1.2: Karteikarte und Datenspeicherung im Rechner

Datenspeicherung im Rechner

Ein Computer speichert Adressen nicht auf Karteikarten, sondern er legt eine Datei an. Der Aufbau dieser Datei wird durch die Felder bestimmt. Für jedes Feld muß dazu eine maximale Länge und ein Feldtyp festgelegt werden. Der Feldtyp bestimmt, welcher Typ von Daten in einem Feld abgelegt wird. Dies können numerische Daten (Zahlen), oder alphanumerische Daten (Buchstaben, Ziffern und Sonderzeichen) sein. Einen Sonderfall stellen die Felder dar, welche nur die Werte Wahr oder Unwahr (engl. true bzw. false) enthalten dürfen und logische, bzw. boolesche Felder genannt werden.

Die Daten einer Adresse sind in einem Datensatz gespeichert. Für jede Adresse hängt der Computer einen Datensatz an die Datei an. Somit besteht eine Datei aus einer Folge von Sätzen.

Satznr	Nachname	Vorname	Strasse	PLZ	Ort	Telefon
00001	Käufer	Otto	Ladengasse 9	5000	Köln 32	(0221) 4711
00002	Meyer	Theo	Eichenstr. 15	4100	Duisburg	(0203) 5612
00003	Maier	Udo	Kolpingstr. 14a	6050	Offenbach	(069) 3467
00004	Maiers	Trude	Bahnhofstr. 11	4300	Essen-West	(0201) 2031
00005	Hütter	Angelika	Sandfuhrstr. 34	4200	Oberhausen	(0208) 5612
00006	Hutter	Sabine	Heideweg 23	6790	Oberammergau	(0122)4554
00007	Ende	Werner	Am Schlußlicht	5428	Endlichhofen	(07142) 568

Abb. 1.3: Datensätze in einer Datei

Da die Summe aller Feldlängen die Länge eines Datensatzes ergibt, läßt sich die Größe einer Datei leicht bestimmen, indem die Anzahl aller Datensätze mit deren Länge multipliziert wird.

Die Begriffe Datei und Datenbank werden häufig verwechselt. Eine Datenbank besteht aus mehreren Dateien. Es handelt sich dabei um Dateien, in denen Daten gespeichert sind, und um solche, die Indizes enthalten und daher Indexdateien genannt werden. Bisher haben wir nur die Datendateien besprochen. Um die Funktion einer Indexdatei zu verstehen, greifen wir wieder das Beispiel mit dem Karteikasten auf.

Sortierung

Eine Adressendatei ist in der Regel nach Nachnamen sortiert. Wenn eine andere Sortierung benötigt wird, zum Beispiel nach Postleitzahlen, so müssen wir alle Karteikarten noch einmal schreiben und von Hand nach Postleitzahlen in einen zweiten Karteikasten sortieren.

Dieses Verfahren hat zwei entscheidende Nachteile:

1. Die Kartei ist jetzt doppelt angelegt und benötigt somit den doppelten "Speicherplatz" (zwei Karteikästen).
2. Das Sortieren geschieht recht langsam.

Einen Sortiervorgang können wir auch auf dem Computer durchführen. Die Datensätze aus der alten Datei werden sortiert in eine neue Datei geschrieben. Obwohl ein Computer diese Aufgabe wesentlich schneller erfüllt als ein Mensch, bleiben die Nachteile des Sortierens auch auf einem Computer bestehen.

Eleganter und schneller wird diese Aufgabe mit Hilfe von Indexdateien gelöst. Anstelle verschieden sortierter Datendateien gibt es nur eine Datendatei und mehrere Indexdateien. Diese Dateien enthalten das jeweilige Sortierfeld, also in unserem Beispiel den Nachnamen oder die Postleitzahl, und die zugehörigen Satznummern in sortierter Form. Jeder Satz in der Datendatei ist mit einer Satznummer versehen, über die man direkt auf diesen Satz zugreifen kann (Abb. 1.4).

Datei Kunden

Satznr	Nachname	Vorname	Strasse	PLZ	Ort	Telefon
00001	Käufer	Otto	Ladengasse 9	5000	Köln 32	(0221) 4711
00002	Meyer	Theo	Eichenstr. 15	4100	Duisburg	(0203) 5612
00003	Maier	Udo	Kolpingstr. 14a	6050	Offenbach	(069) 3467
00004	Maiers	Trude	Bahnhofstr. 11	4300	Essen-West	(0201) 2031
00005	Hütter	Angelika	Sandfuhrstr. 34	4200	Oberhausen	(0208) 5612
00006	Hutter	Sabine	Heideweg 23	6790	Oberammergau	(0122)4554
00007	Ende	Werner	Am Schlußlicht	5428	Endlichhofen	(07142) 568

Indexdatei Nachnamen

Satznr Nachname

00007	Ende
00005	Hütter
00006	Hutter
00001	Käufer
00002	Meyer
00003	Maier
00004	Maiers

Indexdatei PLZ

Satznr PLZ

00002	4100
00005	4200
00004	4300
00001	5000
00007	5428
00003	6050
00006	6790

Abb. 1.4: Datendatei und Indexdateien

Die Länge eines Indexdateisatzes ergibt sich aus der Addition der Feldlänge des Sortierfeldes zu der Länge der Satznummer und der Anzahl der systembedingten Steuerzeichen. Daher kann die Größe einer Indexdatei durch die Multiplikation der errechneten Satzlänge mit der Anzahl der Schlüssel (Anzahl der Sätze in der Datendatei) ermittelt werden.

Dateizugriff über Schlüsselfelder

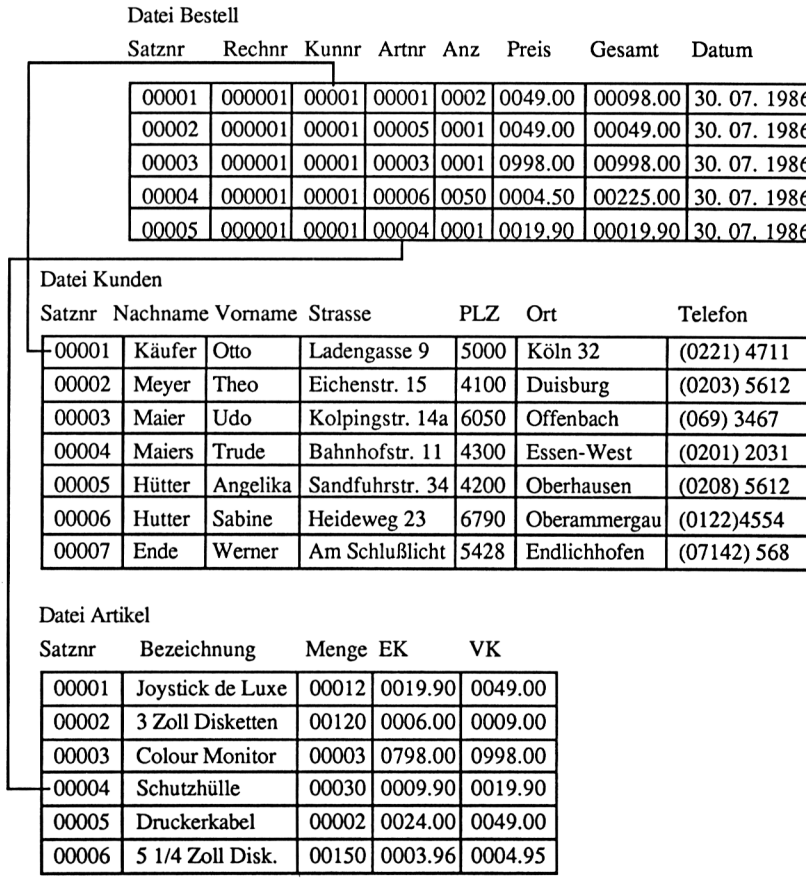


Abb. 1.5: Dateienzugriff über Schlüsselfelder

Wie aus Abb. 1.4 ersichtlich ist, werden die Daten in Form einer zweidimensionalen Tabelle abgespeichert. Eine solche Tabelle bezeichnet man auch als

Relation. Die Zeilen der Tabelle entsprechen den Datensätzen und die Spalten den Datenfeldern. Über ein Schlüsselfeld kann von dieser Tabelle auf eine andere Tabelle, d.h. auf eine andere Datei, zugegriffen werden.

In Abb. 1.5 ist dafür ein Beispiel dargestellt. Das Schlüsselfeld muß sowohl in der Ausgangsdatei als auch in der zu verknüpfenden Datei vorhanden sein. In diesem Beispiel ist eine Datenbank zur Rechnungsschreibung dargestellt. Die Datenbank enthält drei Datendateien:

1. Kunden
2. Bestellungen
3. Artikel

Die Schlüsselfelder sind die Kundennummer (Kunnr) und die Auftragsnummer (Artnr). Hier wird die jeweilige Satznummer der Datei Kunden und Artikel als Kunden- bzw. Artikelnummer verwendet. Durch diesen kleinen Trick spart man ein entsprechendes Feld in diesen Dateien. Damit eine Rechnung erstellt werden kann, wird eine Verbindung über die Kundennummer von der Datei Bestellungen zu der Datei Kunden aufgebaut. So wird die Kundenadresse gefunden. Diese Adresse erscheint im Briefkopf der Rechnung.

Die Datei Bestellungen enthält außerdem die Artikelnummer und die Anzahl des bestellten Artikels. Über die Artikelnummer wird eine Verbindung zu der Artikeldatei aufgebaut. Mit den dort gefundenen Informationen (Artikelbezeichnung, Preis, Lagerbestand) sind alle für die Rechnung notwendigen Daten vollständig. Ein entsprechendes Rechnungsdruckprogramm erzeugt mit diesen Angaben beispielsweise eine solche Rechnung:

```
Otto Käufer
Ladengasse 9

5000 Köln 32



Ihre Kundennummer:     1        Ihre Bestellung vom: 30.07.1986
Rechnungsnummer  :     1        Rechnungsdatum vom :  1.08.1986


ARTNR. ARTIKELBEZEICHNUNG MENGE    PREIS             GESAMT
-------------------------------------------------------------
    1  Joystick de Luxe       2      49.00 DM          98.00 DM
    5  Druckerkabel           1      49.00 DM          49.00 DM
    3  Colour Monitor         1     998.00 DM         998.00 DM
    6  5 1/4 Zoll Disk.      50       4.95 DM         247.50 DM
    4  Schutzhuelle           1      19.90 DM          19.90 DM
                                                     ----------
                                                     1412.40 DM

                      In diesem Preis sind 14% MwSt enthalten.
Zahlbar sofort netto Kasse.
```

Planen einer Datenbank

Bei der Planung einer Datenbank ist darauf zu achten, daß außer den Schlüsselfeldern keine Informationen in den Dateien mehrfach vorhanden sind. Wenn die gesamte Adresse des Kunden und dazu alle Informationen über den bestellten Artikel in der Bestelldatei gespeichert wären, so wären diese Daten redundant, da sie ebenfalls in der Datei Kunden bzw. Artikel enthalten sind. Dies würde zu einem erheblichen Mehraufwand an Erfassung und Pflege der Daten führen. Es gibt aber auch Fälle, bei denen eine Mehrfachspeicherung sinnvoll und notwendig ist. Die Datei Bestellung enthält ein Feld Preis, obwohl ein entsprechendes Feld auch in der Datei Artikel vorhanden ist. In diesem Feld kann ein abweichender Preis gespeichert werden, für den Fall, daß ein Kunde einen Sonderpreis oder einen Rabatt erhält.

Schlüsselfelder

Die Kundennummer und die Artikelnummer sind reine Schlüsselfelder, die nur dazu dienen, eine Verbindung zu einer anderen Datei zu ermöglichen. Man könnte eine solche Verbindung auch über den Namen eines Kunden bzw. die Artikelbezeichnung aufbauen. Diese Schlüssel sind allerdings länger als die entsprechenden Nummern und außerdem nicht eindeutig. Denken Sie nur an die große Anzahl von Menschen in Deutschland, die Schmidt oder Maier heißen. Damit keine Verwechslung auftreten kann, müßte der Schlüssel um den Vornamen des Kunden und die Postleitzahl des Ortes, in dem er wohnt, erweitert werden. Dies würde die Indexdateien entsprechend anschwellen lassen und die Redundanz von Daten fördern, da die Schlüsselfelder in allen zu verknüpfenden Dateien vorhanden sein müssen.

In der Tat ist es aber möglich, nicht nur ein Schlüsselfeld zur Indizierung einer Datei zu verwenden, sondern einen Schlüssel aus mehreren Feldern zu bilden. Nimmt man beispielsweise die Aneinanderreihung der Felder Nachname und Vorname als Schlüssel, so erhält man eine Datei, bei der innerhalb gleicher Nachnamen die Datensätze nach den Vornamen sortiert sind.

Aufgaben einer Datenbank

Nachdem wir nun wissen, wie eine Datenbank aufgebaut ist, sollten wir uns ihren Aufgaben zuwenden. Mit einer Datenbank werden Dateien verwaltet. In jeder Datei können wir Datensätze erfassen, ansehen, ändern und löschen. Die Dateien können miteinander verbunden werden, um Daten zu verknüpfen. Es kann nach jedem Feldinhalt eines Datensatzes innerhalb einer Datenbank gesucht werden.

Sucht man etwa den Namen eines Bekannten und hat den Nachnamen vergessen, weiß aber noch, daß er in Frankfurt wohnt und sein Vorname Paul ist, so kann man sich alle gespeicherten Adressen der Personen anzeigen lassen, die in Frankfurt wohnen und Paul heißen.

Damit in einer Datenbank mit der Suche begonnen wird, müssen in einer speziellen Sprache mehrere Befehle erteilt werden. Ein Datenbankprogramm versteht in der Regel zwei Sprachen.

Die eine Sprache ist relativ einfach und für den Nur-Anwender gedacht. Die ser kann mit ihrer Hilfe Daten erfassen, suchen, ändern und löschen. Zur Programmierung wird dagegen meist eine ganz andere oder zumindest stark erweiterte Sprache verwendet. Für solche Aufgaben ist ein Programmierer zuständig.

Diese Trennung resultiert daraus, daß Datenbanksysteme früher nur auf Großrechnern eingesetzt wurden. Dort eingesetzte Systeme sind aufgrund ihrer größeren Leistung wesentlich komplexer als Datenbanken auf einem Personalcomputer. Hier sind der Anwender und der Programmierer häufig dieselbe Person.

Daher sind Datenbanksysteme für den Personalcomputer so konzipiert, daß sie auch ein Nichtprogrammierer bedienen und sogar programmieren kann. Selbst wenn man nicht programmiert, ist man schon mit wenigen Befehlen in der Lage, die wesentlichen Möglichkeiten einer Datenbank ausnutzen zu können.

dBASE II ist ein solches relationales Datenbanksystem. Im nächsten Kapitel wird gezeigt, welche Leistungsfähigkeit dieses System hat.

Kapitel 2

Möglichkeiten von dBASE II

Nachdem die Grundlagen einer relationalen Datenbank allgemein behandelt wurden, können wir konkret auf dBASE II eingehen. dBASE II besteht aus einer kombinierten Datenbank-Anwender- und Programmiersprache. Die Befehle kann der Anwender entweder direkt oder in Form eines Programms eingeben.

Der interaktive Befehlsmodus

Mit einem einzigen Befehl kann man etwa eine Datei erstellen und Daten erfassen. Alle wichtigen Funktionen der Datenbankverwaltung, wie Datensätze erfassen, suchen, ändern oder löschen, gibt man im sogenannten "Interaktiven Befehlsmodus" ein.

In diesem Modus erfolgt auf jede Eingabe des Anwenders eine direkte Reaktion des Systems. Etwaige Syntax-Fehler werden erkannt und zur Verbesserung angeboten. Vergißt der Anwender einen Befehl, kann er eine Hilfsfunktion aufrufen.

Der Programmiermodus

Der andere Modus ist der Programmiermodus. Hier werden die Befehle zunächst mit einem Texteditor in eine Programmdatei geschrieben, um dann später hintereinander abgearbeitet zu werden.

Mit Hilfe des Programmiermodus kann man häufig benötigte Befehlsfolgen automatisieren. Im Programmiermodus können sogar komplexe Programme wie eine Auftragsverwaltung, eine Rechnungsschreibung oder gar eine Buchhaltung erstellt werden.

Während sich das Arbeiten im interaktiven Befehlsmodus recht schnell erlernen läßt, benötigt das Beherrschen der Programmierung schon etwas mehr Zeit. Der Programmierer muß mehr Befehle kennen als der reine Anwender. Er muß die Reaktion von dBASE II auf alle Befehle im voraus berechnen und

in die Planung seines Programms mit einbeziehen. Für die allermeisten Anwendungen reicht der Befehlsmodus vollständig aus. Somit muß man nicht programmieren können, um Daten zu verwalten.

Die Leistungsfähigkeit des Datenbanksystems dBASE II ist so groß, daß sie nur von der Leistung des Rechners begrenzt wird, auf dem man es einsetzt.

Größe einer Datei

So können beispielsweise maximal 65.535 Datensätze in einer Datei verwaltet werden. Jeder Datensatz darf maximal 1000 Zeichen lang sein. Daraus folgt, daß eine dBASE-II-Datei

1000 x 65.535 = 65.535.000

Zeichen groß sein darf. Auf einer 3-Zoll-Diskette finden aber maximal 170 KBytes, das heißt 180 x 1024 = 184.320 Zeichen Platz. Wir würden 356 Disketten benötigen, um diese Datei abzuspeichern. Jeder Datensatz darf aus bis zu 32 Feldern bestehen, was für die Praxis meist völlig ausreichend ist. In einem Feld dürfen maximal 254 Zeichen stehen.

Grenzen von dBASE II

Die Grenzen von dBASE II bemerkt man (wenn überhaupt) nur als fortgeschrittener Programmierer. So kann man beispielsweise maximal zwei Datenbanken gleichzeitig ansprechen. Der Bildschirmaufbau ist relativ langsam. Es dürfen maximal 64 Variablen zur gleichen Zeit existieren. Die Rechengenauigkeit ist auf 10 Stellen begrenzt. Außer den vier Grundrechenarten lassen sich mit dBASE II keine Rechenoperationen durchführen. Einige dieser Einschränkungen lassen sich umgehen, indem man die Dateiverwaltung mit dBASE II erledigt und Rechnungen mit Hilfe einer Programmiersprache wie BASIC oder eines Kalkulationsprogramms durchführt. Eine dBASE-II-Datei kann in ein sogenanntes Standardformat überführt werden. Diese Datei kann dann von anderen Programmen verarbeitet werden. Auf dem umgekehrten Weg kann dBASE II auch Daten von anderen Programmen übernehmen.

Der Texteditor

Der eingebaute Texteditor ist auf ca. 4000 Zeichen begrenzt. Bei längeren Programmen kann es vorkommen, daß man unbemerkt einen Teil des Programms verliert. Zudem ist dieser Editor recht unkomfortabel, wenn man ihn

zum Beispiel mit dem Editor von Turbo Pascal vergleicht. Um größere Programme zu schreiben, sollte man eine Textverarbeitung wie WordStar oder den erwähnten Turbo-Pascal-Editor verwenden.

Teil III

Arbeiten mit dBASE II im interaktiven Befehlsmodus

Zwischeninhaltsverzeichnis

Kapitel 1

Erstellen einer Dateistruktur

In diesem Kapitel werden folgende Themen behandelt:

Anlegen einer Dateistruktur
Anzeigen einer Dateistruktur

Zu diesen Themen gehören die folgenden Befehle:

CREATE
DISPLAY STRUCTURE

Falls wir uns nicht im dBASE-II-Programm befinden, müssen wir jetzt unsere Arbeitsdiskette in Laufwerk A einlegen und dBASE <RETURN> eingeben. Wer nicht mehr genau weiß, wie er in das CP/M-Betriebssystem oder dBASE II gelangt, sollte unbedingt noch einmal in das Kapitel "Installation von dBASE II" schauen.

Anlegen einer Dateistruktur

Wenn wir jetzt in dBASE II sind und das Tagesdatum eingegeben haben, können wir mit dem Befehl CREATE unsere erste Datei anlegen. Dazu geben wir CREATE <RETURN> direkt hinter dem Prompt, das ist bei dBASE II der Punkt, ein. Da wir dBASE II nicht mitgeteilt haben, wie wir die Datei nennen möchten, fordert es uns auf, einen Namen einzugeben:

```
Bitte Dateinamen eingeben:
```

Ein Dateiname darf maximal acht alphanumerische Zeichen lang sein. Es dürfen auch die Umlaute und das ß verwendet werden. Mit dem DIR-Befehl können wir allerdings feststellen, daß das "ß" im Inhaltsverzeichnis der Diskette als ↑ abgespeichert wird. Außerdem hängt dBASE II an diesen Namen die Erweiterung ".DBF" an. Als Beispiel legen wir eine Datei an, in der wir die Adressen von Kunden abspeichern können. Als Namen geben wir einfach

KUNDEN <RETURN> ein. Vor dem Namen können wir das Diskettenlaufwerk angeben, auf dem die Datei gespeichert werden soll. Arbeiten wir mit zwei Diskettenlaufwerken, geben wir also B:KUNDEN <RETURN> ein, da unsere Datendiskette im zweiten Laufwerk liegt. Im folgenden Text wird das B: nicht extra bei jedem Dateinamen angegeben. Wir müssen daran denken, wenn wir mit zwei Laufwerken arbeiten. Nun zeigt uns dBASE II, wie die Dateistruktur einzugeben ist:

```
Satzstruktur folgendermaßen eingeben:
Feld    Name, Typ, Länge,  Dezimalstellen
001
```

Felder

Aus dem vorangegangenen Kapitel wissen wir, daß eine Datei aus Datensätzen besteht. In jedem Datensatz können wir die Informationen über genau einen Kunden speichern. Ein Datensatz setzt sich aus Datenfeldern zusammen, in unserem Beispiel aus den Feldern:

Kundennummer
Nachname
Vorname
Strasse
Postleitzahl
Ort
Telefonnummer
Umsatz

Für jedes dieser Felder müssen wir dBASE II einen Namen, einen Feldtyp, die Feldlänge und bei numerischen Feldern die Anzahl der Nachkommastellen angeben. Diese Angaben werden durch Kommata getrennt und mit der Eingabe von <RETURN> abgeschlossen. Der Feldname darf maximal zehn alphanumerische Zeichen lang sein und außer dem ":" keine Sonderzeichen, Umlaute oder das "ß" enthalten.

Feldtyp

Die Festlegung, ob ein Feld alphanumerische oder numerische Werte enthalten soll, treffen wir über den Feldtyp. Es stehen folgende Feldtypen zur Auswahl:

C (engl. character)	Für alphanumerische Daten (A-Z, 0-9, Sonderzeichen)
N (engl. numeric)	Für numerische Daten (0-9, Dezimalzahlen)
L (engl. logical)	Für das logische TRUE (Wahr) bzw. FALSE (Unwahr).

Den Feldtyp "L" verwenden wir zunächst nicht, denn er wird nur für das Programmieren benötigt. Wir sollten uns einfach merken, daß Zahlenwerte in Feldern des Typs N und alle anderen Daten im Feldtyp C gespeichert werden. Für jedes Feld muß eine feste Länge angegeben werden. Daher können wir durch die Addition aller Feldlängen eines Datensatzes den Speicherplatz errechnen, den wir für einen Kunden auf der Diskette benötigen. Am Ende dieses Kapitels werden wir mit Hilfe dieses Wertes ermitteln, wie viele Kunden wir auf einer Diskette speichern können. Den letzten Parameter müssen wir nur bei numerischen Feldern angeben, in denen wir Dezimalzahlen abspeichern möchten. Er gibt die Anzahl der Dezimalstellen hinter dem Komma an. Bei der Berechnung der Satzlänge dürfen wir diesen Wert nicht berücksichtigen, denn er ist ja ein Teil der Feldlänge.

Dateistruktur eingeben

Jetzt haben wir genug Theorie gelernt und geben die Struktur für unsere Datei KUNDEN ein:

```
Satzstruktur folgendermaßen eingeben:
Feld    Name, Typ, Länge,   Dezimalstellen
 001     KUNDENNR,N,2
 002     NACHNAME,C,15
 003     VORNAME,C,10
 004     STRASSE,C,15
 005     PLZ,C,4
 006     ORT,C,12
 007     TELEFON,C,12
 008     UMSATZ,N,6,2
 009
```

Nachdem alle Felder eingegeben sind, drücken wir im Feld 009 <RETURN>, um dBASE II mitzuteilen, daß keine weiteren Eingaben mehr folgen.

Daten jetzt eingeben

Die Dateistruktur ist jetzt erstellt, und dBASE II fragt, ob wir Kunden eingeben möchten:

```
Daten jetzt eingeben?
```

Wenn Sie Kunden erfassen möchten, antworten Sie mit <J> für "Ja", ansonsten mit <N> für "Nein". Die Dateneingabe wird durch <RETURN> im Feld KUNDENNR abgebrochen. Der Punkt zeigt uns an, daß der CREATE-Befehl beendet ist. Wir können jetzt einen neuen Befehl eingeben. Zur Kontrolle lassen wir uns die gerade erstellte Dateistruktur mit der Eingabe DISPLAY

STRUCTURE <RETURN> noch einmal anzeigen. Falls wir einen Fehler gemacht haben, geben wir die Struktur neu ein. Leider haben wir vergessen, dBASE II mitzuteilen, welche Dateistruktur wir sehen möchten, und daher werden wir nach dem Namen der zugehörigen Datei gefragt:

```
Keine Datenbank eröffnet, bitte Dateinamen eingeben:
```

Anzeigen einer Dateistruktur

Hier geben wir KUNDEN <RETURN> ein und erhalten die Dateistruktur:

```
Strukturdaten für Datei: A:KUNDEN  .DBF
Anzahl der Sätze: 00000
Datum der letzten Aktualisierung: 01/04/86
Primäre Datei
Feld  Name    Typ  Länge  Dez.st.
001     KUNDENNR     N    002
002     NACHNAME     C    015
003     VORNAME      C    010
004     STRASSE      C    015
005     PLZ          C    004
006     ORT          C    012
007     TELEFON      C    012
008     UMSATZ       N    006    002
** Gesammt **             00079
```

Außer der Dateistruktur bekommen wir die Anzahl der gespeicherten Datensätze und das Datum der letzten Änderung unserer Datei KUNDEN angezeigt. In der letzten Zeile steht die Summe der Feldlängen. Das Wort "Gesammt" ist übrigens kein Druckfehler. dBASE II hat etwas Schwierigkeiten mit der deutschen Sprache. Die Bedeutung des Ausdrucks "Primäre Datei" werden wir später kennenlernen. Wie versprochen, errechnen wir jetzt, wie viele Kunden auf einer Diskette Platz finden. Ein Datensatz der Datei KUNDEN ist 79 Zeichen lang. Auf einer leeren 3-Zoll-Diskette können wir 184.320 Zeichen speichern. Wir rechnen nach der folgenden Formel aus, wie viele Kunden wir abspeichern können:

$$\frac{\text{Freier Speicherplatz}}{\text{Satzlänge}} = \text{max. Anzahl Datensätze}$$

$$\frac{184.320}{79} = 2333$$

Wenn wir mit einem Diskettenlaufwerk arbeiten, müssen auf dieser Diskette jedoch die dBASE-II-Programme und die Datei KUNDEN Platz finden. Die dBASE-II-Programme sind zusammen 61 kByte groß und belegen daher auf der Diskette 61 * 1024 = 62464 Zeichen. Es bleiben für unsere Datei 121.856 Zeichen übrig. Daher können wir in diesem Fall maximal 1542 Kunden abspeichern, wie die folgende Rechnung zeigt:

$$\frac{121.856}{79} = 1542$$

Im Kapitel "Installation von dBASE II" ist beschrieben, wie wir den freien Speicherplatz einer Diskette anzeigen können und wie eine Arbeitsdiskette angelegt wird. Wenn wir die nachfolgende Übung gelöst haben, sollten wir versuchen, eigene Dateistrukturen zu entwerfen, zum Beispiel für Adressen, Kochrezepte, Programme, Videokassetten, Schallplatten oder Bücher.

Übung 2:

1. Zur Speicherung von Artikeln soll eine Datei angelegt werden. Diese Datei wird später noch für weitere Übungen benötigt.

Dateiname: ARTIKEL.DBF
Felder: ARTIKELNR, 4stellig
ARTIKELBEZ, 20stellig
BESTAND, 4stellig
MINBESTAND, 4stellig
EK, 6stellig, 2 Nachkommastellen
VK, 6stellig, 2 Nachkommastellen

Die Abkürzungen der Feldnamen bedeuten:

Artikelnummer
Artikelbezeichnung
Lagerbestand
Mindestlagerbestand
Einkaufspreis
Verkaufspreis

Die Feldtypen müssen selbst bestimmt werden.

2. Wie lang ist ein Datensatz der Datei Artikel?
3. Wie viele Artikel können wir auf einer leeren Diskette speichern?

Kapitel 2

Erfassen von Datensätzen

In diesem Kapitel werden folgende Themen behandelt:

Eröffnen einer Datei
Eingabe von Datensätzen

Zu diesen Themen gehören die folgenden Befehle:

USE
APPEND

Im letzten Kapitel haben wir die Struktur der Beispieldatei KUNDEN.DBF definiert. Diese Datei werden wir nun mit Datensätzen füllen.

Eröffnen einer Datei

Um eine Datei zu bearbeiten, müssen wir dBASE deren Namen mitteilen. Dazu wird der neue Befehl USE <Dateiname> verwendet. In unserem Fall geben wir USE KUNDEN <RETURN> ein, und dBASE antwortet mit dem Punkt. Die Datei KUNDEN.DBF ist jetzt eröffnet, und wir können Datensätze hinzufügen. Der Befehl zur Eingabe von Datensätzen heißt APPEND, was übersetzt anhängen bedeutet. Tatsächlich werden die neuen Datensätze auf der Diskette hinter den alten Datensätzen abgespeichert. Geben wir nun APPEND <RETURN> ein und sehen, was passiert:

```
Satznummer 00001
KUNDENNR   :     :
NACHNAME   :                  :
VORNAME    :            :
STRASSE    :                  :
PLZ        :     :
ORT        :                  :
TELEFON    :              :
UMSATZ     :       :
```

Eingabe von Datensätzen

Es erscheint eine Eingabemaske auf dem Bildschirm, die entsprechend der Dateistruktur, die wir im letzten Kapitel erstellt haben, aufgebaut ist. Die Satznummer ist um eins größer als die Anzahl der gespeicherten Datensätze. In unserem Beispiel geben wir gerade den ersten Kunden ein. Der Cursor steht auf dem Eingabefeld KUNDENNR. Hier darf entsprechend der Feldbeschreibung (KUNDENNR,N,4) eine maximal vierstellige Zahl stehen. Versuchen wir einen Buchstaben einzugeben, ertönt zur Fehlermeldung ein kurzer Piepton. Nun geben wir den Herrn Otto Käufer aus der Ladengasse 9 in 5000 Köln 32 ein, der für DM 99,99 Ware gekauft hat und die Kundennummer 1 haben soll. Bei der Eingabe des Umsatzes muß an Stelle des Kommas ein Punkt eingegeben werden. Dies liegt an der amerikanischen Herkunft von dBASE II. Unser erster Kunde sollte folgendermaßen erfaßt sein:

```
Satznummer 00001
KUNDENNR   :0001:
NACHNAME   :Käufer          :
VORNAME    :Otto      :
STRASSE    :Ladengasse 9    :
PLZ        :5000:
ORT        :Köln 32         :
TELEFON    :            :
UMSATZ     :99.99 :
```

Ein eventueller Eingabefehler kann mit Hilfe von dBASE II leicht verbessert werden. Innerhalb eines Feldes können wir mit den Cursortasten nach links oder rechts wandern und Fehler überschreiben. Erreichen wir den Rand eines Feldes, wechseln wir in das vorhergehende bzw. nachfolgende Feld. Wenn wir in einem Feld keine Eingabe machen möchten, wie in unserem Beispiel im Feld TELEFON, drücken wir einfach <RETURN>, um in das nächste Feld zu gelangen.

Fehlerkorrektur

Zur Eingabe und Fehlerkorrektur können noch weitere von dBASE gebotene Möglichkeiten genutzt werden. Die Tasten unseres Schneider-Computers haben eine zweite und teilweise auch eine dritte Funktion. Drücken wir zum Beispiel die Taste E, so wird ein e am Bildschirm angezeigt. Wenn wir dagegen die Taste Ctrl drücken, festhalten und dann die Taste E betätigen, wandert der Cursor ein Feld nach oben. Für die Ctrl-Taste wird im folgenden das Zeichen ^ gedruckt. Somit bedeutet <^E>, daß die Ctrl- und die E-Taste gleichzeitig gedrückt werden sollen. Mit der Tastenkombination <^X> erreichen wir das nächste Feld, und mit <^S> sowie <^D> können wir den Cursor nach links und rechts bewegen. An Stelle der Cursortasten können wir also auch <^E>, <^X>,

<^S> und <^D> verwenden. Eine weitere nützliche Funktion ist der Einfügemodus, den wir mit <^V> einschalten und den dBASE mit der Meldung "Einfügen" quittiert.

Im Einfügemodus können Zeichen in bereits geschriebene Texte eingefügt werden. Durch nochmaliges Drücken der Tasten <^V> schalten wir den Einfügemodus wieder ab. Wenn wir gewohnt sind, Zeichen mit der Taste DEL zu löschen, müssen wir uns umstellen. Die Eingabe von <DEL> bewegt den Cursor eine Stelle nach links. Um ein Zeichen zu löschen, müssen wir daher entweder <^G> oder <CLS> drücken. Eine Übersicht der Tastenkombinationen finden wir am Ende dieses Kapitels. Sie lassen sich nicht nur beim APPEND-Befehl, sondern auch bei vielen anderen Befehlen einsetzen.

Wenn wir unseren ersten Kunden ordnungsgemäß erfaßt haben, sollten wir die folgenden Datensätze eingeben, da sie in den nächsten Kapiteln immer wieder als Beispiele zur Erklärung von Befehlen benutzt werden. Um die Beispiele am eigenen Computer nachvollziehen zu können, müssen die Sätze genau wie hier abgedruckt eingegeben werden:

```
0002,Meyer,theo,Eichenstr. 15,4100,Duisburg,0203-435612,999.78
0003,Maier,Udo,Kolpingstr. 14a,6050,Offenbach,069-3467,67.20
0004,Maiers,Trude,Bahnhofstr. 11,4300,Essen-West,0201-20316,117.98
0005,Hütter,Angelika,Sandfuhrstr. 34,4200,Oberhausen,0208-56129,79.80
0006,Hutter,Sabine,Heideweg 23,6790,Oberammergau,0122-4554,978.40
0007,Ende,Werner,Am Schlußlicht,5428,Endlichhofen,07142-568,789.50
```

Hier ist die Zusammenstellung aller Tasten, die bei dem Befehl APPEND zur Fehlerkorrektur angewendet werden können:

Tastenkomb.	Entspricht	Wirkung
<^E>	< ↑ >	Cursor ein Feld nach oben
<^X>	< ↓ >	Cursor ein Feld nach unten
<^S>	< ← > und <DEL>	Cursor ein Zeichen nach links
<^D>	< → >	Cursor ein Zeichen nach rechts
<^V>		Umschalten von Überschreib-/ in Einfügemodus
<^G>	<CLR>	Löscht ein Zeichen

Die Cursor-Tasten können nur bei Verwendung des DIN-Zeichensatzes und der DIN-Tastatur wie angegeben verwendet werden!

Übung 3:

1. Wie lautet der Befehl, um die Datei ARTIKEL.DBF zu eröffnen?
2. Bitte geben Sie folgende Artikel in die Datei ARTIKEL.DBF ein:

```
0001,Joystick de Luxe,12,75,19.90,49.00
0002,3 Zoll Disketten,120,50,6.00,9.90
0003,Colour Monitor,3,3,798.00,998.00
0004,Schutzhuelle,23,30,9.90,20.07
0005,Druckerkabel,2,1,24.00,49.42
0006,5 1/4 Zoll Disketten,200,150,3.96,4.94
```

Kapitel 3

Anzeigen von Daten einer Datei

In diesem Kapitel werden folgende Themen behandelt:

Anzeigen von einzelnen Datensätzen
Anzeigen aller Datensätze
Ausdruck von Datensätzen

Zu diesen Themen gehören die folgenden Befehle:

DISPLAY
DISPLAY ALL
GOTO
LIST
SET PRINT OFF
SET PRINT ON

Der DISPLAY-Befehl ermöglicht nicht nur, eine Dateistruktur anzuzeigen, wie wir im Kapitel "Erstellung einer Dateistruktur" gelernt haben. Er ist ein sehr vielseitiger Befehl, den wir nun zur Anzeige von Datensätzen nutzen wollen. Wir verwenden als Beispiel wieder unsere Datei KUNDEN, die wir mit der Eingabe USE KUNDEN <RETURN> eröffnen.

Anzeigen von einzelnen Datensätzen

Die einfachste Möglichkeit, den DISPLAY-Befehl zu verwenden, ist, DISPLAY <RETURN> einzugeben. Wir bekommen den ersten Kunden angezeigt:

```
.use kunden
.display
00001     1  Käufer          Otto       Ladengasse 9    5000 Köln 32
          99.99
```

In dieser Form zeigt der Befehl immer den Satz an, auf den dBase II gerade in der Datei zeigt. Wir können uns dies am Beispiel eines Karteikastens verdeutlichen. Stellen wir uns vor, wir hätten für jeden Kunden eine Karteikarte ge-

schrieben. Alle Karten befinden sich in einem Karteikasten. Wenn wir den Deckel des Kastens öffnen, blicken wir auf die erste Karte, also die Daten des ersten Kunden, die wir lesen können. Die anderen Kunden können wir nicht direkt lesen, denn ihre Karten stecken hinter dieser Karte und werden daher von ihr verdeckt.

Der Befehl USE hat bildlich gesehen den Karteikasten geöffnet, und wir können mit dem Befehl DISPLAY die erste Kundenkarte angezeigt bekommen. In einem Karteikasten müssen wir eine Karteikarte herausziehen, bevor wir sie lesen können. In dBASE II zeigen wir zunächst mit dem Befehl GOTO <Satznummer> auf den Satz, der den gewünschten Kunden enthält, und fordern dBASE II dann mit dem DISPLAY-Befehl auf, diesen Satz anzuzeigen. Geben wir GOTO 2 <RETURN> ein, gefolgt von der Eingabe DISPLAY <RETURN>, wird der zweite Satz angezeigt:

```
.goto 2
.display
00002    2  Meyer          theo      Eichenstr. 15  4100 Duisburg  020
3-435612  999.78
```

Versuchen wir, mit GOTO auf einen Kunden zu zeigen, der noch nicht eingegeben wurde, wie zum Beispiel mit GOTO 1000 <RETURN>, erfolgt die Fehlermeldung:

```
. goto 1000
Satz überschreitet belegten Bereich der Datenbank
goto 1000
Korrigieren und wiederholen? (J/N)
```

Wenn wir eine falsche Eingabe gemacht haben, fragt dBASE II uns, ob wir die Eingabe ändern möchten. Geben wir <J> ein, um zu sehen, was passiert:

```
Ändern von :
```

Wir werden gefragt, welcher Teil unserer Eingabe geändert werden soll. Bei dieser Abfrage geben wir nur den falschen Teil der Eingabe an. In unserem Fall geben wir 1000 <RETURN> ein. Als nächstes müssen wir angeben, durch was "1000" ersetzt werden soll:

```
Ändern nach :
```

Hier geben wir 3 <RETURN> ein, um auf den dritten Satz zu zeigen. Bevor dBASE II unseren Befehl ausführt, fragt es, ob weitere Änderungen durchzuführen sind:

```
goto 3
Weitere Korrekturen ? (J/N)
```

In unserem Fall ist jetzt alles korrekt, und wir können N <RETURN> eingeben. Hätten wir mehrere Fehler in unserer Eingabe, könnten wir mit der Eingabe J <RETURN> die Fehlerkorrektur fortführen.

Wir haben jetzt gelernt, einzelne Sätze über deren Satznummern anzuzeigen. Diese Methode hat den Nachteil, daß wir die Satznummern der Kunden kennen müssen, deren Daten wir uns ansehen möchten. Wenn wir uns alle Kunden oder eine größere Anzahl Kunden ansehen wollen, ist diese Methode zudem sehr umständlich und zeitraubend. Zum Glück gibt es dafür andere Möglichkeiten. Den DISPLAY-Befehl können wir um Parameter erweitern, die bestimmen, welche Sätze angezeigt werden.

Anzeigen aller Datensätze

Um alle Sätze einer eröffneten Datei anzuzeigen, geben wir DISPLAY ALL <RETURN> ein und erhalten zum Beispiel eine solche Ausgabe:

```
.display all
00001     1  Käufer          Otto       Ladengasse 9     5000 Köln 32
          99.99
00002     2  Meyer           theo       Eichenstr. 15    4100 Duisburg
0203-435612   999.78
00003     3  Maier           Udo        Kolpingstr. 14a  6050 Offenbach
069-3467        67.20
00004     4  Maiers          Trude      Bahnhofstr. 11   4300 Essen-West
0201-20316    117.98
00005     5  Hütter          Angelika   Sandfuhrstr. 34  4200 Oberhausen
0208-56129      0.00
00006     6  Hutter          Sabine     Heideweg 23      6790 Oberammergau
0122-4554     978.40
00007     7  Ende            Werner     Am Schlußlicht   5428 Endlichhofen
07142-568     789.50
```

Nach der Ausgabe von maximal 15 Sätzen unterbricht dBASE II die Bildschirmausgabe, damit wir uns die Sätze in Ruhe ansehen können. Die nächsten 15 Sätze bekommen wir angezeigt, nachdem wir irgendeine Taste gedrückt haben.

Am Anfang jedes Satzes steht dessen Satznummer. Die Anzeige der Satznummern können wir mit dem Parameter OFF unterdrücken. Probieren wir dies mit den Eingaben

```
.DISPLAY ALL OFF <RETURN>
```

zur Ausgabe aller Sätze, bzw.

```
.DISPLAY OFF <RETURN>
```

zur Ausgabe eines Satzes ohne Anzeige der Satznummern. Wenn wir nur bestimmte Informationen über die Kunden benötigen, zum Beispiel die Kundennummer, den Nachnamen und den Umsatz, so können wir die Felder hinter dem DISPLAY-Befehl explizit angeben:

```
.DISPLAY ALL KUNDENNR NACHNAME UMSATZ OFF <RETURN>
```

Als Ausgabe erhalten wir dann:

```
1 Käufer          99.99
2 Meyer          999.78
3 Maier           67.20
4 Maiers         117.98
5 Hütter           0.00
6 Hutter         978.40
7 Ende           789.50
```

Es gibt noch wesentlich mehr Möglichkeiten, den DISPLAY-Befehl anzuwenden, für die wir aber weitere Kenntnisse erwerben müssen. Wir werden dem DISPLAY-Befehl daher noch häufiger begegnen. Damit wir unsere Kunden nicht nur auf dem Bildschirm, sondern auch auf dem Drucker ausgeben können, lernen wir jetzt, wie dies am einfachsten geht. Später werden wir im Kapitel "Der Listengenerator Report" noch weitere Möglichkeiten zur Druckausgabe kennenlernen.

Ausdruck von Datensätzen

Um eine Liste aller Kunden auszudrucken, geben wir DISPLAY ALL OFF <^P> <RETURN> ein. Das Neue daran ist die Eingabe von <^P>. Mit <^P> schalten wir den Drucker ein, und alle Bildschirmausgaben werden auf ihm protokolliert. Die nochmalige Eingabe von <^P> schaltet den Drucker aus.

Anstelle von <^P> kann der Drucker auch mit SET PRINT ON <RETURN> eingeschaltet und mit SET PRINT OFF <RETURN> ausgeschaltet werden.

Für den DISPLAY-Befehl können wir fast immer den Befehl LIST verwenden. Der LIST-Befehl wirkt im Unterschied zum DISPLAY-Befehl generell auf alle Datensätze. Dieser Unterschied wird uns sofort klar, wenn wir LIST <RETURN> eingeben. Als Ausgabe erhalten wir alle Datensätze unserer Datei KUNDEN, ähnlich wie bei der Eingabe von DISPLAY ALL <RETURN>.

Wenn wir mehr als 15 Kunden in unserer Datei gespeichert haben, kennen wir mittlerweile auch den zweiten Unterschied zwischen dem DISPLAY- und dem LIST-Befehl. Die Ausgabe durch LIST bricht nämlich nicht jeweils nach 15 Bildschirmzeilen ab, sondern alle Kunden werden ohne Unterbrechung ange-

zeigt. Dies ist sehr praktisch, wenn wir eine Liste auf dem Drucker ausgeben wollen, da der Ausdruck nicht durch dBASE-II-Meldungen unterbrochen wird.

Die Ausgabe auf dem Bildschirm können wir jederzeit mit der Eingabe <^S> unterbrechen; durch den Druck einer beliebigen Taste wird die Ausgabe fortgesetzt. Diese Unterbrechung können wir an jeder Stelle der Ausgabe vornehmen, vorausgesetzt wir reagieren schnell genug. Daher sollten wir die Finger der linken Hand schon auf den Tasten Ctrl und S liegen haben, bevor wir mit der rechten Hand die Eingabetaste betätigen.

Ansonsten können wir mit dem LIST-Befehl die gleichen Parameter einsetzen wie mit dem DISPLAY-Befehl. Zur Übung probieren wir die folgenden Befehlszeilen aus:

```
.LIST OFF <RETURN>
.LIST KUNDENNR NACHNAME UMSATZ OFF <RETURN>
```

Die nächsten Befehlszeilen sollten wir nur dann eingeben, wenn ein Drucker angeschlossen und betriebsbereit ist:

```
.SET PRINT ON <RETURN>
.LIST OFF <RETURN>
.SET PRINT OFF <RETURN>
```

Jetzt fällt es uns bestimmt leicht, die folgende Übung zu lösen.

Übung 4:

1. Der LIST-Befehl läßt sich, sieht man von der unterschiedlichen Bildschirmausgabe einmal ab, durch den DISPLAY-Befehl ersetzen. Bitte ersetzen Sie in den folgenden Befehlszeilen den LIST-Befehl:

   ```
   .LIST OFF
   .LIST NACHNAME,VORNAME,STRASSE,PLZ,ORT OFF
   ```

2. Wozu dient der Parameter OFF?
3. Schreiben Sie die Befehle auf, die notwendig sind, um die Nachnamen, Vornamen und Telefonnummern aller Kunden auf dem Drucker auszugeben.

Kapitel 4

Sortieren in einer Datenbank

In diesem Kapitel werden folgende Themen behandelt:

Physikalische Sortierung
Sortierung
Kopieren einer Datei
Löschen einer Datei
Indizierte Sortierung
Anlegen einer Indexdatei
Eröffnen einer Indexdatei
Wechseln der Indexdatei
Reindizieren

Zu diesen Themen gehören die folgenden Befehle:

COPY
DELETE FILE
INDEX
INDEX ON
REINDEX
SET INDEX TO
SORT

Physikalisches Sortieren

Sortierung

Es gibt zwei Möglichkeiten, eine sortierte Reihenfolge von Datensätzen zu erhalten.

Die erste Möglichkeit besteht darin, die Anordnung der Datensätze in einer Datei zu verändern. Wir können uns diesen Vorgang an einem Karteikasten verdeutlichen. In einer Kartei werden die Karteikarten geordnet in den Karteikasten gelegt, denn sonst weiß man nicht, wo mit der Suche nach einer be-

stimmten Karteikarte begonnen werden soll. Daher sortiert man zum Beispiel die Karteikarten einer Kundenkartei nach den Nachnamen der Kunden. Wenn wir jemanden nach seiner Kundennummer oder nach der Größe seines Umsatzes suchen, benötigen wir jeweils eine neue Kartei, die nach dem jeweiligen Sortierkriterium (Kundennummer, Umsatz) sortiert ist.

Bei unserem Computer sind die Kunden auf der Diskette gespeichert. Wir können ihre Anordnung mit dBASE II leicht ändern. Dazu gibt es den Befehl SORT. Mit diesem Befehl kann nach jedem beliebigen Datenfeld einer Datei sortiert werden. Unsere Datei KUNDEN könnten wir also nach den Feldern KUNDENNR, NACHNAME, VORNAME, STRASSE, PLZ, ORT, TELEFON und UMSATZ sortieren. Es wird allerdings von dBASE II für jedes Sortierkriterium eine neue Datei auf der Diskette angelegt.

Die sortierte Datei benötigt ebensoviel Speicherplatz wie die unsortierte. Daher ist der SORT-Befehl bei größeren Dateien nur mit zwei Diskettenlaufwerken einsetzbar.

Wir wollen jetzt die Datei KUNDEN nach dem Feld NACHNAME sortieren. Die neue, sortierte Datei nennen wir SORTIERT. Dazu eröffnen wir die Datei KUNDEN mit USE KUNDEN <RETURN> und sehen uns den Inhalt mit LIST <RETURN> an:

```
00001    1  Käufer         Otto      Ladengasse 9    5000 Köln 32
          99.99
00002    2  Meyer          theo      Eichenstr. 15   4100 Duisburg
0203-435612  999.78
00003    3  Maier          Udo       Kolpingstr. 14a 6050 Offenbach
069-3467       67.20
00004    4  Maiers         Trude     Bahnhofstr. 11  4300 Essen-West
0201-20316    117.98
00005    5  Hütter         Angelika  Sandfuhrstr. 34 4200 Oberhausen
0208-56129      0.00
00006    6  Hutter         Sabine    Heideweg 23     6790 Oberammergau
0122-4554     978.40
00007    7  Ende           Werner    Am Schlußlicht  5428 Endlichhofen
07142-568     789.50
```

Die Kundennummern sind fortlaufend von 1 bis zum Ende der Datei numeriert. Jetzt geben wir den Befehl zum Sortieren:

```
.SORT ON NACHNAME TO SORTIERT <RETURN>
```

Es dauert einige Sekunden, bis die Sätze sortiert sind. dBASE II teilt uns mit der Meldung "SORT beendet" mit, daß der Sortiervorgang beendet ist. Wenn wir nun LIST <RETURN> eingeben, zeigt sich aber wieder eine ungeordnete Datei. Was haben wir falsch gemacht?

Richtig, die Datei KUNDEN ist ja nicht verändert worden, sondern wir haben die neue Datei SORTIERT angelegt. Diese Datei müssen wir erst mit USE SORTIERT <RETURN> eröffnen, bevor wir uns ihren Inhalt ansehen können. Sehen wir uns nun mit LIST <RETURN> die sortierte Reihenfolge an:

```
. use sortiert
. list
00001    7  Ende           Werner    Am Schlußlicht  5428 Endlichhofen
07142-568     789.50
00002    6  Hutter         Sabine    Heideweg 23     6790 Oberammergau
0122-4554     978.40
00003    5  Hütter         Angelika  Sandfuhrstr. 34 4200 Oberhausen
0208-56129      0.00
00004    1  Käufer         Otto      Ladengasse 9    5000 Köln 32
          99.99
00005    3  Maier          Udo       Kolpingstr. 14a 6050 Offenbach
069-3467       67.20
00006    4  Maiers         Trude     Bahnhofstr. 11  4300 Essen-West
0201-20316    117.98
00007    2  Meyer          theo      Eichenstr. 15   4100 Duisburg   020
3-435612 999.78
```

Aus dem ASCII-Zeichensatz, der im Anhang E angegeben ist, ergibt sich die folgende Reihenfolge, nach der sortiert wird:

Sonderzeichen
Ziffern 0-9
Großbuchstaben A-Z
Umlaute Ä,Ö,Ü
Kleinbuchstaben a-z
Umlaute ä,ö,ü
ß

Wir sollten uns auf jeden Fall merken, daß Ziffern im ASCII-Zeichensatz ein niedrigerer Wert als Buchstaben zugeordnet ist und Umlaute einen höheren Wert besitzen als der entsprechende Vokal. Wir können diese Reihenfolge mit der folgenden Eingabe umkehren:

```
.USE KUNDEN <RETURN>
.SORT ON NACHNAME TO SORTIERT DESCENDING <RETURN>
```

Wenn dBASE II sortiert hat, sehen wir uns die Datei SORTIERT an:

```
.USE SORTIERT <RETURN>
.LIST <RETURN>
```

Die Sätze sind nun absteigend sortiert. Damit steht der Kunde, der vorhin an der ersten Stelle war, jetzt an der letzten Stelle. Dies haben wir durch den Parameter DESCENDING erreicht, den wir an Stelle von ASCENDING einge-

setzt haben. Die Spezifizierung ASCENDING kann bei aufsteigender Sortierreihenfolge auch weggelassen werden. Die Eingabe von "SORT ON NACHNAME TO SORTIERT" bewirkt somit das gleiche wie "SORT ON NACHNAME TO SORTIERT ASCENDING".

Kopieren einer Datei

Um Speicherplatz zu sparen, können wir die sortierten Datensätze in unsere ursprüngliche Datei KUNDEN kopieren und dann die Datei SORTIERT löschen:

```
.USE SORTIERT <RETURN>
.COPY TO KUNDEN <RETURN>
```

Das Ergebnis sehen wir uns mit

```
.USE KUNDEN <RETURN>
.LIST <RETURN>
```

an. Die Datei KUNDEN enthält nun die sortierten Sätze aus der Datei SORTIERT.

Löschen einer Datei

Um die Datei SORTIERT zu löschen, geben wir

```
.DELETE FILE SORTIERT.DBF <RETURN>
```

ein. Zu einem späteren Zeitpunkt werden die Befehle COPY und DELETE FILE genauer erklärt.

Bevor wir das indizierte Sortieren kennenlernen, lösen wir die nächste Übung.

Übung 5:

1. Die Datei KUNDEN ist aufsteigend nach Postleitzahlen zu sortieren, wobei die sortierten Datensätze in der Datei SORTIERT abgelegt werden sollen. Das Ergebnis wird mit dem LIST-Befehl überprüft. (Achtung: Es muß die richtige Datei eröffnet sein!). Wir kopieren jetzt die sortierten Datensätze der Datei SORTIERT in die Datei KUNDEN und löschen dann die Datei

SORTIERT von der Diskette. Die notwendigen Befehle sollen aufgeschrieben werden.
2. Wie ändert man die Sortierreihenfolge beim SORT-Befehl von aufsteigender in absteigende Sortierung um?

Indizierte Sortierung

Die Verwendung des SORT-Befehls hat zwei entscheidende Nachteile. Er benötigt viel Speicherplatz, und die Sortierung geschieht relativ langsam.

Anlegen einer Indexdatei

Diese Nachteile gibt es bei der indizierten Sortierung nicht, da die Datensätze auf der Diskette in ihrer alten Reihenfolge bestehen bleiben. Es wird zwar auch eine weitere Datei pro Sortierkriterium erzeugt, jedoch nicht für Daten, sondern für Indizes.

Eine solche Datei wird in dBASE II mit dem Namenszusatz ".NDX" versehen, anstelle der Erweiterung ".DBF" für Dateien, die Daten enthalten. Das Kürzel NDX steht für "index", die Erweiterung DBF für "database file".

Die Funktion der Indexdatei verdeutlichen wir uns an einem Beispiel. Um eine nach Nachnamen sortierte Datei zu erhalten, sortiert dBASE II das Feld NACHNAMEN der Datei KUNDEN und speichert die sortierte Reihenfolge zusammen mit den jeweils zugehörigen Satznummern in einer Indexdatei ab. Demnach bleibt unsere Datei KUNDEN unverändert, der Zugriff auf einen bestimmten Satz in dieser Datei geschieht über die Indexdatei. Diese Indexdatei sieht vereinfacht so aus:

Ende	00007
Hutter	00006
Hütter	00005
Käufer	00001
Maier	00003
Maiers	00004
Meyer	00002

Wenn wir die Datei KUNDEN sortiert auflisten wollen, muß dBASE II zunächst in der Indexdatei nachsehen, welcher Nachname an der ersten Stelle steht und welche Satznummer zu diesem Namen gehört. Dann wird der entsprechende Satz direkt von der Diskette gelesen und am Bildschirm angezeigt.

Bei dem nächsten Satz wird analog verfahren, bis das Ende der Indexdatei erreicht wird. Wir werden jetzt die notwendigen Befehle eingeben:

```
.USE KUNDEN <RETURN>
.INDEX ON NACHNAME TO NAME <RETURN>
.LIST <RETURN>
```

Mit INDEX ON haben wir die Datei NAME.NDX angelegt, die von jetzt an als Indexdatei genutzt werden kann. Das Anlegen einer Indexdatei geht zwar wesentlich schneller als das physikalische Sortieren mit dem SORT-Befehl, aber bei größeren Dateien kann es schon ein wenig dauern, bis dieser Vorgang abgeschlossen ist.

Eröffnen einer Indexdatei

Wir müssen die Indexdatei aber nicht immer wieder neu anlegen, sondern geben sie einfach an, wenn wir eine Datei eröffnen:

```
.USE KUNDEN INDEX NAME <RETURN>
```

Wie in der letzten Übung werden wir jetzt eine nach Postleitzahlen sortierte Liste herstellen, allerdings mit dem INDEX-Befehl. Zunächst erstellen wir die Indexdatei für Postleitzahlen:

```
.INDEX ON PLZ TO PLZ <RETURN>
```

Durch die Eingabe von LIST <RETURN> überzeugen wir uns, daß wir tatsächlich eine nach Postleitzahlen sortierte Liste erhalten. In dieser Liste sind die Postleitzahlen sortiert, aber bei gleichen Postleitzahlen sind die Nachnamen in zufälliger Reihenfolge genannt. Es ist uns vielleicht schon bei der nach Nachnamen sortierten Liste aufgefallen, daß innerhalb gleicher Nachnamen die Vornamen nicht sortiert waren.

Das geht auch nur dann, wenn nach zwei Feldern sortiert wird. Also müssen wir eine Indexdatei anlegen, die zwei Felder als Sortierkriterium enthält. Das läßt sich mit dBASE II recht einfach realisieren:

```
.INDEX ON NACHNAME + VORNAME TO NAME <RETURN>
.LIST <RETURN>
```

Es ist auch möglich, drei Felder mit dem + -Zeichen zu einem Sortierkriterium zu verbinden, jedoch wird die Indexdatei dadurch entsprechend vergrößert, und es wird mehr Zeit benötigt, um sie anzulegen. Wie wir gesehen haben, ermöglicht die Indexdatei eine schnelle Sortierung, wobei weniger Speicherplatz für das Sortieren benötigt wird als bei dem SORT-Befehl.

Was passiert aber, wenn wir Sätze mit APPEND zu unserer bestehenden Datei hinzufügen, Sätze ändern oder ganz löschen? In diesem Fall muß entweder die Indexdatei mit INDEX ON neu angelegt oder bei der Dateieröffnung der zu ändernden Datei die vorhandene Indexdatei mit angegeben werden. Haben wir mehrere Indexdateien angelegt, zum Beispiel die Indexdateien PLZ.NDX und NAME.NDX, müssen wir bei der Dateieröffnung alle angeben:

```
.USE KUNDEN INDEX NAME,PLZ <RETURN>
.APPEND
```

Die zuerst genannte Indexdatei bestimmt das Sortierkriterium. Geben wir Kunden ein, werden beide Indexdateien laufend aktualisiert. Wie erhalten wir jetzt eine sortierte Liste nach Postleitzahlen?

Die erste Möglichkeit wäre, die Datei KUNDEN neu zu eröffnen und die Reihenfolge der Indexdateien zu ändern. Als zweite Möglichkeit gibt es den Befehl SET INDEX TO. Dieser Befehl ermöglicht, eine vorhandene Indexdatei anzusprechen:

```
.SET INDEX TO PLZ,NAME
```

Es gibt meist mehrere Wege, um mit dBASE II das gleiche Ergebnis zu erhalten. Diese Wege unterscheiden sich in der Anzahl benötigter Tasteneingaben und der Geschwindigkeit, mit der dBASE II unsere Eingaben bearbeitet.

Haben wir zum Beispiel beim Eröffnen unserer Datendatei die Indexdateien vergessen und Veränderungen an der Datei vorgenommen, müssen wir nicht alle zugehörigen Indexdateien neu erstellen.

Wechseln der Indexdatei

Es reicht, wenn wir die Datei noch einmal, diesmal mit den Indexdateien, eröffnen und dann den REINDEX-Befehl anwenden:

```
. USE KUNDEN INDEX NAME, PLZ
. REINDEX
```

Dieser Befehl reindiziert alle eröffneten Indexdateien und zeigt das Ergebnis am Bildschirm an:

```
Neuindizierung der INDEX-Datei - A:NAME     .NDX
00009 Sätze indiziert

Neuindizierung der INDEX-Datei - A:PLZ      .NDX
00009 Sätze indiziert
```

Durch Üben lernt man die verschiedenen Möglichkeiten, die dBASE II anbietet, schnell kennen und anzuwenden. Man benötigt zwar nur wenige Befehle, um mit dBASE II arbeiten zu können, die Arbeit wird aber umso effektiver, je größer der eigene Befehlswortschatz ist.

Übung 6:

1. Welche Vorteile bietet das Indizieren gegenüber dem Sortieren?
2. Welche Befehle sind notwendig, um eine nach Artikelbezeichnungen sortierte Liste unserer Datei ARTIKEL (siehe Übung 1) zu erhalten?
3. Wie erhält man eine nach Postleitzahl, Strasse und Nachname sortierte Liste aus der Datei KUNDEN?

Kapitel 5

Suchen und Anzeigen von Daten einer Datenbank

In diesem Kapitel werden folgende Themen behandelt:

Die Parameter
Die Bedingungen
Relationale Operatoren
Logische Operatoren
Die Substring-Funktion
Suchen und Anzeigen von Daten

Zu diesen Themen gehören die folgenden Befehle:

@()
FIND
LIST FOR
LIST WHILE
LOCATE

Die Parameter

Wir haben im dritten Kapitel "Anzeigen von Daten einer Datei" am Beispiel der Befehle DISPLAY und LIST die Möglichkeit kennengelernt, Parameter hinter Befehlen anzugeben. Es gibt in dBASE II folgende Parameter, die man mit mehreren Befehlen zusammen verwenden kann:

ALL
ON
OFF
FIELD
FIELDS
FOR
WHILE

Mit Hilfe des ALL-Parameters teilen wir dBASE II mit, daß ein Befehl auf alle Datensätze in einer Datei wirken soll. Diesen Parameter können wir zusammen mit den Befehlen

DELETE
DISPLAY
RECALL
REPLACE
RELEASE

einsetzen. Einige dBASE-II-Befehle wirken immer auf alle Datensätze, ohne daß der ALL-Parameter explizit angegeben werden muß. Ein Beispiel dafür ist der LIST-Befehl.

Der Parameter ON dient dazu, eine Funktion einzuschalten, die wir dann mit dem Parameter OFF wieder ausschalten können. Diese Parameter haben wir in Zusammenhang mit dem SET PRINT, DISPLAY- und LIST-Befehl (Ausgabe der Satznummern ausschalten) bereits kennengelernt. Der ON/OFF-Parameter läßt sich bei folgenden Befehlen einsetzen:

DISPLAY (nur OFF)
LIST (nur OFF)
SET

In Verbindung mit dem SET-Befehl gibt es eine große Anzahl von Funktionen, die sich mit dem ON/OFF-Parameter ein- bzw. ausschalten lassen. Der SET-Befehl wird später noch ausführlich besprochen. Es gibt noch eine weitere Bedeutung von ON, die aber nicht mit den oben genannten Befehlen zusammen eingesetzt wird. Der FIELDS-Parameter gibt an, daß sich ein Befehl nur auf bestimmte Felder, die wir hinter dem Parameter angeben müssen, beziehen soll. Wir haben dies bereits in Verbindung mit dem DISPLAY- und LIST-Befehl ausprobiert. Der FIELDS-Parameter muß nicht immer angegeben werden. Meistens reicht es aus, nur die gewünschten Felder hintereinander zu schreiben. Er läßt sich zusammen mit folgenden Befehlen einsetzen:

DISPLAY
JOIN TO
LIST
TOTAL TO

Die gleiche Funktion hat der Parameter FIELD, der mit diesen Befehlen zusammen eingesetzt werden kann:

CHANGE
COPY

Viele der genannten Befehle kennen wir noch nicht, sie werden später noch ausführlich erläutert. Der vollständigen Darstellung halber wurden sie alle aufgeführt. Da die große Anzahl der Befehle und Parameter eine Unzahl von Kombinationsmöglichkeiten zuläßt, ist es nicht möglich, in diesem Buch alle Kombinationen darzustellen. Sie können aber jederzeit hier nachschlagen, wenn Ihnen einer der obigen Parameter unklar ist oder Sie wissen möchten, ob sich ein Befehl mit einem der genannten Parameter einsetzen läßt. Alle dBASE-II-Befehle sind zudem im Anhang mit allen zugehörigen Parametern aufgeführt. Im folgenden Text werden diese Parameter daher nicht mehr bei jedem Beispiel erklärt.

Wir sind noch nicht auf die Parameter FOR und WHILE eingegangen. Sie ermöglichen, aus einer Menge von Datensätzen diejenigen herauszufinden, die eine von uns bestimmte Bedingung erfüllen. Um diese Parameter mit allen ihren Möglichkeiten einsetzen zu können, müssen wir wissen, was eine Bedingung ist und welche Bedingungen es gibt. Die Bedingungen werden im nächsten Abschnitt behandelt. Hier folgt die Aufzählung aller Befehle, bei denen beide Parameter einsetzbar sind:

APPEND FROM
COPY
COUNT
DELETE
LIST
REPORT
SUM
TOTAL TO

Bei diesen Befehlen läßt sich nur der Parameter FOR einsetzen:

CHANGE
DISPLAY
LOCATE

Es gibt noch einige Parameter mehr, wie zum Beispiel ASCENDING und DESCENDING, die wir im Zusammenhang mit dem SORT-Befehl kennenlernten. Diese Parameter sind aber meist nur mit einem Befehl zusammen einsetzbar. Sie werden daher in Zusammenhang mit dem jeweils zugehörigen Befehl erörtert.

Übung 7:

1. Mit welchem Parameter kann man eine Funktion ausschalten?
2. Welche Funktion hat der Parameter ALL?
3. Wozu dient der Parameter FIELD bzw. FIELDS?

Die Bedingungen

Bedingungen helfen uns, aus einer größeren Menge von Datensätzen diejenigen herauszufinden, die wir bearbeiten möchten. Wenn wir aus unserer Kundendatei Herrn Maier anzeigen möchten, dann müssen wir zunächst den Kunden mit dem Nachnamen Maier finden. In unserer Datei KUNDEN gibt es ein Feld NACHNAME, das wir mit dem Namen Maier vergleichen können. Unsere Bedingung würde also lauten: NACHNAME = 'Maier'. Das heißt, wir vergleichen eine spezifizierte Zeichenkette mit dem Feldinhalt des Feldes NACHNAME.

Relationale Operatoren

Die Vergleichsfunktion übernimmt ein sogenannter relationaler Operator. Der Name Maier wurde in Hochkommata gesetzt, da das Feld NACHNAME ein alphanumerisches Feld ist. Bei einem numerischen Feld wie dem Feld UMSATZ würden wir die Hochkommata oder Anführungszeichen, die man an deren Stelle verwenden kann, weglassen. Suchen wir zum Beispiel Karteileichen, d.h. alle Kunden, deren Umsatz gleich null ist, so würde die Bedingung wie folgt lauten:

```
UMSATZ = 0
```

Wollen wir alle Kunden mit einem Umsatz größer als 10000,- DM finden, etwa um zu wissen, welche Kunden einen Jahresbonus erhalten, dann formulieren wir unsere Bedingung so:

```
UMSATZ > 10000
```

An Stelle des Größer-Zeichens ">" könnten wir das Kleiner-Zeichen "<" setzen, falls wir alle Kunden suchen, deren Umsatz kleiner als 10000,- DM ist.

Diese beiden Zeichen gehören ebenfalls zur Gruppe der relationalen Operatoren. Zu Vergleichen können wir diese Operatoren einzeln oder zu zweit verwenden. Dabei sind folgende Kombinationen möglich:

<=	oder	=<	(kleiner gleich)
>=	oder	=>	(größer gleich)
<>	oder	#	(ungleich)

Logische Operatoren

Wir können aber auch die Kunden suchen, deren Nachname mit den Buchstaben E bis K beginnt, indem wir eine Bedingung aus zwei miteinander ver-

knüpften Vergleichen formulieren:

```
NAME > 'D' .AND. NAME < 'L'
```

Das ".AND." heißt übersetzt "Und". Es verbindet beide Vergleiche miteinander. Diese Bedingung ist nur dann erfüllt, wenn beide Vergleiche erfüllt sind.

Zu der Gruppe der sogenannten logischen Operatoren gehört noch das ".NOT.", das ".OR.", die Klammern "()" sowie das $-Zeichen. Ein .NOT. bedeutet die Umkehrung des Vergleichs. Die Bedingung

```
UMSATZ .NOT. > 10000
```

bedeutet somit, daß alle Kunden gesucht werden, deren Umsatz kleiner oder gleich, aber nicht größer als 10000 ist.

Der Operator ".OR." heißt übersetzt "Oder". Eine Bedingung, die zwei mit ".OR." verknüpfte Vergleiche enthält, ist dann erfüllt, wenn mindestens einer der beiden Vergleiche stimmt.

Als Beispiel sehen wir uns die folgende Bedingung an, die aus einer Kundenkartei die Kunden herausfindet, die Maier oder Schulz heißen:

```
NAME = 'Maier' .OR. NAME = 'Schulz'
```

Die Klammer wird immer dann gesetzt, wenn wir eine komplizierte Bedingung haben, bei der einzelne Vergleiche zunächst gezogen werden sollen, bevor eine Verknüpfung stattfindet. Das nächste Beispiel zeigt dies deutlich:

```
(NAME = 'Maier' .AND. PLZ = '4100') .OR. (NAME = 'Schulz' .AND. PLZ =
'8000')
```

Hier würden alle Kunden, die Maier heißen und in Duisburg wohnen sowie alle Kunden, die Schulz heißen und in München wohnhaft sind, gefunden.

Die Substring-Funktion

Mit den bisher besprochenen Bedingungen läßt sich ein Feld ab der ersten Stelle mit einer Zeichenkette vergleichen. Wenn eine gesuchte Zeichenkette erst ab der zweiten oder einer weiteren Stelle des Feldinhaltes beginnt, so wird diese nicht gefunden.

Es gibt aber auch Funktionen, mit denen überprüft werden kann, ob eine Zeichenkette Teil einer anderen Zeichenkette oder eines Datenfeldes ist. Eine solche Funktion ist @('Zeichenkette','Zeichenkette'). Diese Funktion überprüft,

ob die erste angegebene Zeichenkette ein Teil der zweiten Zeichenkette darstellt. Anstelle der Zeichenketten kann auch eine alphanumerische Variable oder ein alphanumerisches Datenfeld angegeben werden.

```
@('allee',STRASSE)
```

Die Funktion gibt die Position der Zeichenkette in dem Feld in Form einer Integer-Zahl aus. Ist die Zeichenkette nicht in dem Feld enthalten, so gibt sie eine Null aus.

Man kann daher mit dieser Bedingung herausfinden, ob im Feld STRASSE die Zeichenkette 'allee' enthalten ist:

```
@('allee',STRASSE) > 0
```

Diese Funktion ist immer dann sehr nützlich, wenn man einen Feldinhalt nicht genau kennt oder nur einen Teil eingeben möchte, um mehrere Datensätze zu selektieren.

Übung 8:

1. Formulieren Sie bitte die folgenden Aussagen mit Hilfe der dBASE-II-Operatoren in Bedingungen um:

   a. Gesucht wird der Kunde Helmut Schwarze aus der Akazienallee 27.
   b. Gesucht werden alle Kunden aus Duisburg (PLZ=4100).
   c. Gesucht wird der Kunde mit der Telefonnummer 71234.
   d. Gesucht werden alle Kunden, die in der Schillergasse oder dem Reiterweg wohnen und einen Umsatz größer null haben.

   Hilfe: Die Feldnamen entnehmen Sie bitte der Datei KUNDEN.

Suchen und Anzeigen von Daten

Wie immer bei Befehlen, die auf Dateien zugreifen, muß die gewünschte Datei zuerst geöffnet werden. Wir arbeiten mit der Datei KUNDEN und geben daher USE KUNDEN <RETURN> ein. Bis jetzt konnten wir uns über die Satznummer einzelne Sätze oder aber alle Sätze anzeigen lassen. Das Neue an dem Befehl LIST FOR ist, daß wir eine Bedingung angeben können.

Die Datenbank wird nach Datensätzen durchsucht, die die Bedingung erfüllen. Wenn wir alle Kunden mit dem Namen Schmidt anzeigen lassen wollen, geben wir LIST FOR NACHNAME='Schmidt' <RETURN> ein. DBASE II ver-

gleicht nun das Feld NACHNAME in jedem Datensatz mit der Zeichenkette 'Schmidt'. Dabei unterscheidet dBASE II zwischen Groß- und Kleinschreibung; der Ausdruck 'SCHMIDT' ist also für dBASE II nicht gleich 'Schmidt'.

Versuchen wir zum Beispiel, alle Kunden mit dem Vornamen Theo zu finden, dann geben wir ein:

```
.LIST FOR VORNAME='Theo' <RETURN>
```

Wenn Sie die Testdaten aus dem Kapitel 1, "Erfassen von Datensätzen", genauso eingegeben haben, wie diese abgedruckt sind, dann wird Theo nicht gefunden. Theo ist aber in den Datensätzen enthalten. Er ist nur nicht richtig geschrieben worden:

```
00001    1  Käufer          Otto       Ladengasse 9     5000 Köln 32
         99.99
00002    2  Meyer           theo       Eichenstr. 15    4100 Duisburg
0203-435612   999.78
00003    3  Maier           Udo        Kolpingstr. 14a  6050 Offenbach
069-3467       67.20
00004    4  Maiers          Trude      Bahnhofstr. 11   4300 Essen-West
0201-20316    117.98
00005    5  Hütter          Angelika   Sandfuhrstr. 34  4200 Oberhausen
0208-56129      0.00
00006    6  Hutter          Sabine     Heideweg 23      6790 Oberammergau
0122-4554     978.40
00007    7  Ende            Werner     Am Schlußlicht   5428 Endlichhofen
07142-568     789.50
```

Die Unterscheidung von Groß- und Kleinschreibung kann mit der !-Funktion abgeschaltet werden. Dann sind für dBASE II beispielsweise die folgenden Zeichenketten gleich:

THEO
Theo
theo
tHeo

Dies wird durch dBASE II realisiert, indem es zum Vergleich die hinter der Funktion angegebene Zeichenkette oder Variable in Großbuchstaben umwandelt. Daher muß die Vergleichszeichenkette groß geschrieben werden:

```
.LOCATE FOR !(VORNAME)="THEO" <RETURN>
```

Durch folgenden Befehlsaufbau kann man Theo ebenfalls finden:

```
.LIST FOR !(VORNAME)=!('Theo') <RETURN>
```

Bei dem folgenden Befehlsaufbau wird Theo nicht gefunden:

```
LIST FOR !(VORNAME)='Theo' <RETURN>.
```

Namen ändern

Die !-Funktion sollte generell bei der Suche nach Namen verwendet werden. Es kann leicht geschehen, daß bei der Eingabe eines Namens die CAPS LOCK-Taste versehentlich gedrückt oder ein Name klein geschrieben wird, da die SHIFT-Taste nicht stark genug angeschlagen wurde.

Die Suche nach einem bestimmten Datensatz kann auch mit dem LOCATE-Befehl durchgeführt werden. Als Ergebnis erhält man die Satznummer des gesuchten Satzes. Der Inhalt des Satzes wird nicht am Bildschirm ausgegeben!

Satznummer suchen

Dieser Befehl eignet sich vorzüglich dazu, die Satznummer eines Satzes zu finden, den man mit EDIT editieren möchte. Zur Anwendung des EDIT-Befehls muß die Satznummer des zu editierenden Satzes bekannt sein.

Da der Befehl eine Datei sequentiell durchsucht, das heißt Datensatz für Datensatz von der momentanen Position aus, ist er recht langsam. Bei einer Datei von 1000 Datensätzen benötigt der Befehl demnach 1000 Dateizugriffe, um den letzten Satz zu finden, dagegen nur einen Zugriff für den ersten Datensatz.

Die Suche sollte immer vom Anfang der Datei ausgehen, damit jeder gesuchte Satz gefunden wird. Wenn man sich nicht bereits dort befindet, muß der Dateizeiger mit GO TOP auf den Anfang der Datei gestellt werden, und man kann dann den Suchbefehl eingeben:

```
. GO TOP
. LOCATE FOR !(VORNAME)="THEO"
```

Wird ein Datensatz gefunden, erscheint diese Meldung:

```
. LOCATE FOR !(VORNAME)="THEO"
Satz: 00002
```

Enthält die durchsuchte Datei dagegen keinen Satz, der die Bedingung erfüllt, erfolgt die Meldung:

```
. LOCATE FOR !(VORNAME)="Theo"
Dateiende erreicht
```

CONTINUE

In einer größeren Datei kann es mehrere Datensätze geben, die die spezifizierte Bedingung erfüllen. Da die Suche nach dem ersten gefundenen Satz abbricht, muß sie wieder gestartet werden. Zur weiteren Suche müssen wir den Befehl CONTINUE <RETURN> eingeben:

```
. CONTINUE
Dateiende erreicht
```

Der Befehl CONTINUE setzt die unterbrochene Suche fort. Wird kein weiterer Satz gefunden, zeigt er das Dateiende an.

Den LOCATE-Befehl sollte man nicht auf indizierte Dateien anwenden, da er dann langsamer arbeitet.

FIND

Eine wesentlich schnellere Möglichkeit, gesuchte Datensätze zu finden, stellt der FIND-Befehl dar. Dieser Befehl arbeitet nur in nach dem Suchfeld bzw. Schlüssel indizierten Dateien. Sucht man nach einem bestimmten Nachnamen aus der Kundendatei, so muß diese nach dem Feld NACHNAME indiziert sein. Die gesuchte Zeichenkette wird in Anführungszeichen oder Hochkommata hinter dem Befehl angegeben:

```
. USE KUNDEN INDEX NAME
. FIND 'Schmidt'
Wert nicht gefunden.
```

Der gesuchte Name ist in der Datenbank nicht vorhanden. Versuchen wir, Herrn Maier zu finden:

```
. FIND 'Maier'
.
```

Es erfolgt keine Fehlermeldung, der gesuchte Name ist gefunden. Die Satznummer des gefundenen Satzes speichert dBASE II in der Systemvariablen # ab:

```
. ? #
      3
```

Läßt man sich die Variable # anzeigen, erhält man die Satznummer. Den ganzen Datensatz sehen wir mit dem DISPLAY-Befehl:

```
. DISPLAY
00003      4 Maiers          Trude       Bahnhofstr. 11   4300 Essen-West
 0202-20316      0.00
```

Durch den FIND-Befehl wird der Dateizeiger auf den gefundenen Satz gesetzt. Daher kann dieser Satz mit DISPLAY direkt angezeigt werden.

Wie wir sehen, haben wir nicht den Herrn Maier, sondern eine Frau Maiers gefunden. Der FIND-Befehl prüft, ob ein Feld mit der angegebenen Zeichenkette beginnt. Er überprüft nicht, ob eine vollständige Übereinstimmung besteht.

Auf den nächsten Datensatz zeigen

Existieren mehrere Datensätze, die die angegebene Zeichenkette enthalten, so müssen diese in einer indizierten Datei logisch hinter dem gefundenen Datensatz stehen. Auf den nächsten Datensatz zeigt man mit SKIP:

```
. SKIP
. DISPLAY
00002       4 Meyer          theo        Eichenstr. 15    4100 Duisburg
 0203-435612     0.00
```

Dieser Satz enthält keinen Nachnamen mehr, der mit Maier beginnt.

LIST FOR

Man kann auch mit dem LIST FOR-Befehl überprüfen, welche Datensätze den gewünschten Namen enthalten. In Verbindung mit FIND geschieht dies wesentlich schneller, als wenn LIST FOR allein verwendet wird:

```
. FIND "M"
. LIST FOR @("M",NACHNAME)
00003       4 Maiers         Trude       Bahnhofstr. 11   4300 Essen-West
 0202-20316      0.00
00002       4 Meyer          theo        Eichenstr. 15    4100 Duisburg
 0203-435612     0.00
```

Mit dem FIND-Befehl haben wir vorhin Frau Maiers gefunden, obwohl wir nach Herrn Maier gesucht haben. Suchen wir einen ganz bestimmten Datensatz, bei dem die angegebene Zeichenkette exakt mit dem Schlüsselfeld übereinstimmen soll, so müssen wir dies dBASE II mitteilen:

```
. SET EXACT ON
. FIND "Maier"
Wert nicht gefunden
.
```

SET EXACT ON

Der Befehl SET EXACT ON teilt dBASE II mit, daß nur Datensätze gesucht werden, bei denen eine exakte Übereinstimmung mit der vorgegebenen Zeichenkette besteht. Da in der Datei KUNDEN kein Herr Maier abgespeichert ist, kann kein Datensatz gefunden werden. Diesen Befehl hebt man mit SET EXACT OFF wieder auf, wenn man ihn nicht mehr benötigt.

Mit FIND können auch Zahlenwerte gefunden werden, diese müssen ebenfalls in Anführungszeichen angegeben werden:

```
. USE KUNDEN
. INDEX ON UMSATZ TO UMSATZ
. FIND "99.99"
. DISPLAY
00001     1  Käufer          Otto       Ladengasse 9     5000 Köln 32
             99.99
```

Makros

Anstelle einer Zeichenkette kann auch ein sogenannter Makro hinter dem FIND-Befehl angegeben werden. Ein Makro ist eine Speichervariable mit vorangesetztem &-Zeichen:

```
. USE KUNDEN INDEX UMSATZ
. STORE 99.99 TO WERT
. FIND &WERT
. DISPLAY
00001     1  Käufer          Otto       Ladengasse 9     5000 Köln 32
             99.99
```

Es ist dabei unerheblich, ob diese Variable alphanumerisch oder numerisch ist. Die Verwendung eines Makros anstelle einer Zeichenkette ist nur bei der Programmierung unter dBASE II wichtig. Wir werden den Makros daher später im vierten Teil dieses Buches noch einmal begegnen.

Kapitel 6

Editieren von Datensätzen

In diesem Kapitel werden folgende Themen behandelt:

Bearbeitung einzelner Datensätze
Bildschirmorientiertes Editieren
Änderungen in allen Datensätzen

Zu diesen Themen gehören die folgenden Befehle:

$()
BROWSE
CHANGE
EDIT
REPLACE

Bearbeitung einzelner Datensätze

Zu den grundlegenden Funktionen einer Datenbankverwaltung gehört, daß man jeden beliebigen Datensatz in der Datei leicht ändern kann. Es kommt gerade in einer Adressenverwaltung wie unserer Kundendatei häufig vor, daß sich die Anschrift eines Kunden ändert. Dann müssen wir diese Eingabe berichtigen können. Nehmen wir an, Herr Otto Käufer sei aus der Ladengasse 9 in die Schloßallee 111 umgezogen. Ein einzelner Datensatz wird mit dem EDIT-Befehl geändert. Es muß dazu die Satznummer des Satzes bekannt sein, den man ändern möchte. Bei zehn Kunden kann man diese durch einen einfachen DISPLAY-ALL oder LIST-Befehl herausfinden, indem man sich alle Datensätze anzeigen läßt. Für zehn Kunden lohnt sich aber die Anschaffung eines Computers nicht. Im Normalfall ist eine solche Datei mehrere hundert Datensätze groß. Also lassen wir dBASE II nach Herrn Käufer suchen und den gefundenen Satz anzeigen:

```
.USE KUNDEN
.LIST FOR NACHNAME='Käufer'
00001     1 Käufer         Otto       Ladengasse 9     5000 Köln 32
  99.99
```

Wie wir sehen, ist Herr Käufer in dem ersten Datensatz abgespeichert. Nun haben wir die benötigte Information, um den EDIT-Befehl anzuwenden:

```
EDIT 1 <RETURN>
```

Als Antwort bekommen wir von dBASE II eine Maske wie bei dem APPEND-Befehl angezeigt, nur mit dem Unterschied, daß die Daten von Satz 1 angezeigt werden:

```
Satznummer 00001
KUNDENNR   :0001:
NACHNAME   :Käufer          :
VORNAME    :Otto       :
STRASSE    :Ladengasse 9    :
PLZ        :5000:ORT        :Köln 32         :
TELEFON    :             :
UMSATZ     :99.99 :
```

Wie beim APPEND-Befehl wird der Cursor auch hier über Tastenkombinationen an die Stellen gesteuert, die geändert werden sollen. Am Ende des Abschnitts sind alle möglichen Tastenbefehle aufgeführt. Unsere Änderung können wir durchführen, indem wir mit <^X> oder wiederholtem Drücken von <RETURN> mit dem Cursor in das Feld STRASSE gelangen. Dort löschen wir mit <^Y> das ganze Feld und geben die neue Straße und Hausnummer ein:

```
Satznummer 00001
KUNDENNR   :0001:
NACHNAME   :Käufer          :
VORNAME    :Otto       :
STRASSE    :Schloßallee 111:
PLZ        :5000:ORT        :Köln 32         :
TELEFON    :             :
UMSATZ     :99.99 :
```

Da haben wir Glück gehabt, der Platz hat gerade ausgereicht. Mit der Eingabe <^W> speichern wir unsere Änderung ab. Sollten wir uns vertan haben, können wir entweder mit den Tastenbefehlen in der Maske ändern, oder aber mit <^Q> den EDIT-Befehl verlassen, ohne daß die vorgenommenen Änderungen gespeichert werden. Wenn wir mehrere Änderungen durchzuführen haben, können wir auch einfach EDIT <RETURN> eingeben und durch wiederholtes Drücken der RETURN-Taste oder von <^X> die Datensätze durchblättern und dort, wo notwendig, Änderungen vornehmen.

Wenn wir am Ende der Datei angelangt sind, fordert dBASE II uns auf:

```
Bitte Satznummer eingeben:
```

Hier müssen wir eine Satznummer eingeben. Wir können dann, wenn alle Änderungen gemacht sind, mit <^W> die Editierung beenden. Die Datendatei ist jetzt auf dem neuesten Stand, aber wir müssen auch an die Indexdateien denken. Sicherheitshalber sollte man bei der Eröffnung einer Datei die zugehörigen Indexdateien immer gleich mit angeben, dann werden diese auch mit aktualisiert. An Stelle von USE KUNDEN <RETURN> geben wir dann beispielsweise USE KUNDEN INDEX ON NAME; PLZ <RETURN> ein. Wir sollten alle Befehle der nachfolgenden Tabelle üben und beherrschen lernen. Für die Anfangszeit ist es sehr praktisch, die Tabelle zu kopieren und über der Tastatur auf den Rechner zu legen. So kann jede Tastenkombination leicht gefunden werden.

Hier ist die Zusammenstellung aller Tasten, die bei dem Befehl EDIT zur Editierung angewendet werden können:

Tastenkomb.	Entspricht	Wirkung
<^E>	<↑> oder <^A>	Der Cursor geht ein Feld nach oben. Im ersten Feld wird der vorige Satz angezeigt.
<^X>	<↓> oder <RETURN> oder <^F>	Der Cursor geht ein Feld nach unten. Im letzten Feld wird der nächste Satz angezeigt.
<^S>	<←> und <DEL>	Der Cursor geht ein Zeichen nach links.
<^D>	<→>	Der Cursor geht ein Zeichen nach rechts.
<^V>		Schaltet vom Überschreib- in den Einsprungmodus um.
<^G>	<CLR>	Löscht ein Zeichen.
<^Y>		Löscht ein Feld.
<^U>		Markiert den Satz zum Löschen.
<^C>		Schreibt die Änderung auf Diskette und zeigt den nächsten Datensatz an.
<^R>		Schreibt die Änderung auf Diskette und zeigt den vorherigen Datensatz an.
<^W>		EDIT wird verlassen, alle Änderungen werden abgespeichert.
<^Q>		Edit wird verlassen, ohne die Änderungen abzuspeichern

Die Cursor-Tasten können nur bei Verwendung des DIN-Zeichensatzes und der DIN-Tastatur wie angegeben verwendet werden!

Bildschirmorientiertes Editieren

Der zuvor behandelte EDIT-Befehl hat den Vorteil, daß alle Datenfelder – sofern es nicht mehr als 23 Felder sind – auf einen Blick zu überschauen sind. Von Nachteil ist, daß man immer nur den Inhalt von einem Datensatz auf dem Bildschirm sehen und verändern kann. Ein weiterer Nachteil dieses Befehls ist der Zwang, die Satznummer des Satzes zu kennen, den man ändern möchte.

Daher bietet sich zum Editieren eher der Befehl BROWSE an. Hier ist die Zusammenstellung aller Tasten, die bei dem Befehl BROWSE zur Editierung angewendet werden können:

Tastenkomb.	Entspricht	Wirkung
<^E>	<^A>	Der Cursor geht ein Feld nach links oder nach oben. Am Anfang des Fensters wird der vorige Satz angezeigt.
<^X>	<^F> oder <RETURN>	Der Cursor geht ein Feld nach rechts oder nach unten. Am Ende des Fensters wird der nächste Satz angezeigt.
<^S>		Der Cursor geht ein Zeichen nach links.
<^D>	<^L>	Der Cursor geht ein Zeichen nach rechts.
<^V>		Schaltet vom Überschreib- in den Einsprungmodus um.
<^B>		Zeigt ein weiteres Feld auf der rechten Seite des Fensters an.
<^G>		Löscht ein Zeichen.
<^Y>		Löscht ein Feld.
<^U>		Markiert den Satz zum Löschen.
<^C>		Der Cursor geht auf den nächsten Datensatz.
<^R>		Der Cursor geht auf den vorherigen Datensatz.
<^W>		BROWSE wird verlassen, alle Änderungen werden abgespeichert.
<^Q>		BROWSE wird verlassen, ohne die Änderungen abzuspeichern.

Im Gegensatz zum EDIT-Befehl zeigt dieser bis zu 19 Datensätze auf einmal an. Jeder Datensatz wird in einer Bildschirmzeile dargestellt. Deshalb werden

– mit einer Ausnahme – auch nur 80 Zeichen des Satzes am Bildschirm angezeigt. Wenn ein Feld eines Datensatzes länger als 80 Zeichen ist, dann wird das Feld, soweit notwendig, in den nächsten Zeilen abgebildet. Da wir bei diesem Befehl wie durch ein Fenster immer nur einen Ausschnitt eines längeren Datensatzes sehen, gibt es die Möglichkeit, die Felder anzugeben, die man editieren möchte. Läßt man diese Angabe weg, werden die Felder in der Reihenfolge angezeigt, in der sie im Datensatz stehen. Dies hört sich komplizierter an, als es in der Praxis ist. Wir eröffnen die Kundendatei – dabei eröffnen wir die Indexdateien immer mit – und beginnen mit dem Editieren:

```
.USE KUNDEN INDEX NAME,PLZ
BROWSE
```

Nachdem wir BROWSE <RETURN> eingegeben haben, bekommen wir einen Ausschnitt der Datei KUNDEN angezeigt:

```
Satznummer :00001
KUND NACHNAME------- VORNAME--- STRASSE-------- PLZ- ORT------------
   7 Ende            Werner     Am Schlußlicht  5428 Endlichhofen
   6 Hutter          Sabine     Heideweg 23     6790 Oberammergau
   5 Hütter          Angelika   Sandfuhrstr. 34 4200 Oberhausen
   1 Käufer          Otto       Schloßallee 111 5000 Köln 32
   3  Maier          Udo        Kolpingstr. 14a 6050 Offenbach
   4 Maiers          Trude      Bahnhofstr. 11  4300 Essen-West
   2 Meyer           theo       Eichenstr. 15   4100 Duisburg
```

Da wir die Indexdateien mit eröffnet haben, wird die Datei sortiert angezeigt. Wir können jetzt mit der RETURN-Taste von einem Feld zum anderen springen, bis wir das Feld erreicht haben, das wir ändern möchten. Sind wir ein Feld zu weit gesprungen, haben wir mit der Tastenkombination <^E> die Möglichkeit, an den Anfang des vorhergehenden Feldes zu gelangen. Wenn wir versuchen, über den Anfang des Fensters zu gelangen, wird – falls vorhanden – der vorhergehende Satz angezeigt.

Falls wir mit <RETURN> am Ende des Fensters sind, wird ein weiterer Satz angezeigt. An jeder Stelle, wo wir uns mit dem Cursor befinden, können wir Änderungen vornehmen. Mit <^U> kann sogar der Satz, auf dem wir uns gerade befinden, zur Löschung markiert werden. Dies wird durch die Anzeige "Gelöscht" in der obersten Zeile angezeigt. Wie den EDIT-Befehl können wir auch den BROWSE-Befehl durch die Tastenkombinationen <^W> oder <^Q> verlassen, je nachdem, ob die Änderungen abgespeichert werden sollen oder nicht.

Wie Ihnen bestimmt schon aufgefallen ist, sehen Sie das Feld TELEFON nicht. Der Datensatz ist zu lang, so daß nicht alle Felder auf einmal am Bildschirm dargestellt werden können. Fehlende Felder zeigt <^B> an, wenn wir uns im letzten sichtbaren Feld befinden. Dafür verschwinden dann eines oder mehre-

re Felder auf der linken Seite unseres Sichtfensters. Meistens ist es günstiger, sich schon vor dem Aufruf von BROWSE zu überlegen, welche Felder man ändern möchte. Diese Felder gibt man dann als Liste hinter dem Befehl an, wie das folgende Beispiel zeigt:

```
. BROWSE FIELDS NACHNAME,VORNAM,TELEFON
Satznummer :00007
NACHNAME------- VORNAME--- TELEFON-----
Ende            Werner     07142-568
Hutter          Sabine     0122-4554
Hütter          Angelika   0208-56129
Käufer          Otto
Maier           Udo        069-3467
Maiers          Trude      0201-20316
Meyer           theo       0203-435612
```

Änderung in allen Datensätzen

Mit den bisher besprochenen Editierfunktionen können Änderungen in allen Datenfeldern einer Datei durchgeführt werden. Der Befehl CHANGE läßt dagegen nur Änderungen in bestimmten Datenfeldern zu, die wir vorher angeben müssen. Wie bei allen Editierbefehlen muß zunächst die Datenbank eröffnet werden, in der editiert werden soll:

```
.USE KUNDEN <RETURN>
```

Jetzt können wir den CHANGE-Befehl ausprobieren:

```
.CHANGE <RETURN>
```

Als Antwort erhalten wir von dBASE II:

```
.CHANGE
Die FIELD-Angabe fehlt
CHANGE
Korrigieren und wiederholen? (J/N)
```

Wir haben den Befehl unvollständig eingegeben, denn wie schon gesagt, es muß angegeben werden, welches oder welche Felder man editieren möchte. Wir drücken <N> und geben den Befehl diesmal vollständig ein:

```
.CHANGE FIELD STRASSE
```

Diesmal war die Eingabe vollständig, und wir erhalten folgende Ausgabe:

```
.CHANGE FIELD STRASSE

Satz: 00001
```

```
STRASSE: Schloßallee 111
Verändern?
```

Bei der Frage "Verändern?" ist nicht danach gefragt, ob, sondern was geändert werden soll. Hier geben wir den Teil des Feldes ein, den wir ändern möchten. Zum Beispiel könnten wir aus der Schloßallee die Akazienallee machen:

```
.CHANGE FIELD STRASSE

Satz: 00001
STRASSE: Schloßallee 111
Verändern? Schloß
nach:     Akazien

STRASSE: Akazienallee 11
Verändern?
```

Der neue Feldinhalt wird von dBASE II angezeigt, da bei dieser Art der Editierung schnell eine Änderung passiert, die man nicht erwartet. Auch hier geschah etwas Unerwartetes. Da das Feld für die gewünschte Änderung nicht groß genug ist, hat dBASE II einfach das letzte Zeichen abgeschnitten. Es gibt auch Änderungen, die sich nicht in einem Schritt durchführen lassen, ohne das ganze Feld neu einzugeben. Daher wird von dBASE II nach jeder Änderung noch einmal gefragt, ob vielleicht weitere Eingaben zu machen sind. Erst wenn wir bei dieser Abfrage die RETURN-Taste drücken, ohne sonst etwas einzugeben, wird die Editierung beendet.

Wenn wir auf die Frage "Verändern?" mit der Eingabe <^Y> antworten, wird das Feld gelöscht.

Der CHANGE-Befehl wirkt ohne den Parameter ALL nur auf den aktuellen Datensatz, mit dem Parameter auf alle Datensätze. Da eine Editierung mit CHANGE wesentlich mehr Tastendrücke erfordert als mit EDIT oder BROWSE, hat er für die praktische Arbeit mit dBASE II eigentlich wenig Bedeutung. Er zeigt aber anschaulich, wie der wesentlich wichtigere REPLACE-Befehl funktioniert.

Der Befehl REPLACE ersetzt Feldinhalte in mehreren Datensätzen automatisch. Dies geschieht auf ähnliche Art, wie wir es mit CHANGE gerade manuell gemacht haben. Daher ist der CHANGE-Befehl eine gute Übung, um die Arbeitsweise von REPLACE verstehen zu lernen. Auch bei diesem Befehl muß die Datenbank eröffnet sein, mit der wir arbeiten möchten. Der Befehl wirkt ohne den Parameter ALL nur auf einen einzelnen Datensatz, bringt dann aber keine Vorteile gegenüber den Funktionen, die wir schon kennengelernt haben. Im Unterschied zum CHANGE-Befehl wird bei REPLACE immer der

gesamte Feldinhalt gegen die angegebene Zeichenkette ausgetauscht. Wir können aber auch Ergebnisse von Berechnungen, also Zahlenwerte gegen die aktuellen Feldinhalte austauschen. Praktische Anwendungen wären zum Beispiel:

- Durch Eingemeindung hat sich ein Ortsname geändert. Der neue Name soll gegen den alten Namen ausgetauscht werden.
- Die Vorwahl eines Ortsnetzes hat sich geändert und muß ausgetauscht werden.
- Der Mehrwertsteuersatz hat sich geändert. Daher müssen die Bruttoverkaufspreise in einer Artikeldatei entsprechend erhöht werden.
- Felder einer Datenbank sollen gelöscht werden.

Alle diese Beispiele haben gemeinsam, daß mehrere Datensätze betroffen sind und genau festlegbar ist, welche Veränderungen gemacht werden müssen.

Wenn wir zu Beginn eines neuen Jahres alle Felder UMSATZ unserer Datei KUNDEN löschen wollen, geht dies ganz einfach so:

```
. USE KUNDEN INDEX NAME,PLZ
. REPLACE ALL UMSATZ WITH 0 FOR UMSATZ > 0
00004 Ersetzung(en)
. LIST
00001     1  Käufer          Otto       Akazienallee 11 5000 Köln 32
              0.00
00002     2  Meyer             theo        Eichenstr. 15    4100 Duisburg
0203-435612     0.00
00003     3  Maier           Udo        Kolpingstr. 14a 6050 Offenbach
069-3467        0.00
00004     4  Maiers          Trude      Bahnhofstr. 11  4300 Essen-West
0201-20316      0.00
00005     5  Hütter          Angelika   Sandfuhrstr. 34 4200 Oberhausen
0208-56129      0.00
00006     6  Hutter            Sabine     Heideweg 23      6790 Oberammergau
0122-4554       0.00
00007     7  Ende              Werner     Am Schlußlicht   5428 Endlichhofen
07142-568       0.00
```

Die Löschung aller Felder UMSATZ war ja recht einfach.

Wie wollen wir aber auf diese Weise bestimmte Vorwahlen gegen andere austauschen? Wir haben kein eigenes Feld VORWAHL, sondern nur ein Feld TELEFON, das Vorwahl und Rufnummer durch einen Bindestrich getrennt

enthält. Wir müssen also einen Teil eines Feldes austauschen. Teile von Zeichenketten können wir mit der $-Funktion erhalten. Diese Funktion wird folgendermaßen angewandt:

```
$(Zeichenkette oder Variable oder Feld, Anfang, Länge)
```

Wenn sich beispielsweise die Vorwahl von Endlichhofen von 07142 in 07244 ändert, müßten wir in allen Feldern, wo diese Vorwahl vorkommt, die ersten fünf Zeichen ändern. Dazu wird das Feld TELEFON mit dem String '07142' plus den restlichen sieben Zeichen des Feldes verglichen. Die $-Funktion lautet:

```
'07142'+$(TELEFON,6,7)
```

Ist der Vergleich erfolgreich, ersetzen wir das Feld TELEFON mit Hilfe der $-Funktion:

```
'07244'+$(TELEFON,6,7)
```

Diese Ausdrücke setzen wir jetzt in den REPLACE-Befehl ein:

```
REPLACE ALL TELEFON WITH '07244'+$(TELEFON,6,7) FOR TELEFON='07142'+
$(TELEFON,6,7) <RETURN>
```

Umgangssprachlich würde diese Befehlszeile in etwa lauten: Ersetze alle Felder TELEFON mit '07244' plus seinen letzten sieben Stellen, wenn die Bedingung zutrifft, daß das Feld TELEFON gleich '07142' plus seinen letzten sieben Stellen ist!

Sehen wir uns an, wie der REPLACE-Befehl zusammen mit der $-Funktion gewirkt hat:

```
. USE KUNDEN INDEX NAME, PLZ.
REPLACE ALL TELEFON WITH '07244'+
$(TELEFON,6,7) FOR TELEFON='07142'+$(TELEFON,6,7)
00001 Ersetzung(en)
. LIST
00001    1  Käufer          Otto       Akazienallee 11 5000 Köln 32
               0.00
00002    2  Meyer          theo       Eichenstr. 15   4100 Duisburg
0203-435612      0.00
00003    3  Maier          Udo        Kolpingstr. 14a 6050 Offenbach
069-3467         0.00
00004    4  Maiers         Trude      Bahnhofstr. 11  4300 Essen-West
0201-20316       0.00
00005    5  Hütter         Angelika   Sandfuhrstr. 34 4200 Oberhausen
0208-56129       0.00
00006    6  Hutter           Sabine     Heideweg 23      6790 Oberammergau
0122-4554        0.00
00007    7  Ende             Werner     Am Schlußlicht   5428 Endlichhofen
07244-568        0.00
```

Im siebten Satz wurde die Vorwahl geändert. Es ist also mit REPLACE auch möglich, Teile eines Feldes zu ändern. Wir wollen jetzt am Beispiel einer Artikeldatei noch die Möglichkeit üben, Feldinhalte gegen Rechenergebnisse auszutauschen. Dies ist die Artikeldatei:

```
. USE ARTIKEL
. LIST
00001      1 Joystick de Luxe      12    75   19.90   49.00
00002      2 3 Zoll Disketten     120    50    6.00    9.90
00003      3 Colour Monitor         3     3  798.00  998.00
00004      4 Schutzhuelle          23    30    9.90   19.90
00005      5 Druckerkabel           2     1   24.00   49.00
00006      6 5 1/4 Zoll Disketten 200   150    3.96    4.90
```

In der letzten Spalte stehen die Bruttoverkaufspreise. Diese Preise enthalten 14 % Mehrwertsteuer. Wenn die Mehrwertsteuer erhöht wird, zum Beispiel auf 15 %, müssen alle Preise in dieser Artikeldatei geändert werden. Dies bedeutet eine Menge Rechenarbeit und kann bei einer größeren Datei stundenlang dauern.

Für solche Arbeiten haben wir dBASE II. Diese Sisyphusarbeit erledigt dBASE II in Sekunden. Zunächst müssen wir wissen, wie das Feld genau heißt, das wir ändern möchten.

Dazu verwenden wir den Befehl DISPLAY STRUCTURE:

```
. DISPLAY STRUCTURE
Strukturdaten für die Datei: A:ARTIKEL .DBF
Anzahl der Sätze: 00006
Datum der letzten Aktualisierung: 21/07/86
Primäre Datei
Feld  Name       Typ  Länge  Dez.st.
001      ARTIKELNR     N     004
002      ARTIKELBEZ    C     016
003      BESTAND       N     004
004      MINBESTAND    N     004
005      EK            N     006     002
006      VK            N     006     002
** Gesammt **                00041
```

Das Feld heißt VK. Die Rechenformel lautet:

$$\frac{VK}{114\%} * 115\%$$

Damit können wir unseren REPLACE-Befehl aufbauen:

```
.REPLACE ALL VK WITH VK/115*114  <RETURN>
```

Das Ergebnis lautet:

```
. REPLACE ALL VK WITH VK/115*114 <RETURN>
*** Überlauf im numerischen Feld
00006 Ersetzung(en)
. LIST
00001      1 Joystick de Luxe     12     75    19.90    49.42
00002      2 3 Zoll Disketten    120     50     6.00     9.98
00003      3 Colour Monitor        3      3   798.00     0.00
00004      4 Schutzhuelle         23     30     9.90    20.07
00005      5 Druckerkabel          2      1    24.00    49.42
00006      6 5 1/4 Disketten     200    150     3.96     4.94
```

Wir haben die Fehlermeldung "*** Überlauf im numerischen Feld" erhalten. Das Ergebnis einer Berechnung paßte nicht mehr in das Feld VK. Daher hat dBASE II dieses Feld mit null gefüllt. Der Colour Monitor, unser teuerster Artikel, kostet jetzt nichts mehr!

Numerische Felder sollten immer größer angelegt werden, als im Moment benötigt. Damit erspart man sich viel Arbeit. Um diesen Fehler zu beheben, müssen die Datensätze in eine andere Datei kopiert (COPY TO), dann die Dateistruktur geändert (MODIFY STRUCTURE) und die Datensätze zurückkopiert werden. Zuletzt ist dann noch mit dem Befehl DELETE FILE die Zwischendatei zu löschen und mit REPLACE der entsprechende Wert zu ändern. Die Daten, die in einem Feld standen, wo ein numerischer Überlauf stattfand, sind für uns verloren. Daher sollte von einer wichtigen und großen Datei immer eine Sicherheitskopie angelegt werden. Der Befehl REPLACE kann sehr viel Arbeit einsparen, aber auch eine Menge Arbeit bereiten, wenn wir als Ergebnis eine ganz oder teilweise zerstörte Datei erhalten.

Kapitel 7

Löschen von Datensätzen

In diesem Kapitel werden folgende Themen behandelt:

Markieren der zu löschenden Datensätze
Aufheben der Löschmarkierung
Löschen der markierten Datensätze
Löschen einer Datenbank

Zu diesen Themen gehören die folgenden Befehle:

DELETE
DELETE FILE
PACK
RECALL

Das Löschen von Datensätzen geschieht in zwei Schritten. Zunächst werden die Sätze mit dem DELETE-Befehl markiert und erst danach mit dem Befehl PACK physikalisch gelöscht. Diese Befehle können nur auf eine geöffnete Datei angewendet werden. Wir eröffnen also die Datei KUNDEN, wenn wir sie nicht schon in Gebrauch haben.

Markieren der zu löschenden Datensätze

Um einen bestimmten Satz zu löschen, zum Beispiel Satz 5, geben wir

```
.DELETE RECORD 5 <RETURN>
```

ein und haben den Satz damit markiert. Sehen wir uns mit

```
.LIST <RETURN>
```

die Datei KUNDEN an, dann stellen wir fest, daß hinter der Satznummer 5 ein Sternchen angezeigt wird:

```
. DELETE RECORD 5
00001 Löschung(en)
```

```
. LIST
00001     1  Käufer          Otto       Schloßallee 111 5000 Köln 32
          99.99
00002     2  Meyer           theo       Eichenstr. 15   4100 Duisburg
0203-435612   999.78
00003     3  Maier           Udo        Kolpingstr. 14a 6050 Offenbach
069-3467       67.20
00004     4  Maiers          Trude      Bahnhofstr. 11  4300 Essen-West
0201-20316    117.98
00005 *   5  Hütter          Angelika   Sandfuhrstr. 34 4200 Oberhausen
0208-56129      0.00
00006     6  Hutter          Sabine     Heideweg 23     6790 Oberammergau
0122-4554     978.40
00007     7  Ende            Werner     Am Schlußlicht  5428 Endlichhofen
07142-568     789.50
```

Dieses Sternchen ist die Löschmarkierung. Während markierte Sätze mit dem LIST- oder DISPLAY-Befehl angezeigt werden, berücksichtigt der SORT-Befehl diese bei der Sortierung nicht. Die sortierte Datei enthält die markierten Sätze nicht mehr.

Löschen der markierten Datensätze

Da es sehr umständlich wäre, wenn wir immer sortieren müßten, um die markierten Sätze zu löschen, lernen wir den Befehl PACK kennen. Geben Sie PACK <RETURN> ein, und sehen Sie, was passiert. Wir erhalten folgendes Ergebnis:

```
. PACK
PACK durchgeführt 00006 Sätze kopiert
```

Nach der Eingabe von LIST <RETURN> gibt es den fünften Satz nicht mehr:

```
. LIST
00001     1  Käufer          Otto       Ladengasse 9    5000 Köln 32
          99.99
00002     2  Meyer           theo       Eichenstr. 15   4100 Duisburg
0203-435612   999.78
00003     3  Maier           Udo        Kolpingstr. 14a 6050 Offenbach
069-3467       67.20
00004     4  Maiers          Trude      Bahnhofstr. 11  4300 Essen-West
0201-20316    117.98
00005     6  Hutter          Sabine     Heideweg 23     6790 Oberammergau
0122-4554     978.40
00006     7  Ende            Werner     Am Schlußlicht  5428 Endlichhofen
07142-568     789.50
```

Dieser Satz ist unwiederbringlich verloren. Wenn wir die Daten noch benötigen, müssen wir diesen Kunden neu erfassen. Wahrscheinlich fragen Sie sich,

ob mit dem DELETE-Befehl nicht auch mehrere Sätze auf einmal markiert werden können. Erinnern wir uns an die Möglichkeiten, die mit dem DISPLAY-Befehl verbunden sind. Es gibt den Parameter ALL, um alle Datensätze anzuzeigen. Diesen Parameter können wir auch beim DELETE-Befehl anwenden:

```
.DELETE ALL <RETURN>
.LIST <RETURN>
```

Nun sind alle verbliebenen Sätze markiert:

```
00001 *  1  Käufer          Otto      Ladengasse 9     5000 Köln 32
        99.99
00002 *  2  Meyer           theo      Eichenstr. 15    4100 Duisburg
0203-435612  999.78
00003 *  3  Maier           Udo       Kolpingstr. 14a  6050 Offenbach
069-3467      67.20
00004 *  4  Maiers          Trude     Bahnhofstr. 11   4300 Essen-West
0201-20316   117.98
00005 *  6  Hutter          Sabine    Heideweg 23      6790 Oberammergau
0122-4554    978.40
00006 *  7  Ende            Werner    Am Schlußlicht   5428 Endlichhofen
07142-568    789.50
```

Aufheben der Löschmarkierung

An diesem Beispiel können wir den RECALL-Befehl üben. Der Befehl RECALL entfernt die Löschmarkierung und läßt sozusagen unsere Datensätze wieder "auferstehen". Irrtümlich vorgenommene Löschungen können somit rückgängig gemacht werden. Dieser Befehl wirkt auf einzelne Sätze, auf alle Datensätze oder auf Datensätze, die von uns spezifizierte Bedingungen erfüllen. Mit der Eingabe RECALL 1 <RETURN> wird die Löschmarkierung des ersten Datensatzes entfernt:

```
. RECALL 1
00001 Reaktivierung(en)
. LIST
00001    1  Käufer          Otto      Ladengasse 9     5000 Köln 32
        99.99
00002 *  2  Meyer           theo      Eichenstr. 15    4100 Duisburg
0203-435612  999.78
00003 *  3  Maier           Udo       Kolpingstr. 14a  6050 Offenbach
069-3467      67.20
00004 *  4  Maiers          Trude     Bahnhofstr. 11   4300 Essen-West
0201-20316   117.98
00005 *  6  Hutter          Sabine    Heideweg 23      6790 Oberammergau
0122-4554    978.40
00006 *  7  Ende            Werner    Am Schlußlicht   5428 Endlichhofen
07142-568    789.50
```

Die Eingabe von RECALL ALL <RETURN> entfernt die Löschmarkierung von allen Datensätzen. Bestimmte Datensätze können mit Hilfe des FOR- sowie des WHILE-Parameters analog zum DELETE-Befehl wieder zur Bearbeitung freigegeben werden.

Üben wir dies mit den folgenden Eingaben:

```
.DELETE ALL <RETURN>
.RECALL ALL WHILE UMSATZ > 0 <RETURN>
.LIST

.DELETE ALL <RETURN>
.RECALL FOR NACHNAME='Schmidt' <RETURN>
.LIST

.DELETE ALL <RETURN>
.RECALL ALL FOR PLZ > '3999' .AND. < '5000' <RETURN>
.LIST
```

Die Löschmarkierung kann auch beim Editieren von Datensätzen mit dem EDIT- oder dem BROWSE-Befehl erfolgen. Durch <^U> markiert man den aktuellen Datensatz. In der obersten Bildschirmzeile erscheint die Meldung "Gelöscht".

Löschen einer Datenbank

Wenn die gesamte Datei nicht mehr benötigt wird, löschen wir diese mit dem Befehl DELETE FILE, gefolgt von dem Dateinamen:

```
. DELETE FILE TEST.DBF
```

Eine Datei, die gelöscht werden soll, darf nicht eröffnet sein, sonst erfolgt die Fehlermeldung:

```
Datei ist bereits eröffnet
```

In diesem Fall muß die Datei zunächst mit USE geschlossen werden, bevor man sie löschen kann. Die Löschung kann natürlich auch nur dann erfolgen, wenn die angegebene Datei existiert. Erhalten wir die Meldung "Datei wurde gelöscht", wurde der Befehl DELETE FILE ausgeführt, und die Datei ist nicht mehr vorhanden.

Mit diesem Befehl sollten wir daher sehr vorsichtig umgehen, denn die so gelöschte Datei läßt sich nicht mehr zurückholen. Während bei dem DELETE- und PACK-Befehl die Dateistruktur erhalten bleibt und nur die Datensätze gelöscht werden, ist nach Anwendung des Befehls DELETE FILE auch die Da-

teistruktur zerstört. Wir sollten diesen Befehl also nie anwenden, wenn wir eine Datei später noch einmal verwenden wollen.

Übung 9:

1. Löschen Sie aus der Datei ARTIKEL den 3., 5. und 6. Datensatz. Dabei ist zu berücksichtigen, daß sich bei einer Löschung die Satznummern ändern!
2. Löschen Sie den Artikel "Colour Monitor"!
3. Heben Sie die Löschmarkierung bei allen Datensätzen wieder auf!
4. Mit welchem Befehl werden Datensätze physikalisch gelöscht?
5. Wie kann man Datensätze während des EDIT-Befehls zur Löschung markieren?

Kapitel 8

Addieren und Zählen in einer Datenbank

In diesem Kapitel werden folgende Themen behandelt:

Addition von Datenfeldern
Zählen von Datensätzen

Zu diesen Befehlen gehören die folgenden Befehle:

COUNT
SET DELETE ON/OFF
SUM

Addition von Datenfeldern

In der Datei KUNDEN gibt es das numerische Feld Umsatz. Die Addition aller Umsätze in unserer Datei KUNDEN ergibt den Gesamtumsatz. Eine solche Summierung eines Feldes über alle Datensätze können wir mit dem Befehl SUM durchführen. Auch dieser Befehl kann nur auf eine eröffnete Datenbank angewendet werden:

```
.USE KUNDEN INDEX NAME,PLZ <RETURN>
```

Der SUM-Befehl funktioniert auch, wenn die Indexdateien nicht mit eröffnet werden. Sicherheitshalber sollten wir diese aber immer mit eröffnen. Die Summe der Umsatzfelder erhalten wir mit folgender Eingabe:

```
.SUM UMSATZ <RETURN>
```

Wir erhalten die Ausgabe:

```
. SUM UMSATZ <RETURN>
1234.78
```

Der Befehl wirkt auf alle nicht zur Löschung markierten Sätze. Den Parameter ALL benötigen wir bei diesem Befehl nicht. Am Beispiel der Datei ARTIKEL werden wir weitere Möglichkeiten sehen, die der Befehl bietet. Dies ist der Inhalt der Datei ARTIKEL:

```
SaNr. ArtNr. Artikelbez.       Bestand Minb.   EK       VK
-------------------------------------------------------------
00001      1 Joystick de Luxe      12     75   19.90    49.00
00002      2 3 Zoll Disketten     120     50    6.00     9.90
00003      3 Colour Monitor         3      3  798.00   998.00
00004      4 Schutzhuelle          23     30    9.90    19.90
00005      5 Druckerkabel           2      1   24.00    49.00
00006      6 5 1/4 Disketten      200    150    3.96     4.90
```

Die Datei enthält die numerischen Felder BESTAND, MINBESTAND, EK und VK. Durch die Summierung aller EK-Felder multipliziert mit dem Bestand erhalten wir den Warenwert. Dafür müssen zunächst zwei Felder multipliziert werden, bevor das Ergebnis aufsummiert wird:

```
.SUM ALL BESTAND*EK <RETURN>
```

Das Ergebnis lautet:

```
. SUM ALL BESTAND*EK
 4420.50
.
```

Unsere Ware ist also 4420,50 DM wert. Immer dann, wenn der Mindestbestand für eine bestimmte Ware unterschritten wird, muß dieser Artikel nachbestellt werden. In unserem Fall müßten 63 Joysticks de Luxe und 7 Schutzhüllen bestellt werden. Mit Hilfe des Parameters FOR können wir dBASE II berechnen lassen, wieviel Kapital wir für die Bestellung benötigen:

```
.SUM ALL (MINBESTAND-BESTAND)*EK FOR MINBESTAND > BESTAND <RETURN>
```

Übersetzt heißt dies in etwa:

Addiere die Differenz von MINBESTAND und BESTAND multipliziert mit dem EK bei allen Datensätzen auf, für die gilt, daß MINBESTAND größer als BESTAND ist.

Das Ergebnis lautet:

```
. SUM ALL (MINBESTAND-BESTAND)*EK FOR MINBESTAND > BESTAND
 1323.00
.
```

Eine weitere interessante Größe, die wir unserer Datenbank entnehmen können, ist der mögliche Gewinn, den wir mit den vorhandenen Artikeln erzielen können. Dazu muß der Nettoverkaufspreis, d.h. der Verkaufspreis vermindert um die Mehrwertsteuer aus dem Feld VK ermittelt werden. Von dem Nettoverkaufspreis ziehen wir dann den EK ab und erhalten den Gewinn pro Artikel. Den Gesamtgewinn erhalten wir durch die Multiplikation dieses Wertes mit dem Bestand für alle Artikel. Dies geht mit dBASE II ebenfalls sehr einfach:

```
. SUM ALL (VK/1.14-EK)*BESTAND
 1110.81
```

Wenn wir den Gewinn durch den Warenwert dividieren und dann mit 100 multiplizieren, erhalten wir daraus die sogenannte Umsatzrentabilität:

```
. ? 1110.81/4420.50*100
        25.12
```

Die Umsatzrentabilität beträgt also 25.12 %. Es lassen sich aus dieser Datei noch wesentlich mehr betriebswirtschaftlich wichtige Faktoren gewinnen. Zum Beispiel kann die Umsatzrentabilität pro Artikel oder Artikelgruppe berechnet und ausgedruckt werden, oder man kann zählen, bei wie vielen Artikeln der Mindestbestand unterschritten ist. Zählen kann der SUM-Befehl allerdings nicht. Im nächsten Abschnitt werden wir daher den Befehl COUNT kennenlernen.

Übung 10:

1. Wie erhalten wir den Gesamtumsatz für alle Kunden aus dem Postleitzahlgebiet 4000 bis 4999?
2. Wie hoch ist die Umsatzrentabilität für den Artikel "Joystick de Luxe"?
3. Wir wollen unseren Mindestbestand für alle Artikel um 30 % erhöhen. Wieviel Kapital benötigen wir, um die benötigte Ware zu bestellen?

Zählen von Datensätzen

In einigen Fällen ist es sehr hilfreich, wenn man die Datensätze zählen kann, die eine spezifizierte Bedingung erfüllen. Die einfachste Anwendung für den COUNT-Befehl besteht darin, alle Datensätze in einer Datei zu zählen.

Dabei werden Sätze, die eine Löschmarkierung enthalten, berücksichtigt. Nach Eingabe des Befehls SET DELETE ON werden diese Sätze von COUNT

nicht mehr mitgezählt. Die Differenz der Zählergebnisse mit und ohne gelöschte Sätze ergibt deren Anzahl.

Diese Anwendung probieren wir aus:

```
. USE KUNDEN INDEX NAME, PLZ
. DELETE ALL FOR PLZ > '5999'
00002 Löschung(en)
. COUNT
Anzahl der Sätze = 00007
. SET DELETE ON
. COUNT
Anzahl der Sätze = 00005
.
```

Mit dem DELETE-Befehl markieren wir zum Teil zwei Datensätze. Der COUNT-Befehl zeigt uns an, daß sich insgesamt 7 Datensätze in der Datei befinden, wovon 5 Datensätze keine Löschmarkierung aufweisen.

Diese Methode eignet sich hervorragend dazu, eine Datei dahingehend zu überprüfen, ob, und wenn ja, wie viele Sätze markiert sind.

Durch SET DELETE OFF <RETURN> wird die Anzeige der markierten Sätze wieder aktiviert.

Der Befehl COUNT FOR läßt sich mit einer beliebigen Bedingung kombinieren. Wir können zum Beispiel überprüfen, wie viele Kunden im Postleitzahlgebiet 4000 bis 4999 wohnen:

```
.COUNT FOR PLZ > '3999' .AND. PLZ < '5000' <RETURN>
```

Als Ergebnis erhalten wir:

```
. COUNT FOR PLZ > '3999' .AND. PLZ < '5000'
Anzahl der Sätze = 00003
```

Übung 11:

1. Wie wird das Zählen markierter Sätze unterdrückt?
2. Welche Befehlszeile wird benötigt, um alle Kunden mit dem Nachnamen "Maier" oder "Meier" zu zählen?
3. Wie erhalten wir die Anzahl der Kunden mit einem Umsatz von mehr als 5000,- DM?

Kapitel 9

Rechnen mit dBASE II

In diesem Kapitel werden folgende Themen behandelt:

Die dBASE-II-Rechenfunktionen
Rechnen mit arithmetischen Operatoren
Rangfolge der arithmetischen Operatoren
Rechengenauigkeit
Einsatz der Integer-Funktion
Die numerischen Speichervariablen
Der Variablenname
Speichern von Werten in numerischen Speichervariablen
Anzeige von Variablen
Löschen von Variablen

Zu diesen Themen gehören die folgenden Befehle und Operatoren:

\+
\-
/
*
()
?
DISPLAY MEMORY
INT
RELEASE
STORE

Die dBASE-II-Rechenfunktionen

Rechnen mit arithmetischen Operatoren

Mit dBASE II können wir wie mit einem Taschenrechner Berechnungen durchführen. Fangen wir mit der Berechnung von 1+1 an. Dazu geben wir ?

1+1 <RETURN> ein. Sofort erscheint das Ergebnis in der nächsten Zeile:

```
. ? 1+1
  2
.
```

Das "?" ist für dBASE II der Befehl, das Rechenergebnis anzuzeigen. Die eigentliche Rechnung folgt dahinter. Rechnen mit dBASE II funktioniert ähnlich wie bei BASIC. Diese Aufgabe hätten wir wahrscheinlich auch im Kopf rechnen können, daher versuchen wir jetzt etwas Schwierigeres. Angenommen, wir hätten mit 4 Personen Lotto gespielt und 3.667.123,42 DM gewonnen. Wir wollen ausrechnen, wieviel jeder Spieler der Tippgemeinschaft erhält. Die Division von Zahlen wird mit dem Zeichen "/" durchgeführt:

```
? 3667123.42/4 <RETURN>
```

Das Komma muß bei dBASE II als Punkt eingegeben werden. Als Ergebnis erhalten wir:

```
. ? 3667123.42/4
   916780.85
.
```

Die übrigen Grundrechenarten Multiplikation und Subtraktion werden mit einem Stern bzw. einem Minuszeichen dargestellt:

```
. ? 4*12
   48
. ? 1234.5-12.31
  1222.19
.
```

Rangfolge von arithmetischen Operatoren

Bei einer längeren Rechenaufgabe wie

$$\frac{5+512}{7*13}-21$$

müssen wir einige Regeln beachten, damit wir das richtige Ergebnis erhalten. Von dBASE II werden zunächst alle Punkt- vor den Strichrechnungen durchgeführt. Stehen zwei Punktrechnungen nebeneinander wie im Beispiel 512/7*13, wird zuerst der linke Ausdruck "512/7" berechnet und das Ergebnis daraus mit 13 multipliziert. Das Ergebnis dieses Ausdrucks wäre:

```
. ? 512/7*13
  949
.
```

Wenn wir eine andere Reihenfolge möchten, müssen wir eine Klammer setzen:

```
. ? 512/(7*13)
    5
.
```

Am Beispiel der obigen Aufgabe wiederholen wir das Gelernte.

```
. ? 5+512/(7*13)-21
   -10
.
```

Bei dieser Rechnung hat dBASE II folgende Schritte durchgeführt

Schritt 1: 7*13=91	(Klammerinhalte werden zuerst berechnet)
Schritt 2: 512/91=5	(Punkt- vor Strichrechnung)
Schritt 3: 5+5=10	(Von links wird begonnen, wenn keine Punktrechnung oder Klammer zu berechnen ist)
Schritt 4: 10–21=–11	

Das Ergebnis ist in zweierlei Hinsicht interessant. Erstens haben wir einen Fehler gemacht, denn wir hätten eine Klammer um den Zähler "5+512" des Bruches setzen müssen, um die obige Aufgabe zu lösen. Statt dessen haben wir die Aufgabe

$$5 + \frac{512}{7*13} - 21$$

gelöst. Zweitens ergibt die Rechnung mit dBASE II in einzelnen Schritten ein anderes Ergebnis als die Eingabe der gesamten Formel.

Rechengenauigkeit

Dieses verblüffende Ergebnis liegt an der Rechengenauigkeit von dBASE II. Wenn wir mit dBASE II Zahlen dividieren, rechnet es nur mit so vielen Stellen hinter dem Komma, wie die Zahlen Nachkommastellen besitzen. Daher ergibt

```
. ? 512/91
    5
```

dagegen aber

```
. ? 512.000000/91
       5.626373
```

ein Ergebnis mit 6 Nachkommastellen. Durch die Rundungsfehler ergibt sich das um –1 verschiedene Resultat unserer Berechnungen. Ein weiteres Problem beim Rechnen mit dBASE II tritt auf, wenn man mit langen Zahlen rechnet. Die folgenden beiden Eingaben liefern ebenfalls verschiedene Ergebnisse:

```
. ? 10.00000000/3
        3.33333333
. ? 10.0000000000/3
          3.3333333300
```

Bei der Berechnung werden nur die ersten zehn Stellen berücksichtigt. Geben wir mehr als zehn Stellen für eine Dezimalzahl ein, dann werden die übrigen Stellen im Ergebnis als Nullen ausgegeben. Wir sollten also bei sehr großen Zahlen und bei Zahlen mit vielen Nachkommastellen vorsichtig sein.

Einsatz der Integer-Funktion

Wenn für uns die Nachkommastellen nicht wichtig sind, da wir nur den ganzteiligen Anteil eines Ergebnisses benötigen, können wir die dBASE-II- Funktion INT verwenden. Wir haben im Kapitel "Erstellen einer Dateistruktur" gesehen, wie die maximale Anzahl von Datensätzen berechnet werden kann, die auf einer Diskette Platz haben. Das Ergebnis dieser Berechnung muß eine ganze Zahl sein, da ein Datensatz entweder in der ganzen Länge oder gar nicht abgespeichert wird. Daher können wir für diese Berechnung die INT-Funktion einsetzen:

```
? INT(184320/79)
  2333
```

Das Ergebnis ist allerdings in diesem Fall das gleiche, als wenn wir die INT-Funktion weggelassen hätten, da dBASE II uns keine Nachkommastellen ausgibt, wenn wir keine Zahl mit Nachkommastellen in der Aufgabe haben.

Bevor wir die Variablen kennenlernen, sollte die nachfolgende Übung gelöst werden.

Übung 12:

1. Die nachfolgenden Aufgaben sollen mit dBASE II gerechnet werden. Das Ergebnis soll auf zwei Stellen hinter dem Komma genau sein.

a. $\dfrac{123 * \frac{12}{4}}{1 + \frac{7}{2}}$ b. $6{,}13 * \dfrac{\frac{12+4}{128}}{7} + 1{,}235$ c. $\dfrac{10}{3}$ d. $\dfrac{18}{4} + \dfrac{12}{26}$

2. Wie lautet das ganzzahlige Ergebnis der nachfolgenden Aufgabe?

$$1{,}23 * 12345 + \frac{5{,}217}{678{,}21}$$

Die numerischen Speichervariablen

Im letzten Abschnitt haben wir gelernt, wie man mit dBASE II rechnen kann. Das Ergebnis der Rechnung wird am Bildschirm angezeigt. Für einige Rechnungen ist es jedoch sehr praktisch, wenn man das Ergebnis abspeichern kann, um es später wieder anzusehen oder in einer weiteren Rechnung zu verwenden. Zu diesem Zweck können in dBASE II sogenannte Variablen definiert werden. Es gibt numerische und alphanumerische Variablen.

In numerischen Variablen werden Zahlen und in alphanumerischen Variablen Zeichenketten, die in Hochkommata eingeschlossen sind, gespeichert. An dieser Stelle wird nur auf die numerischen Variablen eingegangen, da man mit alphanumerischen Variablen nicht rechnen kann. Ohne Variablen zu kennen, haben wir schon mit ihnen gearbeitet, denn die Felder einer Datei sind ebenfalls Variablen. Man nennt sie Feldvariablen. Am folgenden Beispiel erkennen wir, wie eine Feldvariable angesprochen wird:

```
. USE KUNDEN
. ? UMSATZ
    99.99
.
```

Die Datei KUNDEN wurde eröffnet und das Feld Umsatz des ersten Satzes angezeigt. Dieses Feld ist gleichzeitig eine Variable, mit der wir rechnen können. Wenn wir die Mehrwertsteuer für diesen Umsatz berechnen möchten, können wir folgendermaßen verfahren:

```
. STORE UMSATZ * .14 TO MWST
 13.9986
```

Der Befehl STORE speichert das Ergebnis der Rechnung UMSATZ * .14 in der Variablen MWST ab und zeigt es am Bildschirm an.

Solange wir den Computer nicht ausschalten oder der Variablen MWST einen anderen Wert zuweisen, können wir uns dieses Ergebnis immer wieder anzeigen lassen:

```
. ? MWST
    13.9986
```

Der Variablennamen

Diese Art von Variablen nennt man Speichervariablen. Da diese nur im Hauptspeicher des Computers und nicht wie die Feldvariablen zusätzlich auf der Diskette gespeichert sind, geht ihr Inhalt verloren, wenn der Computer ausgeschaltet wird.

Wir dürfen insgesamt 64 Variablen gleichzeitig im Hauptspeicher haben, wobei jede Variable maximal 254 Zeichen lang sein kann. Alle Variablen zusammen dürfen aber unabhängig von ihrer Anzahl maximal 1536 Zeichen im Hauptspeicher belegen.

Der Name einer Variablen kann wie eine Feldbezeichnung maximal 10 Zeichen lang sein und Buchstaben, Ziffern und Doppelpunkte enthalten. Am Anfang eines Variablennamens steht immer ein Buchstabe. Der Name sollte so gewählt werden, daß er gut verständlich und nicht zu verwechseln ist.

Anzeige von Variablen

Wie viele Variablen wir zur Zeit benutzen und wieviel Platz diese belegen, zeigt uns der Befehl DISPLAY MEMORY an:

```
. DISPLAY MEMORY
MWST            (N)    13.9986
** Gesammt **          01 Variablen benutzt  00007 Bytes belegt
```

Löschen von Variablen

Sollten wir eine Speichervariable nicht mehr benötigen, können wir sie löschen. Dazu gibt es den Befehl RELEASE, der eine einzelne oder alle Variablen auf einmal löscht. Die Variable MWST können wir somit mit

```
RELEASE MWST <RETURN>
```

oder

```
RELEASE ALL <RETURN>
```

löschen. Wir sollten darauf achten, alle nicht benötigten Variablen direkt zu löschen, um die Grenze von 64 Variablen nicht zu überschreiten.

Übung 13:

1. Welche der nachfolgenden Namen sind zulässige Variablennamen?

   a. 1A
   b. ZEHN LEERSTELLEN
   c. WILLI
   d. KONTO:NR
   e. :TELEFON
   f. Umsatz

2. Wie viele Variablen dürfen gleichzeitig vorhanden sein?
3. Welcher Befehl löscht Variablen?
4. Mit welchem Befehl zeigt man alle Speichervariablen an?

Teil IV

Programmieren mit dBASE III

Zwischeninhaltsverzeichnis

Kapitel 1

Planung eines Programms

Entwurf einer Datenbank
Planung von Bildschirmmasken
Normierte Programmierung
Testen eines Programms

Beginnt man ohne Planung mit dem Programmieren, so werden später häufig Änderungen notwendig. Häufiges Ändern in Programmen führt zur Unübersichtlichkeit, es entstehen die sogenannten "Spaghettiprogramme".

Um eine solche Entwicklung zu verhindern, wurden Methoden entwickelt wie die Ist-Soll-Analyse, der Programmablaufplan oder das Struktogramm, die zu- sammen mit den Dateistrukturen, den Bildschirmmasken, den Programmlisten und einem Benutzerhandbuch eine brauchbare Dokumentation dessen liefern, was ein Programm leistet und wie es aufgebaut ist.

Es würde den Rahmen dieses Buches sprengen, diese Techniken zu vermitteln. Es werden daher einige organisatorische Hinweise und Hilfsmittel beschrieben, die es ermöglichen, ein Programm einfach zu planen, ohne diese Methoden zu kennen.

Entwurf einer Datenbank

Bevor wir ein Programm schreiben, sollten wir uns zunächst fragen, was das Programm später leisten soll. Da dBASE II ein Datenbanksystem ist, eignet es sich vor allem dazu, Dateien zu verwalten. Wir legen somit als erstes fest, welche Daten wir abspeichern wollen. Nehmen wir als Beispiel eine Adressendatei. Welche Daten müssen dort abgespeichert werden? Schreiben wir auf, was uns dazu einfällt:

Nachname
Vorname
Strasse

Postleitzahl
Ort
Telefon

Diese Angaben reichen für private Zwecke im allgemeinen aus. Ein Geschäftsmann würde bei einer Kundenkartei noch weitere Daten erfassen:

Kundennummer	Rechnungsadresse:	Lieferadresse:
Nachname		
Vorname	Strasse	Strasse
Telefon privat	Postleitzahl	Postleitzahl
Telefon dienstlich	Ort	Ort
Umsatz		
Kredit bis		

An diesen Beispielen sehen wir, daß die Planung einer Datei sich daran orientieren muß, welche Daten benötigt und wie diese ausgewertet werden.

Nachdem wir die Felder einer Datei festgelegt haben, müssen wir entscheiden, wie lang jedes Feld werden soll. Dann wird der sogenannte Feldtyp festgelegt, also bestimmt, ob es sich um ein numerisches, alphanumerisches oder logisches Feld handelt. Felder, die Ziffern enthalten, mit denen aber nicht gerechnet wird (z.B. die Postleitzahl) sollten zur Vermeidung von Fehlern alphanumerisch deklariert werden.

Als Hilfe zur Planung befindet sich ein Entwurfsformular für Dateien im Anhang. Außer den Angaben Feldbezeichnung, Feldtyp und Feldlänge kann man dort vermerken, ob ein Feld als Schlüssel dienen soll.

Eine Datenbank besteht ja nicht nur aus einer Datendatei, sondern auch aus einer oder mehreren Indexdateien. Eine Datendatei sollte aus maximal 25 Feldern bestehen. Eine große Datei wird leicht unübersichtlich. In einigen Fällen benötigen wir also mehr als eine Datendatei. Als Beispiel für eine etwas komplexere Aufgabenstellung wollen wir uns eine Rechnungsschreibung ansehen.

Dazu betrachten wir die bisherige Arbeitsweise in einem Unternehmen ohne Computer. Diese Betrachtung nennt man Ist-Analyse. Der derzeitige Arbeitsablauf wird untersucht, um daraus die Vorgaben für ein Soll-System zu schaffen. In der Übungsfirma werden die Kundenadressen mit einem Karteikasten verwaltet. Werden Rechnungen geschrieben, sucht sich die Sekretärin die Adresse des Kunden heraus. Bei einem neuen Kunden füllt sie eine neue Karteikarte aus und sortiert diese ein.

Dieser Karteikasten könnte durch die Datei KUNDEN ersetzt werden. Die Vorgaben für die Dateistruktur richten sich nach den Daten, die auf den Kar-

teikarten stehen. Mit Hilfe einer geeigneten Indexdatei nimmt dBASE II der Sekretärin die Sortierarbeit ab.

Die Bestellung des Kunden, die auf einem Auftragsblock steht, enthält pro Artikel die Artikelnummer, die Artikelbezeichnung, den Einzelpreis und die Anzahl der bestellten Artikel.

Die Daten über jeden Artikel hat sich der Verkäufer mühsam aus Katalogen bzw. Mikrofilm herausgesucht. Außerdem muß er im Lager nachfragen, ob der betreffende Artikel vorhanden ist oder erst bestellt werden muß und wie lange dies gegebenenfalls dauert. Dann ermittelt er pro Position den Gesamtpreis, indem er die bestellte Menge mit dem Einzelpreis multipliziert. Zuletzt kann er die Gesamtsumme ermitteln und die Rechnung sowie einen Lieferschein ausschreiben.

Ein Computer kann hier viel lästige Routinearbeit einsparen. Alle Artikel können in der Datei ARTIKEL erfaßt werden. Der Verkäufer kann so in Sekunden sehen, ob der bestellte Artikel auf Lager ist. Die Bestellungen gibt der Verkäufer direkt in den Computer ein. Dazu muß er die Kundennummer, die Artikelnummern der bestellten Artikel und die jeweilige Menge eingeben.

Sie werden vielleicht fragen, warum er die Kundennummer und nicht den Namen des Kunden eingibt? Die Kundennummer ist kürzer als der Name. Außerdem ist sie eindeutig, da jede Nummer nur einmal vorkommt. Die bloße Eingabe eines Nachnamens wie "Schmidt" oder "Meier" könnte dazu führen, daß der falsche Kunde die Rechnung erhält.

Diese Daten werden vom Computer in einer Bestelldatei abgelegt. Damit sind alle benötigten Daten gespeichert, um Rechnungen auszudrucken. Die notwendige Rechenarbeit (Summen, Berechnung der MwSt) sowie den Rechnungsdruck erledigt dBASE II.

Wir benötigen also folgende Dateien:

1. Kundendatei
2. Artikeldatei
3. Bestelldatei

Bevor man an die Erstellung der notwendigen Programme geht, muß zunächst überlegt werden, ob die gestellte Aufgabe mit dem Schneider-Computer lösbar ist. Dazu berechnet man die zu erwartende Größe der Dateien. Dies geschieht, indem die Anzahl der Kunden und Artikel ermittelt werden, die zu diesem Betrieb gehören, und diese dann jeweils mit der Satzlänge der Dateien KUNDEN und ARTIKEL multipliziert werden. Ein Satz der Kundendatei ist

79 Bytes und ein Satz der Datei ARTIKEL 44 Bytes lang (s. S. 32 und 33). Dazu wird der Speicherplatz für die dBASE-II-Programme, die Bestelldatei und die benötigten Indexdateien addiert:

Anzahl vorhandener Kunden	:	400	==>	400	*	79	Bytes	=	31600	Bytes
Indexdatei Kundennummer	:	400	==>	400	*	9	Bytes	=	3600	Bytes
Indexdatei Nachname	:	400	==>	400	*	20	Bytes	=	8000	Bytes
Anzahl vorhandener Artikel	:	1000	==>	1000	*	44	Bytes	=	44000	Bytes
Indexdatei Artikelnummer	:	1000	==>	1000	*	9	Bytes	=	9000	Bytes
Anzahl Bestellungen/Tag	:	100	==>	100	*	58	Bytes	=	5800	Bytes
Programmgröße	:								10000	Bytes
							Summe		112000	Bytes

Nach Abzug der dBASE-Systemprogramme verbleiben uns etwa 115000 Bytes auf einer 3-Zoll-Diskette, so daß diese Aufgabe sogar mit einem Diskettenlaufwerk zu bewerkstelligen ist. Für eine solche Anwendung sollte man dennoch über zwei Diskettenlaufwerke verfügen. Die hier zugrunde gelegten Satzlängen entstammen den Beispieldateien in diesem Buch. Sie sind für den gewerblichen Gebrauch meist nicht ausreichend dimensioniert. Zudem sollte bei der Planung die angestrebte Entwicklung des Unternehmens berücksichtigt werden. Was ist, wenn man in einem Jahr 1000 Kunden und 2000 Artikel verwalten muß?

Anzahl vorhandener Kunden	:	1000	==>	1000	*	79	Bytes	=	79000	Bytes
Indexdatei Kundennummer	:	1000	==>	1000	*	9	Bytes	=	9000	Bytes
Indexdatei Nachname	:	1000	==>	1000	*	20	Bytes	=	20000	Bytes
Anzahl vorhandener Artikel	:	2000	==>	2000	*	44	Bytes	=	88000	Bytes
Indexdatei Artikelnummer	:	2000	==>	2000	*	9	Bytes	=	18000	Bytes
Anzahl Bestellungen/Tag	:	300	==>	300	*	58	Bytes	=	17400	Bytes
Programmgröße	:					10000	Bytes			
							Summe		241400	Bytes

In diesem Fall kann der Schneider-Computer die gestellte Aufgabe nur noch mit zwei angeschlossenen Diskettenlaufwerken lösen. Dabei müssen die Dateien und Programme recht geschickt auf die beiden Disketten verteilt werden, damit das System überhaupt noch läuft.

Wahrscheinlich würde sich dieses System in der Praxis nicht bewähren; denn damit das System ordnungsgemäß funktioniert, müssen die Dateien KUNDEN und ARTIKEL gepflegt werden. Das bedeutet: Neue Kunden werden erfaßt, alte Adressen geändert, Lieferungen von Artikeln und neue Artikel müssen ebenfalls eingegeben werden. Dazwischen versucht der Verkäufer die 300 Bestellungen – bei einem 8-Stunden-Arbeitstag alle zwei Minuten eine Bestellung – einzugeben. Es gibt für eine Computeranwendung nicht nur eine technische Grenze! Bei kleineren Unternehmen und im privaten Bereich läßt sich jedoch mit dBASE II und dem Schneider-Computer viel erreichen.

Planung von Bildschirmmasken

Im interaktiven Befehlsmodus haben wir bei den Befehlen APPEND und EDIT sogenannte Bildschirmmasken kennengelernt. Diese Masken dienen dazu, dem Benutzer die Größe von Datenfeldern anzuzeigen und die Ein- sowie Ausgabe von Daten auf bestimmte Felder am Bildschirm zu begrenzen.

Während diese Masken bisher von dBASE II erzeugt wurden, müssen wir als Programmierer eine solche Maske häufig selbst erstellen.

Im Kapitel 3.2 wird gezeigt, wie mit Hilfe des Programms EINGABE eine Maske zur Eingabe von Adressen erzeugt wird.

Die Programmierzeit wird entscheidend verkürzt, wenn eine Maske auf einem Blatt Papier geplant wurde, bevor mit dem Programmieren begonnen wird. Dazu dient das entsprechende Formular im Anhang G. In den Formularkopf trägt man den Programmnamen, Maskennamen, Autor und das Datum zur Dokumentation ein.

Dann wird der statische Teil der Maske mit dem Bleistift entworfen, um den Entwurf leicht ändern zu können:

```
Programm: EINGABE   Maske: ERFASSUNG    Autor: M.-A. Beisecker    Datum: 30/07/1986

   10        20        30        40        50        60        70        80
1
2
3
4
5
6
7            Satznummer
8            KUNDENNUMMER
9            NACHNAME
10           VORNAME
11           STRASSE
12           POSTLEITZAHL
13           WOHNORT
14           TELEFON
15
16
17
18
19
20
21
23
24
```

Der statische Teil der Maske wird einmal am Bildschirm ausgegeben. Er ändert sich so lange nicht, bis der Bildschirm gelöscht oder eine andere Maske angezeigt wird.

Dagegen sind die angezeigten oder abgefragten Felder variabel. Jede Stelle eines numerischen Feldes wird im Formular mit einer 9, jede Stelle eines alphanumerischen Feldes mit einem X gekennzeichnet.

Die Maske des Programms EINGABE sieht somit folgendermaßen aus:

```
Programm: EINGABE   Maske: ERFASSUNG    Autor: M.-A. Beisecker    Datum: 30/07/1986

   10        20        30        40        50        60        70        80
1
2
3
4
5
6
7          Satznummer 99999
8          KUNDENNUMMER 9999
9          NACHNAME     xxxxxxxxxxxxxx
10         VORNAME      xxxxxxxxxx
11         STRASSE      xxxxxxxxxxxxxx
12         POSTLEITZAHL xxxx
13         WOHNORT      xxxxxxxxxxxx
14         TELEFON      xxxxxxxxxxxx
15
16
17
18
19
20
21
23
24
```

Zur Hervorhebung kann man einzelne Bildschirmfelder blinkend, invers oder unterstrichen darstellen. Dies wird auf dem Formular mit den Zeichen -, * und ' angedeutet. In der Legende des Formulars stehen einige Beispiele.

Das Maskenentwurfsblatt gehört zur Dokumentation eines Programms. Nicht jede Maske ist so einfach wie diese hier.

Haben Sie erst einmal etwas Programmiererfahrung gesammelt, werden Sie die Masken aufwendiger gestalten und feststellen, wieviel Zeit das Formular bei der Umsetzung einer Maske in dBASE-Befehle einspart.

Normierte und strukturierte Programmerstellung

Als Programmieranfänger schreibt man ein Programm zunächst in einem sogenannten Pseudocode. Größere Aufgaben werden dazu in kleinere Programme unterteilt, dann die groben Funktionen in diesen Programmen beschrieben. Diese grobe Beschreibung wird schrittweise verfeinert (Top-Down-Verfahren), bis alle gewünschten Programmfunktionen integriert sind.

Eine Rechnungsschreibung ist beispielsweise ein größeres Programmierproblem. Sie enthält eine Adreßverwaltung, eine Artikelverwaltung, eine Bestellverwaltung und den Rechnungsdruck.

Wir greifen einen dieser vier Bereiche zur weiteren Bearbeitung heraus. Die Adreßverwaltung soll es ermöglichen, Adressen neu aufzunehmen, zu ändern, zu löschen und Adreßaufkleber auszudrucken.

Damit der Benutzer zwischen diesen Funktionen wählen kann, benötigt er ein Menü. Daraus ergibt sich eine weitere Verfeinerung in 5 kleinere Bereiche. Diese Teilbereiche sind überschaubar, so daß man mit dem Schreiben des Pseudocodes beginnen kann.

Der Pseudocode für das Menüprogramm lautet:

```
Schalte alle Befehlsmeldungen am Bildschirm aus!
Öffne die Datei KUNDEN!
Speichere Null in der Variablen AUSWAHL!
Wiederhole die folgenden Befehle!
   Zeige ein Auswahlmenue am Bildschirm an!
   Frage den Benutzer, welche Funktion er wünscht!
   Speichere das Ergebnis in der Variablen AUSWAHL!
   Ist die Variable AUSWAHL = Eins, dann
      starte ein Programm zur Eingabe von Kundenadressen!
   Ist die Variable AUSWAHL = Zwei, dann
      starte ein Programm zur Änderung von Kundenadressen!
   Ist die Variable AUSWAHL = Drei, dann
      starte ein Programm zur Löschung von Kundenadressen!
   Ist die Variable AUSWAHL = Vier, dann
      starte ein Programm zum Druck von Adressaufklebern!
   Ist die Variable AUSWAHL = Null, dann
      Schalte alle Befehlsmeldungen wieder ein!
      Schließe die Datei KUNDEN!
      Beende das Programm!
Ende der Wiederholung!
```

Der Pseudocode ist nicht genormt, er kann beliebig formuliert werden. Man beschreibt die Aktionen, die der Computer durchführen soll, in eigenen Worten. Befehle, die wiederholt durchlaufen werden, rückt man etwas ein, um dies besser erkennen zu können.

Zu Beginn eines Programms unterbindet man die Bildschirmausgabe der Ergebnisse von Befehlen. Die direkte Ausgabe eines Ergebnisses könnte die Bildschirmmaske zerstören. Dann wird die benötigte Datei eröffnet, und verwendete Variablen vorbesetzt.

Nach diesen Vorbereitungen beginnt das eigentliche Programm. In diesem Fall wird in Abhängigkeit der Benutzereingabe eines der vier Unterprogramme aufgerufen oder das Programm beendet.

Am Ende des Programms wird die Unterdrückung der Ergebnisse wieder rückgängig gemacht, die geöffnete Datei geschlossen und das Programm abgebrochen. Wir sollten uns merken, daß die letzte Zeile eines Programms nicht immer das logische Programmende darstellt!

Man sollte versuchen, alle Programme nach diesem Schema ablaufen zu lassen. Die Variablennamen sollten kurz aber memotechnisch gewählt werden, und man sollte sich angewöhnen, für gleiche Aufgaben immer denselben Variablennamen zu vergeben. Zum Beispiel könnte man in allen Menüprogrammen die Variable AUSWAHL zur Aufnahme der Benutzerauswahl verwenden.

Da die Variablennamen auf zehn Stellen begrenzt sind, sollte man sich ein Verzeichnis verwendeter Abkürzungen anlegen und sich immer streng an dieses Verzeichnis halten. Im folgenden sind einige gängige Abkürzungen für Variablennamen oder Teile von Variablennamen aufgeführt:

STR	=	Strasse
PLZ	=	Postleitzahl
ART	=	Artikel
BEST	=	Bestell(ung)
KUN	=	Kunde
VERW	=	Verwaltung
LST	=	Liste
DRU	=	Druck
ADR	=	Adresse

Nachdem man den Pseudocode erstellt hat, kann man an die Umsetzung dieses Codes in dBASE-Befehle herangehen. Dazu schreibt man die dBASE-Befehle neben oder unter die Pseudobefehlszeilen:

```
Schalte alle Befehlsmeldungen am Bildschirm aus!
SET TALK OFF
Öffne die Datei KUNDEN!
USE KUNDEN
Speichere Null in der Variablen AUSWAHL!
STORE 0 TO AUSWAHL
```

Für die Codierung von Bildschirmmasken nimmt man die Maskenformulare zu Hilfe. Speichervariablennamen entnimmt man der Dateistruktur. Ist das Programm fertig codiert, gibt man es in den Computer ein und testet es aus.

Testen eines Programms

Es gibt zwei Hauptgruppen an Fehlern, die beim Programmieren auftreten können. Die erste Gruppe sind die Syntaxfehler. Diese Fehler entstehen zum Beispiel durch die falsche Schreibweise eines Befehls oder durch die unzulässige Kombination von Befehlen. Solche Fehler werden von dBASE II nach dem Starten des Programms erkannt und zur Verbesserung angeboten. Sind alle Syntaxfehler verbessert, lernt man die zweite Gruppe der logischen Fehler kennen. Diese Fehler kann dBASE II nicht erkennen. Logische Fehler erkennt man daran, daß das Programm etwas anderes bewirkt, als man erwartet. Solchen Fehlern begegnet man am besten mit einem Schreibtischtest.

Dazu schreibt man auf einem Blatt Papier alle in einem Programm vorkommenden Variablen in Tabellenform auf und geht dann das Programm Befehl für Befehl durch, wobei man die aktuellen Variablenwerte auf dem Blatt einträgt. So kann man relativ schnell feststellen, ob man irgendwo einen Fehler gemacht hat.

Noch schneller geht dieser Test, wenn man dBASE II zu Hilfe nimmt. Bevor das zu testende Programm gestartet wird, schaltet man mit

```
SET STEP ON <RETURN>
```

die Testhilfefunktion ein. Jetzt startet man das Programm, das nach Abarbeitung von einem Befehl unterbrochen wird. Die Variableninhalte kann man sich dann mit DISPLAY MEMORY <RETURN> ansehen.

Als Beispiel folgen hier die Eingaben, die zum Test der ersten fünf Befehlszeilen des Programms BESTVERW, das im Kapitel 9.2 beschrieben ist, benötigt werden:

```
. SET STEP ON
. DO BESTVERW
Schrittweise y:=Schritt, n:=Tastaturbef., ESC:=abbrechenY
Schrittweise y:=Schritt, n:=Tastaturbef., ESC:=abbrechenN
. DISPLAY MEMORY
** Gesammt **     00 Variablen benutzt  00000 Bytes belegt
Schrittweise y:=Schritt, n:=Tastaturbef., ESC:=abbrechenY
Schrittweise y:=Schritt, n:=Tastaturbef., ESC:=abbrechenN
. DISPLAY MEMORY
** Gesammt **     00 Variablen benutzt  00000 Bytes belegt
```

```
Schrittweise y:=Schritt, n:=Tastaturbef., ESC:=abbrechenY
Schrittweise y:=Schritt, n:=Tastaturbef., ESC:=abbrechenN
. DISPLAY MEMORY
SCHLEIFE    (L)   .T.
** Gesammt **    01 Variablen benutzt  00002 Bytes belegt
Schrittweise y:=Schritt, n:=Tastaturbef., ESC:=abbrechenY
Schrittweise y:=Schritt, n:=Tastaturbef., ESC:=abbrechenN
. DISPLAY MEMORY
SCHLEIFE    (L)   .T.
** Gesammt **    01 Variablen benutzt  00002 Bytes belegt
Schrittweise y:=Schritt, n:=Tastaturbef., ESC:=abbrechenY
Schrittweise y:=Schritt, n:=Tastaturbef., ESC:=abbrechenN
. DISPLAY MEMORY
SCHLEIFE    (L)   .T.
** Gesammt **    01 Variablen benutzt  00002 Bytes belegt
Schrittweise y:=Schritt, n:=Tastaturbef., ESC:=abbrechen\
```

Mit der Taste ESC wird dieser Programmodus abgebrochen und mit SET STEP OFF <RETURN> die Testhilfe ausgeschaltet.

Da man nicht sieht, welcher Befehl gerade bearbeitet wird, sollte man einen aktuellen Ausdruck des Programms vor sich liegen haben. Um einen Programmausdruck zu erhalten, verläßt man dBASE mit dem QUIT-Befehl und befindet sich auf der CP/M-Ebene. Arbeitet man mit einem Diskettenlaufwerk, legt man jetzt die CP/M-Diskette in das Laufwerk ein und drückt <^P> zum Starten des Druckers. Hat man zwei Laufwerke, muß man sich auf dem Diskettenlaufwerk befinden, auf dem das komplette CP/M-Betriebssystem ist und ebenfalls <^P> drücken. Dann gibt man TYPE [NO PAGE] B:BEST-VERW.CMD ein. Nach der Aufforderung müssen diejenigen, die mit einem Diskettenlaufwerk arbeiten, die CP/M-Diskette durch die dBASE-Diskette aus- wechseln. Nun erhält man einen Ausdruck des Programms. Zum Schluß gibt man wiederum <^P> ein, damit der Drucker ausgeschaltet wird.

Kapitel 2

Erstellen einer Befehlsdatei

In diesem Kapitel werden folgende Themen behandelt:

Schreiben eines Programms mit dem dBASE-II-Editor
Kommentar in einem Programm
Kommentarausgabe auf dem Bildschirm
Starten eines Programms

Zu diesen Themen gehören die folgenden Befehle:

*
DO
MODIFY COMMAND
NOTE
REMARK

Bis jetzt haben wir Befehle immer direkt hinter dem dBASE-Prompt eingegeben und sofort ein Ergebnis am Bildschirm erhalten. Diesen "interaktiven Befehlsmodus" verlassen wir und gehen über zum "Programmiermodus". Die Befehle, die wir früher direkt eingegeben haben, schreiben wir jetzt in eine sogenannte Befehlsdatei. Nach dem Aufruf dieser Datei werden die Befehle von dBASE II gelesen und in der Reihenfolge bearbeitet, wie wir sie in die Befehlsdatei geschrieben haben. Zu den schon bekannten Befehlen kommen noch einige hinzu, die nur in Verbindung mit dem Programmiermodus sinnvoll sind.

Schreiben eines Programms mit dem dBASE-II-Editor

Eine Befehlsdatei kann mit jedem Textverarbeitungsprogramm, zum Beispiel mit WordStar, erstellt werden. Hier werden wir zunächst den eingebauten Editor verwenden:

```
MODIFY COMMAND TEST <RETURN>
```

Der Editor wird mit MODIFY COMMAND, gefolgt von dem gewünschten Programmnamen, aufgerufen. Ein Programmname darf maximal 8 Zeichen lang sein. Er wird von dBASE um den Namenszusatz ".CMD" (engl.: command = Befehl) erweitert. Auf der Diskette finden wir unser erstes Programm demnach unter dem Namen TEST.CMD.

Nachdem die obige Befehlszeile eingegeben ist, erscheint die Meldung:

```
Neue Datei
```

Diese Meldung teilt uns mit, daß ein Programm mit dem Namen TEST noch nicht vorhanden ist. Dann wird die oberste Bildschirmzeile invers dargestellt. Hier beginnt man mit der Eingabe. Geben Sie bitte das folgende Programm in den Rechner ein:

```
*************************************************************************
* PROGRAMM: TEST    AUTOR:                              DATUM: TT.MM.JJ *
*************************************************************************

NOTE    Ein Sternchen oder der Befehl NOTE am Anfang einer Zeile
NOTE    kennzeichnen einen Kommentar.

REMARK Hurra! Mein erstes Programm läuft .
```

Die Funktion des Programms werden wir später noch genauer untersuchen. Zunächst werden wir alle Möglichkeiten kennenlernen, die uns dBASE II zum Editieren des Programms bietet.

Eine Programmierzeile darf nicht länger als 77 Stellen sein. Ab der 78. Stelle wird die Befehlszeile von dBASE II nicht bearbeitet! Ein Programm sollte höchstens 4000 Bytes lang sein. Längere Programme werden vom Editor nicht vollständig abgespeichert.

Es gibt keine Befehle, um Zeichenketten zu suchen oder Textbausteine zu verschieben. Der eingebaute Editor ist zwar durchaus verwendbar, aber es empfiehlt sich, zum Editieren ein Textverarbeitungsprogramm, z.B. WordStar, zu verwenden.

Beim Schreiben eines Programms kann schnell etwas vergessen oder ein Fehler begangen werden. In der aktuellen Zeile wird ein Fehler einfach überschrieben. Der Einfügemodus, der mit <^V> ein- und auch wieder ausgeschaltet wird, ermöglicht es, Text in die aktuelle Zeile einzufügen. Er schaltet sich automatisch aus, wenn man die Zeile verläßt.

Die Cursortasten sollten nie verwendet werden! Anstelle der Cursortasten verwenden wir <^S> bzw. <^D>, um ein Zeichen nach links bzw. rechts zu gelan-

gen. Mit <^E> bzw. <^X> wechseln wir in die vorherige bzw. nächste Zeile über. Der Befehl <^X> läßt sich auch durch <RETURN> ersetzen. Eine ganze Zeile fügt man mit <^N> ein und löscht sie mit <^T>. Den Inhalt einer Zeile löscht man mit <^Y>.

Alle möglichen Tastenkombinationsbefehle sind in einer Tabelle am Ende des Kapitels aufgeführt.

Nachdem wir das Programm richtig und vollständig eingegeben haben, speichern wir es mit <^W> ab.

Starten eines Programms

Nun ist das Programm fertig und kann gestartet werden:

```
.DO TEST <RETURN>
```

Als Ergebnis erhalten wir:

```
. DO TEST
'Hurra! Mein erstes Programm laeuft.
```

Kommentar in einem Programm

Die Ausgabe dieser Zeile ist die einzige Funktion des Programms. Ein Sternchen bzw. der Befehl NOTE stehen vor einem Kommentar.

Der Kommentar soll dem Programmierer helfen, sich in einem Programm zurechtzufinden. Am Anfang eines Programms sollte der Kommentar den Programmnamen, den Autor, das Datum der Programmierung, das Datum der letzten Änderung, eine kurze Beschreibung der Programmfunktion sowie Erläuterungen zu verwendeten Variablen und Makros enthalten.

Kommentarausgabe auf dem Bildschirm

Die Zeile hinter dem REMARK-Befehl ist ebenfalls Kommentar, der aber für den Anwender des Programms bestimmt ist.

Dieser Kommentar wird am Bildschirm angezeigt, wenn das Programm läuft. Er wird meist zur Fehlersuche verwendet oder zur Anzeige des Programmanfangs sowie des Programmendes.

Hier ist die Zusammenstellung aller Tasten, die man bei dem Befehl Modify COMMAND zur Editierung wenden kann:

Tastenkomb.	Entspricht	Wirkung
<^E>	<^A>	Der Cursor geht eine Zeile nach oben.
<^X>	<RETURN> oder <^F>	Der Cursor geht eine Zeile nach unten.
<^S>		Der Cursor geht ein Zeichen nach links. An der ersten Stelle der Zeile geht er eine Zeile nach oben.
<^D>		Der Cursor geht ein Zeichen nach rechts. An der letzten Stelle der Zeile geht er eine Zeile nach unten.
<^V>		Schaltet vom Überschreib- in den Einfügemodus um.
<^G>	<^L>	Löscht ein Zeichen.
<^Y>		Löscht den Inhalt einer Zeile.
<^T>		Löscht eine Zeile.
<^N>		Fügt eine neue Zeile ein.
<^C>		Schiebt den Bildschirm eine halbe Seite hoch.
<^R>		Schiebt den Bildschirm eine halbe Seite herunter.
<^W>		MODIFY COMMAND wird verlassen, alle Änderungen werden abgespeichert.
<^Q>		MODIFY COMMAND wird verlassen, ohne die Änderungen abzuspeichern.

Kapitel 3

Gestalten einer Bildschirmmaske

In diesem Kapitel werden folgende Themen behandelt:

Zeilenweise Ein- und Ausgabe
Ein- und Ausgabe von Zeichenketten
Umwandlung eines Strings in einen numerischen Wert
Eingaben verschiedenen Typs
Ausgabe von Zeichen
Bildschirmorientierte Ein- und Ausgabe
Erstellen einer Bildschirmmaske
Das Programm EINGABE ersetzt APPEND
Gestaltung von Formaten
Einfache Ausgabe eines Menüs

Zu diesen Themen gehören die folgenden Befehle:

?
@
ACCEPT
APPEND BLANK
ENDTEXT
GET
INPUT
PICTURE
READ
SAY
TEXT
VAL

Zeilenweise Ein- und Ausgabe

In den meisten Programmen ist es notwendig, während des Programmablaufs Daten vom Benutzer zu erfragen. Denken wir zum Beispiel an ein Menü, in

dem der Benutzer zwischen verschiedenen Programmteilen wählen kann. Zu einer solchen Abfrage gehört eine Erklärung für den Benutzer, die am Bildschirm ausgegeben wird. Diese nennt man eine Benutzerführung. Ein Programm wird ja meistens nicht vom Programmierer selbst, sondern von Menschen bedient, die sich mit dem Computer und dem Programminhalt weniger gut auskennen.

Ein- und Ausgabe von Zeichenketten

Der Befehl ACCEPT ermöglicht es, eine Erklärung am Bildschirm zu geben und eine Zeichenkette abzufragen. Diese Zeichenkette wird in einer Variablen abgespeichert. Wenn die Variable noch nicht vorhanden ist, wird sie vom ACCEPT-Befehl angelegt. Mit Hilfe des ACCEPT-Befehls können nur alphanumerische Variablen verarbeitet werden. Eine vorhandene numerische Variable wird von ACCEPT als alphanumerische Variable neu angelegt!

Die Anwendung des Befehles ist denkbar einfach:

```
ACCEPT "Wie heissen Sie?" TO NAME <RETURN>
```

Um diesen Befehl auszuprobieren, benötigen wir kein Programm. Die Befehlszeile kann so eingegeben werden. Nachdem wir die RETURN-Taste gedrückt haben, erhalten wir folgende Ausgabe:

```
Wie heissen Sie?:
```

Der ACCEPT-Befehl setzt hinter jede Ausgabe einen Doppelpunkt. Geben wir hier einen Namen ein, wird er in der Variablen NAME abgespeichert:

```
Wie heissen Sie?:Beisecker
```

Wir können dies leicht mit dem Befehl DISPLAY MEMORY überprüfen:

```
. DISPLAY MEMORY
NAME         (C)  Beisecker
** Gesammt **      01 Variablen benutzt  00010 Bytes belegt
```

Auch wenn wir eine Zahl eingeben, wird diese als Zeichenkette in einer alphanumerischen Variablen abgelegt:

```
. ACCEPT "Geben Sie eine Zahl ein " TO ZAHL
Geben Sie eine Zahl ein: 1000
. DISPLAY MEMORY
NAME         (C)  Beisecker
ZAHL         (C)   1000
** Gesammt **      02 Variablen benutzt  00016 Bytes belegt
```

Umwandlung eines Strings in einen numerischen Wert

Wie wir sehen, wurde die Zeichenkette mit der führenden Leerstelle abgespeichert. Mit einer alphanumerischen Variablen können keine Rechnungen durchgeführt werden. Um mit der Zeichenkette in der Variablen ZAHL rechnen zu können, muß diese in einen numerischen Wert umgewandelt und in einer numerischen Variablen abgespeichert werden. Diese Umwandlung können wir mit dem Befehl VAL vornehmen:

```
STORE VAL(ZAHL) TO N:ZAHL <RETURN>
```

Das Ergebnis der Umwandlung wird in der Variablen N:ZAHL abgespeichert. An dieser Stelle muß die neue Variable N:ZAHL definiert werden! Versucht man, einen Variableninhalt zu verändern und dann in sich selbst abzuspeichern, wird der Inhalt gelöscht. Sehen wir uns die Variablen und deren Inhalte noch einmal an:

```
. DISPLAY MEMORY
NAME         (C)  Beisecker
ZAHL         (C)   1000
N:ZAHL       (N)    1000
** Gesammt **      03 Variablen benutzt  00023 Bytes belegt
```

Die neue numerische Variable N:ZAHL kann zu Berechnungen verwendet werden. So kann man auf einem Umweg einen numerischen Wert aus einer ACCEPT-Eingabe erhalten. Die Abfrage numerischer Werte programmiert man dennoch praktischerweise mit dem Befehl INPUT. Dieser Befehl erkennt den eingegebenen Datentyp selbst und legt, wenn notwendig, eine entsprechende Variable an. Dabei müssen alphanumerische Eingaben in Anführungszeichen gesetzt werden:

```
. INPUT "Wie heissen Sie?" TO NAME
Wie heissen Sie?:Beisecker
Syntaxfehler, Eingabe bitte wiederholen
:"Beisecker"
Beisecker
```

Wenn wie hier die Anführungszeichen vergessen wurden, erfolgt eine Fehlermeldung, und die Eingabe muß noch einmal korrekt erfolgen. Der Befehl eignet sich besonders zur Eingabe numerischer Werte:

```
. INPUT "Geben Sie eine Zahl ein " TO ZAHL
Geben Sie eine Zahl ein: 1000
  1000
. DISPLAY MEMORY
NAME         (C)  Beisecker
ZAHL         (N)    1000
N:ZAHL       (N)    1000
** Gesammt **      03 Variablen benutzt  00024 Bytes belegt
```

Ohne die lästige Umwandlung haben wir sofort eine numerische Variable. Logische Variablen können ebenfalls mit dem INPUT-Befehl erfaßt werden. Dabei dürfen die Buchstaben T, Y, F oder N ohne Anführungszeichen eingegeben werden. Bei Eingabe von T oder Y wird in der logischen Variablen True (d.h. Wahr), ansonsten False (d.h. Unwahr) abgespeichert. Als Beispiel geben wir einmal folgendes ein:

```
INPUT "Geben Sie einen logischen Wert ein " TO LOGO <RETURN>
```

Als Ergebnis erhalten wir:

```
. INPUT "Geben Sie einen logischen Wert ein " TO LOGO
Geben Sie einen logischen Wert ein :T
.T.
. DISPLAY MEMORY
NAME          (C)  Beisecker
ZAHL          (N)    1000
N:ZAHL        (N)    1000
LOGO          (L)  .T.
** Gesammt **      04 Variablen benutzt  00026 Bytes belegt
```

Wie wir sehen, wurde in der Variablen LOGO .T. (True) abgespeichert. Eine einfache Ausgabe der Variableninhalte erhalten wir mit dem ?-Befehl. Ohne Zusatz bewirkt der Befehl einen Zeilenvorschub auf dem Bildschirm oder Drucker. Gibt man hinter dem Befehl eine in Anführungszeichen gesetzte Zeichenkette oder eine Variable an, so wird diese ausgegeben:

```
. ? NAME
Beisecker
. ? ZAHL
       1000
. ? "Zeichenkette"
Zeichenkette
```

Die bisher behandelten Befehle geben eine Variable oder Zeichenkette immer in einer neuen Zeile aus. Dabei kann nicht bestimmt werden, wo die Ausgabe genau erfolgen soll. Die Möglichkeit, an jeder Position des Bildschirms eine Ein- oder Ausgabe zu machen, werden wir im nächsten Abschnitt kennenlernen.

Übung 14:

1. Welchen Variablentyp erzeugt der ACCEPT-Befehl?
2. Wozu eignet sich der INPUT-Befehl besonders?
3. Mit welchem Befehl werden alphanumerische in numerische Variablen umgewandelt?
4. Welcher der behandelten Befehle erzeugt nur eine Ausgabe?

Bildschirmorientierte Ein- und Ausgabefunktionen

Befehle wie APPEND oder EDIT geben eine Bildschirmmaske mit Eingabefeldern aus, wobei man sich innerhalb der Eingabefelder frei bewegen kann. Während sich der Inhalt der Eingabefelder beliebig ändern läßt, bleibt die Maske bestehen. Der Bildschirm rollt auch nicht zeilenweise nach oben, wie es bei Ausgaben durch ACCEPT, INPUT oder ? geschieht.

Erstellen einer Bildschirmmaske

Wir werden nun eine solche Maske mit einem Programm erstellen. Mit Hilfe dieser Maske sollen die Daten für einen Kunden der Datei KUNDEN erfaßt werden. Dazu sollten wir die Maske zunächst auf einem Formular planen; denn um eine Maske zu programmieren, ist es notwendig, die jeweilige Zeile und Spalte zu kennen, in der ein Feld anfängt.

Das Formular zur Maskenplanung in Anhang G ist dabei eine wertvolle Hilfe. Die Zeilen und Spalten sind durchnumeriert, so daß man die Feld-Koordinaten leicht ablesen kann, nachdem man das Feld eingezeichnet hat. Bei der Planung sollte man eine Maske nur mit dem Bleistift zeichnen, damit Änderungen leicht gemacht werden können. Die folgende Maske soll in der Mitte des Bildschirms erscheinen:

```
                (SPALTE 25)

(ZEILE 07)      Satznummer          99999
                KUNDENNUMMER :9999:
                NACHNAME            :XXXXXXXXXXXXXXX:
                VORNAME             :XXXXXXXXXX:
                STRASSE             :XXXXXXXXXXXXXXX:
                POSTLEITZAHL        :XXXX:
                WOHNORT             :XXXXXXXXXXXX:
                TELEFON             :XXXXXXXXXXXX:
                UMSATZ              :999.99:
```

Die Angaben in den Klammern sind Kommentare; sie geben die Position der ersten Ausgabe am Bildschirm an. Anstelle von Daten füllt man in der Planung numerische Felder mit einer "9" und alphanumerische Felder mit einem "X".

Die Stelle des Dezimalpunktes in einem numerischen Feld mit einem Punkt angedeutet. Diese Planung können wir direkt in ein Programm umwandeln. Zur Ausgabe verwenden wir den "@ X,Y SAY"-Befehl.

Durch X wird die Zeile und durch Y die Spalte festgelegt, an der die Ausgabe beginnen soll. Es können Zahlenwerte oder Variablen für X und Y eingesetzt werden:

```
@ 6,24 SAY "Satznummer "
```

bzw.

```
STORE 6 TO X
STORE 24 TO Y
@ X,Y SAY "Satznummer "
```

Die Werte für die Zeilenposition reichen von 0 bis 23, die der Spaltenposition von 0 bis 79. Daher sind die Werte für Zeilen- und Spaltenposition um eins niedriger als die tatsächliche Position anzugeben. Diese Befehle geben die Zeichenkette "Satznummer " am Bildschirm aus. Die Ausgabe der Satznummer müssen Sie noch programmieren:

```
USE KUNDEN INDEX NAME, PLZ
APPEND BLANK
ERASE
@ 6,24 SAY "Satznummer "
@ $,$+1 SAY #
```

Da wir Kunden in einer Datei erfassen möchten, müssen wir die Datei KUNDEN eröffnen. Danach fügen wir einen leeren Satz zu der Datei hinzu. Der Befehl APPEND BLANK erzeugt keine Maske, sondern fügt nur einen Leersatz an die Datei an. Damit zeigt der Satzzeiger der Datei nun auf diesen letzten Satz. In der vierten Programmzeile treffen wir auf eine neue Variante des SAY-Befehls.

Die $-Zeichen ersetzen die letzte Zeilen- bzw. Spaltenposition des Cursors. Somit wird der Wert der Variablen # direkt hinter der Zeichenkette "Satznummer " ausgegeben. Diese Variable enthält immer die aktuelle Position des Satzzeigers. Die nächste Zeile der Maske enthält nicht nur eine Ausgabe, sondern auch eine Eingabe. Eine Eingabe wird mit dem GET-Befehl programmiert:

```
@ 7,24 SAY "KUNDENNUMMER" GET KUNDENNR
```

In die Variable KUNDENNR wird der eingegebene Wert abgespeichert. Geben wir diese Zeile einmal ohne Programm einfach hinter dem Prompt ein. Wie wir feststellen, wird die Zeichenkette ausgegeben und das vierstellige Feld eingerahmt von zwei Doppelpunkten invers dargestellt. Da wir eine Feldvariable der Datei KUNDEN für die Eingabe ausgewählt haben, wird das Eingabefeld entsprechend der Dateistruktur dieser Datei dargestellt. Die inverse Darstellung der Eingabefelder kann mit SET INTENSITY OFF abgeschaltet werden.

Das Programm EINGABE ersetzt APPEND

Wir können jetzt mit dem Schreiben des Programms beginnen. Mit MODIFY COMMAND EINGABE rufen wir den Editor auf und geben das Programm ein:

```
**********************************************************************
* PROGRAMM: EINGABE      AUTOR: M.-A. Beisecker      Datum: 01.07.86 *
* FUNKTION: ERSETZT DEN APPEND-BEFEHL FUER DIE DATEI KUNDEN          *
**********************************************************************

USE KUNDEN INDEX NAME, PLZ
APPEND BLANK
ERASE
@ 6,24 SAY "Satznummer "
@ $,$+1 SAY #
SET INTENSITY OFF
@ 7,24 SAY "KUNDENNUMMER" GET KUNDENNR
@ 8,24 SAY "NACHNAME    " GET NACHNAME
@ 9,24 SAY "VORNAME     " GET VORNAME
@ 10,24 SAY "STRASSE     " GET STRASSE
@ 11,24 SAY "POSTLEITZAHL" GET PLZ
@ 12,24 SAY "WOHNORT     " GET ORT
@ 13,24 SAY "TELEFON     " GET TELEFON
READ
SET INTENSITY ON
USE
```

Den Befehl READ in der zweitletzten Programmzeile kennen wir noch nicht. Dieser Befehl liest die Eingaben in den gerade aktiven Datensatz ein. Ohne den READ-Befehl werden die Daten nicht in die Datei geschrieben.

Wenn wir das Programm mit <^W> abgespeichert haben, können wir es starten:

```
DO EINGABE <RETURN>
```

Innerhalb der Maske, die jetzt auf dem Bildschirm erscheint, können wir mit den Tastenkombinationsbefehlen den Cursor genauso bewegen, als ob wir den APPEND-Befehl aufgerufen hätten. Das Programm erlaubt, einen Datensatz einzugeben und kehrt dann auf die Befehlsebene zurück. Wiederholtes Eingeben von Datensätzen kann nur mit Hilfe einer Schleife programmiert werden. Die dazu notwendigen Befehle werden in Kapitel 4 behandelt.

Gestaltung von Formaten

Bei der Verwendung von GET kann der Anwender eines Programms immer nur Daten des Datentyps der diesem GET-Befehl zugeordneten Variablen eingeben.

Das Feld PLZ in der Datei KUNDEN ist alphanumerisch definiert. Die folgende Programmzeile ordnet einem GET-Befehl die Variable PLZ zu:

```
@ 11,24 SAY "POSTLEITZAHL" GET PLZ
```

Diese Programmzeile ist ein Bestandteil des Programms EINGABE, das wir im letzten Kapitel kennengelernt haben. In das Feld Postleitzahl kann jedes Zeichen eingegeben werden. So kann ein Anwender beispielsweise versehentlich ein "O" anstelle einer Null eingeben. Passiert ein solcher Fehler, führt dies in einer sortierten oder indizierten Datei zu Problemen.

Damit es nicht zu derartigen Eingabefehler kommen kann, besteht die Möglichkeit, mit Hilfe eines Formates näher zu spezifizieren, wie eine Eingabe erfolgen soll.

Dazu gibt man ein PICTURE-FORMAT hinter dem Befehl GET an:

```
@ 11,24 SAY "POSTLEITZAHL" GET PLZ PICTURE '9999'
```

Das Format wird in Anführungszeichen eingeschlossen. Seine Länge ist durch die zugehörige Variable begrenzt. Es darf nicht länger, kann aber kürzer als die Variable sein. Das Format "9999" bedeutet, daß maximal vier Ziffern eingegeben werden dürfen. Eine Neun in einem Format bedeutet, daß an dieser Stelle eine Ziffer eingegeben werden kann.

Anstelle der Neun kann auch das Doppelkreuz (#) verwendet werden:

```
@ 11,24 SAY "POSTLEITZAHL" GET PLZ PICTURE '########'
```

Beide Symbole haben die gleiche Wirkung. Ein weiteres Symbol, das große "X", erlaubt die Eingabe von alphanumerischen Zeichen. Hier darf jedes Zeichen eingegeben werden:

```
@ 12,24 SAY "WOHNORT      " GET ORT PICTURE 'XXXXXXXXXXXX'
```

Wenn man erreichen will, daß nur Buchstaben oder Leerzeichen in ein Feld eingegeben werden dürfen, verwendet man den Buchstaben "A":

```
@ 8,24 SAY "NACHNAME      " GET NACHNAME PICTURE 'AAAAAAAAAAAAAAA'
```

Der Einsatz von "A" kann Probleme mit sich bringen. Soll ein Kunde mit Doppelname abgespeichert werden, so erlaubt dieses Format nicht die Eingabe eines Bindestriches. Setzt man dieses Format zur Eingabe des Wohnortes ein, können keine Ziffern zur Eingabe eines Ortsteiles erfaßt werden. Sinnvoll ist

die Anwendung dieses Symbols beispielsweise in einem Menü, bei dem der Anwender zur Auswahl eines Menüpunktes einen Buchstaben eingeben soll.

Das Symbol "!" läßt alle Zeichen als Eingabe zu, wandelt aber Kleinbuchstaben in Großbuchstaben um:

```
@ 8,24 SAY "NACHNAME      " GET NACHNAME PICTURE '!!!!!!!!!!!!!!!'
```

In der Variablen NACHNAME können somit Großbuchstaben, Ziffern und Sonderzeichen abgespeichert werden, aber keine Kleinbuchstaben.

In einem PICTURE-Format kann man auch die Symbole "$" und "*" verwenden. Diese werden auf dem Bildschirm angezeigt. Erreicht man mit dem Cursor ein so vorbesetztes Eingabefeld, werden die Symbole durch Null ersetzt.

Für ein Eingabefeld haben "*" und "$" also keine praktische Bedeutung. Bei der formatierten Ausgabe verwendet man sie zur Unterdrückung führender Nullen. Dem PICTURE-Format bei der Eingabe entspricht USING bei der Ausgabe. Das USING-Format wird daher zusammen mit dem SAY-Befehl eingesetzt:

```
. ERASE
. STORE 0020000.03 TO ZAHL
20000.03
. @ SAY 0,0 ZAHL USING '******9.99'
```

Als Ausgabe erhalten wir in der linken, oberen Bildschirmecke den in der Variablen ZAHL abgelegten Wert:

```
**20000.03
```

Die Ausgabe der Zahl können wir durch Einfügen einiger Kommata etwas übersichtlicher gestalten:

```
. @ SAY 0,0 ZAHL USING '*,***,**9.99'
```

dBASE II wurde in Amerika entwickelt. In Amerika wird anstelle des Punktes ein Komma verwendet, um in einer größeren Zahl die Tausender anzudeuten. Der Punkt zeigt die Dezimalstellen an. Wir erhalten die folgende Ausgabe:

```
***20,000.03
```

Wie Sie sehen, wird ein Komma durch einen Stern ersetzt, wenn keine von null verschiedene Ziffer voransteht. Anstelle der Sterne können wir auch Dollarzeichen ausgeben:

```
. @ SAY 0,0 ZAHL USING '$,$$$,$$9.99'
```

Die Ausgabe erscheint dann folgendermaßen:

```
$$$20,000.03
```

Mit Hilfe des Formates kann die Ausgabe der Dezimalstellen unterdrückt werden:

```
. @ SAY 0,0 ZAHL USING '*,***,**9'
```

Als Ergebnis erhält man den Integerwert:

```
$$$20,000
```

Wählt man das Format zu klein, wird ein entsprechend kleiner Wert angezeigt:

```
. @ 0,0 SAY ZAHL USING '9.99'

Die Zahl wird zu klein ausgegeben:

0.03
```

Mit dem USING-Format können Zahlen in vielfältiger Form angezeigt werden. Die Ausgabe von Zeichenketten ist dagegen nicht so vielfältig, aber dennoch in manchen Situationen recht hilfreich. Die Symbole "X" oder "A" bewirken ohne Unterschied die Ausgabe irgendeines Zeichens. Indem das USING-Format kürzer als die zugehörige Variable gewählt wird, kann die Ausgabe von Variablen in der gewünschten Länge erfolgen:

```
. STORE 'Autobus' TO STRING
Autobus
. @ 0,0 SAY STRING USING 'xxxx'
```

Diese Variablen-Ausgabe ergibt die Zeichenkette:

```
xxxx
```

Hätten Sie dieses Ergebnis erwartet? Die Symbole "X" und "A" müssen immer groß geschrieben werden, damit sie eine Formatierung bewirken:

```
. @ 0,0 SAY STRING USING 'XXXX'
```

Jetzt erhalten wir das erwartete Ergebnis:

```
Auto
```

Versuchen wir, diese Ausgabe in Großbuchstaben zu erhalten:

```
. @ 0,0 SAY STRING USING '!!!!'
```

Das Ergebnis ist wieder überraschend:

```
!!!!
```

Das Symbol "!" läßt sich zusammen mit USING nicht einsetzen. Die folgende Tabelle zeigt, welche Symbole bei USING und PICTURE eingesetzt werden können und wie sie wirken:

Symbol	Wirkung bei PICTURE	Wirkung bei USING
## oder 9	Ausgabe einer Ziffer.	Eingabe einer Ziffer, eines Punktes, eines Vorzeichens (+, –) oder eines Leerzeichens.
.	Dezimalpunkt	Dezimalpunkt.
,	Tausender	Keine Wirkung.
X	Ausgabe eines Zeichens.	Eingabe eines Zeichens.
A	Wirkung wie bei X.	Eingabe eines Buchstabens oder Leerzeichens.
$ oder *	Unterdrückt führende Nullen und gibt statt dessen "$" bzw. "*" aus.	Gibt "$" oder "*" am Bildschirm aus. Erreicht man die so formatierte Stelle, wird das Symbol automatisch durch Null ersetzt.

Einfache Ausgabe eines Menüs

In vielen Programmen gibt es sogenannte Menüs, die Funktionen eines Programms anzeigen. Durch die Eingabe einer Ziffer kann der Anwender eine gewünschte Funktion aufrufen:

```
Rechnungsschreibung
-------------------------

1. Kundenverwaltung
2. Artikelverwaltung
3. Bestellverwaltung
4. Rechnungsdruck

0. Programmende

BITTE WÄHLEN SIE:
```

Dieses Menü zeigt die verschiedenen Hauptfunktionen des Programmpaketes "Rechnungsschreibung". Das Programmpaket wird im Kapitel 9 vorgestellt.

Ein solches Menü besteht aus Text und einem Eingabefeld. Würde dieses Menü mit "?" oder "SAY" programmiert, wären sehr viele Einzelbefehle notwendig.

Mit den Befehlen TEXT und ENDTEXT programmiert man ein Menü einfacher. Alle Programmzeilen, in einem Programm die zwischen TEXT und ENDTEXT stehen, werden am Bildschirm ausgegeben.

Das Programm sieht also wie folgend aus:

```
STORE 0 TO AUSWAHL

ERASE

TEXT

                        Rechnungsschreibung
                        -------------------

                        1. Kundenverwaltung
                        2. Artikelverwaltung
                        3. Bestellverwaltung
                        4. Rechnungsdruck

                        0. Programmende


                        BITTE WAEHLEN SIE:
ENDTEXT

@ 13,45 SAY "BITTE WAEHLEN SIE:" GET AUSWAHL PICTURE "9"
```

Die Variable AUSWAHL wird am Anfang des Programms initialisiert, das heißt mit Null vorbesetzt. Der Befehl ERASE löscht den Bildschirm. Mit dem TEXT-Befehl wird der Menütext am Bildschirm angezeigt. Die Auswahl des Anwenders erfragt ein GET-Befehl. Das PICTURE-FORMAT verhindert, daß der Anwender mehr als ein Zeichen eingeben kann.

Beim Schreiben eines Textes zwischen TEXT und ENDTEXT sieht man sofort, wie dieser später auf dem Bildschirm dargestellt wird. Daher muß die Planung einer Bildschirmausgabe nicht zunächst auf dem Maskenformular erfolgen, vielmehr kann sie direkt am Bildschirm durchgeführt werden.

Da immer eine ganze Bildschirmzeile abgespeichert wird (auch wenn nur ein Wort in dieser Zeile ausgegeben wird), verbraucht diese Art der Programmierung relativ viel Speicherplatz.

Eine Bildschirmmaske sollte man nicht mit dem Befehl TEXT erstellen, da dieser nur eine Ausgabe von Zeichen, nicht aber die für eine Maske benötigten Eingabefelder erzeugen kann. Dafür sind die Befehle SAY und GET wesentlich besser geeignet, da mit ihnen in einer Programmzeile eine Ein- und Ausgabe programmiert werden kann.

Kapitel 4

Programmieren von Schleifen

In diesem Kapitel werden folgende Themen behandelt:

Einfache Schleife
Verschachtelte Schleifen
Programmierung einer Digitaluhr
Umwandlung eines numerischen Wertes in eine Zeichenkette

Zu diesen Themen gehören die folgenden Befehle:

DO WHILE
ENDDO
STR

Mit DO WHILE lassen sich Schleifen in ein Programm einbauen. Eine Schleife beginnt mit DO WHILE und endet mit ENDDO. Alle Befehle dazwischen werden durchlaufen, solange die Bedingung hinter DO WHILE erfüllt ist.

Einfache Schleife

In dem folgenden Programm wird mit dem ACCEPT-Befehl unser Name abgefragt und danach mit Hilfe der DO WHILE-Schleife zwanzigmal am Bildschirm angezeigt:

```
ACCEPT "WIE HEISSEN SIE?" TO NAME
STORE 1 TO ZAEHLER
ERASE
DO WHILE ZAEHLER =< 20
   ? NAME
   STORE ZAEHLER + 1 TO ZAEHLER
ENDDO
```

In der Schleife ist ein Zähler enthalten, der den Schleifenabbruch nach zwanzig Durchläufen ermöglicht. Der Zähler wird mit eins vorbesetzt und bei je-

dem Schleifendurchlauf um eins erhöht. Geben Sie dieses Programm mit MODIFY COMMAND in den Computer ein, und probieren Sie es aus. Das Programm nennen wir SCHLEIFE:

```
MODIFY COMMAND SCHLEIFE <RETURN>
```

Nachdem Sie das kleine Programm eingegeben haben, speichern Sie es mit <^W> und starten es mit:

```
DO SCHLEIFE <RETURN>
```

Sofort werden Sie nach dem Namen gefragt. Nachdem Sie den Namen eingegeben haben, erscheint er zwanzigmal auf dem Bildschirm. Jeweils unter dem Namen wird der Inhalt der Variablen ZAEHLER ausgegeben. Das Ergebnis des STORE-Befehls wird am Bildschirm angezeigt. Diese Anzeige wird mit SET TALK OFF unterdrückt. Unser Programm sieht also jetzt folgendermaßen aus:

```
SET TALK OFF
ACCEPT "WIE HEISSEN SIE?" TO NAME
STORE 1 TO ZAEHLER
ERASE
DO WHILE ZAEHLER =< 20
   ? NAME
   STORE ZAEHLER + 1 TO ZAEHLER
ENDDO
SET TALK ON
```

In der letzten Programmzeile machen wir die Systemänderung wieder rückgängig. Übersetzt würde dieses Programm etwa wie folgt lauten:

```
Schalte die Ausgabe von Ergebnissen am Bildschirm aus!
Frage "WIE HEISSEN SIE?" am Bildschirm und speichere das Ergebnis in der
Variablen NAME!
Lege die Zahl eins in der Variablen ZAEHLER ab!
Lösche den Bildschirm!
Wiederhole die folgenden Befehle bis zum ENDDO-Befehl so lange, bis
die Variable ZAEHLER den Wert zwanzig hat!
       Gebe die Variable NAME am Bildschirm aus!
       Erhöhe die Variable ZAEHLER um eins!
Ende der Schleife.
Schalte die Ausgabe von Ergebnissen am Bildschirm wieder ein!
```

Verschachtelte Schleifen

Die Befehle innerhalb der Schleife rückt man etwas nach rechts ein, um den logischen Ablauf des Programms deutlicher zu machen. Diese Technik ist insbe-

sondere dann wichtig, wenn zwischen Schleifenbeginn und -ende mehr als nur zwei Befehle stehen und eventuell auch mehrere Schleifen ineinander verschachtelt sind. Im folgenden Programm z.B. sind drei Schleifen ineinander verschachtelt:

```
SET TALK OFF
STORE 0 TO ZAEHLER1
STORE 0 TO ZAEHLER2
STORE 0 TO ZAEHLER3
DO WHILE ZAEHLER1 < 10
   STORE ZAEHLER1 + 1 TO ZAEHLER1
   @ 10,20 ZAEHLER1
   DO WHILE ZAEHLER2 = 10
      STORE ZAEHLER2 + 1 TO ZAEHLER2
      @ 11,20 ZAEHLER2
      DO WHILE ZAEHLER3 = 10
         STORE ZAEHLER3 + 1 TO ZAEHLER3
         @ 12,20 ZAEHLER3
      ENDDO
   ENDDO
ENDDO
SET TALK ON
```

Wie oft werden die einzelnen Schleifen durchlaufen?

Geben wir das Programm mit

```
MODIFY COMMAND SCHLEIFE <RETURN>
```

ein und starten es mit

```
DO SCHLEIFE <RETURN>.
```

Nach dem Start erscheinen drei Zahlen in der Mitte des Bildschirms, die ihren Wert laufend verändern. Diese drei Zahlen stellen die Inhalte der drei Schleifenzähler ZAEHLER1, ZAEHLER2 und ZAEHLER3 dar.

Da sich sowohl die zweite als auch die dritte Schleife innerhalb der ersten, äußeren Schleife befinden, werden diese zuerst abgearbeitet, bevor ein erneuter Durchlauf der äußeren Schleife beginnt.

Die dritte Schleife ist ein Teil der zweiten Schleife und wird deshalb vollständig abgearbeitet, bevor diese in einen weiteren Durchlauf geht. Somit ergibt sich für die innerste Schleife die Anzahl von 10 * 10 * 10 = 1000 Schleifendurchläufen. Die zweite Schleife wird 100 und die äußere Schleife zehnmal durchlaufen. Insgesamt werden also 1110 Schleifendurchläufe durchgeführt, für die der Schneider-Computer 1 Minute und 50 Sekunden benötigt.

Programmierung einer Digitaluhr

Aus diesen Angaben läßt sich ein kleines Uhrenprogramm erstellen:

```
**********************************************************************
* PROGRAMM: UHR      AUTOR: M.-A. Beisecker          Datum: 17.07.86 *
* FUNKTION: Digitaluhr mit Stunden, Minuten und Sekunden             *
**********************************************************************

* INITIALISIEREN

SET TALK OFF
STORE 0 TO ZAEHLER

* BILDSCHIRM LOESCHEN, AKTUELLE UHRZEIT ABFRAGEN

ERASE
INPUT "STUNDE  " TO STUNDE
INPUT "MINUTE  " TO MINUTE
INPUT "SEKUNDE " TO SEKUNDE
ERASE

* SCHLEIFE 1 FUER DIE STUNDE
NDO WHILE STUNDE < 24
   @ 12,15 SAY "UHRZEIT "
   @ 12,24 SAY STR(STUNDE,2,0)
   @ 12,26 SAY ":"
   * SCHLEIFE 2 FUER DIE MINUTEN

   DO WHILE MINUTE < 60
      @ 12,27 SAY STR(MINUTE,2,0)
      @ 12,29 ":"

      * SCHLEIFE 3 FUER DIE SEKUNDEN

      DO WHILE SEKUNDE < 60
         @ 12,30 SAY STR(SEKUNDE,2,0)

         * SCHLEIFE 4 ZUR DARSTELLUNG EINER SEKUNDE

         DO WHILE ZAEHLER < 26
            STORE ZAEHLER + 1 TO ZAEHLER
         ENDDO

         * ENDE DER SCHLEIFE 4

         STORE 0 TO ZAEHLER
         STORE SEKUNDE + 1 TO SEKUNDE
      ENDDO

      * ENDE DER SCHLEIFE 3

      STORE 0 TO SEKUNDE
      STORE MINUTE + 1 TO MINUTE
```

```
  ENDDO

  * ENDE DER SCHLEIFE 2

  STORE 0 TO MINUTE
  STORE STUNDE + 1 TO STUNDE
  IF STUNDE = 24
     STORE 0 TO STUNDE
  ENDIF
ENDDO

* ENDE DER SCHLEIFE 4

SET TALK ON

* LOESCHEN DER VERWENDETEN VARIABLEN

RELEASE STUNDE, MINUTE, SEKUNDE, ZAEHLER
```

In diesem Programm sind vier Schleifen ineinander verschachtelt. Drei Schleifen dienen dazu, die Stunden, Minuten und Sekunden hochzuzählen und am Bildschirm auszugeben. Die vierte Schleife soll eine Sekunde darstellen. Je nachdem, wie viele Durchläufe diese Schleife macht, geht unsere Uhr schneller oder langsamer. Da jeder Computer in der Geschwindigkeit etwas unterschiedlich ist, kann es sein, daß Sie das Programm etwas an Ihren Computer anpassen müssen. Dazu muß nur die Bedingung "ZAEHLER < 26" in der vierten DO WHILE-Schleife geändert werden.

Umwandlung eines numerischen Wertes in einer Zeichenkette

Die Eingabe der aktuellen Zeit am Anfang des Programms erfolgt mit dem INPUT-Befehl. Dieser Befehl erzeugt numerische Variablen in der Länge 4. Da die Stunden, Minuten und Sekunden nur zweistellig ausgegeben werden sollen, müssen die numerischen Variablen vor der Ausgabe in zweistellige Zeichenketten umgewandelt werden. Dazu dient der STR-Befehl. Der Befehl STR(STUNDE,2,0) wandelt den Inhalt der numerischen Variablen STUNDE in eine zweistellige Zeichenkette um, ohne etwaige Dezimalstellen zu berücksichtigen. Diesen Befehl sollten Sie immer dann anwenden, wenn Sie Zahlenwerte aus einer numerischen Variablen in einer bestimmten Länge am Bildschirm oder Drucker ausgeben möchten. In Verbindung mit dem Befehl STORE kann das Ergebnis eines STR-Befehls auch in einer alphanumerischen Variablen abgelegt werden.

Am Ende des Programms werden alle verwendeten Variablen gelöscht. Da immer nur 64 Variablen gleichzeitig vorhanden sein dürfen, sollten die nicht mehr benötigten Variablen am Ende jedes Programms gelöscht werden.

Kapitel 5

Programmieren von Entscheidungen

In diesem Kapitel werden folgende Themen behandelt:

Entscheidungen
Auswahl

Zu diesen Themen gehören die folgenden Befehle:

CASE
DO CASE
ELSE
ENDCASE
ENDIF
IF

Entscheidungen

Damit ein Programm möglichst vielseitig ist, muß der Anwender die Möglichkeit haben, mittels einer Entscheidung auf verschiedene Situationen reagieren zu können.

Solche Entscheidungen müssen wir im täglichen Leben dauernd treffen. Nehmen wir als Beispiel das Klingeln eines Weckers um sechs Uhr morgens. Durch dieses Klingeln werden wir unsanft geweckt und stehen vor der Entscheidung, ob wir aufstehen oder weiterschlafen sollen.

Falls wir keine Schule haben bzw. nicht zur Arbeit gehen müssen, ist das Klingeln des Weckers ein Fehler. Wir können den Wecker abstellen und weiterschlafen.

Ist diese Bedingung nicht erfüllt, müssen wir wohl oder übel aufstehen, uns waschen, anziehen, frühstücken und aus dem Haus gehen.

Dieses Problem würde man folgendermaßen programmieren:

```
IF (.NOT. SCHULE) .OR. (.NOT. ARBEIT)
   DO SCHLAFEN
ELSE
   DO AUFSTEHEN
   DO WASCHEN
   DO ANZIEHEN
   DO ESSEN
   DO GEHEN
ENDIF
```

Dieses Programm ist natürlich nur ein Beispiel für die Programmierung von Entscheidungen mit IF.

Hinter dem IF-Befehl folgt immer eine Bedingung. Ist diese Bedingung erfüllt, werden der oder die Befehle, die zwischen IF und ELSE stehen, bearbeitet, sonst werden die Befehle zwischen ELSE und ENDIF bearbeitet.

Der ELSE-Zweig kann auch ganz entfallen. Dann werden die Befehle zwischen IF und ENDIF bei erfüllter Bedingung abgearbeitet. Im anderen Fall passiert nichts.

Nehmen wir als Beispiel aus der Praxis die Suche nach einem bestimmten Datensatz mit dem FIND-Befehl. Wird der Datensatz gefunden, enthält die Variable # die Satznummer, ansonsten enthält sie eine Null. Der gefundene Datensatz soll mit EDIT geändert werden. Sollte der Datensatz in der Datei nicht vorhanden sein, so ist eine Fehlermeldung für den Anwender auszugeben.

An dieser Stelle muß im Programm entschieden werden, ob ein Datensatz gefunden wurde und dieser Satz bearbeitet werden kann oder ob dem Anwender mitgeteilt werden soll, daß der gesuchte Satz nicht existiert.

Die Bedingung für den IF-Befehl lautet also # > 0:

```
USE KUNDEN INDEX NAME
ACCEPT 'Welchen Namen suchen Sie?' TO SCHLUESSEL
FIND &SCHLUESSEL
IF # > 0
   EDIT #
ELSE
   ? 'Der gesuchte Datensatz ist nicht vorhanden!'
ENDIF
```

Wird der angegebene Name gefunden, ruft das Programm die EDIT Funktion auf, andernfalls wird mit dem ?-Befehl die Fehlermeldung am Bildschirm ausgegeben.

Bei komplexeren Bedingungen können IF-Befehle auch ineinander verschachtelt werden. Dabei müssen dann mehrere Bedingungen erfüllt sein, damit die Befehle im innersten IF-Zweig aktiviert werden.

Nehmen wir als Beispiel ein Menü: Die Variable "AUSWAHL" wird abgefragt und entscheidet, wie der weitere Programmablauf ist:

```
TEXT


                  Adressenverwaltung
                  ------------------

                  1. Adressen eingeben
                  2. Adressen ändern
                  3. Adressen löschen
                  4. Aufkleber drucken

                  Bitte wählen Sie:
ENDTEXT
@ 11,41 GET AUSWAHL
IF AUSWAHL > 0 .AND. AUSWAHL < 5
   IF AUSWAHL = 1
      DO EINGABE
   ENDIF
   IF AUSWAHL = 2
      DO AENDERN
   ENDIF
   IF AUSWAHL = 3
      DO LOESCHEN
   ENDIF
   IF AUSWAHL = 4
      DO DRUCKEN
   ENDIF
ELSE
   @ 13,32 SAY "FALSCHE EINGABE!"
ENDIF
```

Im äußeren IF-Zweig wird entschieden, ob die Eingabe korrekt ist. Da das Menü nur Eingaben von 1 bis 4 vorsieht, werden andere Eingaben mit einer Fehlermeldung quittiert.

Ist die Eingabe des Anwenders korrekt, gibt es vier mögliche Verzweigungen. Die Variable "AUSWAHL" kann einen der Werte 1 bis 4 enthalten. Eine der vier IF-Abfragen muß bei einer gültigen Variablen erfüllt sein, damit der nachfolgende Befehl aufgerufen wird. Dann wird für jeden Fall ein entsprechendes Unterprogramm gestartet.

Bei der Verschachtelung muß darauf geachtet werden, daß alle IF-Zweige mit ENDIF abgeschlossen sind. Der Übersicht halber sollten die Befehle innerhalb eines IF-Zweiges etwas eingerückt werden.

```
IF AUSWAHL > 0 .AND. AUSWAHL < 5
   IF AUSWAHL = 1
      DO EINGABE
   ELSE IF AUSWAHL = 2
           DO AENDERN
        ELSE IF AUSWAHL = 3
                DO LOESCHEN
             ELSE
                DO DRUCKEN
             ENDIF
        ENDIF
   ENDIF
ELSE
   @ 13,32 SAY "FALSCHE EINGABE!"
ENDIF
```

Bei diesem Programmausschnitt wird besonders deutlich, wie wichtig das Einrücken von Befehlen ist, wenn man komplexere Zusammenhänge in einem Programm noch verstehen möchte.

Hier wurden vier IF-Abfragen ineinander geschachtelt. Die Wirkung ist die gleiche, als wenn die inneren IF-Zweige auf der gleichen Ebene hintereinander stehen wie im vorhergehenden Programm. Dieses Programm ist als abschreckendes Beispiel gedacht, um zu zeigen, daß einzelne IF-Zweige wesentlich besser zu verstehen sind als ineinander verschachtelte.

Im nächsten Abschnitt werden Sie mit dem CASE-Befehl eine noch elegantere Möglichkeit kennenlernen, eine Verzweigung zu programmieren.

Auswahl

Der IF-Befehl ist vornehmlich für eine Entweder-Oder-Situation gedacht und wird bei einer größeren Auswahl an möglichen Entscheidungen leicht unübersichtlich. Für die Auswahl von mehr als zwei möglichen Entscheidungen in einer Ebene ist der CASE-Befehl gut geeignet. Eine CASE-Anweisung wird eingeleitet durch den Befehl DO CASE und beendet durch ENDCASE. Jede Verzweigung beginnt mit CASE, danach folgt eine Bedingung. Erinnern wir uns an das Menüprogramm aus dem Abschnitt 5.2:

```
TEXT

                  Adressenverwaltung
                  ------------------

                  1. Adressen eingeben
                  2. Adressen ändern
                  3. Adressen löschen
                  4. Aufkleber drucken
```

```
                    Bitte wählen Sie:
ENDTEXT
@ 11,41 GET AUSWAHL

DO CASE
   CASE AUSWAHL = 1
        DO EINGABE
   CASE AUSWAHL = 2
        DO AENDERN
   CASE AUSWAHL = 3
        DO LOESCHEN
   CASE AUSWAHL = 4
        DO DRUCKEN
   OTHERWISE
        @ 13,32 SAY "FALSCHE EINGABE!"
ENDCASE
```

Die CASE-Anweisung ist um einige Zeilen kürzer als die entsprechenden Programmzeilen mit IF. Der strukturierte Aufbau macht diese Programmzeilen leicht verständlich.

Jeder CASE-Befehl entspricht einem IF-Befehl. Ist eine CASE-Bedingung erfüllt, werden die Befehle bis zum nächsten CASE bearbeitet.

Wenn keine der vorherigen CASE-Bedingungen zutrifft, wird die Befehlszeile hinter OTHERWISE durchlaufen. In diesem konkreten Fall bedeutet das, daß die Variable AUSWAHL größer als vier oder kleiner als null sein muß.

Man ist mit dem CASE-Befehl oder auch dem IF-Befehl nicht beschränkt in der Anzahl der Befehle, die bei erfüllter Bedingung durchlaufen werden. Innerhalb einer CASE-Verzweigung können eine DO WHILE-Schleife und auch mehrere IF-Verzweigungen verschachtelt werden.

Als Programmieranfänger sollte man sich vor solchen Verschachtelungen allerdings hüten. Sehr leicht verliert man den Überblick und versteht dann die Ergebnisse eines Programms nicht mehr.

Dies kann durch die Verwendung von Unterprogrammen verhindert werden. In dem obigen Programmausschnitt werden die Programmteile Eingabe, Ändern, Löschen und Drucken von Adressen jeweils durch eigene Programme realisiert. Auf diese Weise steht hinter jedem CASE-Befehl nur der Aufruf des jeweiligen Unterprogramms. Die CASE-Anweisung kann daher leicht überschaut und verstanden werden.

Da ein Unterprogramm jeweils von der Diskette in den Hauptspeicher geladen werden muß, vergeht immer ein wenig Zeit, bevor es gestartet werden kann. Daher sollte man den Einsatz von Unterprogrammen auch nicht übertreiben und für nicht wenige Zeilen jeweils ein eigenes Programm schreiben.

Kapitel 6

Der Listengenerator REPORT

In diesem Kapitel werden folgende Themen behandelt:

Eingabe der Listenparameter
Ausgabe der Liste auf den Bildschirm
Ausgabe der Liste auf den Drucker
Ersetzen des Systemdatums
Setzen einer Überschrift
Unterdrücken des ersten Seitenvorschubes
Nachträgliches Verändern der Listenparameter

Zu diesen Themen gehören die folgenden Befehle:

MODIFY COMMAND
REPORT
SET ALTERNATE TO
SET DATE HEADING
SET EJECT ON/OFF
SET HEADING
SET MARGIN

Der Listengenerator REPORT fragt in einem Dialog mit dem Anwender alle für eine Listenausgabe benötigten Parameter ab. Diese Parameter sind die gewünschten Felder, deren Feldlängen, die Spaltenüberschriften sowie die Listenüberschrift.

Es können Gesamtsummen und Zwischensummen von numerischen Feldern berechnet und mit ausgedruckt werden.

Die Darstellung der Liste kann auf dem Bildschirm, einem Drucker oder in eine Datei erfolgen.

Im folgenden Text wird gezeigt, wie aus der Datei ARTIKEL mit REPORT eine Liste erzeugt wird.

Der REPORT-Befehl nimmt die Daten für die Liste aus der gerade eröffneten Datei in der Reihenfolge, in der diese indiziert oder sortiert ist. Daher muß vor dem Aufruf von REPORT die zugehörige Datei samt Indexdateien eröffnet werden:

```
. USE ARTIKEL INDEX ARTBEZ
```

Eingabe der Listenparameter

Nach dem Aufruf von REPORT müssen Sie angeben, welche Felder die Liste enthalten soll. Daher müssen Sie alle Feldnamen der eröffneten Datei kennen. Sehen wir uns die Dateistruktur der Datei ARTIKEL also noch einmal an, bevor wir den Listengenerator aufrufen:

```
. DISPLAY STRUCTURE
Strukturdaten für Datei: A:ARTIKEL.DBF
Anzahl der Sätze: 00006
Datum der letzten Aktualisierung: 21/07/86
Primäre Datei
Feld  Name    Typ  Länge  Dez.st.
001     ARTIKELNR     N     004
002     ARTIKELBEZ    C     020
003     BESTAND       N     004
004     MINBESTAND    N     004
005     EK            N     006     002
006     VK            N     006     002
** Gesammt **           00045
```

Geben Sie <^P> oder SET PRINT ON vor dem Befehl DISPLAY STRUCTURE ein, so erhalten Sie einen Ausdruck der Dateistruktur. Ein solcher Ausdruck hilft bei der Eingabe der Listenparameter. Der Listengenerator wird durch die Eingabe:

```
REPORT <RETURN>
```

aufgerufen. Sofort erscheint die Aufforderung, einen Namen für die sogenannte Formulardatei einzugeben:

```
. REPORT
Bitte Dateinamen für Bericht eingeben: ARTIKEL
```

Es kann hier derselbe Name wie für die zugehörige Datenbank gewählt werden. Durch den Namenszusatz ".FRM" kann dBASE II die Datei ARTIKEL.FRM von der Datei ARTIKEL.DBF unterscheiden. Die Frage nach dem Dateinamen entfällt, wenn Sie direkt bei dem Aufruf von REPORT die Formulardatei mit angeben:

```
REPORT FORM ARTIKEL <RETURN>
```

Dabei muß der Zusatz FORM hinter dem REPORT-Befehl stehen. Ist die angegebene Datei auf der Diskette nicht vorhanden, wird nach den Listenparametern gefragt und die Formulardatei angelegt. Sollte schon eine entsprechende Datei existieren, wird die Liste direkt ausgegeben. Die Datei ARTIKEL.FRM ist noch nicht vorhanden, also beginnt der Dialog:

```
Eingabe: m=linker Rand, l=Zeilen pro Seite, w=Zeilenbreite
```

Hier wird die Formulargröße festgelegt. Auf ein DIN A4-Blatt passen 72 Druckzeilen. Falls wir keine Eingabe machen, werden 57 Zeilen pro Seite als Standard genommen. Für ein DIN A4-Blatt würden wir also l=72 eingeben. Der linke Rand ist standardmäßig auf 5 Zeichen gesetzt. Wünschen wir einen größeren oder kleineren Rand, muß dies explizit angegeben werden. Die Zeilenbreite für ein DIN A4-Blatt beträgt normalerweise 80 Zeichen. Unter Zeilenbreite versteht der REPORT-Listengenerator die Druckzeile plus dem linken Rand. Machen wir hier keine Eingabe, wird die Zeilenbreite auf 80 Zeichen gesetzt. Demnach werden bei einem linken Rand von 5 Zeichen höchstens 75 Zeichen pro Zeile gedruckt.

Stellen wir die Parameter wie folgt ein:

m=0,l=72,w=80

Die Eingabe der Zeilenbreite kann in diesem Fall auch unterbleiben. Sie dient nur dokumentarischen Zwecken. Als nächstes müssen wir entscheiden, ob wir eine Seitenüberschrift wünschen:

```
Mit Seitenüberschrift? (J/N) J
```

Wir geben hier wie oben gezeigt <J> ein und erhalten die Abfrage:

```
Bitte Seitenüberschrift eingeben: WAREN-LISTE
```

Unsere Liste soll die Überschrift WAREN-LISTE tragen. Nachträglich läßt sich noch eine zweite Überschrift mit einem SET-Befehl ausgeben. Dies wird später noch gezeigt. Die Beantwortung der nächsten Frage entscheidet darüber, ob die Liste ein- oder zweizeilig ausgedruckt wird:

```
Leerzeile zwischen den Sätzen? (J/N) J
```

Durch zweizeiliges Drucken wird die Liste doppelt so lang wie bei einzeiligem Druck. Dieser Nachteil wird durch bessere Lesbarkeit kompensiert.

Mit der nächsten Frage entscheiden wir theoretisch darüber, ob in der Liste Gesamtsummen und Zwischensummen von numerischen Feldern gebildet werden oder nicht. Leider reagiert die vorliegende dBASE-II-Version 2.41

für den Schneider CPC 6128 auf die Beantwortung der Frage überhaupt nicht. In der Hoffnung, daß dieser Fehler in einer späteren Version behoben ist, werden die fehlenden Abfragen hier angeführt:

```
Gesamtsummen erforderlich? (J/N) J
Bericht mit Zwischensummen? (J/N) N    (Diese Funktion ist nicht
                                        vorhanden !)
```

Mit den folgenden Parametern legen wir fest, welche Felder in welcher Weise ausgegeben werden:

```
Spalte Breite, Inhalt
001    7,ARTIKELNR
```

Hinter der Spaltennummer 1 geben wir ein, in welcher Breite das nachfolgende Feld ausgegeben werden soll. Das Feld ARTIKELNR ist nur 4 Stellen lang, es bleibt also ein Platz von 3 Stellen bis zur Ausgabe des nächsten Feldes. Wählen wir eine Spalte breiter, als das zugehörige Feld lang ist, so wird diese mit Leerstellen aufgefüllt. Wird die Spalte dagegen kürzer gewählt als das zugehörige Feld, so verteilt REPORT dieses Feld bei der Ausgabe auf zwei Zeilen.

Zu jeder Spalte gehört eine Überschrift:

```
Bitte Überschrift eingeben: <ARTNR.;-------
```

Das Kleiner-Zeichen bedeutet, daß die Ausgabe der Überschrift linksbündig erfolgt. Anstelle von diesem Zeichen können wir auch das Größer-Zeichen verwenden. Dieses Zeichen bedeutet, daß die Ausgabe rechtsbündig ausgegeben wird. Gibt man keines der Zeichen an, erfolgt die Ausgabe zentriert. Das Semikolon legt fest, daß die nachfolgenden Zeichen in die nächste Zeile kommen. Die obige Eingabe sieht in der Liste also folgendermaßen aus:

```
ARTNR.
-------
```

Da das Feld ARTNR numerisch ist, müßte hier theoretisch folgende Abfrage erfolgen:

```
Gesamtsummen erforderlich ? (J/N) N
```

Da es nicht sinnvoll ist, Summen von Artikelnummern zu bilden, hätten wir hier wahrscheinlich N eingegeben. Später bei dem Feld EK wäre eine Gesamtsumme dagegen sehr hilfreich. Sollte es in Ihrer dBASE-II-Version also möglich sein, Gesamtsummen auszugeben, können Sie so bei jedem numerischen Feld entscheiden, ob für dieses Feld eine Summe gebildet werden soll oder nicht. Dies gilt ebenso für die Zwischensummen.

Setzen einer Überschrift

Nach dem Feld ARTIKELNR soll das Feld ARTIKELBEZ ausgegeben werden:

```
002    23,ARTIKELBEZ
Bitte Überschrift eingeben:
>ARTIKELBEZEICHNUNG;----------------------
```

Wie wir sehen, wird die Spaltennummer automatisch hochgezählt. Damit später alle Informationen in eine Zeile passen, muß man sich vorher genau überlegen, wie breit jede Spalte werden darf und ob wirklich jedes Feld benötigt wird.

Die restlichen Spalten geben wir wie die ersten beiden ein:

```
003    8,BESTAND
Bitte Überschrift eingeben: >BESTAND;--------

004    8,MINBESTAND
Bitte Überschrift eingeben: >MINIMUM;--------

005    9,EK
Bitte Überschrift eingeben: >EK;---------
```

Bisher wurden die Spalten nur mit einzelnen Feldern gefüllt. Es ist aber auch möglich, zwei numerische Felder zu addieren, subtrahieren, multiplizieren oder dividieren und das Ergebnis in einer Spalte auszugeben:

```
006    9,EK*BESTAND
Bitte Überschrift eingeben: >WERT;---------

007    9,VK
Bitte Überschrift eingeben: >VK;---------
```

Nun sind wir bei Spalte 008 angelangt. Um den Dialog zu beenden, drücken wir einfach <RETURN>:

```
008 <RETURN>
```

Wie Sie vielleicht schon festgestellt haben, lassen sich Fehler in der Eingabe nicht mehr verbessern, nachdem die RETURN-Taste gedrückt wurde. Ein Fehler kann nur durch das Löschen der Datei ARTIKEL.FRM und die Neueingabe mit dem REPORT-Befehl behoben werden. Es gibt aber noch eine elegantere Möglichkeit. Wir editieren die Datei ARTIKEL.FRM mit dem Editor MODIFY COMMAND:

```
. MODIFY COMMAND ARTIKEL.FRM
```

Nachträgliches Verändern der Listenparameter

Es erscheint der folgende Dateiinhalt, ohne die Klammerinhalte, die als Kommentar hinzugefügt wurden:

```
m=0, l=72, w=80 (Formular: Linker Rand,Zeilen pro Seite,Zeilenbreite)
J               (Seitenüberschrift?)
WAREN-LISTE     (Überschrift der Liste)
J               (Leerzeilen zwischen den Sätzen?)
J               (Gesamtsummen?)
7,ARTIKELNR     (Feldbreite,Feld)
<ARTNR.;------- (Überschrift der Spalte)
23,ARTIKELBEZ
>ARTIKELBEZEICHNUNG;----------------------
8,BESTAND
>BESTAND;--------
8,MINBESTAND
>MINIMUM;--------
9,EK
>EK;---------
9,EK*BESTAND
>WERT;---------
9,VK
>VK;---------
```

Auf diese Weise läßt sich eine Formatdatei nicht nur editieren, sondern auch neu anlegen. Nachdem die Datei nun verbessert ist, können wir probeweise einen Ausdruck erzeugen:

```
. REPORT FORM ARTIKEL
Keine Datenbank eröffnet, bitte Dateinamen eingeben
```

Ausgabe der Liste auf dem Bildschirm

Diese Meldung kommt immer dann, wenn vergessen wurde, die zugehörige Datei vorher zu öffnen. Durch die Eingabe von ARTIKEL <RETURN> wird die Listenausgabe gestartet:

```
Seitennr. 00001

                        WAREN-LISTE

ARTNR        ARTIKELBEZEICHNUNG  BESTAND  MINIMUM       EK      WERT      VK

------- ---------------------- -------- -------- -------- --------- -------

      1 Joystick de Luxe             12       75    19.90    238.80   49.00
```

```
     2 3 Zoll Disketten            120       50     6.00    720.00     9.90

     3 Colour Monitor                3        3   798.00   2394.00   998.00

     4 Schutzhuelle                 23       30     9.90    227.70    19.90

     5 Druckerkabel                  2        1    24.00     48.00    49.00

     6 5 1/4 Zoll Disketten        200      150     3.96    792.00     4.90
```

Diese Ausgabe erfolgte am Bildschirm. Später werden wir sehen, wie eine Ausgabe auf dem Drucker erzeugt wird. Die Artikelsätze sind so angeordnet, wie wir sie in die Datei eingegeben haben. Um eine nach der Artikelbezeichnung geordnete Liste zu erhalten, muß die Indexdatei ARTBEZ aktiviert werden:

```
. USE ARTIKEL INDEX ARTBEZ
```

Ersetzen des Systemstatus

Wir können in der Liste ein individuelles Datum ausgeben. Dazu verwendet man den SET DATE-Befehl:

```
. SET DATE TO 22.07.86
```

Damit die neue Liste von der alten unterschieden werden kann, erzeugen wir noch eine zweite Überschrift:

```
. SET HEADING TO Indiziert nach Artikelbezeichnung
```

Jetzt können wir REPORT wieder aufrufen

```
. REPORT FORM ARTIKEL
```

und erhalten die folgende Liste:

```
Seitennr. 00001    Indiziert nach Artikelbezeichnung
22/07/86
                          WAREN-LISTE

ARTNR        ARTIKELBEZEICHNUNG  BESTAND  MINIMUM       EK      WERT      VK

------- ---------------------- -------- -------- -------- --------- -------


     2 3 Zoll Disketten            120       50     6.00    720.00     9.90

     6 5 1/4 Zoll Disketten        200      150     3.96    792.00     4.90

     3 Colour Monitor                3        3   798.00   2394.00   998.00
```

```
5 Druckerkabel                2        1    24.00     48.00    49.00

1 Joystick de Luxe           12       75    19.90    238.80    49.00

4 Schutzhuelle               23       30     9.90    227.70    19.90
```

Die Seitennummer und das Datum können auch unterdrückt werden:

```
REPORT FORM ARTIKEL PLAIN <RETURN>
```

Die Listenüberschrift erscheint nur auf dem ersten Blatt, wenn der Parameter PLAIN verwendet wird.

Ausgabe der Liste auf den Drucker

Bis jetzt wurden die Listen nur auf dem Bildschirm ausgegeben. Nun werden wir sehen, wie eine Liste auf dem Drucker ausgegeben werden kann. Dazu sind verschiedene Verfahren möglich.

Die einfachste Möglichkeit besteht darin, den REPORT-Befehl um den Parameter TO PRINT zu erweitern:

```
.REPORT FORM ARTIKEL TO PRINT <RETURN>
```

Unterdrücken des ersten Seitenvorschubes

Bevor die Liste gedruckt wird, erfolgt zunächst ein Seitenvorschub. Dieser Seitenvorschub kann mit SET EJECT OFF <RETURN> unterdrückt werden. Das Drucken nach der folgenden Methode hat den Nachteil, daß die Liste gleichzeitig auch auf dem Bildschirm ausgegeben wird. Die Ausgabe auf dem Bildschirm kann mit SET CONSOLE OFF abgeschaltet werden. Der Drucker wird über SET PRINT ON aktiviert, so daß sich folgende Befehle ergeben:

```
.SET EJECT OFF <RETURN>
.SET PRINT ON <RETURN>
.SET CONSOLE OFF <RETURN>
.REPORT FORM ARTIKEL <RETURN>
.SET CONSOLE ON <RETURN>
.SET PRINT OFF <RETURN>
```

Anstelle von SET PRINT ON und SET PRINT OFF kann <^P> verwendet werden.

Die Ausgabe kann auch in eine Datei erfolgen. Als Datei läßt sich die Liste mit einer Textverarbeitung weiter verarbeiten. Dazu wird häufig der Parameter

PLAIN hinter dem REPORT-Befehl angegeben. Mit dem Befehl SET ALTERNATE kann die Liste in eine beliebige Datei geschrieben werden:

```
.SET ALTERNATE TO LISTE.TXT <RETURN>
.SET ALTERNATE ON <RETURN>
.REPORT FORM ARTIKEL PLAIN <RETURN>
.SET ALTERNATE OFF <RETURN>
```

Wir müssen daran denken, die Ausgabe in die Datei rückgängig zu machen, da sonst alle Bildschirmausgaben in dieser Datei gespeichert würden.

Übung 15:

Programmieren Sie eine Listenausgabe mit dem REPORT-Befehl von der Datei KUNDEN. Die Ausgabe soll nach dem folgenden Beispiel aufgebaut und nach Namen sortiert sein.

```
Seitennr. 00001

KUNDEN-LISTE

NR    NACHNAME, VORNAME  STRASSE          PLZ  WOHNORT      TELEFON      UMSATZ
----- ------------------ --------------- ---- ------------ ----------- ------
    7 Ende, Werner       Am Schlußlicht  5428 Endlichhofen 07142-568   789.50
    6 Hutter, Sabine     Heideweg 23     6790 Oberammergau 0122-4554   978.40
    5 Hütter, Angelika   Sandfuhrstr. 34 4200 Oberhausen   0208-56129    0.00
    1 Käufer, Otto       Ladengasse 9    5000 Köln 32                   99.99
    3 Maier, Udo         Kolpingstr. 14a 6050 Offenbach    069-3467     67.20
    4 Maiers, Trude      Bahnhofstr. 11  4300 Essen-West   0201-20316  117.98
    2 Meyer, Theo        Eichenstr. 15   4100 Duisburg     0203-435612 999.78
```

Die Ausgabe von Nachname und Vorname wird mit dem TRIM-Befehl programmiert:

```
TRIM(NACHNAME)+", "+TRIM(VORNAME)
```

Kapitel 7

Eine vollständige Artikelverwaltung

In diesem Kapitel werden folgende Themen behandelt:

Das Menüprogramm ARTVERW
Aufbau des Menüprogramms ARTVERW
Erfassen von neuen Datensätzen
Drucken von Listen
Artikel editieren mit dem Programm EDIART
Löschen von Artikeln mit dem Programm LOEART

Zu diesen Themen gehören die folgenden Befehle:

SET COLON ON/OFF
SET TALK ON/OFF
#

Das Menüprogramm ARTVERW

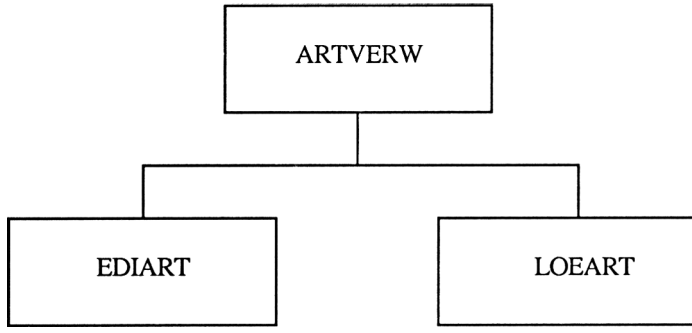
In diesem Kapitel wird ein vollständiges Programmpaket zur Verwaltung von Artikeln vorgestellt. Zur besseren Überschaubarkeit und logischen Gliederung teilt man eine große Aufgabe wie die Artikelverwaltung auf mehrere Programme auf. Dabei wählt der Benutzer über ein sogenanntes Menüprogramm eine gewünschte Funktion aus. Im Programm wird diese Funktion mittels eines Unterprogramms realisiert, das vom Menüprogramm aus aufgerufen wird.

Aufbau des Menüprogramms ARTVERW

Das Programm ARTVERW hat die Aufgabe, das Menübild anzuzeigen und die Eingabe von Artikeln sowie den Ausdruck einer Liste zu ermöglichen. Für diese Funktionen werden keine Unterprogramme benötigt, da sie sich mit wenigen Befehlen programmieren lassen. Wenn der Anwender des Programms

einen Artikelsatz editieren oder löschen möchte, wird dagegen von ART-VERW das jeweils zuständige Unterprogramm EDIART beziehungsweise LOEART aufgerufen.

Hier die graphische Darstellung des Programmpaketes:



Sehen wir uns die ersten sechs Zeilen des Programms ARTVERW an:

```
*************************************************************************
* UNTERPROGRAMM: ARTVERW                                                *
* FUNKTION     : VERWALTEN EINER ARTIKELDATEI                           *
* AUTOR        : M.-A. BEISECKER                  DATUM: 21.07.1986 *
*************************************************************************
```

Ein Sternchen am Anfang der Zeile ist für dBASE II ein Zeichen, daß nun ein Kommentar folgt. Für manche Menschen ist Kommentar reine Speicherplatzverschwendung.

Wie wertvoll ein guter Kommentar in einem Programm ist, zeigt sich spätestens dann, wenn man sich sein Programm nach einem halben Jahr wieder anschaut, um etwa eine Änderung vorzunehmen. Den Computer interessiert eine Kommentarzeile nicht, er überliest sie ganz einfach.

Am Kopf eines jeden Programms sollten der Programmname, die Funktion, das Datum der letzten Bearbeitung sowie der Autor stehen.

Dieses Programm kann für sich alleine aufgerufen werden, also als Hauptprogramm arbeiten. Später soll es uns aber als Unterprogramm in dem Programmpaket Rechnungsschreibung dienen.

Nach dem Programmkopf werden in einem strukturierten Programm die Systemparameter eingestellt und die benötigten Dateien eröffnet:

```
* EINSTELLEN DER SYSTEMPARAMETER UND INITIALISIERUNG DER VARIABLEN

SET TALK OFF
SET COLON OFF
STORE 0 TO AUSWAHL

* OEFFNEN DER DATEI ARTIKEL MIT INDEXDATEI ARTBEZ UND ARTNR

USE ARTIKEL INDEX ARTBEZ, ARTNR
```

Natürlich müssen Sie später nicht jeden Befehl mit einem Kommentar versehen, wie es hier zu Schulungszwecken demonstriert wird.

Der erste Befehl in diesem Programm SET TALK OFF dient dazu, die Ausgabe der Antworten auf Befehle zu unterdrücken. Am Ende des Programms werden wir diese Systemeinstellung mit SET TALK ON wieder rückgängig machen.

Mit dem Befehl SET COLON OFF schalten wir die Ausgabe von Doppelpunkten als Feldbegrenzer bei Eingabefeldern aus. Dieser Befehl kann entfallen, wenn die Doppelpunkte gewünscht werden.

Zuletzt in diesem Programmabschnitt wird die Variable AUSWAHL mit null initialisiert, das heißt vorbesetzt. Diese numerische Variable dient, wie der Name schon andeutet, der Auswahl unter den Menüpunkten.

Es erleichtert das Programmieren sehr, wenn man leicht verständliche und eindeutige Variablennamen wählt. Man sollte sich angewöhnen, für gleiche Aufgaben immer den gleichen Variablennamen zu verwenden, da man sich dann in den Programmen leichter zurechtfindet und Verwechslungen ausgeschlossen werden. Diese Technik begrenzt zudem automatisch die verwendete Anzahl Variablen.

Direkt nach dem Initialisieren wird die benötigte Datei ARTIKEL mit allen vorhandenen Indexdateien eröffnet. Da wir diese Indexdateien bisher noch nicht angelegt haben, wollen wir dies jetzt tun:

```
USE ARTIKEL <RETURN>
INDEX ON ARTIKELNR TO ARTNR <RETURN>
INDEX ON ARTIKELBEZ TO ARTBEZ <RETURN>
```

Ohne die Indexdateien bricht das Programm mit einer Fehlermeldung ab. Sie müssen also vor dem ersten Programmstart erstellt werden.

Als nächstes kommt in unserem Programm das Menübild:

```
* SCHLEIFE ZUM WIEDERHOLTEN DURCHLAUF DES MENUES

DO WHILE AUSWAHL >= 0

* AUFBAU DES MENUES

ERASE

TEXT

                        ARTIKELVERWALTUNG
                        -----------------

                    1. EINGABE NEUER ARTIKEL
                    2. EDITIEREN VON ARTIKELN
                    3. LOESCHEN VON ARTIKELN
                    4. DRUCKEN EINER LISTE

                    0. ZURUECK IN DAS VORHERIGE MENUE


                    BITTE WAEHLEN SIE:
ENDTEXT

@ 13,45 GET AUSWAHL PICTURE "9"
READ
```

Die DO WHILE-Schleife wurde eingebaut, um dieses Programm so lange durchlaufen zu können, bis es durch die Auswahl Null verlassen wird. Nachdem der Bildschirm mit ERASE gelöscht wurde, wird das Menü mit dem TEXT-Befehl aufgebaut. Mit diesem Befehlsaufbau sieht man sofort, wie das Menü später auf dem Bildschirm erscheint. Diese Methode benötigt allerdings mehr Speicherplatz, als wenn der SAY- oder ?-Befehl verwendet wird.

Nach dem Menüaufbau erfolgt die Abfrage der Variablen AUSWAHL.

Wir benötigen nun einen Befehl, um in Abhängigkeit von der Variablen AUSWAHL zu verzweigen. Für diesen Zweck eignet sich das Konstrukt DO CASE vorzüglich:

```
* CASE - VERZWEIGUNG IN ABHAENGIGKEIT DER VARIABLEN AUSWAHL

DO CASE
   CASE AUSWAHL = 1
        APPEND
   CASE AUSWAHL = 2
        DO EDIART
   CASE AUSWAHL = 3
        DO LOEART
   CASE AUSWAHL = 4
```

```
        SET EJECT OFF
        REPORT FORM ARTIKEL
    CASE AUSWAHL = 0
        USE
        SET TALK ON
        SET COLON ON
        RETURN      * ENDE DES PROGRAMMES ARTVERW
ENDCASE

* ENDE CASE - VERZWEIGUNG
```

Im Falle der Neueingabe von Artikeln durch die Wahl 1 wird einfach der Befehl APPEND aufgerufen. Dies erfolgt genauso, als wenn wir den Befehl im interaktiven Befehlsmodus aufgerufen hätten.

So einfach ist das Editieren nicht, denn dort – ebenso wie bei der Löschung von Artikeln – muß ausgewählt werden, welcher Artikel editiert werden soll. Daher wurden für diese beiden Aufgaben externe Unterprogramme geschrieben.

Drucken von Listen

Der Ausdruck einer Artikelliste geht mit dem REPORT-Befehl wieder so einfach, daß Sie kein separates Programm benötigen. Eine etwas aufwendigere Lösung mit eigenem Druckprogramm wird in Kapitel 8 in Verbindung mit der Adressenverwaltung gezeigt.

Da beim Eröffnen der Datei ARTIKEL die Indexdatei ARTBEZ zuerst genannt wird, ist die Liste nach der Artikelbezeichnung sortiert. Durch das Einfügen des Befehls SET INDEX TO ARTNR wird die Liste nach Artikelnummern sortiert ausgegeben:

```
CASE AUSWAHL = 4
        SET INDEX TO ARTNR
        SET EJECT OFF
        REPORT FORM ARTIKEL
        SET INDEX TO ARTBEZ
```

Diese Änderung machen wir nach Ausführung des REPORT-Befehls wieder rückgängig. Durch das Erstellen weiterer Indexdateien und deren Einbau in das Programm oder die Veränderung der Listendatei ARTIKEL.FRM kann die Listenausgabe beliebig gesteuert werden. Doch zurück zu dem Programm:

```
STORE 0 TO AUSWAHL

ENDDO

* ENDE DER SCHLEIFE
```

Damit nach einem Durchlauf die Variable AUSWAHL wieder 0 anzeigt, wird sie neu initialisiert. Dann folgt das physikalische Ende des Programms. Das eigentliche logische Ende ist in der CASE-Verzweigung verborgen:

```
CASE AUSWAHL = 0
     USE
     SET TALK ON
     SET COLON ON
     RETURN     * ENDE DES PROGRAMMS ARTVERW
```

Wie es am Ende eines übergeordneten Programms sein sollte, werden die geöffneten Dateien mit USE geschlossen, und das System wird wieder in den ursprünglichen Zustand gebracht. Der RETURN-Befehl dient dem Rücksprung in das Hauptprogramm MENUE, das uns in Kapitel 9 begegnen wird. Wir sehen uns jetzt an, wie das Programm abläuft:

```
. DO ARTVERW
```

Nachdem das Programm mit DO gestartet wurde, erscheint das Menü:

```
          ARTIKELVERWALTUNG
          -----------------

     1. EINGABE NEUER ARTIKEL
     2. EDITIEREN VON ARTIKELN
     3. LOESCHEN VON ARTIKELN
     4. DRUCKEN EINER LISTE

     0. ZURUECK IN DAS VORHERIGE MENUE


     BITTE WAEHLEN SIE: 1
```

Erfassen von neuen Datensätzen

Wenn wir durch die Eingabe der Eins den Menüpunkt "Eingabe neuer Artikel" wählen, erscheint das APPEND-Menü:

```
Satznummer 00007
ARTIKELNR
ARTIKELBEZ
BESTAND
MINBESTAND
EK
VK
```

Die Felder sind invers dargestellt. Die Feldbegrenzungen durch Doppelpunkte fehlen. Verlassen wir APPEND wie gewohnt durch <^Q>, befinden wir uns wieder im Menü.

Wählen wir den Menüpunkt Vier aus, erscheint eine Liste kurz auf dem Bildschirm, und das Menü kommt zurück. Wir haben vergessen, den Drucker mit SET PRINT ON einzuschalten. Daher erscheint die Liste auf dem Bildschirm anstelle auf dem Drucker. Die Bildschirmausgabe können wir während der Druckausgabe mit SET CONSOLE OFF unterdrücken.

Das CASE für AUSWAHL=4 sieht somit folgendermaßen aus:

```
CASE AUSWAHL = 4
        SET PRINT ON
        SET CONSOLE OFF
        SET EJECT OFF
        REPORT FORM ARTIKEL
        SET PRINT OFF
        SET CONSOLE ON
```

Dieser kleine Fehler hat Ihnen gezeigt, wie Sie zu Testzwecken Listen auf dem Bildschirm darstellen können. Das Programm funktioniert jetzt. Es fehlen nur noch die beiden Unterprogramme, die in den nächsten beiden Abschnitten besprochen werden.

Hier noch einmal das Programm ARTVERW in zusammenhängender Form:

```
*************************************************************************
* UNTERPROGRAMM: ARTVERW                                                *
* FUNKTION     : VERWALTEN EINER ARTIKELDATEI                           *
* AUTOR        : M.-A. BEISECKER                  DATUM: 21.07.1986 *
*************************************************************************

* EINSTELLEN DER SYSTEMPARAMETER UND INITIALISIERUNG DER VARIABLEN

SET TALK OFF
SET COLON OFF
STORE 0 TO AUSWAHL

* OEFFNEN DER DATEI ARTIKEL MIT INDEXDATEI ARTBEZ UND ARTNR

USE ARTIKEL INDEX ARTBEZ, ARTNR

* SCHLEIFE ZUM WIEDERHOLTEN DURCHLAUF DES MENUES

DO WHILE AUSWAHL >= 0

* AUFBAU DES MENUES

ERASE
TEXT

                        ARTIKELVERWALTUNG
                        -----------------

                    1. EINGABE NEUER ARTIKEL
```

```
                    2. EDITIEREN VON ARTIKELN
                    3. LOESCHEN VON ARTIKELN
                    4. DRUCKEN EINER LISTE

                    0. ZURUECK IN DAS VORHERIGE MENUE


                    BITTE WAEHLEN SIE:
ENDTEXT
@ 13,45 GET AUSWAHL PICTURE "9"
READ

* CASE - VERZWEIGUNG IN ABHAENGIGKEIT DER VARIABLEN AUSWAHL

DO CASE
   CASE AUSWAHL = 1
        APPEND
   CASE AUSWAHL = 2
        DO EDIART
   CASE AUSWAHL = 3
        DO LOEART
   CASE AUSWAHL = 4
        SET PRINT ON
        SET CONSOLE OFF
        SET EJECT OFF
        REPORT FORM ARTIKEL
        SET PRINT OFF
        SET CONSOLE ON
   CASE AUSWAHL = 0
        USE
        SET TALK ON
        SET COLON ON
        RETURN     * ENDE DES PROGRAMMS ARTVERW
ENDCASE

* ENDE CASE - VERZWEIGUNG

STORE 0 TO AUSWAHL
ENDDO

* ENDE DER SCHLEIFE
```

Artikel editieren mit dem Programm EDIART

Das Unterprogramm EDIART ermöglicht in Verbindung mit dem Programm ARTVERW, beliebige Artikel zu editieren. Die Auswahl des gewünschten Artikels kann dabei wahlweise über die Artikelnummer oder die Artikelbezeichnung erfolgen.

Dieses Programm kann nur über das Menü aus ARTVERW heraus gestartet werden, da im Programm ARTVERW das System eingestellt und die benötig-

ten Dateien eröffnet werden. Daher starten Sie das Programm mit DO ARTVERW <RETURN>:

```
      ARTIKELVERWALTUNG
      -----------------

   1. EINGABE NEUER ARTIKEL
   2. EDITIEREN VON ARTIKELN
   3. LOESCHEN VON ARTIKELN
   4. DRUCKEN EINER LISTE

   0. ZURUECK IN DAS VORHERIGE MENUE


   BITTE WAEHLEN SIE:2
```

Mit der Auswahl Zwei gelangen wir in das Menü des Programmes EDIART:

```
      EDITIEREN VON ARTIKELN
      ----------------------

   1. AUSWAHL NACH ARTIKELNUMMER
   2. AUSWAHL NACH ARTIKELBEZEICHNUNG



   0. ZURUECK IN DAS VORHERIGE MENUE


   BITTE WAEHLEN SIE:1
```

Die Auswahl Eins führt zu folgender Abfrage:

```
   WELCHE ARTIKELNUMMER (0=ENDE) ? 0003
```

Hier geben wir die Nummer des Artikels ein, den wir editieren möchten. Nach der Eingabe der Nummer Drei sehen wir das Editiermenü:

```
   Satznummer 00001
   ARTIKELNR     3
   ARTIKELBEZColour Monitor
   BESTAND       3
   MINBESTAND    3
   EK          798.00
   VK          998.00
```

Dieses Menü wird durch den EDIT-Befehl erzeugt. Daher können Sie die gleichen Tastenkombinationsbefehle anwenden, als wenn Sie den EDIT-Befehl direkt eingegeben hätten.

Durch <^Q> oder <^W> gelangen Sie wieder zu der Abfrage nach der Artikelnummer. Die Eingabe einer Null bringt uns zurück in das Ausgangsmenü. An dieser Stelle werden Sie durch die Eingabe einer Zwei nach der gewünschten Artikelbezeichnung gefragt:

```
WELCHER ARTIKEL (*=ENDE) ?
```

Diesmal müssen wir die gesuchte Artikelbezeichnung eingeben, unter Berücksichtigung der Groß- und Kleinschrift. Das Programm benötigt nur die ersten Stellen der Bezeichnung, um den gesuchten Artikel zu finden. Die Eingabe von 5 1/4 bewirkt die Ausgabe:

```
Satznummer 00002
ARTIKELNR     6
ARTIKELBEZ5 1/4 Zoll Disketten
BESTAND     200
MINBESTAND  150
EK            3.96
VK            4.90
```

Leider ist das Programm nur in der Lage, den ersten Artikel zu finden, dessen Bezeichnung mit der eingegebenen Zeichenfolge übereinstimmt. Daher müssen wir entweder so viele Stellen eingeben, bis nur ein Artikel gefunden werden kann, oder aber in der EDIT-Funktion blättern, um die nachfolgenden Datensätze anzusehen. Im nächsten Kapitel werden wir eine Erweiterung kennenlernen, mit der alle Datensätze gefunden werden, deren Suchfeld mit der Eingabe beginnt. Durch <^Q> oder <^W> gelangen wir aus dem EDIT-Menü zurück:

```
WELCHER ARTIKEL (*=ENDE) ? *
```

Die Eingabe des Sternes läßt uns zurück in das Auswahlmenü gelangen. Von dort aus kann durch die Eingabe der Null in das Hauptmenü zurückgekehrt werden. Diese sogenannte Benutzerführung bewirkt, daß kein dBASE-II-Befehl mehr eingegeben werden muß, nachdem das Programm ARTVERW gestartet wurde. Selbst jemand, der noch nie vorher mit einem Computer gearbeitet hat, kann so innerhalb kürzester Zeit mit dem Programm arbeiten.

Und so ist dieses Programm aufgebaut:

```
**************************************************************************
* UNTERPROGRAMM: EDIART                                                  *
* FUNKTION     : ERMOEGLICHT DIE DATEI ARTIKEL ZU EDITIEREN              *
* AUTOR        : M.-A. BEISECKER                  DATUM: 22.07.1986 *
**************************************************************************

* INITIALISIERUNG DER VARIABLEN
```

```
STORE 0 TO AUSWAHL

* SCHLEIFE 1 ZUM WIEDERHOLTEN DURCHLAUF DES MENUES

DO WHILE AUSWAHL >= 0

* AUFBAU DES MENUES
ERASE
TEXT

                    EDITIEREN VON ARTIKELN
                    ----------------------

                1. AUSWAHL NACH ARTIKELNUMMER
                2. AUSWAHL NACH ARTIKELBEZEICHNUNG



                0. ZURUECK IN DAS VORHERIGE MENUE


                BITTE WAEHLEN SIE:
ENDTEXT

@ 13,45 GET AUSWAHL PICTURE "9"
READ
```

Der erste Teil unterscheidet sich kaum von dem uns bekannten Programm ARTVERW. Die Unterschiede bestehen im Kommentar und in den unterschiedlichen Menüs. Alle weiteren Programme, die wir kennenlernen werden, sind in diesem Teil fast gleich. Die eigentliche Funktion findet sich in der CASE-Konstruktion. Daher werden wir diese näher untersuchen:

```
* CASE - VERZWEIGUNG IN ABHAENGIGKEIT DER VARIABLEN AUSWAHL

DO CASE
   CASE AUSWAHL = 1

        * INITIALISIERUNG

        STORE "1" TO ARTIKEL

        * DIE INDEXDATEI ARTNR WIRD AKTIVIERT

        SET INDEX TO ARTNR

        * SCHLEIFE 2 ZUM WIEDERHOLTEN DURCHLAUF DER EDITIERFUNKTION

        DO WHILE ARTIKEL > "0"
           ERASE
           ACCEPT "WELCHE ARTIKELNUMMER (0=ENDE) ?" TO ARTIKEL
           FIND &ARTIKEL

           * DIE SATZNUMMER # WIRD VON DEM FIND-BEFEHL GESETZT.
```

```
      * WIRD DER GESUCHTE ARTIKEL NICHT GEFUNDEN, IST # = 0.

      IF # > 0
         EDIT #
      ENDIF

   * ENDE DER SCHLEIFE 2

   ENDDO
   SET INDEX TO ARTBEZ
```

Dies ist der Zweig, der durchlaufen wird, wenn Sie die Auswahl Eins "Editieren nach Artikelnummer" wählen. Nach der Initialisierung und Aktivierung der Indexdatei ARTNR wird der Bildschirm gelöscht und die gewünschte Artikelnummer abgefragt. Die DO WHILE-Schleife bewirkt, daß dieser Programmteil so lange durchlaufen wird, bis man einmal Null eingibt.

Gesucht wird nach der Artikelnummer mit dem FIND-Befehl. Diese sehr schnelle Suche geht nur in Verbindung mit einer Indexdatei. Als Ergebnis wird eine Satznummer in der Variablen # abgelegt. Verläuft die Suche nicht erfolgreich, wird # gleich null gesetzt. Daher wird der EDIT-Befehl nur dann aufgerufen, wenn # größer als null ist. Die Anwendung von EDIT in Verbindung mit # unterdrückt die Abfrage nach einer Satznummer.

Am Ende dieses Programmteils wird wieder die Indexdatei ARTBEZ aktiviert.

```
CASE AUSWAHL = 2
     STORE " " TO ARTIKEL

     * SCHLEIFE 3 ZUM WIEDERHOLTEN DURCHLAUF DER EDITIERFUNKTION

     DO WHILE ARTIKEL <> "*"
        ERASE
        ACCEPT "WELCHER ARTIKEL (*=ENDE) ?" TO ARTIKEL
        FIND &ARTIKEL

        * DIE SATZNUMMER # WIRD VON DEM FIND-BEFEHL GESETZT,
        * WIRD DER GESUCHTE ARTIKEL NICHT GEFUNDEN IST # = 0.

        IF # > 0
           EDIT #
        ENDIF
     ENDDO
```

Im Falle der Auswahl nach Artikelbezeichnung muß keine Indexdatei mehr aktiviert werden. Die benötigte Indexdatei wurde ja schon im Programm ARTVERW eröffnet. Damit Sie wissen, welche Eingabe erwartet wird, wurde die Benutzerführung leicht geändert.

An der großen Übereinstimmung zwischen den Programmen ARTVERW und EDIART zeigt sich der Vorteil einer standardisierten Programmierung. Durch den gleichen Aufbau läßt sich ein neues Programm leicht aus einem alten Programm entwickeln. Wenn in Zukunft in einem Programm nach einem bestimmten Datensatz gesucht wird, geschieht dies immer nach der oben beschriebenen Methode. Kleine Änderungen können so aus einem Editierprogramm ein Löschprogramm machen.

Zur Eingabe in den Computer hier noch einmal das vollständige Programm EDIART:

```
**************************************************************************
* UNTERPROGRAMM: EDIART                                                  *
* FUNKTION     : ERMOEGLICHT DIE DATEI ARTIKEL ZU EDITIEREN              *
* AUTOR        : M.-A. BEISECKER                  DATUM: 22.07.1986 *
**************************************************************************

* INITIALISIERUNG DER VARIABLEN

STORE 0 TO AUSWAHL

* SCHLEIFE 1 ZUM WIEDERHOLTEN DURCHLAUF DES MENUES

DO WHILE AUSWAHL >= 0

* AUFBAU DES MENUES

ERASE
TEXT

                    EDITIEREN VON ARTIKELN
                    ----------------------

                1. AUSWAHL NACH ARTIKELNUMMER
                2. AUSWAHL NACH ARTIKELBEZEICHNUNG



                0. ZURUECK IN DAS VORHERIGE MENUE


                BITTE WAEHLEN SIE:
ENDTEXT

@ 13,45 GET AUSWAHL PICTURE "9"
READ

* CASE - VERZWEIGUNG IN ABHAENGIGKEIT DER VARIABLEN AUSWAHL

DO CASE
   CASE AUSWAHL = 1

        * INITIALISIERUNG
```

```
        STORE "1" TO ARTIKEL

        * DIE INDEXDATEI ARTNR WIRD AKTIVIERT

        SET INDEX TO ARTNR

        * SCHLEIFE 2 ZUM WIEDERHOLTEN DURCHLAUF DER EDITIERFUNKTION

        DO WHILE ARTIKEL > "0"
           ERASE
           ACCEPT "WELCHE ARTIKELNUMMER (0=ENDE) ?" TO ARTIKEL
           FIND &ARTIKEL

           * DIE SATZNUMMER # WIRD VON DEM FIND-BEFEHL GESETZT,
           * WIRD DER GESUCHTE ARTIKEL NICHT GEFUNDEN IST # = 0.

           IF # > 0
              EDIT #
           ENDIF

        * ENDE DER SCHLEIFE 2

        ENDDO
        SET INDEX TO ARTBEZ

   CASE AUSWAHL = 2
        STORE " " TO ARTIKEL

        * SCHLEIFE 3 ZUM WIEDERHOLTEN DURCHLAUF DER EDITIERFUNKTION

        DO WHILE ARTIKEL <> "*"
           ERASE
           ACCEPT "WELCHER ARTIKEL (*=ENDE) ?" TO ARTIKEL
           FIND &ARTIKEL

           * DIE SATZNUMMER # WIRD VON DEM FIND-BEFEHL GESETZT,
           * WIRD DER GESUCHTE ARTIKEL NICHT GEFUNDEN IST # = 0.

           IF # > 0
              EDIT #
           ENDIF
        ENDDO
   CASE AUSWAHL = 0
        RETURN     * ENDE DES PROGRAMMS EDIART
ENDCASE

* ENDE CASE - VERZWEIGUNG

STORE 0 TO AUSWAHL

ENDDO

* ENDE DER SCHLEIFE 1
```

Löschen von Artikeln mit dem Programm LOEART

Das Programm LOEART kann wie das Programm EDIART nur vom Programm ARTVERW aus aufgerufen werden. Nachdem das Programm ARTVERW gestartet und die Auswahl Drei eingegeben wurde, erscheint folgendes Menü:

```
      LOESCHEN VON ARTIKELN
      ---------------------

   1. AUSWAHL NACH ARTIKELNUMMER
   2. AUSWAHL NACH ARTIKELBEZEICHNUNG


   0. ZURUECK IN DAS VORHERIGE MENUE


   BITTE WAEHLEN SIE:2
```

Wählen wir die Zwei und geben den Namen eines Artikels ein, den wir löschen möchten:

```
WELCHER ARTIKEL (*=ENDE) ? 5 1/4
```

Sofort wird der Artikel angezeigt, und wir können entscheiden, ob er gelöscht werden soll oder nicht:

```
00002      6 5 1/4 Zoll Disketten   200   150    3.96    4.90
SOLL DIESER SATZ GELÖSCHT WERDEN (J/N) ? N
```

Geben wir lieber Nein ein, vielleicht brauchen wir ihn später noch. An dieser Stelle wird ein Satz nur mit DELETE zur Löschung markiert. Erst wenn wir alle gewünschten Sätze markiert haben und die Funktion durch die Eingabe des Sterns verlassen, werden durch PACK alle markierten Sätze auf einmal gelöscht.

Der Aufbau des Programms ist analog zu dem Programm EDIART:

```
*************************************************************************
* UNTERPROGRAMM: LOEART                                                 *
* FUNKTION     : ERMOEGLICHT ARTIKEL ZU LOESCHEN                        *
* AUTOR        : M.-A. BEISECKER                  DATUM: 22.07.1986 *
*************************************************************************

* INITIALISIERUNG DER VARIABLEN

STORE 0 TO AUSWAHL
```

```
* SCHLEIFE 1 ZUM WIEDERHOLTEN DURCHLAUF DES MENUES

DO WHILE AUSWAHL >= 0

* AUFBAU DES MENUES

ERASE
TEXT
                        LOESCHEN VON ARTIKELN
                        ---------------------

                    1. AUSWAHL NACH ARTIKELNUMMER
                    2. AUSWAHL NACH ARTIKELBEZEICHNUNG



                    0. ZURUECK IN DAS VORHERIGE MENUE


                    BITTE WAEHLEN SIE:
ENDTEXT
@ 13,45 GET AUSWAHL PICTURE "9"
READ

* CASE - VERZWEIGUNG IN ABHAENGIGKEIT DER VARIABLEN AUSWAHL

DO CASE
   CASE AUSWAHL = 1

        * INITIALISIERUNG

        STORE "1" TO ARTIKEL

        * DIE INDEXDATEI ARTNR WIRD AKTIVIERT

        SET INDEX TO ARTNR

        * SCHLEIFE 2 ZUM WIEDERHOLTEN DURCHLAUF DER LOESCHFUNKTION

        DO WHILE ARTIKEL > "0"
           ERASE
           ACCEPT "WELCHE ARTIKELNUMMER (0=ENDE) ?" TO ARTIKEL
           FIND &ARTIKEL

           * DIE SATZNUMMER # WIRD VON DEM FIND-BEFEHL GESETZT,
           * WIRD DER GESUCHTE ARTIKEL NICHT GEFUNDEN IST # = 0.

           IF # > 0
              DISPLAY
           ACCEPT "SOLL DIESER ARTIKEL GELOESCHT WERDEN (J/N) ?" TO JA
              IF JA="j" .OR. JA="J"
                 DELETE
              ENDIF
           ENDIF

        * ENDE DER SCHLEIFE 2
```

```
        ENDDO

        * ALLE MARKIERTEN SAETZE WERDEN AUF EINMAL GELOESCHT.

        PACK

        * DIE INDEXDATEI ARTBEZ WIRD WIEDER AKTIVIERT.

        SET INDEX TO ARTBEZ

   CASE AUSWAHL = 2
        * INITIALISIERUNG

        STORE " " TO ARTIKEL

        * SCHLEIFE 3 ZUM WIEDERHOLTEN DURCHLAUF DER LOESCHFUNKTION

        DO WHILE ARTIKEL <> "*"
           ERASE
           ACCEPT "WELCHER ARTIKEL (*=ENDE) ?" TO ARTIKEL
           FIND &ARTIKEL

           * DIE SATZNUMMER # WIRD VON DEM FIND-BEFEHL GESETZT,
           * WIRD DER GESUCHTE ARTIKEL NICHT GEFUNDEN, IST # = 0.

           IF # > 0
              DISPLAY
           ACCEPT "SOLL DIESER ARTIKEL GELOESCHT WERDEN (J/N) ?" TO JA
              IF JA="J" .OR. JA="j"
                 DELETE
              ENDIF
           ENDIF
        ENDDO

        * ALLE MARKIERTEN SAETZE WERDEN AUF EINMAL GELOESCHT

        PACK

   CASE AUSWAHL = 0
        RETURN     * ENDE DES PROGRAMMS LOEART
ENDCASE

* ENDE CASE - VERZWEIGUNG

STORE 0 TO AUSWAHL
ENDDO

* ENDE DER SCHLEIFE 1
```

Nachdem die Artikelnummer oder die Artikelbezeichnung eingegeben wurde, beginnt die Suche nach diesem Schlüssel. Falls ein Satz gefunden wird, zeigt das Programm ihn am Bildschirm mit DISPLAY an. Der FIND-Befehl gibt nicht nur die Satznummer des gefundenen Satzes in der Systemvariablen ##

aus, sondern setzt den Zeiger der Datei auch auf diese Satznummer. Daher reicht der DISPLAY-Befehl ohne weiteren Zusatz aus.

Die IF-Abfrage überprüft, ob ein großes oder kleines J eingegeben wurde. In diesem Fall wird der aktuelle Satz zur Löschung markiert. Vor dem Verlassen der jeweiligen CASE-Verzweigung werden die markierten Sätze gelöscht.

An diesem Programm kann man schon recht gut erkennen, wie wichtig es ist, die Befehle bei Verschachtelungen einzurücken. Durch diese Darstellung weiß man sofort, welche Bereiche des Programms noch innerhalb einer Schleife oder einer IF-Abfrage sind und welche nicht.

Wir haben jetzt die gesamte Artikelverwaltung kennengelernt. Vielleicht finden Sie das Programm ganz interessant, benötigen aber keine Artikel-, sondern eine Videocassettenverwaltung. Das Programmpaket ist so allgemein gehalten, daß lediglich die Dateinamen ausgetauscht und neue Daten- und Indexdateien angelegt werden müssen, um etwas anderes zu verwalten. Wenn dann noch mit dem REPORT-Befehl eine dazu passende Liste erzeugt und deren Namen im Programm ARTVERW geändert wird, ist das neue Programmpaket fertig. Am Beispiel einer Adressenverwaltung wird dies im nächsten Kapitel gezeigt.

Kapitel 8

Eine vollständige Adressenverwaltung

In diesem Kapitel werden folgende Themen behandelt:

Das Menüprogramm KUNVERW
Adressen editieren mit EDIKUN
Löschen von Adressen mit LOEKUN
Drucken von Etiketten und Listen mit DRUKUN

Das Menüprogramm KUNVERW

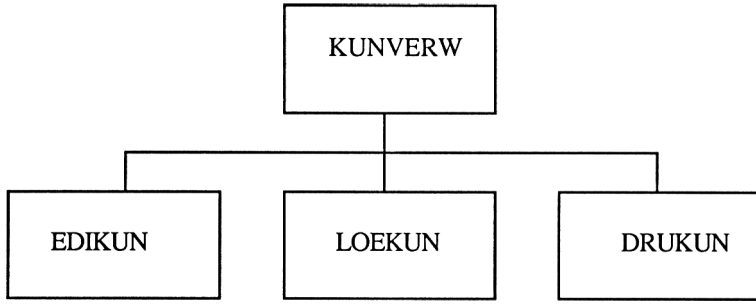
In diesem Kapitel wird ein vollständiges Programmpaket zur Verwaltung von Adressen vorgestellt. Da dieses Programmpaket später in die Rechnungsschreibung eingegliedert wird und die Datei KUNDEN eine Adressendatei ist, wurden die Programme zur Verwaltung von Kundenadressen geschrieben. Durch das Anlegen einer neuen Dateistruktur und einige kleine Änderungen in den Programmen läßt sich die Adressenverwaltung jedoch leicht an jede Adressendatei anpassen.

Wie bei der Artikelverwaltung wurden auch bei der Adressenverwaltung die verschiedenen Aufgaben auf Unterprogramme verteilt. Zusätzlich zu den Programmen, die die Editierung und Löschung von Adressen ermöglichen, kann vom Menüprogramm aus ein Druckprogramm aufgerufen werden. Dieses Druckprogramm ermöglicht es Etiketten nach Kundennummern auswählbar, oder eine Liste aller Kunden auszudrucken.

Das Programm KUNVERW enthält das Hauptmenü und die Routine zur Erfassung neuer Kunden. Damit das Programm lauffähig ist, muß die Indexdatei KUNDENNR angelegt werden. Diese Datei ist, wie der Name vermuten läßt, nach Kundennummern indiziert:

```
. USE KUNDEN
. INDEX ON KUNDENNR TO KUNDENNR
. USE
```

Hier die graphische Darstellung des Programmpaketes:



Nachdem das Programm eingegeben ist, kann es mit DO KUNVERW <RETURN> gestartet werden. Die Bedienung des Programms erfolgt analog zur Artikelverwaltung. Vom Aufbau unterscheidet es sich nur geringfügig von dem Menüprogramm ARTVERW. Daher wird auf eine detaillierte Beschreibung des Programms verzichtet und das Programm an dieser Stelle abgedruckt:

```
**************************************************************************
* UNTERPROGRAMM: KUNVERW                                                 *
* FUNKTION     : VERWALTEN EINER KUNDENDATEI                             *
* AUTOR        : M.-A. BEISECKER                  DATUM: 19.07.1986 *
**************************************************************************

* EINSTELLEN DER SYSTEMPARAMETER UND INITIALISIERUNG DER VARIABLEN

SET TALK OFF
SET COLON OFF
STORE 0 TO AUSWAHL

* OEFFNEN DER DATEI KUNDEN MIT INDEXDATEI NAME, PLZ UND KUNDENNR

USE KUNDEN INDEX NAME, PLZ, KUNDENNR

* SCHLEIFE ZUM WIEDERHOLTEN DURCHLAUF DES MENUES

DO WHILE AUSWAHL >= 0

* AUFBAU DES MENUES

ERASE
TEXT
```

```
                        KUNDENVERWALTUNG
                        ----------------

                     1. EINGABE NEUER KUNDEN
                     2. EDITIEREN VON KUNDEN
                     3. LOESCHEN VON KUNDEN
                     4. DRUCKEN VON LISTEN

                     0. ZURUECK IN DAS VORHERIGE MENUE


                     BITTE WAEHLEN SIE:
ENDTEXT

@ 13,45 GET AUSWAHL PICTURE "9"
READ

*CASE-VERZWEIGUNG IN ABHAENGIGKEIT DER VARIABLEN AUSWAHL

DO CASE
   CASE AUSWAHL = 1
        APPEND
   CASE AUSWAHL = 2
        DO EDIKUN
   CASE AUSWAHL = 3
        DO LOEKUN
   CASE AUSWAHL = 4
        DO DRUKUN
   CASE AUSWAHL = 0
        USE
        SET TALK ON
        SET COLON ON
        RETURN     * ENDE DES PROGRAMMS KUNVERW
ENDCASE

* ENDE CASE - VERZWEIGUNG

STORE 0 TO AUSWAHL
ENDDO

* ENDE DER SCHLEIFE
```

Wer dieses Programm an seine eigene Adressendatei anpassen will, muß lediglich die Namen der Datei KUNDEN und der zugehörigen Indexdateien gegen die entsprechenden Namen seiner Adressendatei und der zugehörigen Indexdateien austauschen. Dies könnte dann zum Beispiel so aussehen:

```
USE ADRESSEN INDEX NAME, PLZ, MERKMAL
```

Zusätzlich kann man durch Verändern des Textblockes und der Kommentare diese Änderung für den Benutzer bzw. Programmierer dokumentieren. Etwas schwieriger werden die Änderungen in den folgenden Programmen, da dort Feldnamen der Datei KUNDEN verwendet werden, die in Ihrer Datei-

struktur möglicherweise nicht vorkommen oder eine andere Bedeutung haben. In den meisten Fällen können Sie die Feldnamen einfach gegen entsprechende aus Ihrer Dateistruktur austauschen. An den entsprechenden Programmteilen wird auf Änderungsmöglichkeiten hingewiesen.

Adressen editieren mit EDIKUN

Das Unterprogramm EDIKUN ermöglicht in Verbindung mit dem Programm KUNVERW beliebige Kundenadressen zu editieren. Die Auswahl der gewünschten Adresse erfolgt dabei entweder über die Kundennummer oder den Nachnamen des Kunden. Dabei werden alle Adressen der Kunden angezeigt, die den eingegebenen Namen haben oder deren Namen mit der eingegebenen Zeichenfolge beginnt. So werden nach der Eingabe von "M" als Namen alle Kunden angezeigt, deren Namen mit M beginnt. Damit kann dieses Programm auch dazu dienen, die Adresse eines Kunden zu finden, dessen Namen man nicht mehr genau weiß.

Sehen wir uns das Programm einmal näher an:

```
**************************************************************************
* UNTERPROGRAMM: EDIKUN                                                  *
* FUNKTION     : ERMOEGLICHT DIE DATEI KUNDEN ZU EDITIEREN               *
* AUTOR        : M.-A. BEISECKER                    DATUM: 20.07.1986 *
**************************************************************************

* INITIALISIERUNG DER VARIABLEN

STORE 0 TO AUSWAHL

* SCHLEIFE 1 ZUM WIEDERHOLTEN DURCHLAUF DES MENUES

DO WHILE AUSWAHL >= 0

* AUFBAU DES MENUES

ERASE
TEXT


                    EDITIEREN VON KUNDEN
                    --------------------

                  1. AUSWAHL NACH KUNDENNUMMER
                  2. AUSWAHL NACH KUNDENNAME


                 0. ZURUECK IN DAS VORHERIGE MENUE
```

```
                  BITTE WAEHLEN SIE:
ENDTEXT
@ 13,45 GET AUSWAHL PICTURE "9"
READ

* CASE-VERZWEIGUNG IN ABHAENGIGKEIT DER VARIABLEN AUSWAHL

DO CASE
   CASE AUSWAHL = 1

        * INITIALISIERUNG

        STORE "1" TO KUNDE

        * DIE INDEXDATEI KUNDENNR WIRD AKTIVIERT

        SET INDEX TO KUNDENNR

        * SCHLEIFE 2 ZUM WIEDERHOLTEN DURCHLAUF DER EDITIERFUNKTION

        DO WHILE KUNDE > "0"
           ERASE
           ACCEPT "WELCHE KUNDENNUMMER (0=ENDE) ?" TO KUNDE
           FIND &KUNDE

           * DIE SATZNUMMER # WIRD VON DEM FIND-BEFEHL GESETZT,
           * WIRD DER GESUCHTE KUNDE NICHT GEFUNDEN IST # = 0.

           IF # > 0
              EDIT #
           ENDIF

        * ENDE DER SCHLEIFE 2

        ENDDO
        SET INDEX TO NAME

   CASE AUSWAHL = 2
        STORE " " TO KUNDE

        * SCHLEIFE 3 ZUM WIEDERHOLTEN DURCHLAUF DER EDITIERFUNKTION

        DO WHILE KUNDE <> "*"
           ERASE
           ACCEPT "WELCHER KUNDE (*=ENDE) ?" TO KUNDE
           FIND &KUNDE

           * DIE SATZNUMMER # WIRD VON DEM FIND-BEFEHL GESETZT,
           * WIRD DER GESUCHTE KUNDE NICHT GEFUNDEN IST # = 0.

           IF # > 0
              EDIT #

              * DATEIENDE ERREICHT ?

              IF .NOT. EOF
```

```
            SKIP
         ENDIF
      ENDIF

      * SCHLEIFE 4 ZUR ANZEIGE DER RESTLICHEN DATENSAETZE.

      DO WHILE @(KUNDE,NACHNAME) > 0
         EDIT #
         IF .NOT. EOF
            SKIP
         ENDIF
      ENDDO

      * ENDE DER SCHLEIFE 4

   ENDDO

   * ENDE DER SCHLEIFE 3

CASE AUSWAHL = 0
   RETURN    * ENDE DES PROGRAMMS EDIKUN
ENDCASE

* ENDE CASE-VERZWEIGUNG

STORE 0 TO AUSWAHL
ENDDO

* ENDE DER SCHLEIFE 1
```

Bis zum Beginn des CASE-Konstruktes ist uns der Programmaufbau von dem Programm EDIART her vertraut. Nach dem Menüaufbau wird je nach Wahl in dem CASE-Konstrukt verzweigt. Wenn wir einen Kunden nach dessen Kundennummer auswählen, wird die entsprechende Indexdatei KUNDENNR aktiviert und der Kunde mit dem FIND-Befehl gesucht. Da jeder Kunde normalerweise eine andere Kundennummer besitzt, kann immer nur maximal ein Kunde gefunden werden. Dies entspricht, abgesehen von den anderen Feldbezeichnungen, exakt dem Algorithmus in dem Programm EDIART.

Der Einfachheit halber wurde bei der Editierung von Artikeln vorausgesetzt, daß es keine zwei Artikel mit derselben Artikelbezeichnung gibt. Diese Annahme können wir bei einer Adressendatei nicht machen, wenn über den Nachnamen auf eine Adresse zugegriffen werden soll.

Jedes Telefonbuch einer größeren Stadt enthält zum Beispiel viele Maier, Müller, Schmidts oder Schulze in verschiedensten Schreibvariationen. Vor allem bei diesen in Deutschland weit verbreiteten Namen kann es passieren, daß man sich nicht sicher ist, wie sich der oder die Betreffende schreibt. Man weiß nicht mehr, ob es Herr Maier, Herr Meier oder Herr Meyer war, dessen

Adresse man ändern oder ansehen möchte. In einem solchen Fall ist es sehr praktisch, wenn man nur ein M für den Nachnamen eingeben muß und alle Kunden angezeigt bekommt, die mit M beginnen.

Sehen wir uns den Teil des Programms genauer an, wo die Suche nach dem Nachnamen stattfindet:

```
CASE AUSWAHL = 2
       STORE " " TO KUNDE

       * SCHLEIFE 3 ZUM WIEDERHOLTEN DURCHLAUF DER EDITIERFUNKTION

       DO WHILE KUNDE <> "*"
          ERASE
          ACCEPT "WELCHER KUNDE (*=ENDE) ?" TO KUNDE
          FIND &KUNDE

          * DIE SATZNUMMER # WIRD VON DEM FIND-BEFEHL GESETZT,
          * WIRD DER GESUCHTE KUNDE NICHT GEFUNDEN IST # = 0.
```

Geben Sie einen Namen ein, wird wie bisher mit dem FIND-Befehl nach einem Datensatz gesucht, der diesen Namen enthält. Ist dieser Datensatz vorhanden, wird die Satznummer des Satzes in der Systemvariablen # abgelegt. Jetzt erfolgt die Abfrage ob # größer null ist, das bedeutet, daß ein Datensatz gefunden wurde:

```
IF # > 0
   EDIT #
   IF .NOT. EOF
      SKIP
   ENDIF
ENDIF
```

Der gefundene Datensatz wird mit EDIT zur Editierung bereitgestellt. Sollte es noch einen weiteren Datensatz geben, der den gesuchten Namen beinhaltet, so muß dieser in einer indizierten Datei direkt hinter dem aktuellen Datensatz sein.

Daher wird mit SKIP auf den logisch nächsten Datensatz gezeigt. Es kann nun sein, daß der aktuelle Datensatz als letzter Satz in der Datei steht. In diesem Fall würde der SKIP-Befehl eine Fehlermeldung auslösen, da es ja keinen folgenden Datensatz mehr gibt. Daher wird vor der Anwendung von SKIP mit Hilfe des IF-Befehls überprüft, ob das Dateiende erreicht ist. Wenn der Dateizeiger am Ende der Datei steht, wird die booleschen Variable EOF auf True gesetzt. Der Name EOF steht für die englischen Wörter End Of File, was übersetzt Dateiende bedeutet.

Gibt es einen weiteren Datensatz, müssen wir überprüfen, ob der eingegebene Name in ihm zumindest teilweise vorkommt:

```
      DO WHILE @(KUNDE,NACHNAME) > 0
         EDIT #
         IF .NOT. EOF
            SKIP
         ENDIF
      ENDDO
   ENDDO
```

Dies wird hier durch die @()-Funktion realisiert. Mit dieser Funktion läßt sich die Position einer gegebenen Zeichenkette oder Variablen in einer zweiten Zeichenkette oder Variablen lokalisieren. Wenn wir die Variablen in der Funktion durch Zeichenketten ersetzen, wird uns dies schnell deutlich:

```
@("M","Meyer")
```

Das Ergebnis wäre eine Eins, da "M" in der Zeichenkette "Meyer" an der ersten Stelle steht. Findet die Funktion keine Übereinstimmung, ist das Ergebnis Null.

Ist das Ergebnis größer eins, wird EDIT aufgerufen, und wir können den Datensatz editieren. Danach wird der Dateizeiger auf den nächsten Datensatz positioniert, falls das Dateiende noch nicht erreicht wurde.

Solange das Dateiende nicht erreicht ist und der nächste Satz die Bedingung erfüllt, wird die Schleife 4 durchlaufen.

Dieses Programm läßt sich leicht erweitern. Man kann weitere Menüpunkte und CASE-Verzweigungen einbauen, um eine Adresse auch nach einer bestimmten Straße oder Postleitzahl suchen zu können. Wir müssen dazu nur die benötigte Indexdatei anlegen, falls sie wie die PLZ-Datei nicht schon vorhanden ist, und im Programm KUNVERW bei der Eröffnung der Datei mit angeben. Die neue Indexdatei wird dann vor dem Such- und Editieralgorithmus aktiviert und die Benutzerführung hinter dem ACCEPT-Befehl geändert. Dabei muß allerdings beachtet werden, daß nicht mehr als 7 Indexdateien pro Datendatei gleichzeitig eröffnet sein dürfen. Im nächsten Programm LOEKUN werden wir den hier besprochenen Algorithmus in erweiterter Form wiederfinden.

Löschen von Adressen mit LOEKUN

Das Programm LOEKUN ermöglicht als Unterprogramm von KUNVERW Adressen von Kunden zu löschen. Die Adressen werden wie beim Editieren

nach Kundennummer oder Kundenname ausgewählt. Eine gefundene Adresse zeigt das Programm am Bildschirm an, und der Benutzer kann entscheiden, ob diese Adresse wirklich gelöscht werden soll. Auch dieses Programm findet bei der Auswahl nach Kundennamen alle Datensätze, die den eingegebenen Namen enthalten.

Anpassungen an eine andere Dateistruktur lassen sich aufgrund der normierten Programmierung leicht vornehmen. Der Programmaufbau entspricht – abgesehen von der Erweiterung, mehrere Datensätze zu finden – dem des Programms LOEART, so daß eine detaillierte Besprechung des Programms entfallen kann.

```
************************************************************************
* UNTERPROGRAMM: LOEKUN                                                *
* FUNKTION     : ERMOEGLICHT KUNDEN ZU LOESCHEN                        *
* AUTOR        : M.-A. BEISECKER                  DATUM: 20.07.1986 *
************************************************************************

* INITIALISIERUNG DER VARIABLEN

STORE 0 TO AUSWAHL

* SCHLEIFE 1 ZUM WIEDERHOLTEN DURCHLAUF DES MENUES

DO WHILE AUSWAHL >= 0

* AUFBAU DES MENUES

ERASE
TEXT


                    LOESCHEN VON KUNDEN
                    -------------------

                  1. AUSWAHL NACH KUNDENNUMMER
                  2. AUSWAHL NACH KUNDENNAME



                 0. ZURUECK IN DAS VORHERIGE MENUE


                 BITTE WAEHLEN SIE:
ENDTEXT
@ 13,45 GET AUSWAHL PICTURE "9"
READ

* CASE-VERZWEIGUNG IN ABHAENGIGKEIT DER VARIABLEN AUSWAHL

DO CASE
   CASE AUSWAHL = 1
```

```
      * INITIALISIERUNG

      STORE "1" TO KUNDE

      *DIE INDEXDATEI KUNDENNR WIRD AKTIVIERT

      SET INDEX TO KUNDENNR

      * SCHLEIFE 2 ZUM WIEDERHOLTEN DURCHLAUF DER LOESCHFUNKTION

      DO WHILE KUNDE > "0"
         ERASE
         ACCEPT "WELCHE KUNDENNUMMER (0=ENDE) ?" TO KUNDE
         FIND &KUNDE

         * DIE SATZNUMMER # WIRD VON DEM FIND-BEFEHL GESETZT,
         * WIRD DER GESUCHTE KUNDE NICHT GEFUNDEN IST # = 0.

         IF # > 0
            DISPLAY
         ACCEPT "SOLL DIESER KUNDE GELOESCHT WERDEN (J/N) ?" TO JA
            IF JA="j" .OR. JA="J"
               DELETE
            ENDIF
         ENDIF

      * ENDE DER SCHLEIFE 2

      ENDDO

      * DIE INDEXDATEI NAME WIRD WIEDER AKTIVIERT

      SET INDEX TO NAME

CASE AUSWAHL = 2

      * INITIALISIEREN

      STORE " " TO KUNDE

      * SCHLEIFE 3 ZUM WIEDERHOLTEN DURCHLAUF DER LOESCHFUNKTION

      DO WHILE KUNDE <> "*"
         ERASE
         ACCEPT "WELCHER KUNDE (*=ENDE) ?" TO KUNDE
         FIND &KUNDE

         * DIE SATZNUMMER # WIRD VON DEM FIND-BEFEHL GESETZT,
         * WIRD DER GESUCHTE KUNDE NICHT GEFUNDEN IST # = 0.

         IF # > 0
            DISPLAY
         ACCEPT "SOLL DIESER KUNDE GELOESCHT WERDEN (J/N) ?" TO JA
            IF JA="J" .OR. JA="j"
               DELETE
            ENDIF
```

```
            IF .NOT. EOF
               SKIP
            ENDIF
         ENDIF

         * SCHLEIFE 4 ERMÖGLICHT MEHRERE DATENSÄTZE ZU FINDEN

         DO WHILE @(KUNDE,NACHNAME) > 0
            DISPLAY
         ACCEPT "SOLL DIESER KUNDE GELOESCHT WERDEN (J/N) ?" TO JA
            IF JA="J" .OR. JA="j"
               DELETE
            ENDIF
            IF .NOT. EOF
               SKIP
            ENDIF
         ENDDO

      * ENDE DER SCHLEIFE 4

      ENDDO

      * ENDE DER SCHLEIFE 3

      * ALLE MARKIERTEN SAETZE WERDEN AUF EINMAL GELOESCHT

      PACK

   CASE AUSWAHL = 0
      RETURN     * ENDE DES PROGRAMMS LOEKUN
ENDCASE

* ENDE CASE-VERZWEIGUNG

STORE 0 TO AUSWAHL

ENDDO

* ENDE DER SCHLEIFE 1
```

Drucken von Etiketten und Listen mit DRUKUN

Mit Hilfe des REPORT-Befehls kann eine Liste aus einem Programm heraus gedruckt werden. Trotz der vielen Möglichkeiten die REPORT zur Gestaltung einer Liste bietet, gibt es Druckausgaben, die sich damit nicht realisieren lassen. Um etwa Adreßaufkleber zu drucken, müssen die Feldinhalte eines Datensatzes nicht nebeneinander, wie beim REPORT-Befehl, sondern untereinander ausgedruckt werden.

Daher wird im Programm DRUKUN die Kundenliste mit dem REPORT-Befehl erzeugt, die Etikettenausgabe jedoch programmiert. Sehen wir uns an,

wie dieses Unterprogramm bedient wird. Nach dem Aufruf des Programms über KUNVERW sehen wir dieses Menü:

```
      DRUCKPROGRAMME
      --------------

 1. LISTE NACH NAMEN
 2. ETIKETTEN NACH KUNDENNR


 0. ZURUECK IN DAS VORHERIGE MENUE


 BITTE WAEHLEN SIE:
```

Wählen wir Menüpunkt Eins, wird eine Kundenliste sortiert nach Nachnamen auf dem Drucker ausgegeben. Bevor wir Eins eingeben, sollten wir sicher sein, daß der Drucker eingeschaltet und betriebsbereit ist, da sonst eine Fehlermeldung erfolgt. Interessanter ist Menüpunkt Zwei, da wir nach seiner Wahl zu folgender Abfrage kommen:

```
VON KUNDENNR:
```

Hier geben wir die Kundennummer ein, ab der Etiketten gedruckt werden sollen. Bei der nächsten Frage wird eingegeben, bis zu welcher Kundennummer ausgedruckt werden soll:

```
BIS KUNDENNR:
```

Wir können jetzt noch wählen, wie viele Etiketten von jedem Kunden gedruckt werden:

```
WIE VIELE KOPIEN ? 1
WUENSCHEN SIE EINEN PROBEDRUCK ? J
```

Die Benutzerführung und Eingabe wurde mit dem ACCEPT-Befehl programmiert. Dieser Befehl erzeugt nur alphanumerische Variablen. Da mit den eingegebenen Werten auch gerechnet wird, mußten die alphanumerischen Eingaben in numerische umgewandelt und in entsprechenden Variablen abgespeichert werden:

```
ACCEPT "VON KUNDENNR:" TO ANFANG
ACCEPT "BIS KUNDENNR:" TO ENDE
ACCEPT "WIE VIELE KOPIEN ?" TO ANZAHL
ACCEPT "WUENSCHEN SIE EINEN PROBEDRUCK ?" TO JA

* UMWANDLUNG VON ALPHANUMERISCH IN NUMERISCH

STORE VAL(ANFANG) TO N:ANFANG
STORE VAL(ENDE)   TO N:ENDE
STORE VAL(ANZAHL) TO N:ANZAHL
```

Die Abfrage hätte mit dem INPUT-Befehl direkt numerische Variablen ergeben. Wie wir später sehen, benötigen wir die alphanumerischen Variablen unter anderem für den Befehl FIND.

Wenn ein Probedruck gewünscht wird, werden auf dem Drucker drei Etiketten wie folgend bedruckt:

```
VORNAME... NACHNAME.......
STRASSE HAUSNR.
PLZ. ORT.........
```

Diese Probeetiketten benutzt man zum Einstellen des Endlosformulars. Man sieht so leicht, ob der Ausdruck auf die Etiketten paßt oder das Formular etwas nach oben, unten oder zur Seite hin verschoben werden muß. Nach dem Ausdruck können wir entscheiden, ob wir einen weiteren Probeausdruck benötigen oder nicht:

```
WUENSCHEN SIE EINEN WEITEREN PROBEAUSDRUCK ? N
```

Verneinen wir diese Frage, werden die Etiketten wie gewünscht ausgedruckt. Dies wird im Programm durch mehrere Schleifen realisiert:

```
* SCHLEIFE 2 ZUR WIEDERHOLUNG DES PROBEDRUCKES

DO WHILE JA="J" .OR. JA="j"
   SET PRINT ON
   SET CONSOLE OFF
   STORE 1 TO ZAEHLER

   * SCHLEIFE 3 ZUR AUSGABE VON 3 ETIKETTEN

   DO WHILE ZAEHLER < 4
      ? "VORNAME... NACHNAME......."
      ? "STRASSE HAUSNR."
      ?
      ? "PLZ. ORT........."
      ?
      ?
      STORE ZAEHLER + 1 TO ZAEHLER
   ENDDO

   * ENDE DER SCHLEIFE 2

   SET PRINT OFF
   SET CONSOLE ON
   ACCEPT "WUENSCHEN SIE EINEN WEITEREN PROBEAUSDRUCK ?" TO JA
ENDDO

* ENDE DER SCHLEIFE 2
```

Vor der Schleife Zwei wurde die Variable JA durch unsere Eingabe auf J gesetzt, so daß der Probeausdruck in der Schleife Drei gestartet wird. Nach diesem Ausdruck werden wir wieder gefragt, ob ein Probeausdruck gewünscht wird, dabei wird in der Variablen JA unsere Eingabe abgelegt. Die äußere Schleife Zwei wird so lange durchlaufen, bis diese Frage mit einer anderen Eingabe als "J" oder "j" beantwortet wird. Dann wird die erste angegebene Kundennummer gesucht und der Etikettendruck gestartet:

```
* INITIALISIEREN

STORE N:ANZAHL TO ZAEHLER

* INDEXDATEI KUNDENNR AKTIVIEREN

SET INDEX TO KUNDENNR

FIND &ANFANG
IF # > 0
   SET PRINT ON
   SET CONSOLE OFF

* SCHLEIFE 4 ZUM AUSDRUCK DER KUNDEN IM BEREICH ANFANG BIS ENDE

DO WHILE N:ANFANG <= N:ENDE .AND. .NOT. EOF

   * SCHLEIFE 5 ZUM WIEDERHOLTEN AUSDRUCK EINES KUNDEN

   DO WHILE ZAEHLER > 0
      ? TRIM(VORNAME)+" "+NACHNAME
      ? STRASSE
      ?
      ? PLZ+" "+ORT
      ?
      ?
      ?
      STORE ZAEHLER - 1 TO ZAEHLER
   ENDDO

   * ENDE DER SCHLEIFE 5

   SKIP
   STORE KUNDENNR TO N:ANFANG
   STORE N:ANZAHL TO ZAEHLER
ENDDO

* ENDE DER SCHLEIFE 4

   SET PRINT OFF
   SET CONSOLE ON
ENDIF

* INDEXDATEI NAME AKTIVIEREN

SET INDEX TO NAME
```

Nachdem die Indexdatei KUNDENNR aktiviert wurde, startet der FIND-Befehl die Suche nach der ersten gewünschten Kundennummer. Findet er diese, werden mit Hilfe der Schleife Vier so lange Etiketten gedruckt, bis die aktuelle Kundennummer größer als N:ENDE ist oder das Dateiende erreicht wird. Die aktuelle Kundennummer, die in der Variablen KUNDENNR steht, wird nach jedem Durchgang in N:ANFANG gespeichert. Da die Kundennummern nicht unbedingt im Abstand von eins aufeinanderfolgen, wurde hier kein Zähler verwendet.

Innerhalb der Schleife Vier sorgt die Schleife Fünf dafür, daß von jedem Kunden die gewünschte Anzahl Etiketten hintereinander gedruckt werden. Sollten Sie ein Etikettenformat verwenden, auf das der Ausdruck nicht paßt, so läßt sich das Programm durch die Ausgabe oder Einsparung einer oder mehrerer Druckzeilen anpassen. Die Ausgabe der Etiketten geschieht mit dem ?-Befehl:

```
? TRIM(VORNAME)+" "+NACHNAME
? STRASSE
?
? PLZ+" "+ORT
?
?
?
```

Fügen wir ein ? hinzu, wird ein weiterer Zeilenvorschub pro Etikett erzeugt. Indem wir ein ? entfernen, verkleinern wir das Etikettenformat um eine Druckzeile. Vergessen Sie nicht den Ausdruck der Probeetiketten ebenfalls entsprechend anzupassen!

Die Felder können auch in einer anderen Anordnung ausgedruckt werden. Da einige Druckzeilen nicht durch Feldinhalte belegt sind, könnten die Etiketten noch weitere Felder enthalten, zum Beispiel die Kundennummer.

Wenn Sie das nun folgende Programm eingegeben haben, ist die Adressenverwaltung vollständig:

```
**************************************************************************
* UNTERPROGRAMM: DRUKUN                                                  *
* FUNKTION     : DRUCKEN VON KUNDENLISTEN                                *
* AUTOR        : M.-A. BEISECKER                  DATUM: 20.07.1986 *
**************************************************************************

* INITIALISIERUNG DER VARIABLEN

STORE 0 TO AUSWAHL

* SCHLEIFE 1 ZUM WIEDERHOLTEN DURCHLAUF DES MENUES

DO WHILE AUSWAHL >= 0
```

```
* AUFBAU DES MENUES

ERASE
TEXT


                          DRUCKPROGRAMME
                          --------------

                       1. LISTE NACH NAMEN
                       2. ETIKETTEN NACH KUNDENNR


                      0. ZURUECK IN DAS VORHERIGE MENUE


                      BITTE WAEHLEN SIE:
ENDTEXT
@ 13,45 GET AUSWAHL PICTURE "9"
READ

* CASE-VERZWEIGUNG IN ABHAENGIGKEIT DER VARIABLEN AUSWAHL

DO CASE
   CASE AUSWAHL = 1
        REPORT FORM KUNDE TO PRINT
   CASE AUSWAHL = 2
        ERASE
        ACCEPT "VON KUNDENNR:" TO ANFANG
        ACCEPT "BIS KUNDENNR:" TO ENDE
        ACCEPT "WIE VIELE KOPIEN ?" TO ANZAHL
        ACCEPT "WUENSCHEN SIE EINEN PROBEDRUCK ?" TO JA

        * UMWANDLUNG VON ALPHANUMERISCH IN NUMERISCH

        STORE VAL(ANFANG) TO N:ANFANG
        STORE VAL(ENDE)   TO N:ENDE
        STORE VAL(ANZAHL) TO N:ANZAHL

        * SCHLEIFE 2 ZUR WIEDERHOLUNG DES PROBEDRUCKES

        DO WHILE JA="J" .OR. JA="j"
           SET PRINT ON
           SET CONSOLE OFF
           STORE 1 TO ZAEHLER

           * SCHLEIFE 3 ZUR AUSGABE VON 3 ETIKETTEN

           DO WHILE ZAEHLER < 4
              ? "VORNAME... NACHNAME......."
              ? "STRASSE HAUSNR."
              ?
              ? "PLZ. ORT........."
              ?
              ?
```

```
            STORE ZAEHLER + 1 TO ZAEHLER
         ENDDO

         * ENDE DER SCHLEIFE 2

         SET PRINT OFF
         SET CONSOLE ON
         ACCEPT "WUENSCHEN SIE EINEN WEITEREN PROBEAUSDRUCK ?" TO JA
      ENDDO

      * ENDE DER SCHLEIFE 2

      * INITIALISIEREN

      STORE N:ANZAHL TO ZAEHLER

      * INDEXDATEI KUNDENNR AKTIVIEREN

      SET INDEX TO KUNDENNR

      FIND &ANFANG
      IF # > 0
         SET PRINT ON
         SET CONSOLE OFF

      * SCHLEIFE 4 ZUM AUSDRUCK DER KUNDEN IM BEREICH ANFANG BIS ENDE

      DO WHILE N:ANFANG <= N:ENDE .AND. .NOT. EOF

         * SCHLEIFE 5 ZUM WIEDERHOLTEN AUSDRUCK EINES KUNDEN

         DO WHILE ZAEHLER > 0
            ? TRIM(VORNAME)+" "+NACHNAME
            ? STRASSE
            ?
            ? PLZ+" "+ORT
            ?
            ?
            ?
            STORE ZAEHLER - 1 TO ZAEHLER
         ENDDO

         * ENDE DER SCHLEIFE 5

         SKIP
         STORE KUNDENNR TO N:ANFANG
         STORE N:ANZAHL TO ZAEHLER
      ENDDO

      * ENDE DER SCHLEIFE 4

      SET PRINT OFF
      SET CONSOLE ON
   ENDIF

      * INDEXDATEI NAME AKTIVIEREN
```

```
       SET INDEX TO NAME

   CASE AUSWAHL = 0
        RETURN     * ENDE DES PROGRAMMS DRUKUN
ENDCASE

* ENDE CASE - VERZWEIGUNG

STORE 0 TO AUSWAHL
ENDDO

* ENDE DER SCHLEIFE 1
```

Kapitel 9

Kombinieren von Datenbanken am Beispiel einer Rechnungsschreibung

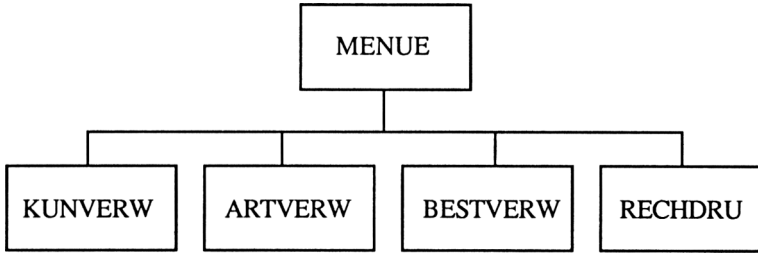
In diesem Kapitel werden folgende Themen behandelt:

Das Menüprogramm
Bestellungen erfassen mit dem Programm BESTVERW
Primäre und sekundäre Dateien
Summieren von Datenbanken
Update von Datenbanken
Rechnungen drucken mit dem Programm RECHDRU

Zu diesen Themen gehören die folgenden Befehle:

SELECT PRIMARY
SELECT SECONDARY
TOTAL
UPDATE

Das Menüprogramm zur Rechnungsschreibung



In diesem Kapitel erstellen wir ein Menüprogramm, von dem die Programme zur Artikel- und Adreßverwaltung sowie die neu zu erstellenden Programme zur Auftragseingabe und zum Rechnungsdruck aufgerufen werden. Wir erhalten so ein Programmpaket zur Fakturierung bzw. Rechnungsschreibung, mit dem ebenfalls die Kundendatei und die Artikeldatei verwaltet wird. Durch die Aufteilung der unterschiedlichen Aufgaben auf verschiedene Programme ist die Programmstruktur leicht zu verstehen.

Das Programm MENUE zeigt das Menübild der Rechnungsschreibung an und verzweigt je nach Benutzereingabe zu den einzelnen Unterprogrammen. Dieses Programm unterscheidet sich nur gering von den Menüprogrammen ARTVERW und KUNVERW, die zur Artikelverwaltung bzw. Kundenverwaltung dienen.

```
*********************************************************************
* UNTERPROGRAMM: MENUE                                              *
* FUNKTION     : MENUEPROGRAMM ZUR RECHNUNGSSCHREIBUNG              *
* AUTOR        : M.-A. BEISECKER                  DATUM: 29.07.1986 *
*********************************************************************

* EINSTELLEN DER SYSTEMPARAMETER UND INITIALISIERUNG DER VARIABLEN

SET TALK OFF
STORE 0 TO AUSWAHL

* SCHLEIFE ZUM WIEDERHOLTEN DURCHLAUF DES MENUES

DO WHILE AUSWAHL >= 0

* AUFBAU DES MENUES

ERASE
TEXT


                        RECHNUNGSSCHREIBUNG
                        -------------------

                    1. ADRESSVERWALTUNG
                    2. ARTIKELVERWALTUNG
                    3. BESTELLVERWALTUNG
                    4. RECHNUNGSDRUCK

                    0. ZURUECK IN DAS VORHERIGE MENUE


                    BITTE WAEHLEN SIE:
ENDTEXT
@ 14,45 GET AUSWAHL PICTURE "9"
READ

* CASE-VERZWEIGUNG IN ABHAENGIGKEIT DER VARIABLEN AUSWAHL
```

```
DO CASE
   CASE AUSWAHL = 1
        DO KUNVERW
   CASE AUSWAHL = 2
        DO ARTVERW
   CASE AUSWAHL = 3
        DO BESTVERW
   CASE AUSWAHL = 4
        DO RECHDRU
   CASE AUSWAHL = 0
        SET TALK ON
        RELEASE ALL
        RETURN     * ENDE DES PROGRAMMS MENUE
ENDCASE * ENDE CASE-VERZWEIGUNG
STORE 0 TO AUSWAHL
ENDDO    * ENDE DER SCHLEIFE
```

Am Anfang des Programms wird mit dem Befehl SET TALK OFF erreicht, daß die Ergebnisse der nachfolgenden Befehle nicht mehr auf dem Bildschirm angezeigt werden. Diese Systemeinstellung ist so lange aktiv, bis sie durch den Befehl SET TALK ON rückgängig gemacht wird.

Deshalb kann dieser Befehl aus den nachfolgenden Programmen ARTVERW und KUNVERW entfernt werden, falls diese Programme immer als Unterprogramme des Programms MENUE und nicht direkt aufgerufen werden. Je nach Wahl des Benutzers verzweigt das Programm im CASE-Konstrukt zu einem der Unterprogramme oder dem Programmende.

Vor diesem steht der Befehl SET TALK ON und der Befehl RELEASE ALL, der alle Speichervariablen löscht. Die Speichervariablen sollten am Ende eines Programms immer gelöscht werden – wenn man sie nicht zur Fehlersuche benötigt – da ihre Anzahl auf 64 begrenzt ist.

Bestellungen erfassen mit dem Programm BESTVERW

Das folgende Programm BESTVERW ermöglicht es, Bestellungen in die Datei BESTELL einzugeben. Dieser Datei werden später die notwendigen Daten zum Rechnungsdruck entnommen. Die Bestelldatei hat folgenden Aufbau:

```
Strukturdaten für Datei: A:BESTELL .DBF
Anzahl der Sätze: 00000
Datum der letzten Aktualisierung: 00/00/00
Primäre Datei
Feld  Name    Typ  Länge  Dez.st.
001     BRECHNR      N     004
002     BKUNDENNR    N     004
003     BARTNR       N     004
```

```
004     BARTBEZ      C    020
005     BMENGE       N    004
006     BPREIS       N    006    002
007     UMSATZ       N    006    002
008     BDATUM       N    008
009     KZ           C    001
** Gesammt **             00058
```

Die Datei BESTELL sollten Sie jetzt mit dem CREATE-Befehl anlegen. Sind Sie sich nicht mehr sicher, wie dieser Befehl angewendet wird, sehen Sie sich noch einmal das Kapitel III.1., "Erstellen einer Dateistruktur" an.

Das Feld BRECHNR enthält die Rechnungsnummer und dient als Schlüsselfeld der Datei. Dazu muß die passende Indexdatei erzeugt werden:

```
. USE BESTELL
. INDEX ON BRECHNR TO RECHNR
```

Im Feld BKUNDENNR ist die Kundennummer gespeichert. Über dieses Feld wird auf die Datei KUNDEN zugegriffen. Mit dem Feld BARTNR, das die Artikelnummer enthält, kann auf die Datei ARTIKEL zugegriffen werden. Über die Artikelnummer ist es möglich, die zugehörige Artikelbezeichnung zu suchen und diese in die Datei BESTELL zu kopieren. Dies erspart die Eingabe der Artikelbezeichnung durch den Benutzer.

Die übrigen Felder der Bestelldatei enthalten die Menge des bestellten Artikels, dessen Preis, einen eventuellen Preisnachlaß in Prozent, die Gesamtsumme, das Bestelldatum und ein Kennzeichen.

Im Programm BESTVERW wird auf die Dateien BESTELL, KUNDEN und ARTIKEL zugegriffen. Bisher haben wir immer nur mit einer Datei gearbeitet. dBASE II erlaubt aber, mit zwei Dateien zu arbeiten. Dazu gibt es verschiedene Möglichkeiten. Die einfachste Möglichkeit ist die der primären und sekundären Datenbank.

Primäre und sekundäre Dateien

Vor dem USE-Befehl geben wir mit dem Befehl SELECT PRIMARY bzw. SELECT SECONDARY an, ob die nachfolgend zu eröffnende Datenbank als primäre oder sekundäre Datenbank zu betrachten ist:

```
SELECT PRIMARY
USE KUNDEN INDEX KUNDENNR
```

bzw.

```
SELECT SECONDARY
USE BESTELL INDEX RECHNR
```

Die Datenbank KUNDEN ist jetzt als primäre und die Datenbank BESTELL als sekundäre Datei eröffnet. Der SELECT-Befehl schaltet zwischen den Datenbanken um. Mit SELECT PRIMARY aktiviert man beispielsweise die primäre Datenbank. Wir können nun nicht auf die Datensätze der sekundären Datenbank zugreifen, die im Hintergrund "wartet". Man kann immer nur auf die Datensätze der gerade aktiven Datenbank zugreifen. Betrachten wir den ersten Teil des Programms BESTVERW:

```
*************************************************************************
* UNTERPROGRAMM: BESTVERW                                               *
* FUNKTION     : EINGABE VON BESTELLUNGEN                               *
* AUTOR        : M.-A. BEISECKER                  DATUM: 27.07.1986 *
*************************************************************************

* EINSTELLEN DER SYSTEMPARAMETER UND INITIALISIERUNG DER VARIABLEN

STORE T TO SCHLEIFE

* OEFFNEN DER DATEI BESTELL MIT INDEXDATEI RECHNR ALS SEKUNDÄRE DATEI

SELECT SECONDARY
USE BESTELL INDEX RECHNR

* SCHLEIFE 1 ZUR WIEDERHOLTEN EINGABE

DO WHILE SCHLEIFE
   STORE "N" TO JA

   * OEFFNEN DER DATEI KUNDEN ALS PRIMAERE DATEI

   SELECT PRIMARY
   USE KUNDEN INDEX KUNDENNR

   * SCHLEIFE 2 ZUR WIEDERHOLTEN EINGABE DER KUNDENNUMMER

   DO WHILE JA # "J"
      ERASE
      STORE 0 TO EKUNDE
      @ 0,0 SAY "BITTE KUNDENNUMMER EINGEBEN (0=ENDE)" TO KUNDE ;
            GET EKUNDE PICTURE "9999"
      READ
      IF EKUNDE = 0
         STORE F TO SCHLEIFE
         STORE "J" TO JA
      ELSE
         STORE STR(EKUNDE,4,0) TO KUNDE
         FIND &KUNDE
         IF # > 0
           @ 1,0 SAY KUNDENNR
            @ 1,5 SAY NACHNAME
            @ 1,25 SAY VORNAME
            @ 1,45 SAY PLZ
            @ 1,49 SAY ORT
            STORE "J" TO JA
```

```
            @ 2,0 SAY "IST DIES DER GEWUENSCHTE KUNDE (J/N)?" ;
                  GET JA PICTURE "!"
            READ
         ELSE
            @ 2,0 SAY "DIESE KUNDENNUMMER GIBT ES NICHT !"
            STORE "N" TO JA
         ENDIF
      ENDIF
   ENDDO

   * ENDE DER SCHLEIFE 2
```

Nach dem Kommentar wird die Datei BESTELL als sekundäre Datei eröffnet. Die dann folgende Schleife Eins wird so lange durchlaufen, bis für die Kundennummer Eine eins eingegeben wird. Somit ist es möglich, so lange Bestellungen einzugeben, bis die logische Variable SCHLEIFE auf False bzw. Unwahr gesetzt wird. Innerhalb dieser Schleife wird die Datenbank KUNDEN als primäre Datenbank aktiviert. Dann wird innerhalb der Schleife Zwei nach der Kundennummer gefragt. Ist die eingegebene Kundennummer gleich null, wird die logische Variable SCHLEIFE mit False und die Variable JA mit J besetzt. Dies sind die Variableninhalte, die zum Abbruch der ersten und zweiten Schleife führen; das Programm wird somit beendet.

Bei einer von null verschiedenen Kundennummer wird dieser Kunde in der Datei KUNDEN gesucht, und falls er gefunden wird, angezeigt. Der Benutzer kann jetzt entscheiden, ob dies der gewünschte Kunde ist. Sollte dies nicht der richtige Kunde sein oder wird kein Kunde mit der eingegebenen Kundennummer gefunden, wird erneut nach der Kundennummer gefragt. Ist der Kunde nicht gespeichert, kann keine Bestellung aufgenommen werden. In diesem Fall müssen die Daten des Kunden mit Hilfe der Adreßverwaltung erfaßt werden. Die Adresse des Kunden wird später für das Programm RECHDRU zur Rechnungsschreibung benötigt.

In der Datei BESTELL wird nach der letzten vergebenen Rechnungsnummer gesucht und diese dann um eins erhöht als neue Rechnungsnummer ausgegeben. Nach den Grundsätzen ordnungsgemäßer Buchführung müssen alle Rechnungen, die ein Kaufmann ausstellt, durchgängig numeriert sein. Enthält die Datei BESTELL noch keinen Datensatz, so wird die Rechnungsnummer auf eins gesetzt. Diese Prozedur wird natürlich nur dann durchlaufen, wenn die eingegebene Kundennummer größer als null ist, also nicht dann, wenn man das Programm abbrechen möchte.

```
IF EKUNDE # 0
     SELECT SECONDARY
     IF .NOT. EOF

        * SUCHE NACH DER LETZTEN VERGEBENEN RECHNUNGSNUMMER
```

```
   GOTO BOTTOM
   STORE BRECHNR + 1 TO ERECHNR
ELSE

   * ENTHAELT DIE DATEI KEINEN SATZ, WIRD DIE RECHNUNGSNUMMER EINS

   STORE 1 TO ERECHNR
ENDIF
@ 4,0 SAY "NEUE RECHNUNGSNUMMER: "
@ 4,22 SAY ERECHNR
```

Um die Variablen, die zur Bildschirmein- und ausgabe verwendet werden, von den Speichervariablen zu unterscheiden, beginnen diese mit dem Buchstaben E. Die Variable ERECHNR enthält die um eins erhöhte Rechnungsnummer aus der Speichervariablen BRECHNR.

Die Kundennummer und die Rechnungsnummer müssen pro Rechnung nur einmal erfaßt werden. Ein Kunde kann mehrere Artikel bestellen, die dann jeweils in einem eigenen Datensatz stehen. In diesem Programm wurde die Anzahl der Artikel, die pro Rechnung erfaßt werden können, auf zwanzig begrenzt.

Durch die Möglichkeit, unbegrenzte Mengen an Rechnungsposten eingeben zu können, wäre das Rechnungsdruckprogramm erheblich komplexer ausgefallen und daher schlechter zu verstehen gewesen.

Bevor die dritte Schleife zur Eingabe der Rechnungsposten beginnt, wird der Zähler, der die Anzahl der Schleifendurchläufe auf maximal zwanzig begrenzen soll, auf null gesetzt. Damit auf die Datenbank ARTIKEL zugegriffen werden kann, wird diese als primäre Datenbank eröffnet. Die Datenbank KUNDEN wird dabei automatisch geschlossen.

Nun können mit Hilfe der dritten Schleife so lange Artikel erfaßt werden, bis die Variable ZAEHLER den Wert 20 enthält oder als Artikelnummer null eingegeben wird. In der Datenbank ARTIKEL wird überprüft, ob die angegebene Artikelnummer vorhanden ist. Falls der Artikel gefunden wird, kopiert das Programm die Artikelbezeichnung in die Variable EARTBEZ. Die bestellte Menge EMENGE wird von der vorhandenen Menge BESTAND im Datensatz der Artikeldatenbank abgezogen, damit die Artikeldatei auf dem aktuellen Stand ist.

Wird weder ein abweichender Preis noch eine Prozentzahl eingegeben, ersetzt das Programm den Inhalt der Variablen EPREIS durch den der Speichervariablen VK aus der Artikeldatenbank. Ansonsten bleibt der eingegebene abweichende Preis in der Variablen EPREIS, wenn keine Prozentzahl eingegeben wurde. In diesem Fall wird die Variable EPREIS durch den prozentual verminderten Inhalt der Speichervariablen VK ersetzt.

Alle für einen Rechnungsposten notwendigen Daten sind jetzt eingegeben und können als Datensatz an die Datei BESTELL angehängt werden. Dazu wird mit SELECT SECONDARY auf diese Datenbank umgeschaltet und mit dem Befehl APPEND BLANK ein Leersatz angehängt. Dieser Leersatz wird mit den Daten gefüllt, indem mit dem STORE-Befehl die Inhalte der Bildschirmvariablen den Speichervariablen zugewiesen werden.

Zuletzt wird der Zähler um eins erhöht und nach der nächsten Artikelnummer gefragt. Sollte eine Artikelnummer nicht gespeichert sein, so wird eine Fehlermeldung ausgegeben. Geben wir null als Artikelnummer ein, wird der Zähler auf 20 gesetzt, um diese Schleife abzubrechen.

```
STORE 0 TO ZAEHLER
SELECT PRIMARY
USE ARTIKEL INDEX ARTNR

* SCHLEIFE 3 ZUR WIEDERHOLTEN EINGABE DER ARTIKEL

DO WHILE ZAEHLER <= 20
   STORE 0 TO EARTNR, EDATUM, EMENGE, EPREIS, EPROZENT
   @ 6,0 SAY "ARTIKELNR. (0=ENDE ? " GET EARTNR PICTURE "9999"
   READ
   IF EARTNR > 0
    @ 7,0 SAY "BESTELLDATUM     ? " GET EDATUM PICTURE "99999999"
    @ 8,0 SAY "BESTELLMENGE     ? " GET EMENGE PICTURE "9999"
    @ 9,0 SAY "ABW. PREIS       ? " GET EPREIS PICTURE "999.99"
    @ 10,0 SAY "PROZENTE         ? " GET EPROZENT PICTURE "99"
    READ
    STORE STR(EARTNR,4,0) TO ARTIKEL
    SELECT PRIMARY
    FIND &ARTIKEL
    IF # > 0
       STORE ARTIKELBEZ TO EARTBEZ

       * DER LAGERBESTAND IN DER DATEI ARTIKEL
       * WIRD UM DIE VERKAUFTE MENGE REDUZIERT

       REPLACE BESTAND WITH BESTAND - EMENGE

       * DER VERKAUFSPREIS WIRD AUS DER ARTIKELDATEI
       * UEBERNOMMEN, WENN KEIN ABWEICHENDER PREIS ANGEGEBEN
       * WURDE. DIESER PREIS WIRD DURCH EINE EINGEGEBENE
       * PROZENTZAHL VERMINDERT.

       IF EPREIS = 0
          STORE VK TO EPREIS
       ENDIF
       IF PROZENT > 0
          STORE VK*(100-EPROZENT)/100 TO EPREIS
       ENDIF

       * DER NEUE SATZ WIRD AN DIE DATEI BESTELL ANGEFUEGT
```

```
            SELECT SECONDARY
            APPEND BLANK

            * DER LEERE SATZ WIRD MIT DEN DATEN GEFUELLT

            REPLACE BKUNDENNR WITH EKUNDE
            REPLACE BRECHNR   WITH ERECHNR
            REPLACE BARTNR    WITH EARTNR
            REPLACE BARTBEZ   WITH EARTBEZ
            REPLACE BDATUM    WITH EDATUM
            REPLACE BMENGE    WITH EMENGE
            REPLACE BPREIS    WITH EPREIS
            REPLACE UMSATZ    WITH EPREIS * EMENGE
         ELSE
            @ 12,0  SAY "DEN ARTIKEL "
            @ 12,12 SAY EARTNR
            @ 12,17 SAY "GIBT ES NICHT!"
         ENDIF
      ELSE
         STORE 20 TO ZAEHLER
      ENDIF
      STORE ZAEHLER + 1 TO ZAEHLER
     ENDDO

     * ENDE DER SCHLEIFE 3

   ENDIF
ENDDO

* ENDE DER SCHLEIFE 1
```

An dieser Stelle könnten wir alle geöffneten Dateien schließen. Damit hätten wir die Aufgabe, ein Erfassungsprogramm zu schreiben, gelöst. Um zu wissen, wieviel Umsatz ein Kunde in einem Jahr tätigt, haben wir in der Datei KUNDEN ein Feld mit dem Namen UMSATZ angelegt. Es bietet sich an dieser Stelle an, dieses Feld auf den aktuellen Stand zu bringen. Dazu muß das Feld Umsatz in der Datenbank BESTELL pro Kundennummer aufsummiert werden. Da die Datei BESTELL nicht nach Kundennummern indiziert ist, legen wir eine neue Indexdatei an.

```
INDEX ON BKUNDENNR TO KNR
TOTAL ON BKUNDENNR TO UMSATZ FIELDS BKUNDENNR, UMSATZ, KZ
```

Summieren von Datenbanken

Danach folgt der Befehl TOTAL, der es ermöglicht, Gesamtsummen von Datenbankfeldern zu erzeugen und diese in einer neuen Datei abzuspeichern. In diesem Beispiel wird in der eröffneten und nach Kundennummern indizierten Datenbank die Summe aller Umsätze pro Kundennummer gebildet, und dann werden die Felder BKUNDENNR, UMSATZ und KZ in der neuen Datei UMSATZ abgespeichert.

Damit nur die neu erfaßten Datensätze bei einer späteren Aktualisierung der Datenbank KUNDEN berücksichtigt werden, enthält die Datei BESTELL das Feld KZ. Nachdem die Datei UMSATZ erzeugt wurde, werden alle Kennzeichenfelder in der Datei BESTELL mit einem Stern gefüllt. Enthält ein Datensatz in der Datei UMSATZ einen Stern, wird er gelöscht und kann so nicht doppelt verarbeitet werden.

```
SET INDEX TO
REPLACE ALL KZ WITH "*"

* DIE NEU ANGELEGTE DATEI UMSATZ WIRD ZUM UPDATE DER DATEI KUNDEN
* VERWENDET. VORHER WERDEN ALLE SAETZE GELOESCHT, DIE SCHON EINMAL
* FRUEHER ERFASST WURDEN, DEREN MERKMAL KZ ALSO = '*' IST.

USE UMSATZ
DELETE ALL FOR KZ="*"
PACK
USE
```

Nachdem die Datei UMSATZ jetzt bereinigt ist, kann auf die Datei KUNDEN umgeschaltet und deren Aktualisierung (engl. update) erfolgen.

Update von Datenbanken

```
SELECT PRIMARY
USE KUNDEN INDEX KUNDENNR
UPDATE FROM UMSATZ ON BKUNDENNR ADD UMSATZ RANDOM
USE
```

Dazu wird der Befehl UPDATE verwendet, der es ermöglicht, Felder einer Datei durch Felder einer anderen Datei ersetzen oder deren Inhalt zu dem gegenwärtigen Inhalt addieren zu lassen. In diesem Fall wird die gerade eröffnete Datenbank KUNDEN durch die Datei UMSATZ über die Variable BKUNDENNR als Schlüssel aktualisiert. Dieser Befehl bewirkt, daß die Datei UMSATZ sequentiell gelesen und dann über das Feld BKUNDENNR der zugehörige Datensatz der Datenbank KUNDEN gesucht wird. Beide Dateien müssen dazu nach der Kundennummer sortiert oder indiziert sein, damit dieses Verfahren funktioniert.

Ist die zu aktualisierende Datenbank nach diesem Schlüssel indiziert, kann man bei dem UPDATE-Befehl den Parameter RANDOM angeben, der eine indexsequentielle Suche nach korrespondierenden Datensätzen durchführt. Ansonsten geschieht diese Suche sequentiell. In dem gefundenen Datensatz wird zu dem vorhandenen Inhalt der Variablen UMSATZ der Wert aus der Datei UMSATZ addiert. Es ist notwendig, daß die Felder, die ausgetauscht oder addiert werden sollen, in beiden Dateien den gleichen Namen tragen.

Am Schluß des Programms werden die nicht mehr benötigten Dateien UMSATZ.DBF und KNR.NDX gelöscht, die Befehlausgabe wieder eingeschaltet und das Programm abgebrochen.

```
DELETE FILE KNR.NDX
DELETE FILE UMSATZ.DBF
SET TALK ON
RETURN
```

Hier noch einmal das vollständige Programm zur Eingabe in den Rechner:

```
*************************************************************************
* UNTERPROGRAMM: BESTVERW                                               *
* FUNKTION     : EINGABE VON BESTELLUNGEN                               *
* AUTOR        : M.-A. BEISECKER                  DATUM: 27.07.1986 *
*************************************************************************

* EINSTELLEN DER SYSTEMPARAMETER UND INITIALISIERUNG DER VARIABLEN

STORE T TO SCHLEIFE

* OEFFNEN DER DATEI BESTELL MIT INDEXDATEI RECHNR ALS SEKUNDÄRE DATEI

SELECT SECONDARY
USE BESTELL INDEX RECHNR

* SCHLEIFE 1 ZUR WIEDERHOLTEN RECHNUNGSEINGABE

DO WHILE SCHLEIFE
   STORE "N" TO JA

   * OEFFNEN DER DATEI KUNDEN ALS PRIMAERE DATEI

   SELECT PRIMARY
   USE KUNDEN INDEX KUNDENNR

   * SCHLEIFE 2 ZUR WIEDERHOLTEN EINGABE DER KUNDENNUMMER

   DO WHILE JA # "J"
      ERASE
      STORE 0 TO EKUNDE
      @ 0,0 SAY "BITTE KUNDENNUMMER EINGEBEN (*=ENDE)" TO KUNDE ;
            GET EKUNDE PICTURE "9999"
      READ
      IF EKUNDE = 0
         STORE F TO SCHLEIFE
         STORE "J" TO JA
      ELSE
         STORE STR(EKUNDE,4,0) TO KUNDE
         FIND &KUNDE
         IF # > 0
           @ 1,0 SAY KUNDENNR
            @ 1,5 SAY NACHNAME
```

```
         @ 1,25 SAY VORNAME
         @ 1,45 SAY PLZ
         @ 1,49 SAY ORT
         STORE "J" TO JA
         @ 2,0 SAY "IST DIES DER GEWUENSCHTE KUNDE (J/N)?" ;
               GET JA PICTURE "!"
         READ
      ELSE
         @ 2,0 SAY "DIESE KUNDENNUMMER GIBT ES NICHT!"
         STORE "N" TO JA
      ENDIF
   ENDIF
ENDDO

* ENDE DER SCHLEIFE 2

IF EKUNDE # 0
   SELECT SECONDARY
   IF .NOT. EOF

      * SUCHE NACH DER LETZTEN VERGEBENEN RECHNUNGSNUMMER

      GOTO BOTTOM
      STORE BRECHNR + 1 TO ERECHNR
   ELSE

      * ENTHÄLT DIE DATEI KEINEN SATZ, WIRD DIE RECHNUNGSNUMMER EINS

      STORE 1 TO ERECHNR
   ENDIF
   @ 4,0 SAY "NEUE RECHNUNGSNUMMER: "
   @ 4,22 SAY ERECHNR
   STORE 0 TO ZAEHLER
   SELECT PRIMARY
   USE ARTIKEL INDEX ARTNR

   * SCHLEIFE 3 ZUR WIEDERHOLTEN EINGABE DER ARTIKEL

   DO WHILE ZAEHLER <= 20
      STORE 0 TO EARTNR, EDATUM, EMENGE, EPREIS, EPROZENT
      @ 6,0 SAY "ARTIKELNR. (0=ENDE) ? " GET EARTNR PICTURE "9999"
      READ
      IF EARTNR > 0
       @ 7,0 SAY "BESTELLDATUM    ? " GET EDATUM PICTURE "99999999"
       @ 8,0 SAY "BESTELLMENGE    ? " GET EMENGE PICTURE "9999"
       @ 9,0 SAY "ABW. PREIS      ? " GET EPREIS PICTURE "999.99"
       @ 10,0 SAY "PROZENTE        ? " GET EPROZENT PICTURE "99"
       READ
       STORE STR(EARTNR,4,0) TO ARTIKEL
       SELECT PRIMARY
       FIND &ARTIKEL
       IF # > 0
          STORE ARTIKELBEZ TO EARTBEZ

          * DER LAGERBESTAND IN DER DATEI ARTIKEL
          * WIRD UM DIE VERKAUFTE MENGE REDUZIERT
```

```
            REPLACE BESTAND WITH BESTAND - EMENGE

            * DER VERKAUFSPREIS WIRD AUS DER ARTIKELDATEI
            * ÜBERNOMMEN, WENN KEIN ABWEICHENDER PREIS ANGEGEBEN
            * WURDE. DIESER PREIS WIRD DURCH EINE EINGEGEBENE
            * PROZENTZAHL VERMINDERT.

            IF EPREIS = 0
               STORE VK TO EPREIS
            ENDIF
            IF PROZENT > 0
               STORE VK*(100-EPROZENT)/100 TO EPREIS
            ENDIF

            * DER NEUE SATZ WIRD AN DIE DATEI BESTELL ANGEFÜGT

            SELECT SECONDARY
            APPEND BLANK
            REPLACE BKUNDENNR WITH EKUNDE
            REPLACE BRECHNR   WITH ERECHNR
            REPLACE BARTNR    WITH EARTNR
            REPLACE BARTBEZ   WITH EARTBEZ
            REPLACE BDATUM    WITH EDATUM
            REPLACE BMENGE    WITH EMENGE
            REPLACE BPREIS    WITH EPREIS
            REPLACE UMSATZ    WITH EPREIS * EMENGE
         ELSE
            @ 12,0  SAY "DEN ARTIKEL "
            @ 12,12 SAY EARTNR
            @ 12,17 SAY "GIBT ES NICHT!"
         ENDIF
       ELSE
          STORE 20 TO ZAEHLER
       ENDIF
       STORE ZAEHLER + 1 TO ZAEHLER
      ENDDO

      * ENDE DER SCHLEIFE DREI

   ENDIF
ENDDO

* ENDE DER SCHLEIFE EINS

* NACHDEM ALLE EINGABEN ERFOLGT SIND, WIRD EINE INDEXDATEI NACH
* KUNDENNUMMERN (KNR) ZUR DATEI BESTELL ANGELEGT.
* DIESE DATEI WIRD BENÖTIGT, UM DIE SUMMEN ALLER UMSÄTZE PRO KUNDE
* IN DER DATEI UMSATZ ABZULEGEN.
* DAS FELD KZ DIENT ZUR KENNZEICHNUNG BEARBEITETER SÄTZE.

INDEX ON BKUNDENNR TO KNR
TOTAL ON BKUNDENNR TO UMSATZ FIELDS BKUNDENNR, UMSATZ, KZ
SET INDEX TO
REPLACE ALL KZ WITH "*"

* DIE NEU ANGELEGTE DATEI UMSATZ WIRD ZUM UPDATE DER DATEI KUNDEN
```

```
* VERWENDET. VORHER WERDEN ALLE SÄTZE GELÖSCHT, DIE SCHON EINMAL
* FRÜHER ERFASST WURDEN, DEREN MERKMAL KZ ALSO = '*' IST.

USE UMSATZ
DELETE ALL FOR KZ="*"
PACK
USE
SELECT PRIMARY
USE KUNDEN INDEX KUNDENNR
UPDATE FROM UMSATZ ON BKUNDENNR ADD UMSATZ RANDOM
USE
DELETE FILE KNR.NDX
DELETE FILE UMSATZ.DBF
SET TALK ON
RETURN
```

Rechnungen drucken mit dem Programm RECHDRU

Dieses Programm dient dazu, aus den Datensätzen der Datei BESTELL eine Rechnung zu drucken. Dabei kann der Benutzer wählen, von welcher Rechnungsnummer bis zu welcher Rechnungsnummer der Ausdruck erfolgen soll. Das Rechnungsdatum wird aus der Systemvariablen DATE() übernommen.

```
STORE DATE() TO DATUM
```

Daher muß das Tagesdatum korrekt eingegeben werden bevor dieses Programm gestartet wird. Da die Kundenadresse über die Kundennummer aus der Datei KUNDEN geholt wird, werden auch in diesem Programm zwei Dateien eröffnet.

```
* ÖFFNEN DER DATEI KUNDEN MIT INDEXDATEI KUNDENNNR ALS PRIMÄRE DATEI

SELECT PRIMARY
USE KUNDEN INDEX KUNDENNR

* ÖFFNEN DER DATEI BESTELL MIT INDEXDATEI RECHNR ALS SEKUNDÄRE DATEI

SELECT SECONDARY
USE BESTELL INDEX RECHNR
```

Dann erfolgt die eingangs erwähnte Abfrage nach den gewünschten Rechnungsnummern mit dem ACCEPT-Befehl.

```
* BILDSCHIRMAUFBAU, ABFRAGE DER GEWÜNSCHTEN AUFTRAGSNUMMERN

ERASE
@ 0,24 SAY "R E C H N U N G S D R U C K"
@ 1,24 SAY "--------------------------"
?
```

```
?
?
?
ACCEPT "VON AUFTRAGSNR." TO ANFANG
ACCEPT "BIS AUFTRAGSNR." TO ENDE

* UMWANDLUNG DER ALPHANUMERISCHEN IN NUMERISCHE VARIABLEN

STORE VAL(ANFANG) TO N:ANFANG
STORE VAL(ENDE)   TO N:ENDE
```

Nach der Eingabe wird die erste Rechnungsnummer mit dem FIND-Befehl gesucht. Ist die gewünschte Rechnungsnummer vorhanden, kann mit dem Ausdruck begonnen werden. Die Schleife Zwei wird so lange durchlaufen, bis eine Rechnungsnummer größer als der angegebene Bereich ist. Sie dient zur Ausgabe der Rechnungsnummer und der Kundenadresse auf dem Drucker. Diese Angaben werden pro Rechnung nur einmal benötigt. Der Ausdruck der Rechnungsposten geschieht mit Hilfe einer weiteren Schleife.

```
* SUCHE NACH DER ERSTEN AUFTRAGSNUMMER

FIND &ANFANG
IF # > 0

   * EINSCHALTEN DES DRUCKERS, UMSCHALTEN AUF DRUCKERFORMATSTEUERUNG

   SET PRINT ON
   SET FORMAT TO PRINT
   SET CONSOLE OFF

   * SCHLEIFE 1 ZUR AUSGABE DER AUFTRAGSNUMMERN

   DO WHILE N:ANFANG <= N:ENDE .AND. .NOT. EOF

      * SUCHE NACH DEM RECHNUNGSEMPFÄNGER IN DER KUNDENDATEI

      STORE STR(BKUNDENNR,4,0) TO KUNDE
      SELECT PRIMARY
      FIND &KUNDE

      * AUSGABE DER KUNDENADRESSE

      @ 11,5   SAY TRIM(VORNAME)+" "+NACHNAME
      @ $+1,5  SAY STRASSE
      @ $+2,5  SAY PLZ+" "+ORT
      @ $+8,5  SAY "Ihre Kundennummer: "
      @ $,$    SAY KUNDENNR
      @ $,$+5  SAY "Ihre Bestellung vom: "
      @ $,$    SAY $(STR(BDATUM,8,0),1,2)
      @ $,$    SAY "."
      @ $,$    SAY $(STR(BDATUM,8,0),3,2)
      @ $,$    SAY "."
      @ $,$    SAY $(STR(BDATUM,8,0),5,4)
      @ $+1,5  SAY "Rechnungsnummer  : "
      @ $,$    SAY BRECHNR
      @ $,$+5  SAY "Rechnungsdatum  vom: "
```

```
@ $,$    SAY $(DATUM,1,2)
@ $,$    SAY "."
@ $,$    SAY $(DATUM,4,2)
@ $,$    SAY ".19"
@ $,$    SAY $(DATUM,7,2)
@ $+3,5  SAY "ARTNR. ARTIKELBEZEICHNUNG  MENGE    PREIS         GESAMT"
@ $+1,5  SAY "------------------------------------------------------------"
SELECT SECONDARY
STORE 1 TO ZAEHLER
STORE BRECHNR TO RECHNR
```

Während die X,Y-Koordinaten des @-Befehls bei der Bildschirmausgabe nur Werte von 0 – 23 bzw. 0 – 79 annehmen dürfen, können beide Koordinaten bei der Druckausgabe im Bereich 0 – 254 liegen.

In diesem Programm werden vor allem relative Koordinaten verwendet. Dabei steht das $-Zeichen für die letzte Druckposition in X- bzw. Y-Richtung.

In der Schleife Zwei werden die Rechnungsposten untereinander ausgegeben. Der Bruttopreis aus der Datei BESTELL wird dabei in den Nettopreis umgerechnet und auf zwei Stellen hinter dem Komma gerundet.

Das Feld UMSATZ wird ebenso behandelt. Dies dient dazu die Mehrwertsteuer extra auszuweisen.

```
* SCHLEIFE 2 ZUR MAXIMALEN AUSGABE VON 20 RECHNUNGSPOSTEN

DO WHILE BRECHNR=RECHNR .AND. ZAEHLER < 21 .AND. .NOT. EOF
   @ $+1,6 SAY BARTNR
   @ $,$+2 SAY BARTBEZ
   @ $,$+1 SAY BMENGE
   @ $,$+4 SAY INT(BPREIS/1.14*100+.5)/100.00 USING "###.##"
   @ $,$+1 SAY "DM"
   @ $,$+5 SAY INT(UMSATZ/1.14*100+.5)/100.00 USING "###.##"
   @ $,$+1 SAY "DM"
   STORE ZAEHLER+1 TO ZAEHLER
   SKIP
   STORE # TO SATZNR
ENDDO
```

Nachdem alle Rechnungsposten gedruckt sind, wird die Gesamtsumme und die Mehrwertsteuer ausgegeben.

```
   * BERECHNUNG DER RECHNUNGSSUMME

   SUM UMSATZ FOR BRECHNR=RECHNR TO SUMME
   @ $+1,54 SAY "----------"
   STORE SUMME/1.14 TO NETTO
   @ $+1,54 SAY NETTO USING "####.##"
   @ $,$+1  SAY "DM"
   @ $+1,45 SAY "14% MwSt"
   @ $,$+1  SAY INT((SUMME-NETTO)*100+.5)/100.00 USING "####.##"
```

```
      @ $,$+1  SAY "DM"
      @ $+1,54 SAY "==========="
      @ $+1,54 SAY SUMME USING "####.##"
      @ $,$+1  SAY "DM"
      @ $+2,5  SAY "Zahlbar sofort netto Kasse."
      @ $+1,0  SAY " "
      GOTO SATZNR
      STORE BRECHNR TO N:ANFANG
   ENDDO

   * ENDE DER SCHLEIFE 2
```

Zuletzt wird der Drucker wieder abgeschaltet, die Datenbanken geschlossen, die verwendeten Variablen gelöscht und zum Hauptprogramm zurückgekehrt.

```
      * ABSCHALTEN DES DRUCKERS, UMSCHALTEN AUF BILDSCHIRMFORMAT
      SET PRINT OFF
      SET FORMAT TO SCREEN
      SET CONSOLE ON
   ENDIF
   * SCHLIESSEN DER DATENBANKEN
   SELECT PRIMARY
   USE
   SELECT SECONDARY
   USE
   * LÖSCHEN DER VERWENDETEN VARIABLEN UND RÜCKKEHR ZUM HAUPTMENUE
   RELEASE ALL
   RETURN
```

Haben wir das Programm richtig eingegeben, erhalten wir beispielsweise einen solchen Rechnungsdruck:

```
Otto Käufer
Akazienallee 11

5000 Köln 32


Ihre Kundennummer:      1        Ihre Bestellung vom: 12.11.1985
Rechnungsnummer  :      1        Rechnungsdatum  vom: 02.09.1986

ARTNR. ARTIKELBEZEICHNUNG  MENGE     PREIS          GESAMT
-------------------------------------------------------------
    1  Joystick de Luxe        1     42.98 DM       42.98 DM
    6  5 1/4 Zoll Disketten   10      5.26 DM       52.63 DM
    4  Schutzhuelle            1     17.46 DM       17.46 DM
    5  Druckerkabel            1     43.77 DM       43.77 DM
                                                   ----------
                                                   156.84 DM
                                     14% MwSt       21.96 DM
                                                   ==========
                                                   178.80 DM

Zahlbar sofort netto Kasse.
```

Das Programm läßt sich leicht dahingehend erweitern, daß man den eigenen Firmennamen und die Kontoverbindung mit ausdrucken läßt. Man kann auch die Zahlungsbedingung oder das Format der Rechnung ändern.

Hier ist das komplette Programm in zusammenhängender Form:

```
**********************************************************************
* UNTERPROGRAMM: RECHDRU                                             *
* FUNKTION     : AUSDRUCKEN VON RECHNUNGEN                           *
* AUTOR        : M.-A. BEISECKER                DATUM: 28.07.1986 *
**********************************************************************

* EINSTELLEN DER SYSTEMPARAMETER UND INITIALISIERUNG DER VARIABLEN

SET TALK OFF
STORE DATE() TO DATUM

* ÖFFNEN DER DATEI KUNDEN MIT INDEXDATEI KUNDENNNR ALS PRIMÄRE DATEI

SELECT PRIMARY
USE KUNDEN INDEX KUNDENNR

* ÖFFNEN DER DATEI BESTELL MIT INDEXDATEI RECHNR ALS SEKUNDÄRE DATEI

SELECT SECONDARY
USE BESTELL INDEX RECHNR

* BILDSCHIRMAUFBAU, ABFRAGE DER GEWÜNSCHTEN AUFTRAGSNUMMERN

ERASE
@ 0,24 SAY "R E C H N U N G S D R U C K"
@ 1,24 SAY "---------------------------"
?
?
?
?
ACCEPT "VON AUFTRAGSNR." TO ANFANG
ACCEPT "BIS AUFTRAGSNR." TO ENDE

* UMWANDLUNG DER ALPHANUMERISCHEN IN NUMERISCHE VARIABLEN

STORE VAL(ANFANG) TO N:ANFANG
STORE VAL(ENDE)   TO N:ENDE

* SUCHE NACH DER ERSTEN AUFTRAGSNUMMER
FIND &ANFANG
IF # > 0

   * EINSCHALTEN DES DRUCKERS, UMSCHALTEN AUF DRUCKERFORMATSTEUERUNG

   SET PRINT ON
   SET FORMAT TO PRINT
   SET CONSOLE OFF

   * SCHLEIFE 1 ZUR AUSGABE DER AUFTRAGSNUMMERN
   DO WHILE N:ANFANG <= N:ENDE .AND. .NOT. EOF

      * SUCHE NACH DEM RECHNUNGSEMPFÄNGER IN DER KUNDENDATEI
```

```
STORE STR(BKUNDENNR,4,0) TO KUNDE
SELECT PRIMARY
FIND &KUNDE

* AUSGABE DER KUNDENADRESSE

@ 11,5   SAY TRIM(VORNAME)+" "+NACHNAME
@ $+1,5  SAY STRASSE
@ $+2,5  SAY PLZ+" "+ORT
@ $+8,5  SAY "Ihre Kundennummer: "
@ $,$    SAY KUNDENNR
@ $,$+5  SAY "Ihre Bestellung vom: "
@ $,$    SAY $(STR(BDATUM,8,0),1,2)
@ $,$    SAY "."
@ $,$    SAY $(STR(BDATUM,8,0),3,2)
@ $,$    SAY "."
@ $,$    SAY $(STR(BDATUM,8,0),5,4)
@ $+1,5  SAY "Rechnungsnummer  : "
@ $,$    SAY BRECHNR
@ $,$+5  SAY "Rechnungsdatum  vom: "
@ $,$    SAY $(DATUM,1,2)
@ $,$    SAY "."
@ $,$    SAY $(DATUM,4,2)
@ $,$    SAY ".19"
@ $,$    SAY $(DATUM,7,2)
@ $+3,5  SAY "ARTNR. ARTIKELBEZEICHNUNG  MENGE    PREIS         GESAMT"
@ $+1,5  SAY "------------------------------------------------------------"
SELECT SECONDARY
STORE 1 TO ZAEHLER
STORE BRECHNR TO RECHNR

* SCHLEIFE 2 ZUR MAXIMALEN AUSGABE VON 20 RECHNUNGSPOSTEN

DO WHILE BRECHNR=RECHNR .AND. ZAEHLER < 21 .AND. .NOT. EOF
   @ $+1,6 SAY BARTNR
   @ $,$+2 SAY BARTBEZ
   @ $,$+1 SAY BMENGE
   @ $,$+4 SAY INT(BPREIS/1.14*100+.5)/100.00 USING "###.##"
   @ $,$+1 SAY "DM"
   @ $,$+5 SAY INT(UMSATZ/1.14*100+.5)/100.00 USING "###.##"
   @ $,$+1 SAY "DM"
   STORE ZAEHLER+1 TO ZAEHLER
   SKIP
   STORE # TO SATZNR
ENDDO

* BERECHNUNG DER RECHNUNGSSUMME
SUM UMSATZ FOR BRECHNR=RECHNR TO SUMME
@ $+1,54 SAY "----------"
STORE SUMME/1.14 TO NETTO
@ $+1,54 SAY NETTO USING "####.##"
@ $,$+1  SAY "DM"      @ $+1,45 SAY "14% MwSt"
@ $,$+1  SAY INT((SUMME-NETTO)*100+.5)/100.00 USING "####.##"
@ $,$+1  SAY "DM"
@ $+1,54 SAY "=========="
@ $+1,54 SAY SUMME USING "####.##"
@ $,$+1  SAY "DM"
@ $+2,5  SAY "Zahlbar sofort netto Kasse."
@ $+1,0  SAY " "
GOTO SATZNR
STORE BRECHNR TO N:ANFANG
```

```
   ENDDO

   * ENDE DER SCHLEIFE 2
   * AUSSCHALTEN DES DRUCKERS, UMSCHALTEN AUF BILDSCHIRMFORMAT

   SET PRINT OFF
   SET FORMAT TO SCREEN
   SET CONSOLE ON
ENDIF

* SCHLIESSEN DER DATENBANKEN
SELECT PRIMARY
USE
SELECT SECONDARY
USE

* LÖSCHEN DER VERWENDETEN VARIABLEN UND RÜCKKEHR ZUM HAUPTMENUE
RELEASE ALL
RETURN
```

Kapitel 10

Tips und Tricks

In diesem Kapitel werden folgende Themen behandelt:

Schneller arbeiten mit dBASE II
Datenaustausch mit anderen Programmen
Makros
Grafik
Maschinensprache

Zu diesen Themen gehören die folgenden Befehle:

CALL
LOAD
PEEK
POKE
SET CALL TO

Schneller arbeiten mit dBASE II

Bei der Eingabe der recht langen dBASE-II-Befehle wünscht man sich eine Möglichkeit, diese Befehle abzukürzen. Die meisten Befehlsworte müssen nicht voll ausgeschrieben werden, sondern es genügt, die ersten vier Buchstaben zu schreiben. Im Programm werden durch die gekürzten Befehle Speicherplatz und bei der Eingabe Zeit gespart. Der häufig benötigte Befehl MODIFY COMMAND besteht zum Beispiel aus den Befehlsworten MODIFY und COMMAND, für die jeweils die ersten vier Buchstaben ausreichend sind:

```
MODI COMM <RETURN>
```

Hier ist eine Übersicht der Abkürzungen gebräuchlicher Befehle:

ACCEPT	ACCE
APPEND	APPE
CONTINUE	CONT

DELETE	DELE
DISPLAY	DISP
DISPLAY MEMORY	DISP MEMO
DISPLAY STRUCTURE	DISP STRU
LOCATE	LOCA
MODIFY COMMAND	MODI COMM
MODIFY STRUCTURE	MODI STRU
SELECT PRIMARY	SELE PRIM
SELECT SECONDARY	SELE SECO

Jedes Befehlswort kann auf diese Weise gekürzt werden, vorausgesetzt es ist länger als vier Zeichen.

Hat man mit DBDIN.SUB den deutschen Zeichensatz und die deutsche Tastaturbelegung geladen, so kann man mit einigen Tastenkombinationen Befehle erzeugen:

Tastenkombination	Befehl
<SHIFT> <.>	CREATE
<SHIFT> <ENTER>	USE
<SHIFT> <1>	APPEND <RETURN>
<SHIFT> <2>	EDIT
<SHIFT> <3>	RENAME
<SHIFT> <4>	DELETE FILE
<SHIFT> <5>	HELP <RETURN>
<SHIFT> <6>	QUIT
<SHIFT> <7>	ERASE <RETURN>
<SHIFT> <8>	LIST <RETURN>
<SHIFT> <9>	LIST FILES ON A LIKE *.* <RETURN>

Viel Zeit kann man mit geschickter Programmierung einsparen. Die indizierte Dateiverwaltung und die zugehörigen Befehle sollte soweit möglich, den sequentiell funktionierenden Befehlen vorgezogen werden.

Ein Befehl wie LOCATE, der eine Datei Datensatz für Datensatz durchsucht und mit der gesuchten Zeichenkette vergleicht, benötigt in einer Datei mit 1000 Datensätzen im Durchschnitt 501 Dateizugriffe. Der Befehl FIND erledigt diese Aufgabe mit maximal 6 Dateizugriffen, also im Durchschnitt mindestens 100mal so schnell wie der LOCATE-Befehl. Daher sollte anstelle des LOCATE-Befehls der FIND-Befehl verwendet werden.

Auch die Befehle DISPLAY und LIST arbeiten sequentiell. Wenn man alle Datensätze anzeigen möchte, ist dies nicht von Bedeutung. Möchte man aller-

dings nur eine bestimmte Gruppe von Datensätzen angezeigt haben, so sollte man sich überlegen, wie dies mit einer Indexdatei zu beschleunigen ist.

Sucht man beispielsweise alle Kunden deren Umsatz gleich null ist, so sollte man die Datei KUNDEN zunächst nach dem Feld Umsatz indizieren. Die niedrigsten Umsätze stehen jetzt zu Beginn der Datei und werden mit dem LIST FOR-Befehl sofort gefunden. Somit kann jetzt direkt bei dem ersten Datensatz entschieden werden, ob überhaupt ein Datensatz in der Datei existiert, der die Bedingung UMSATZ=0 erfüllt.

Befinden sich in der Datei KUNDEN 1000 Datensätze, von denen 40 die Bedingung erfüllen, so werden maximal sechs Dateizugriffe für den FIND-Befehl und 41 Dateizugriffe für den LIST-Befehl, also insgesamt 47 Dateizugriffe anstelle von 1000 Dateizugriffen benötigt.

Diese Technik läßt sich ebenfalls auf die sequentiell arbeitenden Befehle REPORT, COUNT, SUM, TOTAL, COPY und REPLACE anwenden.

Einen echten Zeitvorteil bringt dies aber nur bei größeren Dateien. Wenn höchstens 200 Datensätze in einer Datei gespeichert sind, lohnt sich die Indizierung nicht.

Datenaustausch mit anderen Programmen

Die Daten, die wir mit dBASE II verwalten, können an andere Programme übergeben werden. Ebenso kann dBASE II Daten von anderen Programmen übernehmen. Dies geht zusammen mit Standardprogrammen wie WordStar, Multiplan und anderen, aber auch mit BASIC-Programmen.

Möchten wir dBASE-II-Listen an WordStar übergeben, schreiben wir diese mit dem SET ALTERNATE TO-Befehl in eine Textdatei. Dies ist im Kapitel "Der Listengenerator REPORT" gezeigt. Eine solche Datei kann dann mit WordStar editiert werden.

Für den Austausch von Daten mit anderen Programmen verwendet man eine Datei im Standardformat.

Eine solche Datei enthält am Ende jedes Datensatzes die Steuerzeichen CR (Carriage Return = Wagenrücklauf) und LF (Line Feed = Zeilenvorschub). Andere Steuerzeichen sind nicht erlaubt. Die Datei wird an ihrem Namenszusatz .SDF (StandarDFormat) von einer normalen dBASE-II-Datendatei unterschieden, die auf .DBF endet.

Zur Umwandlung einer DBF-Datei in eine SDF-Datei verwendet man den COPY-Befehl zusammen mit dem Parameter SDF. Die Datei KUNDEN wandelt man zum Beispiel so um:

```
. USE KUNDEN
. COPY TO KUNDEN.SDF SDF
```

Die Datei KUNDEN.SDF kann nun von einem Standardprogramm wie Multiplan in das dort benötigte Format umgewandelt und weiter bearbeitet werden.

Die Dateistruktur der Datei KUNDEN.DBF bestimmt, wie die SDF-Datei aufgebaut ist. In der Datei KUNDEN.SDF sind alle Felder eines Datensatzes ohne ein Trennungszeichen aneinandergefügt. Am Ende des Datensatzes stehen die Steuerzeichen CR und LF. Daher ist jeder Satz in der Datei KUNDEN.SDF um zwei Zeichen länger als die Satzlänge der Dateistruktur KUNDEN.DBF.

Auf diese Datei kann man mit einem BASIC-Programm zugreifen. Mit BASIC können Dateien sequentiell oder random verwaltet werden. Die Datei KUNDEN.SDF sollte random eröffnet werden, da die einzelnen Felder nicht durch Komma oder Anführungszeichen voneinander getrennt sind.

Möchte man die Datei von BASIC aus sequentiell lesen, so muß diese mit dem Parameter DELIMITED erstellt werden:

```
. USE KUNDEN
. COPY TO KUNDEN.SDF DELIMITED
```

Wenn man den Parameter DELIMITED verwendet, muß der Parameter SDF nicht angeben werden. Die einzelnen Felder in der SDF-Datei sind jetzt durch Komma getrennt. Alphanumerische Felder sind in Hochkomma eingeschlossen. Ein BASIC-Programm kann auf diese Datei sequentiell zugreifen.

Sollen die alphanumerischen Felder in Anführungszeichen anstelle von Hochkomma eingeschlossen werden, muß dies hinter dem Parameter DELIMITED angegeben werden:

```
. USE KUNDEN
. COPY TO KUNDEN.SDF DELIMITED WITH "
```

Es lassen sich auch Kommas zur Trennung verwenden, ohne daß die alphanumerischen Felder in Hochkomma oder Anführungszeichen eingeschlossen sind. Zur entsprechenden Umwandlung der Datei KUNDEN wird ein Komma hinter DELIMITED WITH angegeben:

```
. USE KUNDEN
. COPY TO KUNDEN.SDF DELIMITED WITH ,
```

Eine solche SDF-Datei kann als Datendatei für das mischende Drucken mit MAILMERGE dienen. Dies ist sehr praktisch, wenn man Serienbriefe drukken möchte und dazu die Daten einer Adressendatei aus dBASE II benötigt. Praktischerweise kopiert man sich nur die Datensätze in die SDF-Datei, die man für MAILMERGE benötigt. Geschäftsleute schreiben beispielsweise häufig Kunden an, die seit längerer Zeit nichts mehr bestellt haben. Diese Gruppe kann mit dem COPY-Befehl aus der Datei KUNDEN.DBF selektiert werden:

```
. USE KUNDEN INDEX UMSATZ
. COPY TO KUNDEN.SDF DELIMITED WITH , FOR UMSATZ=0
```

Die erzeugte SDF-Datei kann man sich mit MODIFY COMMAND ansehen:

```
. MODIFY COMMAND KUNDEN.SDF
```

Man sieht dann beispielsweise folgende Sätze:

```
1,Käufer,Otto,Akazienallee 11,5000,Köln 32,,0.00
5,Hütter,Angelika,Sandfuhrstr.
34,4200,Oberhausen,0208-56129,0.00
7,Ende,Werner,,5428,Endlichhofen,07244-568,0.00
```

Durch MODIFY COMMAND hat man eine Kontrolle, ob die Sätze so umgewandelt wurden, wie man sie benötigt. Es ist auch möglich, die Sätze mit MODIFY COMMAND zu ändern.

Die SDF-Dateien können von Standardprogrammen nicht nur gelesen, sondern meistens auch erzeugt werden. Von dBASE II können SDF-Dateien in das dBASE-II-Dateiformat überführt werden. Dies gilt auch für BASIC-Dateien, wenn deren Datensätze im Standardformat aufgebaut sind. Zur Übernahme von SDF-Dateien muß zunächst eine passende Dateistruktur existieren, in die die SDF-Datensätze eingelesen werden können. Die Datensätze liest man dann mit dem APPEND-Befehl ein, der um den Parameter SDF erweitert wird:

```
. USE KUNDEN
. APPEND FROM KUNDEN.SDF SDF
```

Ein Dateiname einer Datei, die übernommen werden soll, muß nicht auf .SDF enden. Es kann auch ein frei wählbarer Name wie BASIC.DAT oder DATEN.TXT verwendet werden. Wichtig ist nur, daß dieser Name mit Namenszusatz hinter APPEND FROM angegeben wird.

Die Möglichkeit, dBASE II mit anderen Programmen zu verbinden, ermöglicht unzählige Anwendungen. In Verbindung mit den SUBMIT-Dateien in CP/M können diese Verbindungen für den Anwender unbemerkt erfolgen.

Es darf allerdings nicht unerwähnt bleiben, daß die Speicherkapazität der Schneider-Diskettenlaufwerke diese Möglichkeiten stark einschränkt. Eine SDF-Datei benötigt fast ebensoviel Speicherplatz wie die zugehörige DBF-Datei. Selbst mit zwei Diskettenlaufwerken kann man nicht gleichzeitig dBASE II, CP/M, Multiplan und alle benötigten Dateien auf zwei Disketten unterbringen. Trotz dieser Einschränkung ist der Datenaustausch zwischen Programmen eine nützliche Sache.

Makros

Makros können die Programmierung mit dBASE II erheblich erleichtern. Ein übermäßiger Einsatz von Makros kann aber auch dazu führen, daß Programme unübersichtlich werden und man nach kurzer Zeit das eigene Programm nicht mehr versteht.

Bisher haben wir die Makros nur zusammen mit dem FIND-Befehl eingesetzt:

```
INPUT "Zahl" TO WERT
FIND &WERT
```

Da der FIND-Befehl nur mit Zeichenketten zusammen funktioniert, könnte man ohne ein Makro nicht direkt nach einer Zahl suchen. Durch den Makrozusatz & wird der Inhalt der numerischen Variablen WERT als Zeichenkette und nicht als Zahl interpretiert.

Diese Anwendung eines Makros ist recht unkompliziert. Schwieriger wird es, wenn der Inhalt eines Makros als Befehl interpretiert wird. Dazu speichert man einen Befehl als Zeichenfolge in einer Variablen ab und ruft diese dann als Makro auf:

```
. STORE 'APPEND' TO BEFEHL
. &BEFEHL
```

Es lassen sich auch mehrere Makros zu einem Befehl zusammensetzen:

```
. USE KUNDEN
. STORE 'APPEND' TO BEFEHL
. STORE 'BLANK'  TO PARAMETER1
. STORE 'FROM'   TO PARAMETER2
. STORE 'SDF'    TO PARAMETER3
. STORE 'KUNDEN.SDF' TO DATEI
. &BEFEHL 4
. &BEFEHL &PARAMETER1
. &BEFEHL &PARAMETER2 &DATEI &PARAMETER3
```

Speichert man die Makrovariablen wiederum als Zeichenkette ab, kann man Makros ineinander verschachteln:

```
. USE KUNDEN
. STORE 'APPEND' TO BEFEHL
. STORE 'FROM'   TO PARAMETER1
. STORE 'SDF'    TO PARAMETER2
. STORE 'KUNDEN.SDF' TO DATEI
. STORE '&BEFEHL &PARAMETER1 &DATEI &PARAMETER2' TO MAKRO
. &MAKRO
```

Sieht man sich mit DISPLAY MEMORY die Variablen an, stellt man fest, daß nicht der Makroname, sondern der Inhalt des Makros in der Variablen MAKRO abgespeichert wird:

```
. DISPLAY MEMORY
BEFEHL       (C)  APPEND
PARAMETER1   (C)  FROM
PARAMETER2   (C)  SDF
DATEI        (C)  KUNDEN.SDF
MAKRO        (C)  APPEND FROM KUNDEN.SDF SDF
```

Mit Makros lassen sich auch Variablennamen zusammensetzen:

```
. STORE 'Schneider' TO VARIABLE1
. STORE '-'         TO VARIABLE2
. STORE 'Computer'  TO VARIABLE3
. STORE '1' TO M1
. STORE '2' TO M2
. STORE '3' TO M3
. ? VARIABLE&M1
. ? VARIABLE&M2
. ? VARIABLE&M3
. ? VARIABLE&M1+VARIABLE&M2+VARIABLE&M3
```

Variablennamen lassen sich nur aus Makros zusammensetzen, deren Inhalt eine Zeichenkette ist. Ein Variablenname kann auch aus mehreren Makros zusammengesetzt werden:

```
STORE 'Schneider' TO VARIABLE1
STORE 'VAR'       TO M1
STORE 'IAB'       TO M2
STORE 'LE'        TO M3
STORE '1'         TO M4
? &M1&M2&M3&M4&M5
```

An Beispielen haben wir jetzt die wesentlichen Einsatzmöglichkeiten von Makros kennengelernt. Um diese Funktion sinnvoll einsetzen zu können, muß man schon über etwas Programmiererfahrung verfügen. Speziell bei der Programmierung von Unterprogrammen leisten sie gute Dienste.

Grafik

Die Möglichkeiten von dBASE II sind nicht auf die Verwaltung von Dateien und das Rechnen mit Zahlen beschränkt. Der gesamte Zeichensatz des Schneider-Computers kann für die Gestaltung des Bildschirms verwendet werden. Den vollständigen Zeichensatz finden Sie im Anhang.

Im ASCII-Zeichensatz lassen sich 256 Zeichen darstellen, die von 0 bis 255 durchnumeriert sind. Die Zeichen ab der Nummer 129 sind beim Schneider-Computer Grafikzeichen, mit denen man beispielsweise einen Rahmen auf den Bildschirm zeichnen kann.

Hier ist ein Programm, das alle Grafikzeichen hintereinander auf dem Bildschirm anzeigt:

```
********************
* PROGRAMM: GRAFIK *
********************

ERASE
SET TALK OFF
? "Der Zeichensatz"
?
STORE 2 TO ZEILEN
STORE 32 TO ASCII
DO WHILE ASCII < 256
   DO WHILE ZEILEN < 24 .AND. ASCII < 256
      ? "ASCII:",ASCII,CHR(ASCII)
      STORE ASCII + 1 TO ASCII
      STORE ZEILEN + 1 TO ZEILEN
   ENDDO
   WAIT
   STORE 1 TO ZEILEN
ENDDO
SET TALK ON
```

In diesem kleinen Programm kommen zwei neue Befehle vor:

Der Befehl CHR(ASCII) erzeugt ein ASCII-Zeichen. Da die Variable ASCII alle Werte von 32 bis 255 während des Programmdurchlaufs enthält, werden alle ASCII-Zeichen angezeigt, die diesen Werten entsprechen. Die Zeichen mit ASCII-Werten unter 32 sind Steuerzeichen und ergeben keine Bildschirmdarstellung. Interessant sind bei diesen Zeichen das LF (ASCII=10), CR (ASCII=13) sowie BEL (ASCII=07). Die Zeichen LF und CR wurden schon besprochen. Das Zeichen BEL erzeugt einen Piepton:

```
. ? CHR(7)
```

Der Befehl WAIT hält das Programm so lange an, bis eine Taste gedrückt wird. Er dient dazu ,die Ausgabe der ASCII-Zeichen nach jeweils einer Seite abzubrechen.

Wir versuchen jetzt mit den Grafikzeichen einen kleinen Rahmen aufzubauen. Die Ränder des Rahmens sollen aus doppelten Strichen bestehen. Dazu verwenden wir die Zeichen 134 (obere, linke Ecke), 140 (obere, rechte Ecke), 131 (untere, linke Ecke), 137 (untere, rechte Ecke), 138 (waagerechter Strich) sowie 133 (senkrechter Strich):

```
********************
* PROGRAMM: RAHMEN *
********************

ERASE
SET TALK OFF
@ 5,20 SAY CHR(134)
STORE 21 TO X
DO WHILE X < 60
   @ 5,X SAY CHR(138)
   STORE 1 + X TO X
ENDDO
@ 5,60 SAY CHR(140)
STORE 6 TO Y
DO WHILE Y < 20
   @ Y,20 SAY CHR(133)
   @ Y,60 SAY CHR(133)
   STORE 1 + Y TO Y
ENDDO
@ 20,20 SAY CHR(131)
STORE 21 TO X
DO WHILE X < 60
   @ 20,X SAY CHR(138)
   STORE 1 + X TO X
ENDDO
@ 20,60 SAY CHR(137)
@ 10,38 SAY "GRAFIK"
@ 12,39 SAY "mit"
@ 14,37 SAY "dBASE II"
```

Geben Sie das Programm mit MODIFY COMMAND ein, und lassen Sie sich überraschen, was auf dem Bildschirm erscheint.

Die Maschinensprache

Aus dBASE-II-Programmen heraus können Programme in Maschinensprache aufgerufen werden. Dazu muß die Startadresse des Programms bekannt sein. Diese Adresse gibt man hinter dem SET CALL TO Befehl dezimal an:

```
SET CALL TO 00000
```

Danach kann das Programm durch den Befehl CALL aufgerufen werden. Wenn man eine Variable übergeben will, kann der Name der Variablen hinter CALL angegeben werden:

```
CALL WERT
```

Enthält die Variable eine Zeichenkette, so steht die Adresse des ersten Bytes im HL-Register. Die Länge der Zeichenkette darf nicht verändert werden. Nach dem Assemblerbefehl RET erfolgt die Rückkehr in das dBASE-II-Programm. Mit dem LOAD-Befehl kann eine hexadezimale Datei in den Hauptspeicher geladen werden:

```
LOAD DATEN
```

Der Befehl PEEK zeigt die dezimal angegebene Speicherstelle an. Im Bereich von 4096 bis 31232 arbeitet die Funktion nicht einwandfrei. Die angezeigten Ergebnisse sind nicht richtig. Als Ergebnis des PEEK-Befehls erhalten wir eine Dezimalzahl:

```
. ? PEEK(1000)
   2
```

Mit dem POKE-Befehl können Speicherstellen im Hauptspeicher geändert werden. Dabei sollte man sehr vorsichtig sein, da leicht unvorhergesehene Dinge passieren können. Hier ein Beispiel, wie der Befehl wirkt:

```
. ? PEEK(1000)
   2
. ? POKE 1000,1,2,3,4,5
. ? PEEK(1000)
   1
. ? PEEK(1001)
   2
. ? PEEK(1002)
   3
. ? PEEK(1003)
   4
. ? PEEK(1004)
   5
```

Diese Befehle sollte man nur anwenden, wenn man das Betriebssystem des Computers sehr gut kennt. Dabei muß man beachten, daß das BASIC-Betriebssystem anders aufgebaut ist als das CP/M-Betriebssystem. Außerdem benötigt man gute Kenntnisse in der Assemblersprache.

Anhang

Zwischeninhaltsverzeichnis

Anhang A

Erstellen von Befehlsdateien mit WordStar

Bis jetzt haben wir Befehlsdateien immer mit dem eingebauten dBASE-II-Editor geschrieben und editiert. Dieser Texteditor hat aber einige Nachteile. Programme von mehr als 4000 Zeichen werden nicht vollständig abgespeichert und die komfortablen Funktionen einer Textverarbeitung, wie das Einlesen von Textbausteinen oder das Verschieben von Textblöcken, sind nicht vorhanden.

Daher sollte man längere Programme mit einer Textverarbeitung wie WordStar schreiben. Diese Textverarbeitung enthält auch das Programm Mailmerge, mit dem Serienbriefe geschrieben werden können. Die dazu benötigten Adressen kann man einer dBASE-II-Datei entnehmen.

Wenn man über zwei Diskettenlaufwerke verfügt, legt man die WordStar-Diskette in Laufwerk A und die Diskette mit dBASE II und den Programmen in Laufwerk B. Hat man ein Diskettenlaufwerk, schreibt man das Programm mit WordStar auf einer gesonderten Diskette und kopiert dieses dann auf die Diskette mit dBASE II. Kleinere Fehler verbessert man in diesem Fall am besten mit dem dBASE-II-Editor, da das Kopieren und Diskettenwechseln einige Zeit benötigt.

Legen Sie nun die WordStar-Diskette in Laufwerk A ein, und starten Sie die Textverarbeitung mit der Eingabe von WS:

```
A>WS
```

Es erscheint folgendes Menü auf dem Bildschirm:

```
        Kein Text in Bearbeitung

D=Bearbeitung einer Text-Datei          H=Hilfsstufe setzen
N=Bearbeiten einer Programm-Datei       X=Ausgang Betriebssystem
M=MIX-Druck einer Datei                 P=Datei drucken
F=Inhaltsverzeichnis aus (EIN)          Y=Datei loeschen
L=Angemeldetes Laufwerk wechseln        O=Datei kopieren
R=Programm aufrufen                     E=Dateinamen aendern
S=SpellStar aufrufen
```

Hier wählen Sie den Menüpunkt N aus. Wir dürfen dBASE-II-Programme nicht als Text-Datei schreiben, da eine Textdatei Steuerzeichen enthält, die in einem dBASE-II-Programm nicht erlaubt sind.

WordStar fragt dann, wie das Programm heißen soll, und erklärt, wie ein Programmname aufgebaut ist. Wir müssen daran denken, die Namenserweiterung .CMD mit anzugeben:

```
Name der Datei zum Bearbeiten? TEST.CMD
```

Jetzt sind wir im eigentlichen Texteditor. Im oberen Bildschirmdrittel wird eine Liste möglicher Befehle angezeigt. Ein WordStar-Befehl ist eine Verbindung der Taste CONTROL mit ein bis zwei weiteren Buchstaben. Die CONTROL-Taste wird auf dem Bildschirm als ↑ dargestellt. Außer den direkt angezeigten Befehlen gibt es noch weitere, die man erhält, wenn man die CONTROL-Taste zusammen mit den Tasten Q, K, O oder P betätigt.

Die Q-Taste steht für sogenannte Quick-Befehle, mit denen man sich besonders schnell auf dem Bildschirm bewegen kann. Der Befehl <^QR> bringt uns beispielsweise an den Anfang eines Textes. Mit <^QC> erreichen wir das Textende.

Die K-Taste steht für eine Gruppe von Befehlen, mit denen Textblöcke gelöscht, kopiert oder verschoben werden. Außerdem kann man mit diesen Befehlen Textdateien in die aktuelle Datei einlesen oder Textblöcke in andere Dateien abspeichern. Zunächst markiert man dazu den Anfang des gewünschten Textblocks mit <^KB>. Dann bewegt man den Cursor auf das Ende des gewünschten Blocks und gibt <^KK> ein. Jetzt kann der markierte Block mit <^KV> verschoben, mit <^KC> kopiert oder mit <^KY> gelöscht werden.

Will man eine Datei einlesen, so gibt man <^KR> ein und wird gefragt, welche Datei man einlesen möchte:

```
Name der Datei zum Lesen?
```

Diese Funktion ist sehr wichtig, wenn man Teile eines Programms in ein anderes kopieren will. Weitere wichtige Funktionen sind <^KD> und <^KQ>. Mit <^KD> speichert man das Programm ab, und mit <^KQ> verläßt man das Programm, ohne es abzuspeichern.

Die Befehle, die mit ^O beginnen, sind zur Formatierung des Bildschirms gedacht. Hier sollten wir mit <^OR> den rechten Rand auf 78 Zeichen einstellen:

```
^OR
Rechter Rand in Spalte (ESC wenn Zeiger-Spalte)? 78
```

Diese Einstellung muß jedesmal neu erfolgen, wenn wir ein Programm editieren wollen. Wenn man den rechten Rand auf 78 Zeichen begrenzt, verhindert man spätere Fehler beim Starten des dBASE-II-Programms; denn dBASE II bearbeitet nur die ersten 77 Zeichen einer Programmzeile.

Alle Befehle, die mit ^P zusammenhängen, betreffen den Ausdruck des Programms. Diese Befehle sollten in einem dBASE-II-Programm nicht angewendet werden, da sie Steuerzeichen im Programm erzeugen.

Die beste Möglichkeit, den Umgang mit WordStar zu erlernen, ist viel zu üben. Dazu liest man sich die Bildschirmmitteilungen genau durch und probiert alle Befehle aus.

Wenn man ein Programm eingegeben hat, verläßt man den Editiermodus mit <^KD> und befindet sich wieder im Hauptmenü. Im Inhaltsverzeichnis unterhalb des Menüs ist jetzt der neue Dateiname zu sehen.

Das neue Programm kann man mit WordStar auf das Laufwerk B kopieren und dort mit dBASE II austesten.

Anhang B

Alle dBASE-II-Befehle in alphabetischer Reihenfolge

Mathematischer Operator	Bedeutung
+	Addition
-	Subtraktion
*	Multiplikation
()	Klammer

Relationaler Operator	Bedeutung
<	kleiner als
>	größer als
<= oder =<	kleiner gleich
>= oder =>	größer gleich
<> oder #	ungleich

Logischer Operator	Bedeutung
.NOT.	NICHT
.AND.	UND
.OR.	ODER
()	Klammer

Systemvariable, Kennzeichen	Bedeutung
#	Speichert die aktuelle Satznummer.
*	Markiert zur Löschung vorgesehene Datensätze.
EOF	Logische Variable, die das Dateiende anzeigt.

Funktion	Bedeutung
`+`	Verkettet zwei Zeichenketten.
`-`	Verkettet zwei Zeichenketten. Leerzeichen werden nach hinten verschoben.
`@ 'X','Y'`	Setzt den Cursor auf die angegebene Position.
`&'Variable'`	Makro. Eine Variable wird durch ihren Inhalt ersetzt.
`!('Variable,Zeichenkette')`	Wandelt Klein- in Großbuchstaben um.
`? 'Variable,Zeichenkette'`	Gibt die nachfolgende(n) Variable(n) bzw. Zeichenkette(n) am Bildschirm oder Drucker aus.
`@('Variable,Zeichenkette', 'Variable,Zeichenkette')`	Sucht den Anfang der ersten Zeichenkette in der zweiten Zeichenkette.
`$('Variable,Zeichenkette', 'Anfang','Länge')`	Kopiert den angegebenen Teil der Zeichenkette.
`CHR('Zahl')`	Wandelt die Zahl in ein Zeichen nach ASCII um.
`FILE('Dateiname')`	Ergibt logisch True, wenn die genannte Datei vorhanden ist.
`INT('Variable,Wert')`	Wandelt eine numerische Variable oder eine Dezimalzahl in eine Integerzahl um.
`LEN('Variable,Zeichenkette')`	Gibt die Länge der Variablen bzw. Zeichenkette aus.
`RANK('Variable,Zeichenkette')`	Ergibt den ASCII-Wert eines Zeichens.

Funktion	Bedeutung
`STR('Variable,Wert','Länge', 'Dezimalstellen')`	Wandelt eine numerische Variable oder eine Dezimalzahl in eine Zeichenkette um.
`TEST('Variable')`	Überprüft die Zulässigkeit einer Eingabe in die genannte Variable.
`TRIM('Variable')`	Entfernt die hinteren Leerzeichen.
`TYPE('Variable')`	Gibt den Typ einer Variablen aus.
`VAL('Variable,Zeichenkette')`	Wandelt eine alphanumerische Variable oder eine Zeichenkette in einen numerischen Wert um.

Befehl	Bedeutung
`ACCEPT 'Zeichenkette'TO'Variable'`	Eine Zeichenkette kann von der Tastatur aus in eine Variable eingegeben werden.
`APPEND`	Ermöglicht, neue Sätze zu einer eröffneten Datei hinzuzufügen.
`APPEND BLANK`	Fügt einen Leersatz an eine eröffnete Datei an.
`APPEND FROM 'Dateiname' [SDF] [FOR 'Bedingung'] [WHILE 'Bedingung'] [DELIMITED]`	Aus der angegebenen Datei werden Sätze an das Ende der gerade eröffneten Datei angefügt.
`CANCEL`	Bricht das laufende Programm ab.
`CHANGE ['Bereich'] FIELD 'Feld(er)' [FOR 'Bedingung']`	Führt Änderungen in einer Datei aus.
`CLEAR [GETS]`	Schließt alle geöffneten Dateien und löscht alle Speichervariablen.

Befehl	Bedeutung
`CLEAR [GETS]`	Der Parameter GETS löscht zusätzlich alle GET-Befehle.
`CONTINUE`	Führt einen LOCATE-Befehl fort.
`COPY ['Bereich'] TO 'Dateiname' [STRUCTURE] [FIELD 'Feld(er)'] [FOR 'Bedingung'] [SDF] [WHILE 'Bedingung'] [DELIMITED [WITH 'Endemarke']]`	Kopiert in Abhängigkeit der Parameter Daten aus einer Datenbank in eine andere Datei.
`COPY TO 'Dateiname' STRUCTURE EXTENDED`	Kopiert die Struktur der gerade eröffneten Datei in die Sätze einer neuen Datei. Jeder Satz enthält die Beschreibung eines Feldes.
`COUNT ['Bereich'] [FOR 'Bedingung'] TO ['Variable']`	Zählt die Datensätze, die der angegebenen Bedingung entsprechen.
`CREATE 'Dateiname'`	Erzeugt eine neue Datei.
`CREATE 'Dateiname' FROM 'Dateiname'`	Erzeugt eine neue Datei, deren Dateistruktur in den Datensätzen der Datei hinter "FROM" beschrieben wird.
`DELETE [Bereich] [FOR 'Bedingung'] [WHILE 'Bedingung']`	Markiert Datensätze zur Löschung.
`DELETE FILE 'Dateiname'`	Löscht eine Datei von der Diskette.
`DELETE FILE LIKE 'Dateiname'`	Löscht mehrere Dateien von der Diskette. Die Dateien werden über den ?-"Joker" im Dateinamen angesprochen.

Befehl	Bedeutung
`DISPLAY ['Bereich'] [FOR 'Bedingung'] ['Felder'] [OFF] [FIELDS 'Feld(er)'] [WHILE 'Bedingung']`	Zeigt die ausgewählten Felder an.
`DISPLAY STRUCTURE`	Zeigt die Dateistruktur einer eröffneten Datei an.
`DISPLAY MEMORY`	Zeigt die Speichervariablen an.
`DISPLAY FILES [ON 'Laufwerk'] [LIKE 'Dateiname']`	Zeigt das Inhaltsverzeichnis der Diskette an.
`DISPLAY STATUS`	Zeigt den Systemzustand an.
`DO 'Programm'`	Ruft ein Programm auf.
`DO CASE 'Bedingung'`	Verzweigt das Programm.
`DO WHILE 'Bedingung'`	Führt eine Schleife durch, solange die angegebene Bedingung erfüllt ist.
`EDIT ['nnnnn']`	Ermöglicht, den angegebenen Datensatz zu bearbeiten.
`EJECT`	Der Drucker macht einen Blattvorschub.
`ELSE`	Alternative Verzweigung des Programms. Gehört zum IF-Befehl.
`ENDCASE`	Beendet die CASE-Verzweigung.
`ENDDO`	Beendet die DO WHILE-Schleife.
`ENDIF`	Beendet einen IF-Befehl.

Befehl	Bedeutung
`ENDTEXT`	Beendet einen Text-Block.
`ERASE`	Löscht den Bildschirm.
`FIND 'Schlüssel'`	Sucht einen Datensatz in einer indizierten Datei.
`GO [TO] [RECORD] 'nnnnn'`	Zeigt auf den angegebenen Satz.
`GO [TO] TOP`	Zeigt auf den Anfang der Datei.
`GO [TO] BOTTOM`	Zeigt auf das Ende der Datei.
`HELP ['Befehl']`	Gibt Hilfsinformationen aus.
`IF 'Bedingung'`	Verzweigt das Programm. Gehört zu den Befehlen ELSE und ENDIF.
`INDEX [ON 'Schlüssel' TO 'Indexdatei']`	Legt eine Indexdatei zu der eröffneten Datei an.
`INPUT ['Meldung'] TO 'Variable'`	Gibt eine Meldung aus und speichert die Eingaben des Anwenders in der angegebenen Variablen.
`INSERT [BEFORE] [BLANK]`	Fügt einen Datensatz vor oder nach dem aktuellen Satz ein. Mit dem Parameter BLANK wird ein Leersatz ausgegeben.
`JOIN TO 'Datei' FOR 'Bedingung' [FIELDS 'Feld(er)']`	Verbindet zwei Dateien und kopiert die Sätze, die die Bedingung erfüllen, in eine dritte Datei.
`LIST ['Bereich'] [FOR 'Bedingung'] ['Feld(er)'] [FIELDS 'Feld(er)'] [WHILE 'Bedingung']`	Listet die ausgewählten Felder auf.

Befehl	Bedeutung
`LIST STRUCTURE`	Listet die Dateistruktur einer eröffneten Datei auf.
`LIST MEMORY`	Listet die Speichervariablen auf.
`LIST FILES [ON 'Laufwerk'] [LIKE 'Dateiname']`	Listet das Inhaltsverzeichnis der Diskette auf.
`LOCATE 'Bereich' FOR 'Bedingung'`	Sucht nach einem Datensatz.
`LOOP`	Beendet eine DO-WHILE-Schleife vor dem ENDDO.
`MODIFY COMMAND 'Programmname'`	Ruft den Editor auf zum Editieren eines Programms. Kann auch zur Editierung einer anderen ASCII-Datei verwendet werden.
`MODIFY STRUCTURE 'Dateiname'`	Ändert eine vorhandene Dateistruktur. Alle Daten in dieser Datei werden gelöscht!
`NOTE`	Kommentar in einem Programm. Anstelle von NOTE kann auch ein * stehen.
`PACK`	Entfernt alle zur Löschung markierten Datensätze.
`QUIT`	Beendet dBASE II.
`READ [NOUPDATE]`	Liest die durch GET gemachten Eingaben in einem Menü.
`RECALL ['Bereich']`	Hebt die Löschmarkierungen im angegebenen Bereich wieder auf.
`RELEASE ['Variable']`	Löscht die angegebenen

Befehl	Bedeutung
`[ALL [LIKE 'Bereich']]` `[ALL [EXCEPT 'Bereich']]`	Variablen.
`REMARK 'Kommentar'`	Gibt einen Kommentar am Bildschirm aus.
`RENAME 'Dateiname' TO 'Dateiname'`	Benennt Dateinamen um.
`REPLACE ['Bereich'] 'Feld' WITH 'Ausdruck' FOR 'Bedingung'`	Ersetzt in der gesamten Datei oder abhängig von der spezifizierten Bedingung das angegebene Feld mit dem Ausdruck.

`REPORT ['Bereich']`
`[FORM 'Dateiname']`
`[TO PRINT] [PLAIN]`
`[FOR 'Bedingung']`
`[WHILE 'Bedingung']`
	Startet den Listengenerator oder gibt eine definierte Liste aus.

`RESET ['Laufwerk']`	Teilt dem Betriebssystem mit, daß eine Diskette ausgetauscht wurde.
`RESTORE FROM 'Dateiname' [ADDITIVE]`	Lädt Memory-Variable aus der angegebenen Memory-Datei in den Hauptspeicher. Die vorhandenen Variablen werden gelöscht!
`RETURN`	Beendet ein Unterprogramm.

`SAVE TO 'Dateiname'`
`[ALL LIKE 'Variable']`
`[ALL EXCEPT 'Bereich']`
	Speichert Variable in die angegebene Datei.

`SELECT [PRIMARY] [SECONDARY]`	Schaltet zwischen der Vorder- und Hintergrunddatei um.

Befehl	Bedeutung
`SET ALTERNATE TO 'Dateiname'`	Es wird eine Textdatei erzeugt, die ein Protokoll aller folgenden Bildschirmausgaben enthält. Durch den Befehl "SET ALTERNATE ON" wird das Protokoll gestartet.
`SET BELL ON`	Die akustische Fehlermeldung (Piepton) ist eingeschaltet (Standard).
`SET BELL OFF`	Die Funktion ist abgeschaltet.
`SET CARRY ON`	Die Daten aus dem vorhergehenden Datensatz werden in Verbindung mit dem APPEND-Befehl in einen neuen Datensatz kopiert.
`SET CARRY OFF`	Der Befehl APPEND erzeugt einen leeren Datensatz (Standard).
`SET COLON ON`	Die Feldbegrenzung wird durch Doppelpunkte angezeigt (Standard).
`SET COLON OFF`	Die Funktion ist abgeschaltet.
`SET CONFIRM ON`	Die RETURN-Taste schließt die Eingabe eines Feldes ab.
`SET CONFIRM OFF`	Das Feld wird automatisch verlassen, wenn das letzte Zeichen eingegeben wurde.
`SET CONSOLE ON`	Die Ausgabe erfolgt auf dem Bildschirm (Standard).
`SET CONSOLE OFF`	Es erfolgt keine Ausgabe mehr auf dem Bildschirm.

Befehl	Bedeutung
`SET DATE TO 'tt/mm/jj'`	Das Datum wird neu gesetzt.
`SET DEBUG ON`	ECHO- und STEP-Befehle werden zur Fehlersuche auf dem Drucker ausgegeben.
`SET DEBUG OFF`	Die ECHO- und STEP-Befehle werden auf dem Bildschirm ausgegeben (Standard).
`SET DEFAULT TO 'Laufwerk'`	Als Standard wird jetzt das angegebene Diskettenlaufwerk benutzt.
`SET DELETED ON`	Zum Löschen markierte Sätze können nicht mehr bearbeitet werden.
`SET DELETED OFF`	Die genannten Sätze können mit allen Befehlen außer den COPY- und APPEND-Befehlen bearbeitet werden.
`SET ECHO ON`	Alle Befehle eines laufenden Programms werden zur Fehlersuche auf dem Bildschirm angezeigt.
`SET ECHO OFF`	Die Funktion ist abgeschaltet (Standard).
`SET EJECT ON`	Mit dem REPORT-Befehl kann ein Blattvorschub erzeugt werden (Standard).
`SET EJECT OFF`	Die Funktion ist abgeschaltet.
`SET ESCAPE ON`	Ein laufendes Programm kann mit der ESCAPE-Taste unterbrochen werden (Standard).
`SET ESCAPE OFF`	Die Funktion ist abgeschaltet.

Befehl	Bedeutung
`SET EXACT ON`	Bei einem Vergleich von zwei Zeichenketten müssen alle Zeichen exakt übereinstimmen.
`SET EXACT OFF`	Die Zeichenketten dürfen unterschiedliche Längen haben. Somit werden auch Teile einer Zeichenkette gefunden (Standard).
`SET FORMAT TO PRINT`	Alle @-Befehle wirken auf den Drucker.
`SET FORMAT TO SCREEN`	Alle @-Befehle wirken auf den Bildschirm (Standard).
`SET HEADING TO 'Zeichenkette'`	Die Überschrift einer Druckausgabe durch den REPORT-Befehl wird gesetzt.
`SET INDEX TO 'Dateiname'`	Öffnet die angegebene Indexdatei.
`SET INTENSITY ON`	Eingabefelder werden am Bildschirm invers dargestellt (Standard).
`SET INTENSITY OFF`	Die Funktion ist abgeschaltet.
`SET LINKAGE ON`	Für die Bildschirmdarstellung werden zwei Datenbanken verbunden.
`SET LINKAGE OFF`	Die Funktion ist abgeschaltet.
`SET MARGIN TO nnn`	Der linke Rand der Druckausgabe wird zwischen 0 und 240 eingestellt.
`SET PRINT ON`	Die Ausgabe erfolgt auf dem Drucker.

Befehl	Bedeutung
`SET PRINT OFF`	Die Funktion ist abgeschaltet (Standard).
`SET RAW ON`	DISPLAY- oder LIST-Befehle geben die Datensätze ohne trennendes Leerzeichen zwischen den einzelnen Feldern aus.
`SET RAW OFF`	Zwischen den Feldern wird ein zusätzliches Leerzeichen angezeigt (Standard).
`SET SCREEN ON`	In Verbindung mit den Befehlen APPEND, INSERT und CREATE kann die Bildschirmeditierung benutzt werden (Standard).
`SET SCREEN OFF`	Die Funktion ist abgeschaltet.
`SET STEP ON`	Nach der Ausführung eines Befehls wird das laufende Programm unterbrochen. Diese Funktion ist bei der Fehlersuche sehr nützlich.
`SET STEP OFF`	Die Funktion ist abgeschaltet (Standard).
`SET TALK ON`	Die Ergebnisse von Befehlen werden auf dem Bildschirm angezeigt (Standard).
`SET TALK OFF`	Die Funktion ist abgeschaltet.
`SKIP [+/-] [nnnnn]`	Setzt den Dateizeiger vor bzw. zurück oder auf die angegebene Satznummer.
`SORT ON 'Feldname' TO 'Dateiname' [ASCENDING] [DESCENDING]`	Sortiert die geöffnete Datei nach dem angegebenen Feld in auf- bzw. absteigender Reihenfolge.

Befehl	Bedeutung
`STORE 'Ausdruck' TO 'Variable'`	Speichert das Ergebnis des Ausdrucks oder einen angegebenen Wert in der Variablen.

`SUM ['Bereich'] 'Feld(er)'`
`[TO 'Variable']`
`[FOR 'Bedingung']`
`[WHILE 'Bedingung']`
	Summiert alle Felder einer Datenbank auf, die der Spezifikation entsprechen.

`TEXT`	Zeigt den folgenden Text bis zum Befehl ENDTEXT an.

`TOTAL TO 'Dateiname' ON 'Schlüssel'`
`[FIELDS 'Feld(er)']`
`[FOR 'Bedingung']`
`[WHILE 'Bedingung']`
	Bildet Zwischensummen von numerischen Feldern. Das Ergebnis wird in der angegebenen Datei gesichert.

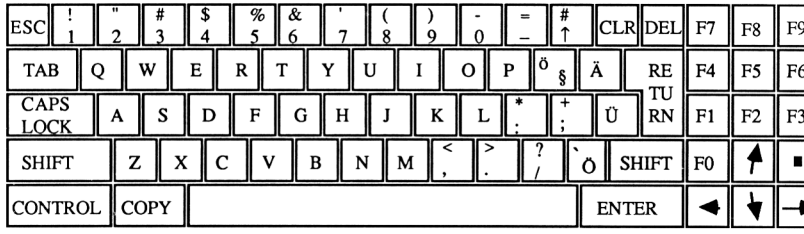
`UPDATE FROM 'Dateiname'`
`ON 'Schlüssel'`
`[ADD 'Feld(er)']`
`[REPLACE 'Feld(er)']`
`[RANDOM]`
	Bringt eine Datenbank mit Daten der angegebenen Datenbank auf den aktuellen Stand.

`USE`	Schließt alle geöffneten Datenbanken.
`USE 'Dateiname' [INDEX 'Dateiname']`	Eröffnet eine Datendatei und die zugehörigen Indexdateien.
`WAIT [TO 'Variable']`	Unterbricht den Programmablauf, bis eine Taste gedrückt wird.

Anhang C

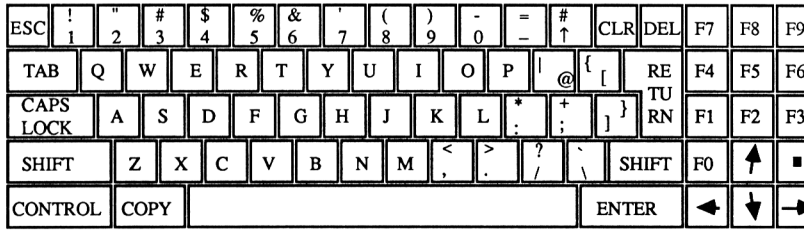
Die Tastaturbelegung

Standardbelegung (ASCII-Tastatur) und DIN-Zeichensatz



Deutsche-Norm-Tastatur (DIN-Tastatur) und DIN-Zeichensatz

Standardbelegung (ASCII-Tastatur) und ASCII-Zeichensatz



Anhang D

Der Zeichensatz

ASCII:	32	Zeichen:		ASCII:	68	Zeichen:	D
ASCII:	33	Zeichen:	!	ASCII:	69	Zeichen:	E
ASCII:	34	Zeichen:	"	ASCII:	70	Zeichen:	F
ASCII:	35	Zeichen:	#	ASCII:	71	Zeichen:	G
ASCII:	36	Zeichen:	$	ASCII:	72	Zeichen:	H
ASCII:	37	Zeichen:	%	ASCII:	73	Zeichen:	I
ASCII:	38	Zeichen:	&	ASCII:	74	Zeichen:	J
ASCII:	39	Zeichen:	´	ASCII:	75	Zeichen:	K
ASCII:	40	Zeichen:	(	ASCII:	76	Zeichen:	L
ASCII:	41	Zeichen:	)	ASCII:	77	Zeichen:	M
ASCII:	42	Zeichen:	*	ASCII:	78	Zeichen:	N
ASCII:	43	Zeichen:	+	ASCII:	79	Zeichen:	O
ASCII:	44	Zeichen:	,	ASCII:	80	Zeichen:	P
ASCII:	45	Zeichen:	–	ASCII:	81	Zeichen:	Q
ASCII:	46	Zeichen:	.	ASCII:	82	Zeichen:	R
ASCII:	47	Zeichen:	/	ASCII:	83	Zeichen:	S
ASCII:	48	Zeichen:	0	ASCII:	84	Zeichen:	T
ASCII:	49	Zeichen:	1	ASCII:	85	Zeichen:	U
ASCII:	50	Zeichen:	2	ASCII:	86	Zeichen:	V
ASCII:	51	Zeichen:	3	ASCII:	87	Zeichen:	W
ASCII:	52	Zeichen:	4	ASCII:	88	Zeichen:	X
ASCII:	53	Zeichen:	5	ASCII:	89	Zeichen:	Y
ASCII:	54	Zeichen:	6	ASCII:	90	Zeichen:	Z
ASCII:	55	Zeichen:	7	ASCII:	91	Zeichen:	[
ASCII:	56	Zeichen:	8	ASCII:	92	Zeichen:	\
ASCII:	57	Zeichen:	9	ASCII:	93	Zeichen:	]
ASCII:	58	Zeichen:	:	ASCII:	94	Zeichen:	^
ASCII:	59	Zeichen:	;	ASCII:	95	Zeichen:	_
ASCII:	60	Zeichen:	<	ASCII:	96	Zeichen:	‘
ASCII:	61	Zeichen:	=	ASCII:	97	Zeichen:	a
ASCII:	62	Zeichen:	>	ASCII:	98	Zeichen:	b
ASCII:	63	Zeichen:	?	ASCII:	99	Zeichen:	c
ASCII:	64	Zeichen:	@	ASCII:	100	Zeichen:	d
ASCII:	65	Zeichen:	A	ASCII:	101	Zeichen:	e
ASCII:	66	Zeichen:	B	ASCII:	102	Zeichen:	f
ASCII:	67	Zeichen:	C	ASCII:	103	Zeichen:	g

ASCII:	104	Zeichen:	h	ASCII:	149	Zeichen:	▌
ASCII:	105	Zeichen:	i	ASCII:	150	Zeichen:	▐
ASCII:	106	Zeichen:	j	ASCII:	151	Zeichen:	▀
ASCII:	107	Zeichen:	k	ASCII:	152	Zeichen:	α
ASCII:	108	Zeichen:	l	ASCII:	153	Zeichen:	ß
ASCII:	109	Zeichen:	m	ASCII:	154	Zeichen:	Γ
ASCII:	110	Zeichen:	n	ASCII:	155	Zeichen:	π
ASCII:	111	Zeichen:	o	ASCII:	156	Zeichen:	Σ
ASCII:	112	Zeichen:	p	ASCII:	157	Zeichen:	σ
ASCII:	113	Zeichen:	q	ASCII:	158	Zeichen:	µ
ASCII:	114	Zeichen:	r	ASCII:	159	Zeichen:	τ
ASCII:	115	Zeichen:	s	ASCII:	160	Zeichen:	Φ
ASCII:	116	Zeichen:	t	ASCII:	161	Zeichen:	Θ
ASCII:	117	Zeichen:	u	ASCII:	162	Zeichen:	Ω
ASCII:	118	Zeichen:	v	ASCII:	163	Zeichen:	δ
ASCII:	119	Zeichen:	w	ASCII:	164	Zeichen:	∞
ASCII:	120	Zeichen:	x	ASCII:	165	Zeichen:	ϕ
ASCII:	121	Zeichen:	y	ASCII:	166	Zeichen:	ε
ASCII:	122	Zeichen:	z	ASCII:	167	Zeichen:	∩
ASCII:	123	Zeichen:	{	ASCII:	168	Zeichen:	≡
ASCII:	124	Zeichen:	|	ASCII:	169	Zeichen:	±
ASCII:	125	Zeichen:	}	ASCII:	170	Zeichen:	≥
ASCII:	126	Zeichen:	~	ASCII:	171	Zeichen:	≤
ASCII:	127	Zeichen:		ASCII:	172	Zeichen:	⌠
ASCII:	128	Zeichen:		ASCII:	173	Zeichen:	⌡
ASCII:	129	Zeichen:	▪	ASCII:	174	Zeichen:	÷
ASCII:	130	Zeichen:	▪	ASCII:	175	Zeichen:	≈
ASCII:	131	Zeichen:	▬	ASCII:	176	Zeichen:	°
ASCII:	132	Zeichen:	▪	ASCII:	177	Zeichen:	∙
ASCII:	133	Zeichen:	▮	ASCII:	178	Zeichen:	·
ASCII:	134	Zeichen:	▪	ASCII:	179	Zeichen:	√
ASCII:	135	Zeichen:	▛	ASCII:	180	Zeichen:	ⁿ
ASCII:	136	Zeichen:	▪	ASCII:	181	Zeichen:	²
ASCII:	137	Zeichen:	▚	ASCII:	182	Zeichen:	■
ASCII:	138	Zeichen:	▮	ASCII:	183	Zeichen:	
ASCII:	139	Zeichen:	▜	ASCII:	184	Zeichen:	
ASCII:	140	Zeichen:	▬	ASCII:	185	Zeichen:	
ASCII:	141	Zeichen:	▙	ASCII:	186	Zeichen:	
ASCII:	142	Zeichen:	▟	ASCII:	187	Zeichen:	
ASCII:	143	Zeichen:	■	ASCII:	188	Zeichen:	
ASCII:	144	Zeichen:	▪	ASCII:	189	Zeichen:	
ASCII:	145	Zeichen:	▮	ASCII:	190	Zeichen:	
ASCII:	146	Zeichen:	▬	ASCII:	191	Zeichen:	
ASCII:	147	Zeichen:	▙	ASCII:	192	Zeichen:	
ASCII:	148	Zeichen:	▮				

ASCII:	193	Zeichen:		ASCII:	225	Zeichen:	
ASCII:	194	Zeichen:		ASCII:	226	Zeichen:	
ASCII:	195	Zeichen:		ASCII:	227	Zeichen:	
ASCII:	196	Zeichen:		ASCII:	228	Zeichen:	
ASCII:	197	Zeichen:		ASCII:	229	Zeichen:	
ASCII:	198	Zeichen:		ASCII:	230	Zeichen:	
ASCII:	199	Zeichen:		ASCII:	231	Zeichen:	
ASCII:	200	Zeichen:		ASCII:	232	Zeichen:	
ASCII:	201	Zeichen:		ASCII:	233	Zeichen:	
ASCII:	202	Zeichen:		ASCII:	234	Zeichen:	
ASCII:	203	Zeichen:		ASCII:	235	Zeichen:	
ASCII:	204	Zeichen:		ASCII:	236	Zeichen:	
ASCII:	205	Zeichen:		ASCII:	237	Zeichen:	
ASCII:	206	Zeichen:		ASCII:	238	Zeichen:	
ASCII:	207	Zeichen:		ASCII:	239	Zeichen:	
ASCII:	208	Zeichen:		ASCII:	240	Zeichen:	
ASCII:	209	Zeichen:		ASCII:	241	Zeichen:	
ASCII:	210	Zeichen:		ASCII:	242	Zeichen:	
ASCII:	211	Zeichen:		ASCII:	243	Zeichen:	
ASCII:	212	Zeichen:		ASCII:	244	Zeichen:	
ASCII:	213	Zeichen:		ASCII:	245	Zeichen:	
ASCII:	214	Zeichen:		ASCII:	246	Zeichen:	
ASCII:	215	Zeichen:		ASCII:	247	Zeichen:	
ASCII:	216	Zeichen:		ASCII:	248	Zeichen:	
ASCII:	217	Zeichen:		ASCII:	249	Zeichen:	
ASCII:	218	Zeichen:		ASCII:	250	Zeichen:	
ASCII:	219	Zeichen:		ASCII:	251	Zeichen:	
ASCII:	220	Zeichen:		ASCII:	252	Zeichen:	
ASCII:	221	Zeichen:		ASCII:	253	Zeichen:	
ASCII:	222	Zeichen:		ASCII:	254	Zeichen:	
ASCII:	223	Zeichen:		ASCII:	255	Zeichen:	
ASCII:	224	Zeichen:					

Anhang E

Lösungen zu den Übungen

Übung 1:

1. Ein Prompt ist die Bereitschaftsanzeige eines Programms, hinter dem wir Befehle eingeben können.

2. + 3.

Prompt	zugehöriges Programm
A>	CP/M Plus
B>	CP/M Plus
*	PIP
.	DBASE

4. Mit dem Programm DISCKIT3.

5. Das Programm PIP.

6. Zum Einstellen des DIN- bzw. ASCII-Zeichensatzes.

7. Zum Belegen der Tastatur entsprechend der Zeichensätze.

8. Mit der Eingabe DBASE <RETURN>.

9. Mit dem Befehl QUIT.

Übung 2:

1.
```
Strukturdaten für Datei: A:ARTIKEL .DBF
Anzahl der Sätze: 00000
Datum der letzten Aktualisierung: 31/12/86
Primäre Datei
Feld  Name        Typ  Länge  Dez.st.
001       ARTIKELNR     N     004
002       ARTIKELBEZ    C     015
003       BESTAND       N     004
004       MINBESTAND    N     004
005       EK            N     006     002
006       VK            N     006     002
** Gesammt **             00040
```

2. 40 Bytes = 40 Zeichen

3. Es können 184.320 / 40 = 4608 Artikel abgespeichert werden.

Übung 3:

1. `USE KUNDEN <RETURN>`

2. s. Kapitel Anzeigen von Daten einer Datei.

Übung 4:

1.
```
DISPLAY ALL OFF <RETURN>
DISPLAY ALL KUNDENNR NACHNAMEUMSATZ OFF <RETURN>
SET PRINT ON <RETURN>
DISPLAY ALL OFF <RETURN>
SET PRINT OFF <RETURN>
```

2. Der Parameter OFF unterdrückt die Ausgabe der Zeilennummern.

3.
```
SET PRINT ON  <RETURN> oder <^P>
LIST NACHNAME VORNAME TELEFON OFF <RETURN>
SET PRINT OFF <RETURN> oder <^P>
```

Übung 5:

1.
```
USE KUNDEN <RETURN>
SORT ON PLZ TO SORTIERT <RETURN> oder
SORT ON PLZ TO SORTIERT ASCENDING <RETURN>
USE SORTIERT <RETURN>
LIST <RETURN>
COPY TO KUNDEN <RETURN>
DELETE FILE SORTIERT.DBF <RETURN>
```

2. Man gibt den Parameter "DESCENDING" am Ende des SORT Befehles an, z.B. "SORT ON PLZ TO SORTIERT DESCENDING".

Übung 6:

1. Eine Indexdatei benötigt weniger Speicherplatz als eine Sort-Datei. Indizieren geht wesentlich schneller als Sortieren. Eine Indexdatei läßt sich aktualisieren, ohne sie neu anzulegen.

2.
```
USE ARTIKEL <RETURN>
INDEX ON ARTIKELBEZ TO ARTBEZ <RETURN>
LIST <RETURN>
```

3.
```
USE KUNDEN <RETURN>
INDEX ON PLZ+STRASSE+NACHNAME TO PLZSTRNA <RETURN>
LIST <RETURN>
```

Übung 7:

1. Mit dem Parameter OFF.
2. Der ALL-Parameter erreicht, daß ein Befehl auf alle Datensätze wirkt.
3. Hinter dem FIELD- bzw. FIELDS-Parameter gibt man die Felder an auf die der vorangegangene Befehl wirken soll.

Übung 8:

1. a.
```
NACHNAME='Schwarze' .AND. VORNAME='Helmut' .AND.
STRASSE='Akazienallee 27'
```
   b.
```
PLZ='4100'
```
   c.
```
@('71234',TELEFON)
```
   d.
```
(STRASSE='Schillergasse' .OR. STRASSE='Reiterweg') .AND.
UMSATZ > 0
```

Übung 9:

1.
```
USE ARTIKEL <RETURN>
DELETE 3 <RETURN>
DELETE 4 <RETURN>
DELETE 4 <RETURN>
```

   oder

```
USE ARTIKEL <RETURN>
DELETE 6 <RETURN>
DELETE 5 <RETURN>
DELETE 3 <RETURN>
```

2.
```
DELETE FOR ARTIKELBEZ="Colour Monitor" <RETURN>
```

3.
```
RECALL ALL <RETURN>
```

4. Dies ist der Befehl PACK.

5. Mit <Ctrl> <U>.

Übung 10:

1.
```
USE KUNDEN INDEX NAME,PLZ <RETURN>
SUM ALL UMSATZ FOR PLZ > '3999' .AND. PLZ > '5000' <RETURN>
```
2.
```
. SUM (VK/1.14-EK)/EK*100 FOR ARTIKELBEZ='Joystick de Luxe'
  115.99
```

   Die Umsatzrentabilität beträgt 115.99 % für diesen Artikel.

3.
```
. SUM ALL (INT(MINBESTAND*1.3)-BESTAND)*EK FOR INT(MINBESTAND*1.3)
  > BESTAND
  1849.90
```

Es werden 1849,90 DM benötigt um den Mindestbestand um 30 % zu erhöhen. Bei dieser Aufgabe ist die INT-Funktion notwendig, da nicht mit Bruchteilen eines Artikels gerechnet werden darf.

Übung 11:

1. Mit dem Befehl SET DELETE ON.
2. `COUNT FOR NACHNAME='Maier' .OR. NACHNAME='Meier'`
3. `COUNT FOR UMSATZ > 5000`

Übung 12:

1. a.
```
? 123.00*12/4/(1+7/2)
      82.00
```
   b.
```
? 6.13*(12+4)/128/7+1.23
       1.33
```
   c.
```
? 10.00/3
    3.33
```
   d.
```
? 18.00/4+12/26
     4.96
```

2.
```
? INT(1.23*12345+5.217/678.21)
 15184
```

Übung 13:

1. a. Nicht zulässig. Eine Ziffer steht am Anfang des Namens.
   b. Nicht zulässig. Der Name ist durch eine Leerstelle unterbrochen und außerdem länger als zehn Zeichen.
   c. Zulässig.
   d. Zulässig.
   e. Nicht zulässig. Ein Doppelpunkt steht am Anfang des Namens.
   f. Zulässig.

2. Es können maximal vierundsechzig Variable gleichzeitig vorhanden sein. Dabei muß darauf geachtet werden, daß die Gesamtanzahl des durch die Variablen belegten Speicherplatzes 1536 Zeichen nicht übersteigt.

3. Der Befehl RELEASE.

4. Mit dem Befehl DISPLAY MEMORY.

Übung 14:

1. Alphanumerische Variable.

2. Zur Eingabe von numerischen und logischen Werten.

3. Mit dem Befehl VAL.

4. Der ?-Befehl.

Übung 15:

```
. USE KUNDEN INDEX NAME
. REPORT
Bitte Dateinamen für Bericht eingeben: KUNDEN
Eingabe: m=linker Rand, l=Zeilen pro Seite, w=Zeilenbreite
m=1, l=72, w=85
Mit Seitenüberschrift? (J/N) J
Bitte Seitenüberschrift eingeben: KUNDEN-LISTE
Leerzeile zwischen den Sätzen? (J/N) N
Gesamtsummen erforderlich? (J/N)J
Spalte  Breite, Inhalt
001     5,KUNDENNR
Bitte Überschrift eingeben: <NR;-----
002     18,TRIM(NACHNAME)+", "+TRIM(VORNAME)
Bitte Überschrift eingeben: <NACHNAME, VORNAME;------------------
003     15,STRASSE
Bitte Überschrift eingeben: <STRASSE;---------------
004     4,PLZBitte Überschrift eingeben: <PLZ;----
005     12,ORT
Bitte Überschrift eingeben: <WOHNORT;------------
006     12,TELEFON
Bitte Überschrift eingeben: <TELEFON;------------
007     6,UMSATZ
Bitte Überschrift eingeben: <UMSATZ;------
```

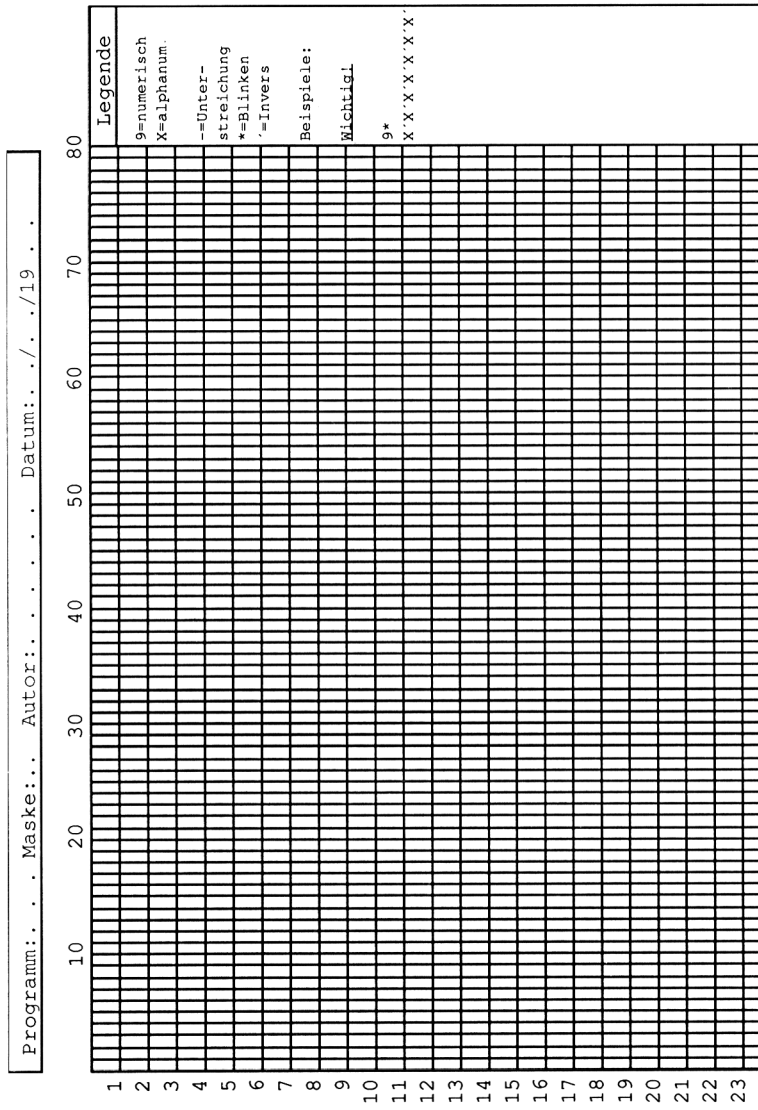
Anhang F

Programmierformulare

Formular zur Planung einer Dateistruktur

Dateiname:	Anwendung:	Datum: . ./ . . /19 . .

Nr.	Feldname	Feldtyp	Feldlänge	Nachkommastellen	Schlüsselfeld?
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					
23					
24					
25					
	Summe:				


Programm:. . . Maske:.. Autor:. . . . . Datum:. ./. ./19 . .
Legende
9=numerisch
X=alphanum.
-=Unter-
streichung
*=Blinken
'=Invers
Beispiele:
Wichtig!
9*
X'X'X'X'X'X'X'
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
10 20 30 40 50 60 70 80

Stichwortverzeichnis

Die SYBEX-Bibliothek

Schneider

ARBEITEN MIT DEM SCHNEIDER CPC
von Hans Lorenz Schneider – eine umfassende und didaktisch aufbereitete Arbeitshilfe für Anfänger, aber auch Fortgeschrittene finden ein Bündel von Tips und Tricks. 288 Seiten, 113 Abbildungen, Best.-Nr.: **3603** (1985)

DAS SCHNEIDER CPC GRAFIKBUCH
von H. L. Schneider – führt Sie schrittweise in die vielfältigen Grafikmöglichkeiten Ihres CPC 464, 664 oder 6128 ein; vom Grundlagenwissen bis zur Hardcopy. 328 Seiten, zahlr. Abbildungen, Best.-Nr. **3611** (1986)

SCHNEIDER CPC STARTEXTER
von Reinhold Krumscheid – 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)

Das SCHNEIDER CPC SYSTEMBUCH
von Günter Woigk – Hier erfahren Sie alles über das Betriebssystem Ihres Schneider CPC und – wie Sie es optimal ausnutzen. Z80 Maschinensprache, Systemadressen, ROM-Routinen u.v.m. Mit vielen Anwendungsbeispielen. Ca. 680 Seiten, Best.-Nr. **3606** (1987)

SCHNEIDER CPC STARDATEI
Reinhold Krumscheid, Autor des CPC StarTexter, hat hier eine leistungsstarke Dateiverwaltung – kompatibel zur Textverarbeitung – vorgelegt; anwendbar auf CPC 464, 664 und 6128. Diskette mit ausführlichem Trainingsbuch, Best.-Nr. **3423** (1986)

SCHNEIDER CPC – EINFÜHRUNG IN WORDSTAR
von Arthur Naiman – Der SYBEX-Bestseller „Einführung in WordStar" in einer auf den Schneider CPC zugeschnittenen Version – ergänzt durch wertvolle Hinweise für die Installation von Druckern sowie Systempatches. 280 Seiten, ca. 40 Abb., Best.-Nr. **3646** (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: 287639 SYBEX UR

SYBEX
6–8, Impasse du Curé
75018 Paris
Tel.: 1/203-95-95
Telex: 211.801 f

SYBEX

Schneider CPC

Arbeiten mit dBASE II

Das Anwenderprogramm dBASE II gehört weltweit zu den erfolgreichsten Datenbankprogrammen. Seit jüngerer Zeit ist dieses Programm auch für Homecomputersysteme zu akzeptablen Preisen verfügbar.

Der Autor des vorliegenden Buchs führt den interessierten Leser kompetent und leicht verständlich in den Umgang mit dBASE II auf dem Schneider CPC ein. Sie lernen mühelos anhand einer Vielzahl praxisnaher Beispiele, erfolgreich mit diesem leistungsfähigen Softwarewerkzeug zu arbeiten.

Aus dem Inhalt:

- Installation von dBASE II
- Das relationale Datenbankmodell
- Möglichkeiten von dBASE II
- Arbeiten im interaktiven Befehlsmodus
- Programmieren mit dBASE II
- Aufbau einer Adreßverwaltung
- Artikelverwaltung mit dBASE II
- Schreiben und Drucken von Rechnungen unter dBASE II
- Datenaustausch mit anderen Programmen
- Makros
- Erstellen von Befehlsdateien mit WordStar
- Übungen mit Lösungen

„Arbeiten mit dBASE II" gehört in die Fachbibliothek jedes Benutzers von dBASE II auf einem Schneider CPC, der Problemlösungen für Datenbank-orientierte private wie auch geschäftliche Aufgaben benötigt.

ISBN 3-88745-660-2

DM 48,—
sFr 44,20
öS 374,—

9 783887 456603

3660

SYBEX

Schneider CPC – Arbeiten mit dBASE II

Beisecker

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/