dBase-Dateien – Byte für Byte

Bekanntlich erzeugt dBase standardmäßig keine ASCII-Dateien, die von anderen Programmen einfach importiert oder weiterverarbeitet werden könnten, sondern verwendet ein eigenes Dateiformat. Zwar können viele Standardprogramme Dateien des dBase-Formats lesen, doch vor allem Programmierern anderer Sprachen kann eine genaue Kenntnis der Struktur von dBase-Dateien nützlich sein – und sei es nur, um die formatierten Daten weiterzuverarbeiten. Ein entsprechendes Listing in Quick Basic zeigt, wie es gemacht wird.

Dank seiner überragenden Marktstellung ist das dBase-Format neben dem ASCII-Format das am weitesten verbreitete. Natürlich ist es für dBase-Programmierer nicht notwendig, sich mit den einzelnen Bytes einer dBase-Datei herumzuschlagen, da dies dBase selbständig erledigt. Doch versuchen Sie einmal, aus einer defekten Bestandsdatei zu retten, was noch zu retten ist. Oder Sie beabsichtigen, auf die unter dBase gespeicherten Informationen mit Hilfe eines (schnelleren) Programms zuzugreifen, das Sie beispielsweise in Assembler, in C oder in (Quick) Basic schreiben möchten, weil Sie komplizierte grafische Auswertungen oder mathematische Berechnungen durchführen oder die Daten in ein Fremdformat konvertieren wollen, das von dBase nicht unterstützt wird. Dazu benötigen Sie genaue Angaben darüber, auf welche Weise dBase-Daten gespeichert werden.

Dieser Artikel informiert im ersten Teil über die Anatomie von dBase-III-Plus-/dBase-IV-Dateien. Im zweiten Teil (Listing) wird anhand eines in Quick Basic geschriebenen Beispielprogramms gezeigt, wie Sie mit Hilfe fremder Programmiersprachen auf die Daten einer dBase-Datei zugreifen.

Kenntnisse über die Anatomie von dBase-Bestandsdateien sind für den Fremdprogrammierer unerläßlich

Damit die dBase-Datei – wie üblich – mit einigen Daten versehen ist, soll von folgender Beispieldatei ausgegangen werden, die gemäß der sequentiellen Speicherung mit zwei Datensätzen ausgestattet ist:

```
Titel des Films: »Ninotschka«
Regisseur: »Lubitsch«
Wie oft gesehen?: »8«
Wann zuletzt gesehen?: »11.12.89«
Noch einmal sehen (J/N) ? : »T«
Bemerkung: »Greta Garbo als Ninotschka!«

Titel des Films: »Casablanca«
Regisseur: »Curtiz«
Wie oft gesehen?: »12«
Wann zuletzt gesehen? : »12.12.89«
Noch einmal sehen (J/N) ? : »F«
Bemerkung:
```

Dabei steht »T« beziehungsweise »F« bei »noch einmal sehen« für »wahr« (»(T)rue«) oder »ja« respektive »falsch« (»(F)alse«) oder »nein«.

Zum Dateiaufbau noch einige terminologische Bemerkungen: Zusammenhängende Informationsblöcke, wie sie jeweils durch einen der vorhergehenden Absätze dargestellt werden, heißen Datensätze. Ein Datensatz in diesem Beispiel wäre also: »Ninotschka, Lubitsch, 8, 11.12.1989, T, Greta Garbo als Ninotschka!«. Die gleichartigen Informationseinheiten innerhalb der Datensätze werden in Datensatzfeldern gespeichert: Das Datensatzfeld »Regisseur« enthält also in dem einen Datensatz den Eintrag »Lubitsch«, im anderen den Eintrag »Curtiz«. Mehrere Datensätze bilden schließlich eine Dateiverwaltung.

```
E:\>debug film.dbf
-L 0
-D 0 13F
2721:0000  8B 5A 07 09 02 00 00 00-E1 00 2F 00 00 00 00 00   .Z........./.....
2721:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2721:0020  54 49 54 45 4C 00 00 00-00 00 00 43 00 00 00 00   TITEL......C....
2721:0030  0F 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2721:0040  52 45 47 49 53 53 45 55-52 00 00 43 00 00 00 00   REGISSEUR..C....
2721:0050  0A 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2721:0060  57 49 45 4F 46 54 47 45-53 00 00 4E 00 00 00 00   WIEOFTGES..N....
2721:0070  02 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2721:0080  57 41 4E 4E 5A 55 4C 47-45 53 00 44 00 00 00 00   WANNZULGES.D....
2721:0090  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2721:00A0  4E 4F 43 48 45 49 4E 4D-41 4C 00 4C 00 00 00 00   NOCHEINMAL.L....
2721:00B0  01 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2721:00C0  42 45 4D 45 52 4B 55 4E-47 00 00 4D 00 00 00 00   BEMERKUNG..M....
2721:00D0  0A 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2721:00E0  0D 20 4E 69 6E 6F 74 73-63 68 6B 61 20 20 20 20   . Ninotschka
2721:00F0  20 4C 75 72 69 74 73 63-68 20 20 20 38 31 39 38    Luritsch   8198
2721:0100  39 31 32 31 31 54 30 30-30 30 30 30 30 30 30 31   91211T0000000001
2721:0110  20 43 61 73 61 62 6C 61-6E 63 61 20 20 20 20 20    Casablanca
2721:0120  43 75 72 74 69 7A 20 20-20 20 31 32 31 39 38 39   Curtiz    121989
2721:0130  31 32 31 32 46 20 20 20-20 20 20 20 20 20 20 1A   1212F          .
-
```

Bild 1. Der MS-DOS-Debugger zeigt beispielhaft den Dateivorspann einer dBase-Datei

In dBase legen Sie nun eine DBF-Datei an, die sich für die Aufnahme der angegebenen Datensätze eignet, indem Sie aus dem Regiezentrum (dBase IV) beziehungsweise Assistenten (dBase III Plus) oder mit Hilfe des Create-Befehls zunächst die Struktur der Bestandsdatei bestimmen. Sie legen dabei fest, welche Felder die Bestandsdatei enthalten soll, welche Art von Informationen in den Feldern gespeichert wird und wie lang die Felder werden, die die Daten aufnehmen. Die Beispieldatei soll die Struktur aus Tabelle 1 haben.

Nachdem die Struktur der Datei festgelegt worden ist, kann im Assistenten, im Regiezentrum oder mit Hilfe des Append-Befehls mit der Eingabe der Informationen begonnen werden. Angenommen also, Sie hätten unter dBase eine Bestandsdatei (sie heiße »film.dbf«) mit der angegebenen Struktur angelegt und die genannten Daten eingegeben. Wie werden die Informationen von dBase intern gespeichert? Eine Möglichkeit, sich eine Datei byteweise anzeigen zu lassen, bietet das MS-DOS-Dienstprogramm »Debug«. Hexadezimaleditoren, wie sie zum Beispiel mit den Norton Utilities oder PC-Tools geliefert werden, eignen sich besser, wenn Sie es mit großen Dateien zu tun haben – und sie bieten außerdem komfortablere Bearbeitungsmöglichkeiten als das erwähnte »Debug«. Bild 1 zeigt Aufruf und Anzeige von Debug. Der Aufruf gestaltet sich noch einfach:

```
debug film.dbf
```

Hinter der Eingabeaufforderung des Debuggers (dargestellt durch einen Bindestrich) sorgt die Anweisung

```
L 0
```

– also »Load« – dafür, daß die Datei »film.dbf« an die Adresse 0000hex (relativ zu einer durch das CS-Register festgelegten Segmentadresse) im Arbeitsspeicher geladen wird. Die zweite Anweisung

```
D 0 13F
```

– also »Dump« – veranlaßt das Debug-Programm, den Inhalt des Arbeitsspeichers ab der Adresse 0000hex bis zur Adresse 013Fhex (319) Byte für Byte auszugeben. Demnach gibt Ihnen Debug von der Adresse 0 bis zur Adresse 13Fhex 320 Byte aus – genau so viel wie nötig, um die vollständige Datei »film.dbf« anzuzeigen, die eine Größe von 320 Byte hat, wie Sie mit dem Dir-Befehl überprüfen können.

Die linke Spalte des Ausgabebildschirms mit den durch Doppelpunkt getrennten Zahlenblöcken gibt Arbeitsspeicheradressen an – diese Spalte braucht Sie nicht zu interessieren. Der mittlere Block gibt die in den Arbeitsspeicher geladene Datei zeilenweise Byte für Byte aus, wobei jedes Ziffernpaar ein Byte im Hexadezimalformat angibt.

Wenn ein Byte für ein druckbares ASCII-Zeichen steht, wird dieses Zeichen im rechten Block der Debug-Ausgabe dargestellt. Nicht druckbare Zeichen werden durch einen Punkt repräsentiert.

Feld	Feldname	Typ	Länge	Dezimalstellen
1	Titel	Zeichen	15	0
2	Regisseur	Zeichen	10	0
3	Wieoftges	numerisch	2	0
4	Wannzulges	Datum	8	0
5	Nocheinmal	logisch	1	0
6	Bemerkung	Memo	10	0

Tabelle 1. Die Struktur der Beispieldatei in dBase

Byte-Position	Inhalt oder Bedeutung
A. Dateivorspann	
	Allgemeine Datei-Informationen
1. Byte	dBase-Version/Memodatei vorhanden?
2.- 4. Byte	Datum der letzten Änderung (Jahr, Monat, Tag)
5.- 8. Byte	Anzahl der Datensätze
9.-10. Byte	Länge des. Dateivorspanns
11.-12. Byte	Länge eines Datensatzes (inkl. Löschkennzeichen)
13.-14. Byte	reserviert
15. Byte	Transaktion abgeschlossen? (nur dBase IV)
16. Byte	Datei verschlüsselt? (nur dBase IV)
17.-28. Byte	reserviert
29. Byte	Mdx- Indexdatei vorhanden? (nur dBase IV)
20.-32. Byte	reserviert
	Feldinformationen (je Feld ein Block a 32 Byte)
1.-10. Byte	Feldname (aufgefüllt durch 00hex)
11. Byte	00hex
12. Byte	Feldtyp (»C«, »N«, »F«, »D«, »L«, oder »M«)
13.-16. Byte	reserviert
17. Byte	Feldlänge
18. Byte	Anzahl der Dezimalstellen
19.-32. Byte	reserviert
	Vorspannende
1 Byte	Wert: 0Dhex
B. Datensätze	
1. Byte	2Ahex, falls Datensatz zum Löschen markiert, sonst: 20hex
	Es schließen sich die Feldinhalte ohne Begrenzungszeichen an. Die Feldinhalte sind als ASCII-Zeichen gespeichert. Die Zeichenfeldinhalte: linksbündig; Die Inhalte numerischer Felder: rechtsbündig. Die Felder haben jeweils die im Vorspann angegebene Länge. Nicht durch Daten benötigter Platz wird durch Leerzeichen (ASCII 20hex) gefüllt. (Einzelheiten im Text).
C. Dateiendemarke	
1 Byte	1Ahex

Tabelle 2. Byteweise Anatomie einer dBase-Datei

Eine dBase-Datei besteht nun aus einem Vorspann, den eigentlichen Datensätzen und einem Byte, das das Ende der Datei markiert. Die Byteverteilung im einzelnen:

- Der Dateivorspann (auch »Dateiheader« oder -kopf genannt) besteht aus einem 32 Byte langen Block, der allgemeine Dateiinformationen enthält (das sind die ersten zwei Zeilen in Bild 1), aus einem Block, der je Datensatzfeld 32 Byte lang ist und die Feldbeschreibung beinhaltet (je Feld also ein Zeilenpaar, in Bild 1 von Zeile 3 bis 14), und schließlich aus einem einzelnen Byte, das das Vorspannende markiert.
- Allgemeine Dateiinformationen:
  - Das erste Byte der Datei ist das am schwierigsten zu interpretierende. Es hat mit der dBase-Version zu tun, unter der die Datei erzeugt wurde, und es enthält außerdem die Information, ob die Datei Memofelder enthält und somit eine Memodatei existieren sollte, in der Memoeinträge (dies sind die bereits erwähnten Texteinträge variabler Länge) gespeichert werden. Das erste Byte von dBase-III-Plus-Dateien hat entweder den Wert 03hex oder den Wert 83hex – im Dualsystem ausgedrückt: 0000 0011 beziehungsweise 1000 0011.
    Dabei stehen die Bits an den Stellen 0 und 1 (Bits werden von rechts nach links beginnend mit der Position 0 gezählt) für die dBase-Version III Plus. Wenn das siebte Bit auf 1 steht, existiert in der Bestandsdatei ein Memofeld – dBase sucht also beim Öffnen der Datei nach einer zugehörigen Memodatei. Steht das siebte Bit auf 0, existiert keine Memodatei. Alle Dateien, deren erstes Byte nicht einen der beiden genannten Werte (03hex oder 83hex) aufweist, werden von dBase III Plus nicht als dBase-Dateien akzeptiert. Das erste Byte einer dBase-IV-Datei hat ebenfalls den Wert 03hex (obwohl 04hex für die Version IV zu erwarten wäre), wenn die Datei kein Memofeld aufweist und ihr damit keine Memodatei zugeordnet ist. Wie dBase III Plus setzt auch dBase IV das siebte Bit des ersten Bytes einer Dbf-Datei auf 1, wenn ihr eine Memodatei zugeordnet ist. Allerdings setzt dBase IV auch Bit 3 auf 1, wenn die Bestandsdatei ein Memofeld enthält, so daß sich für Byte 1 bei vorhandener Memodatei dual 1000 1011 oder hexadezimal 8Bhex ergibt: Und das entspricht genau dem Beispiel, wo tatsächlich ein Memofeld vorgesehen ist – die Beispieldatei gemäß Bild 1 ist also in dBase IV erzeugt worden. Daß dBase IV das erste Byte einer Bestandsdatei auf 8Bhex und nicht wie dBase III Plus auf 83hex setzt, wenn die Datei ein Memofeld enthält, ist der Grund dafür, daß dBase-IV-Dateien mit Memofeldern von dBase III Plus nicht gelesen werden können.
    Die eben gemachten Angaben über das erste Byte von dBase-IV-Dateien müssen mit der Einschränkung versehen werden, daß das erste Byte unter bestimmten Bedingungen – zum Beispiel im Zusammenhang mit SQL-Anwendungen – andere als die angeführten Standardwerte aufweisen kann. Allgemein gilt, daß dBase IV nur Dateien als dBase-Dateien akzeptiert, bei denen die linken drei Bits des Bytes auf 011 stehen. Steht außerdem Bit 7 auf 1, wird eine Memodatei erwartet.
  - Datum der letzten Änderung: Die nächsten drei Bytes einer dBase-Datei stehen für das Datum der letzten Änderung in der Reihenfolge Jahr, Monat, Tag – im vorliegenden Beispiel: 5A 07 09hex. Wenn Sie die Werte in Dezimalwerte umformen, ergibt sich als Datum der 09.07.90.
  - Anzahl der Datensätze in der Datei: Die nächsten vier Bytes geben an, wie viele Datensätze enthalten sind. Dabei ist das am weitesten links stehende Byte das niedrigstwertige, das am weitesten rechts stehende das höchstwertige (ein Format, das für die IBM-Welt typisch ist). Im Beispiel steht das niedrigstwertige Byte auf 02hex, alle höherwertigen Bytes stehen auf 0. Demnach enthält die Bestandsdatei zwei Datensätze, was ja auch den Tatsachen entspricht.
  - Länge des Vorspanns in Byte: Das neunte und zehnte Byte der Datei geben die Länge des Dateikopfs in Byte an – wiederum ist das neunte Byte das niederwertige, das zehnte das höherwertige. In unserem Beispiel ergibt sich eine Vorspannlänge von 125 (E1hex) Byte.
  - Länge eines Datensatzes in Byte: Es folgt ein 2-Byte-Wert, der die Länge eines Datensatzes angibt. Ein Datensatz der Datei »film.dbf« enthält also 47 Byte (2Fhex).
  - Es folgen zwei reservierte Bytes, das 15. Byte hat bei dBase-IV-Dateien den Wert 01hex, wenn ein Begin-Transaction-Befehl nicht abgeschlossen worden ist, und das 16. Byte steht auf 01hex (dBase IV), wenn die Datei verschlüsselt ist. Es folgt eine Reihe reservierter Bytes. Das 29. Byte hat bei dBase-IV-Dateien den Wert 01hex, wenn der Datei eine Mdx-Indexdatei zugeordnet ist. Die restlichen drei Bytes des ersten 32-Byte-Blocks sind wiederum reserviert – in der Beispieldatei stehen all diese Werte auf 0.
- Feldinformationen: Im Vorspann folgt nun pro Feld der Bestandsdatei ein 32-Byte-Block, der jeweils folgendermaßen aufgebaut ist:

- Feldname: Die ersten zehn Bytes enthalten den Feldnamen. Sie werden durch ein folgendes 0hex-Byte abgeschlossen. Feldnamen können Buchstaben, Ziffern und Interpunktionszeichen enthalten, die im ASCII-Format gespeichert werden. Nicht durch Namenszeichen belegte Bytes haben den Wert 0hex – Feldnamen werden also nicht durch Leerzeichen (20hex) aufgefüllt. Vergleichen Sie dazu Bild 1, wo die ersten fünf Bytes, die den Feldnamen angeben, die Werte 54 49 54 45 4Chex haben. Einer ASCII-Tabelle können Sie entnehmen, daß diese Werte die Buchstaben »TITEL« codieren. Es folgen insgesamt sechs Bytes mit dem Wert 0hex.
- Feldtyp: Das 12. Byte jedes 32-Byte-Feldbeschreibungsblocks enthält ein Zeichen im ASCII-Format, das den Feldtyp angibt: »C« steht für Zeichenfelder (Character), »N« für numerische Felder (Numeric), »D« für Datumsfelder (Date), »L« für logische Felder (Logical), »M« für Memofelder und »F« (nur dBase IV) für Gleitkommafelder (Float). Die folgenden vier Bytes sind reserviert.
- Feldlänge in Byte: Das 17. Byte gibt die Feldlänge an (im Beispiel haben die Felder in dieser Reihenfolge die Längen 15, 10, 2, 8, 1 und 10 – jeweils dezimal angegeben). Zeichenfelder können höchstens eine Länge von 254 Byte, numerische und Gleitkommafelder höchstens eine Länge von 20 Byte haben. Datumsfelder sind immer 8 Byte lang, logische Felder 1 Byte und Memofelder (siehe unten) sind 10 Byte lang.
- Dezimalstellen: Das 18. Byte gibt die Anzahl der Dezimalstellen an, was natürlich nur bei numerischen und Gleitkommafeldern interessant ist. Die restlichen Bytes jedes 32-Byte-Feldbeschreibungsblocks sind reserviert.

- Ende des Vorspanns: Es folgt ein Byte, das immer den Wert 0Dhex hat und das Ende des Vorspanns markiert. Ein Wort zu den reservierten Bytes im Dateivorspann: Wenn Sie eine Datei unter dBase bearbeitet haben, kann es sein, daß einige der reservierten Bytes Werte annehmen, die von 0 verschieden sind – abhängig zum Beispiel von der Adresse im Arbeitsspeicher, an der Feldinhalte geladen waren, oder vom Arbeitsbereich, in dem die Datei geöffnet wurde. Wie auch immer, wenn alle reservierten Bytes auf 0 stehen, kann eine Datei problemlos von dBase bearbeitet werden.

- Datensätze: Die eigentlichen Daten schließen sich an den Vorspann an. Jedem Datensatz geht entweder ein Leerzeichen (ASCII: 20hex) oder ein Stern (ASCII: 2Ahex) voran. Ein Stern besagt, daß der folgende Datensatz zum Löschen markiert ist. Jedes Feld eines Datensatzes wird in der im Vorspann festgelegten Länge gespeichert, wobei die Feldinhalte von Zeichenfeldern linksbündig, diejenigen von numerischen Feldern rechtsbündig angeordnet werden. Der durch den Feldinhalt nicht benötigte Platz wird durch Leerzeichen (ASCII: 20hex), nicht durch Bytes mit dem Wert 0hex, gefüllt. Alle Feldinhalte werden durch ASCII-Zeichen dargestellt; für Zeichenfelder sind alle ASCII-Zeichen, für numerische Felder Ziffern, der Dezimalpunkt und das Minuszeichen, für logische Felder die Zeichen »J« (Ja), »N« (Nein), »T« (True), »F« (False) und das Fragezeichen (unbestimmter Wahrheitswert) zugelassen. Datumsfelder werden im Format Jahr (vier Ziffern), Monat (zwei Ziffern) und Tag (zwei Ziffern) gespeichert. Es wird Ihnen leichtfallen, diese Angaben mit dem Beispiel in Bild 1 zu vergleichen.
  Einen Sonderstatus haben die Memofelder in einer Datei: Es wurde bereits erwähnt, daß die Memofeldeinträge Texteinträge von variabler Länge sein können, die nicht in der eigentlichen Bestandsdatei (Dbf-Datei) gespeichert werden, sondern in einer ihr zugeordneten Memodatei (mit dem Namen der Datei und der Ergänzung »dbt«). Im 10 Byte langen Memofeld innerhalb der Dbf-Datei wird lediglich eine Nummer gespeichert, die sich auf die Nummer des – normalerweise 512 Byte großen – Blocks innerhalb der Dbt-Datei bezieht, in dem der eigentliche Memotext zu dem fraglichen Datensatz gespeichert ist (oder in dem er beginnt, falls seine Länge 512 Byte überschreitet). In der Beispieldatei wurde eine Bemerkung über die Hauptdarstellerin des Lubitschfilms eingegeben. Demzufolge weist das Memofeld auf den Block 1 der zugeordneten Dbt-Datei. Der erste 512-Byte-Block der Dbt-Datei (Block 0) ist immer der Vorspann der Dbt-Datei und enthält Informationen über diese Datei. Da zum zweiten Datensatz der Beispieldatei keine Bemerkung eingegeben ist, verweist das Memofeld dieses Datensatzes nicht auf die Dbt-Datei, sondern enthält lediglich Leerzeichen.
  Die einzelnen Felder der Datensätze und die einzelnen Datensätze werden im dBase-Format durch keinerlei Trennzeichen voneinander abgegrenzt.

- Dateiendemarke: Das letzte Byte einer dBase-Datei hat immer den Wert 1Ahex und markiert das Ende der Datei.
  Eine Zusammenfassung aller dieser Angaben gibt Ihnen Tabelle 2.

- dBase im Griff von Quick Basic (Version 4.5): Das Listing gibt ein Quick-Basic-Utility namens »Dbquick« wieder, das Ihnen die allgemeinen Dateiinformationen (Bild 2), die Feldbeschreibungen (Bild 3) und die Inhalte beliebiger Datensätze einer beliebigen dBase-III-Plus- oder dBase-IV-Datei in gut lesbarer Form auf dem Bildschirm anzeigt. Wenn Sie dieses Utility benutzen, können Sie folglich die leidigen Hexadezimaldarstellungen des letzten Abschnitts wieder vergessen. Das Programm, das Sie natürlich erst compilieren (oder auf der Databox beziehen), wird mit

  ```
  dbquick
  ```

  aufgerufen. Sie werden dann aufgefordert, eine dBase-Datei einzugeben (Pfadbezeichnungen sind zulässig, die Extension »dbf« muß nicht eingegeben werden). Anschließend können Sie sich die allgemeinen Dateiinformationen und die Dateistruktur anzeigen lassen. Ferner können Sie durch die Datensätze blättern oder beliebige Datensätze über Ihre Nummer anspringen. Dieses Programm soll in erster Linie illustrieren, wie auf die im letzten Abschnitt detailliert beschriebenen Informationen einer dBase-Datei mit Hilfe einer fremden Programmiersprache zugegriffen werden kann.

Ein Beispielprogramm zeigt die Dateiinformationen an

Die in diesem Zusammenhang interessanten Deklarationen und Anweisungen machen nur den kleinsten Teil des Programmcodes aus – sie werden im folgenden kurz beschrieben. Der größere Teil des Programms sorgt lediglich für die Benutzerführung und die Bildschirmgestaltung – diese Teile des Programms, die für das behandelte Thema weniger wichtig sind, werden nicht kommentiert. Zunächst werden zwei Datentypen, »Header« und »SatzStrukt«, deklariert (Zeilen 11 bis 35). Nach den Prozedurdeklarationen folgt die Deklaration einer Variablen vom Typ »Header« und eines (dynamischen) Arrays vom Typ »SatzStrukt«. Die Variable nimmt in der Prozedur »HoleDatei« die allgemeinen Dateiinformationen des Dateivorspanns auf. Im Array werden für jedes Feld die Feldinformationen des Dateivorspanns gespeichert. 1-Byte-Werte des Vorspanns werden dabei als Zeichenkette der Länge 1, 2-Byte-Werte als Integer- und 4-Byte-Werte als Long-Integer-Werte eingelesen. Ist ein 1-Byte-Wert numerisch, muß für die Ausgabe des entsprechenden Werts auf dem Bildschirm mit Hilfe der Asc-Funktion der (numerische) ASCII-Wert der Zeichenvariable ermittelt werden. Reservierte Bytes werden in Dummy-Variablen eingelesen.

```
          Informationen über dBASE-Datenbankdateien unter QUICK BASIC
                                DKUNDEN.DBF
                        Allgemeine Dateiinformationen
_____________________________________________________________________________



Version:                           dBASE III Plus / dBASE IV
Memodatei:                         nicht vorhanden
Datum der letzten Änderung:        13.9.89
Anzahl der Datensätze:             33
Anzahl der Felder:                 11
Länge der Datensätze in Byte:      195    (incl. Löschkennzeichen)
Länge des Vorspanns in Byte:       385
Transaktionsbit:                   nicht gesetzt
Datenbank:                         nicht kodiert
MDX-Indexdatei:                    nicht vorhanden
Dateilänge in Byte:                6821




_____________________________________________________________________________
I - Allgemeine DateiInfos  *  S - Struktur der Datenbank  *  D - Datensatznummer
N - Nächster Datensatz * V - Vorheriger Datensatz * A - Andere Datei  * E - Ende
```

Bild 2. Das Programm »dbquick.exe«, die compilierte Version des Quick-Basic-Listings, zeigt neben den allgemeinen Dateiinformationen ...

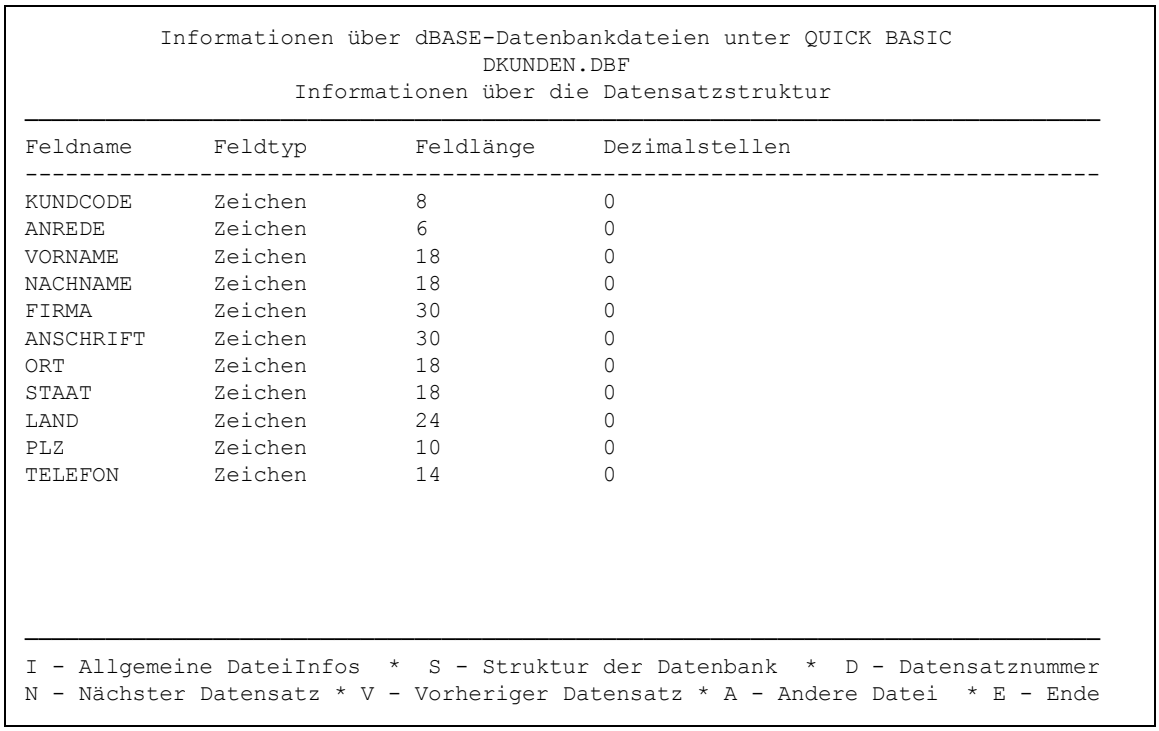

Bild 3. ... auch die Struktur des Datensatzes an

Nach den Deklarationen wird die Prozedur »HoleDatei« aufgerufen, die vom Benutzer einen Dateinamen erfragt (ab Zeile 140). Die Datei wird, falls vorhanden, als Binärdatei geöffnet. Wenn dabei ein Fehler auftritt, verzweigt das Programm in die Fehlerroutine mit der Marke »FehlerProz« (ab Zeile 112). Anschließend wird geprüft, ob die Datei von ihrer Länge her eine dBase-Datei sein kann. Sofern dies der Fall ist, werden mit dem Befehl

```
GET #1, 1, Head
```

die allgemeinen Dateiinformationen des Vorspanns in die Variable vom Datentyp »Header« gespeichert. Dann werden – wegen der Komplexität der Bedingung in zwei Zeilen – die Vorspanninformationen auf Plausibilität geprüft. Diese Fehlerprüfung könnte noch engmaschiger ausfallen, wenn die Vorspanninformationen auf interne Stimmigkeit untersucht würden – darauf wurde aus Platzgründen verzichtet. Nach diesen Fehlerprüfungen werden die Feldinformationen in das (mit Hilfe der Vorspanninformationen neu dimensionierte) Array vom Typ »Satz Struktur« eingelesen. Die Prozedur »ZeigeVorspann« (ab Zeile 262) wertet die allgemeinen Dateiinformationen, die Prozedur »ZeigeStruktur« (ab Zeile 234) die Feldinformationen des Dateivorspanns aus. Die Prozedur »SatzAnzeige« (ab Zeile 187) dient der Anzeige der Datensätze, wobei für lange Felder ein Zeilenumbruch vorgesehen ist. Die beiden letztgenannten Prozeduren geben die Nummer des zuletzt angezeigten Feldes an das Hauptprogramm zurück, falls die Informationen nicht auf eine Bildschirmseite passen. Von den Werten der Variablen »FeldAngez%« beziehungsweise »FeldFertig%« hängt es dabei ab, ob der Benutzer mit der Leertaste weitere Informationen abfragen kann. Bevor die Prozedur »SatzAnzeige« aufgerufen wird, muß die Position, an der sich die Datensatzinformationen befinden, errechnet und mit der Seek-Anweisung festgelegt werden. In die Berechnung fließen die Vorspannlänge, die Nummer des anzuzeigenden Datensatzes und die Datensatzlänge ein. Die Prozedur »Ende« schließt die dBase-Datei und beendet das Programm.

(Karl-Ernst Prankel/al)

```
'* --------------------------------------*
'*  Prog.-Name: DBQUICK.BAS              *
'*  Anzeige von Informationen über       *
'*       dBase-Datenbankdateien          *
'*  Prog.-Sprache: Quick-Basic 4.5       *
'*          alle Grafikkarten            *
'*  (c) Redaktionsbüro Everts&Hagedorn   *
'*  Autor:  Karl-Ernst Prankel           *
'* --------------------------------------*
CONST FALSCH = 0, WAHR = NOT FALSCH
TYPE Header
  Version AS STRING * 1
  Jahr AS STRING * 1
  Monat AS STRING * 1
  Tag AS STRING * 1
  AnzSaetze AS LONG
  LenVorspann AS INTEGER
  LenSatz AS INTEGER
  Dummy1 AS INTEGER
  Transakt AS STRING * 1
  Kodiert AS STRING * 1
  Dummy2 AS STRING * 12
  Mdx AS STRING * 1
  Dummy3 AS STRING * 3
END TYPE
TYPE SatzStrukt
  Feldname AS STRING * 11
  FeldTyp AS STRING * 1
  Dummy01 AS STRING * 4
  LenFeld AS STRING * 1
  LenDezStelle AS STRING * 1
  Dummy02 AS STRING * 2
  ArbeitsBer AS STRING * 1
  Dummy03 AS STRING * 11
END TYPE
DECLARE SUB Ende ()
DECLARE SUB HoleDatei (Head AS ANY, Felder() AS ANY)
DECLARE SUB SatzAnzeige (FeldAngez%, Nr&, Felder() AS ANY)
DECLARE SUB ZeigeStruktur (FeldFertig%, Struktur() AS ANY)
DECLARE SUB ZeigeVorspann (Head AS ANY)
DIM SHARED Fehler
DIM Kopf AS Header
REM $DYNAMIC
DIM FeldInfo(1) AS SatzStrukt
CALL HoleDatei(Kopf, FeldInfo())
DO
  VIEW PRINT
  LOCATE 24: PRINT "I - Allgemeine DateiInfos  *  S - Struktur
der Datenbank  *  D - Datensatznummer"
  LOCATE 25: PRINT "N - Nächster Datensatz * V - Vorheriger
Datensatz * A - Andere Datei  * E - Ende";
  DO
    Eingabe$ = UCASE$(INPUT$(1))
  LOOP UNTIL INSTR("ADEINSV ", Eingabe$) <> 0
  LOCATE 22: PRINT STRING$(80, 196);
  IF Eingabe$ <> " " THEN FeldAngez% = 0: FeldFertig% = 1
  SELECT CASE Eingabe$
    CASE "A"
      CALL HoleDatei(Kopf, FeldInfo())
      SatzNummer& = 0
```

```
 59:     CASE "E"
 60:       CALL Ende
 61:     CASE "I"
 62:       CALL ZeigeVorspann(Kopf)
 63:     CASE "S"
 64:       CALL ZeigeStruktur(FeldFertig%, FeldInfo())
 65:     CASE "D"
 66:       IF Kopf.AnzSaetze <> 0 THEN
 67:         LOCATE 23: PRINT SPACE$(80);
 68:         DO
 69:           LOCATE 24: PRINT SPACE$(80); : LOCATE 24
 70:           PRINT "Geben Sie eine Datensatznummer zwischen 1 und"
+ STR$(Kopf.AnzSaetze) + " ein: ";
 71:           LINE INPUT ; "", SatzNr$
 72:           SatzNummer& = VAL(SatzNr$)
 73:           IF SatzNummer& >= 1 AND SatzNummer& <=
(Kopf.AnzSaetze) THEN
 74:             EXIT DO
 75:           ELSE
 76:             LOCATE 23: PRINT SPACE$(14) + "EINGEGEBENE
DATENSATZNUMMER AUSSERHALB DES BEREICHS!";
 77:           END IF
 78:         LOOP
 79:       END IF
 80:     CASE "N"
 81:       IF SatzNummer& < Kopf.AnzSaetze THEN SatzNummer& =
SatzNummer& + 1
 82:     CASE "V"
 83:       IF SatzNummer& > 1 THEN SatzNummer& = SatzNummer& - 1 ELSE
SatzNummer& = 1
 84:     CASE " "
 85:       IF FeldAngez% > 0 AND FeldAngez% < UBOUND(FeldInfo) THEN
 86:         SEEK #1, SEEK(1) - ASC(FeldInfo(FeldAngez% + 1).LenFeld)
 87:         CALL SatzAnzeige(FeldAngez%, SatzNummer&, FeldInfo())
 88:       END IF
 89:       IF FeldFertig% > 1 AND FeldFertig% < UBOUND(FeldInfo) THEN
 90:         CALL ZeigeStruktur(FeldFertig%, FeldInfo())
 91:       END IF
 92:   END SELECT
 93:   IF INSTR("DNV", Eingabe$) <> 0 THEN
 94:     IF Kopf.AnzSaetze <> 0 AND Kopf.LenVorspann + (Kopf.LenSatz
* SatzNummer&) <= LOF(1) THEN
 95:       SEEK #1, Kopf.LenVorspann + 1 + ((SatzNummer& - 1) *
Kopf.LenSatz)
 96:       CALL SatzAnzeige(FeldAngez%, SatzNummer&, FeldInfo())
 97:     ELSE
 98:       VIEW PRINT 5 TO 22
 99:       CLS 2
100:       LOCATE 12, 5
101:       IF Kopf.AnzSaetze = 0 THEN
102:         PRINT "Die Datenbank ist leer!"
103:       ELSE
104:         PRINT "Die Angaben im Dateivorspann stimmen nicht mit
der Dateigröße überein!": LOCATE , 5
105:         PRINT "Mehr als"; : PRINT (LOF(1) - Kopf.LenVorspann) \
Kopf.LenSatz; : PRINT "Datensätze passen nicht in die Datei!"
106:         SatzNummer& = ((LOF(1) - Kopf.LenVorspann) \
Kopf.LenSatz) + 1
107:       END IF
108:     END IF
109:   END IF
110: LOOP
```

```
111: END
112: FehlerProz:
113: LOCATE 13, 15
114: SELECT CASE ERR
115:   CASE 55
116:     CLOSE #1
117:     RESUME
118:   CASE 64
119:     PRINT "Unzulässiger Dateiname!"
120:   CASE 71
121:     PRINT "Diskette nicht bereit!"
122:   CASE 76
123:     PRINT "Pfad nicht gefunden!"
124:   CASE ELSE
125:     VIEW PRINT 9 TO 15
126:     CLS
127:     ON ERROR GOTO 0
128: END SELECT
129: LOCATE , 15: PRINT "Betätigen Sie eine beliebige Taste!"
130: Eing$ = INPUT$(1)
131: Fehler = WAHR
132: RESUME NEXT
133: REM $STATIC
134: SUB Ende
135: VIEW PRINT
136: CLS
137: CLOSE #1
138: END
139: END SUB
140: SUB HoleDatei (Head AS Header, Felder() AS SatzStrukt) STATIC
141: DO
142:   Fehler = FALSCH
143:   CLS
144:   PRINT SPACE$(10) + "Informationen über dBASE-Datenbankdateien
unter QUICK BASIC" + SPACE$(10)
145:   LOCATE 4: PRINT STRING$(80, 196)
146:   LOCATE 23: PRINT STRING$(80, 196)
147:   LOCATE 24, 1: PRINT "Geben Sie [Laufwerk, Pfad] Namen [und
Extension] einer dBASE -Datei ein!";
148:   LOCATE 25, 1: INPUT ; "[<ENTER> allein beendet das Programm]:
", Dateiname$
149:   IF Dateiname$ = "" THEN CALL Ende
150:   IF INSTR(Dateiname$, ".") = 0 THEN Dateiname$ = Dateiname$ +
".dbf"
151:   LOCATE 2, (82 - LEN(Dateiname$)) \ 2: PRINT UCASE$(Dateiname$)
152:   ON ERROR GOTO FehlerProz
153:   OPEN Dateiname$ FOR BINARY AS #1
154:   ON ERROR GOTO 0
155:   IF NOT Fehler THEN
156:     IF LOF(1) < 66 THEN
157:       IF LOF(1) = 0 THEN
158:         LOCATE 10, 15: PRINT "Die Datei " + Dateiname$
159:         LOCATE , 15: PRINT "hat eine Länge von 0 Byte!"
160:         LOCATE 13, 15: PRINT "Betätigen Sie die Taste <J>, um
die Datei zu löschen,"
161:         LOCATE , 15: PRINT "oder eine beliebige andere Taste!"
162:         IF UCASE$(INPUT$(1)) = "J" THEN CLOSE #1: KILL
Dateiname$
163:       ELSE
164:         LOCATE 11, 15: PRINT "Die Datei " + Dateiname$
165:         LOCATE , 15: PRINT "kann von ihrer Länge her keine
dBASE-Datei sein."
```

```
166:         LOCATE , 15: PRINT "Betätigen Sie eine beliebige
Taste!": Eing$ = INPUT$(1)
167:       END IF
168:       Fehler = WAHR
169:     ELSE
170:      GET #1, 1, Head
171:      IF ASC(Head.Version) MOD 8 <> 3 OR ASC(Head.Jahr) > 99 OR
ASC(Head.Monat) = 0 OR ASC(Head.Monat) > 12 OR ASC(Head.Tag) = 0 OR
ASC(Head.Tag) > 31 OR Head.AnzSaetze > 1000000000 OR Head.AnzSaetze <
0 THEN Fehler = WAHR
172:      IF Head.LenVorspann < 65 OR Head.LenVorspann > 8193 OR
Head.LenSatz < 2 OR Head.LenSatz > 4000 OR ASC(Head.Transakt) > 1 OR
ASC(Head.Kodiert) > 1 OR ASC(Head.Mdx) > 1 THEN Fehler = WAHR
173:      IF Fehler THEN
174:        LOCATE 10, 15: PRINT "Die gewählte Datei ist keine dBASE-
Datei oder"
175:        LOCATE 11, 15: PRINT "sie enthält unzulässige Werte im
Datei-Vorspann!"
176:        LOCATE 13, 15: PRINT "Betätigen Sie eine beliebige
Taste!": Eing$ = INPUT$(1)
177:       END IF
178:     END IF
179:   END IF
180: LOOP WHILE Fehler
181: REDIM Felder((Head.LenVorspann - 33) \ 32) AS SatzStrukt
182: FOR z% = 1 TO UBOUND(Felder)
183:   GET #1, , Felder(z%)
184: NEXT z%
185: CALL ZeigeVorspann(Head)
186: END SUB
187: SUB SatzAnzeige (FeldAngez%, Nr&, Felder() AS SatzStrukt) STATIC
188: VIEW PRINT
189: SatzNrAnzeige$ = "Anzeige von Datensatz Nummer:" + STR$(Nr&)
190: LOCATE 3: PRINT SPACE$((80 - LEN(SatzNrAnzeige$)) \ 2) +
SatzNrAnzeige$ + SPACE$((80 - LEN(SatzNrAnzeige$)) \ 2)
191: zeile% = 1
192: IF FeldAngez% = 0 THEN
193:   IF INPUT$(1, #1) = "*" THEN
194:     LOCATE 4, 24: PRINT " DATENSATZ ZUM LÖSCHEN MARKIERT! "
195:   ELSE
196:     PRINT STRING$(80, 196)
197:   END IF
198: END IF
199: VIEW PRINT 5 TO 22
200: CLS 2
201: FOR z% = FeldAngez% + 1 TO UBOUND(Felder)
202:   zeichenket$ = INPUT$(ASC(Felder(z%).LenFeld), #1)
203:   PRINT MID$(Felder(z%).Feldname, 1, INSTR(Felder(z%).Feldname,
CHR$(0)) - 1); : LOCATE , 11: PRINT ":";
204:   DO WHILE LEN(zeichenket$) > 0
205:     IF LEN(zeichenket$) > 65 THEN
206:       Position% = 65
207:       DO WHILE MID$(zeichenket$, Position%, 1) <> " "
208:         Position% = Position% - 1
209:         IF Position% = 1 THEN Position% = 65: EXIT DO
210:       LOOP
211:       LOCATE , 15: PRINT LEFT$(zeichenket$, Position%);
212:       zeichenket$ = RIGHT$(zeichenket$, LEN(zeichenket$) -
Position%)
213:     ELSE
214:       LOCATE , 15
215:       IF Felder(z%).FeldTyp = "D" THEN
```

```
216:         PRINT RIGHT$(zeichenket$, 2) + "." + MID$(zeichenket$,
5, 2) + "." + LEFT$(zeichenket$, 4);
217:       ELSE
218:         PRINT zeichenket$;
219:       END IF
220:      zeichenket$ = ""
221:     END IF
222:     zeile% = zeile% + 1
223:     IF zeile% = 19 AND (LEN(zeichenket$) > 0 OR z% <
UBOUND(Felder)) THEN
224:       VIEW PRINT
225:       LOCATE 23, 17: PRINT " Leertaste - Fortsetzung der Anzeige
des Satzes ";
226:       z% = z% - 1
227:       EXIT FOR
228:     END IF
229:     IF zeile% <> 19 THEN PRINT
230:   LOOP
231: NEXT z%
232: FeldAngez% = z%
233: END SUB
234: SUB ZeigeStruktur (FeldFertig%, Struktur() AS SatzStrukt) STATIC
235: VIEW PRINT
236: LOCATE 3: PRINT SPACE$(20) + "Informationen über die
Datensatzstruktur" + SPACE$(20)
237: PRINT STRING$(80, 196)
238: PRINT "Feldname      Feldtyp        Feldlänge
Dezimalstellen" + SPACE$(22)
239: PRINT STRING$(80, "-")
240: VIEW PRINT 7 TO 22
241: CLS 2
242: FOR z% = FeldFertig% TO UBOUND(Struktur)
243:   SELECT CASE Struktur(z%).FeldTyp
244:     CASE "C": Typ$ = "Zeichen"
245:     CASE "N": Typ$ = "Numerisch"
246:     CASE "D": Typ$ = "Datum"
247:     CASE "L": Typ$ = "Logisch"
248:     CASE "F": Typ$ = "Gleitkomma"
249:     CASE "M": Typ$ = "Memo"
250:     CASE ELSE: Typ$ = "unbekannt"
251:   END SELECT
252:   PRINT MID$(Struktur(z%).Feldname, 1, INSTR(Struktur(z
%).Feldname, CHR$(0))), Typ$, ASC(Struktur(z%).LenFeld),
ASC(Struktur(z%).LenDezStelle);
253:   IF z% MOD 16 = 0 AND z% < UBOUND(Struktur) THEN
254:     VIEW PRINT
255:     LOCATE 23, 12: PRINT " Leertaste - Fortsetzung der Anzeige
der Datensatzstruktur ";
256:     EXIT FOR
257:   END IF
258:   IF z% MOD 16 <> 0 THEN PRINT
259: NEXT z%
260: FeldFertig% = z% + 1
261: END SUB
262: SUB ZeigeVorspann (Head AS Header) STATIC
263: VIEW PRINT
264: LOCATE 3: PRINT SPACE$(25) + "Allgemeine Dateiinformationen" +
SPACE$(25)
265: LOCATE 23: PRINT STRING$(80, 196);
266: VIEW PRINT 5 TO 22
267: CLS 2
268: LOCATE 8: PRINT "Version:"; : LOCATE , 35
```

```
269: SELECT CASE ASC(Head.Version)
270:   CASE 3: PRINT "dBASE III Plus / dBASE IV"
271:   CASE 131: PRINT "dBASE III Plus"
272:   CASE ELSE: PRINT "dBASE IV"
273: END SELECT
274: PRINT "Memodatei:"; : LOCATE , 35
275: IF ASC(Head.Version) > 130 THEN PRINT "vorhanden" ELSE PRINT
"nicht vorhanden"
276: PRINT "Datum der letzten Änderung:"; : LOCATE , 35
277: PRINT LTRIM$(STR$(ASC(Head.Tag))); "."; LTRIM$(STR$
(ASC(Head.Monat))); ".";
278: IF ASC(Head.Jahr) < 9 THEN PRINT "0";
279: PRINT LTRIM$(STR$(ASC(Head.Jahr)))
280: PRINT "Anzahl der Datensätze:"; : LOCATE , 34: PRINT
Head.AnzSaetze
281: PRINT "Anzahl der Felder:"; : LOCATE , 34: PRINT
(Head.LenVorspann - 33) \ 32
282: PRINT "Länge der Datensätze in Byte:"; : LOCATE , 34: PRINT
Head.LenSatz; : PRINT "   (incl. Löschkennzeichen)"
283: PRINT "Länge des Vorspanns in Byte:"; : LOCATE , 34: PRINT
Head.LenVorspann
284: IF NOT ASC(Head.Version) = 131 THEN
285:   PRINT "Transaktionsbit:"; : LOCATE , 35
286:   IF ASC(Head.Transakt) = 0 THEN PRINT "nicht ";
287:   PRINT "gesetzt"
288:   PRINT "Datenbank:"; : LOCATE , 35
289:   IF ASC(Head.Kodiert) = 0 THEN PRINT "nicht ";
290:   PRINT "kodiert"
291:   PRINT "MDX-Indexdatei:"; : LOCATE , 35
292:   IF ASC(Head.Mdx) = 0 THEN PRINT "nicht ";
293:   PRINT "vorhanden"
294: END IF
295: PRINT "Dateilänge in Byte:"; : LOCATE , 34: PRINT LOF(1)
296: PRINT
297: END SUB
```

Listing. Anzeige von Informationen über DBF-Dateien, die Sie in dBase erzeugt haben

aus DOS International 11/90 – Seite 66-73