Markt&Technik

Dr. Peter Albrecht

dBASE II

für den Schneider CPC

Einführung in die Programmierung und das Arbeiten mit dBASE II.

dBASE II für den Schneider CPC

Dr. Peter Albrecht

dBASE II für den Schneider CPC

Einführung in die Programmierung und das Arbeiten mit dBASE II

Markt & Technik Verlag

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Albrecht, Peter:
dBASE II für den Schneider CPC : Einf. in d. Programmierung u. d. Arbeiten mit dBASE II / Peter Albrecht. —
Haar bei München : Markt-und-Technik-Verlag, 1985
ISBN 3-89090-188-3

15 14 13 12 11 10 9 8 7 6 5 4 3 2
89 88 87 86 85

ISBN 3-89090-188-3

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Jantsch, Günzburg
Printed in Germany

Vorwort

Personal Computer werden nicht nur für den Geschäftsmann, sonder auch für den Privatmann zunehmend erschwinglicher. Einige professionell einsetzbare Computersysteme wie der Schneider CPC sind heute schon zum Preis eines Videorecorders erhältlich. Damit haben die Hersteller eine der Voraussetzungen für den Computereinsatz auf breitester Basis geschaffen. Eine andere, die es zu erfüllen gilt, ist die Bedienerfreundlichkeit. Wenn der Computer ein ebenso weitverbreitetes und selbstverständliches Werkzeug wie das Automobil oder Telefon werden soll, dann muß man mit ihm auch umgehen können, ohne Experte zu sein.

Hier sind nicht zuletzt die Produzenten von Software gefordert. Gerade das Verwalten von größeren Datenbeständen - die klassische Aufgabe der Datenverarbeitung - erfordert umfangreiche Kenntnisse über Datenstrukturen und Datenzugriffsmöglichkeiten, die man beim Endanwender nicht ohne weiteres voraussetzen darf.

Die Antwort der Softwareentwickler darauf sind Datenbankverwaltungssysteme, die sich rasch steigender Beliebheit erfreuen. Das bislang populärste System dieser Art ist sicherlich dBASE II, das als "relationales Datenbanksystem" in seiner Klasse Maßstäbe gesetzt hat.

dBASE II war ursprünglich für 8-Bit-Mikrocomputer geschrieben und später auch auf 16-Bit-Systeme übertragen worden. Für letztere und für künftige 32-Bit-Rechner wurde im Herbst 1984 zusätzlich das Datenbanksystem dBASE III auf den Markt gebracht, das für den Benutzer weitgehend identisch mit dBASE II ist und das sich innerhalb kurzer Zeit ebenfalls an die Spitze der Bestsellerliste setzte. dBASE II ist in einer CP/M-Version nun auch für die Schneider-CPC-Computer verfügbar, und zwar im vollen Leistungsumfang.

Zur Popularität von dBASE tragen sicherlich drei Eigenschaften bei, die es aus der Vielzahl konkurrierender Produkte herausheben:

- die Einfachheit und Schnelligkeit, mit der Dateistrukturen definiert, benutzt und geändert werden können,

- die hohe Flexibilität im Datenzugriff ohne starre Zugriffspfade,

- die integrierte Kommandosprache, die über die Befehle zur reinen Datenmanipulation hinausgeht und eine komplette Anwendungsprogrammierung zuläßt.

Insbesondere die letztgenannte Eigenschaft ist für den geübten Benutzer von größter Bedeutung. Ihm wird die Möglichkeit geboten, auf konventionelle Sprachen wie BASIC oder COBOL völlig zu verzichten und eine Sprache auf höherem Niveau zu benutzen, die jenen an Mächtigkeit weit überlegen ist.

Aber auch der Mikrocomputer-Neuling wird sich mit dBASE schnell anfreunden. Er muß nicht den gesamten Befehlsvorrat von dBASE lernen, bevor er erste Erfolge verzeichnen kann. Schon nach Beherrschung weniger Befehle ist er in der Lage, Dateien zu erstellen, mit Informationen zu laden und auszuwerten. Dabei hilft ihm ein integrierter Report-Generator, ein Unterprogramm, das im Dialog mit dem Benutzer Auswertungen und Berichte gestaltet und in Tabellenform ausdruckt.

Das vorliegende Buch soll nach einer kurzen Einführung in den Komplex "Datenbanken" eine Anleitung für den praktischen Umgang mit dBASE II geben. Die einzelnen Befehle werden weitgehend in der Reihenfolge besprochen, wie sie für die Arbeit in der Praxis benötigt werden. Das bedeutet für den Leser, daß er dBASE schon nach kurzer Zeit für einfache Anwendungen nutzbar und das weitere Vordringen in verfeinerte Datenbanktechniken von seinen individuellen Bedürfnissen abhängig machen kann.

Alle Befehle werden anhand von praxisnahen Beispielen vorgeführt und können vom Leser leicht nachvollzogen werden. Um den systematischen Aufbau der Beispielprogramme nicht zu beeinträchtigen, wurde bewußt auf das Besprechen sämtlicher Kommandos und aller ihrer Varianten verzichtet, deren Aufzählung mehr verwirren würde als klären.

Nach der Besprechung der Befehle wird eine Einführung in die Programmierung mit dBASE II gegeben. Einfache Programme kann schon der Anfänger erstellen, um häufig benutzte Befehlsfolgen zu speichern und bei Bedarf mit einem einzigen Kommando abzurufen. Dem fortgeschrittenen Anwender erlauben Programmschleifen und Verzweigungen die Entwicklung von komplexen, strukturierten Programmen.

Der Programmierteil wird durch eine Reihe von ausgewählten Beispielen abgerundet. Sie sollen nicht nur die Anwendung der Befehle noch einmal verdeutlichen, sondern Ihnen auch Tips und Programmiertechniken vermitteln, die allgemein anwendbar sind. Dabei werden Sie feststellen, wie vielseitig dBASE II als Programmiersprache eingesetzt werden kann.

Schließlich wird im letzten Teil der gesamte Befehlsvorrat mit allen Funktionen noch einmal zusammengefaßt und eine Übersicht über die wichtigsten CP/M-Befehle gegeben, soweit sie in der täglichen Praxis benötigt werden. Dieser Abschnitt soll dem Leser bei seiner späteren Arbeit mit dBASE als kleine Gedächtnisstütze dienen.

Inhaltsverzeichnis

1 Einleitung

Bis vor wenigen Jahren wurden Computer überwiegend nur für eine eng begrenzte Anzahl von Aufgaben genutzt. Isolierte Anwendungen in Bereichen wie Buchhaltung oder Lohn- und Gehaltsabrechnung, die kaum eine direkte Verbindung zu anderen Anwendungen hatten, bildeten die Schwerpunkte. Jedem einzelnen Applikationsprogramm waren seine eigenen Datenbestände in Form von Dateien zugeordnet. Andere Programme konnten in der Regel nicht auf diese Daten zugreifen.

Mit zunehmender Leistungsfähigkeit der Computer wuchs auch die Vielfalt der Anwendungen pro System. Das weckte sehr rasch den Wunsch, die einmal erfaßten und gespeicherten Daten allen Programmen zugänglich zu machen. Daten für jede einzelne Anwendung separat zu erfassen und zu speichern, ist nicht nur sehr aufwendig, sondern führt auch mit fast absoluter Sicherheit zu Datenbeständen mit uneinheitlichem Aktualisierungsstand.

Als Antwort auf diese Forderungen entstanden die Datenbanken bzw. Datenbankverwaltungssysteme. Ihre Aufgabe ist es, Informationen in geordneter Form zu verwalten und dem Benutzer bereitzustellen. Bei echten Datenbanken erfolgt dies in völliger Unabhängigkeit von den Anwendungsprogrammen. Damit können sie flexibel auf wechselnde Informationsbedürfnisse der Benutzer reagieren und auch beliebigen neu hinzugekommenen Programmen ihre Daten zur Verfügung stellen.

Praktisch alle Daten, die in einer Datenbank gespeichert werden, stehen in irgendwelchen Beziehungen zu anderen Daten. Der Betrag eines Monatsgehalts steht in Beziehung zum Namen eines Mitarbeiters, der wieder ist mit weiteren persönlichen Daten, mit dem Namen einer Abteilung oder einer Funktion verknüpft. Oder der Name eines Kunden ist einerseits mit seiner

Kundenadresse, andererseits mit gewissen geschäftlichen Informationen, z.B. Rechnungsdaten, verknüpft.

Bereits diese einfachen Beispiele zeigen, daß die Beziehungen der Daten untereinander recht komplex sein können. Sie müssen aber in einer Datenbank nachgebildet werden, wenn man gezielt auf solche Informationen zugreifen will. Diese Beziehungen legen weitgehend die Ordnung innerhalb einer Datenbank, die sogenannte Datenbankstruktur, fest.

Allgemein werden heute zwei verschiedene Arten von Datenbanksystemen eingesetzt, die sich in ihrem Strukturprinzip unterscheiden:

- Die (älteren) hierarchischen bzw. (neueren) Netzwerk-Datenbanken, in denen von den einzelnen Dateneinheiten mit sogenannten Zeigern auf andere, mit ihnen in Verbindung stehende Daten gewiesen wird. Mit Hilfe dieser Zeiger kann man, von einem Begriff ausgehend, alle zugehörigen Informationen erreichen. Die Gesamtheit aller Zeiger ist ein wesentlicher Bestandteil der Datenbankstruktur. Diese kann außerordentlich komplex werden und muß definiert sein, bevor mit der Datenbank gearbeitet werden kann. Derartige Datenbanksysteme sind zwar für größte Datenmengen geeignet, erfordern aber professionelle Datenbankverwalter. Netzwerk-Datenbanken sind im Bereich der Großrechner heute vorherrschend, dürften aber in nicht allzu ferner Zukunft durch relationale Datenbanken abgelöst werden.

- Die relationalen Datenbanken, denen ein verblüffend einfaches Strukturkonzept zugrunde liegt, die aber trotzdem (oder gerade deswegen) ein Höchstmaß an Flexibilität bezüglich des Datenzugriffs bieten. Nachdem das System dBASE II in diese Kategorie einzustufen ist, soll auf das Prinzip der relationalen Datenbanken etwas näher eingegangen werden.

Der Begriff "relational" stammt aus der Mathematik, und zwar aus der Relationentheorie. Danach ist eine Relation eine "Menge von Tupeln", d.h. eine Tabelle, die aus einer Anzahl gleichartiger Zeilen (den Tupeln) besteht. In die Datenverarbeitung übertragen, stellt eine Datei mit mehreren gleichartigen Datensätzen solch eine Relation dar. Beispiel: Ein Adressenverzeichnis mit Name, Vorname, Anschrift und Telefonnummer für jede eingetragene Person ist eine Relation (Bild 1.1).

NAME	VORNAME	STRASSE	ORT	TELEFON
Maier	Herbert	Amalienstr.59	München	089-912345
Huber	Josef	Sieglindenstr.5	München	089-987654
Kuhn	Peter	Fliederstr.9	Darmstadt	06151-4321
Meister	Werner	Marktplatz 3	Eching	089-753267
Schmid	Herbert	Fasanenstr.2	Berlin	030-135790
Hassold	Paul	Meisenstr.18	Hamburg	040-246875
Hofmann	Gerd	Holzweg 13	Hamburg	040-456987
Hoffmann	Peter	Alte Gasse 15	Coburg	06561-7412

Bild 1.1: Relation "Adressenverzeichnis"

Es ist nun grundsätzlich immer möglich, beliebige Daten und ihre gegenseitige Verknüpfung in zweidimensionalen Tabellen darzustellen, selbst dann, wenn zunächst komplizierte Beziehungen vorliegen. Diese Tatsache wird von relationalen Datenbanken genutzt, um sämtliche Informationen in einfachen Tabellen zu organisieren und zu speichern. Datenelemente, die in logischer Verbindung zueinander stehen, werden zu einem Datensatz zusammenfaßt.

Da alle Datensätze einer Tabelle gleichartig aufgebaut sind, wird das Festlegen der Struktur einer relationalen Datenbank verblüffend einfach. Diese Arbeit reduziert sich auf das Definieren eines Datensatzes:

> Mit der Beschreibung der Struktur eines Datensatzes
> wird die Struktur der gesamten Relation festgelegt.

Logische Beziehungen zwischen Daten verschiedener Relationen werden über den Dateninhalt selbst hergestellt. Zum Beispiel kann eine Kundennummer die Verknüpfung zwischen einer Adressendatei und einer Rechnungsdatei herstellen, wenn dieselbe Nummer in beiden Tabellen auftaucht (Bild 1.2). Damit werden Zeiger, wie sie bei anderen Datenbanksystemen für solche Verknüpfungen verwendet werden, überflüssig.

K.-NR.	NAME	VORNAME	STRASSE	ORT
123	Maier	Herbert	Amalienstr.59	8000 München
234	Huber	Josef	Sieglindenstr.5	8000 München
345	Kuhn	Peter	Fliederstr.9	6100 Darmstadt
098	Meister	Werner	Marktplatz 3	8057 Eching
761	Schmid	Herbert	Fasanenstr.2	1000 Berlin
147	Hassold	Paul	Meisenstr.18	2000 Hamburg
258	Hofmann	Gerd	Holzweg 13	2000 Hamburg

RNR	DATUM	K.-NR.	BETRAG	BEZAHLT	BDATUM
11115	28.01.85	258	1591.50	1200.00	10.02.85
11116	28.01.85	761	654.00	654.00	28.01.85
11117	28.01.85	147	980.00	980.00	10.02.85
11118	28.01.85	123	1234.00	1234.00	28.01.85
12000	28.01.85	345	2111.00	1688.80	28.01.85
12222	28.01.85	098	2155.00	2155.00	10.02.85

Bild 1.2: Zwei Relationen, die über ihren Dateninhalt im Feld K.-NR. (Kundennummer) verknüpft sind

Beachten Sie den ganz wesentlichen Unterschied zwischen den verschiedenen Arten von Datenbanken:

Bei den netzwerkartigen Datenbanken wird die Beziehung zwischen den Datenelementen in der Datenbankstruktur (mit Zeigern) **vorab** festgelegt, was mit einer gewissen Starrheit des Systems verbunden ist. Eine Änderung der Struktur, etwa eine Anpassung an neue Bedürfnisse, verursacht meist einen erheblichen Aufwand für den Datenbankverwalter.

Ob dagegen bei einer relationalen Datenbank zwei Relationen in logischer Beziehung zueinander stehen, wird ausschließlich durch den Inhalt der Tabellen festgelegt und nicht durch ihre Struktur. Die Strukturänderung einer relationalen Datenbank reduziert sich damit auf die einfache Änderung von Tabellen. Ebenso problemlos lassen sich auch neue Relationen (Dateien) erstellen und in die existierende Datenbank integrieren, ohne die vorhandenen Datenbestände zu beeinflussen. Dadurch sind relationale Datenbanksysteme prinzipiell leichter als nichtrelationale zu handhaben.

Bild 1.3 zeigt als Beispiel die Anbindung einer dritten Relation an die in Bild 1.2 gezeigten zwei Tabellen. Die neue Relation enthält zusätzliche Kundeninformationen und steht mit den alten über die Inhalte K.-NR. (Kundennummer) bzw. RNR/LRECH (Rechnungsnummer) in logischer Beziehung.

K.-NR.	NAME	VORNAME	STRASSE	ORT
123	Maier	Herbert	Amalienstr.59	8000 München
234	Huber	Josef	Sieglindenstr.5	8000 München
345	Kuhn	Peter	Fliederstr.9	6100 Darmstadt
098	Meister	Werner	Marktplatz 3	8057 Eching
761	Schmid	Herbert	Fasanenstr.2	1000 Berlin
147	Hassold	Paul	Meisenstr.18	2000 Hamburg

RNR	DATUM	K.-NR.	BETRAG	BEZAHLT	BDATUM
11116	28.01.85	761	654.00	654.00	28.01.85
11117	28.01.85	147	980.00	980.00	10.02.85
11118	28.01.85	123	1234.00	1234.00	28.01.85
12000	28.01.85	345	2111.00	1688.80	28.01.85
12222	28.01.85	098	2155.00	2155.00	10.02.85

K.-NR.	ORT	LRECH	DATUM	UMSATZ
098	8057	12222	05.01.85	5612.89
123	8000	11118	20.01.85	2501.00
147	2000	11117	05.01.85	980.00
345	6100	12000	20.01.85	2016.67
761	1000	11116	05.01.85	3005.55

Bild 1.3: Die Beziehungen zwischen drei Relationen

Wir werden übrigens die hier gezeigten Beispiele später im praktischen Teil nachvollziehen.

Ein weiterer Vorteil relationaler Datenbanken, der mindestens ebenso bedeutend ist wie die einfache Handhabung der Strukturen, ist folgender:

Die Operationen der Relationenalgebra können auf relationale Datenbanken angewandt werden. Damit wird die Verarbeitung von Daten mathematisch exakt beschreibbar. Mit einzelnen Operatorfunktionen können ganze Tabellen in einem Schritt manipuliert werden, d.h., es können Programmbefehle von enormer Mächtigkeit definiert werden. So existieren beispielsweise in dBASE Befehle, die aus einer Tabelle einen Teil abspalten oder zwei Tabellen zu einer neuen zusammenfügen und dem Benutzer eine neue Datensicht ermöglichen.

Die spezifischen Möglichkeiten, eine relationale Datenbank zu manipulieren, sind natürlich in keiner der konventionellen Programmiersprachen wie BASIC oder COBOL berücksichtigt. Es ist daher notwendig, daß ein Datenbanksystem über eine eigene Datenmanipulationssprache verfügt. Bei den meisten Datenbanken beschränkt sich diese auf die eigentliche Verwaltung der Datenbestände, während die Anwendungsprogramme in einer der üblichen Programmiersprachen zu schreiben sind.

Die Entwickler von dBASE II sind hier einen Schritt weitergegangen und haben die Datenmanipulationssprache um alle jene Befehle ergänzt, die für eine vollständige Anwendungsprogrammierung benötigt werden. Damit können nicht nur sämtliche Teile eines Programmpakets in ein und derselben Sprache geschrieben werden, sondern es können auch die Datenmanipulationen direkt in die Verarbeitungsprogramme integriert werden, ohne daß spezielle Schnittstellenprogramme zu schreiben wären. Das Ergebnis sind übersichtliche, gut strukturierte Programme von überraschender Kürze.

Das zuletzt Gesagte mag den Mikrocomputer-Neuling unter den Lesern vielleicht nicht interessieren oder ihn gar etwas verwirren. Er hat sicherlich nicht die Absicht, sofort Anwendungsprogramme zu schreiben, sondern möchte möglicherweise zunächst nur einfachere Dateiverwaltungen mit dBASE realisieren. Das ist selbstverständlich möglich. Schon mit relativ wenigen Anweisungen ist der Benutzer in der Lage, Dateien zu erstellen und zu bearbeiten. Und viele Benutzer werden nie größere Probleme als eine Dateiverwaltung mit dBASE lösen wollen.

Aber zur Programmierung mit dBASE ist es nur ein kleiner Schritt weiter. Denn alle Befehle, die Sie im Laufe dieser Einführung lernen und im direkten Dialog mit dBASE einsetzen werden, können Sie auch im Rahmen von Programmen verwenden. Die restlichen Kommandos, die für die Bil-

dung von Programmschleifen und Verzweigungen benötigt werden, sind rasch dazugelernt. So eignen Sie sich im Umgang mit dBASE gleichzeitig eine Programmiersprache von hohem Niveau an.

Die Anwendungsbereiche von dBASE sind sehr weit gesteckt. Sie decken praktisch alle Gebiete der Datenverarbeitung auf Mikrocomputern ab. Es wäre daher unangebracht, einige klassische Anwendungen, etwa aus dem Finanz- oder Personalbereich, besonders hervorzuheben. Mit gleicher Berechtigung könnte man Beispiele aus dem Materialwesen oder aus dem Sport aufführen. Dagegen erscheint es sinnvoller, einige Aussagen über den physischen Rahmen von dBASE-Anwendungen zu machen, in dem sich der Benutzer bewegen kann oder sollte.

Rein theoretisch ist die Kapazität von dBASE II als Datenbank fast unbegrenzt. Es können beliebig viele Dateien angelegt und mit jeweils bis zu 65535 Datensätzen geladen werden. Bei maximal 1000 Zeichen pro Datensatz ergibt sich ein Kapazitätspotential, das einem größeren System als ein Mikrocomputer alle Ehre machte. Bild 1.4 zeigt die wichtigsten quantitativen Merkmale von dBASE II. Einige der Angaben werden Ihnen zunächst noch nicht viel sagen. Wir werden die Erklärungen aber zu gegebener Zeit nachholen.

In der Praxis wird der Umfang einer Datenbank im wesentlichen von zwei Kriterien begrenzt: von der Kapazität der Massenspeicher und von der Programmlaufzeit für die Bearbeitung der Datenbank.

Den Speicherbedarf einer Datei kann man grob aus dem Produkt der Anzahl von Datensätzen mal der Anzahl von Zeichen pro Datensatz errechnen. Bei einer Datei mit 1000 Sätzen und 100 Zeichen je Satz ist das etwa 100 Kilobyte. Aus dieser einfachen Rechnung sieht man schon, daß eine praktische Grenze durch die verfügbare Diskettenkapazität relativ früh erreicht wird. Das gilt insbesondere dann, wenn man nur mit einem Diskettenlaufwerk arbeitet und alle Dateien auf einer Diskette untergebracht werden müssen. Dann bleiben für die eigentlichen Datenbestände nur etwa 30 bis 40 Kilobyte übrig. Für mittlere bis größere Datenbankaufgaben sollte man daher ein zweites Diskettenlaufwerk anschließen, das die Datendateien aufnimmt.

Selbst wenn Sie vielleicht eines Tages eine Festplatte an Ihren Schneider-Computer anschließen können, wird das System dBASE II immer noch nicht das "schwächste Glied in der Kette" sein.

Daten-Dateien pro System	unbeschränkt
Programm-Dateien pro System	unbeschränkt
Datensätze pro Datei	65535
Satzlänge	1000 Zeichen (Byte)
Felder pro Satz	32
Feldtypen und Feldlänge	
Alphanumerische Felder	254 Zeichen
Numerische Felder	10 Zeichen
Logische Felder	1 Zeichen
Gleichzeitig eröffnete Dateien	
insgesamt	16
Daten-Dateien	2
Indexdateien pro Stammdatei	7
Speichervariable	64

Bild 1.4: Das Mengengerüst von dBASE II

Über die Verarbeitungszeiten kann man keine pauschalen Angaben machen. Sie hängen von verschiedenen Kriterien ab: zum Beispiel von der Größe der Dateien und von der Art des Datenzugriffs. Es gibt Befehle, die für einen sequentiellen Suchvorgang in einigen Tausend Datensätzen vielleicht Minuten benötigen, und es existiert die Möglichkeit, über sogenannte Indexschlüssel innerhalb von ein bis zwei Sekunden jeden beliebigen Datensatz auch in allergrößten Dateien zu finden.

Einer der langsamsten Vorgänge beim Ablauf eines Programms ist der Zugriff auf die Massenspeicher. Sicherlich werden Sie die Lesegeschwindigkeit von der Diskette als rasant im Vergleich zur Magnetbandkassette empfinden. Aber für den Z-80-Prozessor ist das immer noch sehr langsam. Daher geht auch die Anzahl der Zugriffe auf eine Diskette sehr stark in die Gesamtverarbeitungszeit ein.

Ähnliches wie für die Datenbank gilt auch für Anzahl und Umfang der dBASE-Programme. Es können beliebig viele Programme von beliebiger

Länge angelegt werden. Der integrierte Programmeditor kann zwar nur Programme bis 5000 Byte Umfang bearbeiten, aber das ist bei der Mächtigkeit dieser Sprache nicht nur ausreichend, sondern die Programme können auch mit anderen Texteditoren wie z.B. WordStar, die größere Texte zulassen, bearbeitet werden.

dBASE installieren

Die Installation von dBASE II ist derart problemlos, das es unnötig ist, ihr ein eigenes Kapitel zu widmen. dBASE II wird bereits komplett angepaßt an den Schneider CPC ausgeliefert, so daß der Anwender nur noch eine Kopie der Diskette für seine Arbeiten anfertigen muß und dann die Originaldiskette an einem sicheren Ort verwahren kann.

Vor dem erstmaligen Gebrauch von dBASE sollte man sich noch entscheiden, mit welchem Zeichensatz dBASE künftig arbeiten soll. Der Schneider-Computer bietet nämlich die - keinesfalls übliche - Möglichkeit, die Tastatur auf zwei verschiedene Arten zu belegen. Sie haben die Wahl zwischen den internationalen ASCII-Zeichen und dem deutschen DIN-Zeichensatz. Der deutsche Zeichensatz enthält die Umlaute, das "ß" und das "§"-Zeichen, während der ASCII-Standard statt dessen einige Sonderzeichen wie eckige und geschweifte Klammern vorsieht. Da die ASCII-Sonderzeichen mit einer Ausnahme von dBASE nicht gebraucht werden, andererseits aber die deutschen Umlaute in den Datenbeständen sicherlich häufig vorkommen, dürfte die DIN-Tastatur in der Regel für Sie zweckmäßiger sein.

Das Einstellen erfolgt über das mitgelieferte Hilfsprogramm CONFIG, das Ihnen die entsprechenden Fragen stellt. Diese Arbeit ist nur einmal zu erledigen, solange Sie die Tastaturbelegung nicht mehr ändern wollen.

Haben Sie einen Zeichensatz ausgewählt, dann dürfen Sie die Datei CONFIG.COM von Ihrer **Arbeitsdiskette** löschen (nicht von der Originaldiskette). Damit gewinnen Sie etwa 9 Kilobyte Speicherplatz für Ihre Daten.

2 Datenbankverwaltung mit einer Datei

Die üblichen Programmiersprachen wie BASIC, COBOL, PASCAL etc. sind nicht in der Lage, die speziellen Befehle zum Zugriff auf Informationen in Datenbanken zur Verfügung zu stellen, weil sie nicht die Besonderheiten jeder individuellen Datenbankarchitektur berücksichtigen können. Daher besitzt jedes Datenbanksystem seinen eigenen Befehlsvorrat, die sogenannte Datenmanipulationssprache, mit der die Anweisungen zur Bearbeitung der Datenbank formuliert werden.

Das gilt auch für dBASE II. Bevor Sie also mit dBASE arbeiten können, müssen Sie sich eine Reihe von Kommandos aneignen. Allerdings ist es im Gegensatz zu manch anderen Datenbanksystemen möglich, sämtliche Befehle einzeln und im direkten Dialog mit dBASE anzuwenden. Das bedeutet für Sie, daß Sie nicht erst einige Zeit mit theoretischem Lernen verbringen müssen, sondern von Anfang an alle Befehle in der Praxis erproben können. Die Vorstellung aller dBASE-Kommandos wird Gegenstand des ersten Kapitels sein. Später können Sie die einzelnen Befehle zu Programmen zusammenfassen, speichern und bei Bedarf abrufen und laufen lassen.

Mit den folgenden Beispielen werden nicht nur die wichtigsten Befehle erläutert, sondern auch einige Dateien aufgebaut, die später im Rahmen von Programmen wieder benutzt werden. Es ist daher sehr zu empfehlen, die Beispiele auf dem eigenen Computer nachzuvollziehen, während diese Anleitung gelesen wird. Sie sammeln dabei Erfahrung im Umgang mit dBASE und schaffen sich gleichzeitig eine kleine Datenbank für Ihre Übungen.

Schalten Sie nun Ihren Computer ein und laden Sie das Betriebssystem. Dazu ist die Diskette mit CP/M in das Diskettenlaufwerk zu stecken und nach der Eingabe von

|CPM

die ENTER-Taste zu betätigen. Sobald das System bereit ist, erscheint auf dem Bildschirm die Nachricht " A>". Wenn Sie dBASE auf derselben Diskette gespeichert haben wie das Betriebssystem, was der Normalfall ist, dann geben Sie jetzt das Wort

DBASE

ein und drücken die ENTER-Taste, um das Programm in den Hauptspeicher zu laden.

Sollten Sie dBASE auf einer separaten Diskette gespeichert haben, so müssen Sie nach Einlegen der dBASE-Diskette in das Floppy-Laufwerk zuerst die Tastenkombination CTRL-C drücken, um dem Computer den Diskettenwechsel mitzuteilen. Es erscheint dann noch einmal die Systemmeldung "A>", bevor Sie dBASE aufrufen können.

Das Laden von dBASE in den Hauptspeicher dauert einige Sekunden. Anschließend werden Sie aufgefordert, das heutige Datum einzugeben oder - wenn Sie das nicht wollen - einfach die ENTER-Taste zu drücken:

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

(Die ENTER-Taste wird vielfach auch RETURN-Taste genannt.) Das Datum ist gegebenenfalls im deutschen Format Tag/Monat/Jahr einzutippen. Ein ungültiges Datum, z.B. der 30.2.85 oder die Reihenfolge Monat/Tag/Jahr, wird von dBASE zurückgewiesen. Die Eingabe ist auf jeden Fall dann vorteilhaft, wenn Sie innerhalb eines Anwendungsprogramms das aktuelle Datum brauchen, zum Beispiel für Buchungsaufgaben. Dann kann das eingegebene Datum jedesmal automatisch von dBASE zugespielt werden.

Anmerkung: Sie werden später noch einen Befehl zur Datumseingabe kennenlernen, der auch ein anderes Format als Tag/Monat/Jahr zuläßt.

Nach Betätigen der ENTER-Taste (mit oder ohne vorheriger Datumseingabe) erscheint die Anzeige der Versions-Nummer von dBASE, darunter

mehrere Zeilen Information über das Copyright und - das wichtigste - in einer eigenen Zeile ein Punkt, hinter dem der Cursor steht:

*** dBASE II Ver 2.41 19 JUNE 1984

...Copyright...

. (Cursor)

Dieser einzelne Punkt (im dBASE-II-Handbuch als "PROMPT"-Punkt bezeichnet) bedeutet stets, daß dBASE bereit ist, einen Befehl zu empfangen, der unmittelbar hinter diesen Punkt geschrieben wird. Dieser Befehl kann eine Anweisung, eine Frage oder auch der Aufruf eines kompletten Programms sein.

Zum Beispiel können Sie jetzt nach dem zuvor eingegebenen Datum fragen und feststellen, in welchem Format es gegebenenfalls vom Betriebssystem übernommen wurde. Tippen Sie

? Date()

und drücken Sie die ENTER-Taste.

Als ersten Befehl sollten Sie sich das Wort QUIT (= beenden) merken. Mit QUIT können Sie das Programm dBASE beenden und zum Betriebssystem zurückkehren. Auf den Befehl QUIT hin erscheint die dBASE-Ende-Nachricht:

*** Ende des dBASE II-Laufes ***

Denken Sie an Ihre Datensicherung

und dann die Meldung des Betriebssystems " A> ".

Für QUIT und sämliche anderen Befehle von dBASE ist es gleichgültig, ob Sie diese mit Groß- oder Kleinbuchstaben schreiben. In der Praxis, und da insbesondere in Programmen, ist es jedoch zweckmäßig, Befehlsworte groß und Namen von Dateien usw. klein zu schreiben, um sie optisch besser voneinander zu unterscheiden. Wie Sie noch an den Beispielen sehen werden, erleichtert diese Schreibweise die Lesbarkeit von Programmen wesentlich.

2.1 Erstellen einer Datei

Grundsätzlich sind alle Informationen, die sich nach gewissen Kriterien ordnen lassen, geeignet, in einer Datenbank erfaßt und verarbeitet zu werden.

Ein einfaches, aber jedem geläufiges Beispiel hierfür ist ein Adressenverzeichnis. Jede Eintragung, d.h., jeder Datensatz besteht aus einer Anzahl gleichartiger Elemente, den Datenfeldern, wie z.B. Name, Anschrift und Telefonnummer.

Ein solches Adressenverzeichnis soll nun aufgebaut und benutzt werden. Die Eintragungen werden auf eine relativ kleine Anzahl beschränkt, da nur das Prinzip der Datenbank aufgezeigt werden soll. Die Vorgehensweise wäre exakt die gleiche, wenn das Verzeichnis einige tausend Eintragungen enthielte.

2.1.1 Die Dateistruktur

Wenn Sie eine neue Datei anlegen wollen, dann müssen Sie stets zuerst ihre Struktur, gewissermaßen das Gerüst, in das später die Daten geladen werden, definieren. Sie müssen festlegen, welche Arten von Daten die Datei enthalten und wieviel Platz für das einzelne Datenelement reserviert werden soll. Bei einem relationalen Datenbanksystem wie dBASE läßt sich eine Datei als Tabelle darstellen, deren Zeilen aus jeweils einer vollständigen Eintragung (Name, Anschrift usw.) bestehen und deren Spalten jeweils gleichartige Datenelemente (z.B. alle Vornamen in einer, alle Telefonnummern in einer anderen Spalte) enthalten. In Bild 1.1 wurde bereits eine derartige Tabelle gezeigt:

NAME	VORNAME	STRASSE	ORT	TELEFON
Maier	Herbert	Amalienstr.59	München	089-912345
Huber	Josef	Sieglindenstr.5	München	089-987654
Kuhn	Peter	Hauptstr.12	Neuss	02101-1234
Meister	Werner	Marktplatz 3$	Eching	089-753267
Schmid	Herbert	Fasanenstr.2	Berlin	030-135790

Da alle Zeilen (d.h. alle Datensätze) einer Datei gleich strukturiert sind, genügt es, für die Beschreibung der Datei nur die Struktur eines Daten-

satzes zu definieren. Die Anzahl der Datensätze (Zeilen der Tabelle), die aufgenommen werden sollen, ist für die Struktur-Definition unwichtig.

Stellen Sie nun sicher, daß Ihr System bereit ist, dBASE-Befehle zu empfangen, was durch Anzeige des Punktes sichtbar wird.

Für das Generieren einer neuen dBASE-Datei existiert der Befehl

CREATE (Dateiname)

(CREATE = erzeugen, schaffen). Für die neue Datei ist ein Dateiname festzulegen, der bis zu acht Zeichen lang sein darf (entsprechend den Regeln des Betriebssystems). Unter diesem Namen wird die Datei abgespeichert und zur Bearbeitung aufgerufen.

Schreiben Sie CREATE und dahinter das Wort **adressen** als Dateiname:

CREATE adressen

Vor dem Dateinamen können Sie noch das Laufwerk angeben, wenn die Datei auf einem anderen als dem dBASE-System-Laufwerk gespeichert werden soll. Wir gehen hier und im Folgenden davon aus, daß sich alle Dateien zusammen mit dem dBASE-Programm auf einem Laufwerk befinden.

Mit dem Befehl CREATE wird dBASE veranlaßt, ein Hilfsprogramm aufzurufen, das von Ihnen einige Parameter erfragt, wie die Struktur der Datensätze beschaffen sein soll:

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

Für jedes Element (Datenfeld) eines Satzes ist eine Zeile vorgesehen, in die einige Angaben einzutragen sind. Die Felder sind numeriert und werden nacheinander aufgerufen, wobei maximal 32 Felder definiert werden können.

Die erforderlichen Angaben sind:

Name Name, Bezeichnung des Datenfeldes

Feldnamen dürfen bis zu 10 Zeichen lang sein. Sie müssen stets mit einem Buchstaben beginnen, können danach aber auch Ziffern oder einen Doppelpunkt enthalten. Zwischenräume sind nicht erlaubt.

Typ Typ, Art des Datenfeldes

Hier stehen drei Arten zur Auswahl:

C für alphanumerische Felder. Dies ist der bei weitem häufigste Feldtyp.

N für numerische Felder, die arithmetischen Operationen unterworfen werden sollen

L für logische Felder

Länge Feldlänge

Hier geben Sie an, wieviele Zeichen maximal in diesem Feld auftreten dürfen. Falls Sie bei numerischen Feldern Dezimalstellen benötigen, so müssen Sie diese plus eine Stelle für den Dezimalpunkt berücksichtigen.

Beispiel: Wenn maximal die Zahl 9999.99 in einem Feld unterzubringen ist, dann muß **Länge** mit 7 (= 4 Stellen vor, 2 Stellen nach dem Dezimalpunkt plus der Dezimalpunkt selbst) angegeben werden.

Dezimalstellen Anzahl der Dezimalstellen nach dem Komma (bzw. nach dem Dezimalpunkt).

Diese Angabe ist nur dann erforderlich, wenn Sie tatsächlich Stellen nach dem Komma benötigen.

Nach jeder Angabe ist ein Komma zu setzen und am Ende einer Zeile die ENTER-Taste zu drücken.

Vorweg eine Anmerkung zu eventuellen Eingabefehlern: Innerhalb der Zeile, in der sich der Cursor befindet, können Fehler durch Zurücksetzen des Cursors und Überschreiben sofort behoben werden.

Haben Sie die fehlerhafte Zeile bereits verlassen, so gibt es zwei Möglichkeiten: Entweder erkennt dBASE den Fehler (unerlaubte Eingabe), dann wird Ihre Eingabe zurückgewiesen und das betreffende Feld erneut zur Definition aufgerufen. Oder dBASE erkennt den Fehler nicht (erlaubte, aber von Ihnen ungewollte Eingabe), dann führen Sie die Definition der Dateistruktur zu Ende und korrigieren den Fehler später. In diesem Fall müßten Sie etwas vorgreifen und den Befehl MODIFY STRUCTURE anwenden (siehe Kapitel 2.3).

Definieren Sie nun die Adressendatei entsprechend dem Bild 2.1. Hierbei ist zu beachten, daß in Feldnamen kein "ß" und keine Umlaute vorkommen dürfen, weil diese intern vom Rechner wie Sonderzeichen behandelt werden, die für Feldnamen nicht zulässig sind (zum Beispiel besitzen die Umlaute "Ä" und "Ü" die gleichen internen Codes wie die eckigen Klammern "[" und "]" im ASCII-Zeichensatz).

Ein Datensatz der Datei **adressen** besteht also aus fünf Datenfeldern, die entsprechend ihrem späteren Inhalt mit **name**, **vorname**, **strasse**, **ort** und **telefon** benannt sind. Alle Felder sind vom Typ "C", also alphanumerisch. Damit entfällt die Angabe von irgendwelchen Dezimalstellen. Beachten Sie, daß auch die Telefonnummern als alphanumerisch und nicht als numerisch definiert wurden. Der Grund ist, weil sie z.B. zur besseren Lesbarkeit auch Bindestriche enthalten sollen und weil sie nicht arithmetisch zu verarbeiten sind, d.h., sie müssen nicht addiert oder subtrahiert etc. werden.

Die Anzahl und Länge der Felder wurde so gewählt, daß sie aneinandergereiht noch bequem auf dem Bildschirm Platz finden, was für die folgenden Beispiele zweckmäßig ist. Tatsächlich erlaubt dBASE jedoch bis zu 32 Felder pro Datensatz, die zusammen bis zu 1000 Zeichen lang sein dürfen.

```
Satzstruktur folgendermaßen eingeben
 Feld    Name, Typ, Länge, Dezimalstellen
 001     name,c,10
 002     vorname,c,10
 003     strasse,c,15
 004     ort,c,16
 005     telefon,c,11
 006
```

Bild 2.1: Strukturdefinition der Datei **adressen**

Die Definition der Datenstruktur schließen Sie ab, indem Sie zu Beginn des nächsten, in diesem Fall sechsten Feldes sofort ENTER drücken. Damit erkennt dBASE, daß Sie nur fünf Felder innerhalb dieser Datei brauchen, beendet den Aufbau der Struktur und geht zur nächsten Frage über:

Daten jetzt eingeben?

Falls Sie zunächst Eingabefehler zu korrigieren haben, antworten Sie mit "N" und geben den Befehl "MODIFY STRUCTURE" wie in Kapitel 2.3 beschrieben.

2.1.2 Daten erfassen

Andernfalls beantworten Sie diese Frage mit "J" (Ja). Dann wird der Bildschirm gelöscht und Sie können sofort beginnen, die neu geschaffene Datenbank mit Informationen zu laden. Dazu wird eine sogenannte Maske zur Datenerfassung angezeigt, die die Struktur des eben definierten Datensatzes wiederspiegelt (Bild 2.2).

```
Satznummer 00001
NAME       :          :
VORNAME    :          :
STRASSE    :                :
ORT        :                 :
TELEFON    :           :
```

Bild 2.2: Maske zur Datenerfassung

Die erste Zeile dieser Maske zeigt die laufende Nummer des Datensatzes (Satznummer) an, die vom System automatisch vergeben wird. Da eine neue Datei begonnen wird, ist dies hier die Nummer 1. Ist der erste Satz geschrieben, wird die Nummer 2 aufgerufen usw. Insgesamt können theoretisch bis zu 65535 Datensätze pro Datei gespeichert werden, praktisch wird aber die verfügbare Diskettenkapazität ein vorzeitiges Limit setzen.

In jeder Zeile der Maske steht ein Datenfeld. Hinter seinem Namen ist zwischen Doppelpunkten der Raum markiert, in den der Feldinhalt geschrieben werden kann. Die Größe dieses Raumes entspricht der zuvor definierten Feldlänge.

Geben Sie nun den ersten Datensatz ein (Bild 2.3).

```
Satznummer 00001
NAME       :Maier      :
VORNAME    :Herbert    :
STRASSE    :Amalienstr.59  :
ORT        :8000 München     :
TELEFON    :089-912345 :
```

Bild 2.3: Eingabe eines Datensatzes

Solange Sie innerhalb eines Datensatzes bleiben, können Sie den Cursor mit Hilfe der Cursor- oder Pfeil-Tasten (bzw. mit den Tastenkombinationen CTRL-E, CTRL-X, CTRL-S, CTRL-D) in der Maske beliebig hin und her bewegen. Damit können Sie auch zu vorhergehenden Feldern zurückkehren, um Korrekturen oder Ergänzungen vorzunehmen.

Im Unterschied zur Eingabe von Befehlen, Datei- und Feldnamen ist bei den Daten selbst auf Groß- und Kleinschreibung zu achten. Bei späteren Auswertungen der Datenbank, bei Suchläufen oder Vergleichen wird nämlich von dBASE auch die Groß-/Kleinschreibung als Unterscheidungsmerkmal herangezogen (falls nichts Gegenteiliges erwünscht ist). Dies ist zwar eine sehr nützliche Leistung von dBASE, erfordert aber eine gewisse Ordnung und Konsequenz bei der Datenerfassung.

Die Eingabe eines Datensatzes wird beendet, indem Sie vom letzten Feld aus (in diesem Beispiel: TELEFON) die ENTER-Taste betätigen.

Anschließend erscheint die Erfassungsmaske für den nächsten Datensatz, Satznummer 00002.

Geben Sie nun entsprechend die folgenden Datensätze ein:

```
Huber     Josef     Sieglindenstr.5  8000 München   089-987654
Kuhn      Peter     Hauptstr.12      4040 Neuss     02101-1234
Meister   Werner    Marktplatz 3     8057 Eching    089-753267
Schmid    Herbert   Fasanenstr.2     1000 Berlin    030-135790
Hassold   Paul      Meisenstr.18     2000 Hamburg   040-246875
Hofmann   Gerd      Holzweg 13       2000 Hamburg   040-456987
```

```
Maier     Josef    Flurstr.3          8000 München    089-321654
Fischer   Franz    Ginsterweg 9       6850 Mannheim   0621-96385
Hoffman   Peter    Alte Gasse 15      8630 Coburg     09561-7412
Knoll     Klaus    Uferweg 8          5000 Köln       0221-963258
```

Mit der letzten Adresse haben Sie insgesamt elf Datensätze in der Datenbank **adressen** gespeichert.

Beenden Sie nun vorerst die Erfassung, indem Sie bei Erscheinen der Maske für Satznummer 00012 sofort die ENTER-Taste drücken.

Damit erkennt dBASE, daß die Aufnahme von Daten beendet ist, und kehrt zurück in die Bereitschaft, neue Befehle zu empfangen, was wieder durch Erscheinen des Punktes angezeigt wird.

Leser mit Programmier-Erfahrung werden sich vielleicht jetzt fragen, ob bzw. wie die erfaßten Daten nun auf der Diskette zu speichern sind. Die Antwort ist einfach: Sie sind es bereits! Das System dBASE sichert von sich aus in sehr kurzen Abständen sämtliche erfaßten Daten auf der Diskette. Der Benutzer braucht sich zu keinem Zeitpunkt hierüber Gedanken zu machen. Sie könnten ohne weiteres jetzt den Befehl QUIT geben und Ihren Computer abschalten, ohne die aufgenommenen Adressen zu verlieren.

2.1.3 Protokollieren der Eingabe

Nachdem Sie nun elf Adressen in der Datenbank gespeichert haben, ist es sicherlich zweckmäßig, eine Liste, d.h., ein Protokoll Ihrer Eingaben anzufertigen. Dazu werden zwei weitere Befehle benötigt. Zunächst muß die Datei, deren Inhalt gelesen werden soll, aufgerufen oder "eröffnet" werden. Dies geschieht mit dem Befehl

USE (Dateiname)

(USE = benutzen, gebrauchen). Schreiben Sie also:

USE adressen

und drücken Sie die ENTER-Taste.

Damit wird die Datei **adressen** zur Benutzung freigegeben. Sie bleibt solange im Zugriff, bis ein erneuter USE-Befehl eine andere Datei eröffnet und diese dann zur Bearbeitung verfügbar ist.

Um die vollständige Datei auszugeben, schreiben Sie den Befehl

LIST

(LIST = auflisten) und drücken wieder die ENTER-Taste.

Anmerkung: Die ENTER-Taste ist grundsätzlich nach jedem Befehl zu betätigen. Deshalb soll von jetzt an auf den ausdrücklichen Hinweis, diese Taste zu drücken, verzichtet werden.

Mit dem Befehl LIST werden alle Datensätze gelesen und auf den Bildschirm gebracht, wobei ein Datensatz einer Zeile entspricht und die Felder in der Reihenfolge nebeneinander erscheinen, wie sie in der Datenbankstruktur festgelegt sind (Bild 2.4). Zu Beginn eines jeden Satzes steht die vom System vergebene laufende Satznummer.

```
. USE adressen
. LIST

00001 Maier     Herbert   Amalienstr.59     8000 München    089-912345
00002 Huber     Josef     Sieglindenstr.5   8000 München    089-987654
00003 Kuhn      Peter     Hauptstr.12       4040 Neuss      02101-1234
00004 Meister   Werner    Marktplatz 3      8057 Eching     089-753267
00005 Schmid    Herbert   Fasanenstr.2      1000 Berlin     030-135790
00006 Hassold   Paul      Meisenstr.18      2000 Hamburg    040-246875
00007 Hofmann   Gerd      Holzweg 13        2000 Hamburg    040-456987
00008 Maier     Josef     Flurstr.3         8000 München    089-321654
00009 Fischer   Franz     Ginsterweg 9      6850 Mannheim   0621-96385
00010 Hoffman   Peter     Alte Gasse 15     8630 Coburg     09561-7412
00011 Knoll     Klaus     Uferweg 8         5000 Köln       0221-963258
```

Bild 2.4: Die Datei **adressen**

Sie können die Datei auch auf einen Drucker ausgeben, wenn Sie die Tasten-Kombination CTRL-P drücken und dann erneut den Befehl LIST schreiben. Durch nochmaliges Betätigen von CTRL-P wird der Drucker wieder abgeschaltet.

Anmerkung: Bei der dBASE-II-Version für den Schneider CPC, die von Markt&Technik ausgeliefert wird, ist eine Reihe von Funktionstasten bereits mit wichtigen dBASE-Kommandos belegt. So kann der Drucker z.B. auch mit der Tastenkombination CTRL-COPY ein- und ausgeschaltet werden. Wo es angebracht ist, werden wir künftig auf die Funktionstasten hinweisen. Die gesamte Belegung finden Sie zusammengefaßt in Ihrer dBASE-Dokumentation zum Schneider CPC.

Solange Sie sich in dBASE einarbeiten, ist es durchaus empfehlenswert, den Drucker eingeschaltet zu lassen. Sie können dann neben den Dateiinhalten auch alle eingegebenen Befehle bzw. die Reaktionen von dBASE inklusive etwaiger Fehlermeldungen protokollieren lassen.

Der Befehl LIST kann auch verwendet werden, um die Struktur der Datei **adressen** aufzuzeigen. Dazu fügen Sie einfach den Zusatz STRUCTURE an, also:

LIST STRUCTURE

```
.LIST STRUCTURE

Strukturdaten für Datei: A:ADRESSEN.DBF
Anzahl der Sätze:   00011
Datum der letzten Aktualisierung: 00/00/00
Primäre Datei
Feld    Name        Typ   Länge    Dez.st.
001     NAME         C    010
002     VORNAME      C    010
003     STRASSE      C    015
004     ORT          C    016
005     TELEFON      C    011
** Gesamt **           00063
```

Bild 2.5: Die Struktur der Datei **adressen**

Die erste Zeile der Strukturbeschreibung (Bild 2.5) nennt den Dateinamen, dem ein Punkt und dahinter eine Angabe über den Dateityp folgt:

Strukturdaten für Datei: A:ADRESSEN.DBF

Der Dateityp **DBF** (Data Base File = Datenbank-Datei) wird von dBASE automatisch vergeben, wenn mit dem Befehl CREATE eine neue Datei geschaffen wird. Solange Sie innerhalb dBASE arbeiten, brauchen Sie sich aber um den Dateityp nicht zu kümmern. Die zweite Zeile der Dateistruktur sagt Ihnen, wieviele Datensätze augenblicklich in der Datei **adressen** gespeichert sind, nämlich elf:

Anzahl der Sätze: 00011

Danach folgt das Datum der letzten Aktualisierung der Datei, was sich sowohl auf deren Inhalt als auch auf die Dateistruktur beziehen kann:

Datum der letzten Aktualisierung: 00/00/00

Die nächste Mitteilung "Primäre Datei" ist nur interessant im Zusammenhang mit der gleichzeitigen Benutzung mehrerer Dateien und soll vorläufig übergangen werden.

Anschließend folgt eine Auflistung der einzelnen Felder, die für diese Datei definiert wurden, und in der letzten Zeile eine Summe (** Gesamt **), die die Satzlänge als Zeichenanzahl angibt. Tatsächlich ist diese Summe um ein Zeichen größer als die Summe aller Felder, da eine zusätzliche Position für eine Löschmarkierung reserviert ist (siehe auch Bild 2.11).

Die Dateistruktur sollten Sie sich stets ausdrucken lassen. Wenn Sie nämlich später mit der Datenbank arbeiten, müssen Sie die getroffenen Dateivereinbarungen konsequent einhalten bzw. beachten. Insbesondere müssen alle Namen (Datei- und Feldnamen) immer wieder exakt so geschrieben werden, wie in der Datenbankstruktur festgelegt. Auch sind für alphanumerische bzw. numerische Felder teilweise verschiedene Regeln zu beachten. Es ist daher sehr nützlich, wenn Sie im Zweifelsfall auf eine ausgedruckte Dateistruktur als Referenz zurückgreifen können.

2.1.4 Anfügen weiterer Datensätze

Mit dem Befehl

APPEND

(= anhängen) können neue Datensätze an eine vorhandene Datei angefügt werden. Im einfachsten Fall werden die neuen Sätze am Ende der Datei angehängt. Falls Sie die Datei **adressen** nicht schon eröffnet haben, geben Sie zuerst den Befehl

USE adressen

und dann den Befehl

APPEND

der veranlaßt, daß die Eingabemaske für die Datei **adressen** wieder aufgerufen wird, wobei jetzt angezeigt wird, daß als nächster Satz die Nummer 00012 zu erfassen ist (Bild 2.6).

```
Satznummer 00012
NAME      :          :
VORNAME   :          :
STRASSE   :                    :
ORT       :                     :
TELEFON   :           :
```

Bild 2.6: Anfügen eines neuen Datensatzes

Erweitern Sie nun die Datei um die folgenden vier Adressen:

```
Koch      Walter     Heumarkt 14     6000 Frankfurt   0611-753681
Becker    Jakob      Hauptstr.28     6100 Darmstadt   06151-45632
Hofer     Andreas    Ludwigstr. 5    8201 Raubling    08035-1111
Behr      Max        Frankenweg 17   8500 Nürnberg     0911-429681
```

Damit enthält das Adressenverzeichnis nun insgesamt 15 Sätze (Bild 2.7).

00001	Maier	Herbert	Amalienstr.59	8000 München	089-912345
00002	Huber	Josef	Sieglindenstr.5	8000 München	089-987654
00003	Kuhn	Peter	Hauptstr.12	4040 Neuss	02101-1234
00004	Meister	Werner	Marktplatz 3	8057 Eching	089-753267
00005	Schmid	Herbert	Fasanenstr.2	1000 Berlin	030-135790
00006	Hassold	Paul	Meisenstr.18	2000 Hamburg	040-246875
00007	Hofmann	Gerd	Holzweg 13	2000 Hamburg	040-456987
00008	Maier	Josef	Flurstr.3	8000 München	089-321654
00009	Fischer	Franz	Ginsterweg 9	6850 Mannheim	0621-96385
00010	Hoffman	Peter	Alte Gasse 15	8630 Coburg	09561-7412
00011	Knoll	Klaus	Uferweg 8	5000 Köln	0221-963258
00012	Koch	Walter	Heumarkt 14	6000 Frankfurt	0611-753681
00013	Becker	Jakob	Hauptstr.28	6100 Darmstadt	06151-45632
00014	Hofer	Andreas	Ludwigstr. 5	8201 Raubling	08035-1111
00015	Behr	Max	Frankenweg 17	8500 Nürnberg	0911-429681

Bild 2.7: Das erweiterte Adressenverzeichnis

Ein weiterer Befehl, einen neuen Datensatz in eine Datei aufzunehmen, ist

INSERT

(= einfügen), der den neuen Satz an einer beliebigen Position in der Datei einfügt, zum Beispiel um eine bestimmte Reihenfolge der Datensätze zu erreichen.

dBASE verfügt aber über wesentlich mächtigere Befehle, Daten zu ordnen oder geordnet einzugeben, die später noch besprochen werden. Daher soll auf den Befehl INSERT nicht näher eingegangen werden.

2.1.5 Ändern von Datensätzen

Neben dem Erfassen und Speichern von Daten ist es natürlich wichtig, diese zu irgendeinem Zeitpunkt ändern zu können. Angenommen, Personen aus dem Adressenverzeichnis wechseln ihren Wohnsitz, ändern infolge Heirat ihren Namen oder erhalten neue Telefonnummern, dann müssen diese Veränderungen in der Adressendatei nachvollzogen werden.

Als Beispiel sollen zwei Anschriften aktualisiert werden, und zwar zunächst die des Peter Kuhn, Datensatz 3. Stellen Sie sicher, daß die Datei **adressen** eröffnet ist und schreiben Sie den Befehl

EDIT

(= bearbeiten), worauf die Systemnachricht

Bitte Satznummer eingeben:

erscheint, die Sie auffordert, die Nummer des gewünschten Datensatzes einzugeben. Dies ist die Zahl 3.

Daraufhin erscheint wieder die von der Datenerfassung bekannte Maske, die aber jetzt die Daten des dritten Satzes enthält (Bild 2.8). Sie können nun alle Daten dieses Satzes beliebig oft und in beliebiger Reihenfolge überschreiben, wobei mit Hilfe der Cursortasten die gewünschten Felder bzw. Zeichenpositionen angesteuert werden.

Ändern Sie nun die Anschrift so ab, wie in Bild 2.9 wiedergegeben.

```
Satznummer 00003
NAME      :Kuhn      :
VORNAME   :Peter     :
STRASSE   :Hauptstr.12    :
ORT       :4040 Neuss       :
TELEFON   :02101-1234 :
```

Bild 2.8: Datensatz # 00003 in seiner Editiermaske

```
Satznummer  0003
NAME      :Kuhn      :
VORNAME   :Peter     :
STRASSE   :Fliederstr.9   :
ORT       :6100 Darmstadt  :
TELEFON   :06151-4321 :
```

Bild 2.9: Geänderte Anschrift

Die Bearbeitung des Datensatzes wird abgeschlossen, indem die Tastenkombination CTRL-W betätigt wird. Daraufhin wird der geänderte Satz gespeichert und erneut die Systemnachricht

Bitte Satznummer eingeben:

gebracht, worauf Sie den nächsten Satz editieren können.

Wenn Sie mehrere, dicht beieinanderliegende Datensätze editieren wollen, dann können Sie dies auch ohne erneuten EDIT-Befehl tun. Ein ENTER (oder Cursor-nach-unten-Taste) vom letzten Datenfeld aus bewirkt, daß der nächste Satz aufgerufen wird. Entsprechend kann mit Hilfe der nach oben zeigenden Cursor-Taste der vorhergehende Datensatz angesprochen werden. Beide Cursor-Tasten können in Kombination mit der SHIFT-Taste benutzt werden, um von jeder beliebigen Position aus zum nächsten Datensatz vor- oder zurückzublättern. Dies sind zwei der nützlichen Tastenbelegungen, die für Sie auf dem Schneider CPC bereits installiert sind.

Nehmen Sie nun eine weitere Datenänderung vor, und zwar an Datensatz # 00011, indem Sie die Anschrift ändern in

STRASSE : Amselweg 23
ORT : 1000 Berlin
TELEFON : 030-246325

Mit CTRL-W wird die EDIT-Funktion wieder abgeschlossen. Wenn Sie nun anstatt eine neue Satznummer einzugeben nur die ENTER-Taste drücken, dann wird der Editor verlassen und dBASE kehrt in den normalen Betriebszustand zurück.

Wenn Sie nur einen einzigen Datensatz editieren wollen, dann können Sie auch mit dem Kommando EDIT direkt die Satznummer angeben, zum Beispiel:

EDIT 3

Nach Bearbeitung dieses einen Satzes wird dann die EDIT-Funktion sofort wieder verlassen, ohne daß dBASE nach einer weiteren Satznummer fragt.

Die geänderte Adressendatei, die für spätere Beispiele noch gebraucht wird, ist in Bild 2.10 wiedergegeben.

```
00001 Maier     Herbert  Amalienstr.59     8000 München    089-912345
00002 Huber     Josef    Sieglindenstr.5   8000 München    089-987654
00003 Kuhn      Peter    Fliederstr.9      6100 Darmstadt  06151-4321
00004 Meister   Werner   Marktplatz 3      8057 Eching     089-753267
00005 Schmid    Herbert  Fasanenstr.2      1000 Berlin     030-135790
00006 Hassold   Paul     Meisenstr.18      2000 Hamburg    040-246875
00007 Hofmann   Gerd     Holzweg 13        2000 Hamburg    040-456987
00008 Maier     Josef    Flurstr.3         8000 München    089-321654
00009 Fischer   Franz    Ginsterweg 9      6850 Mannheim   0621-96385
00010 Hoffman   Peter    Alte Gasse 15     8630 Coburg     09561-7412
00011 Knoll     Klaus    Amselweg 23       1000 Berlin     030-246325
00012 Koch      Walter   Heumarkt 14       6000 Frankfurt  0611-753681
00013 Becker    Jakob    Hauptstr.28       6100 Darmstadt  06151-45632
00014 Hofer     Andreas  Ludwigstr. 5      8201 Raubling   08035-1111
00015 Behr      Max      Frankenweg 17     8500 Nürnberg   0911-429681
```

Bild 2.10: Die geänderte Adressendatei

Innerhalb der Funktion EDIT gibt es eine Reihe von Tastenfunktionen, die die Bearbeitung von Daten ähnlich wie in einem Textverarbeitungssystem zulassen. Sie werden jeweils durch Kombination der CTRL-Taste mit einer Buchstaben-Taste ausgelöst. Die wichtigsten dieser Funktionen sind:

CTRL-W — schließt die Funktion EDIT ab.

SHIFT-↓ — oder CTRL-C schreibt den (geänderten) Satz auf die Diskette und ruft den nächsten Satz zur Bearbeitung auf.

SHIFT-↑ — oder CTRL-R schreibt den (geänderten) Satz auf die Diskette und ruft den vorhergehenden Satz zur Bearbeitung auf.

CTRL-Q — anulliert die Änderungen im augenblicklich aufgerufenen Satz und verläßt die EDIT-Funktion.

CTRL-G — löscht das Zeichen, über dem sich gerade der Cursor befindet.

CTRL-V — schaltet um zwischen Überschreiben und Einfügen von Zeichen. Wenn "Einfügen" gewählt ist, wird dies am oberen Rand des Bildschirms angezeigt (EINFÜGEN).

CTRL-U markiert den aufgerufenen Datensatz zur späteren Löschung. Es erscheint dann am oberen Rand der Satzmaske das Wort GELÖSCHT. Durch nochmaliges Drücken von CTRL-U kann die Löschung rückgängig gemacht werden.

Das Löschen von Datensätzen ist noch etwas näher zu erläutern:

Die mit CTRL-U markierten Datensätze werden zunächst physisch noch nicht gelöscht, sondern bleiben in der Datei erhalten. Sie sind bei Auflisten der Datei am Satzanfang mit einem Stern (*) gekennzeichnet. Bild 2.11 zeigt die Adressendatei für den Fall, daß die Sätze # 2, 10, 13 und 14 zur Löschung markiert sind. (Der Stern benötigt übrigens die eine zusätzliche Stelle im Datensatz, von der bereits kurz die Rede war.)

```
00001 Maier      Herbert  Amalienstr.59     8000 München     089-912345
00002*Huber      Josef    Sieglindenstr.5   8000 München     089-987654
00003 Kuhn       Peter    Fliederstr.9      6100 Darmstadt   06151-4321
00004 Meister    Werner   Marktplatz 3      8057 Eching      089-753267
00005 Schmid     Herbert  Fasanenstr.2      1000 Berlin      030-135790
00006 Hassold    Paul     Meisenstr.18      2000 Hamburg     040-246875
00007 Hofmann    Gerd     Holzweg 13        2000 Hamburg     040-456987
00008 Maier      Josef    Flurstr.3         8000 München     089-321654
00009 Fischer    Franz    Ginsterweg 9      6850 Mannheim    0621-96385
00010*Hoffman    Peter    Alte Gasse 15     8630 Coburg      09561-7412
00011 Knoll      Klaus    Amselweg 23       1000 Berlin      030-246325
00012 Koch       Walter   Heumarkt 14       6000 Frankfurt   0611-753681
00013*Becker     Jakob    Hauptstr.28       6100 Darmstadt   06151-45632
00014*Hofer      Andreas  Ludwigstr. 5      8201 Raubling    08035-1111
00015 Behr       Max      Frankenweg 17     8500 Nürnberg     0911-429681
```

Bild 2.11: Datei mit den vier zur Löschung markierten Sätzen Nr.2, 10, 13 und 14

Da alle Sätze nach wie vor vorhanden sind, kann die Löschung auch ohne weiteres wieder rückgängig gemacht werden. Physisch gelöscht werden die Datensätze erst, wenn der Befehl

PACK

gegeben wird (Bild 2.12), was Sie aber jetzt nicht machen sollten, da alle 15 Datensätze später noch benötigt werden. Anschließend werden die verbliebenen Datensätze mit neuen laufenden Nummern versehen, d.h. die durchgehende Numerierung wird wieder hergestellt.

```
00001 Maier     Herbert  Amalienstr.59  8000 München    089-912345
00002 Kuhn      Peter    Fliederstr.9   6100 Darmstadt  06151-4321
00003 Meister   Werner   Marktplatz 3   8057 Eching     089-753267
00004 Schmid    Herbert  Fasanenstr.2   1000 Berlin     030-135790
00005 Hassold   Paul     Meisenstr.18   2000 Hamburg    040-246875
00006 Hofmann   Gerd     Holzweg 13     2000 Hamburg    040-456987
00007 Maier     Josef    Flurstr.3      8000 München    089-321654
00008 Fischer   Franz    Ginsterweg 9   6850 Mannheim   0621-96385
00009 Knoll     Klaus    Amselweg 23    1000 Berlin     030-246325
00010 Koch      Walter   Heumarkt 14    6000 Frankfurt  0611-753681
00011 Behr      Max      Frankenweg 17  8500 Nürnberg   0911-429681
```

Bild 2.12: Die vier markierten Sätze aus Bild 2.11 sind hier physisch gelöscht

Abschließend noch eine Bemerkung zur Arbeit mit zwei oder mehr Diskettenlaufwerken:

In allen bisherigen und künftigen Beispielen wird - wie bereits erwähnt - nur ein Diskettenlaufwerk angesprochen. Wenn Sie zwei oder noch mehr Laufwerke an Ihren Schneider-Computer angeschlossen haben, können Sie auch diese für dBASE-Dateien benutzen, wenn Sie vor den Dateinamen den Kennbuchstaben des Laufwerks mit einem Doppelpunkt setzen, zum Beispiel B:adressen für Laufwerk B. Dies gilt für alle Arten von Dateien, auch für die später zu besprechenden Programmdateien.

Wenn Sie mit mehreren Laufwerken arbeiten, ist es sehr wichtig, daß Sie auf die Angabe des jeweils richtigen Laufwerks konsequent achten. Andernfalls könnte eine nicht gefundene Datei zum Abbruch Ihres Programms führen.

2.2 Gezielter Zugriff auf Daten

Mit den im vorausgegangenen Kapitel besprochenen Befehlen sind Sie in der Lage, eine Datenbank aufzubauen und Daten zu erfassen, zu ändern oder zu löschen. Damit können Sie Ihre Datenbestände bereits in jeder Hinsicht pflegen, selbstverständlich noch ohne besonderen Komfort. Bevor aber weitere, verfeinerte Möglichkeiten der Datenbankpflege vorgeführt werden, soll zunächst gezeigt werden, wie man gezielt an diejenigen Informationen in der Datenbank gelangt, die in der jeweiligen Situation von Interesse sind.

2.2.1 Einzelne Felder einer Datei auflisten

Der Befehl LIST, den Sie bereits zum Auflisten einer kompletten Datei benutzten, ist in der Lage, in etwas modifizierter Form einzelne Teile der Datei, bestimmte Datensätze oder auch bestimmte Felder aus der Datei auszuwählen und anzuzeigen.

Angenommen, Sie möchten sich nur die Vornamen, Namen und Telefonnummern der Personen in Ihrem Adressenverzeichnis ausdrucken lassen, dann können Sie den Befehl LIST variieren, indem Sie die gewünschten Feldnamen hinzufügen. Zum Beispiel:

```
USE adressen    (falls nicht schon eröffnet)

LIST vorname,name,telefon
```

Beachten Sie dabei, daß Sie die Feldnamen exakt so schreiben, wie sie in der Dateistruktur festgelegt sind. Als Ergebnis erhalten Sie dann jeweils der Reihe nach den Vornamen, den Namen und dann die Telefonnummer angezeigt (Bild 2.13). Alle anderen, nicht aufgeführten Felder werden unterdrückt.

```
. LIST vorname,name,telefon

00001 Herbert    Maier      089-912345
00002 Josef      Huber      089-987654
00003 Peter      Kuhn       06151-4321
00004 Werner     Meister    089-753267
00005 Herbert    Schmid     030-135790
00006 Paul       Hassold    040-246875
00007 Gerd       Hofmann    040-456987
00008 Josef      Maier      089-321654
00009 Franz      Fischer    0621-96385
00010 Peter      Hoffman    06561-7412
00011 Klaus      Knoll      030-246325
00012 Walter     Koch       0611-753681
00013 Jakob      Becker     06151-45632
00014 Andreas    Hofer      08035-1111
00015 Max        Behr       0911-429681
```

Bild 2.13: Selektion bestimmter Felder aus einer Datei

2.2.2 Bedingtes Auflisten von Datensätzen

Sehr vielseitig verwendbar wird der Befehl LIST in Verbindung mit dem Zusatz FOR (= für) und der Angabe von geeigneten Bedingungen, die erfüllt sein müssen, damit ein Datensatz aufgelistet wird:

LIST FOR (Bedingung)

Da diese Art des gezielten Zugriffs für sämtliche Arbeiten mit dBASE-Datenbanken sehr wichtig ist, soll die Technik, geeignete Bedingungen zu formulieren, schon hier an dieser Stelle etwas ausgiebiger behandelt werden.

Im einfachsten Fall einer Bedingung geben Sie einen Ausdruck vor und suchen alle Datensätze, in denen dieser Ausdruck vorkommt:

LIST FOR (Feldname) = (Ausdruck)

Zum Beispiel ergibt der Feldname **ort** mit dem Ausdruck "8000 München" als Ort im Adressenverzeichnis, also der Befehl

LIST FOR ort = '8000 München'

eine Liste aller Personen, die ihren Wohnort in München haben.

Und mit

LIST FOR name = 'Maier'

werden alle Personen namens Maier gelistet (Bild 2.14).

```
. LIST FOR ort = '8000 München'

00001 Maier   Herbert   Amalienstr.59     8000 München   089-912345
00002 Huber   Josef     Sieglindenstr.5   8000 München   089-987654
00008 Maier   Josef     Flurstr.3         8000 München   089-321654

. LIST FOR name = 'Maier'

00001 Maier   Herbert   Amalienstr.59     8000 München   089-912345
00008 Maier   Josef     Flurstr.3         8000 München   089-321654
```

Bild 2.14: Zwei Beispiele für bedingte Auflistung

Bei der Formulierung der Kommandos und insbesondere der Bedingungen sind einige Regeln zu beachten:

- Der Name des Datenfeldes, das Sie auf die Bedingung hin prüfen (z.B. ort), kann mit Groß- oder Kleinbuchstaben geschrieben sein, muß ansonsten aber mit der Definition entsprechend der Dateistruktur übereinstimmen.

- Der Ausdruck, der mit den Datenfeldern verglichen werden soll (z.B. 8000 München), muß auch bezüglich Groß- und Kleinschreibung korrekt sein, d.h. mit den gespeicherten Daten übereinstimmen. In der gegebenen Datei **adressen** würde ein Ausdruck "8000 MÜNCHEN" nicht gefunden werden, weil dort alle Ortsnamen nur mit großem Anfangsbuchstaben, sonst aber klein geschrieben sind. Wie schon früher gesagt, ist hier eine gewisse Konsequenz bei der Datenerfassung notwendig, die letztlich aber sehr nützlich ist.

- Alphanumerische Ausdrücke wie '8000 München' sind in (einfache oder doppelte) Anführungszeichen zu setzen. Rein numerische Werte werden ohne Anführungszeichen geschrieben. Entscheidend dafür, ob ein Ausdruck als numerisch oder als alphanumerisch zu behandeln ist, ist allein die Definition des Feldes in der Dateistruktur.

- Als Vergleichsoperatoren sind die aus der Mathematik bekannten Symbole zulässig:

  = gleich
  < kleiner
  > größer
  <> ungleich

(Zwischen den beiden Pfeilen für "ungleich" dürfen keine Leerzeichen stehen.) Alle Vergleichsoperatoren sind sowohl für numerische als auch für alphanumerische Daten anwendbar. Es muß also bei der Schaffung einer Datei kein Feld nur deswegen als numerisch definiert werden, weil später vielleicht Vergleiche durchzuführen sind.

Abbildung 2.15 zeigt drei Beispiele für Abfragen nach "gleich", "kleiner als" und "größer als" bei alphanumerischen Werten. In allen Fällen wird die Postleitzahl als Teil des alphanumerischen Feldes **ort** mit einer vorgegebenen Postleitzahl bzw. mit einem Teil davon verglichen.

Im ersten Beispiel werden alle Personen, die in einem Postleitzahl-Bereich unter '8000' wohnen, aufgelistet, während im zweiten alle aus dem Bereich '8xxx', d.h. alle "Bayern" gebracht werden. Dasselbe Ergebnis erhält man, wenn man nur die erste Stelle des Feldes **ort**, nämlich die '8' abfragt, wie im dritten Fall gezeigt wird.

```
. LIST FOR ort < '8000'

00003 Kuhn      Peter     Fliederstr.9      6100 Darmstadt   06151-4321
00005 Schmid    Herbert   Fasanenstr.2      1000 Berlin      030-135790
00006 Hassold   Paul      Meisenstr.18      2000 Hamburg     040-246875
00007 Hofmann   Gerd      Holzweg 13        2000 Hamburg     040-456987
00009 Fischer   Franz     Ginsterweg 9      6850 Mannheim    0621-96385
00011 Knoll     Klaus     Amselweg 23       1000 Berlin      030-246325
00012 Koch      Walter    Heumarkt 14       6000 Frankfurt   0611-753681
00013 Becker    Jakob     Hauptstr.28       6100 Darmstadt   06151-45632


. LIST FOR ort > '7999'

00001 Maier     Herbert   Amalienstr.59     8000 München     089-912345
00002 Huber     Josef     Sieglindenstr.5   8000 München     089-987654
00004 Meister   Werner    Marktplatz 3      8057 Eching      089-753267
00008 Maier     Josef     Flurstr.3         8000 München     089-321654
00010 Hoffman   Peter     Alte Gasse 15     8630 Coburg      09561-7412
00014 Hofer     Andreas   Ludwigstr. 5      8201 Raubling    08035-1111
00015 Behr      Max       Frankenweg 17     8500 Nürnberg    0911-429681


. LIST FOR ort = '8'

00001 Maier     Herbert   Amalienstr.59     8000 München     089-912345
00002 Huber     Josef     Sieglindenstr.5   8000 München     089-987654
00004 Meister   Werner    Marktplatz 3      8057 Eching      089-753267
00008 Maier     Josef     Flurstr.3         8000 München     089-321654
00010 Hoffman   Peter     Alte Gasse 15     8630 Coburg      09561-7412
00014 Hofer     Andreas   Ludwigstr. 5      8201 Raubling    08035-1111
00015 Behr      Max       Frankenweg 17     8500 Nürnberg    0911-429681
```

Bild 2.15: Die Anwendung von Vergleichsoperatoren auf alphanumerische Felder

Bei der Abfrage alphanumerischer Werte ist es wichtig, festzuhalten, daß sich diese wesentlich von der Abfrage numerischer Zahlen unterscheidet. Denn alphanumerische Werte stellen eine Folge von einzelnen Ziffern dar, denen kein Stellenwert wie bei einer Zahl zugeordnet ist. Das heißt, daß z.B. die Ziffernfolge 835 in diesem Sinne größer ist als die Ziffernfolge

39874, weil die erste Ziffer der ersten Folge (8) größer als die erste Ziffer der zweiten Folge (3) ist. So wird auch verständlich, daß die Bedingung "FOR ort = '8' " alle Orte erfaßt, deren erstes Zeichen die Ziffer "8" ist.

2.2.3 Die Substring-Funktion

Während die eben gezeigten Bedingungen nur die Prüfung des gesamten Feldes oder des ersten Teils eines Feldes erlauben, kann man mit Hilfe einer weiteren Variante des Befehls LIST auch beliebige Ausschnitte, also Teile "im Inneren" eines Datenfeldes zur Selektion verwenden.

Hierzu bedient man sich der sogenannten "Substring"-Funktion. Das englische Wort STRING kann man etwa mit "Ausdruck" oder "Zeichenfolge" übersetzen und SUBSTRING bezeichnet einen Teil aus einem String, also eine Untermenge der Zeichenfolge.

Mit dieser Funktion werden alle Datensätze gesucht, für die in dem spezifizierten Feld der angegebene Ausdruck als Teil des Feldinhalts vorkommt. So können Sie beispielsweise alle Personen suchen, deren Name ein "h" enthält.

Die Substring-Funktion in dBASE wird durch das Dollarzeichen "$" dargestellt und wie folgt verwendet:

LIST FOR (Ausdruck)$(Feldname)

Der gesuchte Ausdruck ist wieder in Anführungszeichen zu setzen, wenn er vom Typ "alphanumerisch" ist. Dann folgt das $-Zeichen und dahinter der Name des Feldes, in dem nach dem Ausdruck gesucht werden soll.

Der Befehl, alle Personen zu suchen, deren Name ein "h" enthält, lautet also:

```
LIST FOR 'h'$name
```

und bringt die Anschriften von fünf Personen (Bild 2.16 oben). Es fehlen allerdings diejenigen Namen, die ein großes "H" enthalten, die also mit "H" beginnen, weil dBASE zwischen Groß- und Kleinschreibung unterscheidet. Diese können aber separat mit

LIST FOR 'H'$name

gesucht werden, wie im unteren Teil des Bildes 2.16 gezeigt.

```
. LIST FOR 'h'$name

00003 Kuhn      Peter     Fliederstr.9      6100 Darmstadt   06151-4321
00005 Schmid    Herbert   Fasanenstr.2      1000 Berlin      030-135790
00009 Fischer   Franz     Ginsterweg 9      6850 Mannheim    0621-96385
00012 Koch      Walter    Heumarkt 14       6000 Frankfurt   0611-753681
00015 Behr      Max       Frankenweg 17     8500 Nürnberg    0911-429681

. LIST FOR 'H'$name

00002 Huber     Josef     Sieglindenstr.5   8000 München     089-987654
00006 Hassold   Paul      Meisenstr.18      2000 Hamburg     040-246875
00007 Hofmann   Gerd      Holzweg 13        2000 Hamburg     040-456987
00010 Hoffman   Peter     Alte Gasse 15     8630 Coburg      09561-7412
00014 Hofer     Andreas   Ludwigstr. 5      8201 Raubling    08035-1111
```

Bild 2.16: Groß- und Kleinschreibung als Unterscheidungsmerkmal für die Satzselektion

2.2.4 Mehrere verknüpfte Suchbedingungen

Gelegentlich ist es wünschenswert, die Auswahl noch mehr zu präzisieren und mehrere vergleichende Bedingungen in einem einzigen Befehl zu formulieren. Dazu verfügt dBASE über die drei logischen Grundfunktionen

.AND. (= und, d.h. sowohl ... als auch)
.OR. (= oder, d.h. entweder ... oder)
.NOT. (= nicht)

die, jeweils mit einem Punkt vor und hinter dem Wort geschrieben, mehrere Einzelbedingungen zu einer Gesamtbedingung verbinden. Der Befehl

LIST FOR ('h'$name .OR. 'H'$name)

ergibt beispielsweise eine Liste aller Personen, die ein großes oder kleines "H" in ihrem Namen haben (Bild 2.17).

```
. LIST FOR ('h'$name .OR. 'H'$name)

00002 Huber     Josef     Sieglindenstr.5  8000 München    089-987654
00003 Kuhn      Peter     Fliederstr.9     6100 Darmstadt  06151-4321
00005 Schmid    Herbert   Fasanenstr.2     1000 Berlin     030-135790
00006 Hassold   Paul      Meisenstr.18     2000 Hamburg    040-246875
00007 Hofmann   Gerd      Holzweg 13       2000 Hamburg    040-456987
00009 Fischer   Franz     Ginsterweg 9     6850 Mannheim   0621-96385
00010 Hoffman   Peter     Alte Gasse 15    8630 Coburg     09561-7412
00012 Koch      Walter    Heumarkt 14      6000 Frankfurt  0611-753681
00014 Hofer     Andreas   Ludwigstr. 5     8201 Raubling   08035-1111
00015 Behr      Max       Frankenweg 17    8500 Nürnberg   0911-429681
```

Bild 2.17: Beispiel für zwei mit OR verknüpfte Bedingungen

Als Beispiel für eine UND-Verknüpfung liefert das Kommando

LIST FOR ('H'$name .AND. ort = '8')

alle Personen, deren Name mit "H" beginnt und die gleichzeitig in Bayern wohnen (Bild 2.18).

```
. LIST FOR ('H'$name .AND. ort = '8')

00002 Huber     Josef     Sieglindenstr.5  8000 München    089-987654
00010 Hoffman   Peter     Alte Gasse 15    8630 Coburg     09561-7412
00014 Hofer     Andreas   Ludwigstr. 5     8201 Raubling   08035-1111
```

Bild 2.18: Beispiel für zwei mit AND verknüpfte Bedingungen

Interessant ist die Verknüpfung mehrerer Bedingungen auch, wenn Sie zum Beispiel nach Namen suchen, von denen eventuell verschiedene Schreibweisen in Frage kommen und Sie nicht sicher sind, nach welcher Sie

suchen müssen, wie bei den Namen "Hofmann" und "Hoffman". Das Kommando

LIST FOR ('Hofm'$name .OR. 'Hoffm'$name)

ergibt dann beide Varianten dieses Namens:

```
00007 Hofmann   Gerd    Holzweg 13      2000 Hamburg    040-456987
00010 Hoffman   Peter   Alte Gasse 15   8630 Coburg     09561-7412
```

2.2.5 Eindeutige Suchbedingungen

Bei allen bedingten Befehlen ist es wichtig, die Bedingung eindeutig zu formulieren, und zwar eindeutig in dem Sinne, daß nur solche Datensätze vom System gebracht werden, die Sie tatsächlich wünschen. Wenn Sie beispielsweise alle Telefonnummern im Ortsbereich München (Vorwahlnummer 089) auflisten wollen, dann ist die Suchbedingung

LIST FOR '089'$telefon

nicht eindeutig, weil damit auch Personen aus anderen Ortsbereichen gefunden werden, die zufällig die Ziffernfolge 089 in ihrer Anschlußnummer führen.

Eine Möglichkeit, solche Schwierigkeiten zu umgehen, besteht darin, den Suchbegriff um ein oder mehr Zeichen zu erweitern und damit seine unmittelbare Umgebung mit zu spezifizieren, indem z.B. mit der Vorwahlnummer auch der nachfolgende Bindestrich in den Suchbegriff aufgenommen wird:

LIST FOR '089-'$telefon

Damit ist eindeutig klar, daß es sich hier um eine Vorwahlnummer handelt.

Eine weitere Möglichkeit, die Eindeutigkeit der Suchbedingung zu verbessern, wird mit einer Erweiterung der Substring-Funktion geboten:

LIST FOR (Ausdruck) = $(Feldname,Start,Länge)

Im Unterschied zur bisherigen Verwendung der Substring-Funktion wird hier nach dem Suchbegriff ein Gleichzeichen (=) gesetzt und nach dem $-Zeichen seine Position innerhalb des Feldnamens genau angegeben. Dabei bedeutet "Start" diejenige Zeichenposition (als Zahlenwert) im Feldnamen, von der ab verglichen werden soll, und "Länge" (ebenso als Zahlenwert) gibt die Anzahl der Zeichen an, die zum Vergleich herangezogen werden sollen.

Die Suche nach allen Fernsprechteilnehmern im Ortsbereich München kann dann auch so formuliert werden:

LIST FOR '089' = $(telefon,1,3)

Die gesuchte Zeichenfolge (089) soll also im Feld **telefon** mit dem ersten Zeichen beginnen und drei Zeichen lang sein. Damit ist auch wieder eindeutig die Vorwahlnummer gemeint, und die Auswertung der Datei erfolgt korrekt (Bild 2.19).

Da die voranstehende Null bei jeder Vorwahlnummer gegeben ist, stellt sie kein Unterscheidungsmerkmal dar und kann auch weggelassen werden. Dann ergibt sich folgendes Kommando:

LIST FOR '89' = $(telefon,2,2)

Hier werden also nur zwei Stellen, beginnend bei der zweiten Position, abgefragt.

```
. LIST FOR '089' = $(telefon,1,3)

00001 Maier     Herbert   Amalienstr.59     8000 München    089-912345
00002 Huber     Josef     Sieglindenstr.5   8000 München    089-987654
00004 Meister   Werner    Marktplatz 3      8057 Eching     089-753267
00008 Maier     Josef     Flurstr.3         8000 München    089-32165
```

Bild 2.19: Positionierung des Suchbegriffs im abgefragten Feld

2.2.6 Suchen einzelner Datensätze

Der Befehl LIST bringt stets alle zutreffenden Datensätze gemeinsam zur Anzeige. Daneben besteht aber häufig die Forderung, die Sätze einzeln aufzurufen, zu bearbeiten und zum nächsten überzugehen.

Hierzu bietet dBASE einige weitere Befehle, deren Handhabung weitgehend gleich der Handhabung von LIST FOR ist. Mit dem Befehl

LOCATE FOR

(locate = ausfindig machen) wird der erste Datensatz gesucht, der die gegebene Bedingung erfüllt. Der Satz wird aber nur gesucht und nicht angezeigt bzw. es wird nur die laufende Nummer des gefundenen Satzes auf dem Bildschirm angezeigt. Dieser Befehl ist primär dazu gedacht, im Rahmen von Programmen Datensätze aufzusuchen und für eine Bearbeitung zur Verfügung zu stellen. Dabei muß bzw. soll der Satz nicht auf den Bildschirm gebracht werden.

Dies ist aber dennoch möglich, wenn man anschließend den Befehl

DISPLAY

(display = zeigen) eingibt. Daraufhin erscheint der Satz in einer Zeile so, wie Sie es schon von LIST her kennen.

Um den nächsten Datensatz zu finden,,der die Suchbedingung erfüllt, geben Sie den Befehl

CONTINUE

(continue = fortsetzen) und eventuell danach wieder den Befehl DISPLAY (Bild 2.20).

Auf diese Weise können Sie in Einzelschritten das gleiche erreichen wie zuvor mit dem Befehl LIST, was aber - wie gesagt - im wesentlichen im Rahmen von Programmen interessant ist. In späteren Kapiteln werden noch Beispiele hierfür gegeben.

```
. LOCATE FOR 'Maier'$name

Satz  : 00001    (erscheint als Systemnachricht)

. DISPLAY

00001 Maier   Herbert Amalienstr.59   8000 München   089-912345

. CONTINUE

Satz  : 00008    (erscheint als Systemnachricht)

. DISPLAY

00008 Maier   Josef   Flurstr.3       8000 München   089-321654
```

Bild 2.20: Die Befehle LOCATE, DISPLAY und CONTINUE

Die Bedingungen, die an den Befehl LOCATE FOR geknüpft werden, sind genau wie bei LIST FOR zu formulieren.

Aber im Unterschied zum Kommando LIST, bei dem stets die gesamte Datei von Anfang an durchsucht wird, beginnt der Suchlauf bei LOCATE FOR immer nach dem Datensatz, der gerade aufgerufen ist. Wenn also wie im Beispiel Bild 2.20 der Datensatz 8 als letzter angesprochen war, dann beginnt ein erneuter Suchlauf bei Datensatz 9. Dies ist an sich selbstverständlich (weil andernfalls immer wieder derselbe Satz gefunden würde), wird aber bei der praktischen Arbeit manchmal übersehen und führt dann zu "nichtgefundenen" Datensätzen, wenn der gesuchte Satz weiter oben in der Datei steht.

Der Befehl DISPLAY kann auch ohne vorausgehendes LOCATE zur Suche von Daten benutzt werden, wenn man an das Wort DISPLAY die entsprechenden Bedingungen direkt anknüpft:

DISPLAY FOR (Bedingung)

In dieser Form ist die Wirkung von DISPLAY FOR fast identisch mit der von LIST FOR, d.h. es werden sämtliche Datensätze, die diese Bedingung erfüllen, sofort geliefert. Der einzige Unterschied besteht darin, daß nach jeweils 15 gelisteten Sätzen dBASE anhält und Ihnen Gelegenheit gibt, die

Informationen zu betrachten und erst auf einen Tastendruck hin die nächsten 15 Datensätze ausgibt. In größeren Dateien ist dies eine sehr nützliche Hilfe.

Ebenfalls identisch, und zwar mit dem bedingungslosen LIST, ist der Befehl

DISPLAY ALL

der sämtliche Sätze einer Datei zeigt, wobei aber auch nach jeweils 15 Sätzen eine Pause gemacht wird.

2.2.7 Datenzugriff über die Satznummer

Darüber hinaus gibt zwei weitere Varianten von Display:

DISPLAY Record (Nummer)

DISPLAY Next (Anzahl)

Im ersten Fall wird durch die Angabe der laufenden Nummer ein Datensatz aufgerufen und angezeigt. Im zweiten Fall werden so viele der nachfolgenden Sätze gelistet, wie die Anzahl angibt, wobei mit dem aufgerufenen Satz begonnen wird (Bild 2.21).

```
. DISPLAY RECORD 7

00007 Hofmann   Gerd    Holzweg 13      2000 Hamburg    040-456987


. DISPLAY NEXT 5

00007 Hofmann   Gerd    Holzweg 13      2000 Hamburg    040-456987
00008 Maier     Josef   Flurstr.3       8000 München    089-321654
00009 Fischer   Franz   Ginsterweg 9    6850 Mannheim   0621-96385
00010 Hoffman   Peter   Alte Gasse 15   8630 Coburg     09561-7412
00011 Knoll     Klaus   Amselweg 23     1000 Berlin     030-246325
```

Bild 2.21: DISPLAY RECORD, DISPLAY NEXT

Bei Eröffnung einer Datei mit dem Befehl USE wird stets zunächst der erste Datensatz aufgerufen. Genau genommen wird dabei ein intern von dBASE verwalteter Zähler, der sogenannte Datensatzzeiger (engl. RECORD POINTER) auf die laufende Nummer # 00001 gestellt. Während der anschließenden Bearbeitung der Datei zeigt er auf den augenblicklich aktivierten Datensatz.

Im Laufe einer Dateiverarbeitung kann es vorkommen, daß man aus der Mitte einer Datei wieder an den Anfang zurückkehren muß, um weitere Verarbeitungen zu starten. Die erneute Eingabe des Befehls USE wäre insofern unzweckmäßig, als dann die Datei noch einmal von der Diskette in den Hauptspeicher eingelesen wird, was unötig viel Zeit beansprucht. Statt dessen kann man den Befehl

GO TOP

(= gehe an den Anfang) verwenden, der den Datensatzzeiger auf die laufende Nummer 1 stellt. Weiterhin kann der Befehl

GO (laufende Nummer)

dazu benutzt werden, um unter Nennung einer Satznummer jeden beliebigen Datensatz anzuwählen (Bild 2.22).

```
. GO 3
. DISPLAY

00003 Kuhn    Peter     Fliederstr.9    6100 Darmstadt   06151-4321

. GO TOP
. DISPLAY

00001 Maier   Herbert   Amalienstr.59   8000 München     089-912345

. GO BOTTOM
. DISPLAY

00015 Behr    Max       Frankenweg 17   8500 Nürnberg    0911-429681
```

Bild 2.22: Die Varianten des Befehls GO

Sehr nützlich ist auch der Befehl

GO BOTTOM

(= gehe zum Ende), der den Datensatzzeiger auf den letzten Datensatz richtet. Damit kann dieser Satz, der oft die auch zeitlich letzte Eintragung von Daten enthält, sehr schnell angesprochen werden.

Anmerkung: Bei der Datei, mit der wir uns bisher beschäftigten, ist der Befehl GO TOP in seiner Wirkung identisch mit dem Befehl GO 1. Es gibt aber Dateien (Indexdateien, auf die wir noch zu sprechen kommen), bei denen durch Sortiervorgänge der Satz mit Nummer 1 nicht mehr am Dateianfang steht. Dann ist zwischen beiden Befehlen genau zu unterscheiden.

2.2.8 Abfragen einzelner Felder

Da die Datensätze der Datei **adressen** relativ kurz sind und alle Datenfelder auf je einer Bildschirmzeile Platz finden, war es bislang nicht notwendig, unter den verfügbaren Feldern diejenigen zu selektieren, die betrachtet werden sollten. Dies trifft in der Praxis häufig aber nicht zu. Oft kann nur ein Teil aller Felder gleichzeitig auf dem Bildschirm dargestellt werden, und dann muß selektiert werden.

Eine Möglichkeit, einzelne Felder aus einem Satz zu betrachten, haben Sie bereits mit dem Befehl LIST kennengelernt, für den einzelne Felder spezifiziert werden können. Analog läßt sich auch der Befehl DISPLAY verwenden, zum Beispiel:

```
LIST vorname,name,telefon
```

```
DISPLAY vorname,name,telefon
```

Ist der gewünschte Datensatz bereits im Zugriff (z.B. mittels LOCATE FOR), dann können einzelne Datenfelder sehr rasch durch Eingabe eines Fragezeichens mit nachfolgendem Feldnamen abgefragt werden. Die Antwort erscheint dann jeweils in der folgenden Zeile (Bild 2.23).

```
. LOCATE FOR  vorname = 'Paul'
Satz  : 0006     (= Systemnachricht)

. ? name
Hassold

. ? telefon
040-246875

. ? ort, strasse
2000 Hamburg     Meisenstr.18
```

Bild 2.23: Das Fragezeichen für schnelle Abfragen

Übrigens lassen sich mit Hilfe des Fragezeichens auch kleine Rechnungen durchführen, ähnlich wie auf einem Taschenrechner. Sie müssen lediglich nach dem Fragezeichen die zu berechnende Formel eingeben und erhalten in der nächten Zeile das Resultat.

2.2.9 Zusammenfassung

In diesem zweiten Abschnitt haben Sie die wichtigsten Methoden kennengelernt, wie man auf eine dBASE-Datenbank zugreift, wie man Datensätze sucht, auswählt und einzelne Daten abfragt. Damit können Sie prinzipiell mit einer Datenbank arbeiten und gezielt Informationen abrufen.

Für größere Datenbanken werden allerdings noch eine Reihe weiterer Befehle benötigt, z.B. um Dateien zu sortieren, um in großen Datenbeständen sehr schnell die gewünschten Informationen zu erhalten oder auch um umfassende Auswertungen vornehmen zu können. Diese zusätzlichen Leistungen werden Gegenstand der folgenden Kapitel sein.

2.3 Das Ändern von Dateistrukturen

In der Praxis kommt es häufig vor, daß man existierende Datenverarbeitungssysteme ändern oder erweitern möchte und dazu neue Anwendungsprogramme schreibt. In der Regel greifen die neuen Programme teils auf vorhandene Datenbestände zurück, teils müssen zusätzliche Daten erfaßt und dann entweder in die bestehenden Dateien integriert oder in neue Dateien geladen werden.

Hier taucht bei vielen Dateiverwaltungssystemen das Problem auf, daß die Modifikation existierender Strukturen sehr aufwendig ist. Zudem sind Dateistruktur und Anwendungsprogramm oft derart eng aufeinander abgestimmt, daß eine Dateiänderung unvermeidbar eine Änderung der Anwendungsprogramme nach sich zieht. Der Aufwand zur Wartung solcher Systeme kann dann sehr rasch beträchtliche Ausmaße annehmen.

Relationale Datenbanken sind in aller Regel programmunabhängig und reduzieren das Problem auf ein Minimum. Insbesondere dBASE II gestattet auf sehr einfache Weise, Dateistrukturen zu ändern, ohne die vorhandenen Anwendungsprogramme zu tangieren. Die Änderung einer Dateistruktur in dBASE erfolgt in drei Schritten:

1. Die Daten der betreffenden Datei werden "ausgelagert", d.h. in eine temporäre Datei kopiert, was mit einem einzigen dBASE-Befehl erfolgt.

2. Die Dateistruktur wird geändert, was ebenso komfortabel möglich ist wie das Ändern von Daten mittels der EDIT-Funktion.

3. Die Daten werden aus der temporären Datei in die geänderte Datei unter Berücksichtigung der neuen Struktur zurückgeschrieben. Dies erfolgt wiederum mit einem einzigen Befehl.

Zwei Beispiele sollen nachfolgend das Ändern von Dateistrukturen erläutern.

Zunächst soll das Adressenverzeichnis in eine Kundendatei umgewandelt werden, die in einem kleinen Unternehmen für die Stammkunden geführt wird. Dazu ist es zweckmäßig, jedem Kunden eine Kundennummer zu geben, die ihn eindeutig identifiziert. Außerdem soll in diesem Beispiel die Telefonnummer aus der Datei entfernt werden.

Eröffnen Sie die Datei mit dem Befehl:

USE adressen

Wie bereits angedeutet, müssen im ersten Schritt die Daten aus der Datei **adressen** in eine temporäre Datei ausgelagert werden. Hierzu wird der Befehl

COPY TO (Dateiname)

benutzt, der die gesamte Datei inklusive ihrer Struktur in eine andere, von Ihnen zu benennende Datei kopiert. Diese Datei muß nicht vorher definiert werden. Sie wird es automatisch mit dem Kopierbefehl. Nachdem es sich um eine temporäre Datei handelt und ein Dateiname möglichst aussagekräftig sein soll, geben Sie ihr den Dateinamen **temp** als Abkürzung für temporär:

COPY TO temp

Unter dem Namen **temp** wird nun ein vollständiges Abbild der Datei **adressen** auf die Diskette geschrieben. Die Dauer des Kopiervorgangs hängt wesentlich von der Anzahl und Größe der Datensätze, zum Teil auch von der Schreib-/Lese-Geschwindigkeit der Massenspeicher ab. Die 15 Datensätze dieses Beispiels sind aber in jedem Fall in wenigen Sekunden kopiert. Daraufhin meldet dBASE das Ende des Kopiervorgangs:

00015 Sätze kopiert

Als zweiter Schritt wird nun die Struktur der Datei **adressen** modifiziert, was mit dem Befehl

MODIFY STRUCTURE

eingeleitet wird. Mit Beginn einer Strukturänderung werden alle Daten der Datei gelöscht. Um die Datensätze dadurch nicht zu verlieren, hatten Sie sie zuvor in die temporäre Datei kopiert, von der sie später zurückgeholt werden können. Auf die Nachricht bzw. Warnung von dBASE

MODIFY löscht alle Datensätze - weiter? (J/N)

können Sie also ohne weiteres mit J (=Ja) antworten, worauf der Bildschirm gelöscht wird und die Dateistruktur in einer Editiermaske erscheint (Bild 2.24).

Jede Zeile der Maske beschreibt ein Datenfeld, das mit einer Feldnummer gekennzeichnet ist. 23 Felder sind auf einer Bildschirmseite vorgesehen. Die weiteren Felder bis 32 können mit Hilfe der Cursor-Tasten erreicht werden. Die Kombination SHIFT-Cursor-Taste (nach oben oder unten), die Sie schon vom Befehl EDIT her kennen, kann auch hier zum "Umblättern" benutzt werden. Um den Anschluß zu den benachbarten Feldern nicht zu verlieren, wird das Bild aber jeweils nur um acht Zeilen weitergeschaltet.

Die einzelnen Informationen über die Felder sind Ihnen bereits bekannt: der Name des Feldes, sein Typ, die Feldlänge und die Anzahl der Dezimalstellen. Innerhalb der Maske können Sie nun den Cursor wieder beliebig bewegen und Änderungen vornehmen, wie Sie es schon von der EDIT-Funktion her kennen. Ebenso stehen sämtliche Funktionen, die in Kombination mit der CTRL-Taste bei EDIT möglich sind, zur Verfügung.

Hinzu kommen drei weitere Kombinationen:

CTRL-T — löscht die gesamte Zeile in der der Cursor steht (d.h. das gesamte Feld). Alle nachfolgenden Felder schließen nach oben auf. Diese Funktion können Sie auf der Schneider-Tastatur auch mit SHIFT-CLR erreichen.

CTRL-Y — löscht ebenfalls die gesamte Zeile, jedoch rücken die nachfolgenden Felder nicht nach. Die geleerte Zeile kann sofort neu beschrieben werden. Die Tastenkombination CTRL-CLR führt zum gleichen Ergebnis.

CTRL-N — schiebt alle Felder ab der Zeile des Cursors um eine Zeile nach unten und ermöglicht das Einfügen eines neuen Feldes.

```
          Name      Typ  Länge Dez.st.
Feld  01 : NAME       C   010    000 :
Feld  02 : VORNAME    C   010    000 :
Feld  03 : STRASSE    C   015    000 :
Feld  04 : ORT        C   016    000 :
Feld  05 : TELEFON    C   011    000 :
Feld  06 :                           :
Feld  07 :                           :
Feld  08 :                           :
Feld  09 :                           :
Feld  10 :                           :
Feld  11 :                           :
Feld  12 :                           :
Feld  13 :                           :
Feld  14 :                           :
Feld  15 :                           :
Feld  16 :                           :
Feld  17 :                           :
Feld  18 :                           :
Feld  19 :                           :
Feld  20 :                           :
Feld  21 :                           :
Feld  22 :                           :
Feld  23 :                           :
```

Bild 2.24: Die Maske zur Änderung von Dateistrukturen

Bringen Sie nun den Cursor in die erste Zeile und drücken Sie dann CTRL-N, um oberhalb des Namens das Feld für die Kundennummer einzufügen. Die Kundennummer soll den Namen **kunummer** erhalten, von Typ C und drei Stellen lang sein. Schreiben Sie daher in das neue Feld 01:

```
          Name        Typ   Länge Dez.st.
Feld 01  :kunummer     C     003    000 :
```

Positionieren Sie dann den Cursor auf Feld 6, dem Feld für die Telefonnummer und drücken Sie CTRL-T: Die Zeile wird gelöscht. Damit hat die Struktur der Datei **adressen** die gewünschte Änderung erfahren (Bild 2.25). Mit Betätigung der Tasten CTRL-W wird sie von dBASE übernommen und gespeichert.

```
        Name        Typ  Länge  Dez.st.
Feld 01 :kunummer    C   003    000  :
Feld 02 :NAME        C   010    000  :
Feld 03 :VORNAME     C   010    000  :
Feld 04 :STRASSE     C   015    000  :
Feld 05 :ORT         C   016    000  :
Feld 06 :                            :
Feld 07 :                            :
```

Bild 2.25: Die neue Struktur der Datei **adressen**

In einem letzten Schritt müssen nun noch die Daten aus der temporären Datei in die neue Struktur überspielt werden, was mit dem Befehl

APPEND FROM temp

erfolgt. Den Befehl APPEND (=anhängen, hinzufügen) haben Sie schon früher kennengelernt, als Sie neue Datensätze über die Tastatur eingaben. Der Zusatz FROM bedeutet, daß die Daten von einer anderen Datei, nämlich von der Datei **temp**, übernommen werden sollen. Der Vorgang dauert wieder einige Sekunden und wird mit der dBASE-Nachricht

00015 Sätze hinzugefügt

abgeschlossen. Die neue Datei **adressen** ist komplett.

Mit dem Befehl APPEND werden nur diejenigen Datenfelder zurückgeschrieben, die in der neuen Struktur noch berücksichtigt sind. Da das Feld für die Telefonnummern gelöscht wurde, erscheinen diese in der neuen Datei nicht mehr. Gleichzeitig ist vor den Namen der Kunden Platz für die dreistellige Kundennummer geschaffen worden (Bild 2.26).

00001	Maier	Herbert	Amalienstr.59	8000 München
00002	Huber	Josef	Sieglindenstr.5	8000 München
00003	Kuhn	Peter	Fliederstr.9	6100 Darmstadt
00004	Meister	Werner	Marktplatz 3	8057 Eching
00005	Schmid	Herbert	Fasanenstr.2	1000 Berlin
00006	Hassold	Paul	Meisenstr.18	2000 Hamburg
00007	Hofmann	Gerd	Holzweg 13	2000 Hamburg
00008	Maier	Josef	Flurstr.3	8000 München
00009	Fischer	Franz	Ginsterweg 9	6850 Mannheim
00010	Hoffman	Peter	Alte Gasse 15	8630 Coburg
00011	Knoll	Klaus	Amselweg 23	1000 Berlin
00012	Koch	Walter	Heumarkt 14	6000 Frankfurt
00013	Becker	Jakob	Hauptstr.28	6100 Darmstadt
00014	Hofer	Andreas	Ludwigstr. 5	8201 Raubling
00015	Behr	Max	Frankenweg 17	8500 Nürnberg

Bild 2.26: Die geänderte Datei **adressen**

Anmerkung: Die Ausführung des Befehls APPEND FROM ist eigentlich ein Kopiervorgang, und zwar ein selektiver, bei dem nur diejenigen Daten kopiert werden, die in der Struktur der Zieldatei vorgesehen sind. Die Ursprungsdatei (hier: **temp**) wird durch APPEND nicht zerstört, sondern bleibt voll erhalten, wovon Sie sich überzeugen können, wenn Sie die Befehle

USE temp

LIST

eingeben.

Sie können bzw. sollten sich jetzt mit dem Befehl LIST STRUCTURE oder auch mit dem Befehl DISPLAY STRUCTURE die geänderte Struktur der Datei **adressen** protokollieren (gegebenenfalls mit CTRL-P den Drucker zuschalten). Um die Kundennummern in der Datei nachzutragen, können Sie entweder mit dem Befehl EDIT die Datensätze einzeln aufrufen und ergänzen oder - und das ist in diesem Fall wesentlich schneller - erst mit dem Befehl GO TOP zum Dateianfang gehen und dann einen neuen, EDIT-ähnlichen Befehl verwenden:

BROWSE

BROWSE könnte ungefähr als "in der Datei blättern" übersetzt werden. Mit diesem außerordentlich mächtigen Befehl können 20 aufeinander folgende Datensätze einer Datei gleichzeitig auf den Bildschirm gerufen und bearbeitet werden (Bild 2.27, hier sind die Kundennummern bereits eingetragen). Die Datensätze sind zeilenweise angeordnet und können in beliebiger Reihenfolge durch Cursorbewegung erreicht und bearbeitet werden. In der linken oberen Ecke des Bildschirms ist der Datensatz angegeben, auf dem sich augenblicklich der Cursor befindet.

```
Satznummer :00001
KUN NAME------ VORNAME--- STRASSE---------- ORT------------
123 Maier      Herbert    Amalienstr.59     8000 München
234 Huber      Josef      Sieglindenstr.5   8000 München
345 Kuhn       Peter      Fliederstr.9      6100 Darmstadt
098 Meister    Werner     Marktplatz 3      8057 Eching
761 Schmid     Herbert    Fasanenstr.2      1000 Berlin
147 Hassold    Paul       Meisenstr.18      2000 Hamburg
258 Hofmann    Gerd       Holzweg 13        2000 Hamburg
369 Maier      Josef      Flurstr.3         8000 München
852 Fischer    Franz      Ginsterweg 9      6850 Mannheim
741 Hoffman    Peter      Alte Gasse 15     8630 Coburg
963 Knoll      Klaus      Amselweg 23       1000 Berlin
159 Koch       Walter     Heumarkt 14       6000 Frankfurt
357 Becker     Jakob      Hauptstr.28       6100 Darmstadt
267 Hofer      Andreas    Ludwigstr. 5      8201 Raubling
421 Behr       Max        Frankenweg 17     8500 Nürnberg
```

Bild 2.27: Gleichzeitige Bearbeitung mehrerer Datensätze mit dem Befehl BROWSE

Zusätzlich zu den Cursortasten können folgende Funktionstasten eingesetzt werden (siehe auch Befehl EDIT):

SHIFT-↓ oder CTRL-C erlaubt, den Cursor senkrecht nach unten durch alle Felder gleichen Typs zu bewegen

SHIFT-↑ oder CTRL-R erlaubt, den Cursor senkrecht nach oben durch alle Felder gleichen Typs zu bewegen

Positionieren Sie den Cursor auf das Feld KUN (= Abkürzung für Kundennummer) des ersten Datensatzes und geben Sie die Ziffer 123 ein. Mit CTRL-C springen Sie in das gleiche Feld des zweiten Satzes und tragen die Ziffer 234 ein. Entsprechend verfahren Sie mit allen Sätzen, wobei Sie die Kundennummern aus Bild 2.27 übernehmen. Mit CTRL-W wird die Bearbeitung abgeschlossen und die Funktion BROWSE verlassen. Die modifizierte Datei **adressen** ist nun komplett.

Mit den eben besprochenen Befehlen kann man nicht nur gegebene Dateistrukturen ändern, sondern aus ihnen auch neue Strukturen ableiten und damit neue Dateien erzeugen:

Neben der existierenden Kundendatei, die die Adressen der Kunden enthält, sollen jetzt weitere kundenbezogene Daten mit Informationen über die geschäftlichen Aktvitäten in einer sparaten Datei erfaßt und bearbeitet werden. Die neue Datei, die den Namen **kunden** erhält, muß zunächst einmal Angaben über die Identität des Kunden enthalten. Dazu genügt die Angabe der Kundennummer. Name und Anschrift des Kunden müssen nicht noch einmal erfaßt und verwaltet werden. Sie können später bei Bedarf aus der Datei **adressen** entnommen werden. Zur Beschreibung der geschäftlichen Aktivitäten sollen Datum und Nummer der letzten Rechnung, der bislang (z.B. in diesem Jahr) aufgelaufene Umsatz mit dem Kunden und die Außenstände beim Kunden erfaßt werden. Die Definition der neuen Datei **kunden** leiten Sie mit den folgenden vier Befehlen ein:

```
USE adressen

COPY STRUCTURE TO kunden

USE kunden

MODIFY STRUCTURE
```

Sie rufen also die Datei **adressen** auf, kopieren nur die Struktur (das geht schneller als die gesamte Datei mit Daten) in die neue Datei **kunden** und rufen dann diese auf. Ihre Struktur kann nun anschließend für die neu hinzukommenden Informationen modifiziert werden.

Auf den Befehl MODIFY STRUCTURE hin erscheint die Änderungsmaske mit den Feldbeschreibungen, wie sie zuletzt auch für die Datei **adressen** galten (Bild 2.28, vgl. auch Bild 2.25).

Löschen Sie nun die Felder **name**, **vorname** und **strasse**, indem Sie den Cursor nacheinander auf diese Felder positionieren und CTRL-T drücken. Das Feld **ort** behalten Sie bei, ändern aber die Feldlänge auf 4 Stellen. Dieses verkürzte Feld **ort** wird später die Postleitzahl des Kundenortes aufnehmen. An sich ist diese Information in dieser Datei nicht notwendig, weil die vollständige Anschrift bereits in der Adressendatei vorhanden ist. Für verschiedene schnelle Auswertungen wird es sich aber als nützlich erweisen, die Postleitzahl als einen der möglichen Sortierbegriffe in der Datei **kunden** zu halten.

```
          Name        Typ  Länge Dez.st.
Feld 01   :KUNUMMER    C   003    000 :
Feld 02   :NAME        C   010    000 :
Feld 03   :VORNAME     C   010    000 :
Feld 04   :STRASSE     C   015    000 :
Feld 05   :ORT         C   016    000 :
Feld 06   :                           :
```

Bild 2.28: Die Struktur der Datei **kunden** vor der Modifikation

Ergänzen Sie nun die Dateistruktur um fünf weitere Felder, und zwar um eine fünfstellige Rechnungsnummer **lrech** (= Abkürzung für "letzte Rechnungsnummer"), um ein achtstelliges Datum und um drei numerische Felder (Bild 2.29). Dies sind die Felder für den akkumulierten Umsatz (als Summe aller bisherigen Rechnungen), für den vom Kunden davon bezahlten Teilbetrag und für die Außenstände als Differenz (Umsatz minus bezahlter Teil).

```
          Name        Typ Länge Dez.st.
Feld 01   :KUNUMMER    C   003  000  :
Feld 02   :ORT         C   004  000  :
Feld 03   :lrech       c   005  000  :
Feld 04   :datum       c   008  000  :
Feld 05   :umsatz      n   010  002  :
Feld 06   :bezahlt     n   010  002  :
Feld 07   :offen       n   010  002  :
Feld 08   :
```

Bild 2.29: Die endgültige Dateistruktur **kunden**

Bei den drei numerischen Feldern müssen als Typ das Kennzeichen "n" oder "N" und die Anzahl der Dezimalstellen nach dem Komma eingegeben werden, in diesem Fall die Zahl 002. Die gesamte Feldlänge ergibt sich aus der Summe "Stellen vor dem Komma plus Stellen nach dem Komma plus eine Stelle für das Komma selbst". Bei einer Feldlänge von 10 Zeichen sind das also sieben Stellen vor und zwei Stellen nach dem Komma.

Mit CTRL-W können Sie die Strukturierung der Datei abschließen und mit dem Befehl

APPEND FROM adressen

alle Kundennummern und Postleitzahlen von der Datei **adressen** in die Datei **kunden** kopieren. Da das Feld **ort** der neuen Datei **kunden** nur noch vier Zeichen lang ist, werden aus der Datei **adressen** nur die ersten vier Zeichen der dortigen Orte, nämlich die Postleitzahlen übernommen. Die Ortsnamen selbst werden nicht kopiert (Bild 2.30). Die leeren numerischen Felder werden mit den Zahlenwerten "0.00" entsprechend des festgelegten Formats mit zwei Dezimalstellen beschrieben.

00001	123	8000	0.00	0.00	0.00
00002	234	8000	0.00	0.00	0.00
00003	345	6100	0.00	0.00	0.00
00004	098	8057	0.00	0.00	0.00
00005	761	1000	0.00	0.00	0.00
00006	147	2000	0.00	0.00	0.00
00007	258	2000	0.00	0.00	0.00
00008	369	8000	0.00	0.00	0.00
00009	852	6850	0.00	0.00	0.00
00010	741	8630	0.00	0.00	0.00
00011	963	1000	0.00	0.00	0.00
00012	159	6000	0.00	0.00	0.00
00013	357	6100	0.00	0.00	0.00
00014	267	8201	0.00	0.00	0.00
00015	421	8500	0.00	0.00	0.00

Bild 2.30: Die Datei **kunden**

Um mit der Datei **kunden** arbeiten zu können, sollen einige Zahlenwerte eingegeben werden. Und zwar sollen zuerst die bisherigen Kundenumsätze zum heutigen Stichtag erfaßt werden.

Wenn Sie beim Start von dBASE noch nicht das heutige Datum eingegeben haben, dann sollten Sie dies jetzt nachholen, damit Sie die nächsten Schritte vollständig nachvollziehen können. Dazu können Sie entweder dBASE mit QUIT verlassen und mit einer Datumseingabe neu starten. Oder Sie können auch mit einem speziellen Kommando das aktuelle Datum nachtragen:

SET DATE TO (Datum)

Das Datum geben Sie so ein, wie es beim Start von dBASE verlangt wurde, nämlich im Format Tag/Monat/Jahr, zum Beispiel für den 15.8.1985:

SET DATE TO 15/08/85

Anmerkung: Im Gegensatz zur Datumseingabe beim Starten von dBASE wird hier keine Plausibilitätsprüfung mehr durchgeführt. Sie könnten also auch das amerikanische Format Monat/Tag/Jahr benutzen. Davon möchten wir Ihnen aber ganz dringend abraten. Denn das führt, wenn Sie nicht sehr konsequent sind, eines Tages zu zwei verschiedenen Datenformaten in Ihren Dateien, was eine einheitliche Auswertung verhindern kann.

Eröffnen Sie die Datei **kunden** und geben Sie den Befehl BROWSE:

USE kunden

BROWSE

Nun können Sie alle Umsätze Ihrer Kunden in der bereits beschriebenen Weise eintragen. Die Werte entnehmen Sie dem Bild 2.31.

Mehr als die Umsatzwerte ist zunächst nicht zu erfassen, da die anderen Daten von dBASE zugespielt werden sollen.

00001	123	8000	1234.00	0.00	0.00
00002	234	8000	3218.00	0.00	0.00
00003	345	6100	2005.67	0.00	0.00
00004	098	8057	5612.89	0.00	0.00
00005	761	1000	3005.55	0.00	0.00
00006	147	2000	980.00	0.00	0.00
00007	258	2000	355.45	0.00	0.00
00008	369	8000	654.00	0.00	0.00
00009	852	6850	1200.00	0.00	0.00
00010	741	8630	587.00	0.00	0.00
00011	963	1000	1591.50	0.00	0.00
00012	159	6000	852.00	0.00	0.00
00013	357	6100	2111.00	0.00	0.00
00014	267	8201	698.88	0.00	0.00
00015	421	8500	761.00	0.00	0.00

Bild 2.31: Erfassen der Umsätze je Kunde

2.3.1 Selbsttätiges Generieren von Daten

Beim Aufbau einer Datenbank müssen naturgemäß viele Informationen erstmals über die Tastatur in das System eingegeben werden, was zeitlich aufwendig und mit erheblichen Fehlerrisiken behaftet ist. Deshalb sollte man bei jeder Anwendung bestrebt sein, so wenig Daten wie möglich manuell einzugeben und statt dessen vom System selbst die Daten generieren lassen, wann immer dies möglich ist. Umfassende Fähigkeiten, bereits vorhandene Informationen für die Einrichtung neuer Dateien zu nutzen, sind ein wesentliches Merkmal guter Datenbanksysteme.

dBASE stellt hierfür eine Reihe von Kommandos zur Verfügung. Dazu gehört etwa der eben vorgestellte Befehl APPEND FROM, aber auch einige neue, wie zum Beispiel die Anweisung

REPLACE (Feldname) WITH (Ausdruck)

die mit all ihren Varianten sehr mächtig ist und typische Eigenschaften einer relationalen Dakenbank realisiert. Für einen im Zugriff befindlichen Datensatz läßt sich dieses Kommando wie folgt interpretieren: "Ersetze den Inhalt des Feldes (Feldname) durch den Inhalt oder durch das Ergebnis von (Ausdruck)." Als Ausdruck kann hier ein anderer Feldname, ein Text, eine

Zahl oder eine Formel stehen. Das Einfügen des Wortes ALL führt zu einer Variante

REPLACE ALL (Feldname) WITH (Ausdruck)

die bedeutet, daß jetzt in allen (!) Sätzen der Datei das entsprechende Feld bearbeitet wird. Die Manipulation einer kompletten Datei mit einem einzigen Befehl REPLACE ALL ist nur bei relationalen Datenbanksystemen möglich und ist einer der Gründe für die steigende Beliebtheit dieses Datenbanktyps.

Schreiben Sie:

REPLACE ALL datum WITH Date()

Hiermit werden alle Felder mit Feldnamen **datum** durch den Inhalt des Ausdrucks **Date()** ersetzt, wobei **Date()** der von dBASE fest vergebene Speicher für das aktuelle Datum ist, das Sie beim Start von dBASE bzw. mit dem Kommando "SET DATE TO (Datum)" eingaben. In alle Datumsfelder der Datei **kunden** wird Ihr aktuelles Datum geschrieben. Bild 2.32 zeigt die Datei für den Fall, daß als aktuelles Datum der 15. August 1985 eingegeben wurde.

00001	123	8000	15/08/85	1234.00	0.00	0.00
00002	234	8000	15/08/85	3218.00	0.00	0.00
00003	345	6100	15/08/85	2005.67	0.00	0.00
00004	098	8057	15/08/85	5612.89	0.00	0.00
00005	761	1000	15/08/85	3005.55	0.00	0.00
00006	147	2000	15/08/85	980.00	0.00	0.00
00007	258	2000	15/08/85	355.45	0.00	0.00
00008	369	8000	15/08/85	654.00	0.00	0.00
00009	852	6850	15/08/85	1200.00	0.00	0.00
00010	741	8630	15/08/85	2587.00	0.00	0.00
00011	963	1000	15/08/85	1591.50	0.00	0.00
00012	159	6000	15/08/85	852.00	0.00	0.00
00013	357	6100	15/08/85	2111.00	0.00	0.00
00014	267	8201	15/08/85	698.88	0.00	0.00
00015	421	8500	15/08/85	761.00	0.00	0.00

Bild 2.32: Übernahme des aktuellen Datums

Für das vorliegende Beispiel soll zunächst angenommen werden, daß die Verkäufe (= Umsätze) an die Kunden jeweils zu 80% von diesen bezahlt sind und die Außenstände somit 20% ausmachen. Dann können die Inhalte der Spalten für **bezahlt** und für **offen** mit je einem Befehl berechnet werden. Schreiben Sie:

```
REPLACE ALL bezahlt WITH umsatz * 0.8

REPLACE ALL offen WITH umsatz - bezahlt
```

In sämtliche Felder **bezahlt** werden also 80% des jeweiligen Kundenumsatzes und in alle Felder **offen** die jeweilige Differenz aus Umsatz und bezahltem Teilbetrag geschrieben (Bild 2.33).

```
00001  123  8000  15/08/85  1234.00   987.20   246.80
00002  234  8000  15/08/85  3218.00  2574.40   643.60
00003  345  6100  15/08/85  2005.67  1604.53   404.14
00004  098  8057  15/08/85  5612.89  4490.31  1122.58
00005  761  1000  15/08/85  3005.55  2404.44   601.11
00006  147  2000  15/08/85   980.00   784.00   196.00
00007  258  2000  15/08/85   355.45   284.36    71.09
00008  369  8000  15/08/85   654.00   523.20   130.80
00009  852  6850  15/08/85  1200.00   960.00   240.00
00010  741  8630  15/08/85  2587.00  2069.60   517.40
00011  963  1000  15/08/85  1591.50  1273.20   318.30
00012  159  6000  15/08/85   852.00   681.60   170.40
00013  357  6100  15/08/85  2111.00  1688.80   422.20
00014  267  8201  15/08/85   698.88   559.10   139.78
00015  421  8500  15/08/85   761.00   608.80   152.20
```

Bild 2.33: Die Datei **kunden**

Eine weitere Möglichkeit, Datensätze zu bearbeiten, bietet der Befehl

CHANGE (Bereich) FIELD (Feldname)

(CHANGE = ändern), der in gewisser Weise eine Mittelstellung zwischen EDIT und REPLACE einnimmt. Mit CHANGE werden gezielt einzelne Datenfelder aufgerufen, deren Inhalt dann manuell, d.h. über die Tastatur geändert werden kann.

Um die anfangs pauschale Bezifferung aller bezahlten Umsätze mit 80% vom Verkaufswert etwas abzuwandeln, soll das Feld **bezahlt** bei einigen Datensätzen neue Werte erhalten. Geben Sie dazu den Befehl

CHANGE ALL FIELD bezahlt

der veranlaßt, daß alle Felder **bezahlt** der Reihe nach aufgerufen werden, um gegebenfalls die Inhalte zu ändern. Daraufhin erscheint das zu ändernde Feld des ersten Satzes auf dem Bildschirm, wobei in einer Zeile der bisherige Wert gezeigt wird und in die folgende Zeile hinter das Wort "nach:" ("TO:") ein neuer Betrag geschrieben werden kann:

```
Satz : 00001

BEZAHLT:     987.20
nach:
```

Soll der Wert unverändert bleiben, dann drücken Sie nur die ENTER-Taste. Anschließend bringt dBASE den nächsten Satz. Geben Sie auf diese Weise die folgenden neuen Werte ein:

Satz	BEZAHLT:	nach:
00001	987.20	1234.00
00002		
00003	1604.53	2005.67
00004		
00005		
00006	784.00	0.00
00007	284.36	355.45

00008	523.20	654.00
00009	960.00	0.00
00010	2069.60	0.00
00011	1273.20	0.00
00012		
00013		
00014		
00015		

Die Felder ohne Betragsangaben bleiben unverändert.

Natürlich müssen auch die Werte für die Außenstände (Feld **offen**) entsprechend korrigiert werden, damit diese wieder die Differenz aus Umsatz und bezahltem Anteil ergeben. Dazu wiederholen Sie noch einmal den Befehl

REPLACE ALL offen WITH umsatz - bezahlt

und erhalten die Datei **kunden** mit den neuen Zahlenwerten (Bild 2.34).

00001	123	8000	15/08/85	1234.00	1234.00	0.00
00002	234	8000	15/08/85	3218.00	2574.40	643.60
00003	345	6100	15/08/85	2005.67	2005.67	0.00
00004	098	8057	15/08/85	5612.89	4490.31	1122.58
00005	761	1000	15/08/85	3005.55	2404.44	601.11
00006	147	2000	15/08/85	980.00	0.00	980.00
00007	258	2000	15/08/85	355.45	355.45	0.00
00008	369	8000	15/08/85	654.00	654.00	0.00
00009	852	6850	15/08/85	1200.00	0.00	1200.00
00010	741	8630	15/08/85	2587.00	0.00	2587.00
00011	963	1000	15/08/85	1591.50	0.00	1591.50
00012	159	6000	15/08/85	852.00	681.60	170.40
00013	357	6100	15/08/85	2111.00	1688.80	422.20
00014	267	8201	15/08/85	698.88	559.10	139.78
00015	421	8500	15/08/85	761.00	608.80	152.20

Bild 2.34: Die aktualisierte Datei **kunden**

2.4 Sortieren und Indizieren

2.4.1 Sortieren

Sie haben bereits eine ganze Reihe von Befehlen zur Datenmanipulation kennengelernt, darunter auch Befehle, die man bei manchen anderen Datenbanken vergeblich sucht. Aber vielleicht haben Sie sich schon gefragt, warum bislang vom Sortieren einer Datei noch keine Rede war, wo dies doch bei vielen Dateiverwaltungssystemen eine der wichtigsten Funktionen ist.

Der Grund ist folgender: Das Sortieren einer Datei ist bei einer relationalen Datenbank, und insbesondere bei dBASE, von nicht so großer Bedeutung wie bei anderen Systemen. Alle bisher besprochenen und die meisten künftigen Befehle benötigen keine sortierten Dateien für ihre Ausführung. Lediglich einige Befehle zur Verknüpfung und Auswertung von Dateien setzen eine geeignete Sortierung voraus und selbstverständlich wünscht der Benutzer in ausgedruckten Berichten die Ergebnisse in irgendeiner Form geordnet.

Zum Sortieren einer Datei dient der Befehl

SORT ON (Feldname) TO (Zieldatei) (DESCENDING)

der die Datei nach einem zu benennenden Feld als Sortierbegriff in auf- oder absteigender Reihenfolge sortiert und dann in der anzugebenden Zieldatei abspeichert, wobei für absteigende Reihenfolge der obige Zusatz DESCENDING anzufügen ist.

Die zu sortierende Ursprungsdatei und die Zieldatei müssen stets verschieden benannt sein. Andernfalls weist dBASE den Befehl zurück. Denn eine Datei darf nie "in sich selbst" sortiert werden, da sonst im Falle einer Betriebsstörung während des Sortierens (z.B. Stromausfall) die Datei vernichtet werden könnte. Durch die Trennung von Ursprungs- und Zieldatei wird dafür gesorgt, daß die Ursprungsdatei bis zum Abschluß des Sortierens erhalten bleibt. So kann ein Sortierlauf notfalls wiederholt werden.

Die Datei **kunden** soll nun nach aufsteigender Kundennummer sortiert und in der temporären Datei **temp** abgelegt werden. Anschließend wird die temporäre Datei eröffnet und in die Datei **kunden** zurückkopiert. Dann können

Sie diese wieder aufrufen und betrachten (Bild 2.35). Geben Sie dazu die Befehlsfolge:

```
USE kunden
SORT ON kunummer TO temp
USE temp
COPY TO kunden
USE kunden
LIST
```

```
00001   098   8057   15/08/85   5612.89   4490.31   1122.58
00002   123   8000   15/08/85   1234.00   1234.00      0.00
00003   147   2000   15/08/85    980.00      0.00    980.00
00004   159   6000   15/08/85    852.00    681.60    170.40
00005   234   8000   15/08/85   3218.00   2574.40    643.60
00006   258   2000   15/08/85    355.45    355.45      0.00
00007   267   8201   15/08/85    698.88    559.10    139.78
00008   345   6100   15/08/85   2005.67   2005.67      0.00
00009   357   6100   15/08/85   2111.00   1688.80    422.20
00010   369   8000   15/08/85    654.00    654.00      0.00
00011   421   8500   15/08/85    761.00    608.80    152.20
00012   741   8630   15/08/85   2587.00      0.00   2587.00
00013   761   1000   15/08/85   3005.55   2404.44    601.11
00014   852   6850   15/08/85   1200.00      0.00   1200.00
00015   963   1000   15/08/85   1591.50      0.00   1591.50
```

Bild 2.35: Sortieren nach Kundennummer

Bitte beachten Sie die neue Zuordnung zwischen den Datensätzen und ihren laufenden Satznummern (RECORD #). Zum Beispiel hatte der Datensatz, der jetzt an erster Stelle steht, zuvor die Nummer 00004 usw. Da die Satznummer nur eine durchlaufende Numerierung ist, die nichts mit dem Satzinhalt zu tun hat, ist die Neuzuordnung eigentlich selbstverständlich, wird aber in der Praxis manchmal übersehen. Das kann dann zu Mißverständnissen führen, wenn Datensätze über die Satznummer aufgerufen werden sollen, wie etwa bei dem Befehl EDIT.

Gelegentlich ist es notwendig, eine Datei nach zwei oder mehr Kriterien zu sortieren. Zum Beispiel ist es vorstellbar, daß die Kundendatei regional nach Ortschaften und innerhalb der Ortschaften nach Kundennummern sor-

tiert werden soll, damit die Kunden entsprechend der regionalen Verteilung Ihrer Verkäufer ebenfalls regional in Blöcken zusammengefaßt sind.

Diese Ordnung läßt sich durch zweimaliges Sortieren realisieren, indem man zuerst nach dem untergeordneten Schlüssel (Kundennummer) und dann nach dem übergeordneten (Ort) sortiert. Der Befehlsablauf ist ähnlich dem obigen einmaligen Sortieren: lediglich anstelle des Rückkopierens von Datei **temp** nach Datei **kunden** wird ein zweiter Sortierbefehl gesetzt, der von **temp** nach dem Feld **ort** in die Datei **kunden** sortiert. Diese Sortierung sollen Sie aber jetzt nicht durchführen, da es eine bessere Alternative gibt, die gleich vorgeführt wird.

2.4.2 Indizieren

Eine sortierte Datei repräsentiert immer nur eine Sortierordnung, auch wenn diese mit mehreren Schlüsseln zustande gekommen ist. Wenn die Kundendatei nach Orten und innerhalb der Orte nach Kundennummern sortiert ist, kann sie nicht gleichzeitig nur nach Kundennummern allein sortiert sein. Oder sie kann nicht gleichzeitig nach Umsätzen sortiert sein.

Selbstverständlich kann man für Auswertungen und Berichte jede beliebige Sortierung in eine temporäre Datei vornehmen. Aber diese sollte dann wirklich nur für eine einmalige Auswertung benutzt und nicht ständig parallel zur ursprünglichen Datei geführt werden, um unnötige bzw. sogar gefährliche Redundanzen in der Datenbank zu vermeiden. Die Bearbeitung einer Datenbank, d.h. das Aktualisieren der Daten hat stets in der "Stammdatei" zu erfolgen und nicht in einer temporären.

Dennoch lassen sich mehrere Sortierordnungen gleichzeitig verwalten, und zwar mit Hilfe sogenannter Indexdateien.

Eine Indexdatei ist gewissermaßen eine Teilkopie ihrer Stammdatei, die pro Satz nur noch ein einziges Datenfeld enthält und die nach diesem Feld, dem sogenannten Schlüssel sortiert ist. Bildlich gesprochen stellt die Indexdatei eine einzige sortierte Spalte aus der Stammdatei dar.

Das Wesentliche an der Indexdatei ist, daß sie Zeiger, englisch Pointer, enthält, die von dem einzelnen Feld der Indexdatei auf den zugehörenden Satz in der Stammdatei weisen und diesen bei Bedarf für eine Bearbeitung automatisch aufrufen. Wenn Sie also eine Indexdatei mit dem Befehl LIST betrachten, dann sehen Sie nicht nur die eine sortierte Spalte, sondern die

gesamte Stammdatei in dieser Sortierung. Und Sie können mit der gesamten Datei auch uneingeschränkt arbeiten.

Pro Datenbank bzw. Datei können in dBASE beliebig viele solcher Indexdateien erstellt werden, ja sogar mehrere für ein und dasselbe Feld, z.B. eine Indexdatei in aufsteigender, eine andere in absteigender Reihenfolge des Umsatzes. Damit können verschiedene Sortierordnungen gleichzeitig benutzt werden, ohne daß die Datei auch tatsächlich immer wieder neu sortiert werden muß.

Die allgemeine Form des Befehls zum Indizieren einer Datei lautet:

INDEX ON (Feldname) TO (Indexdatei)

Den Namen der Indexdatei können Sie wie bei jeder anderen Datei frei wählen, wobei Sie sogar denselben Namen wie für die Stammdatei nehmen könnten. dBASE versieht nämlich den Namen einer Indexdatei intern mit dem Zusatz **NDX** als Dateityp und kann daher selbst zwischen Stammdatei (Typ: DBF) und Indexdatei unterscheiden.

Zur Datei **kunden** soll nun eine Indexdatei mit Namen **kundort** angelegt werden, die ein Sortieren nach Orten (Postleitzahlen) ermöglicht. Dazu eröffnen Sie die Datei **kunden** und schreiben:

USE kunden

INDEX ON ort TO kundort

Danach erscheint die Systemnachricht:

00015 Sätze indiziert

Die Indexdatei wird eröffnet, indem man Stammdatei und Indexdatei gemeinsam wie folgt aufruft:

USE kunden **INDEX** kundort

Nun kann man die indizierte Datei (Bild 2.36) wie jede andere benutzen.

00013	761	1000	15/08/85	3005.55	2404.44	601.11
00015	963	1000	15/08/85	1591.50	0.00	1591.50
00003	147	2000	15/08/85	980.00	0.00	980.00
00006	258	2000	15/08/85	355.45	355.45	0.00
00004	159	6000	15/08/85	852.00	681.60	170.40
00008	345	6100	15/08/85	2005.67	2005.67	0.00
00009	357	6100	15/08/85	2111.00	1688.80	422.20
00014	852	6850	15/08/85	1200.00	0.00	1200.00
00002	123	8000	15/08/85	1234.00	1234.00	0.00
00005	234	8000	15/08/85	3218.00	2574.40	643.60
00010	369	8000	15/08/85	654.00	654.00	0.00
00001	098	8057	15/08/85	5612.89	4490.31	1122.58
00007	267	8201	15/08/85	698.88	559.10	139.78
00011	421	8500	15/08/85	761.00	608.80	152.20
00012	741	8630	15/08/85	2587.00	0.00	2587.00

Bild 2.36: Die nach **Orten** indizierte Datei **kunden**

Mit Hilfe der Indexdatei **kundort** wurde eine Sortierung nach Orten bzw. Postleitzahlen und innerhalb gleicher Orte nach Kundennummern erreicht, wobei in der Datei **kunden** die alte Sortierung nach Kundennummern allein nach wie vor verfügbar ist. Das läßt sich auch daraus entnehmen, daß die laufenden Satznummern noch denselben Datensätzen zugeordnet sind wie vorher. Zum Beispiel steht Satz # 00001 des Kunden 098 in der Indexdatei an zwölfter Stelle.

Dies zu beachten ist auch wichtig bei der Anwendung mancher Befehle wie GO TOP oder GO BOTTOM. In der Datei **kunden** weist der Befehl GO BOTTOM auf den letzten, den 15. Datensatz. In der Indexdatei **kundort** dagegen weist derselbe Befehl GO BOTTOM auf den letzten Satz in deren Reihenfolge, das ist Satz # 00012 mit der (höchsten vorkommenden) Postleitzahl 8630.

2.4.3 Suchen in Indexdateien

Legen Sie nun noch eine zweite Indexdatei mit Namen **kundnum** an, die eine Indizierung nach Kundennummern enthält und rufen Sie diese neue Indexdatei anschließend auf:

```
USE kunden

INDEX ON kunummer TO kundnum

USE kunden INDEX kundnum
```

Wenn Sie diese Datei mit dem Befehl LIST betrachten, werden Sie feststellen, daß sie äußerlich identisch mit der Stammdatei **kunden** ist, da jene bereits schon nach Kundennummern sortiert war. Tatsächlich gibt es aber Unterschiede zwischen beiden Dateien, weswegen ein Indizieren durchaus sinnvoll ist. Der wohl wichtigste Unterschied ist die Existenz des Befehls

FIND (Suchbegriff) (= finden)

der für sehr schnelle Suchvorgänge in Indexdateien geeignet ist und insbesondere in großen Dateien bevorzugt angewandt wird. Mit FIND werden die indizierten Felder einer Datei auf den Suchbegriff, der auch als Schlüsselwert bezeichnet wird, abgefragt. Die Suche wird beim ersten auftretenden Datensatz, der im indizierten Feld den Schlüsselwert enthält, beendet. Anschließend kann der Satz bearbeitet werden.

Die Suche mit FIND ist deswegen außerordentlich schnell, weil pro Datensatz nur das eine indizierte Feld abgefragt wird und nicht der gesamte Satz wie etwa bei der Suche mit dem Befehl LOCATE. (Für die Experten unter Ihnen: dBASE sucht in Indexdateien nach dem B-Baum-Prinzip.) Typische Suchzeiten mit FIND liegen bei ein bis zwei Sekunden, auch bei sehr grossen Dateien. Bei kleinen Dateien, wie den in diesen Beispielen benutzten, ist allerdings noch kein Unterschied zu den anderen Suchmethoden zu bemerken.

Um beispielsweise den Datensatz des Kunden Nr.123 zu bearbeiten, ist lediglich der Befehl

FIND 123

zu geben. Mit Erscheinen des "Punktes" ist der Satz bereits im Zugriff und kann bearbeitet oder abgefragt werden, etwa mit

? umsatz,offen

worauf die Antwort erscheint:

1234.00 0.00

Mit Hilfe der sogenannten Makro-Funktion &, die später noch mehrmals verwendet wird, kann man nach Datensätzen suchen, deren Schlüsselwert in einem Speicher steht:

FIND &(Speichername)

Diese Variante wird häufig in Programmen eingesetzt, wenn derselbe Suchbefehl mehrfach auf verschiedene Objekte angewandt werden soll.

Im Unterschied zum Befehl LOCATE beginnt ein Suchen mit FIND stets am Dateianfang und endet beim ersten gefundenen Satz. Vor dem Suchen muß also nicht erst mit GO TOP der Dateianfang erreicht werden. Andererseits ist eine Fortsetzung des Suchens mit CONTINUE bei FIND nicht möglich, kann aber auf andere Weise erzielt werden, wofür später noch ein Beispiel gegeben wird.

2.5 Zusammenfassende Auswertungen und Berichte

Sie haben bereits einige Befehle kennengelernt, mit denen man gezielt auf Informationen aus der Datenbank zugreifen kann. Dies waren die Befehle LIST, LOCATE und DISPLAY in ihren verschiedenen Variationen. Mit ihnen wurden auch die wichtigsten Abfragetechniken besprochen. Alle diese Befehle beziehen sich auf die Darstellung ausgewählter Datensätze, so, wie sie in der Datenbank gespeichert sind.

Daneben bieten insbesondere relationale Datenbanken die Möglichkeit, Dateien als Ganzes auszuwerten, indem diese nach bestimmten Kriterien verdichtet oder Felder über die ganze Datei hinweg summiert werden.

Zusätzlich verfügt dBASE über einen integrierten Listengenerator (auch Reportgenerator genannt), ein Unterprogramm, mit dessen Hilfe komplette Berichte inklusive Überschriften, Summen und Zwischensummen erstellt und gedruckt werden können, die in vielen Fällen dem Anwender spezielle Programme zur Datenauswertung ersparen.

2.5.1 Summenbildung über die gesamte Datei

SUM (Feldnamen)

Mit diesem Befehl werden die Inhalte aller gleichnamigen numerischen Felder (also die Inhalte einer numerischen Spalte) einer Datei summiert. Wenn Sie die Datei **kunden** im Zugriff haben, können Sie mit

SUM umsatz

den Gesamtumsatz errechnen, den Sie mit allen Kunden zusammen erreicht haben und erhalten das Ergebnis: 26866.94.

Entsprechend ergibt

SUM offen

die Summe aller Außenstände: 9610.37.

Es können auch gleichzeitig mehrere Felder summiert werden, wie zum Beispiel:

```
SUM umsatz,bezahlt,offen
          26866.94  17256.57  9610.37
```

Ferner können für die Summenbildung bestimmte Bedingungen vorgegeben werden, die zu einer selektiven Addition von Daten führen. Dazu wird der Befehl SUM um die bekannte FOR-Klausel erweitert:

SUM (Feldnamen) FOR (Bedingung)

Dann werden nur diejenigen Datensätze in der Summe erfaßt, für die eine gegebene Bedingung zutrifft, zum Beispiel

```
SUM umsatz FOR ort = '8'
```

ergibt den Gesamtumsatz mit allen Kunden aus dem Postleitzahlbereich 8 (Bayern). Ergebnis: 14765.77.

```
SUM offen FOR kunummer > '500'
```

ergibt die Summe aller Außenstände der Kunden, deren Kundennummer größer als 500 ist. Ergebnis: 5979.61.

Schließlich ist es möglich, das Rechenergebnis in einem separaten Variablenspeicher abzulegen und bei Bedarf wieder aufzurufen. dBASE II besitzt 64 solcher Speicher, die entweder Zahlenwerte oder alphanumerischen Text von maximal 254 Zeichen Länge aufnehmen können. Sie werden einzeln durch Benennen mit einem Namen definiert und adressiert.

Um das Ergebnis einer Summierung zu speichern, wird der Befehl SUM um den Zusatz TO (Speichername) erweitert:

SUM (Feldname) TO (Speichername)

Der Name des Speichers kann frei definiert werden, jedoch ist es zweckmäßig, den Namen so zu wählen, daß er nicht nur den Speicherinhalt charakterisiert, sondern auch die Unterscheidung zu Feldnamen ermöglicht. Für die Summen der Kundenumsätze etc. sind Speichernamen wie Mumsatz, Moffen etc. zweckmäßig, wobei das vorangestellte M (=Memory) anzeigt, daß es sich um den Namen eines Speichers handelt.

Geben Sie folgende Befehle:

SUM umsatz TO Mumsatz

SUM bezahlt TO Mbezahlt

SUM offen TO Moffen

Nach jedem Befehl wird die Summe auf dem Bildschirm angezeigt und gleichzeitig in dem benannten Speicher abgelegt. Von dort kann sie über den Speichernamen jederzeit wieder abgerufen und verarbeitet werden, allerdings nur solange, wie das Programm dBASE aktiv bleibt. Speicherinhalte werden nämlich nicht auf die Diskette geschrieben, sondern nur im Hauptspeicher des Rechners gehalten und daher mit Verlassen von dBASE gelöscht. Für die Fälle, in denen Inhalte von Speichern für spätere Programmläufe aufbewahrt werden sollen, gibt es spezielle Befehle zur Speicherung auf der Diskette, auf die an anderer Stelle noch eingegangen wird.

2.5.2 Zählen von Datensätzen

COUNT

ist eine weitere nützliche Funktion zur Auswertung einer Datei, mit der die Anzahl der Datensätze einer Datei ermittelt wird. Die in Zusammenhang mit SUM besprochenen Zusätze und Bedingungen sind auf den Befehl COUNT ebenfalls anwendbar. Das Zählergebnis wird auf dem Bildschirm mit "Anzahl der Sätze = ..." angezeigt.

Schreiben Sie folgende Beispiele für die Datei **kunden**:

COUNT	Ergebnis:	Anzahl der Sätze = 00015
COUNT FOR offen > 0		Anzahl der Sätze = 00011
COUNT FOR bezahlt = 0		Anzahl der Sätze = 00004
COUNT FOR ort = '8000'		Anzahl der Sätze = 00003
COUNT to kundzahl		Anzahl der Sätze = 00015

Im letzten Fall wird die Anzahl der Kunden (=Anzahl der Datensätze) in den Speicher mit Namen **kundzahl** geschrieben.

2.5.3 Speichern einzelner Daten

Speicher können auch direkt über die Tastatur mit Daten belegt werden. Hierzu verfügt dBASE über mehrere Befehle, von denen allerdings die meisten zur programmgesteuerten Dateneingabe während eines Programmlaufes dienen und daher später zu besprechen sind. Einer dieser Befehle soll jedoch schon hier erwähnt werden, nämlich:

STORE (Ausdruck) TO (Speichername)

STORE schreibt einen Ausdruck numerischer oder alphanumerischer Art in den zu benennenden Speicher. Damit können beispielsweise Konstanten oder erläuternder Text, die häufiger gebraucht werden, gespeichert, verarbeitet und wieder ausgegeben werden.

Um aus der Datei **kunden** den mittleren Umsatz pro Kunde zu berechnen und mit einer kurzen Erklärung anzuzeigen, schreiben Sie zunächst:

```
STORE 'mittlerer Umsatz pro Kunde' TO hinweis
```

Der mittlere Umsatz errechnet sich aus dem Gesamtumsatz dividiert durch die Anzahl der Kunden. Vor das Rechenergebnis soll der kurze Text zu stehen kommen, so daß der Befehl an dBASE lautet:

```
? hinweis,Mumsatz/kundzahl
```

Als Ergebnis erscheint auf dem Bildschirm:

```
mittlerer Umsatz pro Kunde    1791.12
```

Mit STORE können auch Zahlenwerte als Konstanten gespeichert und bei Bedarf in eine Berechnung einbezogen werden. Beispiel:

```
STORE 100 TO prozent

? Moffen/Mumsatz*prozent                    (Ergebnis: 35.77)
```

Die Namen und der Inhalt aller definierten Speicher können mit dem Befehl

DISPLAY MEMORY

angezeigt bzw. ausgedruckt werden. Gleichzeitig wird der jeweilige Datentyp, numerisch oder alphanumerisch, angezeigt und eine Summe aller definierten Speicher gebildet: Insgesamt sind sechs Speicher, die zusammen 56 Zeichen (Bytes) lang sind, definiert (Bild 2.37).

```
MBEZAHLT     (N)   17256.57
MOFFEN       (N)    9610.37
KUNDZAHL     (N)      15
HINWEIS      (C)   mittlerer Umsatz pro Kunde
PROZENT      (N)      100
MUMSATZ      (N)   26866.94
** Gesamt **        06 Variablen benutzt  00056 Bytes belegt
```

Bild 2.37: Die Darstellung von Speicherinhalten

2.5.4 Datensätze zusammenfassen

Eine wichtige Methode der Dateiauswertung besteht darin, Datensätze nach bestimmten Kriterien gruppenweise zusammenzufassen und deren numerische Werte zu summieren. Mit dem Befehl

TOTAL ON (Feldname) TO (Dateiname)

werden alle Sätze, die in dem spezifizierten Feld (Feldname) den gleichen Schlüsselwert besitzen, zusammengefaßt, die numerischen Felder aller dieser Sätze addiert und die zusammengefaßten Sätze in eine neue Datei (Dateiname) übernommen. Die Struktur der Zieldatei kann, muß aber nicht vorher festgelegt werden. Die ursprüngliche Datei bleibt auch nach Ausführung des Befehls erhalten. Das klingt komplizierter als es tatsächlich ist. Ein Beispiel:

Eröffnen Sie die nach Orten indizierte Kundendatei und geben Sie den Befehl, die Datensätze mit jeweils gleichem Ort zusammenzufassen:

USE kunden INDEX kundort

TOTAL ON ort TO orte

dBASE baut nun eine neue Datei mit Namen **orte** auf, in der alle Kunden mit gleichem Wohnsitz zusammengefaßt sind. Wenn der Befehl zu Ende geführt ist, erscheint die Nachricht

00010 Sätze kopiert

Die ursprünglich 15 Datensätze sind auf 10 komprimiert worden, wobei jetzt pro Ort nur noch ein Datensatz vorhanden ist, dessen numerische Werte Umsatz etc. die Summe aller Kundenumsätze etc. dieses Ortes darstellen.

Mit den Befehlen

USE orte

LIST

können Sie nun die neue Datei **orte** aufrufen und betrachten (Bild 2.38).

00001	761	1000	15/08/85	4597.05	2404.44	2192.61
00002	147	2000	15/08/85	1335.45	355.45	980.00
00003	159	6000	15/08/85	852.00	681.60	170.40
00004	345	6100	15/08/85	4116.67	3694.47	422.20
00005	852	6850	15/08/85	1200.00	0.00	1200.00
00006	123	8000	15/08/85	5106.00	4462.40	643.60
00007	098	8057	15/08/85	5612.89	4490.31	1122.58
00008	267	8201	15/08/85	698.88	559.10	139.78
00009	421	8500	15/08/85	761.00	608.80	152.20
00010	741	8630	15/08/85	2587.00	0.00	2587.00

Bild 2.38: Zusammenfassung der Kundendatei nach Orten

Die Datei **orte** zeigt - nach Postleitzahlen geordnet - die Umsätze, bezahlten Summen und Außenstände pro Ort. Da eine Dateistruktur für **orte** zuvor noch nicht definiert war, wurde die vollständige Struktur der Datei **kunden** übernommen, d.h. inklusive der Felder für Kundennummer und Datum. Selbstverständlich haben die noch enthaltenen Kundennummern und auch das jeweilige Datum hier keine Bedeutung mehr, da die Beträge ja die Summen von mehreren Kunden darstellen.

Man kann die Kundennummern und das Datum unterdrücken, wenn man vor der TOTAL-Bildung die Struktur für die Datei **orte** ohne diese beiden Felder festlegt. Bei großen Dateien ist das in der Praxis zweckmäßig, um Speicherplatz und Rechenzeit zu sparen. Bei kleinen Dateien ist es meistens ausreichend, wenn man die überflüssigen bzw. sinnlos gewordenen Felder erst bei der Anzeige oder beim Drucken eliminiert, indem man den Befehl LIST selektiv anwendet (Bild 2.39).

Um den Befehl TOTAL korrekt anzuwenden, muß die Datei nach dem für die Zusammenfassung gewünschten Feld (hier: **ort**) vorsortiert oder indiziert sein. Daher wurde für das eben angeführte Beispiel die Indexdatei **kundort** benutzt.

```
LIST ort,umsatz,bezahlt,offen

00001    1000   4597.05   2404.44   2192.61
00002    2000   1335.45    355.45    980.00
00003    6000    852.00    681.60    170.40
00004    6100   4116.67   3694.47    422.20
00005    6850   1200.00      0.00   1200.00
00006    8000   5106.00   4462.40    643.60
00007    8057   5612.89   4490.31   1122.58
00008    8201    698.88    559.10    139.78
00009    8500    761.00    608.80    152.20
00010    8630   2587.00      0.00   2587.00
```

Bild 2.39: Eliminieren von überflüssigen Feldern aus der Zusammenfassung

2.5.5 Der integrierte Listengenerator

Eine der wesentlichen Aufgaben eines Datenbanksystems besteht darin, die gespeicherten Daten dem Benutzer in geordneter Form wieder zur Verfügung zu stellen. Deshalb ist das Festlegen von Drucktabellen und das Erstellen entsprechender Druckprogramme eine wichtige, wenngleich bei vielen Systemen aufwendige Arbeit.

Eine geordnete, aber sehr einfache Darstellung einer Datei wurde bislang mit dem Befehl LIST erreicht. Allerdings kann damit eine Datei nur in ihrer augenblicklichen Struktur und nur mit den Daten, die tatsächlich in dieser Datei explizit enthalten sind, dargestellt werden. Andere Informationen wie Überschriften, Summen, Zwischensummen oder andere errechnete Werte sind in der Datei nicht enthalten und müssen folglich bei Bedarf über ein zu schreibendes Auswerte- und Druckprogramm bereitgestellt werden.

Diese Arbeit nimmt Ihnen dBASE weitgehend ab, wenn Sie sich des integrierten Listengenerators bedienen. Dies ist ein Unterprogramm, das durch eine Reihe gezielter Fragen von Ihnen die Informationen erhält, die zur Definition der Drucktabelle gebraucht werden. Aufgrund Ihrer Angaben wird dann von dBASE ein Auswerte- und Druckprogramm erstellt und auf der Diskette gespeichert, das dann beliebig oft benutzt werden kann. In der Praxis lassen sich die meisten Drucktabellen über diesen Reportgenerator erstellen. Nur in relativ wenigen Fällen werden spezielle Druckprogramme zu schreiben sein. Nachfolgend soll an Hand mehrerer Beispiele gezeigt werden, wie mit dem Listengenerator praktisch gearbeitet wird.

Bevor der Reportgenerator abgerufen wird, muß stets diejenige Datei eröffnet sein, auf die das Auswerteprogramm zugreifen soll. Zunächst soll die Kundendatei in ihrer bekannten Form ausgedruckt werden, allerdings mit Überschriften und mit einer Summe für die numerischen Spalten versehen.

Rufen Sie die Datei **kunden** auf:

```
USE kunden
```

Der Befehl zum Aufruf des Listengenerators lautet in seiner allgemeinen Form:

REPORT FORM (Listenname)

Hinter dem Wort FORM (= Formular) ist ein von Ihnen zu bestimmender Name anzugeben, unter dem die Listenstruktur (das "Formular") als Datei gespeichert und bei Bedarf wieder abgerufen wird. Bezüglich des Namens haben Sie wieder völlig freie Wahl, doch sollte der Name wie immer auf den Inhalt Bezug nehmen. Zur Unterscheidung dieses Dateityps von den anderen fügt dBASE intern die Typenbezeichnung **FRM** für FORM an den Dateinamen an.

An den Befehl REPORT FORM (Listenname) kann der Ausdruck **TO PRINT** angehängt werden, der bewirkt, daß die Liste auf den Drucker ausgegeben wird, unabhängig davon, ob dieser mit CTRL-P schon eingeschaltet war oder nicht. Dies sollten Sie immer tun, wenn der Report gedruckt werden soll. Denn mit zunehmender Übung in dBASE werden Sie immer seltener die eingegebenen Befehle auf dem Drucker protokollieren und dann werden Sie ab und zu vergessen, diesen mit CTRL-P recht-zeitig zuzuschalten.

Die erste Liste über die Daten der Kundendatei soll **umsatz1** genannt werden. Schreiben Sie daher:

REPORT FORM umsatz1 TO PRINT

Damit ist eine Report-Datei mit Namen **umsatz1** definiert, deren Inhalt nun durch eine Reihe Fragen von dBASE und Ihre Antworten generiert wird. Die erste Fragezeile lautet:

Bitte angeben: m=linker Rand, l=Zeilen/Seite, w=Zeilenbreite

Hier können Sie einige Optionen zum Formatieren einer Druckseite eingeben, wenn Sie von dem abweichen wollen, was dBASE als Standardformatierung (sogenannte Default-Werte) festgelegt hat, und zwar:

m=linker Rand	linker Rand bis zur ersten Druckposition Default: M = 8 Zeichen
l=Zeilen/Seite	nutzbare Druckzeilen pro Seite Default: L = 57 Zeilen
w=Zeilenbreite	Formularbreite bzw. Zeilenlänge Default: W = 80 Zeichen/Zeile

Die Angabe Zeilenbreite dient ausschließlich zum Zentrieren einer etwaigen Überschrift und stellt keine Festlegung der maximal möglichen Druckbreite dar. Da die Zeilen der folgenden Listen maximal 65 Zeichen lang sein werden, geben Sie den Wert W = 65 ein und drücken die ENTER-Taste.

Daraufhin erscheint die nächste Frage:

Seitenüberschrift? (J/N)

Sie antworten mit J für Ja und können in der nächsten Zeile die Überschrift eintragen:

Bitte Seitenüberschrift eingeben:

Schreiben Sie dahinter: Umsatzübersicht nach Kunden

Als nächstes wird gefragt, ob die Liste mit doppeltem Zeilenabstand gedruckt werden soll, was Sie mit N für Nein beantworten:

Leerzeile zwischen den Sätzen? (J/N)

Zwei weitere Fragen beziehen sich auf Gesamtsummen und Zwischensummen. Zunächst wird gefragt, ob Summen überhaupt erwünscht sind:

Gesamtsummen erforderlich? (J/N)

Summen können für alle numerischen Werte gebildet werden. Nachdem in der Datei **kunden** numerische Felder (Umsatz etc.) vorkommen, die eventuell summiert werden sollen, beantworten Sie die Frage mit **Y** für **Yes** (=Ja). Welche der numerischen Felder tatsächlich summiert werden, wird weiter unten im Einzelfall festgelegt.

Achtung: Hier hat sich bei der Anpassung an den Schneider CPC ein Fehler eingeschlichen. Daher erwartet dBASE die englische Antwort **Yes** anstatt Ja. Eventuell ist der Fehler aber in Ihrer Version schon behoben.

Wenn Summen gebildet werden sollen, und nur dann, fragt dBASE weiter, ob auch Zwischensummen im Report verlangt sind:

Bericht mit Zwischensummen? (J/N)

Diese Frage soll verneint werden (Antwort: N).

Nach diesen einleitenden Fragen wird jetzt das eigentliche Druckformular im Dialog definiert. Prinzipiell geht dBASE davon aus, daß die Liste in Spalten unterteilt ist und ruft daher die Spalten nacheinander zur Festlegung ihres Formats und ihres Inhalts auf:

```
Spalte  Breite Inhalt
001
```

Die einzelnen Spalten werden während ihrer Definition fortlaufend numeriert, wobei Spalte 001 die erste Spalte von links ist. Pro Spalte werden drei Angaben benötigt:

Breite	Sie kann eine beliebige Anzahl Zeichen von einnehmen, muß aber mindestens so groß sein, daß der vorgesehene Inhalt darin Platz findet.
Inhalt	Dies kann der Name für ein Datenfeld oder der Name eines Speichers oder ein Ausdruck, z.B. eine Formel sein.
Überschrift	Sie wird nach den beiden obigen Angaben in einer separaten Zeile definiert.

In die erste Spalte soll die Kundennummer eingetragen werden. Die Spaltenbreite muß mindestens drei Zeichen betragen, damit die Kundennummer Platz findet, kann aber, wie bereits gesagt, auch größer sein. Mit Rücksicht auf die noch festzulegende Spaltenüberschrift wählen Sie die Breite 10 und schreiben nach einem Komma den Feldnamen **kunummer**, also:

```
Spalte  Breite Inhalt
001     10,kunummer
```

In die nächste Zeile ist die Überschrift für dieses Feld, die "Kunden-Nr." heißen soll, einzutragen:

```
Bitte Überschrift eingeben: Kunden-Nr.
```

Damit ist die erste Spalte definiert. Alle weiteren Spalten werden analog festgelegt. In Spalte 2 mit 15 Zeichen Breite soll der Umsatz aufgeführt werden:

```
002  15,umsatz
```

Die zugehörige Überschrift "Umsatz 85" ist kürzer als 15 Zeichen und wird von dBASE in die Mitte der Spalte plaziert, wenn nichts anderes vorgegeben wird. Da die Umsatzzahlen dieses Beispiels wesentlich kleiner als 15 Stellen sind und rechtsbündig geschrieben werden, ist es zwechmäßig, auch die Überschrift rechtsbündig zu schreiben. Das ist möglich, wenn Sie vor den Text der Überschrift das Zeichen > setzen:

Bitte Überschrift eingeben: >Umsatz 85

Analog erlaubt das Zeichen < eine linksbündige Überschrift.

Nachdem es sich bei dem Feld **umsatz** um ein numerisches Feld handelt, wird jetzt die zusätzliche Frage gestellt, ob für dieses individuelle Feld eine Summe zu bilden ist:

Gesamtsummen erforderlich? (J/N)

Antworten Sie mit J und definieren Sie zwei weitere Spalten für die bezahlten Beträge und für die Außenstände. Die Spaltenbreiten sollen 15 bzw. 20 Zeichen betragen, die Überschriften wieder rechtsbündig angeordnet sein und in beiden Fällen Summen gebildet werden:

003 15,bezahlt
Bitte Überschrift eingeben: >davon bezahlt
Gesamtsummen erforderlich? (J/N) J
004 20,offen
Bitte Überschrift eingeben: >Aussenstaende
Gesamtsummen erforderlich? (J/N) J
005

Mehr Spalten werden in diesem Report nicht benötigt. Die Definition des Druckformulars schließen Sie ab, indem Sie zu Beginn der nächsten Spaltendefinition sofort die ENTER-Taste betätigen. Unmittelbar anschliessend erfolgt die Verarbeitung der Datei und das Generieren der Liste (Bild 2.40). Zu Beginn des Reports erscheint eine Seitennummer und das aktuelle Datum, das Sie beim Start von dBASE eingegeben hatten.

Wie bereits erwähnt, wird das eben definierte Listenformular als eine Report-Datei gespeichert und kann jederzeit für eine erneute Auswertung aufgerufen werden. Dazu werden wieder die beiden Befehle zum Eröffnen der Datei und zum Aufruf des Reportgenerators gegeben:

```
USE kunden

REPORT FORM umsatz1 TO PRINT
```

Seitennr. 00001
15/08/85

Umsatzübersicht nach Kunden

Kunden-Nr.	Umsatz 85	davon bezahlt	Aussenstaende
098	5612.89	4490.31	1122.58
123	1234.00	1234.00	0.00
147	980.00	0.00	980.00
159	852.00	681.60	170.40
234	3218.00	2574.40	643.60
258	355.45	355.45	0.00
267	698.88	559.10	139.78
345	2005.67	2005.67	0.00
357	2111.00	1688.80	422.20
369	654.00	654.00	0.00
421	761.00	608.80	152.20
741	2587.00	0.00	2587.00
761	3005.55	2404.44	601.11
852	1200.00	0.00	1200.00
963	1591.50	0.00	1591.50
** Gesamt **			
	26866.94	17256.57	9610.37

Bild 2.40: Die mit dem Reportgenerator erstellte Umsatzübersicht je Kunde

Wenn die Report-Datei bereits existiert, entfällt der Definitionsteil und die Liste wird sofort erstellt. Nach einer Aktualisierung der Datenbank kann also direkt eine neue Liste ausgedruckt werden.

Wie Sie vielleicht bemerkt haben, enthält die Beschreibung des Listenformulars nur Namen von Datenfeldern, aber nicht den Namen der Datei selbst. Das bedeutet, daß die Erstellung des Reports nicht an eine bestimmte Datei gebunden ist, sondern daß alle Dateien, die die benötigten Felder enthalten, als Basis für diese Liste dienen können. In der Praxis sind dies

insbesondere die Indexdateien, die ja bis auf die individuelle Sortierung mit der Stammdatei identisch sind. So können mit einer Report-Datei und verschiedenen Indexdateien verschieden sortierte Listen erstellt werden.

Im folgenden Bericht sollen die Finanzzahlen für die Kunden nach Ortschaften sortiert und pro Ort als Zwischensummen ausgewiesen werden. Ferner sollen die jeweiligen Außenstände als Prozentsatz der Gesamtsumme aller Außenstände angegeben werden.

Da eine Sortierung nach Orten gewünscht ist, wird die Indexdatei **kundort** als Ausgangsdatei herangezogen:

```
USE kunden INDEX kundort
```

Als Basis für die prozentualen Werte der Außenstände ist die Summe aller Felder **offen** zu bilden und in einem Speicher bereitzuhalten. Schreiben Sie daher:

```
SUM offen TO Moffen
```

Die Definition der Drucktabelle (Bild 2.41) läuft ähnlich zum ersten Beispiel ab und soll daher nicht noch einmal beschrieben werden. Lediglich die neu hinzugekommenen Schritte werden näher erläutert.

Als Option wurde zusätzlich zur Formularbreite die Anzahl der Zeilen pro Seite auf L = 48 begrenzt, was vor allem mit Rücksicht auf die Darstellung in diesem Buch geschah, da die zu erwartende Liste länger als eine Buchseite sein wird.

Neu in diesem Report ist, daß jetzt auch Zwischensummen zu berechnen sind. Die Frage "Bericht mit Zwischensummen?" wird daher mit J beantwortet und löst einige weitere Fragen zu den Zwischensummen aus:

Bitte Feldnamen für Zwischensumme eingeben:

Hier wird das Feld angegeben, für das die Zwischensummen zu bilden sind. Nachdem die Umsatzzahlen nach Orten zusammengefaßt werden sollen, ist das Feld **ort** anzugeben. Ähnlich wie schon bei dem Befehl TOTAL werden alle Datensätze in der Zwischensumme erfaßt, deren Feld **ort** den gleichen Inhalt hat.

```
. Report FORM umsatz2 TO PRINT

Bitte angeben: m=linker Rand, l=Zeilen/Seite, w=Zeilenbreite
Mit Seitenüberschrift? (J/N) J
Bitte Seitenüberschrift eingeben: Umsatz pro Kunde, sortiert nach Orten
Leerzeile zwischen den Sätzen? (J/N) J
Gesamtsummen erforderlich? (J/N) J
Bericht mit Zwischensummen? (J/N) J
Bitte Feldnamen für Zwischensummen eingeben: ort
Nur Zusammenfassung des Berichts? (J/N) N
Seitenvorschub nach Zwischensummen? (J/N) N
Bitte Überschrift für Zwischensumme eingeben: PLZ
Spalte Breite Inhalt
001    10,kunummer
Bitte Überschrift eingeben: Kunden-Nr.
002    15,umsatz
Bitte Überschrift eingeben: >Umsatz 85
Gesamtsummen erforderlich? (J/N) J
003    15,offen
Bitte Überschrift eingeben: >Aussenstaende
Gesamtsummen erforderlich? (J/N) J
004    15,offen/Moffen*100
Bitte Überschrift eingeben: in Prozent aller offenen Posten
Gesamtsummen erforderlich? (J/N) J
005
```

Bild 2.41: Die Definition des Reports "umsatz2"

Wenn Zwischensummen gebildet werden, ist es möglich, nur diese und die Endsumme im Bericht erscheinen zu lassen und die Einzelposten zu unterdrücken. Daher wird jetzt die Frage

Nur Zusammenfassung des Berichts? (J/N)

gestellt, die Sie aber mit N beantworten. Ferner kann beim Drucken nach jeder Zwischensumme eine neue Seite begonnen werden:

Seitenvorschub nach Zwischensummen? (J/N)

Auch diese Frage wird verneint.

Schließlich haben Sie die Möglichkeit, für die einzelnen Gruppen, die mit einer Zwischensumme enden, jeweils eine Überschrift zu setzen.

Bitte Überschrift für Zwischensumme angeben:

Verzichten Sie darauf, indem Sie sofort ENTER drücken, dann wird zu Beginn jeder Gruppe lediglich der für diese Gruppe gemeinsame Ort, das heißt die jeweilige Postleitzahl ausgegeben. Andernfalls wird bei Eingabe einer Gruppenüberschrift diese von dBASE unmittelbar vor den Feldinhalt gesetzt und kann diesen somit erläutern. Die Überschrift "PLZ" als Abkürzung für Postleitzahl soll im Bericht jeweils eine neue Gruppe ankündigen.

Die ersten drei Spalten der Liste sind analog zum vorherigen Beispiel aufgebaut. In Spalte 004 wird statt eines Feldes eine Formel als Inhalt angegeben:

```
004     15,offen/Moffen*100
```

Die Formel besagt, daß der Betrag des Feldes **offen** durch die im Speicher **Moffen** stehende Summe aller Außenstände geteilt und das Ergebnis mit 100 multipliziert werden soll, daß also der prozentuale Anteil der einzelnen Außenstände zu ermitteln ist.

Bei der Erstellung der Liste wird diese Formel für jeden Datensatz berechnet. Dies ist natürlich nur möglich, wenn die Variable **Moffen** zuvor definiert wurde, weshalb dieser Wert bereits vor Beginn der Reportgenerierung zu berechnen war. Andernfalls würde das anschließende Erstellen der Liste mit einer Fehlermeldung abgebrochen werden.

Nach Abschluß der Definition wird auch diese Liste sofort bearbeitet und ausgedruckt (Bild 2.42). Sie sehen, daß sich der Bericht über zwei Seiten erstreckt, wobei jede Seite numeriert und mit Datum versehen ist. Ebenso werden die Überschriften auf jeder Seite wiederholt.

Die Spaltenüberschrift "in Prozent aller offenen Posten" ist länger als die Spalte breit ist und wird daher von dBASE auf drei Zeilen verteilt. Dabei nimmt dBASE aufgrund der Zwischenräume zwischen den Worten selbständig eine Unterteilung vor und zentriert jede einzelne Zeile in die Mitte der Spalte.

Seitennr. 00001
15/08/85

Umsatz pro Kunde, sortiert nach Orten

Kunden-Nr.	Umsatz 85	Aussenstaende	in Prozent aller offenen Posten
* PLZ 1000			
761	3005.55	601.11	6.25
963	1591.50	1591.50	16.56
** Zwischensumme **			
	4597.05	2192.61	22.81
* PLZ 2000			
147	980.00	980.00	10.19
258	355.45	0.00	0.00
** Zwischensumme **			
	1335.45	980.00	10.19
* PLZ 6000			
159	852.00	170.40	1.77
** Zwischensumme **			
	852.00	170.40	1.77
* PLZ 6100			
345	2005.67	0.00	0.00
357	2111.00	422.20	4.39
** Zwischensumme **			
	4116.67	422.20	4.39
* PLZ 6850			
852	1200.00	1200.00	12.48
** Zwischensumme **			
	1200.00	1200.00	12.48
* PLZ 8000			
123	1234.00	0.00	0.00
234	3218.00	643.60	6.69
369	654.00	0.00	0.00
** Zwischensumme **			
	5106.00	643.60	6.69

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Seitennr. 00002
15/08/85

          Umsatz pro Kunde, sortiert nach Orten

Kunden-Nr.          Umsatz 85    Aussenstaende   in Prozent
                                                 aller offenen
                                                    Posten
* PLZ 8057
098                  5612.89         1122.58         11.68
** Zwischensumme **
                     5612.89         1122.58         11.68

* PLZ 8201
267                   698.88          139.78          1.45
** Zwischensumme **
                      698.88          139.78          1.45

* PLZ 8500
421                   761.00          152.20          1.58
** Zwischensumme **
                      761.00          152.20          1.58

* PLZ 8630
741                  2587.00         2587.00         26.91
** Zwischensumme **
                     2587.00         2587.00         26.91

** Gesamt **
                    26866.94         9610.37         99.99
```

Bild 2.42: Report mit Zwischensummen

Man kann Reports auch nach Kriterien zusammenfassen, die nicht explizit, sondern nur indirekt als Teil eines Datenfeldes vorkommen. Zum Beispiel enthält die Postleitzahl indirekt Informationen über die regionale Lage eines Ortes, insbesondere, wenn man die erste Stelle der Postleitzahl betrachtet.

Als drittes Beispiel soll nun eine Übersicht erstellt werden, die eine Umsatzverteilung nach diesen Regionen, den Postleitzahlbereichen 1 bis 8 zeigt. Zusätzlich sollen die regionalen Umsätze auch als Prozentwerte, bezogen auf den Gesamtumsatz, ermittelt werden.

Die Daten für die Liste liefert wieder die indizierte Kundendatei **kundort**:

```
USE kunden INDEX kundort
```

Der Gesamtumsatz als Bezugsgröße wird in den Speicher Mumsatz geschrieben:

```
SUM umsatz TO Mumsatz
```

Der prinzipielle Aufbau der Listenstruktur, die den Namen **umsatz3** erhalten soll (Bild 2.43), entspricht dem vorherigen Beispiel, jedoch sind wieder einige neue Formulierungen zu erklären.

Rufen Sie den Listengenerator auf mit:

```
REPORT FORM umsatz3 TO PRINT
```

Die folgende Liste wird relativ schmal sein. Daher wird für die Zentrierung der Überschrift die Formularbreite auf W = 42 festgelegt.

Überschriften in einem Report können, wie Sie bereits gesehen haben, mehr als eine Zeile einnehmen. Im Bericht **umsatz2** wurde eine Überschrift von dBASE auf drei Zeilen verteilt. Gelegentlich will der Benutzer aber selbst festlegen, welche Textteile in welcher Zeile zu stehen haben. Das ist möglich, wenn die einzelnen geplanten Zeilen durch ein Semikolon getrennt werden. Im vorliegenden Fall soll die Überschrift

```
Regionale Umsatzverteilung
```

unterstrichen werden, wozu nach der Überschrift ein Semikolon gesetzt und anschließend eine Reihe von Strichen eingegeben wird, deren Gesamtlänge der Überschrift entspricht:

Regionale Umsatzverteilung;------------------------

Beide Zeilen werden später im Bericht untereinander geschrieben und zentriert.

```
. Report FORM umsatz3 TO PRINT

Bitte angeben: m=linker Rand, l=Zeilen/Seite, w=Zeilenbreite
Mit Seitenüberschrift? (J/N) J
Bitte Seitenüberschrift eingeben: Regionale Umsatzverteilung;----
Leerzeile zwischen den Sätzen (J/N) N
Gesamtsummen erforderlich? (J/N) J
Bericht mit Zwischensummen? (J/N) J
Bitte Feldnamen für Zwischensumme eingeben: $(ort,1,1)
Nur Zusammenfassung des Berichts? (J/N) N
Seitenvorschub nach Zwischensummen? (J/N) N
Bitte Überschrift für Zwischensumme eingeben: Region
Spalte  Breite Inhalt
001     10,kunummer
Bitte Überschrift eingeben: Kunden-Nr.
002     15,umsatz
Bitte Überschrift eingeben: >Umsatz 85
Gesamtsummen erforderlich? (J/N) J
003     15,int(umsatz/mumsatz*100+0.5)
Bitte Überschrift eingeben: >in Prozent vom Gesamtumsatz
Gesamtsummen erforderlich? (J/N) J 004
```

Bild 2.43: Die Struktur des Reports **umsatz3**

Wesentlich an dieser Liste ist die Definition der Gruppen für die Zwischensummen. Wie bereits gesagt, sollen die Kundenumsätze regional zusammengefaßt werden, wobei die Region aus der Postleitzahl abzuleiten ist:

Bitte Feldnamen für Zwischensumme eingeben: $(ort,1,1)

Aus der Vorstellung der Abfragetechniken kennen Sie bereits die Substring-Funktion $(Feldname,Start,Länge), die es gestattet, innerhalb eines Felds nach einer bestimmten Zeichenfolge zu suchen. Dieselbe Funktion ist auch bei der Bildung von Zwischensummen in Reports anwendbar. Die hier

benutzte Bedingung $(ort,1,1) liefert eine Zusammenfassung in Gruppen mit jeweils gleicher erster Stelle in der Postleitzahl, d.h. mit jeweils gleichem Postleitzahlbereich. Selbstverständlich muß die Datei vorher wieder in geeigneter Weise sortiert oder indiziert sein.

Als Überschrift für die Gruppen wird das Wort "Region" gewählt, dem in der gedruckten Liste dann die erste Ziffer der Potleitzahl folgt:

Bitte Überschrift für Zwischensumme eingeben: Region

In Spalte 3 der Liste soll der prozentuale Umsatz ermittelt werden. Dazu wird die bekannte Formel

umsatz/Mumsatz*100

benutzt, die jetzt allerdings noch etwas verfeinert wird. Und zwar sollen im Gegensatz zur Darstellung in der Liste **umsatz2** gerundete ganzzahlige Prozentwerte gezeigt werden. Das wird erreicht, indem zum Prozentwert 0.5 addiert und von der Summe nur der ganzzahlige Anteil genommen wird, was mit Hilfe der Funktion

INT (numerischer Ausdruck)

(= INTEGER = ganze Zahl) geschieht:

```
003     15,INT(umsatz/Mumsatz*100+0.5)
```

Der gedruckte Bericht ist in Bild 2.44 dargestellt.

```
Seitennr. 00001
15/08/85

       Regionale Umsatzverteilung
       --------------------------

Kunden-Nr.        Umsatz 85  in Prozent vom
                                Gesamtumsatz

* Region 1
761                3005.55              11
** Zwischensumme **
                   4597.05              17

* Region 2
147                 980.00               4
258                 355.45               1
** Zwischensumme **
                   1335.45               5

*Region 6
159                 852.00               3
345                2005.67               7
357                2111.00               8
852                1200.00               4
** Zwischensumme **
                   6168.67              22

* Region 8
123                1234.00               5
234                3218.00              12
369                 654.00               2
098                5612.89              21
267                  69.88               3
421                 761.00               3
741                2587.00              10
** Zwischensumme **
                  14765.77              56

** Gesamt **
                  26866.94             100
```

Bild 2.44: Regionale Verteilung der Kundenumsätze

In manchen Fällen, insbesondere bei der Auswertung größerer Dateien, ist es wünschenswert, für eine kurze Übersicht nur die Zwischensummen und Summen eines Reports drucken zu lassen und die Details zu unterdrücken (Bild 2.45). Dazu muß die Frage

Nur Zusammenfassung des Berichts? (J/N)

beim Festlegen der Liste mit J beantwortet werden.

Sollen beide Listen, die detaillierte und der "Summary Report" verfügbar gehalten werden, ist es zweckmäßig, zwei getrennte Listen zu definieren und sie wahlweise abzurufen. Die Änderung einer bereits existierenden Listenstruktur ist zwar sehr schnell möglich, erfordert aber etwas Übung im Umgang mit Programmdateien und soll daher erst an späterer Stelle besprochen werden.

An den Befehl REPORT FORM ... lassen sich auch wieder Bedingungen knüpfen, die mit dem Zusatz FOR eingeleitet werden. Damit können alle bekannten Selektionsmöglichkeiten in die Erstellung einer Liste eingebracht werden. Besonders interessant ist, daß eine FOR-Klausel nicht Bestandteil der Listenstruktur ist und erst bei Abruf des Reports an den REPORT-Befehl angehängt werden muß. Es können also mit einer einzigen Listenstruktur eine ganze Reihe verschiedener Berichte erstellt werden.

Hier einige Beispiele:

USE kunden

REPORT FORM umsatz1 FOR offen = umsatz

ergibt eine Liste aller Kunden, die noch keine Rechnung bezahlt haben (Bild 2.46).

```
Seitennr. 00001
15/08/85

        Regionale Umsatzverteilung
        --------------------------

Kunden-Nr.        Umsatz 85 in Prozent vom
                              Gesamtumsatz

* Region 1
** Zwischensumme **
                   4597.05              17

* Region 2
** Zwischensumme **
                   1335.45               5

*Region 6
** Zwischensumme **
                   6168.67              22

* Region 8
** Zwischensumme **
                  14765.77              56
** Gesamt **
                  26866.94             100
```

Bild 2.45: Wahlweise Darstellung der Zwischensummen und Summen allein

```
Seitennr. 00001
15/08/85
                    Umsatzübersicht nach Kunden

Kunden-Nr.      Umsatz 85  davon bezahlt      Aussenstände

147               980.00          0.00            980.00
741              2587.00          0.00           2587.00
852              1200.00          0.00           1200.00
963              1591.50          0.00           1591.50
** Gesamt **
                 6358.50          0.00           6358.50
```

Bild 2.46: Liste aller Kunden ohne Zahlungseingänge

Mit einer kurzen Folge von fünf Befehlen lassen sich die "besten" Kunden in der Reihenfolge ihrer Umsätze darstellen:

USE kunden

SORT ON umsatz TO temp DESCENDING

USE temp

SET HEADING TO Die umsatzstärksten Kunden

REPORT FORM umsatz1 FOR umsatz > 2000

Zunächst wird die Kundendatei in absteigender Reihenfolge nach Umsätzen sortiert (oder auch indiziert) und in die Datei **temp** geladen. Dann wird diese aufgerufen und zur Listenerstellung herangezogen.

Vor dem Kommando REPORT ist ein neuer Befehl

SET HEADING TO (Text)

(= setze die Überschrift) eingefügt, der es gestattet, in die erste Zeile der Liste eine weitere Überschrift zu setzen, die zum Beispiel die besonderen Bedingungen des Reports erläutert.

Die Liste, die alle Kunden mit Umsätzen von mehr als 2000 Rechnungseinheiten erfaßt, ist in Bild 2.47 dargestellt.

Anstatt die Erfassung im Bericht von einem Umsatzwert abhängig zu machen, kann man auch die Anzahl der ausgegebenen Datensätze direkt begrenzen. Analog zu dem Befehl DISPLAY NEXT kann der Zusatz NEXT auch in Zusammenhang mit dem Befehl REPORT verwendet werden:

USE temp (das ist die eben definierte Datei **temp**)

REPORT NEXT 5 FORM umsatz1

listet die fünf umsatzstärksten Kunden ohne Rücksicht auf die absolute Umsatzhöhe (2.48).

Seitennr. 00001 Die umsatzstärksten Kunden
15/08/85

Umsatzübersicht nach Kunden

Kunden-Nr.	Umsatz 85	davon bezahlt	Aussenstände
098	5612.89	4490.31	1122.58
234	3218.00	2574.40	643.60
761	3005.55	2404.44	601.11
741	2587.00	0.00	2587.00
357	2111.00	1688.80	422.20
345	2005.67	2005.67	0.00
** Gesamt **			
	18540.11	13163.62	5376.49

Bild 2.47 Die Kunden mit Umsätzen über 2000.

Seitennr. 00001 Die umsatzstärksten Kunden
15/08/85

Umsatzübersicht nach Kunden

Kunden-Nr.	Umsatz 85	davon bezahlt	Außenstände
098	5612.89	4490.31	1122.58
234	3218.00	2574.40	643.60
761	3005.55	2404.44	601.11
741	2587.00	0.00	2587.00
357	2111.00	1688.80	422.20
** Gesamt **			
	16534.44	11157.95	5376.49

Bild 2.48: Die fünf umsatzstärksten Kunden

3 Das Arbeiten mit mehreren Dateien

Eines der wichtigen Argumente, das für den Einsatz von Datenbanken spricht, ist deren Fähigkeit, Daten weitgehend redundanzfrei, das heißt jede Information physisch nur an einer Stelle in der Datenbank zu speichern. Dadurch wird Speicherplatz gespart und, vor allem kann die Pflege der Datenbestände relativ einfach und sicher gehandhabt werden.

Andererseits erfordert diese Art der Speicherung eine hohe Flexibilität bezüglich des Datenzugriffs. Denn, wenn Informationen nur jeweils an einer Stelle gespeichert werden, muß unter wechselnden Bedingungen und von jedem Programm aus auf sie zugegriffen werden können. Insbesondere müssen bei Bedarf auch Daten aus verschiedenen Dateien gleichzeitig verfügbar sein.

In der Fähigkeit und in der Art, mehrere Dateien gleichzeitig zu bearbeiten, unterscheiden sich echte Datenbanksysteme von einfachen Dateiverwaltungssystemen und auch die Datenbanksysteme untereinander. dBASE II bietet hier die unbestrittenen Vorteile einer relationalen Datenbank gegenüber anderen Systemen, allerdings in einem Rahmen, der dem Einsatzgebiet eines Mikrocomputers angemessen ist.

3.1 Stapelverarbeitung mehrerer Dateien

Bevor Sie mit Ihrem Computer weiterarbeiten, sollten Sie einmal auf Ihrer dBASE-Diskette den noch verfügbaren Speicherplatz überprüfen, indem Sie vom Betriebssystem aus den CP/M-Befehl

STAT *.*

eingeben, der in dieser Form zusätzlich alle Dateien mit ihrer Größe in Byte anzeigt. (Dazu muß die Datei STAT.COM auf Ihrer Diskette sein.) Wahrscheinlich ist der freie Platz nicht mehr sehr groß. Dann sollten Sie alle Datendateien (Typ DBF) mit Ausnahme der Datei **kunden** auf eine andere Diskette kopieren, insbesondere die Datei **adressen**, die Sie später noch brauchen können. Temporäre Dateien können Sie einfach löschen. So schaffen Sie sich Platz für die folgenden Beispiele.

Denn wir müssen jetzt zwei neue Dateien erstellen. Die erste mit Namen **rechnung** soll sämtliche Rechnungen an die Kunden und Zahlungseingänge von den Kunden erfassen und verwalten. Eröffnen Sie die neue Datei mit

CREATE rechnung

und definieren Sie die Dateistruktur nach Bild 3.1. Das Festlegen der Dateistruktur ist Ihnen inzwischen so geläufig, daß auf eine detaillierte Erklärung verzichtet werden kann. In Feld 001 soll die fünfstellige Rechnungsnummer (RNr) als Identifikationsmerkmal für alle Rechnungen stehen.

```
. CREATE rechnung

Satzstruktur folgendermaßen eingeben
Feld    Name, Typ, Länge, Dezimalstellen
001     rnr,c,5
002     datum,c,8
003     kunummer,c,3
004     betrag,n,10,2
005     bezahlt,n,10,2
006     bdatum,c,8
007
```

Bild 3.1: Die Struktur der Datei **rechnung**

Das Rechnungsdatum in Feld 002 wird mit acht Stellen festgelegt und kann gegebenenfalls wieder vom aktuellen Datum übernommen werden. Die Kundennummer wird exakt so definiert wie schon bisher und dient als Bindeglied zu den kundenbezogenen Dateien, die früher erstellt wurden und künftig mit der Rechnungsdatei zusammenarbeiten sollen. In Feld 004 ist der Rechnungsbetrag vorgesehen und Feld 005 soll später Zahlungseingänge auf diese Rechnung aufnehmen. Dazu wird in Feld 006 das Datum **bdatum** des Zahlungseingangs erfaßt.

Zum Stichtag 15.8.85 seien sechs Rechnungen erfaßt, die zum Teil bereits Zahlungseingänge vorweisen (Bild 3.2). Nehmen Sie diese sechs Datensätze in die Datei **rechnung** auf. Sie stellen die Ausgangsbasis für die späteren Bearbeitungen dar.

```
00001  11115  15/08/85  963   1591.50      0.00
00002  11116  15/08/85  369    654.00    654.00  15/08/85
00003  11117  15/08/85  147    980.00      0.00
00004  11118  15/08/85  123   1234.00   1234.00  15/08/85
00005  12000  15/08/85  357   2111.00   1688.80  15/08/85
00006  12222  15/08/85  098   2155.00    556.00  15/08/85
```

Bild 3.2: Die Datei **rechnung**

Als zweite Datei soll eine sogenannte Bewegungsdatei eingerichtet werden, deren Struktur identisch mit der von **rechnung** ist und die alle Transaktionen (neue Rechnungen und Zahlungseingänge) erfaßt, bevor diese endgültig in die Datei **rechnung** übernommen werden.

Bewegungsdateien sind in der Datenverarbeitung sehr verbreitet. Sie werden meist benutzt, um die Aktivitäten (Datenänderungen) etwa eines Tages zu sammeln und am Abend gemeinsam in einem sogenannten Tagesabschluß in die Stammdatei zu übernehmen. Dieser täglich einmalige Vorgang ist bei komplexen Datenstrukturen in der Regel schneller als die wiederholte Einzelbearbeitung der Datenbank für jede Transaktion. Hinzu kommen wichtige Gründe der Datensicherheit: Fehlerhafte Eingaben können noch vor der eigentlichen Verarbeitung korrigiert werden und gefährden nicht die Stammdatei. Die Bewegungsdatei läßt sich leicht als Tagesprotokoll ausdrucken und eventuell korrigieren.

Für manche Anwendungen, insbesondere bei mehreren Benutzern eines Datenbanksystems, ist es allerdings nachteilig, daß durch das Einschalten der Bewegungsdatei die Stammdatei nicht sofort aktualisiert wird und damit die neuesten Informationen nicht sofort in der Stammdatei zur Verfügung stehen. Sie als alleiniger Benutzer Ihrer dBASE-Datenbank können aber Ihre Stammdatei zu jeder beliebigen Zeit aktualisieren.

Die Bewegungsdatei ist sehr einfach erstellt. Es muß lediglich die Struktur der Datei **rechnung** in die neue Datei **bewegung** kopiert werden:

```
COPY STRUCTURE TO bewegung
```

In die noch leere Bewegungsdatei sollen zunächst fünf Datensätze (Transaktionen) eingetragen werden (Bild 3.3). Dazu rufen Sie die Datei **bewegung** auf und benutzen die Eingabemaske:

```
USE bewegung

APPEND
```

Die ersten beiden Sätze stellen Zahlungseingänge dar, für die lediglich die Rechnungsnummer, der bezahlte Betrag und das Datum zu erfassen sind. Die Kunden, die diese Zahlung leisteten, müssen nicht eingegeben werden, da sie unter den Rechnungsnummern 11115 bzw. 12222 bereits in der Datei **rechnung** erfaßt sind. Aus dem gleichen Grund sind auch Rechnungsdatum und Rechnungsbetrag nicht erneut zu erfassen. Für das aktuelle Datum des Zahlungseingangs, im Bild 3.3 der 20.8.85, sollten Sie auf jeden Fall ein anderes Datum als das Rechnungsdatum wählen, damit Sie die spätere Aktualisierung der Stammdatei verfolgen können.

```
00001  11115                          0.00   1200.00  20/08/85
00002  12222                          0.00   1599.00  20/08/85
00003  12345  20/08/85  234     1500.00      0.00
00004  12346  20/08/85  345     1798.00      0.00
00005  12347  20/08/85  761     2195.00      0.00
```

Bild 3.3: Bewegungsdatei zur Rechnungserfassung

Die Datensätze 00003 bis 00005 sind neu erfaßte Rechnungen mit den laufenden Rechnungsnummern 12345 bis 12347. Für sie wird das Rechnungsdatum, ebenfalls der 20.8.85, die Kundennummer und der Rechnungsbetrag eingegeben. Zahlungseingänge sind natürlich hier noch nicht vorhanden.

Die fünf Datensätze der Bewegungsdatei repräsentieren alle Transaktionen des 20.8.85. Sie sollen in die Datei **rechnung** eingespielt werden und diese aktualisieren, was in zwei Schritten erfolgt.

Zuerst werden die drei neuen Rechnungen in die Datei **rechnung** übernommen. Sie kennen den Befehl APPEND FROM, der es ermöglicht, Daten aus einer anderen Datei zu kopieren. Er wurde schon benutzt, um ausgelagerte Daten in die Datei zurückzuholen, nachdem deren Struktur geändert wurde. Derselbe Befehl kann jetzt auch angewandt werden, um bestimmte Sätze der Datei **bewegung** an die Datei **rechnung** anzuhängen.

Da Rechnungsnummern in der Regel in aufsteigender Reihenfolge vergeben werden, kann die Rechnungsnummer als Kriterium dafür dienen, ob es sich um eine neue Rechnung handelt oder nicht. Im vorliegenden Fall sind alle Rechnungen mit Nummern über 12222 neu und müssen in die Stammdatei übernommen werden. Dann kann der Befehl zur Übernahme der Datensätze wie folgt formuliert werden:

```
USE rechnung

APPEND FROM bewegung FOR rnr > '12222'
```

Mit der Systemmeldung "0003 Sätze hinzugefügt" sind die Rechnungen mit den Nummern 12345 bis 12347 in die Rechnungsdatei übertragen (Bild 3.4).

Anmerkung: Selbstverständlich hätten auch andere Kriterien, wie z.B. das Rechnungsdatum, zur Selektion der neuen Rechnungen benutzt werden können.

00001	11115	15/08/85	963	1591.50	0.00	
00002	11116	15/08/85	369	654.00	654.00	15/08/85
00003	11117	15/08/85	147	980.00	0.00	
00004	11118	15/08/85	123	1234.00	1234.00	15/08/85
00005	12000	15/08/85	357	2111.00	1688.80	15/08/85
00006	12222	15/08/85	098	2155.00	556.00	15/08/85
00007	12345	20/08/85	234	1500.00	0.00	
00008	12346	20/08/85	345	1798.00	0.00	
00009	12347	20/08/85	761	2195.00	0.00	

Bild 3.4: Aufnahme neuer Rechnungen in die Rechnungsdatei

Die Zahlungseingänge dürfen nicht als neue Sätze in die Datei **rechnung** übernommen werden, da für sie bereits Sätze mit den Informationen über die Rechnung (Rechnungsnummer, Betrag, Datum, Kundennummer) existieren. Die Übertragung der Zahlungseingänge stellt also eine Änderung bestehender Datensätze dar und ist auf eine andere Weise vorzunehmen.

Für derartige Probleme verfügt dBASE über einen sehr mächtigen Befehl, nämlich

UPDATE FROM (Dateiname) ON (Feldname) ADD/REPLACE (Felder)

(UPDATE = aktualisieren), der es gestattet, Satz für Satz einzelne Felder aus einer Datei in eine andere zu übernehmen und sie dort zu verarbeiten.

Der korrekte Einsatz des Befehls UPDATE setzt voraus, daß beide Dateien nach dem unter **ON** genannten Feld sortiert sind. Für das vorliegende Beispiel, in dem die Rechnungsnummer **rnr** dieses Feld sein soll, trifft dies bereits zu. Dann können Sie schreiben:

```
USE rechnung

UPDATE FROM bewegung ON rnr ADD bezahlt REPLACE bdatum
```

Zunächst wird die Datei aufgerufen, die aktualisiert werden soll: **rechnung**. Mit FROM wird die Datei **bewegung** angesprochen, aus der die neuen Informationen zu übernehmen sind, und mit ON wird ein Datenfeld als Schlüssel angegeben, der zur Identifizierung und Zuordnung der einzelnen Sätze dienen soll: Die Rechnungsnummer **rnr**.

Zum besseren Verständnis sei der interne Ablauf bei Ausführung des Befehls UPDATE kurz erläutert:

Beide Dateien sind nach dem Schlüsselbegriff **rnr** sortiert. Der Datensatzzeiger weist in ihnen jeweils auf den ersten Satz. Nun wird die FROM-Datei **bewegung** nach derjenigen Rechnungsnummer abgesucht, die im ersten Satz der Datei **rechnung** steht, nämlich die Rechnungsnummer 11115. Diese Nummer ist auch in der Bewegungsdatei vorhanden. Also gehören beide Sätze zusammen. Die Daten können aktualisiert werden. Dann springt dBASE zum zweiten Satz der Rechnungsdatei (Rechnungsnummer 11116) und sucht wieder in der Bewegungsdatei nach einem passenden Satz. Der ist nicht vorhanden, also springt dBASE zum dritten Satz usw. Beim Datensatz mit der Rechnungsnummer 2222 schließlich findet sich wieder ein entsprechender Satz in der Bewegungsdatei, dessen Daten übernommen werden. Auf diese Weise werden sämtliche Datensätze der Datei überprüft und gegebenenfalls aktualisiert.

Schließlich werden im Befehl UPDATE noch die Felder genannt, die aktualisiert werden sollen, wobei man pro Feld wählen kann, ob die Daten aus der FROM-Datei zu den Daten der Stammdatei addiert werden oder diese ersetzen sollen. Im vorliegenden Beispiel soll der Zahlungseingang zu eventuell schon früher eingegangenen Teilzahlungen addiert werden, während das Datum des Zahlungseingangs übernommen wird und ein vielleicht vorhandenes Datum einer früheren Teilzahlung überschreibt. Damit steht immer das Datum der letzten Zahlung in diesem Feld. Sie können das an den Datensätzen 00001 und 00006 in Bild 3.5 erkennen, die das Datum 20.8.85 aufweisen.

```
00001  11115  15/08/85  963    1591.50   1200.00  20/08/85
00002  11116  15/08/85  369     654.00    654.00  15/08/85
00003  11117  15/08/85  147     980.00      0.00
00004  11118  15/08/85  123    1234.00   1234.00  15/08/85
00005  12000  15/08/85  357    2111.00   1688.80  15/08/85
00006  12222  15/08/85  098    2155.00   2155.00  20/08/85
00007  12345  20/08/85  234    1500.00      0.00
00008  12346  20/08/85  345    1798.00      0.00
00009  12347  20/08/85  761    2195.00      0.00
```

Bild 3.5: Die aktualisierte Rechnungsdatei

Dieselbe Bewegungsdatei könnte auch benutzt werden, um die Kundendatei **kunden** zu aktualisieren. Dann müssten aber beim Erfassen der Zahlungseingänge in der Bewegungsdatei auch die Kundennummern mit aufgenommen werden. Denn diese wären gegebenenfalls als gemeinsamer Schlüssel für die Funktion UPDATE zu wählen.

Anstatt diese Arbeiten durchzuführen, wird gleich eine Alternative gezeigt, mit der die beiden Dateien **rechnung** und **kunden** gleichzeitig und ohne Umweg über eine Bewegungsdatei aktualisiert werden können.

Nachdem alle Informationen aus der Bewegungsdatei in die Rechnungsdatei (und in der Praxis auch in die Kundendatei) eingearbeitet sind, müssen aus der Bewegungsdatei schließlich noch alle Datensätze gelöscht werden, um für die nächsten Transaktionen vorbereitet zu sein. Das darf nie vergessen werden, weil sonst die Gefahr besteht, daß Datensätze mehrfach in die anderen Dateien überspielt werden. Zum Löschen der Datensätze kann der Befehl

DELETE (Bereich) FOR (Bedingung) (= löschen)

verwendet werden, für den ein Wirkungsbereich (Bereich) anzugeben ist. Die wahlweise anzuhängende FOR-Klausel gestattet es, das Löschen nur unter bestimmten Bedingungen wirksam werden zu lassen. Im vorliegenden Fall sollen alle Sätze der Datei **bewegung** bedingungslos gelöscht werden, also können Sie schreiben:

```
USE bewegung

DELETE ALL

PACK
```

Mit dem Befehl DELETE ALL werden alle Sätze gelöscht.

3.2 Gleichzeitiges Aktualisieren von zwei Dateien mit unterschiedlicher Dateistruktur

Eine Bewegungsdatei hat, wie schon angedeutet, neben vielen Vorzügen auch Nachteile. Ein Nachteil ist, daß die eigentliche Stammdatei nicht im Augenblick der Datenerfassung, sondern erst später im einer sogenannten Stapelverarbeitung (oder Batchverarbeitung) aktualisiert wird. Das ist manchmal unbrauchbar oder zumindest unerwünscht. Zudem stört bei älteren dBASE-Versionen eventuell die Bedingung, daß bei Verwendung des Befehls UPDATE beide Dateien eine zumindest partiell gleiche Dateistruktur aufweisen müssen. Das ist nicht immer realisierbar.

Für Fälle, in denen eine sofortige Aktualisierung zweier Dateien mit unterschiedlicher Struktur gefordert wird, bietet dBASE die Möglichkeit, mit einer kurzen Befehlskette jede beliebige Aktualisierung zu erzielen.

dBASE ist in der Lage, zwei Dateien gleichzeitig im Zugriff zu halten. Diese werden "primäre" (englisch PRIMARY) und "sekundäre" (englisch SECONDARY) Datei genannt. Ist nur eine Datei eröffnet, was in allen bisherigen Beispielen der Fall war, so wird diese intern stets als die primäre Datei betrachtet. Wenn Sie die Datei **rechnung** mit

```
USE rechnung
```

eröffnen, dann können Sie eine zweite Datei mit

```
SELECT SECONDARY
USE kunden INDEX kundnum
```

aufrufen. Jetzt haben Sie gleichzeitig die Datei **rechnung** und die nach Kundennummern indizierte Kundendatei im Zugriff.

Sie mögen sich vielleicht fragen, was das Besondere daran sein soll, zwei Dateien im gleichzeitigen Zugriff zu haben. Schließlich haben Sie sämtliche Dateien im sofortigen Zugriff, wenn Sie nur den Befehl USE (Dateiname) geben. Es besteht aber ein wichtiger Unterschied:

Mit dem Befehl USE wird eine Datei von der Diskette gelesen und ihr Datensatzzeiger auf den ersten Satz gerichtet, was Sie erkennen, wenn Sie sofort danach DISPLAY schreiben. Jede Dateibearbeitung, wie z.B. Suchen,

startet von diesem ersten Satz aus. Mit erneutem Befehl USE wird eine andere Datei und deren erster Satz aufgerufen. Die erste Datei wird geschlossen und ist für Befehle nicht mehr ansprechbar.

Mit SELECT SECONDARY und nachfolgendem USE (Dateiname) wird zwar die zweite Datei eröffnet, aber die erste nicht geschlossen. Der Datensatzzeiger für die erste Datei bleibt auf dem zuletzt bearbeiteten Datensatz stehen. Die zweite Datei erhält einen eigenen Datensatzzeiger, der auf den aktuellen Satz weist. Nun kann man abwechselnd jede Datei für sich bearbeiten, indem man zur ersten mit dem Befehl

SELECT PRIMARY

und zur zweiten mit dem Befehl

SELECT SECONDARY

wechselt, wobei man stets wieder zu dem zuletzt bearbeiteten Datensatz zurückkehrt und nicht an den Dateianfang. Letzteres ist sehr wichtig, da nicht immer erneut nach dem gewünschten Satz gesucht werden muß.

Nehmen Sie nun in die Rechnungsdatei eine neue Rechnung auf:

SELECT PRIMARY

APPEND

Geben Sie ein:

```
rnr       : 23611
datum     : 20/08/85
kunummer  : 369
betrag    : 1244.00
```

Damit ist die Rechnungsdatei bereits aktualisiert (Bild 3.6).

00001	11115	15/08/85	963	1591.50	1200.00	20/08/85
00002	11116	15/08/85	369	654.00	654.00	15/08/85
00003	11117	15/08/85	147	980.00	0.00	
00004	11118	15/08/85	123	1234.00	1234.00	15/08/85
00005	12000	15/08/85	357	2111.00	1688.80	15/08/85
00006	12222	15/08/85	098	2155.00	2155.00	20/08/85
00007	12345	20/08/85	234	1500.00	0.00	
00008	12346	20/08/85	345	1798.00	0.00	
00009	12347	20/08/85	761	2195.00	0.00	
00010	23611	20/08/85	369	1244.00	0.00	

Bild 3.6: Rechnungsdatei

Nun sollen diese Daten in die Kundendatei übernommen werden. Dazu muß der Kunde mit Nummer 369 gefunden werden. Speichern Sie die Kundennummer und wählen Sie dann die Kundendatei:

STORE kunummer TO Mkunde

SELECT SECONDARY

Nun wird der Datensatz mit der Kundennummer 369 gesucht, wobei der sehr schnelle Befehl FIND eingesetzt wird:

FIND &Mkunde

Zum Aktualisieren des Datensatzes können nun die üblichen Befehle benutzt werden, wobei lediglich definiert werden muß, aus welchen Dateien die angesprochenen Felder zu entnehmen sind. Dazu wird den Feldern aus der Primärdatei ein "P.", denen aus der Sekundärdatei ein "S." vorangestellt. Der Vorsatz entfällt für die Felder der jeweils SELECTierten Datei.

Augenblicklich ist die sekundäre Kundendatei, die aktualisiert werden soll, gewählt, so daß die Felder der primären Rechnungsdatei den Vorsatz "P." erhalten. Übertragen Sie nun die Rechnungsnummer und das Rechnungsdatum und addieren Sie den Rechnungsbetrag zum Umsatz:

REPLACE datum WITH P.datum

REPLACE lrech WITH P.rnr

REPLACE umsatz WITH umsatz + P.betrag

Insgesamt ist diese Befehlsfolge, die in Bild 3.7 noch einmal wiedergegeben ist, sicherlich zu lang, um sie für jede einzelne Transaktion neu einzugeben. Das ist aber auch nicht nötig. Im nächsten Kapitel werden Sie sehen, wie man solche Befehlsfolgen als Programme speichern und abrufen kann.

```
. USE rechnung
. SELECT SECONDARY
. USE kunden INDEX kundnum
. SELECT PRIMARY
. APPEND
. STORE kunummer TO Mkunde
. SELECT SECONDARY
. FIND &Mkunde
. REPLACE datum WITH P.datum
. REPLACE lrech WITH P.rnr
. REPLACE umsatz WITH umsatz + P.betrag
```

Bild 3.7: Die Befehlsfolge zum Aktualisieren zweier Dateien

00001	098	8057		15/08/85	5612.89	4490.31	1122.58
00002	123	8000		15/08/85	1234.00	1234.00	0.00
00003	147	2000		15/08/85	980.00	0.00	980.00
00004	159	6000		15/08/85	852.00	681.60	170.40
00005	234	8000		15/08/85	3218.00	2574.40	643.60
00006	258	2000		15/08/85	355.45	355.45	0.00
00007	267	8201		15/08/85	698.88	559.10	139.78
00008	345	6100		15/08/85	2005.67	2005.67	0.00
00009	357	6100		15/08/85	2111.00	1688.80	422.20
00010	369	8000	23611	20/08/85	1898.00	654.00	0.00
00011	421	8500		15/08/85	761.00	608.80	152.20
00012	741	8630		15/08/85	2587.00	0.00	2587.00
00013	761	1000		15/08/85	3005.55	2404.44	601.11
00014	852	6850		15/08/85	1200.00	0.00	1200.00
00015	963	1000		15/08/85	1591.50	0.00	1591.50

Bild 3.8: Aufnahme der Rechnung Nr.23611 in die Kundendatei

Bild 3.8 zeigt die aktualisierte Kundendatei, für die im Datensatz # 00010 die neue Rechnung ergänzt wurde. Sie sehen, daß das Datum für die letzte Rechnung von 15/08/85 auf 20/08/85 geändert und der Umsatzwert erhöht ist. Der Wert **offen** als Differenz zwischen **umsatz** und **bezahlt** ist noch nicht neu berechnet, wird aber gleich nachgeholt.

Die Übernahme eines Zahlungseingangs von der Rechnungsdatei in die Kundendatei läuft ähnlich ab. Primärdatei ist wieder **rechnung** und Sekundärdatei **kunden INDEX kundnum**. Angenommen für die Rechnungsnummer 11117 sei eine Zahlung eingegangen, dann muß der entsprechende Satz zuerst in der Rechnungsdatei gesucht und dann editiert werden:

```
SELECT PRIMARY

LOCATE FOR rnr = '11117'

EDIT #
```

Mit dem Befehl EDIT # wird derjenige Datensatz editiert, der augenblicklich im Zugriff ist, nämlich der mit LOCATE gefundene. Das Zeichen # wird von dBASE als Symbol für die laufende Nummer des Datensatzes verstanden und kann als solches auch in Befehlen verwendet werden. Zum Beispiel kann mit STORE # TO (Speichername) die Nummer eines Datensatzes abgespeichert und später wieder zurückgeholt werden.

Nun kann der bezahlte Betrag über die Editiermaske eingegeben werden. Schreiben Sie in Feld **bezahlt** den Wert 980.00 und geben Sie das Datum 20/08/85 ein. Mit CTRL-W wird die EDIT-Funktion wieder verlassen und die nächsten Befehle können eingegeben werden:

```
STORE kunummer TO Mkunde

SELECT SECONDARY

FIND &Mkunde
```

Wie im vorausgegangenen Beispiel wird die Kundennummer vorübergehend abgespeichert, die sekundäre Datei **kundnum** gewählt und anhand der gespeicherten Kundennummer der Datensatz für diesen Kunden gesucht. An sich muß nun lediglich der bezahlte Betrag in die Kundendatei übernommen werden. Da aber die Rechnungsnummer in **kundnum** noch fehlt, soll dies hierbei nachgeholt werden:

REPLACE lrech WITH P.rnr

REPLACE bezahlt WITH bezahlt + P.bezahlt

Mit dem zweiten REPLACE-Befehl wird die in der Kundendatei gespeicherte Summe aller Zahlungseingänge von diesem Kunden um den Betrag der letzten bezahlten Rechnung erhöht.

Wenn alle Transaktionen zur Aktualisierung beider Dateien beendet sind, müssen zuletzt noch die Außenstände, also die Felder mit Namen **offen** in der Kundendatei neu berechnet werden. Das erfolgt wieder mit

REPLACE ALL offen WITH umsatz - bezahlt

Bild 3.9 zeigt die neu berechnete Kundendatei.

Achtung: Aus Gründen der Übersichtlichkeit wurde die Befehlsfolge hier etwas vereinfacht. Der zweite REPLACE-Befehl ist in dieser Form nur dann korrekt, wenn die Rechnung mit einem einzigen Zahlungseingang beglichen wird. Wenn Teilzahlungen erfolgen, werden die ersten zweimal oder öfter in die Kundendatei akkumuliert. Dieses Problem läßt sich beheben, wenn man den Betrag der Teilzahlung zuerst in einen Speicher schreibt und von dort aus in beide Dateien addiert.

00001	098	8057		15/08/85	5612.89	4490.31	1122.58
00002	123	8000		15/08/85	1234.00	1234.00	0.00
00003	147	2000	11117	15/08/85	980.00	980.00	0.00
00004	159	6000		15/08/85	852.00	681.60	170.40
00005	234	8000		15/08/85	3218.00	2574.40	643.60
00006	258	2000		15/08/85	355.45	355.45	0.00
00007	267	8201		15/08/85	698.88	559.10	139.78
00008	345	6100		15/08/85	2005.67	2005.67	0.00
00009	357	6100		15/08/85	2111.00	1688.80	422.20
00010	369	8000	23611	20/08/85	1898.00	654.00	1244.00
00011	421	8500		15/08/85	761.00	608.80	152.20
00012	741	8630		15/08/85	2587.00	0.00	2587.00
00013	761	1000		15/08/85	3005.55	2404.44	601.11
00014	852	6850		15/08/85	1200.00	0.00	1200.00
00015	963	1000		15/08/85	1591.50	0.00	1591.50

Bild 3.9: Die Kundendatei nach Abschluß der Bearbeitung

3.3 Erzeugen einer neuen Datei aus zwei bestehenden

Es ist charakteristisch für eine Datenbank, daß die gespeicherten Informationen über mehrere oder gar viele Dateien verteilt sind, aber physisch jeweils nur an einem einzigen Ort innerhalb der Datenbank vorkommen (letzteres wird zumindest angestrebt). Dies führt zu der Notwendigkeit, gelegentlich Informationen aus mehreren Dateien zusammenzuziehen und gemeinsam darzustellen.

Sofern Informationen nur aus einzelnen Datensätzen mehrerer Dateien benötigt werden, können die eben besprochenen Techniken mit den Befehlen LOCATE, FIND, SELECT PRIMARY und SELECT SECONDARY angewandt werden. Sollen aber zwei Dateien in ihrer Gesamtheit gemeinsam dargestellt oder ausgewertet werden, dann wird besser ein neuer Befehl eingesetzt:

JOIN TO (Dateiname) FOR (Bedingung) FIELDS (Felder)

Der Befehl JOIN (= vereinigen) gestattet, zwei Dateien oder Teile davon zu einer neuen Datei zu vereinigen. Die beiden Ursprungsdateien werden mit SELECT PRIMARY bzw. SELECT SECONDARY eröffnet. Der Name der neu gebildeten Datei ist wie üblich frei wählbar. Die neuen Datensätze werden unter den festgelegten Bedingungen gebildet und setzen sich aus beliebigen Feldern der Ursprungsdateien zusammen.

Sehr wichtig ist die FOR-Klausel, die bei dem Befehl JOIN keine Option darstellt, sondern unbedingt und wohlüberlegt anzugeben ist. Sie entscheidet darüber, welche Sätze der ersten mit welchen Sätzen der zweiten Datei zu neuen Datensätzen kombiniert werden. Rechnerintern läuft die Ausführung des Befehls JOIN etwa folgendermaßen ab:

Zunächst weist in beiden Dateien, in der primären und in der sekundären, der Datensatzzeiger auf den ersten Satz. Dann wird der erste Satz der primären Datei nacheinander mit jedem Satz der sekundären Datei verglichen und auf die FOR-Bedingung hin abgefragt. Ist die Bedingung erfüllt, wird ein neuer Datensatz gebildet, der sich aus den angegebenen Datenfeldern beider Ursprungsdateien zusammensetzt. Ist die Bedingung nicht erfüllt, wird kein Satz gebildet.

Wenn der erste Satz der primären mit allen Sätzen der sekundären Datei verglichen ist, dann wiederholt sich der gesamte Vorgang für den zweiten Datensatz der primären Datei usw. bis schließlich jeder Satz der einen mit jedem Satz der anderen Datei verglichen ist.

Es ist leicht einzusehen, daß bei entsprechenden FOR-Klauseln unter Umständen sehr viele Datensätze erzeugt werden können, und zwar maximal so viele, wie das Produkt der Anzahl an Datensätzen beider Dateien ausmacht. Zwei Dateien mit je 1000 Datensätzen können somit bis zu 1 Million neue Datensätze bilden, was natürlich das System sprengen würde. Es ist daher größte Sorgfalt auf eine geeignete Bedingung für die Vereinigung von Dateien zu legen.

Häufig soll jeweils ein Satz aus den beiden Ursprungsdateien verknüpft werden, so daß die neue Datei ebenso viele Sätze enthält wie die Ausgangsdateien.

Beispielsweise wollen Sie eine Liste aller Kunden mit der vollständigen Anschrift und dazu die augenblicklich offenen Rechnungsbeträge haben, um diese Kunden anschreiben zu können. Dann können Sie aus der Kundendatei und aus der Adressendatei jeweils diejenigen Sätze vereinigen, die eine gemeinsame Kundennummer aufweisen:

```
USE kunden

SELECT SECONDARY

USE adressen

SELECT PRIMARY

JOIN TO kundadr FOR kunummer = S.kunummer FIELDS name,strasse,;
S.ort,offen
```

Die neue Datei heißt **kundadr** (zusammengezogen aus **kunden** und **adressen**). Die FOR-Bedingung besagt, daß immer dann ein neuer Satz zu bilden ist, wenn aus der primären und aus der sekundären Datei Sätze mit gleicher Kundennummer zusammentreffen. Da jede Kundennummer pro Datei nur einmal auftritt, wird für jeden Kunden genau ein neuer Datensatz gebildet. Dieser besteht aus den Feldern **name, strasse, ort** und **offen.**

Im allgemeinen ist es nicht notwendig, vor die Feldnamen als Zeichen ihrer Herkunftsdatei den Vorsatz "P." (für primär) oder "S." (für sekundär) zu setzen. Solange die Felder nur in einer der beiden Ursprungsdateien vorhanden sind, greift dBASE von selbst auf die richtige Datei zu. Wenn aber der gleiche Feldname in beiden Dateien vorkommt (wie z.B. **ort**), dann muß spezifiziert werden, aus welcher Datei der Feldinhalt zu entnehmen ist. In diesem Beispiel ist der vollständige Ort aus der sekundären Datei **adressen** gewünscht und nicht die Postleitzahl aus der Kundendatei, deren Feld auch **ort** heißt.

Anmerkung: Der gesamte Befehl JOIN mußte wegen seiner Länge über zwei Zeilen geschrieben werden. Das ist möglich, wenn man die erste Zeile mit einem Semikolon abschließt, dann die ENTER-Taste drückt und in der nächsten Zeile weiterschreibt.

Unter dem Dateinamen **kundadr** kann nun die neu gebildete Datei aufgerufen und bearbeitet werden (Bild 3.10).

00001	Meister	Marktplatz 3	8057	Eching	1122.58
00002	Maier	Amalienstr.59	8000	München	0.00
00003	Hassold	Meisenstr.18	2000	Hamburg	0.00
00004	Koch	Heumarkt 14	6000	Frankfurt	170.40
00005	Huber	Sieglindenstr.5	8000	München	643.60
00006	Hofmann	Holzweg 13	2000	Hamburg	0.00
00007	Hofer	Ludwigstr. 5	8201	Raubling	139.78
00008	Kuhn	Fliederstr.9	6100	Darmstadt	0.00
00009	Becker	Hauptstr.28	6100	Darmstadt	422.20
00010	Maier	Flurstr.3	8000	München	1244.00
00011	Behr	Frankenweg 17	8500	Nürnberg	152.20
00012	Hoffman	Alte Gasse 15	8630	Coburg	2587.00
00013	Schmid	Fasanenstr.2	1000	Berlin	601.11
00014	Fischer	Ginsterweg 9	6850	Mannheim	1200.00
00015	Knoll	Amselweg 23	1000	Berlin	1591.50

Bild 3.10: Bilden einer neuen Datei aus zwei bestehenden

Die Bildung neuer Datensätze kann auch von mehreren Bedingungen gleichzeitig abhängig gemacht werden. Damit läßt sich der Befehl JOIN selektiv anwenden.

Zum Beispiel können die Informationen aus der Datenbank dazu benutzt werden, im Anschluß an die Erfassung der Rechnungsdaten auch die Rechnungen selbst inklusive der Kundenadressen zu schreiben. Dann ist es notwendig, nur die neuen Rechnungen, d.h. diejenigen ab einer bestimmten Rechnungsnummer zu selektieren und mit den Kundenanschriften zu vereinigen. Wenn die neu zu bildende Datei **kundrech** (= für Kundenrechnungen) heißen und die zu schreibenden Rechnungen oberhalb der Rechnungsnummer 12222 beginnen sollen, kann diese Befehlsfolge verwendet werden:

```
USE rechnung

SELECT SECONDARY

USE adressen

SELECT PRIMARY

JOIN TO kundrech FOR (kunummer = S.kunummer .AND. rnr > '12222');
FIELDS kunummer,name,ort,rnr,datum,betrag
```

Bedingung für die Bildung der neuen Datensätz ist also, daß sowohl gleiche Kundennummern vorliegen als auch die Rechnungsnummern größer als 12222 sind. Daraus ergeben sich insgesamt vier Datensätze für die Ausstellung von Rechnungen (Bild 3.11).

```
00001  234 Huber    8000 München     12345 20/08/85   1500.00
00002  345 Kuhn     6100 Darmstadt   12346 20/08/85   1798.00
00003  761 Schmid   1000 Berlin      12347 20/08/85   2195.00
00004  369 Maier    8000 München     23611 20/08/85   1244.00
```

Bild 3.11: Selektive Bildung einer neuen Datei

3.4 Zusammenfassung

Mit Abschluß dieses Kapitels haben Sie alle wichtigen Befehle zur Bearbeitung einer dBASE-Datenbank kennengelernt. Insbesondere haben Sie gelernt,

- Dateien anzulegen und Informationen zu laden, zu ändern und zu löschen
- Dateien abzufragen und auszuwerten
- Dateistrukturen zu ändern, auch wenn die Datei bereits Datensätze enthält
- Dateien zu sortieren und zu indizieren
- den dBASE-Report-Generator anzuwenden
- mehrere Dateien zu verknüpfen.

Mit den bis jetzt bekannten Hilfsmitteln können Sie praktisch jedes Datenbankproblem im Rahmen eines Mikrocomputersystems lösen. Dabei bleibt der Lösungsweg immer übersichtlich, weil sich grundsätzlich jede Datei einer relationalen Datenbank wie dBASE in einer anschaulichen Tabellenform darstellen läßt.

Sie haben auch die Mächtigkeit der dBASE-Befehle kennengelernt und gesehen, wie man mit einer Folge von Befehlen echte Datenverarbeitung betreiben kann. Es ist daher naheliegend, daß dBASE noch über eine Reihe weiterer Befehle verfügt, die Möglichkeiten bieten, die bislang einzeln benutzten Kommandos zu vollwertigen, ja sogar sehr komplexen Programmen zusammenzustellen. Hiervon soll im nächsten Kapitel die Rede sein.

4 Programmieren mit dBASE II

Je länger Sie sich mit dBASE beschäftigen weren, desto komplexer werden vermutlich die Probleme, die Sie mit diesem Programmpaket lösen möchten. Und dann wird auch die Bearbeitung der Datenbank nicht mit einem oder wenigen Befehlen zu erledigen sein, sondern Sie werden größere, und zwar immer wieder dieselben Befehlsketten für Ihre Verarbeitung benötigen.

Es ist daher nur konsequent, wenn dBASE die Möglichkeit bietet, solche Befehlsfolgen in Dateien abzuspeichern und bei Bedarf mit einem einzigen Kommando wieder aufzurufen und erneut ablaufen zu lassen. Die hierfür verfügbaren Steuerkommandos sollen im ersten Teil dieses Kapitels besprochen werden.

Wenn diese Befehle im zweiten Teil noch durch Programmverzweigungen und Programmschleifen ergänzt werden, dann werden Sie mit dBASE eine Programmiersprache zur Verfügung haben, die als sogenannte höhere Sprache der "klassischen" Konkurrenz wie BASIC oder COBOL an Mächtigkeit um ein Mehrfaches überlegen ist.

Die folgenden Beispiele bauen weiterhin auf den bekannten Dateien **adressen**, **kunden** und **rechnung** sowie auf deren Indexdateien auf. Es ist empfehlenswert, von diesen drei Dateien Duplikate, sogenannte "Backup"-Kopien anzulegen. Dann können Sie neben dem reinen Nachvollziehen der Beispiele im Buch auch mit zusätzlichen, eigenen Daten arbeiten und anschließend durch Rückkopieren der Duplikate den Datenbestand entsprechend den Buchbeispielen wiederherstellen.

4.1 Einfache Befehlsketten

Grundsätzlich unterscheiden sich dBASE-Programme nicht von den einzeln eingegebenen und ausgeführten Befehlen. Jeder bisher besprochene Befehl kann unverändert in ein Programm übernommen werden. Unterschiedlich ist lediglich die Art der Befehlseingabe: Um die Befehle speichern zu können, müssen sie in eine Befehlsdatei (engl. COMMAND FILE) geschrieben werden. Dort werden sie gespeichert und nach Aufruf des Programmnamens ausgeführt. Eine Kommandodatei wird mit dem Befehl

MODIFY COMMAND (Dateiname)

eröffnet, wobei der Dateiname wieder frei wählbar ist und entsprechend den Regeln des Betriebssystems maximal acht Zeichen umfassen darf. Die Identifizierung des Dateityps erfolgt intern durch den Zusatz **CMD**, der von dBASE vergeben wird.

Eröffnen Sie die erste Programmdatei mit Namen **program1**:

MODIFY COMMAND program1

Daraufhin wird der gesamte Bildschirm gelöscht und der Cursor in die linke obere Ecke plaziert. Es mag Ihnen nach den bisherigen Erfahrungen mit dBASE vielleicht etwas ungewöhnlich vorkommen, daß der Bidschirm völlig leer ist und keinerlei Nachricht zeigt. Aber Sie befinden sich jetzt im sogenannten Programm-Editor, dem Hilfsprogramm zum Erstellen von Anwendungsprogrammen und der gesamte Bildschirm steht Ihnen für Befehlszeilen zur Verfügung.

In dieser Betriebsart können wieder die bekannten EDIT-Funktionen benutzt werden, wobei Sie sich besonders an die Tastenfunktionen

CLR	oder CTRL-G zum Löschen einzelner Zeichen
SHIFT-CLR	oder CTRL-T zum Löschen ganzer Zeilen
CTRL-V	oder COPY zum Einfügen einzelner Zeichen
CTRL-N	zum Einfügen ganzer Zeilen

sowie an die Tasten zur Cursorsteuerung erinnern sollten. Sie werden Ihnen später beim Ändern oder Korrigieren von Programmen sehr hilfreich sein.

Im Übrigen kann jeder übliche Texteditor, der unter dem gleichen Betriebssytem verfügbar ist, zur Erstellung von dBASE-Programmen benutzt werden. Wenn Sie zum Beispiel das Textverarbeitungssystem WordStar im Einsatz haben, dann können Sie mit Hilfe von WordStar Programme für dBASE schreiben. Sie müssen lediglich der Textdatei den Programmnamen geben und - in diesem Fall selbst - den Dateityp vorschreiben, z.B.

program1.CMD

In **program1** soll nun eine kurze Befehlskette gespeichert werden, die aus der Datei **kunden** die fünf umsatzstärksten Kunden anzeigt. Dazu wird die Kundendatei in absteigender Reihenfolge sortiert und in eine temporäre Datei kopiert, von der dann die ersten fünf Sätze ausgegeben werden.

Üblicherweise sollte man ein Programm mit einem Befehl beginnen, der den Bildschirm von Informationen aus eventuell vorausgegangenen Operationen löscht. Das geschieht mit

ERASE

Der Befehl ERASE löscht den Bildschirm und positioniert den Cursor in die linke obere Ecke. Er kann übrigens auch wie die anderen Befehle einzeln benutzt werden.

Schreiben Sie nun das Programm, wobei Sie wie früher nach jeder Kommandozeile die ENTER-Taste betätigen und in die letzte Programmzeile den Befehl

RETURN

schreiben, der das Programm beendet und eine Rückkehr in die vorherige Kommandoebene von dBASE bewirkt:

```
ERASE
USE kunden
SORT ON umsatz TO temp DESCENDING
USE temp
DISPLAY NEXT 5
RETURN
```

Der Programm-Editor wird wieder verlassen, indem man - wie bei den anderen EDIT-Funktionen - die Tastenkombination CTRL-W drückt. Das geschriebene Programm wird nun auf der Diskette gespeichert, der Bildschirm wird gelöscht und es erscheint der gewohnte Punkt als Zeichen dafür, daß dBASE wieder bereit ist, einzelne Befehle zu empfangen und sofort auszuführen. Aus diesem Betriebszustand heraus kann nun das Programm **program1** aufgerufen werden, wenn der Befehl

DO (Programm-Name)

gegeben wird. Das Kommando DO (= ausführen) kann, wie hier, als einzelner Befehl eingeben, kann aber auch innerhalb eines Programms benutzt werden, um ein weiteres (Unter-)Programm aufzurufen. Dazu werden später noch Beispiele folgen. Mit dem Kommando

DO program1

wird die gespeicherte Befehlsfolge abgearbeitet, wobei eventuelle Systemnachrichten wie üblich auf dem Bildschirm angezeigt werden, z.B. "SORT beendet" nach Ausführung des Sortierbefehls. Anschließend erscheinen die fünf gesuchten Kunden mit ihren Umsatzwerten (Bild 4.1).

```
. DO program1

SORT beendet

00001  098 8057   15/08/85    5612.89    4490.31    1122.58
00002  234 8000   15/08/85    3218.00    2574.40     643.60
00003  761 1000   15/08/85    3005.55    2404.44     601.11
00004  741 8630   15/08/85    2587.00       0.00    2587.00
00005  357 6100   15/08/85    2111.00    1688.80     422.20
```

Bild 4.1: Aufruf und Ausführung von **program1**

Die Berechnung dieser Tabelle läßt sich beliebig oft wiederholen, wobei selbstverständlich zwischendurch die Datei **kunden** anderweitig bearbeitet werden kann.

Ferner kann man das Programm erneut editieren und einzelne Befehle ändern. Dazu wird wieder das Kommando

MODIFY COMMAND program1

gegeben, woraufhin das Programm in den Editor geladen wird und zur Bearbeitung auf dem Bildschirm erscheint. Das modifizierte Programm wird dann auf der Diskette an Stelle des alten gespeichert. Die alte Programmversion ist aber noch verfügbar, und zwar als Backup-Datei unter dem Dateinamen **program1.BAK**. Sie ist für dBASE nicht mehr im unmittelbaren Zugriff, könnte aber im Bedarfsfall in eine **CMD**-Datei umbenannt (z.B. mit dem Betriebssystem-Befehl REN) und damit reaktiviert werden. Allerdings ist zu beachten, daß bei Änderung eines Programms nur immer die letzte Version als **BAK**-Datei gespeichert bleibt. Alle früheren Versionen werden endgültig gelöscht.

Als zweites Beispiel für ein einfaches Programm soll noch einmal die Liste aus Bild 2.58 mit Kundenadressen und Rechnungsbeträgen erstellt werden, die zur Ausfertigung der Rechnungen dient (Bild 3.2). Eröffnen Sie die Programmdatei **programm2** mit

MODIFY COMMAND programm2

Zur besseren Übersicht sollen die Spalten mit Überschriften versehen werden, was sehr einfach mit der Funktion "?" erfolgt:

```
? 'KNr. Kunde        Wohnsitz    Rechnung/Datum    Betrag'
?
```

Es ist wieder zu beachten, daß der gesamte Text in Anführungszeichen zu setzen ist. Die beiden Fragezeichen in den folgenden Zeilen haben die Funktion, zwei Leerzeilen zwischen Überschrift und Listenbeginn einzufügen.

Mit dem Zusatz OFF zum Befehl LIST werden bei Auflistung der Datensätze die laufenden Nummern unterdrückt. Bild 4.3 zeigt die vom Programm generierte Liste.

```
ERASE
USE rechnung
SELECT SECONDARY
USE adressen
SELECT PRIMARY
JOIN TO kundrech FOR (kunummer=S.kunummer.AND.rnr>'12222');
FIELDS kunummer,name,ort,rnr,datum,betrag
USE kundrech
? 'KNr. Kunde      Wohnsitz    Rechnung/Datum     Betrag'
?
LIST OFF
RETURN
```

Bild 4.2: Programm zum Ausdrucken einer Datei

```
. DO program2

KNr. Kunde          Wohnsitz    Rechnung/Datum     Betrag

234 Huber      8000 München     12345 20/08/85     1500.00
345 Kuhn       6100 Darmstadt   12346 20/08/85     1798.00
761 Schmid     1000 Berlin      12347 20/08/85     2195.00
369 Maier      8000 München     23611 20/08/85     1244.00
```

Bild 4.3: Die mit Programm erstellten Rechnungsdaten

Selbstverständlich können auch die vom Reportgenerator erzeugten Listen über ein Programm abgerufen werden. Da das Aufbereiten und Sortieren der Dateien, die dem eigentlichen Report vorausgehen, auch Teil des Programms sein können, läßt sich das Erstellen eines Berichts mit allen Vorbereitungen in einem einzigen Befehl zusammenfassen.

4.1.1 Bildschirmmasken

Wir haben eben den Fragezeichen-Befehl benutzt, um einen zusätzlichen Text in die Liste aufzunehmen. Durch eine geeignete Anzahl von Leerzeichen zwischen den einzelnen Wörtern läßt sich der Text innerhalb der Zeile beliebig positionieren. Für eine präzise Positionierung von Texten und Daten ist jedoch eine andere Funktion besser geeignet, nämlich

@ (Zeile,Position)

mit der genau angegeben wird, in welcher Zeile und an welcher Position innerhalb der Zeile eine zu schreibende Information beginnen soll. Zeile und Position werden als Nummern angegeben, wobei für Darstellungen auf dem Bildschirm die Zeilennummern 0 bis 23 und die Positionsnummern 0 bis 79 in Frage kommen, d.h. jede mögliche Zeichenposition auf dem Bildschirm kann einzeln angesprochen werden.

Das Befehls-Symbol "@" (englisch genannt "AT"-Zeichen oder im Computerjargon "Klammeraffe") ist eines der Sonderzeichen auf der Tastatur. Im deutschen Zeichensatz finden Sie statt dessen das Paragraph-Zeichen "§", das den gleichen ASCII-Code besitzt. Lassen Sie sich auch nicht verwirren, wenn Sie mit dem ASCII-Zeichensatz arbeiten und Ihr Bildschirm das Zeichen @, der Drucker aber eventuell das Paragraphzeichen § ausgibt (wenn letzterer mit einem deutschen Zeichensatz arbeitet).

Die Ausgabe eines Zeichens erfolgt unmittelbar hinter bzw. unter der mit "@ (Zeile,Position)" gekennzeichneten Stelle, das heißt auf die Funktion

@ 0,0

hin wird das nächste Zeichen in Zeile 1, Position 1 geschrieben. Die tatsächliche Schreibposition ist also gegenüber der Angabe in der Funktion um eine Stelle nach unten bzw. rechts versetzt, was zwar etwas ungewöhnlich ist, aber in der Praxis keineswegs stört.

Zur Ausgabe von Text oder Daten wird an die Funktion "@" der Zusatz SAY (= sage, schreibe) gehängt:

@ (Zeile,Position) SAY (Ausdruck)

Als Ausdruck können ein Text (wieder in Anführungszeichen), Namen von Datenfeldern oder Speichern oder auch Formeln stehen.

Eine sehr wichtige Funktion erfüllt @ beim Formatieren und Beschreiben von Druckformularen. Die Druckpositionen können mit @ so exakt festgelegt werden, daß man ohne weiteres vorgedruckte Formulare zur Ausgabe der Daten verwenden kann. Für Druckoperationen dürfen sowohl die Zeilen- wie auch die Positionsnummern zwischen 0 und 254 liegen, womit sich auch sehr große Formulare bedrucken lassen.

Bild 4.4 zeigt die Positionierbefehle einer Bildschirmmaske zur Ausgabe von Kundeninformationen. In Bild 4.5 sehen Sie das Ergebnis für den Fall, daß der vierte Satz der Adressen-Datei im Zugriff ist. Wir werden diese Maske gleich für ein kleines Informationssystem verwenden.

```
@ 1,30 SAY 'Kunden-Information'
@ 5,5 SAY TRIM(vorname) + ' ' + name
@ 5,50 SAY 'kunden-Nr. ' + kunummer
@ 7,5 SAY strasse
@ 8,5 SAY ort
@ 12,5 SAY 'Letzte Rechnung vom: ' + datum
@ 15,5 SAY 'bisheriger Umsatz: DM'
@ 17,5 SAY 'offene Posten:     DM'
```

Bild 4.4: Positionierbefehle für eine Bildschirmmaske

```
                    Kunden-Information


Werner Meister                          Kunden-Nr. 098

Marktplatz 3
8057 Eching



Letzte Rechnung vom:

bisheriger Umsatz: DM
offene Posten:     DM
```

Bild 4.5: Bildschirmmaske für einen einzelnen Kunden

Im Unterschied zu den bisherigen Druckfunktionen, insbesondere zur Funktion CTRL-P, kann man mit Hilfe der Funktion "@" gezielt nur die Informationen in denjenigen Zeilen ausdrucken lassen, die mit "@" beginnen, und alle anderen Nachrichten vom Druck ausklammern. Dazu wird zu Beginn eines entsprechenden Programmteils der Befehl

SET FORMAT TO PRINT

(= Ausgabe der formatierten Zeilen auf den Drucker)

gegeben und am Ende mit

SET FORMAT TO SCREEN

(SCREEN = Bildschirm) die Ausgabe der Informationen wieder auf den Bildschirm gelegt. Die Zeilen, die auf den Drucker ausgegeben werden, erscheinen nicht auf dem Bildschirm.

4.1.2 Programmgesteuerte Dateneingabe

Um individuelle Kundeninformationen abzurufen, ist es erforderlich, daß während des Programmlaufs vom Benutzer Daten eingegeben werden können, die das Programm entsprechend steuern. dBASE verfügt über mehrere solcher Befehle. Ihnen allen ist gemeinsam, daß bei ihrem Auftreten das Programm anhält, auf eine Eingabe über die Tastatur wartet und dann die Verarbeitung fortsetzt. Auf den Befehl

ACCEPT (Erläuterung) TO (Speichername)

(ACCEPT = empfangen) hin wartet das System auf eine (ausschließlich) alphanumerische Dateneingabe über die Tastatur und schreibt die Information in den genannten Speicher. Als "Erläuterung" kann man einen beliebigen Text angeben, der während des Programmlaufs dem Benutzer mitteilt, welche Art von Daten das System erwartet.

Das folgende Beispiel soll diesen Befehl sowie jenen zum Formatieren im praktischen Gebrauch zeigen: Es soll ein Programm erstellt werden, das die wichtigsten Daten eines individuell ausgewählten Kunden in Form eines Informationsblattes ausdruckt (Bild 4.5).

Die für diese Informationen benötigten Daten sind auf zwei Dateien verteilt: Kundenname und Anschrift stehen in der Datei **adressen**, die restlichen Daten in der Kundendatei. Die Kundennummer findet sich in beiden Dateien und stellt die logische Verbindung zwischen ihnen her.

Der Zugriff auf die Daten erfolgt derart, daß zunächst über die Tastatur der Name des gewünschten Kunden eingegeben wird und dann seine An-

schrift und Kundennummer gesucht und ausgedruckt werden. An Hand der Kundennummer wird in der zweiten Datei auf die verknüpften Zahlenwerte und das Datum der letzten Rechnung zugegriffen.

Geben Sie dem neuen Programm den Namen **info** (für Information):

```
MODIFY COMMAND info
```

Wenn sich dBASE im Programmeditor befindet, dann können Sie in das Programm Bemerkungen einfügen, die z.B. das Programm erläutern oder schwierige Passagen erklären, was sich besonders dann als nützlich erweist, wenn Sie nach längerer Zeit das Programm wieder einmal ändern wollen. Zum Beispiel schreiben Sie in die erste Zeile eine Überschrift:

```
* Programm info
```

Das Symbol "*" am Anfang einer Zeile wird von dBASE als Beginn eines einzeiligen Kommentars verstanden, der bei der späteren Programmausführung übersprungen wird. Das eigentliche Programm beginnt wieder mit einem Befehl, den Bildschirm zu löschen (Bild 4.6) und mit der Überschrift für das Informationsblatt:

```
ERASE
@ 1,30 SAY 'Kunden-Information'
```

Wie vorher erläutert, wird also die Überschrift "Kunden-Information" in die zweite Zeile auf Position 31 des Bildschirms oder Ausdrucks gesetzt.

Die beiden benötigten Dateien werden als primäre und sekundäre Datei eröffnet, wobei für die Kundendatei die nach Kundennummern sortierte Indexdatei **kundnum** benutzt wird:

```
USE adressen
SELECT SECONDARY
USE kunden INDEX kundnum
```

Anmerkung: Es wäre auch möglich, zunächst nur die Datei **adressen** und erst später unmittelbar vor dem Zugriff die Datei **kunden** zu eröffnen, ohne mit SELECT SECONDARY zu arbeiten. Bei einmaligem Programmdurchlauf sind beide Wege gleichwertig. Bei einer mehrmaligen Abfrage mittels Programmschleife, wie im nächsten Beispiel gezeigt, ist aber die gewählte Methode wesentlich schneller, weil die Dateien nicht immer wieder neu von

der Diskette gelesen werden müssen. Ebenso dient die Benutzung der Indexdatei **kundnum** anstelle ihrer Stammdatei ausschließlich der Steigerung der Verarbeitungsgeschwindigkeit.

Zum Suchen der Kundenanschrift muß die primäre Datei **adressen** gewählt und der gewünschte Name des Kunden eingegeben werden. Hierfür soll auf dem Bildschirm vier Zeilen unter der Überschrift die Aufforderung zur Eingabe des Namens erscheinen:

```
SELECT PRIMARY
@ 5,5
ACCEPT 'Name des Kunden' TO Mname
LOCATE FOR Mname$name
```

Der eingegebene Name wird im Speicher **Mname** abgelegt und dann im Suchbefehl LOCATE mit den Namen in der Datei verglichen. Dabei wird die Substring-Funktion "$" benutzt, mit deren Hilfe auch unvollständig eingegebene Namen gefunden werden können. Es genügt also, nur einige Buchstaben aus dem Namen des Kunden anzugeben, solange der Kunde damit eindeutig identifiziert werden kann. (Selbstverständlich ist es auch möglich, das Programm derart zu erweitern, daß bei mehrdeutigen Eingaben alle in Frage kommenden Namen zur Auswahl angezeigt werden.)

Ist der Kunde in der Datei gefunden, dann können Anschrift und Kundennummer ausgedruckt werden. Entsprechend dem gewünschten Druckbild ist die Formatierung wie folgt anzugeben:

```
SET FORMAT TO PRINT
@ 1,30 SAY 'Kunden-Information'
@ 5,5 SAY TRIM(vorname) + ' ' + name
@ 5,50 SAY 'Kunden-Nr. ' + kunummer
@ 7,5 SAY strasse
@ 8,5 SAY ort
```

Diese Gruppe von Formatier-Befehlen enthält eine neue Funktion

TRIM (Feldname)

(TRIM = abschneiden) die aus dem Datenfeld **vorname** die dem Vornamen folgenden Leerzeichen löscht, soweit solche vorhanden sind. Das heißt, das Datenfeld wird auf die tatsächliche Länge seines Inhalts verkürzt und gestattet damit, den folgenden Nachnamen in unmittelbarem Anschluß an den

Vornamen zu drucken. Vorname und Nachname sind nicht durch unnötig viele Leerstellen dazwischen getrennt, sondern wie üblich nur durch eine einzige, die aber jetzt besonders mit ' ' anzugeben ist.

Wie Sie weiter aus den Befehlen entnehmen können, lassen sich innerhalb einer SAY-Anweisung mehrere Datenfelder oder Texte mittels "+"-Zeichen aneinanderketten. Dies ist aber nur dann erlaubt, wenn alle Datenfelder dieser Zeile vom Typ "alphanumerisch" sind. Es dürfen nicht gleichzeitig numerische und alphanumerische Felder in derselben Befehlszeile erscheinen, wohl aber in derselben Druckzeile.

Bevor aus der sekundären Datei die restlichen Daten gesucht werden können, muß die Kundennummer als gemeinsames Identifikationsmerkmal in einem Speicher gepuffert werden, von wo sie als Suchbegriff für das folgende FIND-Kommando entnommen wird:

```
STORE kunummer TO Mnr
SELECT SECONDARY
FIND &Mnr
```

Sie erinnern sich, daß als Argument des Befehls FIND der Suchbegriff in der Indexdatei sowohl direkt als auch indirekt angegeben werden kann. Im vorliegenden Fall handelt es sich um eine indirekte Angabe, und zwar soll nach demjenigen Begriff gesucht werden, der in dem Speicher **Mnr** steht. Die Funktion

&(Name)

gestattet, den Befehl FIND mit beliebigen Kundennummern auszuführen, ohne den Befehl selbst immer wieder anpassen zu müssen. Mit Hilfe der &-Funktion kann man Dateien indirekt ansprechen, was zu hochinteressanten Möglichkeiten führt, von denen eine später noch als Beispiel gezeigt wird.

Nach Auffinden des gewünschten Datensatzes stehen die restlichen Informationen zur Verfügung und sind wieder formatiert auszudrucken, wie der untere Teil des Bildes 4.6 zeigt. Beachten Sie, daß - wie schon erwähnt - numerische und alphanumerische Datenfelder nicht gemeinsam in einem Druckbefehl stehen dürfen.

```
* Programm info
ERASE
@ 1,30 SAY 'Kunden-Information'
USE adressen
SELECT SECONDARY
USE kunden INDEX kundnum
SELECT PRIMARY
@ 5,5
ACCEPT 'Name des Kunden ' TO Mname
LOCATE FOR Mname$name
SET FORMAT TO PRINT
@ 1,30 SAY 'Kunden-Information'
@ 5,5 SAY TRIM(vorname) + ' ' + name
@ 5,50 SAY 'Kunden-Nr. ' + kunummer
@ 7,5 SAY strasse
@ 8,5 SAY ort
STORE kunummer TO Mnr
SELECT SECONDARY
FIND &Mnr
@ 12,5 SAY 'Letzte Rechnung vom: ' + datum
@ 15,5 SAY 'bisheriger Umsatz: DM'
@ 15,35 SAY umsatz
@ 17,5 SAY 'offene Posten:     DM'
@ 17,35 SAY offen
SET FORMAT TO SCREEN
RETURN
```

Bild 4.6: Das vollständige Programm **info**

Daher müssen die Beträge **umsatz** und **offen** getrennt von ihrem Begleittext formatiert werden, stehen aber im Ausdruck jeweils in einer Zeile. Sie werden später eine Möglichkeit kennenlernen, wie man numerische Felder in alphanumerische umwandeln und damit dieses Problem umgehen kann.

Bevor das Programm mit RETURN abgeschlossen wird, sollte immer die Ausgabe der formatierten Daten wieder auf den Bildschirm zurückgelegt werden:

SET FORMAT TO SCREEN

Andernfalls könnte es einmal vorkommen, daß Sie in einem nachfolgenden Programm Informationen auf dem Bildschirm vermissen und diese statt dessen gedruckt werden. Denn sämtliche Befehle in dBASE behalten ihre Wirkung auch über das Ende eines Programms hinaus solange bei, bis sie durch anderslautende Befehle aufgehoben werden oder bis dBASE beendet wird. So bleiben z.B. auch die Dateien über das Programmende hinaus geöffnet, bis neue Dateien aufgerufen und sie selbst dadurch geschlossen werden.

Mit CTRL-W wird die Erstellung des Programms abgeschlossen. Anschließend können die Informationen über die Kunden mit

DO info

abgefordert werden, wobei jeweils nach kurzer Zeit (die für die Eröffnung der beiden Dateien benötigt wird) das Programm nach dem Namen des Kunden fragt und dann die entsprecheden Daten liefert. Bild 4.7 zeigt ein Beispiel.

```
                    Kunden-Information


Werner Meister                          Kunden-Nr. 098

Marktplatz 3
8057 Eching


Letzte Rechnung vom: 15/08/85

bisheriger Umsatz: DM          5612.89

offene Posten:     DM          1122.58
```

Bild 4.7: Informationsblatt über einen einzelnen Kunden

4.2 Programmschleifen

Wenn Sie Informationen über mehrere Kunden unmittelbar hintereinander wünschen, werden Sie es als lästig empfinden, jedesmal das Programm mit "DO info" aufrufen zu müssen. Zudem werden jedesmal die zwei Dateien erneut eröffnet, d.h., von der Diskette gelesen, was unnötig viel Zeit kostet.

Der wiederholte Zugriff auf die Kundendaten läßt sich wesentlich eleganter realisieren, indem die Dateien nur einmal eröffnet werden und dann der eigentliche Abfrageteil als Programmschleife ausgebildet wird, die beliebig oft durchlaufen werden kann.

Das nachfolgend beschriebene Programm **info2**, das eine solche Schleife enthält, ist weitgehend identisch mit dem eben besprochenen und könnte sehr leicht aus diesem entwickelt werden. Da aber hierfür einige Programmzeilen einzufügen, andere zu löschen sind, ist es bei diesen ersten Beispielen der Übersichtlichkeit wegen zweckmäßiger, das Programm neu zu schreiben.

Das gesamte Programm kann nach Bild 4.8 aufgebaut werden, wobei nur noch die Passagen näher erläutert werden sollen, die gegenüber dem ersten Programm neu sind.

Wie schon im letzten Abschnitt gesagt, behalten die dBASE-Befehle ihre Wirkung auch über das Ende eines Programms bei. Das heißt, wenn Sie später das Programm **info2** starten, dann könnte zum Beispiel ein früher gegebener Befehl SELECT SECONDARY noch wirksam sein. Als Folge würde die erste Datei, die Sie im neuen Programm eröffnen, von dBASE als sekundäre Datei behandelt werden und nicht - wie erwartet - als primäre, was letztlich zu einem Scheitern des Programms führen würde.

Es ist daher unbedingt empfehlenswert, vor Eröffnung der ersten Datei den Befehl

SELECT PRIMARY

zu geben, damit die Zuordnung in jedem Fall eindeutig ist.

```
* Programm info2

ERASE
@ 1,30 SAY 'Kunden-Information'
SELECT PRIMARY
Use adressen
SELECT SECONDARY
Use kunden INDEX kundnum
STORE 'J' TO antwort
DO WHILE !(antwort) <> 'N'
     ERASE
     SELECT PRIMARY
     @ 5,5
     ACCEPT 'Name des Kunden' TO Mname
     LOCATE FOR Mname$name
     ERASE
     @ 1,30 SAY 'Kunden-Information'
     @ 5,5 SAY TRIM(vorname) + ' ' + name
     @ 5,50 SAY 'Kunden-Nr. ' + kunummer
     @ 7,5 SAY strasse
     @ 8,5 SAY ort
     STORE kunummer TO Mnr
     SELECT SECONDARY
     FIND &Mnr
     @ 12,5 SAY 'Letzte Rechnung vom: ' + datum
     @ 15,5 SAY 'bisheriger Umsatz: DM' + STR(umsatz,10,2)
     @ 17,5 SAY 'offene Posten:     DM' + STR(offen,10,2)
     @ 21,5 SAY 'weitere Informationen: ENTER,;
                                                    sonst  N eingeben'
     WAIT TO antwort
     SELECT PRIMARY
ENDDO
```

Bild 4.8: Das Programm **info2**

Programmschleifen werden in dBASE mit dem Befehl

DO WHILE (Bedingung)

eingeleitet, was übersetzt etwa soviel bedeutet wie "Führe (die folgenden Befehle) solange aus, wie die Bedingung erfüllt ist". Daraufhin werden alle nachfolgenden Programmschritte ausgeführt, bis der Befehl

ENDDO

erreicht ist, der das Ende der DO-WHILE-Schleife markiert. Dann wird diese erneut durchlaufen.

Ist die Bedingung nicht mehr erfüllt, dann wird die DO-WHILE-Schleife verlassen und das Programm zu Ende geführt. Dazu ist es notwendig, daß die Parameter der Bedingung verändert und die Bedingung selbst geprüft werden können.

Um den Durchlauf einer Programmschleife in Gang zu setzen, ist durch geeignete Vorwahl der Parameter dafür zu sorgen, daß die Bedingung zunächst auf jeden Fall erfüllt ist. Die häufigste Art, Parameter für die Bedingung festzulegen, ist die Vorgabe bestimmter Werte in einem Speicher. Im vorliegenden Beispiel wird der Buchstabe "J" (für "Ja") in dem Speicher mit Namen **antwort** abgelegt:

```
STORE 'J' TO antwort
```

Der Speicherinhalt soll später darüber Auskunft geben, ob weitere Informationen über Kunden erwünscht werden. Deshalb wird er vor der ersten Abfrage auf "Ja" gesetzt, kann aber im Verlauf des Programms mit "N" für "Nein" belegt werden, so daß die darauffolgende Abfrage zur Beendigung des Programms führt.

Der Befehl für die Eröffnung der Programmschleife ist dann folgendermaßen zu formulieren:

```
DO WHILE antwort < > 'N'
```

oder in diesem Fall besser:

```
DO WHILE !(antwort) < > 'N'
```

Bei beiden Schreibweisen soll die Programmschleife solange durchlaufen werden, wie der Inhalt des Speichers **antwort** verschieden von "N" ist. In der zweiten Variante wird aber eine "Großbuchstaben-Funktion"

!(Text)

verwendet, die alle Kleinbuchstaben des genannten Texts oder Speicherinhalts in Großbuchstaben umwandelt. Damit werden stets Großbuchstaben der "**antwort** mit dem Großbuchstaben "N" verglichen, unabhängig davon, mit welcher Buchstabengröße Sie die **antwort** eingeben.

Im Anschluß an den Befehl DO WHILE folgt die Befehlskette, die Sie vom vorhergehenden Beispiel schon kennen, wobei die Befehle zur Ausgabe auf den Drucker herausgenommen und weiter unten die Befehle zur Anzeige von **umsatz** und **offen** etwas modifiziert wurden, worauf gleich noch eingegangen wird.

Zur besseren Übersichtlichkeit werden alle Befehle innerhalb der DO-WHILE-Schleife um einige Zeichen eingerückt. Dies ist keine Notwendigkeit seitens dBASE, sondern dient ausschließlich der besseren Lesbarkeit des Programms und sollte grundsätzlich getan werden. Sie sehen dann sofort, an welcher Stelle der Befehl ENDDO die Schleife abschließt.

Nach Ausgabe der Informationen über den ersten Kunden muß nun eingegeben und abgefragt werden, ob weitere Informationen über andere Kunden erwünscht sind. Dazu wird eine Hinweiszeile auf dem Bildschirm eingeblendet:

@ 21,5 SAY 'weitere Informationen: ENTER, sonst N eingeben'

Anschließend erfolgt ein Befehl

WAIT TO antwort

der die allgemeine Form

WAIT TO (Speichername)

hat und der das Programm anhalten läßt, bis Sie über die Tastatur eine Eingabe machen. Diese wird in den Speicher mit Namen **antwort** geschrieben und kann anschließend an beliebiger Stelle im Programm abgefragt werden. In diesem Fall wird der Speicherinhalt zu Beginn der folgenden Schleife im Befehl

DO WHILE !(antwort) < > 'N'

ausgewertet. Bei 'N' oder 'n' als Inhalt des Speichers wird das Programm beendet.

Im Unterschied zu den Befehlen ACCEPT und INPUT, mit denen ganze Wörter oder Zahlen eingegeben werden und die mit der ENTER-Taste abgeschlossen werden, nimmt der Befehl WAIT nur eine einzelne Taste als Eingabe an und braucht nicht mit ENTER abgeschlossen zu werden. Damit bietet WAIT eine sehr schnelle Eingabemöglichkeit insbesondere für Fälle, in denen eine Auswahl aus verschiedenen Alternativen zu treffen ist, wie es typischerweise in der später gezeigten Menütechnik vorkommt.

Mit der hier gezeigten Programmierung wird die Ausgabe von Kundeninformationen mit jeder Taste fortgesetzt, solange es nicht die Taste "N" ist. Es muß also nicht unbedingt die ENTER-Taste zur Fortsetzung gedrückt werden (was aber ohne weiteres realisierbar wäre).

Der Befehl WAIT kann auch ohne den Zusatz "TO (Speicher)" benutzt werden. In diesem Fall hält das Programm bei WAIT an und wird auf einen beliebigen Tastendruck hin fortgesetzt. Das ist manchmal recht zweckmäßig, um irgendwelche Zwischenergebnisse kurz zu betrachten.

Bei der Besprechung des Kommandos SAY wurde gesagt, daß numerische Felder nicht in einer Befehlszeile mit alphanumerischen Feldern erscheinen dürfen. Diese Einschränkung kann mit der Funktion

STR(numerischer Ausdruck,Länge,Dezimalstellen)

(STR = Abkürzung für STRING = Zeichenfolge) umgangen werden. Mit STR wird ein numerischer Ausdruck in eine alphanumerische Zeichenkette umgewandelt und kann dann als solche entsprechend benutzt werden. Als numerischer Ausdruck kann wie üblich eine Zahl, ein Speichername oder auch ein Feldname stehen. Die Länge des umzuwandelnden Ausdrucks (in

Anzahl Zeichen) muß, die Anzahl der anzuzeigenden Dezimalstellen kann angegeben werden.

Die Funktion STR wird hier benutzt, um die Werte für den Umsatz und die Außenstände in einer Befehlszeile mit ihrem Begleittext unterzubringen:

```
@ 15,5 SAY 'bisheriger Umsatz: DM' + STR(umsatz,10,2)

@ 17,5 SAY 'offene Posten:     DM' + STR(offen,10,2)
```

Es ist zu beachten, daß ein mit STR behandelter Ausdruck alle Eigenschaften einer numerischen Zahl verliert und daher nicht in arithmetischen Operationen weiterverwendet werden kann.

Mit dem hier erstellten Programm können Sie nun beliebige Kunden in beliebiger Reihenfole abfragen. Sie werden dabei feststellen, daß die Antwortzeiten des Systems wesentlich schneller gegenüber dem ersten Programm **info** geworden sind, weil die einmal eröffneten Dateien im Zugriff bleiben und nicht immer wieder von der Diskette gelesen werden müssen. Durch Druck auf die Taste "N" kann das Programm verlassen werden.

Anmerkung: Falls Sie sich über das erstellte Programm ein Protokoll anfertigen möchten, was in der Praxis unbedingt getan werden sollte, können Sie dies machen, indem Sie dBASE verlassen und vom Betriebsystem aus das Programm mit den Befehlen

```
CTRL-P          (= Drucker einschalten) und

TYPE info2.CMD
```

drucken lassen. Dabei dürfen Sie nicht vergessen, den Dateityp **CMD** anzugeben. Gleiches gilt für das Protokollieren des Programms über ein Textverarbeitungssystem, was ebenso möglich ist.

4.3 Programmverzweigungen

Mit dem Befehl DO können Programme nicht nur vom Benutzer gestartet, sondern auch von anderen Programmen aus abgerufen werden. Es ist meistens möglich (und vorteilhaft), anstatt eines sehr großen Programms mehrere kleine zu entwickeln und die einzelnen Programme miteinander zu verketten. Dies führt zu einer gewissen Strukturierung des Gesamtprogramms, die sich sehr positiv auf die Übersichtlichkeit und damit auf die Wartungsfreundlichkeit auswirkt.

Besonders nützlich ist die Arbeitweise mit Unterprogrammen in der sogenannten Menütechnik, bei der alle wesentlichen Funktionsblöcke einer Anwendung auf dem Bildschirm angezeigt werden und der Benutzer durch Auswahl über die Tastatur das gewünschte Programm aufruft. Nach Abschluß des Programms kehrt das System zum Menü zurück und ermöglicht die erneute Auswahl einer Funktion.

Als Beispiel für eine Programmverzweigung soll ein Menüprogramm entwickelt werden, das verschiedene Einzelprogramme, unter anderem auch das Abfrage-Programm **info2**, in ein Programmpaket zur Bearbeitung aller kundenbezogenen Dateien einbindet. Im einzelnen sollen folgende Funktionen ermöglicht werden:

- eine Bildschirm-Information über individuelle Kunden
- die Ausgabe einer Adressenliste aller Kunden
- die Aufnahme neuer Kunden in das System
- die Bearbeitung neuer und alter Rechnungen.

Das Menüprogramm besteht prinzipiell aus zwei Teilen: aus dem Teilprogramm zur Anzeige des Menüs (Bild 4.9) auf dem Bildschirm und aus dem Abfrageteil mit den Verzweigungsbefehlen. Beide Teile werden wieder in eine DO WHILE-Schleife eingebunden, damit der Benutzer nach Erledigung eines Unterprogramms wieder zum Menü zurückgeführt wird.

```
                    Programm-Auswahl


          (1)  Informationen über einzelne Kunden

          (2)  Kundenadressen auflisten

          (3)  neue Kunden aufnehmen

          (4)  Rechnungs-Bearbeitung

          (0)  ENDE



          Bitte wählen Sie eine Funktion
```

Bild 4.9: Bildschirmmaske zur Programmauswahl

Eröffnen Sie die neue Programmdatei, die den Namen **menue** erhalten soll, mit

MODIFY COMMAND menue

Zu Beginn dieses Hauptprogramms müssen noch keine Dateien eröffnet werden; dies erfolgt erst individuell in den späteren Unterprogrammen. Daher kann sofort mit dem Befehl DO WHILE begonnen werden:

STORE T TO menue

DO WHILE menue

Dies ist eine neue Art, die Bedingung für DO WHILE anzugeben, die kurz erklärt werden muß: Zunächst wird in den Speicher **menue** (beachten Sie, daß der Name **menue** sowohl für den Speicher als auch für das Programm verwendet werden darf, da es sich ja um verschiedene Dateitypen handelt) die Konstante "T" geladen. "T" steht für "TRUE" (= wahr) und ist ein in dBASE fest definierter logischer Wert. Das Gegenstück dazu ist der logische Wert "F" für "FALSE" (= falsch), der ebenfalls konstant ist. Die Symbole beider Werte werden stets ohne Anführungszeichen geschrieben, wodurch sie sich von den normalen alphanumerischen Zeichen "T" und "F" unterscheiden.

Der Befehl "DO WHILE menue" bedeutet, daß die Programmschleife solange durchlaufen werden soll, wie der Inhalt des Speichers **menue** den logischen Wert "T" besitzt.

Noch kürzer kann man die gesamte Anweisung so schreiben:

```
DO WHILE T
```

Hier lautet die Bedingung: "solange, wie die Konstante "T" den logischen Wert "T" besitzt." Da dies per Definition der Fall ist, kann man auf diese Weise eine Schleife ohne Ausgang aufbauen. Selbstverständlich müssen dann andere Möglichkeiten zum Verlassen der Schleife, nämlich bedingte Programmverzweigungen, geschaffen werden.

Der Programmabschnitt zum Aufbau der Bildschirmmaske beginnt mit einem Löschbefehl und bringt dann die bekannten Befehle zur Formatierung des Bildschirms (Bild 4.10, oberer Teil). Danach steht die Aufforderung, durch Druck auf eine der genannten Zifferntasten das gewünschte Programm aufzurufen. Da zur Auswahl einer der Alternativen eine einzige Ziffer ausreicht, kann wieder der Befehl WAIT TO benutzt werden:

```
WAIT TO antwort
```

Abhängig von der betätigten Zifferntaste muß dann in eines der fünf Unterprogramme verzweigt werden.

```
STORE T TO menue
DO WHILE menue
     ERASE
     @ 1,30 SAY 'Programm-Auswahl'
     @ 5,15 SAY '(1)  Informationen über einzelne Kunden'
     @ 7,15 SAY '(2)  Kundenadressen auflisten'
     @ 9,15 SAY '(3)  neue Kunden aufnehmen'
     @ 11,15 SAY '(4)  Rechnungs-Bearbeitung'
     @ 13,15 SAY '(0)  ENDE'
     @ 19,15 SAY 'Bitte wählen Sie eine Funktion'
     WAIT TO antwort
     DO CASE
             CASE antwort = '1'
                 DO info2
             CASE antwort = '2'
                 USE adressen INDEX alphabet
                 ERASE
                 @ 1,1 SAY ' Kunde  Straße   Wohnort  Kunden-Nr.'
                 DISPLAY OFF ALL $(vorname,1,1)+'.',name,strasse,ort,;
                                              kunummer
                 WAIT
             CASE antwort = '3'
                 USE adressen INDEX alphabet
                 APPEND
             CASE antwort = '4'
                 ?
             CASE antwort = '0'
                 RETURN
    ENDCASE
ENDDO
```

Bild 4.10: Programm für ein Bildschirm-Menü

Für Programmverzweigungen existieren in dBASE zwei verschiedene Befehle, die sehr ähnlich gehandhabt werden. Zunächst der Befehl, der in praktisch allen anderen Programmiersprachen ebenfalls zu finden ist:

IF (Bedingung)
(Befehle für erfüllte Bedingung)
ELSE
(Befehle für nicht erfüllte Bedingung)
ENDIF

Ein Befehlskomplex, der mit IF (= falls) eingeleitet wird, muß stets mit dem Befehl ENDIF als Ende-Markierung abgeschlossen werden. Wenn die IF-Bedingung erfüllt ist, dann werden die unmittelbar folgenden Programmschritte ausgeführt, und zwar alle bis zu dem Befehl ELSE. Von dort an wird der Rest bis zu ENDIF übersprungen und erst danach das Programm fortgesetzt. Wenn die IF-Bedingung nicht erfüllt ist, werden nur die dem ELSE (= andernfalls) folgenden Schritte ausgeführt. Die Anweisung ELSE mit den entsprechenden Programmschritten kann aber ganz entfallen. Dann werden bei Nichterfüllung der IF-Bedingung alle weiteren Befehle bis zu ENDIF ignoriert.

Ein Beispiel für eine Verzweigung mit IF wird im Anschluß an dieses Menü gezeigt. Zunächst soll die zweite Möglichkeit der Programmverzweigung angewandt werden, da diese bei mehr als zwei Alternativen übersichtlicher, kürzer und schneller ist. Es ist dies die Kommandogruppe

DO CASE
CASE (Bedingung)
(Befehle für erfüllte Bedingung)
:
:
ENDCASE

Mit DO CASE (CASE = falls) wird ein Block von Bedingungen und davon abhängigen Schritten eingeleitet und mit ENDCASE abgeschlossen. Dazwischen können beliebig viele CASE-Bedingungen mit den gegebenenfalls auszuführenden Befehlen eingefügt werden.

Im Gegensatz zu IF bzw. ENDIF muß bei Verwendung von CASE nur ein einziges ENDCASE für sämtliche CASE-Bedingungen gesetzt werden. Ein weiterer Unterschied besteht darin, daß nach der ersten erfüllten CASE-Bedingung alle weiteren übersprungen werden, während bei IF in jedem Fall alle IF-Bedingungen abgefragt werden, was mehr Verarbeitungszeit kostet. Im vorliegenden Menüprogramm ist als Bedingung jeweils abzufragen, welche der Zifferntasten gedrückt wurde. Also lauten die entsprechenden Befehle:

CASE antwort = '1' , CASE antwort = '2' usw.

Hinter jeder Bedingung ist die resultierende Befehlsfolge, in die verzweigt werden soll, anzugeben, also

```
CASE antwort = '1'
   DO info2
```

für die Betätigung der Taste 1. Wird diese gedrückt, dann ruft dBASE das früher erstellte Programm info2 zur Verarbeitung auf.

Nach Betätigen der Taste 2 soll eine Adressenliste aller Kunden erstellt werden. Dazu wird die nach Namen indizierte Datei **adressen** eröffnet und gelistet. (Vergessen Sie nicht, die Indexdatei zuvor mit INDEX ON name TO alphabet zu erstellen!) Anstatt aber die Adressen mit LIST anzuzeigen, werden sie mit DISPLAY abgerufen, damit der Betrachter seitenweise alle Adressen durchgehen kann (wenn es mehr als 15 Adressen sind). Zusätzlich ist noch ein Befehl WAIT angefügt, der die letzten 15 Adressen solange auf dem Bildschirm hält, bis irgendeine Taste betätigt wird und das Menü wieder erscheint.

Bei der Auflistung der Kundennamen wird wieder eine Substring-Funktion benutzt, und zwar

```
$(vorname,1,1)
```

die von den Vornamen der Kunden nur jeweils den ersten Buchstaben nimmt und dem Nachnamen voranstellt. Aus optischen Gründen wurde hinter den Anfangsbuchstaben noch ein Punkt gesetzt:

```
$(vorname,1,1)+'.'
```

Im Fall "Taste 3" sollen neue Kundenadressen aufgenommen werden. Dazu wird wieder die Indexdatei benutzt und mit APPEND ein neuer Datensatz aufgenommen. Da die Indexdatei nach Kundennamen geordnet ist, werden die neu aufgenommenen Kunden sofort alphabetisch richtig eingeordnet, ohne daß ein zusätzlicher Sortierlauf notwendig wäre.

Die Alternative 4 existiert noch nicht. Sie können zwar die Bedingung "CASE antwort ='4'" schreiben, aber eine Anweisung für ein nicht existierendes Programm, z.B. DO rechnung, sollte man nicht verwenden. Andernfalls würde bei Betätigung der Taste 4 dieses Unterprogramm vergeblich gesucht und das gesamte Programm mit einer Fehlernachricht abgebrochen werden. Als Hinweis für sich selbst kann man aber z.B. ein Fragezeichen setzen, um anzudeuten, daß hier noch ein Programmteil nachzutragen ist.

Mit Betätigung der Taste "0" soll das Menüprogramm beendet werden. Dazu ist lediglich RETURN zu schreiben. Wenn Sie mit dem fertigen Programm **menue** arbeiten, dann können Sie die Funktionen 1 bis 3 benutzen, die Sie schon einzeln kennengelernt haben (siehe Bild 4.11). Auf Betätigung der Taste 4 hin wird der Bildschirm kurz gelöscht, dann aber wird sofort das Menü wieder aufgebaut. Das gilt übrigens für die Betätigung jeder nicht spezifizierten Taste und ist ein Zeichen dafür, daß sämtliche Bedingungen abgefragt wurden, aber kein Unterprogramm ausgelöst wurde, d.h. die DO-WHILE-Schleife wurde genau einmal rasch durchlaufen. Die einzige Möglichkeit, das Menüprogramm zu verlassen, ist die Betätigung der Taste "0".

Kunde	Straße	Wohnort	Kunden-Nr.
J. Becker	Hauptstr.28	6100 Darmstadt	357
M. Behr	Frankenweg 17	8500 Nürnberg	421
F. Fischer	Ginsterweg 9	6850 Mannheim	852
P. Hassold	Meisenstr.18	2000 Hamburg	147
G. Hofman	Holzweg 13	2000 Hamburg	258
J. Huber	Sieglindenstr.5	8000 München	234
K. Knoll	Amselweg 23	1000 Berlin	963
W. Koch	Heumarkt 14	6000 Frankfurt	159
P. Kuhn	Fliederstr.9	6100 Darmstadt	345
H. Maier	Amalienstr.59	8000 München	123
J. Meier	Flurstr.3	8000 München	369
W. Meister	Marktplatz 3	8057 Eching	098
H. Schmid	Fasanenstr.2	1000 Berlin	761

Bild 4.11: Die vom Programm **menue** erstellte Kundenliste

4.4 Online-Programme

Begünstigt durch die hohen Rechenleistungen heutiger Computer herrscht in der Softwareentwicklung allgemein der Trend vor, neue Programme nach Möglichkeit so zu schreiben, daß die Daten direkt erfaßt und sofort verarbeitet werden, um permanent die aktuellsten Informationen verfügbar zu haben. Man spricht dann von einer Online-Verarbeitung. Um diese zu realisieren, müssen neben einer ausreichenden Systemleistung vor allem geeignete Befehle zur direkten Datenerfassung angeboten werden.

Im Laufe dieser Einführung haben Sie bereits einige Möglichkeiten kennengelernt, Daten über die Tastatur zu erfassen und in eine Datenbank aufzunehmen. Neben den Befehlen ACCEPT und INPUT war dies primär der Befehl APPEND, der das Erfassen kompletter Datensätze und ihr Anfügen an das Ende einer Datei erlaubt. APPEND zeichnet sich besonders dadurch aus, daß automatisch eine Erfassungsmaske zur Verfügung gestellt wird.

Natürlich hat ein derartiger Komfort auch seine Nachteile, die vor allem in einer gewissen Unflexibilität liegen. Die Erfassungsmaske des Befehls APPEND ist starr: es werden stets alle Felder eines Datensatzes gezeigt, wobei sie in derjenigen Reihenfolge auf dem Bildschirm erscheinen, die bei der Definition der Dateistruktur vorgegeben wurde. Tatsächlich werden aber in der Praxis oft nur einige Felder eines Datensatzes und diese vielleicht in anderer Reihenfolge für die Erfassung benötigt. Dann ist es zweckmäßig, eine individuelle Erfassungsmaske zu programmieren.

Prinzipiell läßt sich eine solche Maske mit den bekannten Befehlen zur Bildschirmformatierung (@ .... SAY) und den Kommandos ACCEPT bzw. INPUT realisieren, wie bereits für einzelne Daten in den Abfrageprogrammen gezeigt wurde. dBASE verfügt aber auch über einen weiteren Befehl, der speziell für die Datenerfassung geschaffen wurde:

GET (Feldname) mit dem Folgebefehl **READ**

GET vereinigt in sich in gewissem Sinne die Eigenschaften aller drei Befehle ACCEPT, INPUT und APPEND. Zusammen mit entsprechenden Formatierbefehlen kann eine individuelle Editiermaske erstellt werden, die sowohl numerische als auch alphanumerische Daten aufnimmt. Innerhalb

einer GET-Maske sind alle bekannten Editierfunktionen verfügbar. Wenn das letzte GET-Feld mit ENTER abgeschlossen wird, werden alle Daten der Erfassungsmaske gemeinsam eingelesen. Auf diese Weise können bis zu 64 Felder gleichzeitig editiert und mit einem Lesebefehl erfaßt werden.

Zur Demonstration des Befehls GET soll nun ein Programm geschrieben werden, das sowohl neu erstellte Rechnungen als auch Zahlungseingänge von Kunden in einer Rechnungsdatei erfaßt. Gleichzeitig sollen die erfaßten Daten auch zum Fortschreiben der Kundendatei dienen. Anschliessend wird das Programm in das zuvor erarbeitete Menü aufgenommen und kann von dort aus abgerufen werden.

Der Programm-Name soll **rechnung** sein:

MODIFY COMMAND rechnung

Die Programmbeschreibung soll sich wieder auf die neu hinzugekommenen Befehle und Routinen beschränken. Das vollständige Programm ist in Bild 4.14 aufgeführt.

Sie wissen, daß neben den programmierten, also erwarteten Systemantworten auch gewisse von dBASE veranlaßte Nachrichten wie z.B. "00015 Sätze kopiert" usw. auf dem Bildschirm erscheinen. Diese Meldungen sind in der Anfangsphase recht nützlich, für den erfahreneren Benutzer aber nicht mehr unbedingt notwendig. Das gilt besonders dann, wenn äquivalente Nachrichten ohnehin einprogrammiert sind.

Für das folgende Programm sollen diese Systemnachrichten ausgeblendet werden, wozu der Befehl

SET TALK OFF

als eine weitere Variante von SET benutzt wird. Bevor das Programm wieder verlassen wird, also unmittelbar vor einem RETURN-Befehl, sollte der Dialog mit

SET TALK ON

wieder eingeschaltet werden, damit er gegebenenfalls für andere Anwendungen wieder zur Verfügung steht.

Als primäre Datei wird die nach Rechnungsnummern indizierte Datei **rechnung** und als sekundäre Datei die Indexdatei **kundnum** eröffnet:

```
SET TALK OFF
ERASE
@ 2,12 SAY 'Rechnungs-Bearbeitung'
SELECT PRIMARY
USE rechnung INDEX nummer
SELECT SECONDARY
USE kunden INDEX kundnum
```

Da die Möglichkeit bestehen soll, mehrere Rechnungen unmittelbar hintereinander zu bearbeiten, wird der gesamte folgende Programmteil wieder in eine DO WHILE-Schleife eingebettet:

```
DO WHILE T
```

Wie Sie wissen, ist dies eine Schleife ohne Ausgang. Es muß also später im Programm für einen speziellen Ausgang gesorgt werden.

Für das gesamte Programm sind zwei Fälle zu unterscheiden: im ersten Fall wird eine neue Rechnung erfaßt (Bild 4.12), während im zweiten Fall mit einem Zahlungseingang ein existierender Rechnungssatz fortgeschrieben wird (Bild 4.13). Da unter Umständen der Benutzer unsicher ist, ob eine Rechnung bereits erfaßt ist, soll das Programm dies selbst überprüfen und auch selbst in das entsprechende Unterprogramm (neue Rechnung oder Zahlungseingang) verzweigen. Das ist recht einfach möglich, indem zunächst die Rechnungsnummer eingegeben und ihr Vorhandensein in der Rechnungsdatei geprüft wird:

```
ERASE
@ 5,1
ACCEPT 'Rechnungs-Nr. ' TO Mrn
SELECT PRIMARY
FIND &Mrn
```

```
          neue Rechnung

     Rechnungs-Nr.

     Kunden-Nr.:

     Betrag: DM




 für weitere Eingaben ENTER, sonst N drücken
```

Bild 4.12: Erfassungsmaske für neue Rechnungen

Ist die Rechnung bereits erfaßt, dann wird über die Rechnungsnummer ihr Datensatz gefunden. Ist sie neu, dann findet dBASE keinen Datensatz für diese Nummer. In diesem Fall (das gilt nur bei Anwendung des Befehls FIND) weist der Datensatzzeiger auf die laufende Satznummer 00000, für die zwar real kein Datensatz existiert, die aber wie jede andere in Befehlen verwendet werden kann.

Mit zwei CASE-Bedingungen, die sich auf die Satznummer beziehen, kann damit in entsprechende Programmteile verzweigt werden:

CASE # = 0 für die Satznummer 00000, wenn also kein Satz existiert und eine neue Rechnung aufzunehmen ist

CASE # > 0 für vorhandene Satznummern bzw. Rechnungen.

Nach dem Befehl

DO CASE

kann also für jeden der beiden Fälle eine eigene Erfassungsmaske und ein Befehlsblock zur Datenübernahme entworfen werden.

Für neue Rechnungen müssen die Rechnungsnummer, die Kundennummer und der Rechnungsbetrag über die Maske "neue Rechnung", Bild 3.10, eingegeben werden. Das Aufnehmen einer neuen Rechnung erfolgt wieder mit dem Befehl APPEND, allerdings mit einer Variante davon:

APPEND BLANK

APPEND BLANK hängt an die eröffnete Datei einen "leeren" Datensatz an, ohne diesen zu editieren. Die Datenerfassung erfolgt dann nicht über die APPEND-Maske, sondern über die individuell gestaltete Maske mittels GET-Anweisungen. Die Befehle hierfür sind schnell geschrieben:

```
CASE # = 0
   ERASE
   @ 2,14 SAY 'neue Rechnung'
   APPEND BLANK
   @ 5,10 SAY 'Rechnungs-Nr. ' + Mrn
   @ 8,10 SAY 'Kunden-Nr.: ' GET kunummer PICTURE '999'
   @ 10,10 SAY 'Betrag: DM ' GET betrag
   READ
```

Zusammen mit dem Befehl GET kann man die Formatierfunktion

PICTURE '(Format)'

(PICTURE = Bild) verwenden. Mit ihrer Hilfe läßt sich sicherstellen, daß die erfaßten Daten stets im selben Format gespeichert werden. Unkorrekt formatierte Daten werden richtiggestellt oder zurückgewiesen. Das hier angeführte Beispiel bewirkt, daß für die Eingabe der Kundennummern nur Ziffern, aber keine Buchstaben zugelassen werden. Auf ähnliche Weise kann man auch komplexere Datenformate vorgeben.

Mit dem Befehl READ werden die Daten für **kunummer** und **betrag** in den leeren neuen Datensatz geschrieben, sobald beide Werte eingegeben sind. Dazu müssen die Rechnungsnummer, die schon zu Programmbeginn eingegeben wurde, die aber noch in dem Speicher **Mrn** steht, und das aktuelle Datum in die Datei übernommen werden:

```
REPLACE rnr WITH Mrn
REPLACE datum WITH date()
```

Anschließend soll die Kundendatei fortgeschrieben werden. Um in ihr den Datensatz für den Kunden zu finden, wird die Kundennummer in einem Speicher **Mkunde** zwischengespeichert und als Suchbegriff in der sekundären Datei benutzt:

```
STORE kunummer TO Mkunde
SELECT SECONDARY
FIND &Mkunde
```

Zur Fortschreibung des Kundensatzes muß das Feld für die "letzte Rechnung" die neue Rechnungsnummer erhalten und müssen das Rechnungsdatum übernommen und der Rechnungsbetrag zum bisherigen Umsatz addiert werden. Schließlich sind die Außenstände neu zu berechnen:

```
REPLACE lrech WITH Mrn
REPLACE datum WITH date()
REPLACE umsatz WITH umsatz + P.betrag
REPLACE offen WITH umsatz - bezahlt
```

Damit ist die neue Rechnung in beide Dateien eingearbeitet. Die Frage an den Benutzer, ob weitere Rechnungen bearbeitet werden sollen, wird erst am Ende der Programmschleife gestellt, wenn auch die zweite Alternative, die Bearbeitung von Zahlungseingängen, geschrieben ist. Die Frage kann dann für beide Fälle gemeinsam benutzt werden.

Das Teilprogramm für die Zahlungseingänge, das mit CASE # > 0 eingeleitet wird, ist ganz analog aufgebaut. Zunächst wird wieder die Erfassungsmaske (Bild 4.13) erstellt. Neben der Rechnungsnummer als Identifikationsmerkmal werden der Rechnungsbetrag, etwaige bereits geleistete Teilzahlungen und die noch offene Differenz in die Maske eingeblendet. Vom Benutzer ist nur die eingegangene Zahlung, die ein Teilbetrag sein darf, einzugeben. Alle anderen Transaktionen werden von dBASE selbstständig durchgeführt.

Um die Zahlung später in die Kundendatei zu übernehmen, muß der Betrag zwischengespeichert werden. Der dafür vorgesehene Speicher **Mbezahlt** muß zunächst leer sein, d.h. eventuell vorausgegangene Zahlungen für andere Rechnungen müssen gelöscht sein. Dazu wird der Wert 0 in den Speicher geladen. Das Einblenden der Beträge, die den bisherigen Zahlungsstand wiederspiegeln, kann mit einem einzigen Befehl, dem ein Fragezeichen vorausgeht, erfolgen:

```
CASE # > 0
   ERASE
   STORE 0 TO Mbezahlt
   @ 2,15 SAY 'Zahlungseingang'
   @ 5,10 SAY 'Rechnungs-Nr. ' + rnr
   @ 8,4 SAY ' Betrag  Teilzahlung   offen'
   ? betrag,bezahlt,betrag - bezahlt
   @ 11,4 SAY 'neue Zahlung: DM' GET Mbezahlt
   READ
```

```
                    Zahlungseingang


          Rechnungs-Nr.


Betrag  Teilzahlung   offen
0.00      0.00      0.00

neue Zahlung: DM




für weitere Eingaben ENTER, sonst N drücken
```

Bild 4.13: Erfassungsmaske für Zahlungseingänge

Das Aktualisieren der Rechnungsdatei erfolgt mit zwei Befehlen:

```
REPLACE bezahlt WITH bezahlt + Mbezahlt
REPLACE bdatum WITH date()
```

Der bezahlte Betrag wird in das Feld **bezahlt** (zu den eventuell vorhandenen Teilzahlungen) addiert und das Datum des Zahlungseingangs eingetragen. Anschließend muß wieder die Kundennummer als Suchbegriff für die Kundendatei zwischengespeichert werden. Dort wird dann ebenfalls der bezahlte Betrag in das Feld **bezahlt** addiert und die Außenstände werden neu berechnet:

```
STORE kunummer TO Mkunde
SELECT SECONDARY
FIND &Mkunde
REPLACE bezahlt WITH bezahlt + Mbezahlt
REPLACE offen WITH umsatz - bezahlt
```

Damit ist auch das zweite Teilprogramm beendet, und beide Teile können nun mit

```
ENDCASE
```

abgeschlossen werden. Es bleibt lediglich noch der Ausgang aus dem Programm zu formulieren, der mit der Nachricht

```
@ 18,4 SAY 'für weitere Eingaben ENTER, sonst N drücken'
```

eingeleitet wird. Bei Betätigung der Taste "N" für Nein soll das Programm verlassen werden, sonst nicht:

```
WAIT TO antwort
IF !(antwort) = 'N'
   SET TALK ON
   RETURN
ENDIF
```

Als letzter Befehl wird schließlich die DO-WHILE-Schleife geschlossen. Das gesamte Programm **rechnung** ist noch einmal in Bild 4.14 dargestellt.

```
* Programm rechnung

SET TALK OFF
ERASE
@ 2,12 SAY 'Rechnungs-Bearbeitung'
SELECT PRIMARY
USE rechnung INDEX nummer
SELECT SECONDARY
USE kunden INDEX kundnum
DO WHILE T
     ERASE
     @ 5,1
     ACCEPT 'Rechnungs-Nr. ' TO Mrn
     SELECT PRIMARY
     FIND &Mrn
     DO CASE
             CASE # = 0
                     ERASE
                     @ 2,14 SAY 'neue Rechnung'
                     APPEND BLANK
                     @ 5,10 SAY 'Rechnungs-Nr. ' + Mrn
                     @ 8,10 SAY 'Kunden-Nr.: ' GET kunummer
                     @ 10,10 SAY 'Betrag: DM ' GET betrag
                     READ
                     REPLACE rnr WITH Mrn
                     REPLACE datum WITH date()
                     STORE kunummer TO Mkunde
                     SELECT SECONDARY
                     Find &Mkunde
                     REPLACE lrech WITH Mrn
                     REPLACE datum WITH date()
                     REPLACE umsatz WITH umsatz + P.betrag
                     REPLACE offen WITH umsatz - bezahlt
```

```
            CASE # > 0
                    ERASE
                    STORE 0 TO Mbezahlt
                    @ 2,15 SAY 'Zahlungseingang'
                    @ 5,10 SAY 'Rechnungs-Nr. ' + rnr
                    @ 8,4 SAY ' Betrag    Teilzahlung     offen'
                    ? betrag,bezahlt,betrag - bezahlt
                    @ 11,4 SAY 'neue Zahlung: DM' GET Mbezahlt
                    READ
                    REPLACE bezahlt WITH bezahlt + Mbezahlt
                    REPLACE bdatum WITH date()
                    STORE kunummer TO Mkunde
                    SELECT SECONDARY
                    FIND &Mkunde
                    REPLACE bezahlt WITH bezahlt + Mbezahlt
                    REPLACE offen WITH umsatz - bezahlt
    ENDCASE
    @ 18,4 SAY 'für weitere Eingaben ENTER, sonst N drücken'
    WAIT TO antwort
    IF !(antwort) = 'N'
            SET TALK ON
            RETURN
    ENDIF
ENDDO
```

Bild 4.14: Programm **rechnung** zur Rechnungsbearbeitung

Als letzte Programmierarbeit können Sie nun noch das Programm **rechnung** in das zuvor erstellte Menü einbinden, in welchem Sie nach der Bedingung "CASE antwort = '4'" das Fragezeichen durch die Anweisung

DO rechnung

ersetzen (Bild 4.15).

```
STORE T TO menue
DO WHILE menue
     ERASE
     @ 1,30 SAY 'Programm-Auswahl'
     @ 5,15 SAY '(1)  Informationen über einzelne Kunden'
     @ 7,15 SAY '(2)  Kundenadressen auflisten'
     @ 9,15 SAY '(3)  neue Kunden aufnehmen'
     @ 11,15 SAY '(4)  Rechnungs-Bearbeitung'
     @ 13,15 SAY '(0)  ENDE'
     @ 19,15 SAY 'Bitte wählen Sie eine Funktion'
     WAIT TO antwort
     DO CASE
             CASE antwort = '1'
                     DO info2
             CASE antwort = '2'
                     USE adressen INDEX alphabet
                     ERASE
                     @ 1,1 SAY ' Kunde   Straße   Wohnort  Kunden-Nr.'
                     DISPLAY OFF ALL $(vorname,1,1),name,strasse,ort,;
                                                    kunummer
                     WAIT
             CASE antwort = '3'
                     USE adressen INDEX alphabet
                     APPEND
             CASE antwort = '4'
                     DO rechnung
             CASE antwort = '0'
                     RETURN
     ENDCASE
ENDDO
```

Bild 4.15: Das Programm **menue** mit eingebundenem Programm zur Rechnungsbearbeitung

Zur Demonstration des Ablaufs der Rechnungsbearbeitung wollen wir drei Transaktionen, zwei Rechnungserfassungen und einen Zahlungseingang, durchführen und die Aktualisierung der Dateien beobachten.

Starten Sie dBASE vom Betriebssystem aus und geben Sie ein Datum ein, das in den Dateien möglichst noch nicht vorkommt, z.B. 22.08.85. Dann lassen sich die Transaktionen besser verfolgen.

Ferner sollten Sie sich die (Index)-Dateien **rechnung** und **kunden** in ihrem Zustand vor Beginn der Buchungsarbeiten ausdrucken lassen. Auch das wird Ihnen helfen, die Bewegungen in den Dateien zu beobachten. Wenn Sie alle bisherigen Beispiele nachvollzogen und keine eigenen neuen Datensätze hinzugefügt haben, dann entsprechen die Dateiinhalte den Abbildungen 4.16 und 4.17.

Sollten Ihre Datensätze inzwischen anders aussehen, dann ist das auch kein Problem, erfordert aber etwas mehr Aufmerksamkeit, wenn Sie die Dateien vor und nach den Transaktionen vergleichen.

```
. USE rechnung INDEX nummer
. LIST

00001  11115 15/08/85 963    1591.50     1200.00 20/08/85
00002  11116 15/08/85 369     654.00      654.00 15/08/85
00003  11117 15/08/85 147     980.00      980.00 20/08/85
00004  11118 15/08/85 123    1234.00     1234.00 15/08/85
00005  12000 15/08/85 357    2111.00     1688.00 15/08/85
00006  12222 15/08/85 098    2155.00     2155.00 20/08/85
00007  12345 20/08/85 234    1500.00        0.00
00008  12346 20/08/85 345    1798.00        0.00
00009  12347 20/08/85 761    2195.00        0.00
00010  23611 20/08/85 369    1244.00        0.00
```

Bild 4.16: Der Inhalt der Rechnungsdatei zu Beginn der Transaktionen

```
. Use kunden INDEX kundnum
. LIST
00001  098 8057        15/08/85   5612.89   4490.31   1122.58
00002  123 8000        15/08/85   1234.00   1234.00      0.00
00003  147 2000 11117  15/08/85    980.00    980.00      0.00
00004  159 6000        15/08/85    852.00    681.60    170.40
00005  234 8000        15/08/85   3218.00   2574.40    643.60
00006  258 2000        15/08/85    355.45    355.45      0.00
00007  267 8201        15/08/85    698.88    559.10    139.78
00008  345 6100        15/08/85   2005.67   2005.67      0.00
00009  357 6100        15/08/85   2111.00   1688.80    422.20
00010  369 8000 23611  20/08/85   1898.00    654.00   1244.00
00011  421 8500        20/08/85    761.00    608.80    152.20
00012  741 8630        15/08/85   2587.00      0.00   2587.00
00013  761 1000        15/08/85   3005.55   2404.44    601.11
00014  852 6850        15/08/85   1200.00      0.00   1200.00
00015  963 1000        15/08/85   1591.50      0.00   1591.50
```

Bild 4.17: Die Kundendatei zu Beginn der Transaktionen

Starten Sie nun das Programm und wählen Sie die Funktion "Rechnungs-Bearbeitung" durch Druck auf die Taste "4". Nach kurzer Zeit, die für die Eröffnung der beiden Indexdateien benötigt wird, fragt das Programm nach einer Rechnungsnummer. Geben Sie nacheinander zwei neue Rechnungen ein:

1. Rechnung:	Rechnungs-Nr.:	30001
	Kunden-Nr.:	345
	Betrag:	895.00
2. Rechnung:	Rechnungs-Nr.	30002
	Kunden-Nr.:	147
	Betrag.	1750.00

Wie Sie sehen, erkennt das Programm an den Rechnungsnummern, daß es sich um neue Rechnungen handelt und bringt von selbst die richtige Erfassungsmaske. Wenn Sie jetzt die Rechnungs-Nr. 30001 erneut eingeben, ist diese bereits in der Rechnungsdatei vorhanden und das Programm bringt die Erfassungsmaske für Zahlungseingänge (Bild 4.18), in die Sie den Betrag DM 500.00 eingeben.

```
                    Zahlungseingang

                  Rechnungs-Nr. 30001

        Betrag   Teilzahlung      Offen
        895.00        0.00        895.00
neue Zahlung: DM    500.00
```

Bild 4.18: Zahlungseingang für Rechnungs-Nr. 30001

Wenn Sie jetzt die Frage, ob weitere Eingaben erfolgen sollen, mit "N" beantworten, dann kehrt das Programm zum Menü zurück. Von hier aus können beliebige andere Funktionen abgerufen werden. Verlassen Sie aber zunächst das Programm durch Drücken der Taste "0" und betrachten Sie sich die beiden Indexdateien, mit denen Sie eben gearbeitet haben (Bilder 4.19 und 4.20).

Die Rechnungsdatei wurde, wie Sie sehen, um die zwei Datensätze der laufenden Nummern 00011 und 00012 ergänzt, die für die neuen Rechnungen angelegt wurden. Für die Rechnungs-Nr. 30001 wurde bereits auch die erste Teilzahlung über DM 500.- mit dem Datum der Einzahlung verbucht.

In der Kundendatei finden Sie die eingegebenen Rechnungen unter den entsprechenden Kundennummern, für die bereits Datensätze existierten, verbucht. Für den Kunden Nr.147 wurde die Rechnung mit der Nummer 30002 aufgenommen. Die Nummer der vorhergehenden Rechnung (11117) wurde ebenso wie das Rechnungsdatum überschrieben und die neuen Rechnungsbeträge wurden zu den alten Werten addiert: Der akkumulierte Umsatz erhöhte sich von DM 980.- auf DM 2730.-, während gleichzeitig der gesamte neue Rechnungsbetrag über DM 1750.- in der Spalte **offen**, d.h. als Außenstände erscheint.

Analog wurde auch die Rechnung mit Nr.30001 im Datensatz des Kunden Nr.345 verbucht. Hier wurde zusätzlich die Teilzahlung über DM 500.- in das Feld **bezahlt** addiert.

```
. USE rechnung INDEX nummer
. LIST

00001  11115 15/08/85 963    1591.50     1200.00 20/08/85
00002  11116 15/08/85 369     654.00      654.00 15/08/85
00003  11117 15/08/85 147     980.00      980.00 20/08/85
00004  11118 15/08/85 123    1234.00     1234.00 15/08/85
00005  12000 15/08/85 357    2111.00     1688.00 15/08/85
00006  12222 15/08/85 098    2155.00     2155.00 20/08/85
00007  12345 20/08/85 234    1500.00        0.00
00008  12346 20/08/85 345    1798.00        0.00
00009  12347 20/08/85 761    2195.00        0.00
00010  23611 20/08/85 369    1244.00        0.00
00011  30001 22/08/85 345     895.00      500.00 22/08/85
00012  30002 22/08/85 147    1750.00        0.00
```

Bild 4.19: Die Rechnungsdatei nach den Transaktionen

```
. Use kunden INDEX kundnum
. LIST

00001  098 8057        15/08/85   5612.89   4490.31   1122.58
00002  123 8000        15/08/85   1234.00   1234.00      0.00
00003  147 2000 30002  22/08/85   2730.00    980.00   1750.00
00004  159 6000        15/08/85    852.00    681.60    170.40
00005  234 8000        15/08/85   3218.00   2574.40    643.60
00006  258 2000        15/08/85    355.45    355.45      0.00
00007  267 8201        15/08/85    698.88    559.10    139.78
00008  345 6100 30001  22/08/85   2900.67   2505.67    395.00
00009  357 6100        15/08/85   2111.00   1688.80    422.20
00010  369 8000 23611  20/08/85   1898.00    654.00   1244.00
00011  421 8500        15/08/85    761.00    608.80    152.20
00012  741 8630        15/08/85   2587.00      0.00   2587.00
00013  761 1000        15/08/85   3005.55   2404.44    601.11
00014  852 6850        15/08/85   1200.00      0.00   1200.00
00015  963 1000        15/08/85   1591.50      0.00   1591.50
```

Bild 4.20: Die Kundendatei nach Abschluß der Transaktionen

4.5 Zusammenfassung

In dem zurückliegenden Abschnitt haben Sie gelernt, wie man die dBASE-Kommandos zu Programmen und Unterprogrammen zusammenfassen und in Programmdateien abspeichern kann. Über die Befehle zur reinen Datenbankmanipulation hinaus bietet dBASE eine Reihe weiterer Befehle, die eine Programmierung kompletter Anwendungspakete auf einem hohen Sprachniveau erlauben. Auf konventionelle Programmiersprachen wie BASIC oder COBOL muß dabei nicht zurückgegriffen werden. Die vorgeführten Beispiele sollten Ihnen einen kleinen Eindruck davon vermitteln.

Selbstverständlich ist es nicht möglich, in relativ kleinen Beispielen Programmverzweigungen für alle möglichen, in der Praxis auftretenden Varianten zu berücksichtigen. Diese können ein Programm sehr rasch enorm ausweiten. Das Gleiche gilt für Programmroutinen zur Plausibilitätskontrolle von Dateneingaben oder zur Sicherung der Datenbestände. Alle diese Zusatz- und Hilfsprogramme müssen zumindest in der professionellen Praxis das eigentliche Hauptprogramm ergänzen. Aber auch hierfür ist dBASE eingerichtet.

Um die eventuell erforderlichen Zusatzprogramme nicht gänzlich unerwähnt zu lassen, werden im folgenden Kapitel einige einzelne und unabhängige Programmroutinen beschrieben, die bei entsprechender Anpassung ihrer Parameter leicht in andere Programme integriert werden können.

Aber bevor Sie diese Beispiele auf Ihrem Computer programmieren und austesten, sollten Sie wieder einmal Ihre Diskette überprüfen und alle bisherigen Dateien und Programme auf eine andere Diskette kopieren. Für die folgenden Programme brauchen Sie nur die Adressendatei und das Menüprogramm. Dessen Unterprogramme und die Kunden- bzw. Rechnungsdatei benötigen Sie nur, wenn Sie mit den Beispielen wie bisher "arbeiten" wollen.

5 Beispiele

In den vorangegangenen Abschnitten haben Sie einen kleinen Eindruck über die Möglichkeiten erhalten, mit dBASE komplette Anwendungen zu realisieren, ohne auf herkömmliche Programmiersprachen zurückgreifen zu müssen. Es ließen sich nun beliebig viele Beispiele, etwa aus den Bereichen Rechnungswesen, Fakturierung oder Personalwesen ausarbeiten, was aber nicht nur den Rahmen dieses Buches sprengen würde. Wegen der jeweiligen speziellen Anwendungsproblematik wären auch die einzelnen Beispiele immer nur für einen relativ kleinen Leserkreis von Interesse.

Statt dessen sollen einige Beispiele für Programmroutinen gezeigt werden, die als Bausteine in großen Programmen mehr oder weniger universell, d.h. unabhängig von fachspezifischen Komponenten einsetzbar sind:

- verschiedene Möglichkeiten, Programmschleifen zu bilden
- ein Etiketten-Druckprogramm
- ein Programm, das den Benutzer zwingt, regelmäßig Sicherheitskopien seiner Dateien anzulegen
- die Möglichkeit, unbefugte Benutzer von bestimmten Programmen oder Daten fernzuhalten
- eine schnelle Suchroutinefür komplexe Suchbegriffe
- Suchen in Dateien bei nicht eindeutigem Suchbegriff
- dialoggesteuerte Erzeugung neuer Dateistrukturen
- Ansteuern von zwei Druckern in einem Programm

5.1 Verschiedene Programmschleifen

Programmschleifen sind Teilprogramme, die während der Verarbeitung mehrfach zyklisch durchlaufen werden oder zumindest durchlaufen werden können. Die Anzahl der Zyklen wird stets von einer Bedingung abhängig gemacht, deren Erfüllung in jedem Zyklus entweder durch das System oder durch den Benutzer gesteuert wird. Sämtliche Programmschleifen werden in dBASE immer mit dem Befehl DO WHILE (Bedingung) eingeleitet, für dessen Formulierung man die Anweisung am besten so liest: "Durchlaufe die Schleife solange, wie die Bedingung erfüllt ist."

Manche anderen Programmiersprachen bieten auch die Möglichkeit, mit Sprunganweisungen wie z.B. GOTO (Adresse) die Wiederholung von Programmteilen zu veranlassen. In dBASE wurde bewußt auf diese Variante verzichtet, da derartige Sprunganweisungen leicht zu unstrukturierten Programmen führen, die bei hinreichender Größe unübersichtlich und fehleranfällig werden. Bei der Überprüfung von Programmen ist dann der chronologische Programmablauf oftmals nur schwer zu rekonstruieren.

Demgegenüber ist der Programmierer bei der Verwendung von dBASE gezwungen, seine Programme in einzelne Komponenten strukturiert zu entwickeln, wobei jedes Unterprogramm in sich abgeschlossen ist und damit übersichtlich bleibt. Der Befehl DO (Programm-Name) ist der einzige Weg, in andere Programmteile als Unterprogramme zu springen, aus denen man mit dem Befehl RETURN wieder zum Ausgangspunkt zurückkehrt. Zusammen mit dem Befehl DO WHILE (Bedingung) lassen sich so Programm-Hierarchien aufbauen, in denen die einzelnen Programme beliebig oft und in beliebiger Reihenfolge aufgerufen werden können und der Benutzer schließlich immer wieder zur obersten Programmebene (meist ein Menü) zurückgeführt wird.

Die folgenden Beispiele zeigen die wichtigsten Bedingungen für die Bildung von Programmschleifen, wobei wegen der Bedeutung dieser Routinen bewußt Wiederholungen von bereits erläuterten Varianten in Kauf genommen wurden.

5.1.1 DO WHILE .NOT. EOF

Diese Bedingung wird bei der sequentiellen Bearbeitung einer Datei benutzt, wobei in jedem Schleifendurchlauf ein Datensatz verarbeitet wird, bis schließlich das Dateiende erreicht ist. Die Prüfung des Dateiendes erfolgt mit der Funktion

EOF

(EOF = END OF FILE = Dateiende), die von ihrem logischen Wert "F" (= falsch) in den logischen Wert "T" (= TRUE = wahr) übergeht, sobald der Datensatzzeiger auf das Dateiende weist. Die Schleife

DO WHILE .NOT. EOF

wird also solange durchlaufen, wie der Wert von "EOF" nicht "wahr" ist, d.h. solange, wie das Dateiende noch nicht erreicht ist. Bild 5.1 zeigt ein kleines Programm, das alle Adressen einer Datei der Reihe nach auf den Bildschirm holt, wobei die jeweils nächste auf einen beliebigen Tastendruck hin ausgelöst wird. Der Aufruf des jeweils folgenden Datensatzes wird mit dem Befehl

SKIP

(= springen) ausgelöst. Durch Zufügen einer positiven oder negativen Zahl kann der Datensatzzeiger auf einen entsprechend weiter entfernten Datensatz gerichtet werden.

```
USE adressen
DO WHILE .NOT. EOF
     ERASE
     @ 5,10 SAY TRIM(vorname)+' '+name
     @ 7,10 SAY strasse
     @ 9,10 SAY ort
     WAIT
     SKIP
ENDDO
RETURN
```

Bild 5.1: Schleife zum Abarbeiten einer Datei

5.1.2 DO WHILE EOF

DO WHILE # = 0

Auch diese Schleifenbedingungen sind sehr nützlich. Beide sind gleichbedeutend, letztere kann aber nur in Indexdateien benutzt werden.

Sie bewirken, daß die Schleife solange nicht verlassen wird, wie sich der Datensatzzeiger am Ende der Datei befindet. Diese Variante kann dann eingesetzt werden, wenn man bei Suchläufen in einer Datei damit rechnen muß, daß der gesuchte Satz nicht existiert oder nicht gefunden wird und man dem Benutzer eine Chance geben will, den Suchlauf zu wiederholen. Dann kann man mit einem Befehl LOCATE oder FIND in der Programmschleife den Suchlauf bei Bedarf beliebig oft auslösen.

In Bild 5.2 ist für eine Indexdatei eine Schleife eingerichtet, die bei vergeblichem Suchen nach einem Kundennamen eine Systemmeldung bringt und dem Benutzer die Wahl läßt, entweder den Suchlauf zu wiederholen oder die Suche abzubrechen. Wird der gesuchte Name sofort gefunden, dann wird die gesamte Schleife übersprungen und die Adresse angezeigt.

Die Alternative, mit einem IF-Befehl das Dateiende abzufragen, läßt zwar auch einen zweiten Suchlauf zu, aber keine weiteren. Sie ist daher nur bedingt brauchbar.

Ganz besonders möchten wir Sie darauf hinweisen, daß die Betätigung der ENTER-Taste ein Leerzeichen erzeugt. Dieses kann wie jedes andere Zeichen gespeichert, abgefragt und verglichen werden. Wenn zum Beispiel wie hier statt der Eingabe eines Namens nur die ENTER-Taste betätigt wird, kann man dies zu einer Verzweigung zum Programmende nutzen:

```
IF Mname = ' '
   RETURN
ENDIF
```

```
ERASE
USE adressen INDEX alphabet
ACCEPT 'Name des Kunden' TO Mname
FIND &Mname
DO WHILE # = 0
     ERASE
     @ 5,5 SAY 'Name nicht gefunden'
     @ 7,5 SAY 'Bitte die Eingabe wiederholen'
     @ 8,5 SAY '      oder nur ENTER drücken'
     @ 10,0
     ACCEPT 'Name des Kunden' TO Mname
     IF Mname = ' '
             RETURN
     ENDIF
     FIND &Mname
ENDDO
ERASE
@ 5,10 SAY TRIM(vorname)+' '+name
@ 7,10 SAY strasse
@ 9,10 SAY ort
RETURN
```

Bild 5.2: Schleife zur Wiederholung eines Suchlaufs

5.1.3 DO WHILE T

In übergeordneten Programmen wie Menüs, von denen aus Unterprogramme abgerufen werden, wird häufig eine unbedingte Schleife benutzt, die für sich allein eine "Schleife ohne Ende" darstellt. Solch eine Schleife wird beispielsweise mit

DO WHILE T

gebildet, was bedeutet, daß die Schleife solange durchlaufen wird, wie die Bedingung "T" den logischen Wert "wahr" einnimmt. Da "T" (= TRUE) eine logische Konstante mit dem Wert "wahr" ist, ist die Bedingung unabänderlich immer erfüllt. Ein Ausgang aus der Schleife wird über separate Bedingungen erzielt, deren Eintreffen zu Verzweigungen in Unterprogramme oder zum Programmende führt (Bild 5.3).

Den gleichen Effekt wie DO WHILE T erzielt man mit allen ("wahren") Bedingungen, wenn deren Inhalt innerhalb des Programmes nicht geändert wird.

Beispiele:

```
STORE T TO menue
DO WHILE menue          der Wert von menue ist "wahr"

STORE 99 TO zahl
DO WHILE zahl = 99      solange die Zahl 99 nicht geändert wird

STORE 'X' TO zeichen
DO WHILE zeichen = 'X'  solange das X nicht überschrieben wird
```

Der entscheidende Nutzen derartiger Routinen liegt darin, daß ein Ausgang aus der Schleife nur über die Verzweigung in separate Unterprogramme (auch wenn diese noch so kurz sind) möglich ist, die letztlich immer wieder zum Hauptprogramm, dem Menü, zurückführen. Damit wird der Benutzer zu seinem eigenen Vorteil zu einer strukturierten Programmierung gezwungen.

```
DO WHILE T
     ERASE
     @ 1,30 SAY 'Programm-Auswahl'
     @ 5,15 SAY '(1) Informationen über einzelne Kunden'
     @ 7,15 SAY '(2) Kundenadressen auflisten'
     @ 9,15 SAY '(3) neue Kunden aufnehmen'
     @ 11,15 SAY '(4) Rechnungs-Bearbeitung'
     @ 13,15 SAY '(0) ENDE'
     @ 19,15 SAY 'Bitte wählen Sie eine Funktion'
     WAIT TO antwort
     DO CASE
             CASE antwort = '1'
                     DO info2
             CASE antwort = '2'
                     USE adressen INDEX alphabet
                     DISPLAY ALL
             CASE antwort = '3'
                     USE adressen INDEX alphabet
                     APPEND
             CASE antwort = '4'
                     DO rechnung
             CASE antwort = '0'
                     RETURN
     ENDCASE
ENDDO
```

Bild 5.3: Beispiel für eine unbedingte Schleife

Hinweis: Wenn Sie die Menü-Unterprogramme **info2** und **rechnung** nicht mehr auf Ihrer aktiven Diskette haben, dann müssen Sie entweder die Befehle "DO info2" und "DO rechnung" löschen oder an deren Zeilenanfang einen Stern (*) setzen. Dann werden sie vom Programm als Kommentare interpretiert und übergangen. Andernfalls "stürzt" Ihr Programm ab.

5.1.4 DO WHILE (Speicher) <> ' '

Im Gegensatz zu der eben erläuterten Programmschleife läßt diese genau eine Tastenfunktion zu, um im Anschluß an die Schleife das Programm fortzusetzen:

Die Betätigung jeglicher Buchstaben- oder Zifferntasten führt zu einem Verweilen im Menü oder zu einer Verzweigung in ein Unterprogramm. Lediglich bei Betätigung der ENTER-Taste, und nur bei dieser, ist die Schleifenbedingung nicht mehr erfüllt (Bild 5.4).

```
STORE 'x' TO antwort
DO WHILE antwort < > ' '
     ERASE
     @ 1,30 SAY 'Programm-Auswahl'
     @ 5,15 SAY '(1) Informationen über einzelne Kunden'
     @ 7,15 SAY '(2) Kundenadressen auflisten'
     @ 9,15 SAY '(3) neue Kunden aufnehmen'
     @ 11,15 SAY '(4) Rechnungs-Bearbeitung'
     @ 13,15 SAY 'ENTER-Taste = ENDE'
     @ 19,15 SAY 'Bitte wählen Sie eine Funktion'
     WAIT TO antwort
     DO CASE
             CASE antwort = '1'
                     DO info2
             CASE antwort = '2'
                     USE adressen INDEX alphabet
                     DISPLAY ALL
             CASE antwort = '3'
                     USE adressen INDEX alphabet
                     APPEND
             CASE antwort = '4'
                     DO rechnung
     ENDCASE
ENDDO
RETURN
```

Bild 5.4: Eine Schleife, die mit der ENTER-Taste verlassen wird

5.1.5 DO WHILE antwort$'(Ausdruck)'

Hier wird ein Teilprogramm solange durchlaufen, wie der Inhalt des Speichers **antwort** im "Ausdruck" enthalten ist. Durch Anwendung der Funktion "$" genügt es für die Erfüllung der Bedingung, daß die **antwort**-Zeichen als Teil des Ausdrucks vorkommen. Man kann nun als "Ausdruck" eine Folge von Tastensymbolen festlegen, die einzeln im Programm als Funktionswahltasten vorkommen. Wenn man wie üblich die **antwort** als ein einziges Zeichen eingibt, dann hat die Bedingung den Charakter einer mehrfachen logischen ODER-Verknüpfung:

DO WHILE antwort$'1234567890' entspricht:

DO WHILE antwort = ('1'.OR.'2'.OR:'3'.OR.....),

Bild 5.5 zeigt ein Beispiel, in dem alle Zifferntasten ein Verweilen im Programm bewirken und alle Buchstabentasten zum Programmende führen.

```
STORE '8' TO antwort
DO WHILE antwort$'1234567890'
     ERASE
     @ 5,15 SAY '(1) Informationen über einzelne Kunden'
     @ 7,15 SAY '(2) Kundenadressen auflisten'
     @ 9,15 SAY '(3) neue Kunden aufnehmen'
     @ 11,15 SAY '(4) Rechnungs-Bearbeitung'
     @ 13,15 SAY 'beliebige Buchstabentaste = ENDE'
     WAIT TO antwort
     DO CASE
             CASE antwort = '1'
                     DO prog1
             CASE antwort = '2'
                     DO prog2
             CASE antwort = '3'
                     DO prog3
             CASE antwort = '4'
                     DO prog4
     ENDCASE
ENDDO
RETURN
```

Bild 5.5: Programmschleife mit mehreren alternativen Bedingungen

5.1.6 Zählschleifen

In vielen Programmierkursen dienen "Zähler" als Einführungsbeispiele für Programmschleifen, weil sie im Aufbau sehr einfach sind. Bei der Programmierung kommerzieller Anwendungen werden sie nicht allzu häufig gebraucht, sollen aber hier doch kurz erwähnt werden.

Grundsätzlich wird immer ein Zählspeicher auf einen Anfangswert gesetzt, der dann mit jedem Schleifendurchlauf um einen bestimmten Betrag erhöht oder erniedrigt wird. Der aktuelle Wert des Zählspeichers wird in der Schleifenbedingung mit einem Grenzwert verglichen.

Im folgenden Beispiel wird ein Zähler Z zunächst auf den Wert 0 gesetzt und dann in jeder Schleife um 1 erhöht. Dies wird so lange wiederholt, wie der Zählerstand kleiner als der Grenzwert 127 ist, d.h., es finden genau 127 Schleifendurchläufe statt.

Bei jedem Durchlauf wird der Zählerstand Z (0 bis 126) angezeigt, und daneben das Zeichen, das sich auf Ihrem Bildschirm ergibt, wenn Z als sein ASCII-Wert interpretiert wird.

```
* Ausgabe aller ASCII-Codes
SET TALK OFF
STORE 0 TO Z
DO WHILE Z < 127
   ? STR(Z) + ' '+ CHR(Z)
   STORE Z + 1 TO Z
ENDDO
SET TALK ON
RETURN
```

Bild 5.6: Zählschleife

```
55    7
56    8
57    9
58    :
59    ;
60    <
61    =
62    >
63    ?
64    @
65    A
66    B
67    C
68    D
```

Ergebnis: ASCII-Codes (Ausschnitt)

5.2 Etiketten-Druckprogramm

Das folgende Programm ist in der Lage, für eine ganze Datei bestimmte Daten auf Etiketten, die über einen Endlosträger dem Drucker zugeführt werden, ausdrucken zu lassen. Dies kommt typischerweise bei der Erstellung von Aufklebern für Adressen in Serienbriefen oder von Aufklebern zur Warenauszeichnung vor.

Hier soll als Beispiel die bereits früher erzeugte Adressendatei benutzt werden, um Kundenadressen im Etikettenformat zu drucken. Wenn nicht die gesamte Adressendatei ausgewertet werden soll, dann wird man in der Praxis zunächst durch selektives Kopieren aus der Stammdatei die eigentliche Druckdatei generieren.

Es sei angenommen, daß die Etiketten fünfzeilig beschreibbar sind und einen Abstand von zwei Zeilen zum nächsten Etikett haben. Dementsprechend kann das Druckbild gewählt werden (Bild 5.7). Über der dreizeiligen Adresse ist eine weitere Zeile vorgesehen, in die ein für jeden Drucklauf individueller Text, z.B. ein Datum eingeblendet werden kann.

```
Muster-Etikett

Jakob Becker
Hauptstr.28
6100 Darmstadt


Muster-Etikett

Max Behr
Frankenweg 17
8500 Nürnberg
```

Bild 5.7: Muster für zwei Etiketten

Der Zusatztext wird mit ACCEPT in einen Speicher mit Namen **text** geladen, aus dem er dann später für den Druck abgerufen wird (Bild 5.8).

Die einzelnen Datensätze werden nacheinander in einer Schleife

DO WHILE .NOT. EOF

bearbeitet, die solange durchlaufen wird, bis das Dateiende erreicht ist. Hierzu dient die integrierte Funktion "EOF" (= END OF FILE = Dateiende), die vom logischen Wert "F" (= falsch) in den logischen Wert "T" (=wahr) übergeht, sobald der Datensatzzeiger auf das Dateiende weist. (Siehe auch Beispiel 5.1: Verschiedene Programmschleifen).

Nach Einschalten des Druckers mit

SET PRINT ON

wird das Druckbild für die Etiketten mit dem Befehl "?" gesteuert, wobei auch der Zwischenraum zwischen zwei Etiketten mit einem "?" pro Leerzeile anzugeben ist:

```
? text
?
? TRIM(vorname)+' '+name
? strasse
? ort
?
?
```

Die Funktion TRIM entfernt aus dem Datenfeld **vorname** jeweils die Leerpositionen am Ende des Vornamens, wenn dieser kürzer als die spezifizierte Feldlänge ist. Damit rückt das nächste Feld **name** unmittelbar an den Vornamen heran. Um den üblichen Abstand zwischen Vorname und Nachname zu erhalten, wurde ein Leerfeld mit +' '+ wieder eingefügt.

Das Druckformat könnte selbstverständlich auch mit dem Befehl "@" zusammen mit SAY festgelegt werden, würde aber unnötigerweise ein Problem aufwerfen: Bei diesem Befehl wird nämlich immer dann ein Seitenvorschub ausgelöst, wenn ein "@" mit einer Zeilennummer gelesen wird, die kleiner als die vorausgegangene Zeilennummer ist. Das heißt, mit jedem neuen Etikett würde ein Seitenvorschub erfolgen.

Bei vielen Druckern läßt sich zwar die Anzahl der Zeilen pro Seite so einstellen (im Beispiel wären es sieben), daß ein "Seitenvorschub" das Aufschalten zur ersten Zeile des nächsten Etiketts bewirkt, aber einfacher ist die Alternative mit dem Befehl "?", der einen Seitenvorschub erst gar nicht auslöst.

Wenn Etiketten zu drucken sind, ist es meist erforderlich, zu Beginn den Endlosträger im Drucker zu justieren. Das ist am einfachsten möglich, wenn man ein einzelnes Etikett ausdrucken und die Druckposition beurteilen bzw. korrigieren kann.

Das vorliegende Programm läßt dies zu, indem zunächst nur ein Etikett gedruckt wird und der Benutzer anschließend gefragt wird, ob ein weiteres zur Justage benötigt wird oder ob jetzt alle Etiketten gedruckt werden können.

Dies wird realisiert, indem zu Beginn des Programms der Speicher **Meinzel** mit dem Wert "E" (für EINZEL) geladen wird. Wenn nach dem Druck eines Etiketts dieser Speicher abgefragt wird und den Wert "E" enthält, dann werden dem Benutzer über ein Menü drei Alternativen zur Auswahl gestellt:

(E) Einzeldruck für Justage fortsetzen

(A) Alle Etiketten von Anfang an drucken

(0) ENDE

Der Benutzer kann somit das Druckbild justieren und ein weiteres Etikett einzeln abrufen. Der Vorgang wird solange wiederholt, bis die Justage korrekt ist. Dann kann man den Druck sämtlicher Etiketten von Anfang an starten. Wenn einmal die Taste "A" betätigt wurde, dann wird während des folgenden Etikettendrucks das Auswahlmenü nicht mehr aufgerufen.

Schließlich hat der Anwender die Möglichkeit, das Programm ganz abzubrechen.

```
ERASE
@ 5,25 SAY 'Etikettendruck-Programm'
@ 10,0
ACCEPT 'zusätzlicher Text' TO text
USE adressen INDEX alphabet
STORE 'E' TO Meinzel
DO WHILE .NOT. EOF
     SET PRINT ON
     ? text
     ?
     ? TRIM(vorname)+' '+name
     ? strasse
     ? ort
     ?
     ?
     SET PRINT OFF
     SKIP
     IF !(Meinzel) = 'E'
             ERASE
             @ 10,5 SAY '(E) Einzeldruck für Justage fortsetzen'
             @ 12,5 SAY '(A) Alle Etiketten von Anfang an drucken'
             @ 14,5 SAY '(0) ENDE'
             @ 17,5 SAY 'Bitte wählen Sie eine Funktion'
             WAIT TO Meinzel
             IF !(Meinzel) = 'A'
                     GO TOP
             ENDIF
             IF !(Meinzel) = '0'
                     RETURN
             ENDIF
     ENDIF
ENDDO
RETURN
```

Bild 5.8: Programm für Etiketten-Druck

5.3 Vom Programm erzwungene Sicherungskopien

Datenbestände aufzubauen ist bei größeren Datenmengen eine langwierige und teure Angelegenheit. Daher sollten alle Daten vor Verlust oder Zerstörung gut geschützt werden. Die wichtigste Maßnahme zum Schutz von Datenbeständen ist die regelmäßige Erstellung von Sicherungskopien, die bei Verlust der Originaldaten benutzt werden können.

An sich ist es sehr leicht, Kopien von Dateien oder ganzen Disketten herzustellen. Das eigentliche Problem besteht darin, die Sicherungskopien wirklich konsequent und regelmäßig anzulegen, damit die Informationen im Bedarfsfalle auch aktuell sind. Hier wird in der Praxis häufig gesündigt, insbesondere von Benutzern, die keine detaillierten Kenntnisse über die Datenverarbeitung besitzen und die Risiken bzw. Folgen eines Datenverlustes unterschätzen.

Im folgenden Beispiel wird vorgeführt, wie man den Benutzer, d.h. auch sich selbst zwingen kann, Sicherungskopien im Rahmen der Routineverarbeitung anzulegen. Da Sicherungskopien stets auf separaten Disketten angelegt werden, ist das Programm für Computer mit zwei Diskettenlaufwerken geschrieben.

Das Grundprinzip ist dabei folgendes: Die Sicherungskopie für eine Datei wird über ein eigenes Programm erstellt. Bei dieser Gelegenheit wird in einem separaten Speicher ein diesbezüglicher Hinweis (und zwar der Text "frei") gesetzt und auf Diskette gespeichert. Werden irgendwelche Veränderungen an der Datei vorgenommen, dann wird der Hinweis "frei" gelöscht.

Bevor nun eine Auswertung der Datei erfolgen kann, wird zunächst der Hinweisspeicher auf seinen Status hin abgefragt. Wird der Inhalt "frei" gelesen, kann das Auswertungsprogramm durchgeführt werden. Andernfalls wird dieses gesperrt und der Benutzer stattdessen aufgefordert, zuerst die Sicherungskopie zu erstellen.

Bild 5.9 zeigt das Programm **siko** zur Erstellung einer Sicherungskopie für die schon früher benutzte Datei **adressen**. Es enthält eine kurze Anweisung für den Benutzer, die auf dem Bildschirm eingeblendet wird und ihm mitteilt, welche Handgriffe zu tun sind.

```
DO WHILE T
ERASE
     @ 10,5 SAY 'Bitte Diskette für Sicherungskopie'
     @ 11,5 SAY 'in Laufwerk B einlegen und dann:'
     @ 14,5 SAY ' (S)  Start Sicherungskopie'
     @ 16,5 SAY ' (0)  zurück zum Hauptmenü'
     WAIT TO antwort
     IF !(antwort) = 'S'
             ERASE
             @ 10,5 SAY 'Sicherungskopie wird erstellt'
             USE adressen
             COPY TO B:adressen
             STORE 'frei' TO Msiko
             SAVE TO speicher
     ENDIF
     IF !(antwort) = '0'
             RETURN
     ENDIF
ENDDO
```

Bild 5.9: Programm zum Erstellen einer Sicherungskopie

Das Wesentliche an dem Programm ist die Eintragung des Wortes "frei" in den Speicher **Msiko** und dessen Speicherung auf Diskette in einer Speicher-Datei mit Namen **speicher**:

STORE 'frei' TO Msiko

SAVE TO speicher

Mit SAVE (= sichern) werden alle augenblicklich definierten Speicher mit ihren Inhalten in einer Datei vom Typ .MEM (= MEMORY) auf der Diskette gespeichert. So bleibt der Hinweis "frei" auch nach Abschalten des Computers erhalten und ist bei der nächsten Dateibearbeitung wieder verfügbar.

Das gesamte Programm läuft in einer DO-WHILE-Schleife, damit für mehrere Kopien oder bei Betätigung einer falschen Taste das Programm nicht erneut aufgerufen werden muß, sondern die Maske mit den Anweisungen wieder erscheint. Sie führt den Benutzer auch zurück in das Hauptprogramm.

Anmerkung: Das Programm läßt sich bezüglich der zu sichernden Dateien auch verallgemeinern, wenn in der Maske zuerst nach der Datei gefragt wird und diese dann indirekt aufgerufen wird, wobei die Funktion "&" zu benutzen ist, z.B. mit USE &datei und COPY TO B:&datei.

Zur Auswertung des Hinweises "frei" müssen im Hauptprogramm entsprechende Abfragen installiert werden, die gegebenenfalls eine Weiterbearbeitung der Datei verhindern.

In Fortführung der früheren Beispiele soll dies für das Programm **menue** erfolgen, in dem die Liste aller Kunden mit ihren Adressen nur dann gedruckt wird, wenn über die neueste Version der Datei eine Sicherungskopie existiert. Dazu sind vier Änderungen am Programm **menue** durchzuführen:

1. Das Programm **siko** ist in das Menü aufzunehmen (Bild 5.10).
2. Die Speicher-Datei **speicher** ist von der Diskette zu laden.
3. Der Inhalt des Hinweisspeichers **Msiko** muß eine Programmverzweigung steuern, um das Druckprogramm entweder freizugeben oder zu sperren.
4. Nach der Funktion "neue Kunden aufnehmen", also nach Änderung der Adressendatei muß der Hinweis "frei" gelöscht werden.

```
                Programm-Auswahl

        (1)  Informationen über einzelne Kunden
        (2)  Kundenadressen auflisten
        (3)  neue Kunden aufnehmen
        (4)  Rechnungs-Bearbeitung
        (5)  Sicherungskopie
        (0)  ENDE

        Bitte wählen Sie eine Funktion
```

Bild 5.10: Programm-Menü mit zusätzlicher Option zum Erstellen von Sicherungskopien

Das vollständige Programm **menue** ist in Bild 5.11 wiedergegeben und Ihnen zum größten Teil bekannt, so daß sich eine Erläuterung auf die wenigen Ergänzungen beschränken kann. Die Speicher-Datei wird mit dem Befehl

RESTORE FROM speicher

eingelesen, bevor das Programm beginnt. Mit RESTORE (= wiederherstellen) werden alle in der Speicherdatei vorhandenen Daten in die früher definierten Speicher zurückgeschrieben. Gleichzeitig werden die bis dahin aktiven Speicher gelöscht. Damit wird der Status der letzten Dateibearbeitung in den Speicher geholt.

Die Existenz einer gültigen Sicherungskopie der Adressendatei wird vor der Auflistung der Kundenadressen, also mit Beginn des Programmteils

CASE antwort = '2'

überprüft. Steht im Speicher **Msiko** das Wort "frei", dann können die Datei aufgerufen und die Liste gedruckt werden, andernfalls erscheint die Aufforderung, eine Sicherungskopie zu erstellen. Jeder anschließende Tastendruck führt zurück zum Menü, von dem aus das Kopierprogramm gestartet wird.

Schließlich ist noch im Programmteil

CASE antwort = '3'

das Wort "frei" im Speicher **Msiko** zu löschen, nachdem neue Datensätze aufgenommen wurden. Der neue Status wird sofort wieder in der Speicher-Datei **speicher** auf Diskette festgehalten.

```
RESTORE FROM speicher
DO WHILE T
     ERASE
     @ 1,30 SAY 'Programm-Auswahl'
     @ 5,15 SAY '(1)  Informationen über einzelne Kunden'
     @ 7,15 SAY '(2)  Kundenadressen auflisten'
     @ 9,15 SAY '(3)  neue Kunden aufnehmen'
     @ 11,15 SAY '(4)  Rechnungs-Bearbeitung'
     @ 13,15 SAY '(5)  Sicherungskopie'
     @ 15,15 SAY '(0)  ENDE'
     @ 19,15 SAY 'Bitte wählen Sie eine Funktion'
     WAIT TO antwort
     DO CASE
             CASE antwort = '1'
                     DO info2
             CASE antwort = '2'
                     ERASE
                     IF Msiko = 'frei'
                             USE adressen INDEX alphabet
                             @ 1,1 SAY 'Kunde Strasse  Wohnort  Kunden-Nr.'
                             DISPLAY OFF ALL $(vorname,1,1),name,strasse,;
                                                    ort,kunummer
                             WAIT
                     ELSE
                             @ 10,5 SAY 'Machen Sie zuerst eine ;
                                                          Sicherungskopie!'
                             @ 15,5 SAY '    Drücken Sie irgendeine Taste'
                             WAIT
                     ENDIF
             CASE antwort = '3'
                     USE adressen INDEX alphabet
                     APPEND
                     STORE ' ' TO Msiko
                     SAVE TO speicher
             CASE antwort = '4'
                     DO rechnung
             CASE antwort = '5'
                     DO siko
             CASE antwort = '0'
                     RETURN
     ENDCASE
ENDDO
```

Bild 5.11: Programm **menue** mit Zwang zu Sicherungskopien

5.4 Zugriffskontrolle mit Paßwort

Bei Datenbanksystemen, die von mehreren Anwendern gleichzeitig benutzt werden, ist es üblich, daß über den Benutzerkreis eine gewisse Kontrolle ausgeübt wird. Jeder Teilnehmer muß sich dem System gegenüber mit seinem Namen und/oder einem sogenannten Paßwort identifizieren, bevor er Zugang zur Datenbank erhält. Personen ohne zugelassenes Paßwort werden zurückgewiesen.

Darüber hinaus werden häufig bestimmte Funktionen für manche Teilnehmer gesperrt. Beispielsweise dürfen zu Personaldaten wie Gehältern etc. nur die authorisierten Mitarbeiter der Personalabteilung Zugriff haben. Auch die Änderung von Daten darf nur von berechtigten und qualifizierten Mitarbeitern durchgeführt werden, selbst wenn vielleicht ein größerer Personenkreis Zugang zu den Lesefunktionen hat und die Daten betrachten darf.

Da dBASE II als Datenbanksystem für Mikrocomputer und Einzelbenutzer entwickelt wurde, besitzt es keine integrierten Komponenten zur Steuerung einer Zugriffskontrolle. Wenn das System nur von einer Person benutzt wird, taucht das Problem der Zugriffsberechtigung erst gar nicht auf.

Dennoch ist der Fall denkbar, daß mehrere Personen eine gemeinsame dBASE-Datenbank benutzen wollen und eine Regelung der Zugriffsrechte notwendig wird. Wie dies realisiert werden kann, zeigt das folgende Beispiel.

Es muß vorausgeschickt werden, daß es vor unberechtigten Benutzern, die sowohl dBASE als auch das Betriebssystem im Detail kennen, keinen absoluten Schutz gibt, was übrigens analog auch für alle anderen Datenbanksysteme gilt. Ein Schutz bezieht sich immer nur auf die im Rahmen der normalen Verarbeitung verfügbaren Funktionen.

Für die Implementierung einer Zugriffskontrolle soll noch einmal das Programm **menue** dienen. Es ist deshalb als Beispiel sehr gut geeignet, weil von ihm aus verschiedene Unterprogramme abrufbar sind, für die unterschiedliche Berechtigungen vergeben werden können.

Zur Verwaltung der Daten über die berechtigten Personen wird zunächst eine neue Datei **passwort** mit folgender Struktur angelegt:

Feld	Name	Typ	Länge
001	name	c	015
002	passwort	c	010
003	schluessel	c	001

In Feld 001 wird der Name des Berechtigten und in Feld 002 sein von ihm gewähltes individuelles Paßwort eingetragen. Als "schluessel" kann eine Berechtigungsstufe festgelegt werden, zum Beispiel eine der drei Alternativen:

a uneingeschränkte Berechtigung

b Berechtigung, Informationen abzufragen, aber nicht zu ändern

c Berechtigung, nur bestimmte Informationen zu lesen

Das Prüfen der Zugriffsberechtigung erfolgt in zwei Stufen. In der ersten wird in einem separaten Programm (Bild 5.12), das dem Programm **menue** vorgeschaltet wird, Name und Paßwort des Teilnehmers erfragt und mit den gespeicherten Werten aus der Paßwortdatei verglichen. Ist der Name nicht gespeichert oder das Paßwort falsch, dann wird der Teilnehmer abgewiesen. Mit der nächsten Tastenbetätigung wird dBASE beendet.

Die Existenz des Namens in der Datei **passwort** wird einfach durch die Position des Datensatzzeigers nach Ausführung des Befehls LOCATE festgestellt: Existiert der Name nicht, dann wird bis zum Dateiende gesucht und die Bedingung

CASE EOF

(EOF = END OF FILE = Dateiende) tritt ein. Sind Name und Paßwort korrekt, dann wird das Schlüsselzeichen in dem Speicher **Mcode** abgelegt und das Hauptprogramm **menue3** aufgerufen.

```
SET TALK OFF
DO WHILE T
     ERASE
     USE passwort
     ACCEPT 'Ihr Name' TO Mname
     ACCEPT 'Ihr Passwort' TO Mpass
     LOCATE FOR name = Mname
     DO CASE
             CASE EOF
                     @ 15,5 SAY 'Sie haben keine Zugriffsberechtigung'
                     WAIT
                     QUIT
             CASE passwort <> Mpass
                     15,5 SAY 'Sie haben keine Zugriffsberechtigung'
                     WAIT
                     QUIT.
             CASE passwort = Mpass
                     STORE schluessel TO Mcode
                     DO menue3
     ENDCASE
ENDDO
```

Bild 5.12: Programm zum Überprüfen von Name und Paßwort

Die zweite Stufe der Zugriffskontrolle regelt die funktionsabhängige Berechtigung aufgrund des vergebenen Berechtigungsschlüssels. Hier muß vor Eröffnen der einzelnen zu schützenden Dateien oder vor dem Aufruf der entsprechenden Unterprogramme die Abfrage des Schlüssels mit geeigneter Sperrmöglichkeit vorgesehen werden.

Das bekannte Programm **menue** wird nun derart zum Programm **menue3** modifiziert, daß die Funktion (1), "Informationen über einzelne Kunden", nur mit den Schlüsseln "a" und "b" zugänglich ist und die Funktion (4), "Rechnungsbearbeitung", ausschließlich von Inhabern des Schlüssels "a" aufgerufen werden kann.

Das Programm **menue3** ist in Bild 5.13 dargestellt. Sie sehen, daß die Funktionen (1) und (4) des Menüs nur bei zutreffendem Code ausgeführt werden:

IF Mcode $'ab' (= wenn der Code "a" oder "b" ist) bzw.

IF Mcode = 'a'

Andernfalls erscheint die Nachricht "Sie haben hierfür keine Berechtigung" und danach wieder das Menü. Im Unterschied zur ersten Stufe der Zugriffskontrolle bleibt der Benutzer im Programm. Ihm sind lediglich bestimmte Funktionen verwehrt.

Auf diese Weise können beliebige Zugriffskontrollen in Programme implementiert werden, wobei die Kontrolle sogar auf der Ebene einzelner Datenfelder in den Unterprogrammen erfolgen kann.

```
DO WHILE T
     ERASE
     @ 1,30 SAY 'Programm-Auswahl'
     @ 5,15 SAY '(1) Informationen über einzelne Kunden'
     @ 7,15 SAY '(2) Kundenadressen auflisten'
     @ 9,15 SAY '(3) neue Kunden aufnehmen'
     @ 11,15 SAY '(4) Rechnungs-Bearbeitung'
     @ 13,15 SAY '(5) '
     @ 15,15 SAY '(0) ENDE'
     @ 19,15 SAY 'Bitte wählen Sie eine Funktion'
     WAIT TO antwort
     DO CASE
             CASE antwort = '1'
                     IF Mcode$'ab'
                        DO info2
                     ELSE
                        ERASE
                        @ 10,5 SAY 'Sie haben hierfür keine Berechtigung'
                        WAIT
                     ENDIF
```

Bild 5.13: Programm **menue3** mit Zugriffsberechtigung (Fortsetzung nächste Seite)

```
            CASE antwort = '2'
                    ERASE
                    USE adressen INDEX alphabet
                    @ 1,1 SAY 'Kunde Strasse  Wohnort  Kunden-Nr.'
                    DISPLAY OFF ALL $(vorname,1,1),name,strasse,;
                                              ort,kunummer
                    WAIT
            CASE antwort = '3'
                    USE adressen INDEX alphabet
                    APPEND
            CASE antwort = '4'
                    IF Mcode = 'a'
                       DO rechnung
                    ELSE
                       ERASE
                       @ 10,5 SAY 'Sie haben hiefür keine Berechtigung'
                       WAIT
                    ENDIF
            CASE antwort = '5'
                    *
            CASE antwort = '0'
                    QUIT
    ENDCASE
ENDDO
```

Bild 5.13: Programm **menue3** mit Zugriffsberechtigung

5.5 Schnelles zweistufiges Suchen

Wie mittlerweile hinreichend bekannt ist, verfügt dBASE über zwei verschiedene Arten von Suchbefehlen, deren Benutzung von der jeweiligen Situation abhängt:

- die Befehle LOCATE FOR (Bedingung)

  DISPLAY (ALL) FOR (Bedingung)

  LIST FOR (Bedingung),

  die mit geeigneten Bedingungen praktisch jedes beliebige Suchproblem innerhalb einer Datei lösen können. Einziger Nachteil ist die bei sehr großen Dateien relativ lange Suchzeit.

- der Befehl FIND (Ausdruck)

  für Indexdateien, der außerordentlich schnell ist, aber nur den Zugriff über den für diese Indexdatei definierten Schlüssel zuläßt.

Bei Schlüsselwerten, die in der Datei mehrfach vorkommen, wird mit dem Befehl FIND nur der erste zutreffende Datensatz gefunden. Wenn in einer Adressendatei der Name als Indexschlüssel festgelegt ist, dann liefert der Suchbefehl FIND 'Maier' nur die erste Person mit diesem Namen, nicht aber alle anderen "Maier". Es lassen sich zwar kompliziertere Indexschlüssel definieren (z.B. Name + Vorname), die aber das Suchen deutlich verlangsamen.

Eine effizientere Methode, schnell und trotzdem präzise auf den gewünschten Datensatz zuzugreifen, besteht in der Kombination der Befehle FIND und LOCATE.

Dabei wird gewissermaßen in einer ersten Stufe mit FIND das erste Auftreten des Schlüsselwertes festgestellt. Da die Indexdatei per Definition nach dem Schlüssel sortiert ist, folgen alle anderen Datensätze mit gleichem Schlüsselwert unmittelbar danach. Um in obigem Beispiel zu bleiben: alle anderen Personen mit Namen "Maier" folgen direkt nach der ersten.

Dann kann man in einer zweiten Stufe aus dem relativ kleinen Block aller "Maier" mit LOCATE gezielt die gesuchte Person ermitteln. Bild 5.14 zeigt eine solche Kombination von FIND und LOCATE für die Suche in einer Adressendatei. Gesucht wird hier nach dem Datensatz eines zu nennenden Kunden. Zur besseren Identifizierung des Kunden kann (wahlweise) sein Vorname mit angegeben werden. Das Suchen erfolgt wie beschrieben, wobei die zweite Stufe mit LOCATE nur zur Anwendung kommt, wenn tatsächlich ein Vorname angegeben wurde:

```
FIND &Mname
IF Mvor < > ' '
   LOCATE FOR (Mname$name .AND. Mvor$vorname)
ENDIF
```

Für die Suchbedingungen wurde die Funktion "$" gewählt, die ermöglicht, daß Teile, etwa die Anfangsbuchstaben von Name und Vorname, ausreichen, um den Datensatz zu finden.

Wenn die Suche ergebnislos verläuft, wird eine entsprechende Nachricht eingeblendet. Die Abfrage für diesen Fall muß beide Suchbefehle umfassen, wobei zwei verschiedene Meldungen für "nicht gefunden" vom System kommen können:

# = 0	wenn bereits mit FIND kein Satz für den angegebenen Namen gefunden wird
EOF	wenn mit LOCATE kein Satz mit der angegebenen Kombi-nation aus Name und Vorname gefunden wird.

Beide Fälle lassen sich in einem Befehl abfragen:

```
IF (EOF .OR. # = 0)
   ? 'Namen nicht gefunden'
ENDIF
```

```
USE adressen INDEX alphabet
DO WHILE T
     ERASE
     ACCEPT 'Name des Kunden' TO Mname
     ?
     ACCEPT 'Vorname' TO Mvor
     FIND &Mname
     IF Mvor <> ' '
             LOCATE FOR (Mname$name .AND. Mvor$vorname)
     ENDIF
     ERASE
     @ 5,0
     IF (EOF .OR. # = 0)
             ? 'Namen nicht gefunden'
             WAIT
     ELSE
             DISPLAY
             WAIT
     ENDIF
     STORE ' ' TO antwort
     DO WHILE !(antwort) <> 'J'
             @ 15,5 SAY 'Weitere Adressen suchen?'
             @ 17,5 SAY '(J)  Ja'
             @ 18,5 SAY '(N)  Nein'
             WAIT TO antwort
             IF !(antwort) = 'N'
                     RETURN
             ENDIF
     ENDDO
ENDDO
RETURN
```

Bild 5.14: Zweistufiges Suchen mit FIND und LOCATE

5.6 Schnelles Suchen und "Umblättern"

Auch im folgenden Beispiel geht es um einen möglichst schnellen Zugriff in großen Dateien, wenn der Schlüsselwert mehrfach vorkommt. Manchmal ist es unzweckmäßig oder dem Benutzer sogar unmöglich, den gesuchten Datensatz genauer als mit dem Wert des Indexschlüssels zu beschreiben. Dann kann man sich wieder die Sortierfolge der Indexdatei zunutze machen und mit einem FIND-Kommando den Block von Datensätzen selektieren, in dem der gewünschte Satz aufgrund seines Schlüsselwertes zu finden ist.

Anstatt nun mit detaillierteren Angaben und mit LOCATE weiterzusuchen, kann man auch mit Hilfe der Tastatur in der Datei vor- und zurück"blättern", bis man den gewünschten Satz gefunden hat. In der Praxis ist diese Methode oft nützlich, wenn verschiedene Datensätze als Alternativen zur Bearbeitung in Frage kommen. Beispiel: In einer Lieferantendatei sind für eine Firma mehrere Filialen mit ihren Adressen gespeichert. Dann kann man durch Angabe des Firmennamens die Gruppe aller Filialen schnell ansteuern und manuell eine Filiale aussuchen.

Für die schon mehrfach verwendete Adressendatei ist in Bild 5.15 ein Beispiel für diese Suchmethode gegeben. Unmittelbar nach Ausführung des Suchbefehls **FIND &Mname** wird zunächst überprüft, ob überhaupt ein Datensatz gefunden wurde. Ist dies nicht der Fall, dann hat der Benutzer die Möglichkeit, entweder seine Eingabe (eventuell korrigiert) zu wiederholen oder die Suche abzubrechen.

Ein gefundener Datensatz wird auf dem Bildschirm eingeblendet. Darunter erscheint ein kleines Menü für die anschließende, manuell gesteuerte Betrachtung der benachbarten Sätze.

Mit den Tasten "N" bzw. "Z" kann jeweils um einen Datensatz vor- oder zurückgegangen werden, während Taste "A" zum Abbruch der Suche führt. Mit Betätigen der ENTER-Taste wird aus dem Suchprogramm in ein in der Praxis nachfolgendes Verarbeitungsprogramm übergeleitet. Nur mit dieser Taste kann die übergeordnete Programmschleife

```
DO WHILE antwort <> ' '
```

verlassen werden (siehe auch Beispiel 5.1.4).

```
ERASE
USE adressen INDEX alphabet
ACCEPT 'Name des Kunden' TO Mname
FIND &Mname
STORE 'x'TO antwort
DO WHILE antwort < > ' '
     DO WHILE # = 0
             ERASE
             @ 5,5 SAY 'Name nicht gefunden'
             @ 7,5 SAY 'Bitte Eingabe wiederholen'
             @ 8,5 SAY 'oder mit ENTER abbrechen'
             @ 10,0
             ACCEPT 'Name des Kunden' TO Mname
             IF Mname =' '
                     RETURN
             ENDIF
             FIND &Mname
     ENDDO
     ERASE
     @ 3,10 SAY TRIM(vorname)+' '+name
     @ 5,10 SAY strasse
     @ 6,10 SAY ort
     @ 15,5 SAY 'Ist das der gesuchte Kunde?'
     @ 17,5 SAY 'Wenn Ja, ENTER drücken'
     @ 19,5 SAY '(N)  Nein, weiter zum nächsten Namen'
     @ 20,5 SAY '(Z)  Nein, zurück zum vorhergehenden'
     @ 21,5 SAY '(A)  Nein, Suche abbrechen'
     WAIT TO antwort
     DO CASE
             CASE !(antwort) = 'N'
                     SKIP
             CASE !(antwort) = 'Z'
                     SKIP -1
             CASE !(antwort) = 'A'
                     RETURN
     ENDCASE
ENDDO
ERASE
@ 10,5 SAY 'Die Daten des Kunden '+name+' sind verfügbar'
RETURN
```

Bild 5.15: Programm zum schnellen Suchen mit "Umblättern"

5.7 Programmgesteuertes Erzeugen von Dateien

dBASE verfügt über eine interessante Fähigkeit, die im Original-Handbuch kaum erwähnt wird, die dem (überwiegend professionellen) Programmierer aber ganz erstaunliche Möglichkeiten bietet. Es ist dies die Fähigkeit, mit einem speziellen Kopierbefehl die Strukturbeschreibung einer Datei als Dateiinhalt in eine andere Datei, die nachfolgend als Strukturdatei bezeichnet werden soll, zu übernehmen:

COPY TO (Name der Strukturdatei) STRUCTURE EXTENDED

Die einzelnen Datensätze der Strukturdatei bestehen jeweils aus der Beschreibung eines Feldes der Ursprungsdatei. Wenn zum Beispiel der Kopierbefehl auf die Struktur der Datei **adressen** angewandt wird, dann umfaßt die Strukturdatei fünf Datensätze mit den Feldbeschreibungen für die Kundennummer, den Namen, Vornamen, die Straße und den Wohnort des Kunden (Bild 5.16).

```
. USE adressen
. COPY TO struktur STRUCTURE EXTENDED
. USE struktur
. LIST

00001 KUNUMMER  C   3  0
00002 NAME      C  10  0
00003 VORNAME   C  10  0
00004 STRASSE   C  15  0
00005 ORT       C  16  0
```

Bild 5.16: Die Datensätze einer Strukturdatei

Die Strukturdatei **struktur** kann nun wie jede andere Datei bearbeitet und manipuliert werden. Sie besitzt auch selbst wieder eine Struktur, die sichtbar gemacht werden kann und die praktisch die Grunddefinitionen aller dBASE-Dateistrukturen enthält (Bild 5.17). Die Struktur einer Strukturdatei enthält die vom System vorgeschriebenen Regeln für die Erstellung einer Datei, wie die Felder definiert werden müssen, wie lang der Feldname sein darf (10 Zeichen) usw.

```
. USE struktur
. LIST STRUCTURE

Strukturdaten für Datei: A:STRUKTUR.DBF
Anzahl der Sätze:  00005
Datum der letzten Aktualisierung: 00/00/00
Primäre Datei
Feld    Name      Typ   Länge   Dez.st.
001   FIELD:NAME   C     010
002   FIELD:TYPE   C     001
003   FIELD:LEN    N     003
004   FIELD:DEC    N     003
** Gesamt **          00018
```

Bild 5.17: Die Struktur aller Strukturdateien

Für die Praxis ist es wichtig, daß eine Strukturdatei wie eine normale Datei bearbeitet und auch in eine "normale" Datei wieder zurückverwandelt werden kann. Das bedeutet, daß der gesamte Befehlsvorrat von dBASE auch zur Manipulation von Dateistrukturen und nicht nur von Dateiinhalten verwendet werden kann.

Es ist beispielsweise möglich, für individuelle ad-hoc-Abfragen der Datenbank Programmroutinen aufzurufen, die eine menügesteuerte Erstellung geeigneter Dateien ermöglichen. Diese können dann aus bestehenden Dateien mit Daten versorgt und ausgewertet werden. Es ist sogar möglich, Programme zu entwerfen, die eine gegebene Datei automatisch an den tatsächlichen Dateiinhalt anpassen und nicht benötigte Felder entfernen oder andere Felder auf die Länge des längsten Feldinhalts verkürzen.

Als Beispiel soll ein kleines Programm gezeigt werden, mit dem man im Dialog neue Dateien erstellen kann. Die Benutzerführung kann man dabei so gestalten, daß keine speziellen Kenntnisse über dBASE für die Dateidefinition mehr benötigt werden.

Zunächst generiert man eine leere Strukturdatei, die anschließend im Dialogteil des Programms mit den Informationen über die Struktur der zu erstellenden Datei geladen wird. Geeignet ist hierfür jede Strukturdatei, so auch die bereits existierende **struktur**, deren Datensätze mit den Befehlen

```
USE struktur
DELETE ALL
PACK
```

gelöscht werden. Diese Prozedur braucht nur ein einziges Mal zu erfolgen, wenn man im Generierprogramm nicht diese Datei selbst, sondern eine Kopie davon benutzt und lädt. Die Ausgangsdatei **struktur** kann dann immer wieder erneut verwendet werden.

Das Generierprogramm (Bild 5.18) beginnt daher mit dem Kopieren der Strukturdatei, nachdem der neu zu bildenden Datei ein Name gegeben wurde:

```
ACCEPT 'Name der zu erstellenden Datei ' TO Mdatei
USE struktur
COPY TO struktl
USE struktl
```

In der darauffolgenden DO-WHILE-Schleife werden auf die bekannte Weise leere Datensätze an die Datei gehängt und mit den Informationen über jeweils ein Datenfeld geladen. Zur Adressierung der Strukturdatei-Felder sind dabei in den GET-Anweisungen die aus Bild 5.17 ersichtlichen Namen zu verwenden. Nach der Anzahl der Dezimalstellen wird nur gefragt, wenn das Feld zuvor als numerisch definiert wurde.

Natürlich ließen sich an dieser Stelle mehr Funktionen zur Erhöhung des Bedienerkomforts oder zum Vermeiden von Eingabefehlern einbauen. Im Interesse der Übersichtlichkeit wurde aber darauf verzichtet.

Wenn alle Felder festgelegt sind, wird die Programmschleife verlassen und die neue Datei aus der eben definierten Struktur generiert. Hierfür existiert eine Variante des Befehls CREATE:

CREATE (Dateiname) FROM (Name der Strukturdatei)

Da der Name der neuen Datei zu Programmbeginn in den Speicher Mdatei geschrieben wurde, lautet der Befehl zum Generieren:

```
CREATE &Mdatei FROM struktl
```

Auf Wunsch kann nun die neue Datei sofort mit Daten geladen und bearbeitet werden.

```
ERASE
ACCEPT 'Name der zu erstellenden Datei ' TO Mdatei
USE struktur
COPY TO strukt1
USE strukt1
STORE 'x' TO antwort
DO WHILE !(antwort) < > 'N'
     ERASE
     APPEND BLANK
     @ 2,5 SAY 'Festlegung des '+STR(#,5)+'. Datenfeldes'
     @ 5,5 SAY 'Feldname (max. 10 Zeichen)' GET FIELD:NAME
     @ 7,5 SAY 'Feldtyp (C oder N oder L)' GET FIELD:TYPE
     @ 9,5 SAY 'Feldlänge (Anzahl Zeichen)' GET FIELD:LEN
     READ
     IF !( FIELD:TYPE) = 'N'
             @ 11,5 SAY 'Anzahl der Dezimalstellen' GET FIELD:DEC
             READ
     ENDIF
     @ 15,5 SAY 'weitere Felder definieren?'
     @ 17,5 SAY '(J)  Ja'
     @ 18,5 SAY '(N)  Nein'
     WAIT TO antwort
ENDDO
ERASE
@ 10,5 SAY 'Bitte warten, Datei wird generiert'
CREATE &Mdatei FROM strukt1
USE &Mdatei
@ 15,5 SAY 'Wollen Sie sofort Daten eingeben? (J/N)'
WAIT TO antwort
IF !(antwort) = 'J'
     APPEND
ENDIF
RETURN
```

Bild 5.18: Programm zur dialoggesteuerten Dateierstellung

5.8 Druckersteuerung mit dBASE

Die meisten Drucker, insbesondere alle Matrixdrucker, lassen sich auf verschiedene Schriftarten (Schmalschrift, Fettschrift usw.) umschalten. Der Anwender kann diese entweder am Gerät einstellen oder über seine Software (z.B. Textsystem) aufrufen. Allerdings ist kaum ein Programm in der Lage, sämtliche Druckvarianten eines Matrixdruckers oder gar Laserdrukkers gleichzeitig zu unterstützen. Mit dBASE II jedoch können Sie alle Möglichkeiten Ihres Druckers ausschöpfen.

Die Steuerzeichen für den Drucker bestehen meistens aus sogenannten Escape-Sequenzen, einer Zeichenfolge, die mit dem Code der Taste ESC (=ESCAPE) beginnt. Die Escape-Sequenzen sind von Drucker zu Drucker verschieden und müssen im Einzelfall im Druckerhandbuch nachgelesen werden. Zum Beispiel schaltet die Sequenz **ESC E** den Drucker Epson FX-80 auf Fettdruck.

Damit nun der Drucker eine solche Escape-Sequenz nicht als Text ausdruckt, sondern als Befehl interpretiert, muß diese als Druckbefehl codiert werden. Dazu dient die Funktion

CHR(numerischer Wert)

die in sehr ähnlicher Form den BASIC-Programmierern unter Ihnen bekannt ist. CHR setzt den "numerischen Wert" in dessen ASCII-Code um, der dann vom Drucker als Befehl akzeptiert wird. So ergibt etwa die Funktion CHR(27) den ASCII-Code ESC, also den Code der Escape-Taste.

Die Zuordnung der numerischen Werte zum ASCII-Code findet sich in den ASCII-Tabellen, die den meisten Computer- und Druckerhandbüchern beiliegen (im Bedienungshandbuch zum CPC 464 im Anhang III/Seite 1). Die für dBASE wesentlichen Dezimalwerte mit ihren ASCII-Codes können Sie auch der folgenden Tabelle entnehmen.

Tabelle der ASCII-Codes und ihrer Dezimalwerte

Dez.	ASCII	Dez.	ASCII	Dez.	ASCII	Dez.	ASCII
0	NUL	36	$	72	H	108	l
1	SOH	37	%	73	I	109	m
2	STX	38	&	74	J	110	n
3	ETX	39	'	75	K	111	o
4	EOT	40	(	76	L	112	p
5	ENQ	41	)	77	M	113	q
6	ACK	42	*	78	N	114	r
7	BEL	43	+	79	O	115	s
8	BS	44	,	80	P	116	t
9	HT	45	-	81	Q	117	u
10	LF	46	.	82	R	118	v
11	VT	47	/	83	S	119	w
12	FF	48	0	84	T	120	x
13	CR	49	1	85	U	121	y
14	SO	50	2	86	V	122	z
15	SI	51	3	87	W	123	{ ä
16	DLE	52	4	88	X	124	| ö
17	DC1	53	5	89	Y	125	} ü
18	DC2	54	6	90	Z	126	~ ß
19	DC3	55	7	91	[Ä		
20	DC4	56	8	92	\ Ö		
21	NAK	57	9	93	] Ü		
22	SYN	58	:	94	^		
23	ETB	59	;	95	_		
24	CAN	60	<	96	'		
25	EM	61	=	97	a		
26	SUB	62	>	98	b		
27	ESC	63	?	99	c		
28	FS	64	@ §	100	d		
29	GS	65	A	101	e		
30	RS	66	B	102	f		
31	US	67	C	103	g		
32	SP	68	D	104	h		
33	!	69	E	105	i		
34	"	70	F	106	j		
35	#	71	G	107	k		

Die oben genannte Sequenz ESC E kann dann unter dBASE als Befehl für Fettdruck beim FX-80 so formuliert werden:

```
SET PRINT ON
? CHR(27)+CHR(69)
```

Die einzelnen Elemente der Escape-Sequenz werden innerhalb eines Fragezeichen-Befehls mittels Plus-Zeichen zu einem String verbunden. Damit dieser als Drucksteuerbefehl wirksam sein kann, muß der Drucker zuvor mit SET PRINT ON aktiviert werden. Die eingestellte Druckart bleibt solange erhalten, bis sie durch eine andere ersetzt oder bis der Drucker ausgeschaltet wird. Der Befehl SET PRINT OFF ändert die Einstellung nicht. Sie können, z.B. innerhalb eines Programms, den Drucker beliebig oft aktivieren und de-aktivieren, ohne die einmal gewählte Schriftart zu verlieren.

Wenn Sie häufiger die Schriftart wechseln wollen, werden Sie das wiederholte Eintippen der Escape-Sequenzen mühsam finden. Dann ist es zweckmäßig, sich die wichtigsten Sequenzen als Variablen in einer Memory-Datei bereitzuhalten. Bei Bedarf wird die Datei eingelesen und der Variablenname der gewünschten Schriftart als Druckbefehl eingegeben. Die Namen sollten dabei so kurz wie möglich sein, damit sie sich im eigentlichen Text nicht allzu störend bemerkbar machen. Bild 5.19 zeigt einige Vorschläge für solche Namen, denen als Beispiel die Escape-Sequenzen für den Epson-Drucker FX-80 zugeordnet wurden (Bild 5.20 a).

N	Normalschrift (10 Pitch)
E	Enge Schrift (17 Pitch)
EL	Eliteschrift (12 Pitch)
F	Fettdruck
W1	Weite Schrift Typ 1
W2	Weite Schrift Typ 2
WF	Weite und fette Schrift
H	Hochgestellte Schrift (für Hochzahlen)
T	Tiefgestellte Schrift (für Indexzahlen)
M	Mitte (Rückstellen von Hoch- oder Tiefschrift)

Bild 5.19: Variablennamen für verschiedene Druckarten

```
DISPLAY MEMORY

F       (C)     CHR(27)+CHR(33)+CHR(8)
E       (C)     CHR(27)+CHR(33)+CHR(4)
EL      (C)     CHR(27)+CHR(33)+CHR(5)
N       (C)     CHR(27)+CHR(33)+CHR(2)
W1      (C)     CHR(27)+CHR(33)+CHR(33)
W2      (C)     CHR(27)+CHR(33)+CHR(32)
WF      (C)     CHR(27)+CHR(33)+CHR(40)
H       (C)     CHR(27)+CHR(83)+CHR(48)
T       (C)     CHR(27)+CHR(83)+CHR(1)
M       (C)     CHR(27)+CHR(84)
```

Bild 5.20 a: Escape-Sequenzen für den FX-80

Anmerkung: Für manche Druckarten (z.B. für Fettdruck) gibt es mehrere alternative Escape-Sequenzen, die sich vor allem darin unterscheiden, wie man die Druckart wieder ausschaltet. Wir haben aus den verschiedenen Möglichkeiten solche ausgewählt, die mit **einem** gemeinsamen Befehl (N = Normal) zum Rückstellen funktionieren. Lediglich Hoch- und Tiefstellen muß separat mit der Sequenz für M rückgängig gemacht werden.

Beim Schneider-Drucker NLQ 401 wird jede Auszeichnung durch eine eigene Escapesequenz wieder gelöscht. Hier könnte man zum Beispiel solche Namen vergeben:

E1	Enge Schrift EIN
E0	Enge Schrift AUS
F1	Fettdruck EIN
F0	Fettdruck AUS
W1	Weite Schrift EIN
W0	Weite Schrift AUS

Bild 5.20 b zeigt entsprechende Escapesequenzen für den NLQ 401.

```
DISPLAY MEMORY

E1        (C)       CHR(15)
E0        (C)       CHR(18)
F1        (C)       CHR(27)+CHR(69)
F0        (C)       CHR(27)+CHR(70)
W1        (C)       CHR(14)
W0        (C)       CHR(20)
```

Bild 5.20 b: Steuer-Sequenzen für den Schneider-Drucker NLQ 401

Bild 5.21 zeigt ein kleines Testprogramm, das alle definierten Schriften aktiviert und jeweils eine Zeile davon druckt (das Ergebnis sehen Sie in Bild 5.22). Die Variablenspeicher werden mit der Makrofunktion & aufgerufen, weil ihr Inhalt nicht ausgegeben, sondern interpretiert werden soll.

```
* Testprogramm dTEST.CMD zum Testen der Druckersteuerung
SET PRINT ON
? 'Normaldruck 10 Pitch'
?
? &F+'Fettdruck bei 10 Pitch'+&N
?
? &E+'Engschrift 17 Pitch'+&N
?
? &EL+'Eliteschrift 12 Pitch'+&N
?
? &W1+'Weitschrift 1'+&N
?
? &W2+'Weitschrift 2'+&N
?
? &WF+'Weit und Fett'+&N
?
? 'Formel mit '+&H+'Hochzahl: '+&M+'X'+&H+'2'+&M
?
? 'Indexzeichen: X'+&T+'min'+&M
SET PRINT OFF
RETURN
```

Bild 5.21: Testprogramm für Druckersteuerung

Wie Sie sehen, können die Drucksteuerbefehle ohne weiteres in einer Zeile mit Texten stehen. Sie müssen nur, wie alle anderen Strings, durch Plus-Zeichen verkettet werden.

Ferner fällt Ihnen sicherlich auf, daß jede Zeile mit einem Rückstellbefehl zur Normalschrift abschließt. Das ist kein Zwang, aber guter Programmierstil. Bevor eine neue Schriftart gewählt wird, sollte man immer zuerst auf "Normal" schalten. Denn manche Druckarten schließen sich gegenseitig aus. Bei manchen Druckern ist z.B. die Fettschrift nicht einschaltbar, solange *Kursivschrift* aktiviert ist. Solche druckerspezifischen Feinheiten werden beim Programmieren leicht übersehen, und dann verbringt man später wertvolle Zeit mit Fehlersuchen.

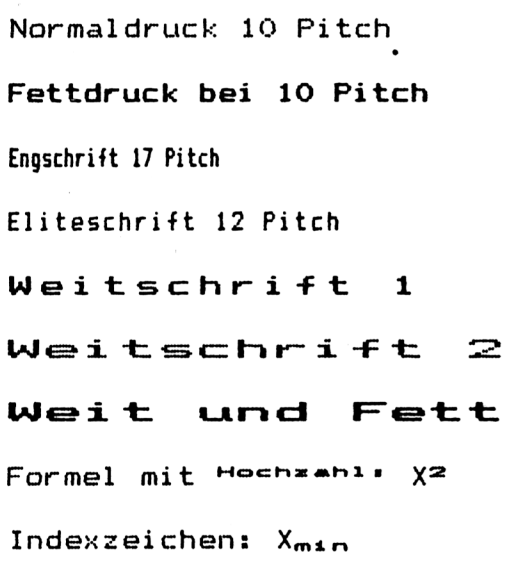

Bild 5.22: Verschiedene Druckmuster

5.9 Kommunikation mit anderen Programmen

Wenn zwei Programme untereinander Daten austauschen sollen, dann müssen sie eine gemeinsame Sprache sprechen. Der Empfänger muß in der Lage sein, die Informationen des Senders zu verstehen, d.h. richtig zu interpretieren. Dazu gehört nicht nur das Erkennen der einzelnen Zeichen, sondern auch das Erkennen z.B. der Wörter (Datenfelder) oder der Sätze, also das korrekte Interpretieren des Datenformats.

Nicht zuletzt für die Datenkommunikation wurden sogenannte Standards geschaffen, von denen Sie schon mindestens einen kennen: Den ASCII-Standard für die Zeichendarstellung (siehe ASCII-Tabelle in Beispiel 5.8). Ein anderer ASCII-Standard, der sehr weit verbreitet ist, befaßt sich mit dem Format von Datendateien. Danach besteht eine ASCII-Datei aus Zeilen, denen ein Datensatz entspricht, und innerhalb einer Zeile aus Datenfeldern mit jeweils fester Länge.

Dies ist genau das Format, das den Tabellen der relationalen Datenbanken wie dBASE entspricht. Daher kann dBASE ohne Probleme praktisch jede ASCII-Datei lesen und kann auch seine Datenbank-Dateien als ASCII-Dateien ausgeben. Letzteres erfolgt durch den Kopierbefehl

COPY TO (Dateiname) SDF

```
00001  123 Maier      Herbert    Amalienstr.59    8000 München        1000.00
00002  234 Huber      Josef      Sieglindenstr.5  8000 München         643.60
00003  345 Kuhn       Peter      Fliederstr.9     6100 Darmstadt       560.00
00004  098 Meister    Werner     Marktplatz 3     8057 Eching         1122.58


COPY TO test SDF

123Maier     Herbert   Amalienstr.59  8000 München        1000.00
234Huber     Josef     Sieglindenstr.58000 München         643.60
345Kuhn      Peter     Fliederstr.9   6100 Darmstadt       560.00
098Meister   Werner    Marktplatz 3   8057 Eching         1122.58
```

Bild 5.23: Datenbank- und SDF-Datei

Der Zusatz SDF (= Standard-Daten-Format) sorgt dafür, daß die Zieldatei im ASCII-Format mit dem Dateityp **TXT** angelegt wird. Bild 5.23 zeigt einige Sätze einer dBASE-Datei und darunter die zugehörige SDF-Datei.

Sie sehen, daß beide Dateien im Aufbau äußerlich gleich sind. Lediglich die dBASE-spezifischen Elemente sind in der SDF-Datei nicht mehr enthalten: Die laufende Satznummer und das eine Leerzeichen zur optischen Trennung der Datenfelder fehlen. Auch eine dBASE-Datensatzstruktur existiert nicht mehr. Die SDF-Datei ist eine reine Textdatei.

Manche Systeme arbeiten mit modifizierten Standard-Dateien, bei denen die einzelnen Felder durch Kommas voneinander getrennt werden. Manchmal wird auch jedes alphanumerische Feld zusätzlich von Sonderzeichen umklammert. Auch solche Dateien kann dBASE II erzeugen, wenn der Befehl

COPY TO (Dateiname) DELIMITED [WITH (Sonderzeichen)]

benutzt wird. Mit DELIMITED WITH wird angegeben, in welche Sonderzeichen die alphanumerischen Datenfelder einzubetten sind. Zum Beispiel ergibt der Befehl

COPY TO test1 DELIMITED WITH "

eine Textdatei wie in Bild 5.24 (oben), in der alle Felder durch Kommas getrennt und die alphanumerischen Felder in Anführungszeichen gesetzt sind. Sie sind auch auf die Länge ihres tatsächlichen Inhalts reduziert, während die numerischen Felder ihre volle Länge beibehalten.

Zwei Varianten dieses Befehls verdienen besondere Beachtung: Wenn nur DELIMITED angegeben wird, also ohne WITH, dann wählt dBASE II den Apostroph als (Standard-)Begrenzer der alphanumerischen Felder (Bild 5.24, Mittelteil). Und der Befehl

COPY TO test3 DELIMITED WITH ,

bewirkt, daß das Komma die alleinige Feldbegrenzung darstellt, was zu einer Datei wie im unteren Teil von Bild 5.24 führt. Dieses Dateiformat wird z.B. in WordStar und anderen Textverarbeitungssytemen benutzt, um Adressen für Serienbriefe zu verarbeiten.

```
. COPY TO test1 DELIMITED WITH "

"123","Maier","Herbert","Amalienstr.59","8000 München",    1000.00
"234","Huber","Josef","Sieglindenstr.5","8000 München",     643.60
"345","Kuhn","Peter","Fliederstr.9","6100 Darmstadt",      560.00
"098","Meister","Werner","Marktplatz 3","8057 Eching",     1122,58


. COPY TO test2 DELIMITED

'123','Maier','Herbert','Amalienstr.59','8000 München',    1000.00
'234','Huber','Josef','Sieglindenstr.5','8000 München',     643.60
'345','Kuhn','Peter','Fliederstr.9','6100 Darmstadt',      560.00
'098','Meister','Werner','Marktplatz 3','8057 Eching',     1122,58


. COPY TO test3 DELIMITED WITH ,

123,Maier,Herbert,Amalienstr.59,8000 München,1000.00
234,Huber,Josef,Sieglindenstr.5,8000 München,643.60
345,Kuhn,Peter,Fliederstr.9,6100 Darmstadt,560.00
098,Meister,Werner,Marktplatz 3,8057 Eching,1122,58
```

Bild 5.24: Verschiedene Varianten des Befehls COPY...DELIMITED

Sie können also auf Ihrem Schneider-Computer mit einer dBASE-Datenbank Ihre Adressen verwalten und dann gezielt einzelne oder alle in WordStar/Mailmerge-Briefe übernehmen. Im WordStar-Brief müssen dabei zwei Mailmerge-Punktbefehle eingetragen werden:

.DF (dBASE-Textdatei)

zur Bezeichnung der Adressen-Datei und

.RV (Felderliste)

zur Angabe der Datenfelder (der "Variablen"), die in dieser Datei vorkommen. Überall im Text des Serienbriefes, wo Variable (z.B. Name, Ort) aus der dBASE-Datei einzusetzen sind, wird deren Name, eingeklammert in &-Zeichen, geschrieben. Ein WordStar/Mailmerge-Brief würde etwa wie in Bild 5.25 aussehen. Diese Kombination zweier professioneller Softwarepakete (dBASE und WordStar) bietet Ihnen mehr Leistung als die zur Zeit modernen "Integrierten Systeme" auf größeren Personal Computern.

```
.DF test3.TXT
.RV name, vorname, strasse, ort, betrag

Herrn &vorname& &name&                                    23. August 1985
&strasse&
&ort&


Sehr geehrter Herr &name&,

wie wir leider feststellen mußten, ist unsere letzte Rechnung über

                                    DM &betrag&

offenbar noch nicht beglichen.....
```

Bild 5.25: WordStar-Serienbrief mit Datenübernahme aus dBASE

Die genannten Dateitypen kann dBASE II nicht nur erzeugen, sondern ihren Inhalt auch auch wieder in Datenbank-Dateien zurückholen. Voraussetzung dafür ist, daß eine geeignete dBASE-Dateistruktur existiert, deren Felder nach Anzahl und Länge auf die Felder der Text-Dateien abgestimmt sind. Die Feldnamen spielen dabei keine Rolle, da die Felder der ASCII-Dateien ja keine Namen besitzen. In diese Datei können dann die Textdateien eingelesen werden mit

APPEND FROM (Name der übertragenen Datei) SDF

(für SDF-Dateien) bzw. mit

APPEND FROM (Name der übertragenen Datei) DELIMITED

(für Dateien mit Feldbegrenzern). Die Kommas und Feldbegrenzungen werden dabei von dBASE selbständig entfernt, sie dürfen also bei der Strukturdefinition der Datenbank-Datei nicht mitgezählt werden. Der Befehl APPEND hängt wie üblich die eingelesenen Sätze an das Ende der Datei. Um die obigen vier Dateien (Bilder 5.23 und 5.24) in eine dBASE-Datei einzulesen, geben Sie die Befehle

```
USE adressen
APPEND FROM test SDF
APPEND FROM test1 DELIMITED
APPEND FROM test2 DELIMITED
APPEND FROM test3 DELIMITED
```

und erhalten 16 Datensätze in der Adressendatei (die als zuvor leer angenommen wurde, Bild 5.26). Im Anschluß daran können die normalen Verarbeitungsprogramme ablaufen.

```
00001  123 Maier     Herbert   Amalienstr.59    8000 München      1000.00
00002  234 Huber     Josef     Sieglindenstr.5  8000 München       643.60
00003  345 Kuhn      Peter     Fliederstr.9     6100 Darmstadt     560.00
00004  098 Meister   Werner    Marktplatz 3     8057 Eching       1122.58
00005  123 Maier     Herbert   Amalienstr.59    8000 München      1000.00
00006  234 Huber     Josef     Sieglindenstr.5  8000 München       643.60
00007  345 Kuhn      Peter     Fliederstr.9     6100 Darmstadt     560.00
00008  098 Meister   Werner    Marktplatz 3     8057 Eching       1122.58
00009  123 Maier     Herbert   Amalienstr.59    8000 München      1000.00
00010  234 Huber     Josef     Sieglindenstr.5  8000 München       643.60
00011  345 Kuhn      Peter     Fliederstr.9     6100 Darmstadt     560.00
00012  098 Meister   Werner    Marktplatz 3     8057 Eching       1122.58
00013  123 Maier     Herbert   Amalienstr.59    8000 München      1000.00
00014  234 Huber     Josef     Sieglindenstr.5  8000 München       643.60
00015  345 Kuhn      Peter     Fliederstr.9     6100 Darmstadt     560.00
00016  098 Meister   Werner    Marktplatz 3     8057 Eching       1122.58
```

Bild 5.26: Aus Textdateien eingelesene Datensätze

Auf diese Weise lassen sich auch Tabellen aus dem Kalkulationsprogramm Multiplan nach dBASE II übernehmen und dort weiterverarbeiten. Dazu muß eine dBASE-Dateistruktur entsprechend der Multiplan-Tabelle erzeugt und letztere als Textfile gespeichert werden. Bild 5.27 zeigt einen Ausschnitt aus einer für Multiplan typischen Tabelle und Bild 5.28 eine Dateistruktur, die dem Tabelleninhalt (ohne Überschriften und ohne Summenzeile) angepaßt ist.

	1	2	3	4	5	6	7
1			Budget	Blatt 1			
2							
3			in 1000 DM				
4							
5		Januar	Februar	März	April	Mai	Juni
6	----------	----------	----------	----------	----------	----------	----------
7	Gehälter	72	74	73	75	76	74
8	Pers.Nebenk.	21	23	24	25	24	25
9	Betriebsmittel	10	4	8	6	9	8
10	Dienstleist.	20	24	31	26	22	25
11	Abschreibungen	15	15	15	15	16	17
12	Sonst. Kosten	5	2	6	9	1	10
13	----------	----------	----------	----------	----------	----------	----------
14	Summe	143	142	157	156	148	159

Bild 5.27: Multiplan-Tabelle

```
Strukturdaten für Datei: C:BUDGET.DBF
Anzahl der Sätze:   00000
Datum der letzten Aktualisierung: 00/00/00
Primäre Datei
Feld     Name        Typ  Länge      Dez.st.
001      TITEL        C    015
002      JAN          N    010
003      FEB          N    010
004      MAR          N    010
005      APR          N    010
006      MAI          N    010
007      JUN          N    010
** Gesamt**              00076
```

Bild 5.28: Dateistruktur für Tabelle

Von der Multiplan-Tabelle wird nun mit dem Multiplan-Befehl

Druck Platte/Diskette

Dateinamen eingeben: budget.txt

eine Textkopie erstellt. Dabei sollte man zuvor mit den Befehlen *Druck Randbegrenzung* (linker und oberer Rand = 0, unterer Rand = 7) und *Druck*

Optionen (Z7:12) die zu speichende Tabelle so begrenzen, daß nur die Zeilen 7 bis 12 (Bild 5.27) übernommen werden. Alle anderen Zeilen, die nur Überschriften oder Summen enthalten, werden für dBASE nicht gebraucht und müßten dort ohnehin gelöscht werden. Dann kann mit

```
USE budget
APPEND FROM budget SDF
```

die Texttabelle in die dBASE-Datei eingelesen werden (Bild 5.29). Auf diese Weise kann man beliebig viele Tabellen nacheinander einlesen und unter dBASE konsolidieren, z.B. mehrere Abteilungen oder verschiedene Zeiträume zusammenfassen. Zur Auswertung sortiert man die Datei nach "Titeln" (Kostenstellen) und erzeugt über den Reportgenerator Gruppen- und Endsummen.

00001	Gehälter	72	74	73	75	76	74
00002	Pers.Nebenk.	21	23	24	25	24	25
00003	Betriebsmittel	10	4	8	6	9	8
00004	Dienstleist.	20	24	31	26	22	25
00005	Abschreibungen	15	15	15	15	16	17
00006	Sonst. Kosten	5	2	6	9	1	10

Bild 5.29: Die aus der Multiplan-Tabelle übernommenen dBASE-Daten

6 Befehlsübersicht

Die folgende Übersicht beschreibt kurz sämtliche dBASE-Befehle in alphabetischer Reihenfolge. Die Befehle mit den Sonderzeichen "*", "?" und "@" werden anschließend erläutert.

Soweit erforderlich, werden zu jedem Befehl ein oder mehrere Beispiele gezeigt, die Ihnen helfen sollen, diesen syntaktisch richtig anzuwenden. Für eine ausführliche Beschreibung der Kommandos sei auf das dBASE-Handbuch verwiesen.

ACCEPT (Erläuterung) TO (Speichername)

Mit diesem Befehl können alphanumerische Ausdrücke von der Tastatur in den bezeichneten Speicher übernommen werden. Als "Erläuterung" kann ein beliebiger Text auf dem Bildschirm eingeblendet werden, der den Benutzer über die Art der erwarteten Daten informiert.

Beispiel:

ACCEPT 'Der Name des Kunden ist: ' TO Mname

APPEND

Mit APPEND werden neue Datensätze an die mit dem Befehl USE eröffnete Datei angehängt und zur Eingabe von Daten editiert. Nach Erfassung eines Datensatzes wird der nächste automatisch editiert.

APPEND BLANK als Variante hängt einen einzigen neuen Datensatz an die Datei, ohne ihn gleichzeitig zu editieren. APPEND BLANK ist geeignet, innerhalb von Programmen neue Datensätze aufzunehmen, deren Inhalt nicht notwendigerweise über die Tastatur erfaßt wird.

APPEND FROM (Dateiname) FOR (Bedingung)

erlaubt, Datensätze aus einer Datei (Dateiname) zu entnehmen und an die eröffnete Datei anzuhängen. Die zu übernehmenden Sätze können mit Hilfe der FOR-Bedingung selektiert werden.

Beispiel:

APPEND FROM bewegung FOR nummer > 35

Der Befehl APPEND FROM kann durch einen weiteren Zusatz ergänzt werden: **SDF** bzw. **DELIMITED**. Damit können Datensätze aus Dateien mit einem anderen Dateiformat übernommen werden. Der Zusatz SDF (Standard Data Format) wird benützt, um Dateien im ASCII-Format, d.h., mit festen Feldlängen (wie in dBASE) einzulesen. DELIMITED wird angegeben, wenn die Felder der Quelldatei von variabler Länge und durch Sonderzeichen (z.B. Kommas) voneinander getrennt sind.

Beispiele:

1. SDF-Datei:

Maier	Herbert	8000 München
Kuhn	Peter	6100 Darnstadt
Hassold	Paul	2000 Hamburg

APPEND FROM (Dateiname) SDF

2. ASCII-Datei:

```
Maier,Herbert,8000 München
Kuhn,Peter,6100 Darnstadt
Hassold,Paul,2000 Hamburg
```

APPEND FROM (Dateiname) DELIMITED

BROWSE

Mit BROWSE werden bis zu 19 Datensätze einer Datei gleizeitig editiert. Die Datensätze werden zeilenweise dargestellt. Der Cursor kann in beliebiger Richtung über alle Sätze und Felder bewegt werden, um Änderungen an den Daten vorzunehmen.

Neben den Cursortasten können insbesondere folgende Tastenfunktionen innerhalb BROWSE benutzt werden:

SHIFT-↓	oder CTRL-C bewegt den Cursor senkrecht nach unten durch alle Datenfelder gleichen Typs.
CTRL-G	oder CLR löscht das Zeichen, auf dem der Cursor steht.
SHIFT-↑	oder CTRL-R bewegt den Cursor senkrecht nach oben durch alle Datenfelder gleichen Typs.
CTRL-B	verschiebt das Bildfenster um ein Datenfeld nach rechts (wenn nicht alle Felder gleichzeitig sichtbar sind)
CTRL-Z	verschiebt das Bildfenster um ein Datenfeld nach links (wenn nicht alle Felder gleichzeitig sichtbar sind)
CTRL-U	oder SHIFT-COPY markiert den Datensatz, in dem sich der Cursor befindet, zur späteren Löschung.
CTRL-V	oder COPY schaltet zwischen "Überschreiben" und "Einfügen" um.
CTRL-W	schließt die Funktion BROWSE ab.

BROWSE FIELDS (Felderliste) erlaubt das Editieren von ausgewählten Datenfeldern. Die Felder erscheinen in der Reihenfolge (nebeneinander) im Editor, wie sie in der Felderliste angegeben sind.

CANCEL

beendet einen Programmablauf und veranlaßt die Rückkehr in die Bereitschaft, einzelne Befehle zu empfangen.

CASE siehe DO CASE

CHANGE (Bereich) FIELD (Feldnamen) FOR (Bedingung)

erlaubt das Editieren einzelner oder mehrerer Datenfelder, wobei jedes eventuell zu ändernde Feld einzeln aufgerufen wird. Mit der FOR-Bedingung können die Datenfelder ausgewählt werden.

Beispiele:

```
CHANGE FIELD datum

CHANGE ALL FIELD umsatz FOR ort = '8'
```

CLEAR

schließt alle Dateien und löscht alle Speicher. Eine anschließend eröffnete Datei wird stets als primäre Datei behandelt, auch wenn vor dem CLEAR der Befehl SELECT SECONDARY gegeben war.

CLEAR GETS

als Variante schließt die mit GET-Befehlen zuvor aufgerufenen Eingabefelder. Die Bildschirmmaske selbst bleibt dabei erhalten.

Programmbeispiel:

```
@ 5,5 SAY .... GET ....
@ 7,5 SAY .... GET ....
IF (Bedingung)
   CLEAR GETS
ELSE
   READ
ENDIF
```

CONTINUE

veranlaßt nach einem vorausgegangenen LOCATE-Befehl die weitere Suche nach dem nächsten Datensatz, der die Bedingung des Befehls LOCATE erfüllt.

COPY TO (Dateiname) FIELD (Feldnamen) FOR (Bedingung)

COPY TO (Dateiname) kopiert die mit USE eröffnete Datei komplett, d.h. Dateistruktur und Dateiinhalt in die zu benennende Datei. Vor dem Namen der Zieldatei kann das Diskettenlaufwerk angegeben werden, auf dem die neue Datei gespeichert werden soll: z.B. B:adressen.

Mit dem optionalen Zusatz FIELD (Feldnamen) kann der Kopiervorgang auf bestimmte Felder begrenzt werden. Es werden dann nur die aufgeführten Felder kopiert, z.B.: FIELD name,ort,kunummer.

Ebenfalls wahlweise zu verwenden ist eine **FOR**-Klausel, mit der nur diejenigen Datensätze kopiert werden, die diese Bedingung erfüllen.

Schließlich kann mit dem Zusatz **SDF** oder mit **DELIMITED (WITH (Zeichen))** der Zieldatei ein anderes als das dBASE-Format gegeben werden, das die Weiterverarbeitung durch andere Programme wie z.B. Textverarbeitung erlaubt.

Mit der Option SDF wird eine Standard-ASCII-Datei mit fester Satzlänge angelegt. Der Zusatz DELIMITED (WITH (Zeichen)) erzeugt Dateien mit variabler Satzlänge, deren Felder durch ein Komma getrennt sind. Mit WITH (Zeichen) kann ein Sonderzeichen angegeben werden, das die alphanumerischen Felder einschließt. Ohne die Angabe von WITH schließen einfache Anführungszeichen die alphanumerischen Felder ein.

Wichtiger Sonderfall: WITH , (Komma) bewirkt, daß die alphanumerischen Felder *nicht* in Sonderzeichen eingeschlossen werden.

Beispiele:

COPY TO adressen SDF

```
Maier       Herbert     640.00
Kuhn        Peter      1244.00
Hassold     Paul        345.00
```

COPY TO adressen DELIMITED WITH *

```
*Maier*,*Herbert*, 640.00
*Kuhn*,*Peter*, 1244.00
*Hassold*,*Paul*, 345.00
```

COPY TO adressen DELIMITED

```
'Maier','Herbert', 640.00
'Kuhn','Peter', 1244.00
'Hassold','Paul', 345.00
```

COPY TO adressen DELIMITED WITH ,

```
Maier,Herbert,640.00
Kuhn,Peter,1244.00
Hassold,Paul,345.00
```

COPY TO (Dateiname) STRUCTURE EXTENDED

kopiert die Struktur der Datei als Datensätze in die Zieldatei. Siehe Beispiel im Kapitel 5.7.

Anmerkung: Datensätze, die zur Löschung markiert sind, werden von COPY nicht verarbeitet. Damit kann COPY auch zur physischen Löschung von Sätzen verwendet werden.

COUNT FOR (Bedingung) TO (Speichername) FOR (Bedingung)

zählt die Anzahl der Datensätze in der aufgerufenen Datei. Bei Angabe einer FOR-Bedingung werden nur die Datensätze gezählt, die diese Bedingung erfüllen. Mit der Klausel TO (Speichername) kann das Zählergebnis gespeichert werden.

CREATE (Dateiname)

Mit diesem Befehl wird die Struktur einer neuen Datei generiert. Das Festlegen der Strukturparameter erfolgt unter Benutzerführung. Pro Datenfeld, das definiert werden soll, sind der Feldname, der Feldtyp und die Feldlänge (dazu evtl. die Anzahl der Dezimalstellen) anzugeben.

DELETE (Bereich) FOR (Bedingung)

Markiert alle Datensätze im vorgegebenen Bereich der eröffneten Datei für eine spätere Löschung, wobei innerhalb des Bereichs weiter selektiert werden kann, indem die FOR-Klausel mit der entsprechenden Auswahlbedingung angewandt wird.

Beispiele:

DELETE RECORD 2	Datensatz Nr.2
DELETE NEXT 7	die nächsten 7 Sätze
DELETE ALL	alle Datensätze
DELETE NEXT 15 FOR umsatz < 1000	die nächsten 15 Sätze, sofern der Umsatz kleiner als 1000 ist.

Die Markierung wird äußerlich durch einen **Stern (*)** vor dem ersten Datenfeld dargestellt und kann innerhalb von Befehlen, z.B. in Abfragen, verwendet werden. Die eigentliche physische Löschung aller markierten Datensätze erfolgt mit dem Befehl PACK (siehe dort).

Beispiel:

```
IF .NOT. *
   (Befehle für zutreffende Bedingung)
ENDIF
```

DELETE FILE (Dateiname)

löscht die genannte Datei von der Diskette. Im Gegensatz zu einzelnen Datensätzen werden Dateien sofort auch physisch gelöscht. Dateien, die momentan geöffnet sind, können nicht gelöscht werden.

DISPLAY (Bereich) FOR (Bedingung) (Felder) OFF

Mit DISPLAY lassen sich Datensätze aus einem bestimmten Bereich der aufgerufenen Datei anzeigen. In einer FOR-Klausel kann eine Auswahl aus dem angegebenen Bereich getroffen werden.

Wenn nicht der vollständige Datensatz angezeigt werden soll, dann können die einzelnen, anzuzeigenden Felder spezifiziert werden.

Wahlweise kann der Zusatz OFF an den Befehl gehängt werden, der die Ausgabe der laufenden Satznummer unterdrückt.

Beispiele:

```
DISPLAY

DISPLAY NEXT 4

DISPLAY ALL FOR umsatz > 3500

DISPLAY kunummer,umsatz,offen

DISPLAY ALL name,vorname OFF
```

Werden mit einem DISPLAY-Befehl mehr als 15 Datensätze aufgerufen, dann werden zunächst nur die ersten 15 angezeigt. Danach wartet DISPLAY

auf einen (beliebigen) Tastendruck, um die nächsten 15 Sätze auszugeben. Das wird solange wiederholt, bis alle spezifizierten Datensätze angezeigt sind. Im Unterschied zum Befehl LIST können also mit DISPLAY auch grössere Dateien in Ruhe am Bildschirm betrachtet werden.

DISPLAY FILES ON (Laufwerk) LIKE (Dateityp)

listet alle Dateien auf der Diskette (Laufwerk), wobei das Laufwerk nur dann anzugeben ist, wenn es vom augenblicklich aktivierten abweicht. Mit der Klausel LIKE (Dateityp) kann der Typ der gewünschten Dateien angegeben werden. Ohne den Zusatz LIKE werden ausschließlich die Datenbank-Dateien vom Typ **DBF** gelistet.

DISPLAY MEMORY

listet alle momentan definierten Speicher mit ihren Inhalten, ihrer Größe und ihrem Typ auf.

DISPLAY STATUS

liefert Informationen über:

- die eröffneten Dateien/Indexdateien und ihre SELECTierten Arbeitsbereiche
- den Status aller SET-Kommandos (ON/OFF).
- die Belegung der Funktionstasten

DISPLAY STRUCTURE

liefert die Darstellung der Struktur der augenblicklich benutzten Datei.

DO (Programm-Name)

ruft ein gespeichertes Programm auf und bringt es zur Ausführung.

DO CASE

leitet eine oder meist mehrere Abfragen mit anschliessender bedingter Programmverzweigung ein. Der gesamte Abfrageblock wird mit dem Befehl **ENDCASE** abgeschlossen. Jede einzelne Abfrage beginnt mit dem Befehl

CASE (Bedingung)

und einer nachfolgenden Bedingung. Ist die Bedingung erfüllt, dann werden die der Bedingung folgenden Programmzeilen bis zur nächsten CASE-Bedingung oder bis zum Befehl ENDCASE ausgeführt. Die weiteren Bedingungen werden nicht mehr überprüft, sondern sofort übersprungen. Ist die Bedingung nicht erfüllt, dann wird die nächste geprüft. Als Alternative zu den Anweisungen für erfüllte Bedingungen kann man mit dem optionalen Zusatz

OTHERWISE (=andernfalls)

in den folgenden Zeilen Kommandos angeben, die ausgeführt werden, wenn keine der vorangegangenen Bedingungen erfüllt ist.

Es ist zu beachten, daß CASE-Bedingungen innerhalb einer Kommandodatei nicht ineinander geschachtelt werden dürfen (im Gegensatz zu IF-Bedingungen). Unterprogramme, in die aus einer CASE-Bedingung heraus verzweigt wird, dürfen aber erneut CASE-Kommandos beinhalten.

DO WHILE (Bedingung)

initiiert eine Programmschleife, die solange durchlaufen wird, wie die Bedingung erfüllt ist. Nach dem letzten Befehl innerhalb der Schleife ist der Abschlußbefehl

ENDDO

zu setzen. Wird eine Programmschleife infolge Nichterfüllung der Bedingung verlassen, dann kommt als nächstes der dem ENDDO folgende Befehl zur Ausführung.

Beispiele:

DO WHILE antwort = 'ja'	läßt eine Schleife solange durchlaufen, bis eine anderslautende "antwort" eingegeben wird.
DO WHILE zahl < 100	solange wie die "zahl" kleiner als 100 ist.
DO WHILE .NOT. EOF	dient zur satzweisen Bearbeitung einer ganzen Datei in einer Programmschleife. EOF ist eine in dBASE fest definierte Abkürzung für END OF FILE (= Dateiende).

EDIT

editiert den augenblicklich im Zugriff befindlichen Datensatz. Der Satz wird auf dem Bildschirm in einer Maske dargestellt, innerhalb der man den Cursor beliebig bewegen und Änderungen am Inhalt der Felder vornehmen kann.

Mit EDIT (Datensatz-Nummer) kann auch ein Satz, der nicht im direkten Zugriff ist, editiert werden.

Neben den Cursortasten sind folgende Tastenkombinationen innerhalb der Funktion EDIT anwendbar:

SHIFT-↓	oder CTRL-C schreibt den editierten Satz auf die Diskette und holt den folgenden Satz zur Bearbeitung auf den Bildschirm.
CTRL-G	oder CLR löscht das Zeichen, auf dem der Cursor steht.
CTRL-Q	anulliert die Änderung des editierten Satzes und verläßt die EDIT-Funktion.

SHIFT-↑	oder CTRL-R schreibt den editierten Satz auf die Diskette und holt den vorhergehenden Satz zur Bearbeitung auf den Bildschirm.
CTRL-U	markiert den editierten Satz zur Löschung.
CTRL-V	oder COPY schaltet zwischen "Überschreiben" und "Einfügen" um.
CTRL-W	schließt die EDIT-Funktion ab.

EJECT

veranlaßt einen Seitenvorschub, wenn der Drucker eingeschaltet und aktiviert ist.

ENDDO

schließt eine DO WHILE-Schleife ab. Siehe DO WHILE.

ERASE

löscht den Bildschirm und positioniert den Cursor in die linke obere Ecke.

FIND (Text)

dient zum schnellen Aufsuchen von Datensätzen in Indexdateien. Gefunden wird der erste Datensatz, in dessen Indexfeld (Schlüsselfeld) der angegebene Text als Feldinhalt oder als Teil davon auftritt. Im letzteren Fall muß es der Anfangsteil des Feldinhalts sein.

Mit **FIND &(Speichername)** kann der Suchbegriff einem Speicher entnommen werden (siehe auch "&").

FIND sucht stets vom Dateianfang aus und nur bis zum ersten zutreffenden Satz. Alle weiteren Sätze, für die der Suchbegriff auch zutrifft, können mit FIND nicht gefunden werden. Allerdings folgen diese in der Indexdatei dem gefundenen Satz unmittelbar.

Existiert kein Datensatz mit dem angegebenen Suchbegriff, dann erscheint die dBASE-Nachricht: Wert nicht gefunden (NO FIND). Der Datensatzzeiger weist dann auf die Nummer 00000, die wie jede andere andere Satznummer mit der Funktion # abgefragt weren kann.

Beispiele:

FIND 123

FIND &name

GET

ruft, zusammen mit dem Positionierbefehl "@", Daten aus Speichern oder aus Dateien auf den Bildschirm, die dort mit dem Befehl READ editiert und wieder eingelesen werden. Siehe auch READ. Mit dem optionalen Zusatz

PICTURE '(Datenmaske)'

werden die eingegebenen Daten formatiert. So kann man in den Lesebefehl Plausibilitätskontrollen einbauen oder Vorsorge treffen, daß alle Daten in einem einheitlichen Format erfaßt werden. Für jede Position innerhalb der Maske läßt sich der zulässige Zeichentyp festlegen. So bedeutet die Angabe der Ziffer "9", daß nur Ziffern oder Vorzeichen sowie Dezimalpunkte erlaubt sind. In Positionen, die mit "A" markiert sind, können nur Buchstaben eingetragen werden. "X" läßt alle Zeichen zu. Die meisten anderen Zeichen können als "Konstanten" verwendet werden, d.h. sie werden als fester Bestandteil aller Eingaben behandelt und eingelesen.

Beispiele:

@ 4,20 GET umsatz

@ 5,5 SAY 'Betrag ' GET betrag PICTURE '9999.99'

@ 4,10 SAY 'Datum ' GET datum PICTURE '99/99/99'

@ 3,5 SAY 'Name ' GET name PICTURE 'AAAAAAAAAAAAAA'

@ 5,5 SAY 'KFZ-Nr. ' GET autonummer PICTURE 'AAA-AXX999'

Bei der Eingabe des Datums werden die Schrägstriche vom System selbst gesetzt. In die Felder für die KFZ-Kennzeichen dürfen anfangs nur Buchstaben, am Ende nur Ziffern eingetragen werden.

GO oder auch **GOTO**

Mit GO bzw. GOTO wird der Datensatzzeiger auf eine bestimmte Datensatznummer gesetzt. Folgende Varianten sind möglich:

GO RECORD (Nummer) oder einfach

GO (Nummer)	zur direkten Adressierung eines Satzes, z.B. GO 12
GO (Speichername)	zur indirekten Adressierung eines Datensatzes, dessen Nummer in dem bezeichneten Speicher steht
GO TOP	setzt den Zeiger auf den ersten Satz in der Datei
GO BOTTOM	setzt den Zeiger auf den letzten Satz in der Datei

HELP

liefert allgemeine Informationen über dBASE und einzelne Befehle, wenn diese mit angegeben werden.

Beispiel:

HELP UPDATE

IF (Bedingung)

erlaubt von der Bedingung abhängige Programmverzweigungen. Der vollständige Befehl lautet:

IF (Bedingung)
  (Befehle für zutreffende Bedingung)
ELSE
  (Befehle für nicht zutreffende Bedingung)
ENDIF

Der Befehl ELSE (= andernfalls) mit seinen nachfolgenden Befehlen für eine nicht zutreffende Bedingung ist wahlweise möglich. ENDIF dagegen muß angegeben werden.

Ist die IF-Bedingung erfüllt, dann werden die nachfolgenden Befehle ausgeführt, bis der Befehl ELSE oder - wenn dieser nicht vorhanden ist - der Ende-Befehl ENDIF auftritt.

IF-Befehle können ineinander geschachtelt werden, jedoch muß für jedes IF ein ENDIF vorhanden sein.

INDEX ON (Feldname) TO (Dateiname)

Mit INDEX wird für die eröffnete Datei (Stammdatei) eine Indexdatei angelegt. Indiziert wird nach dem mit (Feldname) bezeichneten Schlüsselfeld (Indexfeld). Die erzeugte Indexdatei (Dateiname) ist nach dem Schlüsselfeld sortiert.

Zum Eröffnen der Indexdatei wird das Kommando

USE (Stammdatei) INDEX (Indexdatei)

gegeben. Manipulationen an der Indexdatei wie APPEND, EDIT, REPLACE, READ oder BROWSE führen automatisch auch zum Aktualisieren der Stammdatei.

Pro Stammdatei können beliebig viele Indexdateien angelegt werden, von denen maximal sieben gleichzeitig aktualisiert werden können. (Siehe auch SET INDEX ON).

Beispiel:

```
USE adressen                          (Stammdatei)
INDEX ON name TO alphabet
USE adressen INDEX alphabet           (Indexdatei)
```

INPUT (erläuternder Text) TO (Speichername)

erlaubt die Übernahme von Einzelinformationen über die Tastatur in einen Speicher. Im Gegensatz zu dem Befehl ACCEPT können sowohl numerische als auch alphanumerische und logische Daten eingegeben werden. Alphanumerische Informationen müssen allerdings in Anführungszeichen gesetzt werden.

Zur Erläuterung kann ein Text eingegeben werden, der vor jeder Dateneingabe auf dem Bildschirm eingeblendet wird.

Beispiel:

```
INPUT 'Rechnungsbetrag: ' TO Mbetrag
```

INSERT

Mit INSERT wird ein einzelner Datensatz in eine Datei eingefügt. Das Einfügen erfolgt unmittelbar nach dem Satz, auf den der Datensatzzeiger augenblicklich weist.

INSERT BEFORE fügt den neuen Satz vor diesem Datensatz ein.

INSERT BLANK fügt einen leeren Datensatz ein

JOIN TO (Dateiname) FOR (Bedingung) FIELDS (Feldnamen)

Der Befehl JOIN verknüpft zwei als primäre und sekundäre bezeichnete Dateien zu einer dritten, deren Datensätze aus Kombinationen der beiden ersten Dateien gebildet werden.

Der Name der neuen Datei ist als (Dateiname) anzugeben. In der FOR-Klausel werden die Bedingungen für die Bildung der neuen Datensätze festgelegt.

Die Anzahl der neu gebildeten Datensätze ist wesentlich von der FOR-Bedingung abhängig. Maximal kann jeder Satz der primären mit jedem Satz der sekundären Datei verknüpft werden, was unter Umständen zu sehr grossen Dateien führen kann.

Hinter FIELDS werden die Felder aus den beiden Ursprungsdateien gelistet, die auch in der neuen Datei vorhanden sein sollen.

Beispiele:

```
USE RECHNUNG
SELECT SECONDARY
USE ADRESSEN
SELECT PRIMARY

JOIN TO mahnung FOR kunde = S.kunde FIELDS name,betrag

JOIN TO mahn FOR (kunde =S.kunde .AND. rnr > 12000);
              FIELDS name,betrag,datum,ortstrasse
```

LIST (Felder) FOR (Bedingung) OFF

gibt zeilenweise alle Datensätze einer Datei aus. Die Datenfelder erscheinen nebeneinander in der Reihenfolge, wie sie in der Dateistruktur erfaßt sind.

Wenn nicht der vollständige Datensatz ausgegeben werden soll, dann können die einzelnen, anzuzeigenden Felder spezifiziert werden. Mit der FOR-Klausel kann eine Auswahl aus dem Bereich getroffen werden.

Der optionale Zusatz OFF unterdrückt die laufende Satznummer.

Beispiele:

LIST

LIST vorname, name OFF

LIST FOR umsatz > 3500

LIST FILES ON (Laufwerk) LIKE (Dateityp)

listet alle Dateien auf der Diskette (Laufwerk), wobei das Laufwerk nur dann anzugeben ist, wenn es vom augenblicklich aktivierten abweicht. Mit der Klausel LIKE (Dateityp) kann der Typ der gewünschten Dateien angegeben werden. Ohne den Zusatz LIKE werden ausschließlich die Datenbank-Dateien vom Typ **DBF** gelistet.

Beispiel:

LIST FILES LIKE *.FRM

LIST MEMORY

listet alle momentan definierten Speicher mit ihren Inhalten, ihrer Größe und ihrem Typ auf. Der Befehl ist mit DISPLAY MEMORY identisch.

LIST STATUS ist identisch mit DISPLAY STATUS und liefert Informationen über den augenblicklichen Betriebszustand von dBASE, nämlich über

- die eröffneten Dateien/Indexdateien und ihre SELECTierten Arbeitsbereiche
- den Status aller SET-Kommandos (ON/OFF).
- die Belegung der Funktionstasten (falls vorhanden und Installiert)

LIST STRUCTURE

liefert die Darstellung der Struktur der augenblicklich benutzten Datei. Der Befehl ist mit DISPLAY STRUCTURE identisch.

LOCATE FOR (Bedingung)

Mit LOCATE wird ein Datensatz in einer Datei gesucht, für den die genannte (Bedingung) zutrifft. In Gegensatz zu dem Befehl FIND sucht LOCATE stets von der Dateiposition aus, auf die augenblicklich der Datensatzzeiger weist. Mit LOCATE können also nacheinander sämtliche Sätze gefunden werden, für die die Bedingung zutrifft. Die dem ersten gefundenen Satz folgenden Suchschritte werden mit dem Befehl

CONTINUE

ausgelöst. Die Suchbedingung kann sich auf jedes Datenfeld beziehen, wobei auch logische Verknüpfungen über mehrere Felder zulässig sind.

Beispiele:

```
LOCATE FOR name = 'Meier'

LOCATE FOR kunummer = '123'

LOCATE FOR umsatz = 500.00

LOCATE FOR (name = 'Meier' .AND. 'J'$vorname)
```

LOOP

Der Befehl LOOP veranlaßt innerhalb einer DO WHILE-Schleife die sofortige Rückkehr zum Schleifenanfang. Damit können - meist zusammen mit IF-Befehlen - große Schleifen bedingt verkürzt werden.

In der Praxis wird LOOP kaum benötigt, da der gleiche Effekt, nur übersichtlicher, auch mit IF/ELSE-Befehlen erreicht werden kann.

MODIFY COMMAND (Dateiname)

dient zum Aufbauen und Ändern von Befehlsdateien. Der Befehl arbeitet im FULL SCREEN EDIT MODE, d.h. der gesamte Bildschirm steht für das Editieren zur Verfügung, und es sind alle Möglichkeiten der Cursorsteuerung gegeben. Folgende Funktionstasten stehen zum Editieren zur Verfügung:

SHIFT-↓	oder CTRL-C blättert den Bildschirm eine Halbe Seite abwärts.
SHIFT-↑	oder CTRL-R blättert den Bildschirm eine Halbe Seite aufwärts.
CLR	oder CTRL-G löscht das Zeichen unter dem Cursor.
CTRL-CLR	oder CTRL-Y löscht die Zeile ab der Cursorposition.
SHIFT-CLR	oder CTRL-T löscht die laufende Zeile.
CTRL-N	fügt eine Leerzeile ein.
CTRL-V	oder COPY schaltet zwischen "Überschreiben" und "Einfügen" um.
CTRL-Q	anulliert die Änderung des editierten Satzes und verläßt den Editor.
CTRL-W	schließt die EDIT-Funktion ab.

Reportdateien lassen sich mit MODIFY COMMAND ebenfalls ändern. In diesem Fall (und nur in diesem) muß aber der Dateityp ".FRM" des Reportprogramms im Dateinamen mit angegeben werden.

MODIFY STRUCTURE

dient zum Ändern von Datendatei-Strukturen. Hierbei wird die Datei in ihrer Struktur auf den Bildschirm ausgegeben und kann im FULL SCREEN EDIT MODE, d.h. mit allen Möglichkeiten der Cursorsteuerung geändert werden. Folgende Funktionstasten stehen zum Editieren zur Verfügung:

SHIFT-↓	oder CTRL-C blättert den Bildschirm eine Halbe Seite abwärts.
SHIFT-↑	oder CTRL-R blättert den Bildschirm eine Halbe Seite aufwärts.
CLR	oder CTRL-G löscht das Zeichen unter dem Cursor.
CTRL-CLR	oder CTRL-Y löscht das Datenfeld ab der Cursorposition.
SHIFT-CLR	oder CTRL-T löscht die laufende Zeile (d.h. das Datenfeld).
CTRL-N	fügt eine Leerzeile ein.
CTRL-V	oder COPY schaltet zwischen "Überschreiben" und "Einfügen" um.
CTRL-Q	anulliert die Änderung des editierten Satzes und verläßt den Editor.
CTRL-W	schließt die EDIT-Funktion ab.

NOTE

Im Anschluß an den Befehl NOTE können Kommentare in einer Programmdatei untergebracht werden, die bei Ablauf des Programms übersprungen werden.

Die gleiche Wirkung wie NOTE hat das Symbol *

Beispiele:

NOTE Beginn des Programms rechnung

* Beginn des Programms rechnung

PACK

Pack bewirkt das physische Löschen von Datensätzen, die mit dem Befehl DELETE markiert wurden.

Alternativ zu PACK kann auch der Befehl COPY zum Löschen von markierten Sätzen dienen, wobei allerdings eine zweite Datei als Zieldatei benötigt wird.

Nach Abschluß von PACK erscheint eine Systemnachricht, die über die verbliebenen Datensätze Auskunft gibt.

Beispiel: 'PACK' durchgeführt 00012 Sätze kopiert

QUIT

Quit beendet das Programm dBASE und gibt die Kontrolle an das Betriebssystem zurück.

QUIT TO (Programm-Dateien-Liste)

erlaubt, dBASE zu verlassen und sofort andere Aktivitäten von der Kontrollebene des Betriebssystems aus zu starten. Dazu müssen das Betriebssystem und eventuell benötigte Dienstprogramme auf demselben Laufwerk gespeichert sein wie das dBASE-System. Die Betriebssystem-Kommandos sind in Anführungszeichen zu setzen.

Wird QUIT TO innerhalb eines Programms benutzt, dann kehrt das System nach Ausführung der Betriebssystembefehle ins Programm, und zwar zur nächsten Befehlszeile zurück. Die Datendateien werden allerdings dabei geschlossen.

Beispiel:

```
QUIT TO 'STAT *.*'
```

READ

Mit READ wird eine Editiermaske aktiviert, die zuvor mit Hilfe der Befehle "@" und GET definiert wurde.

Die mit GET (Feldname) aufgerufenen Felder werden an den mit @(Zeile, Position) festgelegten Bildschirmpositionen zur Anzeige gebracht und können wie im Editierbetrieb geändert werden. Neben Feldern aus Datensätzen können auch Speicherinhalte editiert werden.

Zusätzlich kann die Option **NOUPDATE** (lies: No-Update) benutzt werden, die das (sonst automatische) Aktualisieren der eröffneten Indexdateien verhindert. Dies ist dann sehr zeitsparend, wenn von den vorausgegangenen GET-Operationen keine indizierten Felder betroffen waren, wenn eine Aktualisierung von Indexdateien also nicht notwendig ist.

Beispiel:

```
@ 5,15 SAY 'Rechnungsbetrag: ' GET betrag
@ 7,15 SAY 'bisherige Zahlung: ' GET bezahlt
READ NOUPDATE
```

RECALL

Der Befehl RECALL macht die Löschmarkierung eines Datensatzes rückgängig. Er ist damit das Gegenstück zum Befehl DELETE.

Mit RECALL können Bereichsangaben oder auch FOR-Klauseln verbunden werden.

Beispiele:

RECALL RECORD 4

RECALL ALL FOR name = 'M'

REINDEX

veranlaßt das erneute Indizieren einer Datei, für die bereits eine Indexdatei existiert (und die eröffnet sein muß). Das Ergebnis von REINDEX entspricht dem des Befehls INDEX ON.

RELEASE (Speichernamen)

Mit RELEASE können einzelne oder alle Speicher gelöscht werden. Um ausgewählte Speicher zu löschen, können diese Optionen benutzt werden:

RELEASE ALL LIKE (Ausdruck)

RELEASE ALL EXCEPT (Ausdruck)

Beispiele:

RELEASE Mbezahlt,Moffen

RELEASE ALL	löscht alle Speicher
RELEASE ALL LIKE abc*	löscht alle Speicher, deren Namen mit abc beginnen

RELEASE ALL EXCEPT abc*	löscht alle Speicher mit Ausnahme derer, die mit abc beginnen

REMARK

Mit REMARK werden Kommentare in ein Programm eingefügt, die (im Gegensatz zu NOTE) während des Programmlaufs auf dem Bildschirm erscheinen bzw. ausgedruckt werden.

RENAME (alter Dateiname) TO (neuer Dateiname)

RENAME erlaubt die Umbenennung von Dateien, und zwar jeder Datei, die im Directory aufgeführt ist.

In der Regel ist der Dateityp mit anzugeben. Wird dies unterlassen, dann nimmt dBASE stets den Typ **DBF** (Datenbankdatei) an.

Beispiele:

RENAME menue.CMD TO auswahl.CMD

RENAME rechnung TO umsaetze

REPLACE (Bereich) (Feld) WITH (Ausdruck)

REPLACE ersetzt im angegebenen Bereich den Inhalt des genannten Datenfeldes durch einen anderen Wert (Ausdruck).

Der Bereich kann, muß aber nicht angegeben werden. Wird der nicht angegeben, dann gilt der Befehl REPLACE nur für den augenblicklich im Zugriff befindlichen Datensatz.

Als Ausdruck kann ein Wert, ein Name eines Feldes oder Speichers oder eine Formel stehen.

Wahlweise kann eine **FOR**-Klausel verwendet werden, die den Gültigkeitsbereich des Befehls weiter einschränkt.

Die Option **NOUPDATE** bewirkt, daß die automatische Aktualisierung einer zugehörigen Indexdatei unterbleibt. NOUPDATE kann zur Einsparung von Rechenzeit benutzt werden, wenn bekannt ist, daß durch REPLACE keine Schlüsselfelder verändert wurden.

Beispiele:

REPLACE bezahlt WITH Mbezahlt

REPLACE ALL offen WITH umsatz-bezahlt FOR offen = 0

REPLACE bezahlt WITH Mbezahlt NOUPDATE

REPORT FORM

Mit REPORT wird der integrierte Listengenerator veranlaßt, einen Bericht auszugeben. Ist dieser Bericht noch nicht definiert und seine Struktur noch nicht abgespeichert, dann wird er zuerst im Dialog mit dem Benutzer generiert.

Die generierte Liste kann beliebig viele Spalten enthalten, deren Inhalte aus Datenfeldern, Speichern, Konstanten oder aus deren Kombination in Formeln bestehen können. Dazu können Überschriften, Zwischensummen mit Gruppenüberschriften und Summen gebildet werden.

Das Druckformat kann auf Wunsch geändert werden, indem der linke Rand, die Anzahl der Zeilen pro Seite und die Anzahl der Zeichen pro Zeile eingegeben werden.

Normalerweise wird beim Erstellen eines Reports eine komplette Datei ausgewertet, die eröffnet sein muß. Mit einer **FOR**-Klausel können aber bestimmte Datensätze, die ausschließlich in den Report einfließen sollen, selektiert werden.

Mit dem Zusatz **TO PRINT** kann der Bericht allein auf einen Drucker ausgegeben werden, ohne daß andere Programmteile ebenfalls in den Druck einbezogen werden.

Mit der Option **PLAIN** können die Seitennummern, das Datum und die Überschriften der Folgeseiten des Reports unterdrückt werden.

Eine bereits gespeicherte Report-Datei kann mit dem Befehl MODIFY COMMAND (Dateiname.**FRM**) editiert werden. Allerdings sind im Editor nur die Antworten sichtbar, nicht aber die Fragen, die beim ursprünglichen Definieren des Reports gestellt wurden.

Beispiele:

REPORT FORM bericht1

REPORT FORM bericht2 FOR gebiet = 'Deutschland'

REPORT FORM bericht1 TO PRINT

REPORT FORM bericht2 PLAIN

RESET (Laufwerk)

ermöglicht die Fortführung eines dBASE-Programms nach einem Diskettenwechsel, ohne daß die Kontrolle vorübergehend an das Betriebssystem zurückgegeben werden muß. Alle Dateien, die vor dem Wechsel geöffnet waren, werden von RESET wieder eröffnet (sofern sie noch auf der aktiven Diskette vorhanden sind).

RESTORE FROM (Dateiname)

Mit RESTORE werden die in einer Speicherdatei (vom Typ MEM) enthaltenen Daten in die Speicher eingelesen. Alle zuvor in den Speichern stehenden Daten werden dabei gelöscht.

Der Befehl kann mit dem Zusatz **ADDITIVE** versehen werden, der die eingelesenen Speicherdaten zu den vorhandenen hinzufügt, letztere also nicht löscht.

Siehe auch SAVE (Speichern von Variablen in einer Datei).

Beispiel:

RESTORE FROM variabel

RESTORE FROM variabel ADDITIVE

RETURN

beendet ein Programm und gibt die Kontrolle an die Ebene zurück, von der aus das Programm gestartet wurde.

SAVE TO (Dateiname)

SAVE speichert für alle augenblicklich definierten Speicher deren Namen und Inhalte in der genannten Datei. Von dort können sie zu einem späteren Zeitpunkt mit RESTORE FROM (Dateiname) zurückgeladen werden. dBASE II kann auch Teilmengen der Speichervariablen in eine Datei schreiben. Dazu werden diese spezifiziert mit:

SAVE TO (Dateiname) ALL LIKE (Ausdruck)

Beispiele:

SAVE TO variabel

SAVE TO variabel ALL LIKE abc* speichert alle Variablen, deren Namen mit abc beginnen.

SELECT

in Verbindung mit **PRIMARY** oder **SECONDARY** wählt einen derbeiden möglichen Dateibereiche für den unmittelbaren Zugriff.

Beim Start von dBASE wird zunächst im primären Dateibereich gearbeitet. Mit SELECT SECONDARY kann zusätzlich zu einer bereits eröffneten

Datei eine zweite eröffnet werden. Mit SELECT PRIMARY bzw. SELECT SECONDARY lassen sich beide Dateien dann abwechselnd bearbeiten.

Das Wesentliche an diesem Befehl ist, daß jede der beiden Dateien ihren eigenen Datensatzzeiger besitzt, der beim Wechsel von einer Datei zur anderen auf dem gegenwärtigen Datensatz stehen bleibt und diesen für eine spätere Rückkehr im Zugriff behält. Damit kann eine Satz-zu-Satz-Kommunikation zwischen beiden Dateien auf gebaut werden.

Treten in der primären und sekundären Datei Felder mit gleichem Namen auf, so können sie in Befehlen durch den **Vorsatz "P."** bzw. **"S."** unterschieden werden.

Beispiele:

```
USE kunden
SELECT SECONDARY
USE rechnung
SELECT PRIMARY
REPLACE datum WITH S.datum

JOIN TO mahnung FOR kunmmer = S.kunmmer ;
FIELDS S.ort,strasse....
```

SET (Funktion) ON / OFF

schaltet eine Reihe von Betriebsarten ein (ON) oder aus (OFF). Durch das entgegengesetzte Kommando SET (Funktion) OFF bzw. ON wird der ursprüngliche Zustand wieder hergestellt.

SET ALTERNATE ON	schreibt die Ausgabe in eine Datei auf Diskette.
SET BELL OFF	schaltet den Alarmton für Fehleingaben ab.
SET CALL ON	ermöglicht den Aufruf von Assemblerroutinen.
SET CARRY ON	erlaubt, beim Erfassen von neuen Datensätzen mit APPEND Daten aus dem zuvor erfaßten Satz in den augenblicklich aufgerufenen zu übernehmen.

SET COLON OFF	unterdrückt die Doppelpunkte, die bei der Funktion GET die Feldbegrenzung anzeigen.
SET CONFIRM ON	schaltet einen Zwang ein, eine Dateneingabe in eine Editiermaske zu bestätigen, bevor zum nächsten Feld übergegangen wird.
SET CONSOLE OFF	unterdrückt die Ausgabe der Ergebnisse auf den Bildschirm.
SET DEBUG ON	gibt die Resultate der ECHO- und STEP-Funktionen auf den Drucker aus.
SET DELETED ON	läßt dBASE alle Datensätze ignorieren, die zum Löschen markiert sind.
SET ECHO ON	läßt alle ausgeführten Befehle eines Programms auf dem Bildschirm erscheinen.
SET EJECT OFF	unterdrückt den Seitenvorschub vor Druckbeginn eines Reports.
SET ESCAPE OFF	unterbindet die Funktion der ESCAPE-Taste.
SET EXACT ON	zwingt zur Eingabe des vollständigen Suchbegriffs bei FIND-Befehlen (anstatt nur einiger Zeichen).
SET INTENSITY OFF	schaltet die Möglichkeit aus, gewisse Editier-Operationen mit halber Bildschirmhelligkeit darzustellen.
SET LINKAGE ON	synchronisiert eine primäre und eine sekundäre Datei für parallele Verarbeitung.
SET PRINT ON	gibt die Ergebnisse auf den Drucker aus.
SET RAW ON	setzt bei Verwendung der Befehle LIST und DISPLAY Zwischenräume zwischen die Felder.

SET SCREEN OFF — ersetzt bei den Befehlen APPEND, INSERT und EDIT die Editiermasken durch sequentiellen Aufruf der einzelnen Zeilen.

SET STEP ON — hält ein Programm nach jedem einzelnen Schritt an und wartet auf eine Entscheidung des Benutzers, ob der nächste Befehl des Programms ausgeführt oder ob andere Befehle über die Tastatur gegeben werden sollen. Diese Funktion ist sehr nützlich zum Suchen von Programmfehlern.

SET TALK OFF — schaltet die dBASE-Systemnachrichten ab.

SET (Funktion) TO (...)

Mit dieser Variante des Befehls SET können die Standardwerte (die sogenannten Default-Werte) einiger Funktionen verändert werden.

SET ALTERNATE TO (Dateiname) eröffnet die genannte Datei und veranlaßt, daß die gesamte Kommunikation mit dBASE, die normalerweise auf dem Bildschirm erscheint, in diese Datei geschrieben wird. Die Datei ist vom Typ ".TXT" und kann von anderen Softwaresystemen, z.B. für Textverarbeitung, übernommen werden.

SET DATE TO mm/dd/yy erlaubt die Eingabe des aktuellen Datums auch während des Arbeitens mit dBASE.

SET DEFAULT TO (Laufwerk) bestimmt das genannte Laufwerk zum Standardlaufwerk. Beim Start von dBASE ist das stets Laufwerk A.

Beispiel: SET DEFAULT TO B: veranlaßt, daß alle USE-Kommandos die Dateien auf der Diskette im Laufwerk B suchen.

SET FORMAT TO PRINT leitet alle Befehle, die mit dem Positionierbefehl "@" beginnen, auf den Drucker. Damit können formatierte Bildschirmdarstellungen direkt vom Drucker übernommen werden.

SET FORMAT TO SCREEN leitet die Ergebnisse der "@"-Befehle (wieder) auf den Bildschirm.

SET FORMAT TO (Formatdatei) ermöglicht, die "@"-Befehle aus einer speziellen Formatdatei zu entnehmen.

SET HEADING TO (Ausdruck) erlaubt, für einen Ausdruck mit dem Reportgenerator eine einmalige zusätzliche Überschrift einzufügen.

SET INDEX TO (Indexdateien) dient zum gleichzeitigen Aktualisieren von bis zu sieben Indexdateien. Die erstgenannte Indexdatei ist die sogenannte Masterdatei, mit der die Suchoperationen ausgeführt werden und deren Sortierordnung der Benutzer sieht. Alle anderen genannten Indexdateien sind nicht direkt sichtbar, werden aber mit den Befehlen APPEND, EDIT etc. ebenfalls aktualisiert.

SET MARGIN TO n setzt den linken Rand eines gedruckten Reports auf die n-te Position, wobei n eine Zahl zwischen 1 und 254 sein muß.

SKIP

setzt den Datensatzzeiger um eine Satznummer höher, so daß dieser auf den nächsten Datensatz zeigt. Durch Angabe einer Zahl mit Vorzeichen kann auf jeden beliebigen Satz gesprungen werden.

Beispiele:

SKIP

SKIP 3

SKIP -7

SORT ON (Feldname) TO (Dateiname)

sortiert die augenblicklich eröffnete Datei nach dem genannten Feld und speichert die sortierte Datei unter dem angegebenen Dateinamen. Normalerweise wird in aufsteigender Reihenfolge (numerisch oder alphanumerisch) sortiert. Absteigende Reihenfolge kann mit dem Zusatz **DESCENDING** gewählt werden.

Beispiele:

SORT ON name TO liste

SORT ON umsatz TO umsatz2 DESCENDING

STORE (Ausdruck) TO (Speichername)

STORE speichert den Ausdruck oder, wenn dieser eine Formel ist, sein Ergebnis in dem genannten Speicher.

Beispiele:

STORE 'Bemerkung' TO text

STORE 12 TO zahl

STORE zahl*6 TO zahl ergibt die neue Zahl 72 im Speicher "zahl"

SUM (Feldnamen) TO (Speichername) FOR (Bedingung)

Mit SUM können die Inhalte numerischer Felder aus der eröffneten Datei addiert werden. Wenn der Summierbereich nicht durch eine FOR-Klausel eingeschränkt wird, dann wird die gesamte Datei in die Summierung einbezogen.

Mit der Option TO (Speichername) kann das Ergebnis in einen Speicher geschrieben werden.

Beispiele:

SUM umsatz,offen

SUM betrag TO erloes

SUM offen FOR name = 'Meier'

TOTAL ON (Schlüssel) TO (Dateiname) FIELD (Liste) FOR (Bedingung)

Mit dem Befehl TOTAL können Dateien komprimiert werden, d.h., es werden diejenigen Datensätze zusammengefaßt und ihre numerischen Felder addiert, die in dem bezeichneten Schlüsselfeld gleichen Inhalt aufweisen. Die neue komprimierte Datei wird unter dem angegebenen Dateinamen gespeichert.

Mit der Option FIELD können die Felder genannt werden, die in der komprimierten Datei noch enthalten sein sollen (ohne die Angabe FIELD wird die komplette Dateistruktur übernommen).

Die ebenfalls optionale FOR-Klausel dient zur Einschränkung des Gültigkeitsbereichs für den Befehl.

Für eine vollständige Zusammenfassung aller Sätze mit gleichem Schlüssel ist es erforderlich, daß die Datei vor Ausführung des Befehls TOTAL nach diesem Schlüssel vorsortiert wird.

Beispiele:

TOTAL ON ort TO kund:ort — faßt die numerischen Daten einer Datei nach Orten zusammen

TOTAL ON name TO kunde FIELD name,betrag — ergibt für eine Rechnungsdatei eine Zusammenfassung aller Rechnungsbeträge pro Kunde

UPDATE FROM (Dateiname) ON (Schlüssel)

UPDATE aktualisiert die eröffnete Datei mit Daten aus einer anderen (FROM-)Datei. Die Beziehung zwischen zusammengehörenden Datensätzen beider Dateien wird über den gleichen Inhalt in den Schlüsselfeldern hergestellt. Beide Dateien müssen nach dem angegebenen Schlüssel sortiert bzw. indiziert sein.

Das Aktualisieren selbst kann auf zweierlei Weise erfolgen:

Mit dem Zusatz **ADD (Feldnamen)** werden die Inhalte der genannten Felder in die eröffnete Datei addiert.

Mit dem Zusatz **REPLACE (Feldnamen) WITH (Feldnamen)** werden die genannten Felder der aktiven Datei überschrieben mit den Daten aus der FROM-Datei. Der Teil "WITH (Feldnamen)" kann entfallen, wenn beide Felder derselben Namen haben.

ADD und REPLACE dürfen in einem Befehl gleichzeitig verwendet werden.

Beispiel:

```
UPDATE FROM bewegung ON rnr ADD bezahlt ;
                                REPLACE datum WITH termin
```

USE (Dateiname)

eröffnet die genannte Datei zur Bearbeitung. Mit

USE (Dateiname) INDEX (Indexdateiname)

werden Indexdateien eröffnet.

Beispiele:

```
USE kunden

USE rechnung INDEX nummer
```

WAIT TO (Speichername)

WAIT veranlaßt ein Programm, anzuhalten und auf einen einzelnen Tastendruck zu warten. Das Zeichen der gedrückten Taste kann wahlweise in den genannten Speicher geschrieben werden, um es dort z.B. für Abfragen zur Verfügung zu halten. Anschließend wird das Programm fortgesetzt.

Beispiele:

WAIT

WAIT TO antwort

* (Stern)

Der Stern zu Beginn einer Zeile erlaubt Kommentare in Programme einzufügen, die nicht verarbeitet werden. Er hat die gleiche Wirkung wie der Befehl NOTE.

? (Fragezeichen)

Mit dem Fragezeichen können Informationen aus Datensätzen, die gerade im Zugriff sind, abgefragt oder Formeln berechnet werden.

Ferner können nach einem Fragezeichen in Anführungszeichen gesetzte Texte in ein Programm aufgenommen werden, die dann auf dem Bildschirm oder im Ausdruck erscheinen. Ein einzelnes Fragezeichen in einer Programmzeile ergibt bei Ablauf des Programms eine Leerzeile.

Beispiele:

? name

? umsatz,offen

? 5*4

? 'Beispiel für Text'

@ (Zeile,Spalte) SAY (Ausdruck)

Das Zeichen @ bewirkt die gezielte Positionierung von Informationen auf dem Bildschirm und in Ausdrucken und ist damit der wesentliche Befehl zur formatierten Ein- oder Ausgabe von Daten. Es ist in seiner Wirkung identisch mit dem Paragraphzeichen, das auf deutschen Tastaturen überwiegend zu finden ist.

Im Anschluß an das Zeichen @ sind die Koordinaten anzugeben, auf die das nächste Zeichen zu positionieren ist. Für die Bildschirmzeile ist eine Zahl zwischen 0 und 23 und für die Spalte eine zwischen 0 und 79 anzugeben.

Zur Druckformatierung können für Zeilen und Spalten die Positionen 0 bis 254 angegeben werden.

Die Koordinaten können auch indirekt aus Speichern abgerufen werden.

Mit dem Zusatz SAY (Ausdruck) wird definiert, welche Daten an die spezifizierte Position geschrieben werden sollen. Hier kann ein Feldname, eine Formel oder ein Text stehen, wobei letzterer in Anführungszeichen zu setzen ist.

Mit einem weiteren Zusatz

USING '(Datenmaske)'

kann die Ausgabe der Daten formatiert werden. Dabei gelten dieselben Regeln wie für das Formatieren mit PICTURE beim Befehl GET.

Anstatt oder zusätzlich zu SAY kann **GET** verwendet werden, um individuelle Masken zur Datenerfassung aufzubauen.

Der Positionierbefehl @ gestattet zusammen mit den SAY-GET-Befehlen auch eine relative Bildschirmadressierung. Die Koordinaten werden dann als Speichervariablen angegeben.

Beispiele:

```
@ 4,10

@ 4,10 SAY 'Kunden-Name: ' + name

@ 6,15 SAY offen/umsatz*100

@ 8,15 SAY 'Rechnungs-Nr. ' GET rnr

@ 10,25 GET datum

@ Mzeile,Mspalte SAY 'Rechnungs-Nr. '
```

7 Funktionsübersicht

Die nachfolgend zusammengestellten Funktionen, von denen die meisten bereits in ihrer praktischen Anwendung vorgeführt wurden, sind stets innerhalb von dBASE-Befehlen einzusetzen, in denen sie einzeln oder untereinander kombiniert auftreten können.

Arithmetische Operatoren

\+ ist das Symbol sowohl für die Addition arithmetischer Ausdrücke als auch für das Zusammenfügen von alphanumerischen Texten.

Beispiele:

```
? 71 + 18
@ 5,5 SAY 'Sehr geehrter Herr ' + name
```

\- ist das Symbol für die Subtraktion arithmetischer Ausdrücke und für das Zusammenfügen von alphanumerischen Texten, wobei aber etwaige Leerstellen zwischen den Textabschnitten entfernt werden.

* in arithmetischen Ausdrücken: Multiplikation

in Dateien wird der Stern als Markierung für einen zu löschenden Datensatz verwendet und kann als Funktion in Befehlen abgefragt werden.

Beispiele:

Multiplikation: STORE umsatz * 0.2 TO gewinn

Löschmarkierung:
```
IF *
   SKIP
ENDIF
```

/ Divisionszeichen

( ) Klammern werden in Befehlen und Formeln nach den üblichen Rechenregeln benutzt.

Vergleichsoperatoren

können sowohl in numerischen als auch in alphanumerischen Ausdrücken eingesetzt werden.

< kleiner als

> größer als

= gleich

<> nicht gleich (ohne Leerzeichen zwischen den beiden Pfeilen!)

<= gleich oder kleiner als

>= gleich oder größer als

Beispiele:

```
DO WHILE zahl < 100
LOCATE FOR kunummer = '123'
IF name <> 'Meier'
```

Logische Operatoren

werden vor allem eingesetzt, um mehrere einzelne Bedingungen zu einer zu verknüpfen. Zur Unterscheidung gegenüber Textzeichen muß vor und nach dem logischen Operator je ein Punkt gesetzt werden (bei T und F nicht notwendig).

.NOT. logisches NICHT (Negation)

.AND. logisches UND (= "sowohl als auch")

.OR. logisches ODER (= "entweder ... oder")

.T. logischer Wert "WAHR" (engl. TRUE)

.F. logischer Wert "FALSCH" (engl. FALSE)

Beispiele:

```
DO WHILE .NOT. EOF
LOCATE FOR (name = 'Meier' .OR. name = 'Meyer')
LOCATE FOR (name = 'Meier' .AND. vorname = 'Josef')
DO WHILE T
```

Sonstige Funktionen

CHR (numerischer Ausdruck)

setzt den numerischen Ausdruck in den äquivalenten ASCII-Code um. Diese Funktion kann zur unmittelbaren Steuerung von Peripheriegeräten (z.B. Druckern) verwendet werden. CHR ist die Umkehrfunktion von RANK.

Beispiel:

```
CHR(13)          ergibt die Funktion der RETURN-Taste
CHR(65)          ergibt das Zeichen A
```

date()

ist ein vom System fest definierter Speicher, der beim Start von dBASE oder später mit dem Befehl SET DATE TO das aktuelle Datum aufnimmt und auf Anforderung bereitstellt.

Beispiel:

```
REPLACE datum WITH date()
```

EOF

END OF FILE- (Dateiende-) Funktion. Mit ihr kann das Ende einer Datei abgefragt werden.

Beispiele:

```
? EOF                                (die dBASE-Antwort ist .T. oder .F.)
DO WHILE .NOT. EOF
IF EOF
    @ 5,5 SAY 'Datei-Ende erreicht'
ENDIF
```

FILE (Dateiname)

fragt, ob die genannte Datei existiert. Als Antwort wird der logische Wert .T. (WAHR) oder .F. (FALSCH) ausgegeben.

INT (numerischer Ausdruck)

trennt von dem angegebenen numerischen Ausdruck alle Nachkommastellen und gibt den ganzzahligen Teil aus.

Beispiele:

? INT(123.456)	ergibt 123
INT(umsatz)	ergibt die vollen DM-Beträge

LEN (alphanumerischer Ausdruck)

ergibt einen ganzzahligen numerischen Wert, der die Anzahl der Zeichen in dem angegebenen Ausdruck angibt.

Beispiele:

? LEN(Beispiel)	ergibt die Zahl 8
LEN(Speichername)	ergibt die Zeichenlänge des Ausdrucks im Speicher

RANK (alphanumerischer Ausdruck)

liefert den (numerischen) ASCII-Wert des ersten Zeichens aus dem angegebenen Ausdruck. RANK ist die Umkehrfunktion von CHR.

Beispiele:

RANK('A')	ergibt die Zahl 65
RANK(Variable)	
RANK(CHR(65))	ergibt wieder die Zahl 65

STR (numerischer Ausdruck,Länge,Dezimalstellen)

wandelt einen numerischen Ausdruck in einen alphanumerischen (STRING, Zeichenkette) um. Neben dem Ausdruck selbst ist die gewünschte (maximale) Länge des Textes anzugeben. Wenn der numerische Ausdruck Dezimalstellen besitzt, kann die Anzahl derjenigen Dezimalstellen angegeben werden, die berücksichtigt werden sollen.

Die Funktion kann u.a. dazu benutzt werden, in Formatierbefehlen numerische und alphanumerische Felder gemeinsam aufzuführen (was sonst nicht möglich wäre).

Beispiele:

```
? STR(123.456,9,1) ergibt den "Text" 123.4
@ 5,5 SAY name + STR(umsatz,10,2)
```

TYPE (Ausdruck)

ermittelt den Datentyp des angegebenen Ausdrucks. Als Datentyp wird entweder "C" (für CHARACTER = alphanumerisch) oder "N" (für numerisch) oder "L" (für logisch) ausgegeben.

Beispiele:

? TYPE(Beispiel)	ergibt die Antwort "C"
? TYPE(bezahlt)	ergibt die Antwort "N", wenn "bezahlt" ein numerisches Feld ist.

TRIM (alphanumerischer Ausdruck)

entfernt aus einem alphanumerischen Datenfeld alle nachfolgenden Leerzeichen, d.h. die nicht benützten Zeichenpositionen.

Beispiel:

```
Datenfeld "vorname":   12 Zeichen, Inhalt: Peter
Datenfeld "name":      12 Zeichen, Inhalt: Kuhn

? vorname + name              ergibt: Peter       Kuhn
? TRIM(vorname) + name        ergibt: PeterKuhn
? TRIM(vorname)+' '+name      ergibt: Peter Kuhn
```

VAL (alphanumerischer Ausdruck)

wandelt einen alphanumerischen Ausdruck in einen numerischen um. Der Ausdruck muß mit Ziffern (bzw. mit einem Vorzeichen vor den Ziffern) beginnen und darf einen Dezimalpunkt enthalten. Eventuell nachfolgende Buchstaben werden nicht berücksichtigt.

Beispiele:

```
? VAL('123')          ergibt die Zahl 123
? VAL('123xyz')       ergibt die Zahl 123
```

* (Stern)

Markierung für zu löschende Datensätze (Beispiel siehe unter "artithmetische Operatoren").

(Nummernzeichen)

ist das Zeichen für die laufende Nummer eines Datensatzes (Satznummer, RECORD NUMBER).

Beispiel:

```
IF # = 0
    @ 5,5 SAY 'Kein Datensatz vorhanden'
ENDIF
```

!(Ausdruck)

wandelt alle im genannten Ausdruck vorkommenden Kleinbuchstaben in Großbuchstaben um. Ziffern und Sonderzeichen bleiben unverändert. Mit dieser Funktion kann die Eingabe von Text über die Tastatur einheitlich auf "Großbuchstaben" gesetzt werden, um zum Beispiel die Auswertung von Tastatureingaben zu vereinfachen.

Beispiel:

```
IF !(antwort) = 'N'
```

$ (alphanumerischer Ausdruck,Start,Länge)

ergibt eine bestimmte Zeichenfolge aus dem alphanumerischen Ausdruck. Die Zeichenfolge wird an der "Start"-Position begonnen und hat die angegebene Länge.

Beispiel:

```
? $(Beispiel,4,5)          ergibt: spiel
```

'Ausdruck'$(Feldname)

als Variante der Funktion "$" wird benutzt, um in Datenfeldern nach bestimmten Zeichenfolgen zu suchen.

Beispiel:

```
LOCATE FOR 'Mei'$ name
```

@ (1.alphanumerischer Ausdruck, 2.alphanumerischer Ausdruck)

sucht innerhalb des zweiten genannten Ausdrucks nach einer Zeichenfolge, die dem ersten Ausdruck entspricht und gibt deren Position als Zahlenwert an.

Beispiele:

? @('def','abcdefghi')	ergibt die Zahl 4
? @('K','Peter Kuhn')	ergibt die Zahl 7

& (Speichername)

dient zur indirekten Adressierung von Befehlsobjekten. Zur Verarbeitung im Befehl gelangen das Feld oder der Datensatz oder die Datei, deren Name in dem angegebenen Speicher steht.

Beispiele:

FIND &Mnummer	sucht in einer Indexdatei den Satz, der im indizierten Feld mit dem Inhalt des Speichers "Mnummer" übereinstimmt.
COPY TO &Laufwerk:(Dateiname)	kopiert die eröffnete Datei auf diejenige Diskette, deren Laufwerk im Speicher "Laufwerk" angegeben ist.

8 Übersicht wichtiger CP/M-Befehle

Das Betriebssystem CP/M von Digital Research, das insbesondere die Mikroprozessoren Intel 8080, 8085 und Zilog Z 80 unterstützt, ist gewissermaßen zum Industriestandard für kommerzielle 8-Bit-Mikrocomputer geworden. Kein anderes 8-Bit-Betriebssystem konnte sich auch nur annähernd so durchsetzen wie CP/M. Aufgrund seiner Verbreitung haben viele Softwarehäuser ihre 8-Bit-Anwendungsprogramme primär für den Einsatz unter CP/M geschrieben. Dazu gehören auch die professionellen Softwarepakete WordStar, Multiplan und dBASE II, die jeweils in ihrer Kategorie zu den Spitzenprodukten im 8-Bit-Markt gehören.

Aufgrund seiner Marktbeherrschung könnte man annehmen, CP/M sei ein großes, komplexes oder besonders leistungsfähiges Betriebssystem. Dem ist aber nicht so. Im Vergleich zu neueren Betriebssystemen für 16-Bit-Rechner ist CP/M ausgesprochen klein. Sein Speicherbedarf (ohne Hilfsprogramme) liegt etwa bei 12 KByte, und sein Befehlsvorrat beschränkt sich auf die nötigsten Funktionen. Dadurch ist jedoch CP/M einfach in der Handhabung, was sicherlich entscheidend zu seiner Beliebtheit beigetragen hat.

Für die Schneider-Computer werden zwei CP/M-Versionen angeboten: Das "klassische" CP/M 2.2 für den CPC 464 und CPC 664 mit 64-K-Speichererweiterung (incl. Erweiterungs-Software) und CP/M PLUS für den Schneider 6128. Wir wollen nachfolgend aus den knapp 20 CP/M-Kommandos diejenigen erläutern, die für Sie als Anwender wichtig sind. Das sind vor allem die Befehle zur Datei- und Diskettenverwaltung. Dabei gibt es zwei Arten von Kommandos: solche, die Teil des eigentlichen CP/M-Betriebssystemkerns sind, und solche, die in CP/M-Hilfsprogrammen (Dienstprogramme oder Utilities genannt) enthalten sind. Um letztere Befehle auszuführen, muß jeweils das entsprechende (meist gleichnamige) Dienstprogramm auf der aktiven Diskette gespeichert sein.

Wir werden zu jedem Befehl angeben, ob er in den CP/M-Kern intergriert oder extern als Utility verwaltet wird. In letzterem Fall geben wir auch die Dateigröße in Kilobyte an. Dann können Sie selbst entscheiden, welche Befehle Sie im Einzelfall auf Ihrer Arbeitsdiskette verfügbar haben wollen oder auf welche Sie lieber zugunsten vor mehr freiem Speicher verzichten. Ferner geben wir auch an, welche Befehle ausschließlich für CP/M 2.2 oder CP/M PLUS vorgesehen sind. Das sind aber nur ganz wenige, die meisten sind für beide Betriebssysteme gleich.

CONFIG (extern, 10 KB)

(**CONFIG**URE = gestalten). Die Datei CONFIG.COM wird als einzige Utility **nicht auf der CP/M-Diskette**, sondern mit der dBASE-II-Software ausgeliefert, weil dieses Programm speziell auf die Bedürfnisse des dBASE-Anwenders unter den Schneider-CPC-Benutzern abgestimmt ist.

CONFIG ruft ein Konfigurationsprogramm auf, mit dessen Hilfe Sie die Tastaturbelegung Ihres Schneider-Computers verändern können. Sie haben die Wahl zwischen den internationalen ASCII-Zeichen und dem deutschen DIN-Zeichensatz. Der deutsche Zeichensatz enthält die Umlaute, das "ß" und das "§"-Zeichen, während der ASCII-Standard statt dessen einige Sonderzeichen vorsieht:

Dez.	ASCII	DIN
64	@	§
91	[	Ä
92	\	Ö
93	]	Ü
123	{	ä
124	|	ö
125	}	ü
126	~	ß

Zusätzlich kann man mit CONFIG auch die Anordnung der Tasten so modifizieren, daß sie der deutschen Norm für Schreibmaschinen entspricht. Dabei werden z.B. die Plätze der Tasten für die Buchstaben "Y" und "Z" vertauscht. Diese Änderung sollte natürlich von einer Neubeschriftung der betroffenen Tasten begleitet sein.

Für dBASE II ist sowohl die ASCII- als auch die DIN-Tastatur brauchbar. Da die ASCII-Sonderzeichen mit Ausnahme von "@" (AT-Zeichen, auch "Klammeraffe" genannt) von dBASE nicht benötigt werden, andererseits aber die deutschen Umlaute in den Datenbeständen sicherlich vorkommen, dürfte die DIN-Tastatur für Sie zweckmäßiger sein. Das "@", das dBASE als Positionierbefehl benutzt, wird dann durch das Paragraph-Zeichen "§" ersetzt. Das Konfigurationsprogramm für dBASE wird aufgerufen mit

CONFIG DBASE.COM <ENTER>

Hinweis: Da die Umlaute und das "ß" vom Computer intern als Sonderzeichen behandelt werden, dürfen sie - wie die meisten anderen Sonderzeichen - nicht in Datei- oder Feldnamen verwendet werden. Das gilt nicht nur für dBASE, sondern generell. Es gibt sogar einige Softwareprodukte, die das Erstellen (Schreiben) von Dateien mit Umlauten im Namen zulassen, nicht aber deren Lesen, was natürlich besonders unangenehm ist, weil man später an die erfaßten Informationen nicht mehr herankommt. Als "letzte Rettung" kann man dann nur noch versuchen, solche Dateien umzubenennen (siehe Befehl REN).

Haben Sie sich einmal für eine Tastenbelegung entschieden, dann können Sie die Datei CONFIG.COM von Ihrer Arbeitsdiskette löschen, keinesfalls aber von Ihrer Originaldiskette.

DIR (intern)

(**DIR**ECTORY = Inhaltsverzeichnis) gibt alle Dateinamen mit Dateitypen der aktiven Diskette auf den Bildschirm aus. Der Inhalt einer Diskette im Laufwerk B (falls angeschlossen) kann mit **DIR B:** angezeigt werden.

Die Variante **DIR (Name.Typ)** erlaubt das selektive Betrachten des Diskettenihalts, das bei längeren Verzeichnissen sinnvoll ist, wenn man bestimmte Dateien oder Dateigruppen betrachten möchte. Dabei kann das Zeichen Stern (*) benutzt werden, um "alle" Dateien einer Gruppe anzuzeigen.

Beispiele:

DIR *.DBF	alle Datenbankdateien vom Typ DBF
DIR MENUE.*	alle Dateien mit Namen "menue", unabhängig vom Typ
DIR DB*.*	alle dBASE-II-Systemdateien

DIR [FULL] **(nur CP/M PLUS, intern)**

Diese Variante liefert das komplette Inhaltsverzeichnis der Diskette bei gleichzeitiger Angabe der Größe der gespeicherten Dateien. Der äquivalente CP/M 2.2 Befehl ist STAT *.* (siehe STAT).

ERA (intern)

(**ERA**SE = löschen) löscht die in Anschluß an den Befehl genannte(n) Datei(en) von der Diskette. Dabei wird die Datei physisch nicht sofort gelöscht, sondern nur ihr Name im Disketteninhaltsverzeichnis (Directoy), so daß der Platz auf der Diskette für neue Dateien freigemacht wird. Die gelöschte Datei wird also erst dann physisch überschrieben, wenn eine andere Datei den Platz benötigt. Diese Tatsache machen sich einige käufliche Programme zunutze, mit denen man mit ERA gelöschte Dateien zurückgewinnen kann. Selbstverständlich funktioniert diese Rettung nur, wenn nach dem Löschen nicht schon neue Dateien auf die Diskette geschrieben wurden.

Beispiele:

```
ERA ADRESSEN.DBF
ERA *.NDX
ERA TEST.*
ERA T*.*
ERA *.*
```

FILECOPY (extern, 3 KB)

wird benutzt, um bei Computern mit nur einem Diskettenlaufwerk Dateien (FILEs) von einer Diskette auf eine andere zu kopieren. Zusammen mit dem Befehl wird der Name der zu kopierenden Datei eingegeben. Anschließend fordert das System den Benutzer auf, die Diskette mit der Ursprungsdatei (engl. SOURCE disk) in das Laufwerk zu stecken. Die Datei wird dann in den Hauptspeicher des Rechners kopiert. Dann erscheint die Aufforderung, nun die Zieldiskette (engl. DESTINATION disk) in das Laufwerk einzulegen, damit die Datei aus dem Hauptspeicher übernommen werden kann.

Wenn große Dateien nicht vollständig im Hauptspeicher Platz finden, dann werden sie von CP/M in einzelnen Abschnitten kopiert, d.h. der eben beschriebene Vorgang des Diskettenwechsels wiederholt sich ein oder mehrere Male.

FORMAT (extern, 3 KB)

formatiert eine Diskette für den Gebrauch unter CP/M. Das Formatieren ist ein Schreibvorgang, mit dem auf der Diskette eine Struktur, bestehend aus sogenannten Spuren und Sektoren, erstellt wird, die den Programmen zur Adressierung bei Schreib- und Lesevorgängen dient. Eine formatierte Diskette ist in eine Vielzahl von Blöcken zu je 512 Byte aufgeteilt, denen über die Spuren und Sektoren Adressen zugeordnet sind.

Da die verschiedenen Betriebssysteme unterschiedliche Formatierungen verwenden, werden fabrikneue Disketten stets unformatiert ausgeliefert. Das Formatieren muß daher der erste Arbeitsgang vor Benutzung einer neuen Diskette sein. Aber auch "gebrauchte" Disketten werden gelegentlich neu formatiert, z.B. beim Wechsel auf ein anderes Betriebssystem oder um "defekte" Disketten zu reparieren. Es kommt vor, daß Disketten unbrauchbar werden, weil permanent Schreib- oder Lesefehler auftreten. Durch erneutes Formatieren können solche Disketten oft wieder verwendbar werden, sofern sie nicht physisch beschädigt sind.

Nach Eingabe des Befehls FORMAT wird das Formatierprogramm in den Hauptspeicher geladen, und es erscheint eine Aufforderung des Systems, die zu formatierende Diskette in das Laufwerk A zu stecken. Drücken Sie auf gar keinen Fall die ENTER-Taste, bevor Sie das nicht gemacht haben. Andernfalls formatieren Sie Ihre eigene CP/M-Betriebssystem-Diskette, die damit zerstört wäre. Denn beim Formatieren von beschriebenen Disketten werden sämtliche Daten unwiederbringlich gelöscht.

Auch wenn Ihr Computer über zwei Diskettenlaufwerke verfügt und Sie mit FORMAT B: in Laufwerk B formatieren wollen, sollten Sie in Laufwerk A nie eine beschriebene Diskette stecken lassen. Außerdem sollten Sie Ihre CP/M-Systemdiskette mit einem Schreibschutz versehen, dann ist sie auch vor irrtümlichem Löschen geschützt. (Bei den 3-Zoll-Disketten ist ein Schreibschutz in Form eines kleinen Riegels bereits eingebaut, den Sie nur in die entsprechende Position schieben müssen.)

PIP (extern, 8 KB)

(Peripheral Interchange Program = Programm zum Datenaustausch zwischen Peripheriegeräten). Dieses Dienstprogramm wird zum Kopieren von Disketten-Dateien benutzt. Sein Name ist darauf zurückzuführen, daß Diskettenlaufwerke vom Computer wie Peripheriegeräte behandelt werden.

Im Unterschied zu FILECOPY können mit PIP Dateien innerhalb einer Diskette kopiert werden, wobei dann natürlich Ursprungs- und Zieldatei unterschiedliche Namen haben werden. Bei Computern mit zwei Laufwerken kopiert PIP auch von einem Laufwerk zum anderen.

Zum Kopieren geben Sie das Wort PIP ein und warten, bis das Kopierprogramm in den Hauptspeicher geladen ist. Anschließend meldet sich das Betriebssystem mit einem Stern (*). Dann können Sie die Dateinamen eintippen, wobei zuerst die Ziel- und dann die Ursprungsdatei genannt wird:

(Zieldatei) = (Ursprungsdatei)

Beim Arbeiten mit zwei Diskettenlaufwerken muß vor dem Namen noch die Bezeichnung des Laufwerks angegeben werden, (A: oder B:). Nach dem Kopieren erscheint wieder der Stern, d.h. Sie können sofort weitere Dateien zum Kopieren eingeben. Wenn Sie nur eine Datei oder Dateigruppe zu kopieren haben, kann die gesamte Anweisung auch in einer einzigen Befehlszeile geschrieben werden:

PIP (Zieldatei) = (Ursprungsdatei)

Die einzige Datei, die mit PIP nicht kopiert werden kann, ist die Datei des Betriebssystemkerns ($OSC.SYS). Sie wird mit dem Befehl SYSGEN kopiert.

Beispiele:

PIP MENUE3.CMD = MENUE.CMD

PIP A: = B:TEST.TXT	kopiert die Datei von B: nach A:
PIP A: = B:*.DBF	kopiert alle DBF-Dateien von B: nach A:
PIP B: = A:DB*.*	kopiert alle dBASE-Systemdateien von A: nach B:
PIP A:*.BAK = B:*.DBF	sichert die Datenbankdateien aus B: als Backup-Kopien auf Diskette A
PIP A: = B:*.*	kopiert den gesamten Diskettenінhalt

REN (intern)

(RENAME = umbenennen) dient zum Umbenennen von Dateien und wird meistens im Rahmen der Diskettenverwaltung benutzt, z.B. um nachträglich eine gewisse Ordnung und Systematik in die Namensgebung der Dateien zu bekommen.

In einigen Fällen müssen Dateien umbenannt werden: Wenn Sie z.B. Sicherungskopien mit dem Dateityp BAK angelegt haben und diese zur Benutzung aktivieren wollen, dann muß in den meisten Fällen der ursprüngliche Dateityp wiederhergestellt werden. Dies gilt für alle dBASE-Dateitypen (DBF, NDX, CMD, FRM) und ebenso für das Textverarbeitungssystem WordStar. Im Gegensatz zu allen anderen Dateien läßt WordStar eine Bearbeitung von Backup-Dateien (Typ BAK) aus Sicherheitsgründen nicht zu.

Zum Umbenennen muß zuerst der neue, dann der alte Dateiname genannt werden:

REN (neuer Name.Typ) = (alter Name.Typ)

SHOW (**nur CP/M PLUS**, extern)

(Show = zeigen) zeigt den noch verfügbaren Speicherplatz auf der Diskette an. Der äquivalente CP/M 2.2 Befehl ist STAT.

STAT (**nur CP/M 2.2**, extern, 6 KB)

(STATUS = Belegungszustand der Diskette) zeigt den noch verfügbaren Speicherplatz (in Kilobyte) auf der Diskette an.

Die Variante **STAT *.*** liefert das komplette Inhaltsverzeichnis der Diskette unter Angabe der Größe der gespeicherten Dateien und des noch verfügbaren Speicherplatzes auf der Diskette. Die Werte werden auch hier in Kilobyte angegeben.

Neben FILECOPY und PIP ist STAT das am häufigsten gebrauchte Dienstprogramm, das bei ausreichendem Platz immer auf der Diskette verfügbar sein sollte.

SYSGEN (extern, 2 KB)

(SYSTEM GENERATION = Erstellen einer Systemdiskette) ist ein Kopierprogramm, das ausschließlich zum Kopieren des CP/M-Betriebssystemkerns bestimmt ist. Wie weiter oben erläutert wurde, kann das Betriebssystem nicht mit PIP kopiert werden.

Jede Arbeitsdiskette, die zum Starten des Computers geeignet sein soll, muß das mit SYSGEN kopierte Betriebssystem enthalten. Eine dBASE-Systemdiskette wird z.B. so erstellt:

- Formatieren einer neuen Diskette
- Übertragen des Betriebssystems mit SYSGEN
- Kopieren der dBASE-Dateien mit FILECOPY oder PIP

Die Datei SYSGEN.COM sollte nicht auf die Arbeitsdisketten kopiert werden, damit man nicht in Versuchung kommt, das Betriebssystem "unkontrolliert" zu vervielfältigen. Es ist nämlich besser, den CP/M-Kern immer nur von einer Master-Diskette zu entnehmen, und nicht Kopien von Kopien zu benutzen, deren Herkunft mit der Zeit kaum mehr nachzuvollziehen ist. Auch bei Disketten gibt es - wie bei Audiokassetten - schlechte oder schwache Kopien, die zwar gerade noch funktionieren, von denen man aber keine weiteren brauchbaren Kopien ziehen kann. Bei Text- oder Datendisketten fallen fehlerhafte Stellen schnell auf, aber bei Betriebssystemdisketten können versteckte Fehler wie eine Zeitbombe wirken: Irgendwann stürzt das System ohne ersichtlichen Grund ab.

Die Handhabung von SYSGEN ist bei einem und bei zwei Diskettenlaufwerken praktisch gleich. Das System fordert Sie auf, zuerst die Ursprungsdiskette (Master-Diskette) in das Laufwerk A zu stecken und ENTER zu drücken. Anschließend sollen Sie die Ziel-Diskette in dasselbe oder - bei zwei Laufwerken - in das Laufwerk B schieben. Nach Abschluß des Kopierens werden Sie gefragt, ob weitere Disketten mit der Systemsoftware beschrieben werden sollen.

TYPE (intern)

(schreiben) ermöglicht die Ausgabe eines Dateiinhalts auf den Bildschirm oder Drucker. Der Befehl ist für alle Dateitypen zulässig, wirklich sinnvoll ist er aber nur bei Dateien im ASCII-Code, also bei Dateien, die Text enthalten. Anders codierte Dateien, wie z.B. Datenbankdateien oder COM-Dateien, liefern keine lesbaren Informationen, sondern führen nur zu scheinbar unkontrollierten Bildschirmreaktionen und akustischen Signalen.

Für dBASE-Anwender ist der Befehl TYPE sehr nützlich, um Programm- und Reportdateien (Typ CMD bzw. FRM) auszugeben und insbesondere für Protokollzwecke auszudrucken. Denn dBASE II verfügt selbst über keinen Befehl zum Drucken dieser Dateien. Auch die mit SET ALTERNATE erstellten Protokolle über einen dBASE-Programmablauf können mit TYPE ausgegeben werden.

Das Ein- und Ausschalten des Druckers erfolgt mit der Tastenkombination CTRL-P, bei der von dBASE vorab eingestellten Tastenbelegung auch mit der Kombination CTRL-COPY.

Index

Weitere Fachbücher aus unserem Verlagsprogramm

COMMODORE

Das Commodore 128-Handbuch

1985, 250 Seiten

In diesem Buch finden Sie einen Querschnitt durch alle wichtigen Funktions- und Anwendungsbereiche des Commodore 128. Sie werden mit dem C64/C128-Modus und der Benutzung von CP/M 3.0 vertraut gemacht, erfahren alles über die Grafik- und Soundmöglichkeiten des C128, lernen die Techniken der Speicherverwaltung und das Banking kennen und werden in die Programmierung mit Assemblersprache sowie die Grafikprogrammierung des 80-Zeichen-Bildschirms eingeführt. Ein umfassendes Handbuch, das Sie immer griffbereit haben sollten!

Best.-Nr. MT 809, ISBN: 3-89090-195-9
(sFr 47,80/öS 405,60) **DM 52,—**

BASIC 7.0 auf dem Commodore 128

1985, 250 Seiten

Ganz gleich, ob Sie bereits über Programmierkenntnisse verfügen oder nicht, dieses Buch wird Ihnen helfen, den größtmöglichen Nutzen aus dem leistungsstarken BASIC 7.0 des Commodore 128PC zu ziehen. Sie eignen sich bei der Durcharbeitung dieses Buches alle notwendigen Kenntnisse an, um immer anspruchsvollere Aufgabenstellungen zu bewältigen: Listenverarbeitung, indexsequentielle Dateiverwaltung, Grafikdarstellungen und Sounderzeugung. Ein unentbehrliches Lehrbuch, das sich auch für den geübten Anwender als Nachschlagewerk eignet.

Best.-Nr. MT 808, ISBN: 3-89090-170-0
(sFr 47,80/öS 405,60) **DM 52,—**

C64 — Wunderland der Grafik

Juli 1985, 236 Seiten inklusive Beispielsdiskette

Dieses Buch zeigt eine Vielzahl sehr interessanter Lösungen, um die grafischen Möglichkeiten des Commodore 64 optimal zu nutzen. Als Krönung enthält es ein zuschaltbares Assemblerprogramm, das umfangreiche grafische und einige neue BASIC-Befehle anbietet. Im zweiten Teil des Buches wird eine Möglichkeit gezeigt, wie man bis zu 70 verschiedene Farben erzeugen kann. Viele Beispielprogramme begleiten die Reise durch das Wunderland der Grafik.

Best.-Nr. MT 756, ISBN: 3-89090-130-1
(sFr. 45,10/öS 382,20) **DM 49,—**

Das C64-Profihandbuch

Juli 1985, 410 Seiten

Ein Buch, das alle wichtigen Informationen für professionelle Anwendungen mit dem C64 enthält. Mit allgemeinen Algorithmen, die auch auf andere Rechner übertragbar sind, und vielen Utilities, getrennt nach BASIC- und Maschinenprogrammen. Besonders nützlich: erweiterte PEEK- und POKE-Funktionen.

Best.-Nr. MT 749, ISBN: 3-89090-110-7
(sFr. 47,80/öS 405,60) **DM 52,—**

Programmieren unter CP/M mit dem C64

Juni 1985, 290 Seiten

Wenn Sie wissen wollen, wie das Betriebssystem CP/M 2.2 auf dem C64 implementiert ist, außerdem einiges über Turbo-Pascal, Nevada-Fortran, MBasic-80 erfahren wollen, dann ist dieses Buch genau richtig für Sie! Mit Schaltplänen zur eigenen Fertigung des CP/M-Moduls. Für eingefleischte C64-Profis.

Best.-Nr. 751, ISBN: 3-89090-091-7
(sFr. 47,80/öS 405,60) **DM 52,—**

C64 — Programmieren in Maschinensprache

August 1985, 327 Seiten inklusive Beispieldiskette

In diesem Buch finden Sie über 100 Beispiele zur Assembler-Programmierung mit viel Kommentar und Hintergrundinformationen: Das Schreiben von Maschinenprogrammen · Rechnen und Texten mit vorhandenen Routinen · Bedienung von Drucker und Floppy · wie man BASIC- und Maschinenprogramme verknüpft · Erstellen von eigenen Befehlen in Modulform. Für Profis!

Best.-Nr. MT 830, ISBN: 3-89090-168-9
(sFr. 47,80/öS 405,60) **DM 52,—**

Einführungskurs: Commodore 64

Mai 1984, 276 Seiten

Die Programmiersprache Basic · Einsatzgebiete des Commodore 64-Basic: Grafik, Musik, Dateiverwaltung · mit vielen Beispielprogrammen, häufig benötigten Tabellen und nützlichen Tips · für Einsteiger und Fortgeschrittene.

Best.-Nr. MT 685, ISBN: 3-89090-017-8
(Sfr. 35,—/öS 296,40) **DM 38,—**

Commodore 64 — leicht verständlich

Juni 1984, 154 Seiten

Informationen für den Computer-Neuling · Installation und Inbetriebnahme · Programmieren in Basic · Grafik und Töne · Auswahl von Hardware und Zubehör · Software für Ihren Computer · die ideale Einführung in das Arbeiten mit Ihrem Commodore 64.

Best.-Nr. MT 700, ISBN: 3-89090-022-4
(Sfr. 27,50/öS 232,40) **DM 29,80**

Ihr Heimcomputer Commodore 64

August 1984, 296 Seiten

Alles Wissenswerte im Umgang mit dem Commodore 64 · Planung, Kauf und Inbetriebnahme der Anlage · Einsatz fertig gekaufter oder selbst erstellter Programme · Schwächen und Stärken der altbewährten und neuesten Programmiersprachen · die gängigsten Software-Angebote für jeden Einsteiger.

Best.-Nr. MT 701, ISBN: 3-89090-044-5
(Sfr. 35,—/öS 296,40) **DM 38,—**

Das Commodore 64-LOGO-Arbeitsbuch

September 1984, 225 Seiten

Kinder lernen auf dem Commodore 64 mit der Schildkröte als Lehrer: Bilder malen · Grafikeffekte erzeugen · Wörter verarbeiten · Prozeduren und Variablen · Umgang mit Begriffen wie: Längenmaß, Winkel, Dreieck, Quadrat.

Best.-Nr. MT 720, ISBN: 3-89090-063-1
(Sfr. 31,30/öS 265,20) **DM 34,—**

Die angegebenen Preise sind Ladenpreise

Sie erhalten Markt&Technik-Bücher bei Ihrem Buchhändler

Markt & Technik Verlag AG Unternehmensbereich Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München

Weitere Fachbücher aus unserem Verlagsprogramm

35 ausgesuchte Spiele für Ihren Commodore 64
September 1984, 141 Seiten
Programmieren Sie selbst 35 faszinierende Spiele · geschrieben in Commodore 64-BASIC · mit Farbe, Grafiken und Ton · Vorschläge zur Programmabwandlung · für kreative Computerfans, die Ihre Programmierkenntnisse vertiefen wollen!
Best.-Nr. MT 774, ISBN: 3-89090-064-X
(Sfr. 23,—/öS 193,40) **DM 24,80**

Lehrspielzeug Computer: C 64/VC-20
Juli 1984, 139 Seiten
Speziell für Kinder entwickelt führt dieses Buch spielerisch in die Basic-Welt des Commodore 64/VC-20 ein · mit vielen lehrreichen Spielprogrammen und Grafikmöglichkeiten · kleinere Kinder benötigen die Hilfe ihrer sachkundigen Eltern.
Best.-Nr. MT 695, ISBN: 3-89090-011-9
(Sfr. 23,—/öS 193,40) **DM 24,80**

Basic mit dem Commodore 64
April 1984, 320 Seiten
Ein Basic-Lehrbuch für den jugendlichen Anfänger · übersichtlich gegliederte Lernprogramme · Alles über INPUT-GOTO · Let-Befehle · Editorfunktionen · POKE-Befehle für die Grafik · geeignet auch als Leitfaden für Lehrer und Eltern.
Best.-Nr. MT 657, ISBN: 3-922120-91-1
(Sfr. 44,20/öS 374,40) **DM 48,—**

Computerspiele & Wissenswertes
Februar 1984, 156 Seiten
Eine Sammlung von interessanten und nützlichen Maschinenprogrammen · schnelle binäre Arithmetik · Basic-Erweiterungen · mit unterstützendem Assembler-Listing · für den fortgeschrittenen Programmierer.
Best.-Nr. MT 601, ISBN: 3-922120-62-8
(Sfr. 27,50/öS 232,40) **DM 29,80**
Best.-Nr. MT 602 (Beispiele auf Diskette)
(Sfr. 38,—/öS 342,—) **DM 38,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Das große Spielebuch — Commodore 64
Februar 1984, 141 Seiten
46 Spielprogramme · Wissenswertes über Programmiertechnik · praxisnahe Hinweise zur Grafikherstellung · alles über Joystick- und Paddleansteuerung · das Spielebuch mit Lerneffekt.
Best.-Nr. MT 603, ISBN: 3-922120-63-6
(Sfr. 27,50/öS 232,40) **DM 29,80**
Best.-Nr. MT 604 (Beispiele auf Diskette)
(Sfr. 38,—/öS 342,—) **DM 38,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Spiele für den Commodore 64
November 1984, 196 Seiten
Bewährte alte und raffinierte neue Spiele für Ihren Commodore 64 · klar und übersichtlich gegliederte Programme im Commodore-BASIC · Sie lernen: wie man Unterprogramme einsetzt · eine Tabelle aufbauen und verarbeiten · Programme testen · mit vielen Programmiertricks · für Anfänger.
Best.-Nr. MT 792, ISBN: 3-89090-074-7
(Sfr. 23,—/öS 193,40) **DM 24,80**
Best.-Nr. MT 795 (Beispiele auf Diskette)
(Sfr. 38,—/öS 342,—) **DM 38,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Computer für Kinder — Ausgabe Commodore 64
1984, 112 Seiten
Ein Buch für Kinder und ihre Lehrer · ideal für die erste Begegnung mit Computern, ihren Eigenwilligkeiten und ihren unerschöpflichen Möglichkeiten · leichtverständliche Erläuterungen rund um den Commodore 64 · alle Programmbeispiele in BASIC.
Best.-Nr. PW 709, ISBN: 3-921803-41-1
(Sfr. 27,50/öS 232,40) **DM 29,80**

Grafik & Musik auf dem Commodore 64
Oktober 1984, 336 Seiten
68 gut strukturierte und kommentierte Beispielprogramme zur Erzeugung von Sprites und Klangeffekten · Sprite-Tricks · Zeichengrafik · hochauflösende Grafik · Musik nach Noten · spezielle Klangeffekte · Ton und Grafik · für fortgeschrittene Anfänger, die alle Möglichkeiten des C64 ausnutzen wollen.
Best.-Nr. MT 743, ISBN: 3-89090-033-X
(Sfr. 35,—/öS 296,40) **DM 38,—**

Mehr als 32 Basic-Programme für den Commodore 64
März 1984, 279 Seiten
Programme speziell für den Commodore 64 · umfassende praktische Anwendungen · jede Menge Lehr- und Lernhilfen · super Spiele · für Basic-Neulinge und Experten.
Best.-Nr. MT 613, ISBN: 3-922120-66-0
(Sfr. 45,10/öS 382,20) **DM 49,—**
Best.-Nr. MT 614 (Beispiele auf Diskette)
(Sfr. 48,—/öS 432,—) **DM 48,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Commodore 64 Listings — Band 1: Spiele
Oktober 1984, 199 Seiten
Mit ausführlicher Dokumentation · Spielanleitung · Variablen für die Änderung der Spiele · vollständige Listings für: Bürger Joe · Nibbler · Zingel Zangel · Universe · Würfelpoker · Maze-Mission · der magische Kreis · Todeskommando Atlantik · Enterprise.
Best.-Nr. MT 748, ISBN: 3-89090-068-2
(Sfr. 23,—/öS 193,40) **DM 24,80**
Best.-Nr. MT 804 (Beispiele auf Diskette)
(Sfr. 38,—/öS 342,—) **DM 38,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Commodore 64 Listings
Band 2: Dateiverwaltung · Schule · Hobby
Oktober 1984, 179 Seiten
Ein Buch mit Programmen für die ganze Familie · DATAVE — Eine Dateiverwaltung · mathematische Funktionen · Konjugation und Deklination in Latein · Regressionsanalyse · Bundesligatabelle.
Best.-Nr. MT 766, ISBN: 3-89090-071-2
(Sfr. 23,—/öS 193,40) **DM 24,80**

Commodore 64 — Multiplan
Multiplan jetzt auch für den Commodore 64 · der volle Leistungsumfang der 16-Bit-Version · Einführung in die Arbeitsweise von Tabellenkalkulationsprogrammen · praxisnahe Beispiele · Beschreibung aller Befehle und Funktionen · nicht nur für Anfänger.
Best.-Nr. MT 655, ISBN: 3-922120-89-X
(sFr. 44,20/öS 374,40) **DM 48,—**

Die angegebenen Preise sind Ladenpreise

Sie erhalten Markt & Technik-Bücher bei Ihrem Buchhändler

Markt & Technik Verlag AG Unternehmensbereich Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München

Weitere Fachbücher aus unserem Verlagsprogramm

Der Atari als Musikbox
November 1984, 196 Seiten
Eine musikalische Einführung in die Computerprogrammierung · was Sie über Resonanz und Harmonie wissen müssen · Musikprogramme in BASIC für zwei, drei und vier Stimmen sowie für einen Kanon · besondere Geräuscheffekte · eine Lieder-Bibliothek · für Anfänger.
Best.-Nr. MT 797, ISBN: 3-89090-075-5
(Sfr. 27,50/öS 232,40) **DM 29,80**

Ausgesuchte Atari-Programme mit Listings
Oktober 1984, 171 Seiten
Mehr als 25 Programme — vom alltäglichen Kleinkram bis zu geschäftlichen Anwendungen und Dienstprogrammen · Girokontoführung · Adressenverzeichnis · Joggingkontrolle · für Anfänger, die den Umgang mit dem Computer und die Grundbegriffe des Programmierens lernen wollen.
Best.-Nr. MT 759, ISBN: 3-89090-070-4
(Sfr. 29,50/öS 249,60) **DM 32,—**

SCHNEIDER-FAMILIE

Der CPC 464 für Ein- und Umsteiger
Februar 1985, ca. 260 Seiten
Eine praxisorientierte Spiel- und Arbeitshilfe für den Schneider CPC 464 · BASIC · Grafik · Sound · Tastaturanwendung · Kassettenrecordereinsatz · alle Befehle kompakt und systematisch dargestellt · modular aufgebaute Beispielprogramme auch zur Textverarbeitung und Datenverwaltung · der ideale Grundstock für Ihre CPC 464-Programmbibliothek!
Best.-Nr. MT 801, ISBN: 3-89090-090-9
(Sfr. 42,30/öS 358,80) **DM 46,—**

CPC 464 — Programmieren in Maschinensprache
Juli 1985, 276 Seiten
Vom Speicheraufbau bis hin zum Z80-Befehlssatz wird der fortgeschrittene BASIC-Programmierer in das Innenleben seines Schneider-Computers eingeweiht. Wichtige ROM-Routinen und ausgewählte Werkzeuge wie Disassembler und Monitor werden als nützliche Utilities für die eigene Programmerstellung mitgeliefert. **Alle Beispiele auf Kassette erhältlich.**
Best.-Nr. MT 829, ISBN: 3-89090-166-2
(sFr. 42,30/öS 358,80) **DM 46,—**
Best.-Nr. MT 833 (Kassette) **DM 19,90***
(sFr. 19,90/öS 179,10)
*** inkl. MwSt. Unverbindliche Preisempfehlung**

Das »ROM« im Schneider CPC 464/664
September 1985, ca. 500 Seiten
Ausführliche Hardware-Beschreibung: Prozessor Z80A, Videocontroller 6845 CRTC, Gate Array 20 RA 043, Sound Generator AY-3-8g12, I/C-Baustein 8255 PIO, Expansion-Port. Die ROMs: Speicheraufteilung, Interrupt-Verwaltung, Datenformate, Erweiterungs- und Änderungsmöglichkeiten. Das ROM-Listing: Betriebssystem, BASIC-Interpreter.
Best.-Nr. MT 711, ISBN: 3-89090134-4
(sFr. 58,90/öS 499,20) **DM 64,—**

CPC BASIC-Kurs
Juli 1985, ca. 250 Seiten
Dieses Buch soll den Einstieg in die Bedienung und Programmierung der Schneider-Familie (464, 664, 6128) erleichtern und richtet sich daher an alle Anwender, für die das Gebiet »Computer« noch Neuland ist. Ein Buch, das für jeden Schneider CPC-Besitzer interessant ist.
Best.-Nr. MT 828, ISBN: 3-89090-167-0
(sFr. 42,30/öS 358,80) **DM 46,—**

SINCLAIR

Maschinencode-Programme für den ZX Spectrum
Juni 1984, 204 Seiten
Nützliche Maschinencode-Programme mit Ihrem ZX Spectrum · Sortierung von Fließkommazahlen · Übernahme von Parametern direkt von einem Basic-Programm · Flußdiagramme · für Profis und solche, die es werden wollen.
Best.-Nr. MT 702, ISBN: 3-89090-023-2
(sFr. 29,50/öS 249,60) **DM 32,—**

ZX-Spectrum Abenteuerspiele
September 1984, 208 Seiten
Die Entstehungsgeschichte der Abenteuerspiele mit repräsentativen Beispielen für jede »Epoche« · Ein Programm speziell für Ihren ZX-Spectrum: »Das Auge des Sternenkriegers«, ein Grafik-Abenteuerspiel, das Sie in Atem hält!
Best.-Nr. MT 712, ISBN: 3-89090-047-X
(Sfr. 27,50/öS 232,40) **DM 29,80**

Astronomie-Programme für den ZX-Spectrum
September 1984, 268 Seiten
Eine phantastische Reise in die Welt des Kosmos mit Ihrem ZX-Spectrum: Der Julianische Kalender · Die Mondphasen · Eigene Satelliten starten · Keplers Umlaufbahnen · Die Umlaufbahn Plutos · Interessant nicht nur für Hobby-Astronome.
Best.-Nr. MT 732, ISBN: 3-89090-048-8
(sFr. 27,50/öS 232,40) **DM 29,80**

Schnelles Rechnen mit dem ZX81
Oktober 1984, 276 Seiten
Das Betriebssystem · der BASIC-Interpreter · Gleitkomma-Macro-Befehle zur Verkürzung der Rechenzeiten · alle Programmbeispiele sind lauffähig auf dem ZX81 mit dem 1K-RAM-Speicher, ein 16K-Speicher vereinfacht die Programmentwicklung.
Best.-Nr. MT 706, ISBN: 3-89090-073-9
(Sfr. 27,50/öS 232,40) **DM 29,80**

ZX-Spectrum Hardware
Januar 1985, 147 Seiten
Dieses Buch vermittelt Ihnen ein fundiertes Basiswissen über Aufbau und Entwicklung eigener Hardware · Ausführliche Beschreibung der einzelnen ICs mit Abbildungen und 2-System-Schaltplänen · Anschluß einer PIO-Ansteuerung von Dezimalanzeigen · Leuchtdioden · Relais · DIL-Schalter · Eine akkugepufferte Hardwareuhr mit vierstelliger Anzeige · Soundgenerator mit drei Kanälen.
Best.-Nr. MT 737, ISBN: 3-89090-092-5
(Sfr. 27,50/öS 232,40) **DM 29,80**

Die angegebenen Preise sind Ladenpreise

Sie erhalten Markt&Technik-Bücher bei Ihrem Buchhändler

Markt & Technik Verlag AG Unternehmensbereich Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München

Weitere Fachbücher aus unserem Verlagsprogramm

TI 99/4A

21 LISTige Programme für den TI-99/4A
November 1984, 224 Seiten
Umfangreiche Spiele aller Art für den TI-99/4A · nützliche Utilities · Adressenverwaltung · Vokabel-Programm · für manche Programme ist das Extended-BASIC-Modul, die Speichererweiterung (32 K), ein Disketten-Laufwerk oder Joysticks erforderlich!
Best.-Nr. MT 754, ISBN: 3-89090-065-8
(Sfr. 23,—/öS 193,40) **DM 24,80**

PROGRAMMIERSPRACHEN

Basic für Einsteiger
Juni 1984, 239 Seiten
Ein Arbeitsbuch für den absoluten Anfänger · Basic-Anweisungen Schritt für Schritt erklärt und anhand von einfachen Beispielen erläutert · das beliebte Arbeitsmittel für Lehrkräfte und für den interessierten Computerfan.
Best.-Nr. MT 680, ISBN 3-89090-024-0
(Sfr. 29,50/öS 249,60) **DM 32,—**

Basic-Dialekte im Vergleich
1983, 105 Seiten
Konvertierung von Apple-, Commodore- und TRS-80-Programmen · Grundlagen der jeweiligen Betriebssysteme · Untersuchung verschiedener Basic-Dialekte · alphabetische Auflistung aller Befehle für die verschiedenen Anpassungsrichtungen.
Best.-Nr. MT 564, ISBN 3-922120-53-9
(Sfr. 29,50/öS 249,60) **DM 32,—**

Basic-Programmier-Handbuch
März 1984, 506 Seiten
Grundlagen · Basic und seine Dialekte · geschäftliche und wissenschaftliche Anwendungen · Spiele · Lernprogramme · alles über Programmsteuerung · Schleifen und Verzweigungen · Amortisationsprogramm · numerische Funktionen · Stringfunktionen · Variationen mit PEEK und POKE · der Zauberwürfel.
Best.-Nr. MT 658, ISBN 3-922120-92-X
(Sfr. 71,80/öS 608,40) **DM 78,—**

77 Basic-Programme
1980, 208 Seiten
Eine nützliche Sammlung von Programmlösungen der häufigsten Fragestellungen im Kapitalwesen, Statistik und Mathematik sowie im Alltag · 77 Basic-Programme für den CBM-Computer sorgfältig getestet und vollständig dokumentiert · für Anfänger.
Best.-Nr. PW 256, ISBN 3-9221803-06-3
(Sfr. 35,90/öS 304,20) **DM 39,—**

BASIC-Grundkurs mit dem Commodore 64
März 1985, 377 Seiten
Ein praxisorientierter Leitfaden für die Programmierung in BASIC · die Besonderheiten des Commodore-BASIC · umfangreiche Befehlsübersicht · Einführung in die aktuelle Thematik der Datenkommunikation: BTX oder MailBox · das ideale Buch für Jungprogrammierer, die ihre Anfangsschwierigkeiten überwinden wollen!
Best.-Nr. MT 633, ISBN 3-89090-045-3
(Sfr. 40,50/öS 343,20) **DM 44,—**

MSX Basic
April 1985, 236 Seiten
Alles über den neuen Heimcomputerstandard MSX: zusätzlich zum »normalen« BASIC können mit insgesamt mehr als 150 Befehlen und Funktionen Grafiken erstellt, Töne erzeugt, Melodien komponiert und ganze Spielhandlungen programmiert werden · 32 Sprites garantieren abwechslungsreiche Action-Spiele · die Hardware des MSX-Systems · nützliche Hinweise zur Dateibehandlung · das MSX-BASIC anhand der Entwicklung eines Spielszenarios mühelos lernen · drei vollständige Spiele: Der eisige Planet, Autorennen und Bilder entwerfen · mit ausführlicher Befehlsübersicht · für Anfänger!
Best.-Nr. MT 805, ISBN 3-89090-107-7
(Sfr. 40,50/öS 343,20) **DM 44,—**
Best.-Nr. MT 825 (Beispiele auf Kassette)
(Sfr. 19,80/öS 178,20) **DM 19,80***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Das Commodore 64-LOGO-Arbeitsbuch
September 1984, 225 Seiten
Kinder lernen auf dem Commodore 64 mit der Schildkröte als Lehrer: Bilder malen · Grafikeffekte erzeugen · Wörter verarbeiten · Prozeduren und Variablen · Umgang mit Begriffen wie: Längenmaß, Winkel, Dreieck, Quadrat.
Best.-Nr. MT 720, ISBN 3-89090-063-1
(Sfr. 31,30/öS 265,20) **DM 34,—**

LOGO — Grafik, Sprache, Mathematik
März 1984, 257 Seiten
Eine Einführung in LOGO als Lehr- und Lernsprache unter besonderer Berücksichtigung des Apple-LOGO · Grafikprozeduren · Zeichenkettenmanipulationen · Probleme der Rekursivität · Sprachbildung und Sprachforschung · Grundlagen der Arithmetik · mit umfassendem Glossar.
Best.-Nr. MT 648, ISBN 3-922120-60-1
(Sfr. 38,60/öS 327,60) **DM 42,—**

ALLGEMEININTERESSE

Computerchinesisch für Einsteiger
Juli 1984, 107 Seiten
Ein praxisnahes Lexikon, das Personal Computer-Benutzern und solchen, die es werden wollen, das Lesen von Fachzeitschriften, Büchern, Bedienungsanleitungen und Datenblättern erleichtert · über 1000 häufig benötigte Fachbegriffe klar und verständlich erläutert · mit zahlreichen Abbildungen.
Best.-Nr. MT 690, ISBN 3-89090-019-4
(Sfr. 25,90/öS 218,40) **DM 28,—**

Im Land der Abenteuer
Juni 1984, 146 Seiten
Ein Lösungsbuch für zahlreiche Computerspiele · Tod in der Karibik · Transsylvanien · Unternehmen Asteroid · Das geheimnisvolle Haus · Zauberer und Prinzessin · Das goldene Vlies · Zeitzone · Der dunkle Kristall.
Best.-Nr. MT 699, ISBN 3-89090-021-6
(Sfr. 27,50/öS 232,40) **DM 29,80**

Mikrocomputer Grundwissen
1978, 304 Seiten
Eine allgemeinverständliche Einführung in die Mikrocomputer-Technik · optimal als Einstieg für Elektronik-Laien.
Best.-Nr. PW 156, ISBN 3-921803-02-0
(Sfr. 33,10/öS 280,80) **DM 36,—**

Die angegebenen Preise sind Ladenpreise

Sie erhalten Markt&Technik-Bücher bei Ihrem Buchhändler

Markt & Technik Verlag AG Unternehmensbereich Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München

DR. PETER ALBRECHT
geboren 1944, studierte Nachrichtentechnik an der Technischen Universität München und promovierte dort mit einer Arbeit über Mikrowellenhalbleiter zum Dr.-Ing. Danach war er 12 Jahre in verschiedenen Funktionen bei der NCR GmbH tätig.
Tätigkeitsschwerpunkte: Hardware- und Software-Entwicklung sowie Produktmanagement von Personal Computern, Unternehmensplanung, Aufbau und Leitung eines Rechenzentrums, Integration des Personal Computers in den Bereich der etablierten Datenverarbeitung, dezentralisierte Verarbeitung bei zentraler Datenkontrolle.
Heute ist er als Unternehmensberater in Augsburg tätig.

dBASE II für den Schneider CPC

Datenbanksysteme sind heute ein fester Bestandteil der Mikrocomputeranwendungen. Das bislang populärste System dieser Art ist sicherlich dBASE II, das als »relationales Datenbanksystem« in der Klasse der 8-Bit-Rechner Maßstäbe setzte.
Zur Beliebtheit von dBASE tragen drei Eigenschaften bei, die es aus der Vielzahl konkurrierender Produkte herausheben:

- die Einfachheit und Schnelligkeit, mit der Dateistrukturen definiert, benutzt und geändert werden können;
- die hohe Flexibilität im Datenzugriff ohne starre Zugriffspfade;
- die integrierte Kommandosprache, die über die Befehle zur reinen Datenmanipulation hinausgeht und eine komplette Anwendungsprogrammierung zuläßt.

Insbesondere die letztgenannte Eigenschaft ist für den geübten Benutzer von größter Bedeutung. Ihm wird die Möglichkeit geboten, auf konventionelle Programmiersprachen wie BASIC vollständig zu verzichten und eine echte Datenbanksprache auf hohem Niveau zu benutzen, die jenen an Mächtigkeit weit überlegen ist.
Das vorliegende Buch gibt nach einer kurzen Einführung in den Komplex »Datenbanken« eine Anleitung für den praktischen Umgang mit dBASE II. Schon nach Beherrschung weniger Befehle ist der Anwender in der Lage, Dateien zu erstellen, mit Informationen zu laden und auszuwerten. Dabei hilft ihm ein integrierter Reportgenerator, der im Dialog mit dem Benutzer Berichte gestaltet und in Tabellenform ausdruckt.
Im Unterschied zu dem schon früher erschienenen Buch »Das Datenbanksystem dBASE II« (MT 740) geht dieses speziell auf die dBASE-Version für die Schneider-CPC-Computer mit dem Betriebssystem CP/M ein.
Für die Schneider-CPC-Computer sind unter anderem erschienen:
WordStar für den Schneider CPC (MT 779)
MULTIPLAN für den Schneider CPC (MT 835)

ISB N 3-89090-188-3

4 001057 901889

Markt&Technik
Verlag Aktiengesellschaft

DM 49,-
sFr. 45,10
öS 382,20

Albrecht

dBASE II für den Schneider CPC

M&T

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/