dBASE II

Assembly-Language
Relational Database Management
System

VOL. I *USER MANUAL*

Ashton-Tate
9929 WEST JEFFERSON BOULEVARD
CULVER CITY, CALIFORNIA 90230
(213)204-5570

dBASE II

Assembly Language
Relational Database Management System

WRITTEN BY
WAYNE RATLIFF

Software, Consultation, Design, and Production (SCDP)

USER MANUAL

dBASE II
USER MANUAL

IMPORTANT NOTICE REGARDING USE OF dBASE II

Copyright (C) 1981 Ashton-Tate
9929 West Jefferson Boulevard
Culver City, California 90230
(213) 204-5570

Willful violation of the Copyright Law of the United States can result in statutory damages of up to $50,000 in addition to actual damages, plus criminal penalties of imprisonment up to one year and/or a $10,000 fine.

The dBase II Computer Program is copyrighted and all rights are reserved by Ashton-Tate. Only you, as original purchaser, may use the dBase II Computer Program and only on a single computer system. Use of the dBase II Computer Program purchased hereby by any other entity or on a computer other than the one for which it is being purchased is an unauthorized use. As an original purchaser of dBase II, you are hereby licensed only to read the Program from its medium into the memory of a computer solely for the purpose of executing the Program. Except for the limited purpose of system back-up as specified in Section IV of the License Agreement, copying, duplicating, selling, or otherwise distributing the dBase II Computer Program is a violation of the law.

The dBase II User Manual is copyrighted and all rights are reserved by Ashton-Tate. The dBase II User Manual may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without the express written permission of Ashton-Tate.

The User Manual has been written by Software Consultation, Design, and Production (SCDP) in conjunction with Ashton-Tate. While reasonable efforts have been taken in the preparation of this manual to assure its accuracy, Ashton-Tate or SCDP assumes no liability resulting from any inaccuracies or omissions in this manual, or from the use of the information contained herein.

dBASE II IS LICENSED ON AN "AS IS" BASIS. THERE ARE NO WARRANTIES EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO

IMPLIED WARRANTIES OF MERCHANTABILITY OF FITNESS FOR A PARTICULAR PURPOSE AND ALL SUCH WARRANTIES ARE EXPRESSLY AND SPECIFICALLY DISCLAIMED.

Ashton-Tate or SCDP shall have no liability or responsibility to you or any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by dBASE II, the User Manual, or other computer programs sold by Ashton-Tate, including but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of such computer programs.

NOTE: Good data processing procedure dictates that the user test the program, run and test sample sets of data, and run the system in parallel with the system previously in use for a sufficient period of time to insure that results of operation of the programs are satisfactory.

dBase II is a trademark owned by Ashton-Tate.

Introduction and installation: 6

Introduction	6	
Typographic conventions used in this manual	6	
System Requirements	7	
dBASE II Specifications	7	
Making a backup	7	
Installing dBASE II on your system	8	INSTALL

Section I: 13

How to CREATE a database	14	CREATE
Entering data into your new database	16	
Modifying data in a database	18	EDIT
Full Screen Editing Features	19	
An introduction to dBASE II to commands and the error correction dialog	20	USE, DISPLAY, LIST
Expanding commands with expressions	21	LIST
Looking at your data records	23	DISPLAY
Positioning yourself in the database	24	GO, GOTO, SKIP
The interactive ? command	25	?
Adding more records to a database	26	APPEND, INSERT
Cleaning up a database	28	DELETE, RECALL, PACK
Section I Summary	29	

Section II: 31

Using expressions for selection and control	32	
Constants and variables	33	STORE
dBASE II operators	37	
Logical operators	38	
Substring logical operator	40	
String operators	41	
Changing an empty database structure	42	MODIFY
Duplicating databases and structures	43	COPY
Adding and deleting fields with data in the database	45	COPY, USE, MODIFY
Dealing with CP/M and other "foreign files"	47	COPY, APPEND
Renaming database fields	49	COPY, APPEND
Modifying data rapidly	50	REPLACE, CHANGE
Organizing your databases	52	SORT, INDEX
Finding the information you want	54	FIND, LOCATE
Getting information out of all that data	56	REPORT
Automatic counting and summing	58	COUNT, SUM
Summarizing data and eliminating details	59	TOTAL
Section II Summary	60	

Section III: 61

Setting up a command file (writing your first program)	62	MODIFY COMMAND <file>
Making choices and decisions	64	IF..ELSE..ENDIF
Repeating a process	66	DO WHILE..
Procedures (subsidiary command files)	67	DO <file>
Entering data interactively during a run	68	WAIT, INPUT, ACCEPT
Placing data and prompts exactly where you want them	69	@..SAY..GET
A command file that summarizes what we've learned	72	
Working with multiple databases	75	SELECT PRIMARY/SECONDARY
Generally useful system commands and functions	76	
A few words about programming and planning your command files	77	

Section IV: 79

Expanding your control with functions	80	
Changing dBASE II parameters and defaults	84	SET..
Merging records from two databases	86	UPDATE
JOINING entire databases	87	JOIN
Full screen editing and formatting	88	SET FORMAT TO SCREEN @..SAY..GET..PICTURE..
Formatting the printed page	90	SET FORMAT TO PRINT @..SAY..USING..
Setting up and printing a form	91	
Time to regroup	93	

Section V: 95

Database Basics	96	
A brief introduction to database organization	98	
dBASE II Records, Files and Data Types	99	
dBASE II OPERATION SUMMARY	102	
dBASE II FUNCTION SUMMARY	103	
dBASE II COMMAND SUMMARY	104	
Commands grouped by what you want done	109	
109 File structure		
110 File operations		
110 Organizing database		
110 Combining databases		
111 Editing, updating, changing data		
111 Using variables		
112 Interactive input		
112 Searching		
112 Output		
113 Programming		

Section VI: 115

A working accounting system	115	

Introduction

dBASE II is a database management tool that allows easy manipulation of small and medium sized databases using English-like commands. With dBASE II you can:

* Create complete database systems.

* Easily add, delete, edit, display and print data from your database, with a minimum of data duplication on file.

* Gain a large measure of program/data independence, so that when you change your data your don't have to change your programs, and vice-versa

* Generate reports from one or more databases, automatically do multiplication, division, sub-totals, totals and other data manipulation every time you use them.

* Use the full-screen editing capability to set up a screen format, so that you see exactly what you're going to get, and enter data by simply "filling in the blanks."

dBASE II is an extremely powerful system. To get the most out of it, please take the time to read the instructions before you start using it. The time will be well spent.

Typographic conventions used in this manual:

Lowercase in the screen representations indicates material that you type in.

Uppercase in the screen representations indicates the dBASE II prompts and responses. In text, uppercase is used for dBASE II commands.

^...^ will be used in the text of this manual to set off dBASE II commands and materials you type. Occasionally, they may are used in the screen representations if needed for clarity. DO NOT TYPE THE SYMBOLS.

[...] square brackets will be used to indicate parts of a dBASE II command that are optional.

<...> bracket portions of a dBASE II command that are to be filled in with real information. E.g.: <filename> means the name of a file is to be inserted. They are also used in text to bracket field names and file names.

<enter> means press the carriage return or "enter" key on your keyboard. DO NOT TYPE THIS WORD, NOR THE SYMBOLS.

System Requirements

dBASE II requires the following hardware and software environment:

* 8080, 8085 or Z-80 based microprocessor system (Like the TRS-80/II, Northstar, Apple II with the Z-80 card, etc.)
* 48K bytes minimum of memory (dBASE II uses locations from 5CH to A400H) for most micros, 56k for Apple, Heath, North Star and a few others.
* CP/M (version 1.4 or 2.x), CDOS OR CROMIX operating systems.
* One or more mass storage devices (usually floppy disk drives)
* A cursor-addressable CRT if full screen operations are to be used.
* Optional text printer (for some commands).

dBASE II Specifications

Records per database file	65535	max
Characters per record	1000	max
Fields per record	32	max
Characters per field	254	max
Largest number	$\pm 1.8 \times 10^{63}$	approx
Smallest number	$\pm 1 \times 10^{-63}$	approx
Numeric accuracy	10	digits
Character string length	254	characters max
Command line length	254	characters max
Report header length	254	characters max
Index key length	100	characters max
Expressions in SUM command	5	max

BEFORE YOU DO ANYTHING ELSE, MAKE A COPY OF THE dBASE II DISC. STORE THE ORIGINAL IN A SAFE PLACE AND USE THE COPY.

Install a system disk in drive A and the dBASE II disk in drive B. Now type:

^PIP A:=B:*.*[OV]

The letter "O" is necessary to make certain that your operating system will copy all of the data from the distribution disk.

If you are working with a single drive, use the COPY or BACKUP commands, and follow the screen prompts.

Backups are essential, and should be done frequently. If you have a short session on your computer, once a session may be enough, otherwise do it much more frequently than that. You can balance the cost of doing the backups versus the cost of your data better than we can, but since you can rewrite disks, the cost of the backups is low. What's your entire accounting database worth?

This can't be over-emphasized!

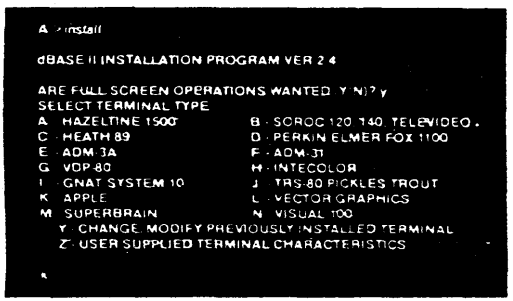
INSTALLING dBASE II ON YOUR SYSTEM.

Load the copy (you did make a copy, right?) of dBASE II into your logged-on drive and do any initialization that has to be done (control-C , reset, etc.)

Now type ^INSTALL^ to customize dBASE II to your system. (DO NOT TYPE THE "^" SYMBOLS.)

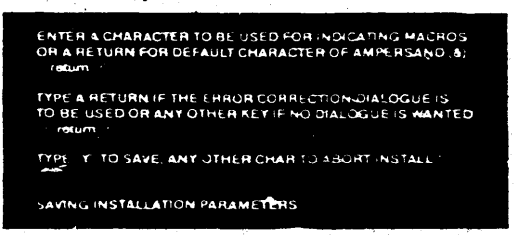
If your terminal does not have cursor X-Y positioning (see your manual), type ^N^ in answer to the prompt. Otherwise, type ^Y^. This provides you with the ability to do full-screen editing, a convenient way to enter data and work with your databases. Rather than ending up typing on the last line of the screen, with everything else scrolling up, you can position the cursor wherever you want it using dBASE II commands.

dBASE II then lists terminal types. If yours is listed, type the appropriate letter. If your terminal is not listed, type ^Z^.



If you selected one of the listed terminals, dBASE II then asks you which character you want to use for macro substitution (described in Section IV, used in Section VI and defined in Part II of this manual). If the ampersand will not conflict with your word processor, type <enter>. Otherwise, type in the symbol you want to use.

Initially, you will want to use the error correction dialogue, so type <enter>. This will allow you to correct an error without having to re-enter the entire command (page/ 20). (You can disable this feature later by using the "Y--CHANGE/MODIFY" option above).



At the end of the installation procedure, you can complete the installation by typing ^Y^, or you can abort the installation and return the terminal to whatever condition it was in before you started the procedure.

If your terminal is not listed and you typed ^Z^, dBASE II lists the terminal commands that you will require to complete the installation procedure for your terminal. You may also want to use this customization procedure to change the normal defaults that have been selected for your terminal (reverse video with certain commands, for example).

```
USER SUPPLIED SPECS ROUTINE

FOR THIS METHOD, YOU WILL NEED THE HEX
OR DECIMAL CODES THAT CAN BE SENT FROM
YOUR COMPUTER TO THE VIDEO TERMINAL TO
CONTROL IT

THE CODES OR SEQUENCES THAT YOU WILL
NEED ARE:

  DELETE A CHAR SEQUENCE
  DIRECT CURSOR POSITIONING SEQUENCE
  CLEAR SCREEN COMMAND
  HOME CURSOR COMMAND
        (CLEAR AND HOME CAN BE COMBINED)
  OPTIONAL: BRIGHT/DIM COMMANDS OR

TYPE 'Y' IF YOU WISH TO CONTINUE
Y
```

If you know your terminal codes for the above procedures, type ^Y^ to continue. dBASE II then prompts you through the entry of the codes. The example shown below is for an IBM 3101/12 terminal. This terminal does not allow highlighting or reverse video, so <enter> was typed for these questions.

dBASE II shows the previous values of the control bytes, so we have indicated the new values we typed in between two "^" symbols. DO NOT TYPE THESE SYMBOLS.

```
WILL YOU BE ENTERING COMMANDS AS HEX OR
DECIMAL? TYPE 'D' FOR DECIMAL OR
'H' FOR HEXADECIMAL
h

COMMANDS ARE ENTERED AS A SEQUENCE OF
NUMBERS TYPE A CARRIAGE RETURN TO END
A SEQUENCE

NOW ENTER THE CODES FOR CHARACTER
DELETION THIS IS THE SEQUENCE "BACKSPACE,
SPACE, BACKSPACE" ON MOST TERMINALS IF
THIS IS TRUE FOR YOUR TERMINAL, THEN
TYPE 'Y'

Y
```

```
--DIRECT CURSOR POSITIONING--

THE CURSOR CONTROL SEQUENCE IS
USUALLY A 3 TO 4 BYTE SEQUENCE. THE
FIRST ONE OR TWO BYTES ARE USUALLY
FIXED AND THE REMAINING BYTES CONTAIN
THE LINE AND COLUMN NUMBERS.

FIRST, ENTER THE POSITION IN THE
SEQUENCE THAT HOLDS THE COLUMN
NUMBER
'4'

NEXT, ENTER THE POSITION IN THE
SEQUENCE THAT HOLDS THE LINE NUMBER
'3'

MANY TERMINALS ADD A CONSTANT TO THE
LINE AND COLUMN NUMBERS. ENTER THE
CONSTANT BIAS FOR YOUR TERMINAL
'20'

NOW ENTER THE SKELETON FOR THE DIRECT
CURSOR COMMAND. ENTER A ZERO IN
THE PLACES WHERE COLUMN AND LINE
NUMBERS GO
(11 BYTE MAX)

ENTER CONTROL CODE BYTE 1: 03 '1B'
ENTER CONTROL CODE BYTE 2: 00 '59'
ENTER CONTROL CODE BYTE 3: 00 '0'
ENTER CONTROL CODE BYTE 4: 00 '0'
ENTER CONTROL CODE BYTE 5: 00 <return>

IS THIS CORRECT (Y/N)? y
```

```
--DIM/BRIGHT VIDEO/REVERSE VIDEO--

ENTER THE COMMAND THAT WILL SWITCH TO
HIGH INTENSITY OR NORMAL VIDEO
(5 BYTE MAX)

ENTER CONTROL CODE BYTE 1: 1D <return>

IS THIS CORRECT (Y/N)? y
```

```
--CLEAR AND HOME COMMAND(S)--

ENTER THE COMMAND(S) THAT WILL CLEAR
THE SCREEN AND PLACE THE CURSOR IN
THE UPPER LEFT CORNER OF THE TERMINAL
(11 BYTE MAX)

ENTER CONTROL CODE BYTE 1: 0C '1B'
ENTER CONTROL CODE BYTE 2: 00 '4C'
ENTER CONTROL CODE BYTE 3: 00 <return>

IS THIS CORRECT (Y/N)? y
```

```
ENTER THE COMMANDS TO BE ISSUED WHEN
ENTERING THE FULL-SCREEN EDITING MODE
(IF ANY)
(11 BYTE MAX)

ENTER CONTROL CODE BYTE 1: 00 <return>

IS THIS CORRECT (Y/N)? y
```

```
ENTER THE COMMAND THAT WILL SWITCH TO
STANDARD INTENSITY OR NORMAL VIDEO
TO RESET THE SCREEN AFTER FULL-SCREEN
OPERATIONS
(5 BYTE MAX)

ENTER CONTROL CODE BYTE 1: 1D

IS THIS CORRECT (Y/N)? y
```

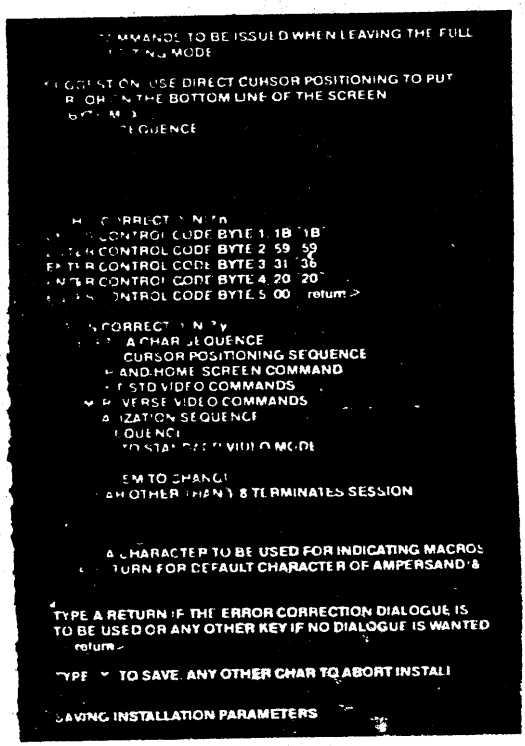
```
ENTER THE COMMANDS TO BE ISSUED WHEN
LEAVING THE FULL-SCREEN EDITING MODE

SUGGESTION: USE DIRECT CURSOR POSI
TIONING TO PUT CURSOR ON THE BOTTOM
LINE OF THE SCREEN.
(11 BYTE MAX)

ENTER CONTROL CODE BYTE 1: 1D '1B'
ENTER CONTROL CODE BYTE 2: 17 '59'
ENTER CONTROL CODE BYTE 3: 03 '31'
ENTER CONTROL CODE BYTE 4: 00 '20'
ENTER CONTROL CODE BYTE 5: 2E <return>

IS THIS CORRECT (Y/N)? y
```

```
ENTER A CHARACTER TO BE USED FOR INDICATING
MACROS OR A RETURN FOR DEFAULT CHARACTER OF
AMPERSAND (&) <return>

TYPE A RETURN IF THE ERROR CORRECTION DIALOGUE IS
TO BE USED OR ANY OTHER KEY IF NO DIALOGUE IS
WANTED. <return>

TYPE 'Y' TO SAVE, AND OTHER CHAR TO ABORT INSTALL
y
SAVING INSTALLATION PARAMETERS
```

```
MODIFY EXISTING SPECS ROUTINE

FOR THIS METHOD YOU WILL NEED THE HEX OR DECIMAL
CODES THAT CAN BE SENT FROM YOUR COMPUTER TO THE
VIDEO TERMINAL TO CONTROL IT

TYPE 'Y' IF YOU WISH TO CONTINUE
y

WILL YOU BE ENTERING COMMANDS AS HEX OR DECIMAL?
TYPE 'D' FOR DECIMAL OR 'H' FOR HEXADECIMAL
h

COMMANDS ARE ENTERED AS A SEQUENCE OF NUMBERS
TYPE A CARRIAGE RETURN TO END A SEQUENCE

1 - DELETE A CHAR SEQUENCE
2 - DIRECT CURSOR POSITIONING SEQUENCE
3 - CLEAR AND HOME SCREEN COMMAND
4 - BRIGHT/STD VIDEO COMMANDS
5 - DIM/REVERSE VIDEO COMMANDS
6 - INITIALIZATION SEQUENCE
7 - EXIT SEQUENCE
8 - RESET TO STANDARD VIDEO MODE

SELECT ITEM TO CHANGE
ANY CHAR OTHER THAN 1-8 TERMINATES SESSION
```

To modify an installed dBASE II system, type ^INSTALL^, then ^Y^ or ^N^ in response to the full-screen editing query, then select the ^Y^ option from the terminal listing. dBASE II responds with the following sequence of commands. In this example, we wanted to change the "EXIT" sequence to position the cursor on the 23rd line rather than the 17th line when leaving the full screen editing mode. (You'll find out about this as we go through dBASE instructions later in this manual.)

Notice that the numbers are entered in hexadecimal and the lines are numbered from 0 to 23, columns from 0 to 79.



dBASE II is now installed, and you can begin using it immediately.

Bring up dBASE II by typing ^dBASE^.

A prompt line asks for the date. If you enter a date, this will be recorded in your files as the last access every time you add to or delete from the file, and can be useful for keeping track of updates. If you want to ignore it, just hit <enter>.

dBASE II loads into memory, displays a sign-on message and shows the prompt dot (.) to indicate that it is ready to accept commands.

To show you how powerful and easy to use dBASE II actually is, the first thing we'll do is create a database and enter data into it.

It will only take a few minutes.

Section I: 13

How to CREATE a database..... 14
CREATE
Entering data into your new database..... 16
Modifying data in a database..... 18
EDIT
Full Screen Editing Features..... 19
An introduction to dBASE II to commands and the
error correction dialog..... 20
USE, DISPLAY, LIST
Expanding commands with expressions..... 21
LIST
Looking at your data records..... 23
DISPLAY
Positioning yourself in the database..... 24
GO, GOTO, SKIP
The interactive ? command..... 25
Adding more records to a database..... 26
APPEND, INSERT
Cleaning up a database..... 28
DELETE, RECALL, PACK
Section I Summary..... 29

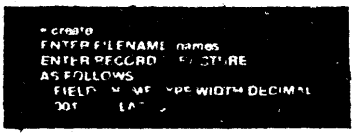
In this section, we create a database and enter data. We also introduce you to some dBASE II commands that will be developed and added to throughout the rest of this manual. For a complete definition of a command, check Part II.

How to CREATE a database

We'll start by creating a database of names for a mailing list system. Each record in the database will contain the following information:

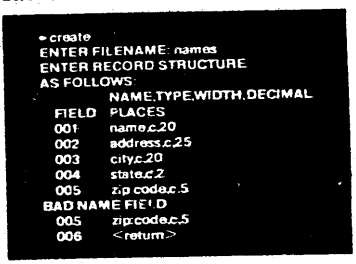
NAME: up to 20 characters long
ADDRESS: up to 25 characters long
CITY: up to 20 characters long
STATE: 2 characters long
ZIP CODE: 5 characters

First, type ^CREATE^.
dBASE II responds with: ENTER FILENAME:.
Enter a filename starting with a letter and up to 8 characters long (limited by CP/M), no colons, no spaces. Since this is a file of names, let's call it something that makes sense to a human being: type ^Names^.
When you hit return, dBASE II creates a file called <NAMES.DBF>. The part of the name after the period is the CP/M file name extension, and is short for database file (Section V, File Types).
In a database management system, each one of the items that we want to enter into a single related grouping is called a field and the grouping is called a record (Section V, Database Basics). In our example, each record will have 5 fields. dBASE needs to know the name of each field, what type of data it will contain, how long it is and how many decimal places if the data is numeric.



Field names can be up to 10 characters long, and may be entered in upper and/or lowercase. The name must start with a letter and cannot contain spaces, but can contain digits and embedded colons. Don't abbreviate any more than you have to: the computer will understand what you mean, but people might not.
The type of data is specified by a single letter: C for Character, N for Numeric and L for Logical. In this case, all fields contain character data.
Field width can be any length up to 254 characters. If the field is numeric and decimal places are specified, remember that the decimal point also takes one character position.

We know what names we want to give our fields, the type of data that they will contain, and their lengths so type the information in now. Here's what the screen looks like when you're finished:



Notice what happened at field 5: we made an error by entering a space in the field name, so dBASE II told us what the error was and gave us a chance to correct it.

Notice also that the data type for the ZIP code was specified as "character", even though we normally think of the digits here as numbers.

This was done because a dBASE II command such as ^TOTAL^ can total all the numeric fields in a record (without you specifically listing them all). Doing so with the ZIP code field would simply be a waste of time. We can still use the relational operators ("greater than", "less than", "equal or not equal to") with the character data, so this will not interfere with any ZIP code sorting we may want to do later.

When dBASE II asked us for the specifications for a sixth field, we hit <enter> to end the data definition. dBASE II saved the data structure, then asked if we wanted to enter data in it.

The <Names.DBF> database is immediately ready for data entry, so type ^y^. On the next page we tell you how to enter the data.

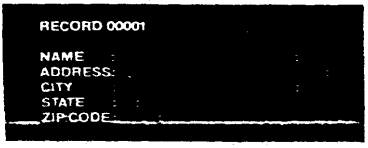
Entering data into your new database

If you do not have full screen editing on your terminal, the record number and the field names will appear one at a time below whatever has been typed on the screen up until now. The length of each field is shown by two colons, with the cursor positioned for you to start writing. When you fill the field or press <enter>, the next field will appear. After the last field in a record has been filled (or ignored), you start on a new record.

To stop entering data, hit <enter> when the cursor is at the first character position of the first field in a new record.

If you installed dBASE II with full screen editing, the screen will be erased, then the record number and all the fields will be displayed starting in the upper left-hand corner of the screen, with the cursor at the first character position of the first field.

(If you chose one of the standard terminals on the installation list, the field names may be in reverse video or at half-intensity. If you want to change this later, you can disable it by using the "Y - CHANGE/MODIFY" option in the installation procedure).



NOTE: If this doesn't look like your screen, there is a problem with INSTALL. Please re-do the installation.

Field lengths are indicated by two colons. When a field is filled or you hit <enter>, the cursor jumps down to the next field. The cursor can be moved back up to a previous field by holding the control key down and pressing the letter E once: ^control-E^, abbreviated as ^ctl-E^. When you are finished with the last field, dBASE II presents another empty record.

Enter the following names and addresses. We'll be using them soon to show you some of the powerful features of dBASE II.

ALAZAR, PAT	123	Crater Rd., Everett, WA	98206
BROWN, JOHN	456	Minnow Pl., Burlington, MA	01730
CLINKER, DUANE	789	Charles Dr., Los Angeles, CA	90036
DESTRY, RALPH	234	Mahogany St., Deerfield, FL	33441
EMBRY, ALBERT	345	Sage Ave., Palo Alto, CA	94303
FORMAN, ED	456	Boston St., Dallas, TX	75220
GREEN, TERRY	567	Doheny Dr., Hollywood, CA	90046
HOWSER, PETER	678	Dusty Rd., Chicago, IL	60631

If you make any mistakes that can't be corrected by backspacing and writing over them, read the next two pages on editing before moving on to the next record. If you accidentally get back to the dBASE dot prompt, type:

^USE Names^
^APPEND^

and continue with your entries. (This will be explained later in the manual).

To stop entering data, after you've entered the last ZIP code and while you are on the first character of the first field of the next record, hit <enter>. If you have typed in some data or moved the cursor, hold the control key down and press the letter "Q: (^control-Q^).

dBASE leaves the data entry mode and presents its dot prompt (.) to show you that it's ready for your commands.

If you want to stop now, simply type **^QUIT^**.

^QUIT^ must be typed every time you terminate a dBASE II session. This automatically closes all files properly. Unless you do so, you may destroy your database.

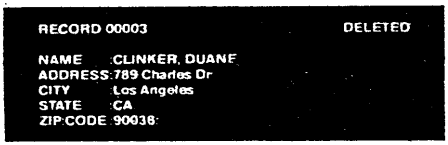
Modifying data with EDIT

If you made any errors in the entries, you can correct them quickly and easily in the Full Screen Edit mode. Type:

^USE Names^
^EDIT <number>^

where "number" is the number of one of the records in the database.

dBASE brings up the entire record and you can use the Full Screen Editing commands to modify any or all of the data in the record. To move to the next record, use ^ctl-C^. To move to the previous record. use ^ctl-R^. To try it, type ^EDIT 3^.



If you mark a record for deletion by using ^ctl-U^, "DELETED" appears at the top of the screen. Pressing ^ctl-U^ again removes the word and "un-deletes" the record. If you ^LIST^ (pp.20 and 21) or ^DISPLAY^ (pp. 20 and 23) your database, you will see an asterisk next to all records marked for deletion.

To abort full-screen editing, use ^ctl-Q^. This does not make the changes that were on the screen when you exited.

To exit gracefully and save the changes made so far, use ^ctl-W^ (^ctl-O^--the letter "O"--with Superbrain).

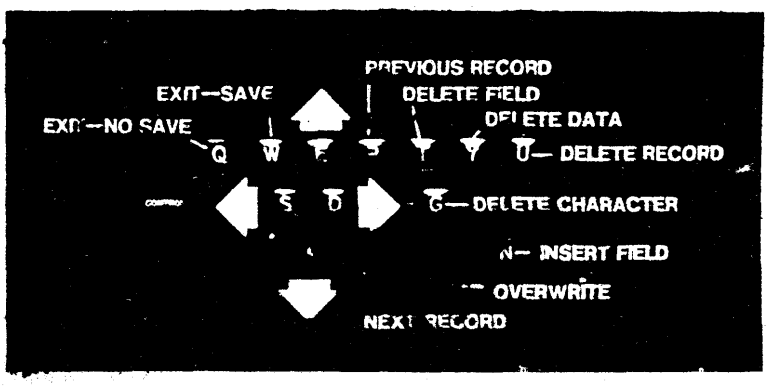
FULL SCREEN EDITING FEATURES:

ctl-X moves cursor down to the next field (or ctl-F).
ctl-E moves cursor back to the previous field (or ctl-A).
ctl-D moves cursor ahead one character.
ctl-S moves cursor back one character.
ctl-V toggles between overwrite and insert modes.
ctl-G deletes the character under the cursor.
<Rubout> deletes the character to the left of the cursor.
ctl-P toggles your printer ON and OFF.
ctl-Q quits and returns to normal dBASE II operation without making changes, even in the MODIFY mode.

^MODIFY^ functions:
ctl-T deletes the field where the cursor is and moves all the lower fields up.
ctl-Y clears the current field to blanks, but leaves all fields where they were.
ctl-N moves fields down one position to make room for insertion of a new field at the cursor position.

^APPEND^ functions:
ctl-R writes the record to disk and moves to the next record.
<Enter> when cursor is at the initial position of a new record resumes normal dBASE II operation.
ctl-Q erases the record and resumes normal dBASE II operation

^EDIT^ functions: (Do not use in ^APPEND^ mode)
ctl-C writes the record to disk and advances to the next record.
ctl-R writes the record to disk and backs up to the previous record.
ctl-U toggles the record deletion mark on and off.
ctl-W saves any changes made and resumes normal dBASE II operation. With Superbrain, use ^ctl-O^ (the letter "O").
ctl-Q aborts any changes in the record you're working on and prints the coordinate prompt. Hit <enter> to resume normal dBASE II operation.



An introduction to dBASE II commands and the error correction dialog (USE, LIST, DISPLAY)

dBASE II commands are generally verbs. You type them in when you see the dBASE II dot (.) prompt.

When you want to tell dBASE II which database file you want to work with, you type ^USE <filename>^'.

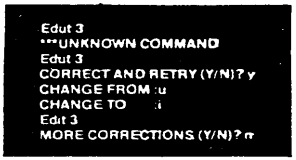
To look at the record you are on, type ^DISPLAY^.

To see all the records in the database, type ^LIST^. (To stop and start the scrolling, use ^control-S^.)

dBASE II commands can be abbreviated to four letters, but if you use more letters they must all be correct (^DISPLAY^, ^DISP^ and ^DISPLA^ are valid commands; DISPRAY is not.)

If you chose the error correction dialog when you installed dBASE II, the command line is scanned and you are prompted with error messages when mistakes are detected. You get a second chance to make corrections without having to retype the entire line.

Type ^EDUT 3^.



dBASE II repeats a command it does not know. If you decide to change it, you do not have to retype the entire command.

In response to "CHANGE FROM:" type in enough of the wrong part of the command so that it is unambiguous, then hit <enter>.

In response to "CHANGE TO:" type in the replacement for the material you want changed.

In this example, we changed only a single letter, but you'll find this feature useful when you are testing and debugging long command lines.

Tip: The ^ERASE^ command erases the screen and positions the prompt dot at the upper left-hand corner of the screen so that you can start new commands with a clean slate.

Expanding commands with expressions and relational operators (LIST)

One of the most powerful features of dBASE II is the ability to expand and "tailor" the commands.

You can add "phrases" and expressions to most commands to further define what the commands will do. Commands can be entered in upper and lowercase letters, and command lines can be up to 254 characters long. To extend the line beyond the width of your display, type in a semicolon (;) as the last character on the line (no space after it). dBASE II will use the next line as part of the command.

Since dBASE II is a relational DBMS, you'll find the relational operators useful:

```
<    : less than
>    : greater than
=    : equal to
<=   : less than or equal to
>=   : greater than or equal to
```

These commands mean exactly what the explanation on the right says. They generate a logical value as a result (True or False). If the expression is True, the command is performed. If the expression is false, the command is not performed.

Earlier, we mentioned that the LIST command will show all the records in the database (to stop and start the scrolling, use ^ctl-S^). The full form of the command is:

```
^LIST [OFF] [FOR <expression>]^
```

If the optional OFF is used, the record numbers will not be displayed.

If the optional FOR clause is used, dBASE II will list only the records for which the expression is true. Type the following, using single quotes around the character data (more on data types in Section II):

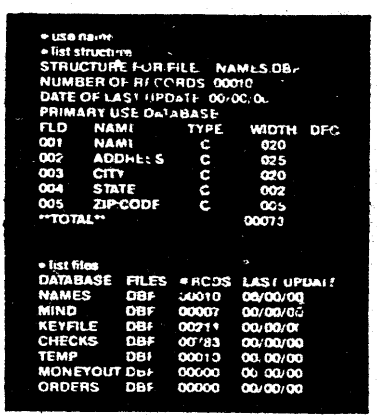
```
^USE Names^
^LIST^
^LIST OFF^
^LIST FOR Zip:Code = '9'^
^LIST OFF FOR Zip:Code < '8'^
^LIST FOR Name='GREEN'^
```

Notice that when you enter only part of the contents of the field, that is all that is compared by dBASE. We did not need Mr. Green's full name, for example, although we might have used it if our database contained several GREEN's.

In addition to precisely selecting data from your database, the LIST command can be used to provide you with system information.

^LIST STRUCTURE^ shows you the structure of the database in USE.

^LIST FILES^ shows the names of the database (.DBF) files on the logged-in drive. ^LIST FILES ON <drive>^ shows the database files on another drive (do NOT use the usual CP/M colon).



Looking at data with DISPLAY

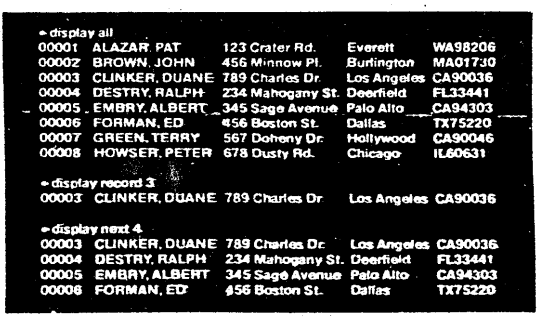
The ^DISPLAY^ command is similar to ^LIST^. Its full form is:

```
        [All      ]
DISPLAY [Record n] [OFF][FOR <expression>]
        [Next n   ]
```

This gives you the option of specifying the **scope** for the ^DISPLAY^ command (also ^LIST^).

Specifying "Record n" displays only that record; "Next n" displays the next "n" records, including the current record. ^DISPLAY ALL^ is the same as ^LIST^, except that ^LIST^ will scroll all the records in the database up the screen, while ^DISPLAY ALL^ shows you the database in groups of 15 records at a time (pressing any key displays the next 15 records). Type the following:

```
^DISPLAY All^
^DISPLAY Record 3^
^DISPLAY Next 4^
```



As with ^LIST^, the optional **FOR** clause can be used to select specific data by using logical expressions.

The DISPLAY command can also be used like the LIST command for system functions:

```
DISPLAY STRUCTURE = LIST STRUCTURE.
DISPLAY FILES = LIST FILES.
```

Both ^LIST^ and ^DISPLAY^ can show you specific types of files on a drive using the CP/M "wild cards". ^DISPLAY FILES LIKE *.COM ON B^, for example, would display all the ".COM" files on drive B. If uncertain, check your CP/M manual, then use this form:

```
^DISPLAY FILES LIKE <wild card>^
```

Positioning commands (GO or GOTO and SKIP)

Once you have your database set up, you can also move from record to record quickly and easily with dBASE II. Type the following:

```
^USE Names^
^GO TOP^
^DISPLAY^
^GO BOTTOM^
^DISPLAY^
^GOTO 5^
^DISPLAY^
^8^
^DISPLAY^
```

```
. use names
. go top
. display
00001  ALAZAR, PAT        123 Crater Rd.    Everett    WA98206

. go bottom
. display
00008  HOWSER, PETER      678 Dusty Rd.     Chicago    IL60631

. goto 5
. display
00005  EMBRY, ALBERT      345 Sage Avenue   Palo Alto  CA94303

. 8
. display
00008  HOWSER, PETER      678 Dusty Rd.     Chicago    IL60631
```

^GO TOP^ (or ^GOTO TOP^) moves you to the first record in the database. ^GO BOTTOM^ moves you to the last record. You can go to a specific record by using ^GOTO^ <number> (or GO <number>). And you can even eliminate the GO and just specify the record number.

^SKIP^ moves you to the next record. ^SKIP ± n^ moves you forward or backward "n" records. You can also use ^^SKIP ±<variable/expression>, with the number of records skipped determined by the value of the variable or expression (both defined later). Type the following:

```
^DISPLAY^
^SKIP-3^
^DISPLAY^
^SKIP^
^DISPLAY^
```

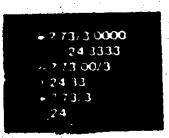
```
. display
00008  HOWSER, PETER      678 Dusty Rd.     Chicago    IL60631

. skip-3
RECORD: 00005

. display
00005  EMBRY, ALBERT      345 Sage Avenue   Palo Alto  CA94303

. skip
RECORD: 00006

. display
00006  FORMAN, ED         456 Boston St.    Dallas     TX75220
```

The interactive ? command

The ^?^ command allows you to use dBASE II in the calculator mode. Simply type in the question mark and a space followed by the quantity or mathematical function you want evaluated and dBASE II will provide the answer on the next line. Using ?? puts the answer on the same line.

Type the following:

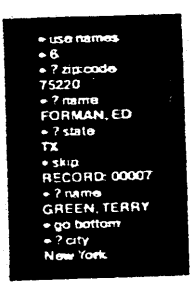
```
^? 73/3.0000^
^? 73.00/3^
^? 73/3^
```



The ^?^ command shows the answers to a mathematical operation to the same number of decimal places as the maximum in the numbers entered.

You can also think of ^?^ as meaning: "What is ...", with the dots replaced by an expression, a variable (a field name or a memory variable), a dBASE II function or a list of these separated by commas. Type the following:

```
^USE Names^
^6^
^? Zip:Code^
^? Name^
^SKIP^
^? Name^
^GO BOTTOM^
^? City^
```



In the section on functions and commands, we'll show you how the ^?^ can be used to access other dBASE II functions, and to display CRT prompts to the operator from a command file.

Adding more data with the APPEND and INSERT commands

You can add data to any database quickly and easily with a one-word command. First choose the database file into which you want to enter data by typing ^USE <filename>^, then typing in the command ^APPEND :

^USE Names^
^APPEND^

dBASE II responds by displaying the record number that follows the last record in the file and the fields for that database. If you fill in the record, it is added onto the end of the file (appended).

The display includes the names of the fields, with colons showing field lengths. The cursor is at the first position where you can start to enter data. If you fill up the entire field with data, the cursor automatically moves down to the next field. If not, hit <enter>.

If there is no data to be entered in a field, use <enter> to move the cursor to the next field. Character fields will automatically be filled with blanks, numeric fields will show a zero. When entering numeric data, if there are no digits after the decimal, there is no need to type the decimal. dBASE II automatically puts in the decimal point and the necessary number of following zeros.

Records can be inserted into a specific location in a database (to keep them alphabetical, for example) by typing

^INSERT [BEFORE] [BLANK]^

Using the word ^INSERT^ alone inserts the record just after the current record. Specifying BEFORE will insert the record just before the current record. In either case, you are prompted the same way as with the ^APPEND^ and ^CREATE^ commands. If BLANK is specified, an empty record is inserted and there are no prompts.

Add the following names alphabetically to the <Names.DBF> database:

EDMUNDS, JIM	392 Vicarious Way, Atlanta, GA	30328
INDERS, PER	321 Sawtelle Blvd., Tucson, AZ	85702
JENKINS, TED	210 Park Avenue, New York, NY	10016

The sequence of commands is:

```
^USE Names^
^5^
^INSERT BEFORE^   (enter the data for the first name)
^APPEND^          (enter the data for the last names)
```

In the ^INSERT^ mode, when you fill the last field, dBASE II will return to the command mode (dot prompt).

To exit the ^APPEND^ mode, position the cursor at the start of a new field, then hit <enter> or ^control-Q^.

In either mode, you can exit from inside a record by using ^ctl-W^ (^ctl-O^ with Superbrain). This will save what has been entered up to that point and return you to the command mode.

^Cleaning up a database (DELETE, RECALL, PACK)^

Deletions can be made directly from dBASE II as well as in the ^EDIT^ mode.

To delete the current record, type ^DELETE^.

To delete more than one record, use the form ^DELETE <scope>^, where the scope is the same as for other dBASE II commands: All, Record n, or Next n.

To make the deletions conditional, expand the command to:

^DELETE [scope] [FOR <expression>]^

where "expression" is a condition or set of conditions that must be met. (This is developed in more detail in Section II).

Type ^DELETE FILE <drive>:<filename>^ to delete a file. But once you've done this, the data is gone forever, so be careful.

Unlike files, records marked for deletion can be recovered. Rather than erasing the data, ^DELETE^ marks each record with an asterisk. You will see the asterisks when you ^LIST^ or ^DISPLAY^ the records. dBASE II then ignores these records, and does not use them in any processing.

To restore the records, use the following command:

^RECALL [scope] [FOR <expression>]^

This operates the same way ^DELETE^ does, with the scope and condition being optional. If a conditional expression is used, it does not have to be the same as was used to mark the records for deletion.

At some point, however, you will want to clean up your files to clarify displays or to make more room for storage. To do this, type:

^PACK^.

This erases all records marked for deletion, and tells you how many records are in the database.

Note: once you use this command, the records are lost forever.

To see how these commands work, type the following:

```
^USE Names^
^LIST^
^DELETE RECORD 2^
^DELETE RECORD 4^
^LIST^
^RECALL RECORD 4^
^LIST^
^PACK^
^LIST^
```

The screen below shows the first few records in our <Names.DBF> as we perform these commands.

```
. list
00001  ALAZAR, PAT       123 Crater Rd.     Everett       WA98206
00002  BROWN, JOHN       456 Minnow Pl.     Burlington    MA01730
00003  CLINKER, DUANE    789 Charles Dr.    Los Angeles   CA90036
00004  DESTRY, RALPH     234 Mahogany St.   Deerfield     FL33441
00005  EDMUNDS, JIM      392 Vicarious Way  Atlanta       GA30328

. delete record 2
00001 DELETION(S)
. delete record 4
00001 DELETION(S)
. list
00001  ALAZAR, PAT       123 Crater Rd.     Everett       WA98206
00002 *BROWN, JOHN       456 Minnow Pl.     Burlington    MA01730
00003  CLINKER, DUANE    789 Charles Dr.    Los Angeles   CA90036
00004 *DESTRY, RALPH     234 Mahogany St.   Deerfield     FL33441
00005  EDMUNDS, JIM      392 Vicarious Way  Atlanta       GA30328

. recall record 4
00001 RECALL(S)
. list
00001  ALAZAR, PAT       123 Crater Rd.     Everett       WA98206
00002 *BROWN, JOHN       456 Minnow Pl.     Burlington    MA01730
00003  CLINKER, DUANE    789 Charles Dr.    Los Angeles   CA90036
00004  DESTRY, RALPH     234 Mahogany St.   Deerfield     FL33441
00005  EDMUNDS, JIM      392 Vicarious Way  Atlanta       GA30328

. pack
PACK COMPLETE, 00004 RECORDS COPIED
. list
00001  ALAZAR, PAT       123 Crater Rd.     Everett       WA98206
00002  CLINKER, DUANE    789 Charles Dr.    Los Angeles   CA90036
00003  DESTRY, RALPH     234 Mahogany St.   Deerfield     FL33441
00004  EDMUNDS, JIM      392 Vicarious Way  Atlanta       GA30328
```

Section I Summary

At this point, you have learned about the power over data that a relational database management system like dBASE II can give you.

You can now ^CREATE^ a new database and start entering data in mimutes.

If you want to change the data, this is easily done with ^EDIT^, ^DELETE^, ^RECALL^ and ^PACK^.

You can ^APPEND^ or ^INSERT^ more data as required, and ^LIST^ and ^DISPLAY^ entire files or precisely selected records. You can also ^GOTO^ and ^SKIP^ around within a database quickly and easily.

Additionally, dBASE II can be used interactively as a powerful calculator (and more) with the ^?^ command.

We have introduced you to expressions and how they can be used to expand the power of dBASE II commands. In the next section, we will go into this in more detail and show you how to get useful information out of your databases quickly and easily.

Before that, please ^CREATE^ these two files, as we will need them for other examples.

```
. create
ENTER FILENAME MoneyOn
ENTER RECORD STRUCTURE
AS FOLLOWS
      NAME TYPE WIDTH
FIELD DECIMAL PLACES
001   Check Date C 7
002   Check Nmbr C 5
003   Client C 3
004   JobNumberN 3
005   Name C 20
006   Descrip C 20
007   Amount N 9 2
008   Bill Date C 7
009   Bill Nmbr C 7
010   Hours N 6 2
011   Emp Nmbr N 3
012
```

```
. create
ENTER FILENAME orders
ENTER RECORD STRUCTURE
AS FOLLOWS
      NAME TYPE WIDTH
FIELD DECIMAL PLACES
001   CustNmbr C 9
002   Item C 20
003   Qty N 4
004   Price N 7 2
005   Amount N 9 2
006   BackOrdr L 1
007   OrdrDate C 6
008
```

Section II: 31

Using expressions for selection and control..... 32
Constants and variables..... 33
STORE
dBASE II operators..... 37
Logical operators..... 38
Substring logical operator..... 40
String operators..... 41
Changing an empty database structure..... 42
MODIFY
Duplicating databases and structures..... 43
COPY
Adding and deleting fields
with data in the database..... 45
COPY, USE, MODIFY
Dealing with CP/M and other "foreign files"..... 47
COPY, APPEND
Renaming database fields..... 49
COPY, APPEND
Modifying data rapidly..... 50
REPLACE, CHANGE
Organizing your databases..... 52
SORT, INDEX
Finding the information you want..... 54
FIND, LOCATE
Getting information out of all that data..... 56
REPORT
Automatic counting and summing..... 58
COUNT, SUM
Summarizing data and eliminating details..... 59
TOTAL
Section II Summary..... 60

In this section, we develop the use of expressions to modify dBASE II commands. This is may be the most important part of learning how to use dBASE II effectively.

The dBASE II commands can be learned fairly easily because they are English-like, and learning another command is a matter of increasing your vocabulary (and your repertoire) by another word.

Expressions, combined with the commands, give you the fine control you need to manipulate your data to perform specific tasks. Once you have learned how to handle expressions, you will only have to learn two more things about programming to be able to write effective applications command files. (These are how to make decisions and how to repeat a sequence of commands, covered in Section III).

Using expressions for selection and control

We gave you a brief introduction to expressions that can be used with dBASE II commands in Section I.

As you saw, they are a powerful way to extend the commands and manipulate your data quickly and easily. If you check the index of commands in Section VI, you'll see that many dBASE II commands can be modified in the form:

^<COMMAND> [FOR <expression>]^

This extended power gives you a flexibility that you simply do not get with other database management systems. We've been told by experienced programmers that they can write a program (a dBASE II command file) for an application in as little as one-tenth the time it would take them using BASIC or even higher level languages such as COBOL, FORTRAN and PL/1.

But to take advantage of this power, you need to understand how to work with expressions and operators, then how to combine the modified commands into command files that will perform the same tasks again and again.

The next few pages will get you started. Ultimately, experience is going to be the best teacher.

Reminder: as we introduce commands through the text, we try to explain a particular aspect of the command that will allow you to a few more things with your database. This means that we do not cover the entire command at one time. To find out all that a command can do, use the summary at the end of Part I and the definitions of Part II.

Note: If, after you've finished this Section, you are still uncertain about how to write expressions that make the dBASE II commands do exactly what you want done, you may want to look at some beginning programming texts at your local library. Most of them discuss expressions within the first two chapters or so.

Constants and variables (STORE)

Expressions in dBASE II are used to help select and manipulate the data in your database (see ^DISPLAY^). The quantity that you manipulate may be either a **constant** or a **variable**.

Constants are data items that do not change, no matter where they appear in a database or within the computer. They are literal values because they are exactly what they represent. Examples are numerals such as 3 and the logical values T and F.

Characters and character strings (all the printable characters plus spaces) can also be constants, but must be handled a bit differently.

"Strings" are simply a collection of characters (including spaces, digits and symbols) handled, modified, manipulated and otherwise used as data. A "substring" is a portion of any specific string.

If a character or collection of characters is to be treated as a string constant, it must be enclosed in single or double quotes or in square brackets so the computer understands that it is to deal with the characters as characters. To see what we mean, get dBASE II up on your computer and USE <Names>. Type:

```
^dBASE^
^USE Names^
^? 'Name'^
^? Name^
```

In response to the first "What is..." (the ^?^ command), the computer responded with NAME because that was the value of the constant. When you eliminated the single quotes, the computer first checked to see if the word was a command. It wasn't, so it then checked to see if it was the name of a variable.

Variables are data items that can change. Frequently they are the names of database fields whose contents can change. In this case, the computer found that our database had a field called <Name> so it gave us the data that was in that field at that time. Type the following:

```
^SKIP 3^
^? Name^
```

Now type ^USE^. Since we do not specify a file na the computer simply closes all files.

If we type ^? Name^ again, the computer tells us that we made an error. In this case, we tried to use a variable that did not exist because we were no longer USing a file with a matching field name.

The variables can also be **memory variables** rather than field names. dBASE II reserves an area of memory for storing up to 64 variables, each with a maximum length of 254 characters, but with a maximum total of 1536 characters for all the variables.

You might want to think of this as a series of 64 pigeon-holes available for you to tuck data into temporarily while working out a problem.

Variable names can be any legal dBASE II identifier (start with a letter, up to ten characters long, optional embedded colon and numbers, no spaces).

You can use a memory variable for storing temporary data or for keeping input data separate from field variables. In one session, for example, we might "tuck" the date into a pigeon-hole (variable) called <Date>. During the session, we could get it by asking for <Date>, then place it into any date field in any database without having to re-enter it (see GetDate.CMD in Section VI).

To get data (character, numeric or logical) into a memory variable, you can use the ^STORE^ command. The full form is:

^STORE <expression> TO <memory variable>^

Type the following:

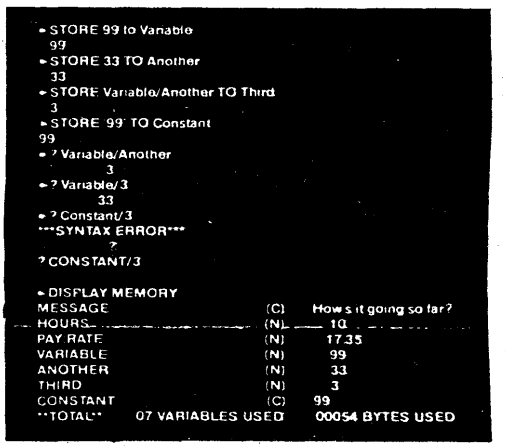
```
^STORE "How's it going so far?" TO Message
^STORE 10 TO Hours^
^STORE 17.35 TO Pay:Rate^
^? Pay:Rate*Hours^
^? Message^
```

```
. STORE "How's it going so far?" TO Message
How's it going so far?
. STORE 10 TO Hours
  10
. STORE 17.35 TO Pay:Rate
  17.35
. ? Pay:Rate*Hours
                173.50
. ? Message
How's it going so far?
```

Notice that we used double quotes around the character string (a constant) in the first line because we wanted to use the single quote as an apostrophe inside the string.

If this isn't clear yet, try experimenting with and without the quotes to get the distinction between constants and variables. To start you off, type the following:

```
^STORE 99 TO Variable^
^STORE 33 TO Another^
^STORE Variable/Another TO Third
^STORE '99' TO Constant^
^? Variable/Another^
^? Variable/3^
^? Constant/3^
^DISPLAY MEMORY
```



Entering a value into a variable automatically tells dBASE II what the data type is. From then on, you cannot mix data types (by trying to divide a character string by a number, for instance.)

RULES: Character strings that appear in expressions must be enclosed in <u>matching</u> single or double quote marks or square brackets. Character strings may contain any of the printable characters (including the space). If you want to use the ampersand (&) as a character, it <u>must</u> be between two spaces because it is also used for the dBASE II macro function (described later).

The last command in the previous screen representation is another form of ^DISPLAY^ that you'll find useful. (You can also ^LIST MEMORY^.)

You can eliminate a memory variable by typing ^RELEASE <name>^, or you can get rid of all the memory variables by typing ^RELEASE ALL^.

Type the following (you may want to ^ERASE^ the screen first):

```
^DISPLAY MEMORY^
^RELEASE Another^
^DISPLAY MEMORY^
^RELEASE ALL^
^DISPLAY MEMORY^
```

Tip: When naming any variables, try to use as many characters as necessary to make the name meaningful to humans.

Another tip: If you use only nine characters for database field names, when you want to use the name as a memory variable, you can do so by putting an "M" in front of it. What it stands for will be clearer when you come back to clean up your programs later than if you invented a completely new and different name.

dBASE II operators

Operators are manipulations that dBASE II performs on your data. Some of them will be familiar; others may take a bit of practice.

Arithmetic operators should be the most familiar. They generate arithmetic results.

() : parentheses for grouping
* : multiplication
/ : division
+ : addition
- : subtraction

The arithmetic operators are evaluated in a sequence of precedence. The order is: parentheses; multiply and divide; add and subtract. When the operators have equal precedence, they are evaluated from left to right. Here are some examples:

```
17/33*72 + 8 = 45.09     (divide, multiply then add)
17/(33*72 +8) = 0.00644 (multiply,add then divide)
17/33*(72 +8) = 41.21    (divide, add then multiply)
```

Relational operators make comparisons, then generate logical results. They take action based on whether the comparison is True or False.

< : less than
> : greater than
= : equal to
<> : not equal to
<= : less than or equal to
>= : greater than or equal to

Type the following:

```
^USE Names^
^LIST FOR Zip:Code <= '70000'^
^LIST FOR Address <> '123'^
^LIST FOR Name = 'HOWSER'^
```

```
. LIST FOR Zip:Code <= 70000
00003  DESTRY, RALPH      234 Mahogany St      Deerfield      FL33441
00004  EDMUNDS, JIM       392 Vicarious Way    Atlanta        GA30328
00008  HOWSER, PETER      678 Dusty Rd         Chicago        IL60631
00010  JENKINS, TED       210 Park Avenue      New York       NY10016

. LIST FOR Address <> 123
00002  CLINKER, DUANE     789 Charles Dr       Los Angeles    CA90036
00003  DESTRY, RALPH      234 Mahogany St      Deerfield      FL33441
00004  EDMUNDS, JIM       392 Vicarious Way    Atlanta        GA30328
00005  EMBRY, ALBERT      345 Sage Avenue      Palo Alto      CA94303
00006  FORMAN, ED         456 Boston St        Dallas         TX75220
00007  GREEN, TERRY       567 Doheny Dr        Hollywood      CA90046
00008  HOWSER, PETER      678 Dusty Rd         Chicago        IL60631
00009  INDERS, PER        321 Sawtelle Blvd    Tucson         AZ85702
00010  JENKINS, TED       210 Park Avenue      New York       NY10016
. LIST FOR Name = HOWSER
00008  HOWSER, PETER      678 Dusty Rd         Chicago        IL60631
```

The logical operators greatly expand the ability to refine data and manipulate records and databases. Explaining them in depth is beyond the scope of this manual, but if you are not familiar with them, most computer texts have a chapter very near the beginning that explains their use. They generate logical results (True or False). They are listed below in the order of precedence within an expression (.NOT. is applied before .AND., etc.):

```
  ()    : parentheses for grouping
 .NOT.  : boolean not (unary operator)
 .AND.  : boolean and
 .OR.   : boolean or
  $     : substring logical operator
          (substring search)
```

```
^LIST FOR (JobNumber=730 .OR. JobNumber=731);
          AND. (Bill:Date >= '791001' .AND.;
                Bill:Date <= '791031')^
```

displays all the October, 1979 records for costs billed against job numbers 730 and 731 (notice how the command line was extended with the semi-colons).

If you're not familiar with logical operators, start with the basic fact that these operators will give results that are True or False. In our example, dBASE II asks the following questions about each record:

1) Is JobNumber equal to 730 (T or F)?
2) Is JobNumber equal to 731 (T or F)?
3) Is Bill:Date greater than or equal to '791001' (T or F)?
4) Is Bill:Date less than or equal to '791031' (T or F)?

dBASE II then performs three logical tests (.OR., .AND., .AND.) before deciding whether the record should be displayed or not.

Parentheses are used as they would be in an arithmetic expression to clarify operations and relations. Because of the first .AND., dBASE II will display records only when the conditions in both parenthetical statements are true.

Evaluating the first expression, it first checks the <Job:Number> field. If the value in the field is 730 or 731, this sub-expression is set to True. If the field contains some other value, this sub-expression is False and the record will not be displayed.

If the first sub-expression is true, dBASE II must still check the contents of the <Bill:Date> field to evaluate the second sub-expression. If the contents of the field are between '791001' and '791031, inclusive, this expression is true, too, and the record will be displayed. Otherwise, the complete expression is false and dBASE II will skip to the next record, where it proceeds through the same evaluation.

Let's try some of this with <Names.DBF>. Type the following:

```
^USE Names^
^DISPLAY all FOR Zip:Code > '5' .AND. Zip:Code < 9^
^DISPLAY all FOR Name < 'F'^
^DISPLAY all FOR Address > '400' .AND. Address < '700'^
^DISPLAY all FOR Address > '400' .OR. Address < '700'^
```

```
. USE Names
. DISPLAY all FOR Zip:Code > '5' AND. Zip:Code < 9
00006  FORMAN, ED              456 Boston St.        Dallas          TX75220
00008  HOWSER, PETER           678 Dusty Rd.         Chicago         IL60631
00009  INDERS, PER             321 Sawtelle Blvd.    Tucson          AZ85702
. DISPLAY all FOR Name < 'F'
00001  ALAZAR, PAT             123 Crater Rd.        Everett         WA98206
00002  CLINKER, DUANE          789 Charles Dr.       Los Angeles     CA90036
00003  DESTRY, RALPH           234 Mahogany St.      Deerfield       FL33441
00004  EDMUNDS, JIM            392 Vicarious Way     Atlanta         GA30328
00005  EMBRY, ALBERT           345 Sage Avenue       Palo Alto       CA94303
. DISPLAY all FOR Address > '400' AND. Address < '700'
00006  FORMAN, ED              456 Boston St.        Dallas          TX75220
00007  GREEN, TERRY            567 Doheny Dr.        Hollywood       CA90046
00008  HOWSER, PETER           678 Dusty Rd.         Chicago         IL60631
. DISPLAY all FOR Address > '400' OR. Address < '700'
00001  ALAZAR, PAT             123 Crater Rd.        Everett         WA98206
00002  CLINKER, DUANE          789 Charles Dr.       Los Angeles     CA90036
00003  DESTRY, RALPH           234 Mahogany St.      Deerfield       FL33441
00004  EDMUNDS, JIM            392 Vicarious Way     Atlanta         GA30328
00005  EMBRY, ALBERT           345 Sage Avenue       Palo Alto       CA94303
00006  FORMAN, ED              456 Boston St.        Dallas          TX75220
00007  GREEN, TERRY            567 Doheny Dr.        Hollywood       CA90046
00008  HOWSER, PETER           678 Dusty Rd.         Chicago         IL60631
00009  INDERS, PER             321 Sawtelle Blvd.    Tucson          AZ85702
00010  JENKINS, TED            210 Park Avenue       New York        NY10016
```

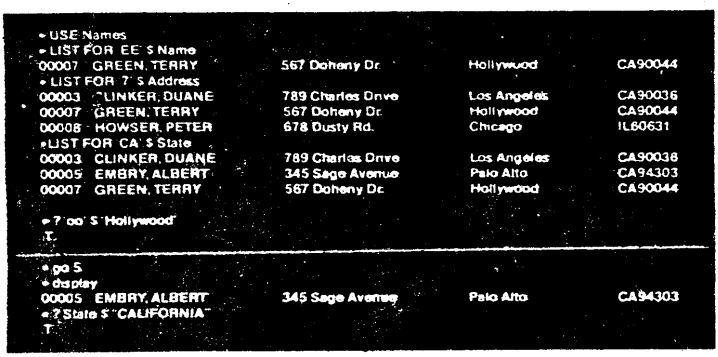
Notice what happened with the last command: all the records were displayed. If you're not familiar with logical operators, this kind of non-selective "selection" will have to guarded against.

The $ substring logical operator is extremely useful because of its powerful search capabilities. The format is:

^<substring> $ <string>^

This operator searches for the substring on the left within the string on the right. Either or both terms may be string variables as well as string constants. To see how this works, type the following:

```
^USE Names^
^LIST FOR 'EE' $ Name^
^LIST FOR '7' $ Address^
^LIST FOR 'CA' $ State^
^? 'oo' $ 'Hollywood'^
^. GO 5^
^: DISPLAY^
^? State $ "CALIFORNIA"^
```



With this function we could have, for example, simplified the structure of our mailing list names file. The states could have been entered as part of the address. To call out names within a specific state, we could have simply typed the following, where XX is the abbreviation for the state we want:

^<COMMAND> FOR 'XX' $ Address^

String operators generate string results.

+ = string concatenation (exact)
- = string concatenation (moves blanks)

Concatenation is just another one of those fancy computer buzzwords. All it really means is that one character string is stuck on to the end of another one. Type the following:

```
^USE Names^
^? Name + Address^
^? Name - Address^
^? 'The name in this record is ' + Name;
   - ' and the address is '+ Address^
```

```
. USE Names
. ? Name + Address
ALAZAR, PAT          123 Crater Rd.
. ? Name - Address
ALAZAR, PAT123 Crater Rd
. ? 'The name in this record is ' + Name; - ' and the address is ' + Address
The name in this record is ALAZAR, PAT and the address is 123 Crater Rd
```

The ^+^ and ^-^ both join two strings. The "plus" sign joins the string exactly as they are found. The "minus" sign moves the trailing blanks in a string to the end of the string. They are not eliminated, but for many purposes this is enough, as they do not show up between the strings being joined.

If you want to eliminate the trailing blanks, you can use the ^TRIM^ function. This is used by typing ^STORE TRIM(<variable>) TO <variable>^. As an example, we could have typed:^STORE TRIM(Name) TO (Name)^ to eliminate the blanks following the characters of the name.

To eliminate all of the trailing blanks in our example, we could have typed: ^STORE TRIM(Name - Address) TO Example^.

Now that we've introduced you to expressions and dBASE II operators, we'll continue with other dBASE II commands. We'll be giving you some practice in using expressions and operators as we work our way up to develpping command files.

Changing an empty database structure (MODIFY)

WARNING: the ^MODIFY^ command will destroy your database. Please follow instructions carefully.

When there is no data in your database, the **^MODIFY^** command is the fastest and easiest way to add, delete, rename, resize or otherwise change the database structure. This destroys any data in the database so don't use it after you've entered data. (Later we'll show you a way to do so, safely.)

<MoneyOut.DBF> has no data in it yet, so we'll work with it. A useful change would be to rename <JobNumber> to <Job:Nmbr> so that the abbreviation is consistent with <Emp:Nmbr> and <Bill:Nmbr>. Type the following:

^USE MoneyOut^
^LIST STRUCTURE^ (page 22)
^MODIFY STRUCTURE^
^y^ (in response to the question)

```
. use MoneyOut
. list structure
STRUCTURE FOR FILE: MONEYOUT.DBF
NUMBER OF RECORDS: 00000
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
FLD     NAME        TYPE    WIDTH     DEC
001     CLIENT      C       004
002     JOBNUMBER   C       003
003     BILL DATE   C       006
004     SUPPLIER    C       028
005     DESCRIP     C       010
006     HOURS       N       006       002
007     EMP:NMBR    C       002
008     AMOUNT      N       009       002
009     BILL:NMBR   C       006
010     CHECK:NMBR  C       005
011     CHECK DATE  C       006
**TOTAL**                   00086

. modify structure
MODIFY ERASES ALL DATA RECORDS  PROCEED? (Y/N) y
```

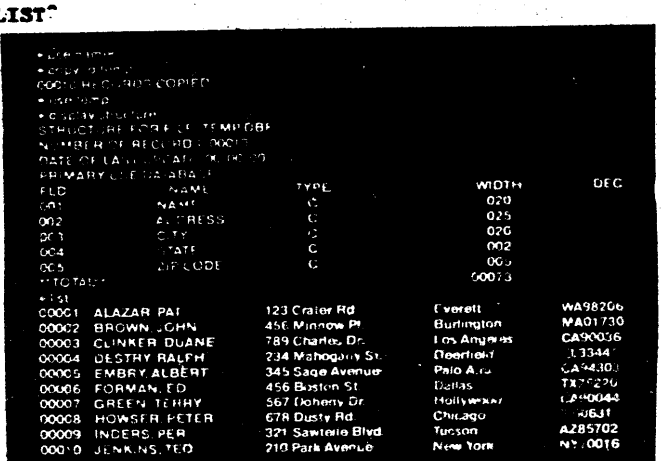
dBASE II erases the screen and lists the first 16 (or fewer) fields in the database. Use ^Ctl-X^ to move down one field. Just type in the new field name over the old one (use a space to blank out the extra letter).

You can exit **^MODIFY^** in either of two ways: **ctl-W** changes the structure on disk, then resumes normal dBASE II operation (^ctl-O^ for Superbrain). **ctl-Q** quits and returns to normal dBASE II operation without making the changes. This actually gets you back without destroying the database, but play it safe and have a backup file (see next page).

Duplicating databases and structures (COPY)

Duplicating a file without going back to your computer operating system is straightforward. Type the following:

```
^USE Names^
^COPY TO Temp^
^USE Temp^
^DISPLAY STRUCTURE^
^LIST^
```



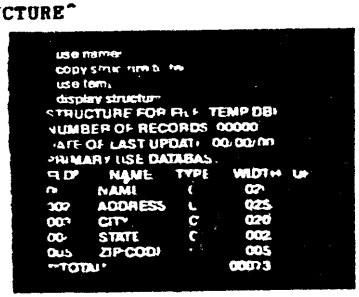
Warning: When you ^COPY^ to an existing filename, the file is written over and the old data is destroyed.

^COPY TO TEMP^ created a new database called <Temp.DBF>. It is identical to the <Names.DBF>, with the same structure and the same data. The command can be expanded even further:

```
^COPY TO <filename> [STRUCTURE] [FIELD list]
```

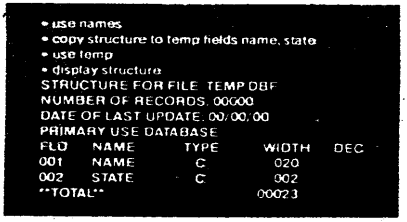
With this command, you can copy only the structure or some of the structure to another file. Type the following:

```
^USE Names^
^COPY TO Temp STRUCTURE^
^USE Temp^
^DISPLAY STRUCTURE^
```



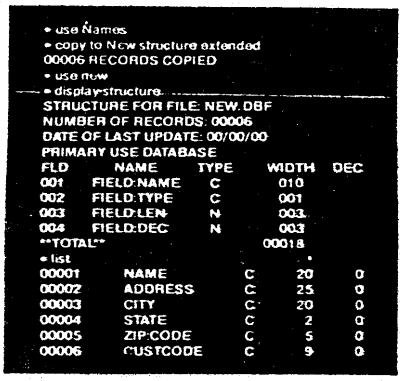
We can copy a portion of the structure by listing only the fields we want in the new database. Type:

```
^USE Names^
^COPY TO Temp STRUCTURE FIELDS Name, State^
^USE Temp^
^DISPLAY STRUCTURE^
```



FOR ADVANCED PROGRAMMERS: COPY can also be used to give your program access to a database structure. Type:

```
^USE Names^
^COPY TO New STRUCTURE EXTENDED^
^USE New^
^LIST^
```



The <New.DBF> database records describe the <Names> database structure, and an application program has direct access to this information (see Review.CMD, Section VI).

Alternatively, a file with the same structure as <New.DBF> could be embedded in a program so that the operator could enter the structure for a file without learning dBASE II. The program would then create the database for him with the following command:

```
^CREATE <datafile> FROM <structurefile>^
```

Adding and deleting fields with data in the database

As you expand the applications for dBASE II, you'll probably want to add or delete fields in your databases.

^MODIFY STRUCTURE^ alone would destroy all the data in your database, but used with ^COPY^ and ^APPEND^, it lets you add and delete fields at will.

The strategy consists of copying the structure of the database you want to change to a temporary file, then making your modifications on that file. After that is done, you bring in the data from the old file into the new modified structure.

As an example, we'll use our <Names> file and our <Orders> file. At some point, it would be useful to list the orders placed by a given customer. This could be done easily by adding a customer number field to <Names> file to match the field in the <Orders> file. To do so without destroying the records we already have, type the following:

```
^USE Names^
^COPY TO Temp STRUCTURE^
^USE Temp^
^MODIFY STRUCTURE^
^y^                          (in answer to the prompt)
```

Use the Full Screen Editing features to move down to the first blank field and type in the changes in the appropriate columns (name is "CustNmbr", data type is "C", length is 9). Now type ^ctl-W^ (^ctl-O^ with Superbrain) to save the changes and exit to the dBASE II dot prompt.

^DISPLAY STRUCTURE^ to make sure that it's right. If it is we can add the data from <Names> by typing:

```
^APPEND FROM Names^
```

We could also have changed field sizes: the ^APPEND^ command transfers data to fields with matching names.

```
. display structure
STRUCTURE FOR FILE: TEMP.DBF
NUMBER OF RECORDS: 00010
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
FLD   NAME       TYPE   WIDTH  DEC
001   NAME       C      020
002   ADDRESS    C      025
003   CITY       C      020
004   STATE      C      002
005   ZIPCODE    C      005
006   CUSTNMBR   C      009
**TOTAL**               00082
```

Our new file <Temp> should now have the new field we wanted to add and all of the old data. ^DISPLAY STRUCTURE^ then ^LIST^ to make sure that a power line glitch or a bad spot on the floppy hasn't messed anything up.

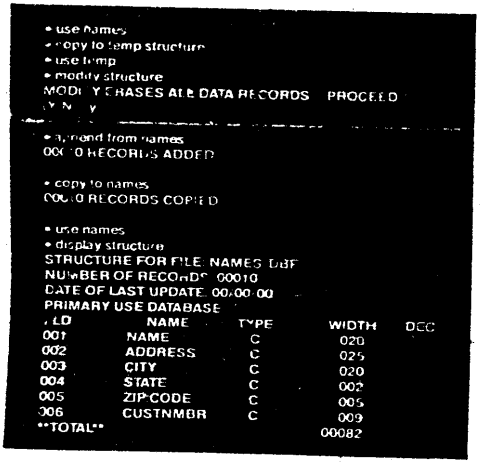
If the data got transferred correctly, we can finish up by typing:

```
^COPY TO Names
^USE Names^
```

The ^COPY^ command writes over the old structure and data. After displaying and listing the new <Names> file, you can ^DELETE FILE Temp^.

To summarize, the procedure can be used to add or delete fields in a database in the following sequence:

```
^USE <oldfile>^
^COPY TO <newfile> STRUCTURE^
^USE <newfile>^
^MODIFY STRUCTURE^
^APPEND FROM <oldfile>^
^COPY TO <oldfile>^
```



Dealing with CP/M and other "foreign" data files (more on COPY and APPEND)

dBASE II information can be changed into a form that is compatible with other processors and systems (BASIC, PASCAL, FORTRAN, PL/1, etc.). dBASE II can also read data files that have been created by these processors.

With CP/M, the standard data format (abbreviated as SDF in dBASE II) includes a carriage return and line feed after every line of text. To create a compatible data file (for wordprocessing, for example) from one of your databases, you use another form of the ^COPY^ command. Type:

```
^USE Names^
^COPY TO SysData SDF^
```

This command creates a file called <SysData.TXT>. Now ^QUIT^ dBASE II and use your word processor to look at the file. You'll find that you can work with it exactly as if you had created it under CP/M.

The Standard Data Format also allows dBASE II to work with data from CP/M files. However, the data must match the structure of the database that will be using it.

If we had used a wordprocessor to create a file called <NewData.TXT>, we could add it to the <Names.DBF> file with this command. NOTE: the spacing of the data must match the structure of the database. If the <NewData.TXT> file contained the following information:

```
FREITAG, JEAN       854 Munchkin Ave.     Houston     TX77006
GOULD, NICOLE       73 Radnor Way         Radnor      PA19089
PETERS, ALICE       676 Wacker Dr.        Chicago     IL60606
GREEN, FRANK        441 Spicer Ave.       Tampa       FL33622

    (20)                (25)                (20)        (2) (5)
```

we would add it to the <Names> file by typing the following:

```
^USE Names^
^APPEND FROM NewData.TXT SDF^
```

Adding data to an existing file from a system file takes only seconds.

The procedure is similar if your "foreign" files use different delimiters. A common data file format uses commas between fields and single quotes around strings to delimit the data. To create or use these types of data files, use the word **DELIMITED** instead of **SDF**. To see how this works, type:

^COPY TO Temp DELIMITED^

then go back to your operating system to look at your data.

If your system has a different delimiter, you can specify it in the command: **^DELIMITED [WITH <delimiter>]^** (do NOT type the "<" and ">" symbols). If your system uses only commas and nothing around strings, use: **^DELIMITED WITH ,^**.

The full forms of **^COPY^** and **^APPEND^** for working with system data files are:

```
                                          [SDF         ]
COPY [scope] TO <filename> [FIELD list] [STRUCTURE ]  [FOR <expression>]
                                          [DELIMITED [WITH <delimiter>]]

APPEND FROM <filename.TXT> [SDF         ]  [FOR <expression>]
                           [DELIMITED [WITH <delimiter>]]
```

Both commands can be made selective by using a conditional expression, and the scope of **^COPY^** can be specified as for other dBASE II commands.

NOTE: While dBASE II automatically generates extensions for files it creates, you must specify the ".TXT" filename extension when APPENDing from a system data file.

NOTE: With the APPEND command, any fields used in the <expression> must exist in the database to which the data is being transferred.

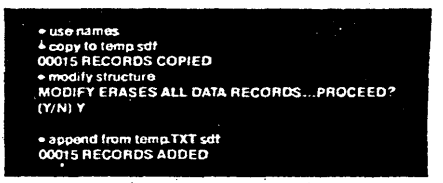
Renaming database fields with COPY and APPEND

As we sad earlier, ^APPEND^ transfers data rrom one file to another for matching fields. If a field name in the FROM file is not in the file in USE, the data in that field will not be transferred.

However, the full form does allow you to transfer only data, and we can use this feature to rename the fields in a database. If we wanted to rename <CustNmbr> to <CustCode> in <Names.DBF>, we would type:

```
^USE Names^
^COPY TO Temp SDF^              (data only to Temp.TXT)
^MODIFY STRUCTURE^
^APPEND FROM Temp.TXT SDF^      (after changing field name)
```

Now when you ^DISPLAY STRUCTURE^, the last field will be called <CustCode>. Don't forget to change the name of the <CustNmbr> field in our <Orders> database so that the fields match.



Data in a <.TXT> file created by using the SDF (or DELIMITED) option is kept in columns that are spaced like the fields were in the original file. While you can edit a <.TXT> file with your word processor, this can be dangerous:

Warning: Do not change field positions or sizes: the data you saved is saved by position, not by name! If you change the field sizes when you modify the structure, you will destroy your database when you bring the saved data back into it.

When you ^COPY^ data to a <.TXT> file, you can use the full command to specify the scope, fields and conditions (see earlier explanation).

Modifying data rapidly (REPLACE, CHANGE)

Changes can be made rapidly to any or all of the records using the following command:

^REPLACE [scope] <field> WITH <data> [, <field> WITH <data>,...]
[FOR <expression>]^

This is an extremely powerful command because it REPLACES a "<field-that-you-name> WITH <whatever-you-write-in-here>". You can REPLACE more than one field by using a comma after the first combination, then listing the new fields and data as shown in the center brackets.

The "data" can be specific new information (including blanks), or it could be an operation, such as deducting state sales tax from all your bills because you have a resale number (REPLACE all Amount WITH Amount/1.06).

You can also make this replacement conditional by using the FOR and specifying your conditions as an expression.

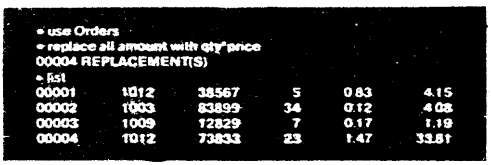
To show you how this works, we need to add some data to both the <Names> and <Orders> database files.

First, **^USE Name^** then type **^EDIT 1^**. Now enter a ^1001^ in the <CustCode> field, using the full screen editing features to get into position. Use **^ctl-C^** to move on to the next record when you are finished customer codes should be entered as four-digit numbers, with the record number as the last two digits (1001, 1002, 1003, etc.)

Now **^USE Orders^** and **^APPEND^** the following order information (do not type the column headings):

(Cust)	(Item)	(Qty)	(Price)^
^1012	38567	5	.83^
^1003	83899	34	.12^
^1009	12829	7	.17^
^1012	73833	23	1.47^

^USE Orders^
^REPLACE All Amount WITH Qty*Price
^LIST^



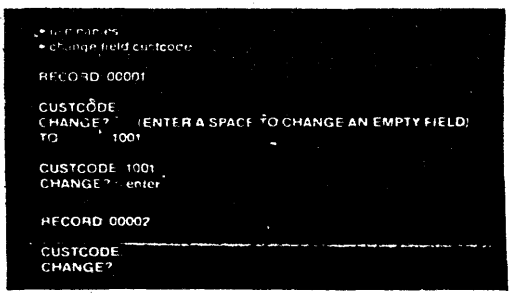
You'll also find **^REPLACE^** useful in command files to fill in a blank record that you have appended to a file. Data from memory variables in your program is frequently used to fill in the blank fields.

Changes to a few fields in a large number of records can also be made rapidly by using:

^CHANGE [scope] FIELD <list> [FOR <expression>]^

The "scope" is the same as for other dBASE II commands. At least one field must be named, but several field names can be listed if separated by commas. This command finds the first record that meets the conditions in the "expression", then displays the record name and contents with a prompt. To change the data in the field, type in the new information. To leave it the way it was, hit <enter>. If the field is blank and you want to add data, type a space.

Once you have looked at all the listed fields within a record, you are presented with the first field of the next record that meets the conditions you set. To return to dBASE II, hit the ^ESCAPE^ key.



Organizing your databases (SORT, INDEX)

Data is frequently entered randomly, as it was in our <Names> database. This not necessarily the way you want it. so dBASE II includes tools to help you organize your databases by **SORTING** and **INDEXING** it.

INDEXED files allow you locate records quickly (typically within two seconds even with floppy disks).

Files can be sorted in ascending or descending order. The full command is:

^SORT ON <fieldname> TO <filename> [DESCENDING]

The <fieldname> specifies the key on which the file is sorted and may be character or numeric (not logical). The sort defaults to ascending order, but you can over-ride this by specifying the descending option.

To sort on several keys, start with the least important key, then use a series of sorts leading up to the major key. During sorting, dBASE II will move only as many records as it must.

To sort our <Names> file so that the customers are in alphabetical order, type:

^USE Names^
^SORT ON Name TO Temp^
^USE Temp^
^LIST^
^COPY TO Names^

```
. use names
. sort on name to temp
SORT COMPLETE
. use temp
. list
00001  ALAZAR, PAT       123 Crater Rd.        Everett      WA982061001
00002  BROWN, JOHN       456 Minnow Pl.        Burlington   MA017301002
00003  CLINKER, DUANE    789 Charles Dr.       Los Angeles  CA900361003
00004  DESTRY, RALPH     234 Mahogany St.      Deerfield    FL334411004
00005  EMBRY, ALBERT     345 Sage Avenue       Palo Alto    CA943031005
00006  FORMAN, ED        456 Boston St.        Dallas       TX752201006
00007  FREITAG, JEAN     854 Munchkin Ave.     Houston      TX770061017
00008  GOULD, NICOLE     73 Radnor Way         Radnor       PA190891012
00009  GREEN, FRANK      441 Spicer Ave.       Tampa        FL33622
00010  GREEN, TERRY      567 Doheny Dr.        Hollywood    CA900441007
00011  HOWSER, PETER     678 Dusty Rd.         Chicago      IL606311008
00012  INDERS, PER       321 Sawtelle Blvd.    Tucson       AZ857021009
00013  JENKINS, TED      210 Park Avenue       New York     NY100161010
00014  PETERS, ALICE     676 Wacker Dr.        Chicago      IL60606
. copy to names
00014 RECORDS COPIED
```

WARNING: Do not SORT a database to itself. A power line "glitch" could destroy your entire database if it came along at the wrong moment.

Instead, sort to a temporary file, then **^COPY^** it back to the original file name after you've confirmed the data.

A database can also be INDEXED so that it appears to be sorted. The form of the ^INDEX^ command is:

^INDEX ON <key (variable/expression)> TO <index filename>^

This creates a file with the new name and the extension <.NDX>. Only the data within the "key" is sorted, although it appears that the entire database has been sorted. The key may be a variable name or a complex expression up to 100 characters long. It cannot be a logical field. To organize our customer database by ZIP code, type:

^USE Names^
^INDEX ON Zip:Code TO Zips^
^USE Names INDEX Zips^
^LIST^

We could also index our database on three keys by typing:

^INDEX ON Name + CustCode + State TO Compound^

Numeric fields used in this manner must be converted to character type. If CustCode were a numeric field with 5 positions and 2 decimal places, ^STR^ function (described later) performs the conversion like this:

^INDEX ON Name + STR(CustCode,5,2) + State TO Compound^

To take advantage of the speed built into an INDEX file, you have to specify it as part of the ^USE^ command:

^USE <database name> INDEX <index filename>^

Positioning commands (GO, GO BOTTOM, etc.) given with an INDEX file in use move you to positions on the index, rather than the database. ^GO BOTTOM^, for example, will position you at the last record in the index rather than the last record in the database.

Changes made to key fields when you ^APPEND^, ^EDIT^, ^REPLACE^ or ^PACK^ the database, are reflected in the index file in USE.

Other index files for your database can be updated by typing: ^SET INDEX TO <index File 1>, <index File 2>, ...<index File n>^. Then perform your ^APPEND^, ^EDIT^, etc. All named index files will now be current.

A major benefit of an INDEXED file is that it allows you to use the ^FIND^ command (described next) to locate records in seconds, even with large databases.

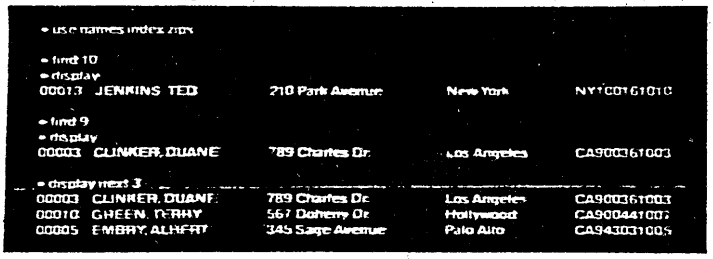
Finding the information you want (FIND, LOCATE)

If you know what data you are looking for, you can use the FIND command (but only when your database is indexed, and the index file is in USE). A typical FIND time is two seconds with a floppy disk system.

Simply type FIND <character string> (without quote marks), where the "character string" is all or part of the contents of a field.

This string can be as short as you like, but should be long enough to make it unique. "th", for example, occurs in a large number of words; "theatr" is much more limited. Type the following:

```
^USE Names INDEX Zips^
^FIND 10^
^DISPLAY^
^FIND 9^
^DISPLAY^
^DISPLAY Next 3^
```



If the key is not unique, dBASE II finds the first record that meets your specifications. This may or may not be the one you're looking for. If no record exists with the identical key that you are looking for, dBASE II displays **NO FIND.**

^FIND^ can also be used with files that have been INDEXED on multiple keys. The disadvantage of a compound key (which may not be a disadvantage in your application) is that it must be used from the left when you access the data. That is, you can access the data by using the FIND command and just the Name, or the Name and CUstCode, or all three fields, but could not access it using the State or CustCode alone. To do that, you would either have to use the LOCATE command (next), or have another file indexed on the State field as the primary key.

When looking for specific kinds of data, use

^LOCATE [scope] [FOR <expression>]^

This command is used when you are looking for specific data in a file that is not indexed on the key you are interested in (file is indexed on zip codes, but you're interested in states, etc.)

If you want to search the entire database, you do not have to specify the scope, as **^LOCATE^** starts on the first record. To search part of a file, use **^LOCATE Next <number>^**. The search will start at the record the pointer is on and look at the next "number" of records. If this would move the pointer past the end of the file, **LOCATE** examines every record from the pointer position to the end of the file.

If you are looking for data in a character field, the data should be enclosed in single quotes. Type the following:

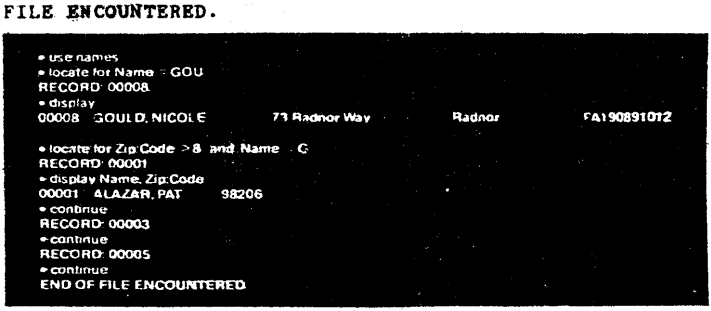
```
^USE Names^
^LOCATE FOR Name='GOU'^
^DISPLAY^
^LOCATE FOR Zip:Code>'8' .AND.  Name < 'G'^
^DISPLAY Name, Zip:Code^
```

If a record is found that meets the conditions in your expression, dBASE II signals you with: **RECORD n.** You can display or edit the record once it is located.

If there may be more than one record that meets your conditions, type **CONTINUE** to get the next record number.

```
^CONTINUE^
^CONTINUE^
^CONTINUE^
```

If dBASE II cannot find your record within the "scope" that you defined, it will display: **END OF LOCATE** or **END OF FILE ENCOUNTERED.**



Getting information out of all that data (the REPORT command)

FIND and **LOCATE** are fine for locating individual records and data items, but in most applications you will want data summaries that include many records that meet certain specifications. The ^**REPORT**^ command lets you do this quickly and easily.

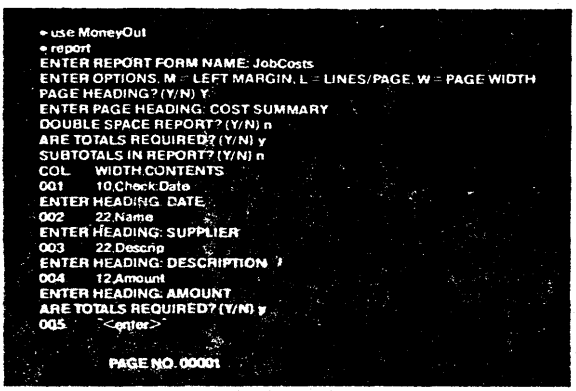
If you are using single sheets of paper in your printer, first type ^**SET EJECT OFF**^ to turn the initial formfeed off. Now select the database you want the report from and create your custom report format by typing:

```
^SET EJECT OFF^
^USE <database>^
^REPORT^
```

dBASE II then leads you through a series of prompts to create a custom format for the report. You specify which fields from the database you want, report and column headings, which columns should be totalled, etc. The standard defaults are 8 columns from the left edge of the paper for the page offset, 56 lines per page, and a page width of 80 characters.

You can try this with the files you've created on the demonstration disk, but the <Names> and <Orders> databases that we've used as examples so far don't have enough data in them to really show you how powerful dBASE II can be. For our examples from here on we will be using <MoneyOut.DBF> and other databases that are part of an existing business system. (The entire system is in Section VI, including database structures and the command files that run it.)

This would be a good time for you to create a database structure that you would actually use in your business. Enter data in it, then substitute it for <MoneyOut> in our examples.



When you have defined all the contents of the report, hit <enter> when prompted with the next field number. dBASE II immediately starts the report to show you what you have specified, and will go through the entire database if you let it. To stop the report, hit the <escape> key.

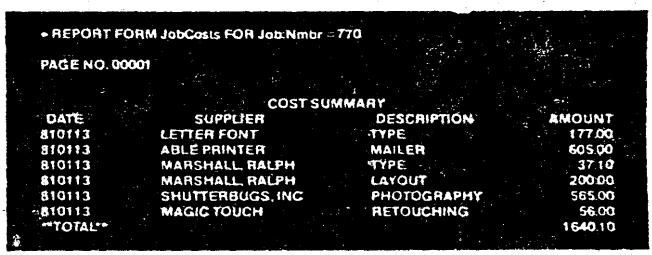
At the same time, dBASE II saves the format in a file with the extension .FRM, so that you can use it without having to go through the dialog again. The full form of the command is:

^REPORT FORM <formname> [scope] [FOR <expression>] [TO PRINT]^

By typing

^REPORT FORM JobCosts FOR Job:Nmbr='770'^

we can get a listing of all the job costs for job number 770 without having to redefine the format.



```
• REPORT FORM JobCosts FOR Job:Nmbr = 770

PAGE NO. 00001

                          COST SUMMARY
DATE        SUPPLIER              DESCRIPTION        AMOUNT
810113      LETTER FONT           TYPE                 177.00
810113      ABLE PRINTER          MAILER               605.00
810113      MARSHALL, RALPH       TYPE                  37.10
810113      MARSHALL, RALPH       LAYOUT               200.00
810113      SHUTTERBUGS, INC      PHOTOGRAPHY          565.00
810113      MAGIC TOUCH           RETOUCHING            56.00
**TOTAL**                                             1640.10
```

You can change the information in the heading by typing **^SET HEADING TO character string^** (up to 60 characters and spaces, no quote marks). The "scope" defaults to "all" when not specified.

The expression could have been expanded with other conditions, and the entire report could have been prepared as a hardcopy by adding TO PRINT at the end of the command.

This report capability can be used for just about any business report, from accounts payable (FOR Check:Nmbr=' '), to auto expenses (FOR Job:Nmbr='4 ') to anything else you need.

Automatic counting and summing (COUNT, SUM)

In some applications, you won't need to see the actual records, but will want to know how many meet certain conditions, or what the total is for some specified condition (How many widgets do we have in stock? How many are on back order? What is the total of our accounts payable?)

For counting use:

^COUNT [scope] [FOR conditions] [TO memory variable]^

This command can be used with none, some or all of the modifiers.

Unqualified, it counts all the records in the database. The "scope" can be limited to one or a specified number of records, and the "condition" can be any complex logical expression (see earlier section on expressions). The result of the count can be stored in a memory variable, which is created when the command is executed if it did not exist.

To get totals, use:

^SUM field(s)[scope][FOR condition][TO memory variable(s)]^

You can list up to 5 numeric fields to total in the database in USE. If more than one field is to be totaled, the field names are separated by commas. The records totaled can be limited by using the "scope" and/or conditional expressions after the FOR (Client <> 'SEM' .AND. Amount > 10...).

If memory variables are used (separated by commas), remember that totals are stored based on position. If you don't want to store the last fields in memory variables but do want to see what the amounts are, there's no problem: simply name the first few variables that you want. If there's a gap (you want to save the first, third and fourth field totals out of six), name memory variables for the first four fields then RELEASE the second one after the SUM is done.

```
• USE MoneyOut
• COUNT FOR Amount < 100 TO Small
COUNT = 00067
• SUM Amount FOR Job Nmbr = 770 TO Cost
  1640.10
• display memory
SMALL                    (N)                67
COST                     (N)              1640.10
**TOTAL**   02 VARIABLES USED   00012 BYTES USED
```

Summarizing data and eliminating details (TOTAL)

^TOTAL^ works similarly to the sub-total capability in the REPORT command except that the results are placed in a database rather than being printed out:

TOTAL ON <key> TO <database> [FIELDS list] [FOR conditions]

NOTE: The database that the information is coming from must be presorted or indexed on the key that is used in this command.

This command is particularly useful for eliminating detail and providing summaries. The screen shows what happens with our <MoneyOut> database:

```
^USE MoneyOut^
^INDEX ON Job:Nmbr TO Jobs^
^USE MoneyOut INDEX Jobs^
^TOTAL ON Job:Nmbr TO Temp FIELDS Amount FOR Job:Nmbr >699;
                                    .AND. Job:NMbr < 800^
^USE Temp^
^LIST^
```

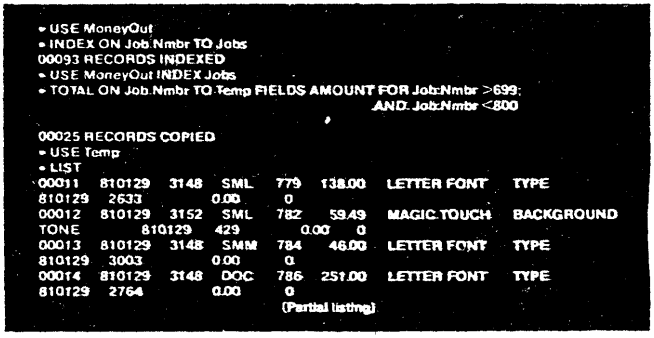
The new database has one entry for each job number, and a total for all the costs against that job number in our <MoneyOut> database. One problem with the new database, however, is that only two of the fields contain useful information.

This can be handled with one more command line. ^TOTAL^ transfers all the fields if the database named did not exist, but uses the structure of an existing database. In the commands above, we could have limited the fields in the new database by creating it first, before we used the ^TOTAL^ command:

```
^COPY TO Temp FIELDS Job:Nmbr, Amount^
```

Now when we ^TOTAL^ to <Temp>, the new database will contain only the job numbers and cotals. Try it with your database.

This same technique can be used to summarize quantities of parts, accounts receivable or any other ordered (SORTed or INDEXed) information.



Section II Summary

This section has broadened the scope of what you can now do with dBASE II.

We have shown you how different operators (arithmetic, relational and string) can be used to modify dBASE II commands to give you a greater degree of control over your data than is possible with other database management systems.

Since data structures are the basis of database systems, we have covered a number of different ways in which you can alter the these structures, with or without data in the database.

We have also shown you how to enter, alter and find the specific information you may be looking for. We have also introduced new global commands that make it possible for you to turn all that data into information with a single command (COUNT, SUM, REPORT, TOTAL).

In the next section, we will show you how to set up dBASE II command files (programs), so that you can automate your information processes.

Section III:

Setting up a command file (writing your first program).....	62	MODIFY COMMAND <file>
Making choices and decisions.....	64	IF..ELSE..ENDIF
Repeating a process.....	66	DO WHILE..
Procedures (subsidiary command files).....	67	DO <file>
Entering data interactively during a run.....	68	WAIT, INPUT, ACCEPT
Placing data and prompts exactly where you want them.....	69	@..SAY..GET
A command file that summarizes what we've learned..	72	
Working with multiple databases.....	75	SELECT PRIMARY/SECONDARY
Generally useful system commands and functions.....	76	
A few words about programming and planning your command files.....	77	

If you understand how to write expressions, you are very close to being able to write programs.

There are four basic programming structures that you can use to get a computer to do what you want to do:

- **Sequence**
- **Choice/Decision**
- **Repetition**
- **Procedures**

You've already seen that dBASE II processes your commands sequentially in the order in which you give them. In this section we'll explain how you make choices (IF...ELSE), how you can make the computer repeat a sequence of commands (DO WHILE..), and how to use sub-files of commands (procedures).

Then we'll show you how to use these simple tools to write command files (programs) that will solve your applications problems.

Setting up a command file (writing your first program)

The commands we've introduced so far are powerful and can accomplish a great deal, yet only scratch the surface of the capabilities of dBASE II. The full power comes into play when you set up command files so that the commands you enter once can be repeated over and over.

When you create a command file you are programming the computer, but since dBASE II uses English-like commands, it's a lot simpler than it sounds. Also, because dBASE II is a relational database management system, you work with increments of data and information, rather than bits and bytes.

To set up a command file, you list the commands you want performed in a CP/M file with a <.CMD> extension to its name, using a text editor or word processor.

dBASE II starts at the top of the list and processes the commands one at a time until it is done with the list.

Other computer languages operate exactly the same way. In BASIC the sequence is very visible because each program line is numbered. In other languages (dBASE II among them), the sequence is implied and the computer will process the first line on the page, then the second line, etc. Some languages use separators (such as colons) between command statements; dBASE II simply uses the carriage return to terminate the command line.

The only time the sequence is not followed is when the computer is specifically told to go and do something else. Usually, this is based on some other conditions and the computer must make a decision based on expressions or conditions that you have set up in the command file. We'll tell you more about this later.

For now, let's create a command file called <Test>.

You can do this using a text editor or wordprocessor, but there's an easier way with dBASE II. Type:

^MODIFY COMMAND Test^

dBASE II now presents you with a blank screen that you can write into using the full screen editing features described earlier. Use them now to enter the short program at the top of the next page (do not type the "^" symbols).

The end of a line indicates the end of a command (unless you use a semicolon), so keep the list of commands as shown on the next page.

```
^USE Names^
^COPY Structure TO Temp FIELDS Name, ZipCode^
^USE Temp^
^APPEND FROM Names^
^COUNT FOR Name = 'G' TO G
^DISPLAY MEMORY^
^? 'We have just successfully completed our first command file.'
```

When you're finished, use clt-W (ctl-O with Superbrain) to get back to the dBASE II prompt. Now type:

```
A> ^dBASE Test^
```

If you typed the program in exactly the way it was printed, it crashed. Now type **^MODIFY COMMAND Test^** again and insert a semi-colon to correct the <Zip:Code> field name.

Once you get to writing larger command files of your own, you'll find that this built-in editor is one of the most convenient features of dBASE II, since you can write, correct and change programs without ever having to go back to the system level of the computer. Currently, this built-in editor can back up only about 5,000 lines, so editing should be planned in one direction for larger files.

The command file itself is trivial but does show you how you can perform a sequence of commands from a file with a single system command. This is similar to the way you use .COM files in your operating system.

If you are already in dBASE II (with the dot prompt), you type:

```
. ^DO <filename>^
```

where <filename> has the <.CMD> extension.

TIP: You may want to rename the main dBASE file to <DO.COM>, so that you can type **^DO <filename>^** whether you're in your system or in dBASE II. To do this with CP/M, type: **A> ^REN DO.COM=dBASE.COM^**

Making choices and decisions (IF..ELSE)

Choices and decisions are made in dBASE II with ^IF..ELSE..ENDIF^. This is used much as it is used in ordinary English: IF I'm hungry, I'll eat, (OR) ELSE I won't. With a computer, you use the identical construction, but do have to use exactly the words that it understands.

Simple decision: If only a single decision is to be made, you can drop the ELSE and use this form:

```
IF condition [.AND. cond2 .OR.  cond3 ....]
    do this command
    [cmd2]
    [....]
ENDIF
```

The "condition" can be a series of expressions (up to a maximum of 254 characters) that can be logically evaluated to being true or false. Use the logical operators to tie them together. Using our <MoneyOut> file, we might set up the following decision:

```
IF Job:Nmbr = '730' .AND.  Amount.  > 99.99;
       .OR.  Supplier = 'MAGIC TOUCH';
       .OR.  Bill:Date > '791231'
    do this command
    [cmd 2]
    [ ... ]
ENDIF
```

If all the conditions are met, the computer will perform the commands listed between the IF and the ENDIF (in sequence), then go on to the next statement following the ENDIF. If the conditions are not met, the computer skips to the first command following the ENDIF.

Two choices: If there are two alternate courses of action that depend on the condition(s), use the IF..ELSE statement this way:

```
IF condition(s)
   do command(s) 1
ELSE
   do command(s) 2
ENDIF
```

The computer does either the first set of commands or the second set of commands, then skips to the command following the ENDIF.

Multiple choice: Frequently, you have to make a choice from a list of alternatives. An example might be a the use of a screen menu to select one of several different procedures that you want to perform. In that case, you use the IF..ELSE..IF construction.

This is the same IF..ELSE that we've described, but you use it in several levels (called "nesting"), as shown below.

```
IF conditions 1
   do commands 1
ELSE
   IF conditions 2
      do commands 2
   ELSE
      IF conditions 3
         do commands 3
      ELSE
         .
         .
         .
      ENDIF 3
   ENDIF 2
ENDIF 1
```

This structure can be nested as shown as far as it has to be to choose the one set of commands required from the list of alternatives. It is used frequently in the working accounting system at the end of Part I.

Notice that each IF must have a corresponding ENDIF or your program will bomb.

TIP: dBASE II does not read the rest of the line after an ENDIF, so you can add in any identification you want to, as we did above. It helps keep things straight.

Repeating a process (DO WHILE..)

Repetition is one of the major advantages of a computer. It can continue with the same task over and over without getting bored or making mistakes because of the monotony. This is handled in most computer languages with the **DO WHILE** construction:

```
DO WHILE conditions
   do command(s)
ENDDO
```

While the conditions you specify are logically true, the commands listed will be performed.

Tip: Remember that these commands must change the conditions eventually, or the loop will continue forever.

When you know how many times you want the process repeated, you use the structure like this:

```
STORE 1 TO Index                 * Start counter at 1
DO WHILE Index < 11              * Process 10 records
   IF Item = ' '                 * If there is no data,
      SKIP                       * skip the record and
      LOOP                       * go back to the DO WHILE,
   ENDIF blank                   * without doing ProcessA
   DO ProcessA                   * Do file ProcessA.CMD
   STORE Index+1 TO Index        * Increase counter by 1
ENDDO ten times
```

In this example, if there is data in the <Item> field, the computer performs whatever instructions are in another command file called ProcessA.CMD, then returns to where it was in this command file. It increases the value of the variable Index by 1, then tests to see if this value is less than 11. If it is, the computer proceeds through the DO WHILE instructions again. When the counter passes 10, the computer skips the loop and performs the next instruction after the **ENDDO**.

The **LOOP** instruction is used to stop a sequence and cause the computer to go back to the start of a **DO WHILE** that contains the instruction.

In this case, if the Item field is blank, the record is not processed because the **LOOP** command moves the computer back to the **DO WHILE Index < 11**. The record with the blank is not counted, since we bypass the command line where we add 1 to the counter.

The problem with **LOOP** is that it short-circuits program flow, so that it's extremely difficult to follow program logic. The best solution is to avoid the **LOOP** instruction entirely.

Procedures (subsidiary command files)

The ability to create standard procedures in a language greatly simplifies programming of computers.

In BASIC, these procedures are called sub-routines. In Pascal and PL/I, they are called procedures. In dBASE II they are command files that can be called by a program that you write.

In our previous example, we called for a procedure when we said DO ProcessA. "ProcessA" is another command file (with a .CMD extension to its name). The contents of this command file might be:

```
IF Status = M
   DO PayMar
ELSE
   IF Status = S
      DO PaySingle
   ELSE
      IF Status = H
         DO PayHouse
      ENDIF
   ENDIF
ENDIF
RETURN
```

Once again, we can call out further procedures which can themselves call other files. Up to 16 command files may be open at a time, so if a file is in USE, up to 15 other files can be open. Some commands use additional files (REPORT, INSERT, COPY, SAVE, RESTORE and PACK use one additional file; SORT uses two additional files).

This is seldom a limitation, however, since any number of files can be used if they are closed and no more than 16 are open at any time.

A file is closed when the end of the file is reached, or when the ^RETURN^ command is issued by a command file. The RETURN command returns control to the command file that called it (or to the keyboard if the file was run directly).

The RETURN command is not always strictly necessary, as control returns to the calling file when the end of a file is encountered, but it is good programming practice to insert it at the end of all your command files.

Big tip: Notice that the command lines are indented in our examples. This is not necessary, but it increases command file clarity tremendously, especially when you have nested structures within other structures. Using all uppercase for the commands, and both upper- and lowercase for the variables helps, too.

Entering data interactively during a run (WAIT, INPUT, ACCEPT)

For many applications, the command files will have to get additional data from the operator, rather than just using what is in the databases.

You command files can be set up so that they prompt the operator with messages that indicate the kind of information that is needed. One good example is a menu of functions from which one is selected. Another use might be to help ensure that accounting data is entered correctly. The following commands can do this.

^WAIT [TO memory variable]^

halts command file processing and waits for a single character input from the keyboard with a WAITING prompt. Processing continues after any key is pressed (as with the dBASE II DISPLAY command).

If a variable is also specified, the input character is stored in it. If the input is a non-printable character (<enter>, control character, etc.), a blank is entered into the variable.

^INPUT ['prompt'] TO memory variable^

accepts any data type from the CRT terminal to a named memory variable, creating that variable if it did not exist.

If the optional prompting message (in single or double quotes, but both delimiters the same) is used, it appears on the user terminal followed by a colon showing where the data is to be typed in. The data type of the variable (character, numeric or logical) is determined by the type of data that is entered. Character strings must be entered in quotes or square brackets.

^ACCEPT ['prompt'] TO memory variable^

accepts character data without the need for delimiters. Very useful for long input strings.

Tips on which to use when:

WAIT can be used for rapid entry (reacts instantly to an input), but should not be used when a wrong entry can do serious damage to your database.

ACCEPT is useful for long strings of characters as it does not require quote marks. It should also be used for single character entry when the need to hit <enter> can improve data integrity.

INPUT accepts numeric and logical data as well as characters, can be used like ACCEPT.

Placing data and prompts exactly where you want them (@..SAY..GET)

The ^?^, ^ACCEPT^ and ^INPUT^ commands can all be used to place prompts to the operator on the screen.

Their common drawback for this purpose is that the prompts will appear just below the last line already on the screen. This works, but there's a better way.

If your terminal supports X-Y cursor positioning, another dBASE II command lets you position your prompts and get your data from any position you select on the screen:

```
^@ <coordinates> [SAY <'prompt'>]^
```

This will position the prompt (entered in quotes or square brackets) at the screen coordinates you specify. The coordinates are the row and column on the CRT, with 0,0 being the upper left-hand "home" position. If we specified "9,34" as the coordinates, our prompt would start on the 10th row in the 35th column.

Note: If you installed half intensity or reverse video, the prompt will be at half intensity or in reverse video. To disable this, re-do the installation procedure and use the "Modify/Change" option.

The SAY.. is optional because this command can also be used to erase any line (or portion of a line) on the screen. Bring dBASE II up and type:

```
^ERASE^
^@ 20,30 SAY 'What?'^
^@  5,67 SAY 'Here...'^
^@ 11,11 SAY "That's all."^
^@ 20, 0^
^@  5, 0^
^@ 11.16^
```

Instead of just showing a prompt, the command can be used to show the value of an expression with one or more variables. The form is:

```
^@ <coordinates>[SAY <expression>]
```

Type the following in dBASE II:

```
^USE Names^
^@ 13,9 SAY Zip:Code
^@ 13,6 SAY State
^SKIP 3^
^@ 23, 5 SAY Name + ', ' + Address
```

The command can be expanded further to show you the values of variables being used (memory variables or field names in a database) at whatever screen position you specify. (The variables used by both **GET** and **SAY** must exist before you call them out or you will get an error.)

^@ <coordinates>[SAY <expression>][GET <variable>]

To see how this works, type the following (do NOT QUIT dBASE when you're done--there's more to come):

```
^ERASE^
^USE Names^
^@ 15, 5 SAY 'State' GET State
^@ 10,17 GET Zip:Code
^@  5, 0 SAY 'Name' GET Name
     (Stay in dBASE)
```

This sequence has positioned the values of the variables (with and without prompts) at various places on the screen. With this facility, you can design your own input forms so that the screens that your operator sees will look just like the old paper forms that were used before.

To get data into the variables on the screen at your chosen locations, type:

^READ^

The cursor positions itself on the first field you entered. You can now type in new data, or leave it the way it was by hitting <enter>. When you leave this field, it goes to the second variable you entered.

Change the data in the remaining two fields. When you finish with the last one, you are back in dBASE II. Now type **^DISPLAY^**. The record now has the new data you entered.

As you can see, **GET** works somewhat like the **INPUT** and **ACCEPT** commands. It is much more powerful than either because it allows you to enter many variables.

A database may have a dozen or two fields (up to 32), but for any given data entry procedure, you may be entering data in only half a dozen of those. Rather than using **APPEND**, which would list all the fields in the database on the screen, you can use **^APPEND BLANK^** to create a record with empty fields, then **GET** only the data you want.

Our <Names> file is not a good example (the accounting system at the end of this section is better), but we can us it to show how to selectively get data into a database with a large structure.

To give you more practice with command files, create a file called <Trial.CMD> with the following commands in it:

```
^ERASE^
^? 'This procedure allows you to add new records to
the'^
^? 'NAMES.DBF database selectively.  We will be
adding'^
^? 'only the Name and the Zip:Code now.'^
^?^
^? 'Type S to stop the procedure,'^
^? '<enter> to continue.'^
^WAIT TO Continue^

^USE Names^
^DO WHILE Continue <> 'S' .AND. Continue <> 's'
^    APPEND BLANK^
^    ERASE^
^    @ 10, 0 SAY "NAME" GET Name^
^    @ 10,30 SAY "ZIP CODE" GET Zip:Code^
^    READ^

^    ? '  S to stop the procedure,'^
^    ? '<enter> to continue.'^
^    WAIT TO Continue^
^ENDDO^
^RETURN^
```

When you're back to CP/M, type ^dBASE Trial^ (or ^DO Trial^ if you renamed the dBASE.COM file as we suggested), Now enter data into several records. After you've finished, LIST the file to see what you've added.

As you can see, data entry is simple and uncluttered.

The screen can be customized by placing prompts and variable input fields wherever you want them.

NOTE: You must use the ^ERASE^ or ^CLEAR GETS^ command after every 64 ^GET's^. Use the latter command if you do not want to change the screen.

A command file that summarizes what we've learned

Before you read on, you can run the following file to see what it does. Type ^dBASE Sample^ if you're in CP/M or ^DO Sample^ if you're in dBASE II. Respond to the prompts. After you've run it, you can come back and go through the documentation. It summarizes most of what we've covered so far and includes copious commentary.

```
*********************** SAMPLE.CMD ***********************
* This command file prompts the user with screen
* messages and accepts data into a memory variable, then
* performs the procedure selected by the user. This is only
* a program fragment, but it does work.
*      We haven't written the procedures that are called
* by the menu yet, so instead, we can have the computer
* perform some actions that show us what it does
* and which paths it takes (stubbing).
*     Normally, dBASE II shows the results of the commands
* on the CRT. This can be confusing, so we SET TALK OFF.

SET TALK OFF
USE MoneyOut
ERASE

* It's good housekeeping to erase the screen before you
* display any new data on it.
* Our substitute display function can be used to put
* information on the CRT screen like this:

?
?
?
?
? '       OUTGOING CASH MENU'
?
?
? '       0 = Exit'
? '       1 = Accounts Payable Summary'
? '       2 = Enter New Invoices'
? '       3 = Enter Payments Made'
?
? '       Your Choice is Number'
WAIT TO Choice
ERASE

* Since we haven't developed the procedurss to do these
* three items yet we'll have the computer display
* different comments, depending on which alternative is
* selected from the menu.
```

```
IF Choice = '1'
   @ 0,20 SAY 'One'
ELSE
   IF Choice = '2'
      @ 1,20 SAY 'Two'
   ELSE
      IF Choice = '3'
         @ 2,20 SAY 'Three'
      ELSE
         @  7,20
         @  8,20 SAY ' ANY OTHER CHARACTER INPUT EXCEPT 1, 2, OR 3 '
         @  9,20 SAY ' CAUSES THIS COMMAND FILE TO TERMINATE AFTER '
         @ 10,20 SAY ' PRINTING OUT THIS MESSAGE.  NOTICE THAT THE '
         @ 11,20 SAY ' DIGITS HAD TO BE IN QUOTE MARKS IN THE "IF" '
         @ 12,20 SAY ' STATEMENTS ABOVE BECAUSE THE WAIT COMMAND   '
         @ 13,20 SAY ' ACCEPTS ONLY CHARACTER INPUTS               '
         @ 14,20 SAY '                                             '
      ENDIF 3
   ENDIF 2
ENDIF 1

* Each IF must have a corresponding ENDIF.  We've also
* put a label after the ENDIF to indicate with IF it
* belongs with, to make certain that we have closed all
* the loops.

?
?
?
?
INPUT 'Do you want to continue (Yes or No)?' TO Decision
ERASE
IF Decision
   INPUT "Okay, let's have a number, quickly." TO Number
ELSE
   @ 10,20 SAY "  WHY NOT?  "
   WAIT
ENDIF
ERASE
@ 10,20 SAY " I'M NOT READY FOR THAT.  GOOD-BYE.  "

* This next DO WHILE loop provides a delay of a few seconds
* to keep the last message on the CRT long enough to be read
* before the program terminates.  You may find this useful
* in command files that you write.  To change the delay time,
* either change the limit (100) or the step (+ 1).
STORE 1 TO X
DO WHILE X < 100
   STORE X + 1 TO X
ENDDO
ERASE
RETURN
```

You may want to run the program again. Try all the alternatives, then try entering inputs that are definitely wrong. You'll see how the program works and how dBASE II handles errors.

While it's only a program fragment and doesn't do any useful work, <Sample.CMD> does point up quite a few things:

1. Using **ERASE** frequently is good housekeeping that's easy to do.
2. Using indentation helps make the operation of the program clearer. That's also why we used upper- and lowercase letters. The computer sees them all as uppercase, but this way is much easier for us humans.
3. The "?" can be used to space lines on the display and to show character strings (in quotes or brackets).
4. The **WAIT** command waits for a single character before letting the program move on. The input then must be treated as a character, the way we did in the nested IF's by putting quotes around the values we were looking for.
5. The **INPUT** command waits for and accepts any data type, but characters and strings must be in single or double quotes or square brackets. When you have an apostrophe in your message, use the double quote or square brackets to define the string or the computer gets confused.
6. You don't have to predefine variables. Just make up another name whenever you need one (up to a maximum of 64 active at any one time).
7. Logical values can be treated in shorthand. "IF Decision" in the program worked as if we had said: "IF Decision = T".
8. The **RETURN** at the end of the program isn't necessary, but was tacked on because you would need it if this were a sub-procedure in another command file. That's how the computer knows that it should go back where it came from, rather than just quitting.

Working with multiple databases (PRIMARY, SECONDARY, SELECT)

As we've seen, when you first start working with DBASE II, you type ^USE <filename>^ to tell dBASE II which file you're interested in, then proceed to enter data, edit, etc.

To work on a different database, you type ^USE <NewFile>^. dBASE II closes the first file and opens the second one, with no concern on your part. You can use any number of files this way, both from your terminal and in command files. You can also close a file without opening a new one by typing ^USE^.

When you USE a file, dBASE II "rewinds" it to the beginning and positions you on the first record in the file. In most cases, this is exactly what you want. In some applications, however, you will want to access another file or files without "losing your place" in the first file.

dBASE II has an exceptionally advanced feature that permits you to work in two separate active areas at the same time: PRIMARY and SECONDARY. You switch between them with the ^SELECT^ command

You are automatically placed in the PRIMARY area when you first start. To work on another database without losing your position in the first one, type in ^SELECT SECONDARY^, then ^USE <newfile>^. To get back to the original work area, type ^SELECT PRIMARY^, then continue with that database.

The two work areas can be used independently. Any commands that move data and records operate only in the area in USE.

Information, however, can be transferred from one area to the other using P. and S. as prefixes for variables. If you are in the PRIMARY area, use the S. prefix for variables you need from the SECONDARY area; if you are in the SECONDARY area, use the P. prefix for variables you need from the PRIMARY area.

As an example, this command is used in the <NameTest.CMD> file in the accounting system at the end of this Part of the manual. Individual records in a file in the PRIMARY area are checked against all the records in another file in the SECONDARY area.

The same command is also used in the <TimeCalc.CMD>, <DepTrans.CMD> and <Payroll.CMD> files.

While you may not think of an application now, keep the command in mind: you'll find it useful.

Generally useful system commands and functions

MODIFY COMMAND <filename> lets you modify the named command file directly from dBASE II using the normal full screen editing features.

BROWSE displays up to 19 records and as many fields as will fit on the screen. To see fields off the right edge of the screen, use ctl-B to scroll right. Use ctl-z to scroll left.

CLEAR resets dBASE II, clearing all variables and closing all files.

RESET is used after a disk swap to reset the operating system bit map. Please read the detailed description in the command dictionary (Part II) before using it.

***** allows comments in a command file, but the comments are not displayed when the command file is executed. This allows notes to the programmer without confusing the operator. There must be at least one space between the word or symbol and the comment, and the note cannot be on the same line as a command. REPEAT: commands and comments must be on separate lines.

REMARK allows comments to be stored in a command file, then displayed as prompts to the operator when the file is used. There must be at least one space between the word and the remark, and the remark cannot be on a command line.

RENAME <oldfile> TO <newfile> changes file names in the CP/M directory. Do NOT try to rename files in USE.

QUIT TO '<system .COM file list>' allows you to terminate dBASE II and automatically start execution of CP/M and other .COM files. Each .COM file named must be in single quotes, and separated from other file names (in single quotes) by commas.

You can also use the "?" command to call out the following functions:

is the record number function. When called, it provides the value of the current record number.

***** is the deleted record function, and returns a True value if the record is deleted, False if not deleted.

EOF is the end of file function. It is True if the end of the file in USE has been reached, False if it has not been reached.

A few words about programming and planning your command files

The first thing to do when you want to set up a command file is to turn the computer off.

That's right: that's where many programmers go wrong. They immediately start "coding" a solution, before they even have a clear idea of what they're trying to do.

A much better approach is covered in a number of texts on structured programming and some of the structured languages. One reference you might check is Chapter 2 in "An Introduction to Programming and Problem Solving in Pascal" by Schneider, Weingart and Perlman. Another is Chapters 1, 4 and the first few pages of 7 in "Pascal Programming Structures" by Cherry. Then if you really want to get into programming, there's an excellent text on PL/I called "PL/I Structured Programming" by Joan Hughes.

Briefly, here's the approach:

Start by defining the problem in ordinary English. Make it a general statement.

Now define it further. What inputs will you have? What form do you want the outputs and reports in?

Next, take a look at the exceptions. What are the starting conditions? What happens if a record is missing?

Once you've defined what you want to do, describe the details in modified English. The texts call it "pseudocode". All this means is that you use English terms that are somewhat similar to the instructions that the computer understands.

You might write your program outline like this:

```
Use the cost database
Print out last month's unpaid invoices
Write a check for each unpaid invoice.
```

Adding a bit more detail, it looks like this:

```
USE CostBase
Print out last month's unpaid invoices using
   the SUMMARY.FRM file
Start at the beginning of the database
And go through to the end:
If the invoice has not been paid
   Pay the invoice
   And enter it in the database
Do this for every record
```

In perhaps two more steps, this could be translated into a command file like this:

```
USE CostBase
* Print a hardcopy summary for December, 1980.
REPORT FORM Summary FOR Bill:Date >= '801201' .AND.
      Bill:Date <= '801231' TO PRINT
GOTO TOP                  * Go to the first record
DO WHILE .NOT. EOF        * Repeat for the entire file
   IF Check:Nmbr=' '      * If the invoice isn't paid,
      DO WriteCheck       * write a check, then
      DO Update           * update the records
   ENDIF
SKIP                      *Go to the next record
ENDDO
```

The term <u>top-down, step-wise refinement</u> can be applied to this procedure, but that's forty-three dollars worth of words to say: "Start at the top, then divide and conquer".

Actually, it's just a sensible approach to solving most kinds of problems. First state the overall problem, trying to define what it is and what it isn't. Then gradually get into more and more detail, solving the details that are easy to solve, putting the more complicated details aside for later solution (again, perhaps in parts).

At this stage in our example, we haven't done the <Summary.FRM> file nor the <WriteCheck.CMD> and <Update.CMD> files, but it doesn't matter.

And in fact, we're probably better off not worrying about these details because we can concentrate on the overall problem solution. We can come back after we've tested our overall solution and clean up these procedures then.

Tip: You can still test a partial program like this by using what programmers call stubs. Set up the command files that you've named in the program and enter three items: a message that let's you know the program reached it, WAIT and RETURN. dBASE II will go to these procedure files, give you the message, then return and continue with the rest of the program after you hit any key.

Section IV:

Expanding your control with functions.....	80	
Changing dBASE II parameters and defaults.....	84	SET..
Merging records from two databases.....	86	UPDATE
JOINING entire databases.....	87	JOIN
Full screen editing and formatting.....	88	SET FORMAT TO SCREEN @..SAY..GET..PICTURE..
Formatting the printed page.....	90	SET FORMAT TO PRINT @..SAY..USING..
Setting up and printing a form.....	91	
Time to regroup.....	93	

By now you should be writing command files that can perform useful work for you.

To help you further, in this section we will introduce more functions, a few more commands and go into quite a bit of detail on how you can print out your data in exactly the format you want it.

Expanding your control with functions

Functions are special purpose operations that may be used in expressions to perform things that are difficult or impossible using regular arithmetic, logical and string operations. dBASE II functions fall into the same three categories, based on the results they generate.

Functions are called up by typing in ^?^ then a space and the function. They can be called from the terminal or within command files.

NOTE: the parentheses shown below must be used.

(Remember that "strings" are simply a collection of characters (including spaces, digits and symbols) handled, manipulate and otherwise used as data. A "substring" is a portion of any specific string.)

Don't worry about memorizing them now, but do scan the descriptions so that you know where to look when you need one of them in a command file.

!(<variable/string>)

is the lower- to uppercase function. It changes all the characters from 'a'..'z' in a string or string variable to uppercase. Any other characters in the string are unaffected. You'll see this used frequently in the accounting system (Section VI) to convert inputs from the keyboard into a standard form in the files. This makes it simpler when searching for data later, since you will know that all of the data is stored in uppercase, regardless of how it was entered.

TYPE(<expression>)

is the data type function and yields a C, N or L, depending on whether the expression data type is Character, Numeric or Logical.

INT(<variable/expression>)

is the integer function. It "rounds off" a number with a decimal, but does it by throwing away everything to the right of the decimal. The term inside the parentheses (you must use the parentheses) can be a number, the name of a variable or a complex expression. In the latter case, the expression is first evaluated, then an integer is formed from the results.

Note that INT(123.86) yields 123, while INT(-123.86) yields -123. A call to a variable yields a truncated integer formed from the current value of that variable. If we were on record 7 of MoneyOut.DBF, a call to INT(Amount) would produce 2333, the integer part of $2,333.75.

To rounded to the nearest whole number (rather than chop), use this form: INT(value + 0.5). The value within parentheses is first determined, then the integer function of that is taken.

The integer function can also be used to round a value to any number of decimal places. INT(value*10 + 0.5)/10 rounds a value to the nearest decimal place because of the order of precedence of operations (parentheses, then integer, then divide). To round to two places, use "100" in place of the "10"s. For 3 places, use "1000", etc.

VAL(<variable/string/substring>)

is the string to integer function. It converts a character string or substring made up of digits, a sign and up to one decimal point into the equivalent numeric quantity. VAL('123') yields the number 123. VAL(Job:Nmbr) yields the numeric value of the contents of the job number field in our MoneyOut database, since we stored all Job Numbers as characters. You can also use it with the substring operator: VAL($(<string>)).

STR(<expression/variable/number>, <length>, <decimals>)

is the integer to string function. It converts a number or the contents of a numeric variable into a string with the specified length and the specified number of digits to the right of the decimal point. The specified length must be large enough to encompass at least all the digits plus the decimal point. If the numeric value is shorter than the specified field, the remaining portion is filled with blanks. If the decimal precision is not specified, "0" is assumed.

This function is used quite often in the accounting system to simplify displays. Numbers are converted to strings then concatenated with (joined to) other strings of characters for displays.

LEN(<variable/string>)

is the string length function. It tells you how many characters there are in the string you name. This can be useful when the program has to decide how much storage to allocate for information with no operator intervention. However, if a character field variable name is used, this function returns the size of the field, not the length of the contents (since any unused positions are filled with blanks by dBASE II).

$(<expression/variable/string>, <start>, <length>)
is the substring function. It selects characters from a string or character variable, starting at the specified position and continuing for the specified length.

As an example, if we had a variable called <Date> whose value was '810823', the function ^$(Date,5,2)^ would give us '23'. To convert these numerals to a number, we could use ^VAL($(Date,5,2)).

You'll find an example of this in the <DateTest.CMD> file in Section VI, where groups of two characters are taken from a 6-character date field, converted to integers (using the VAL(...) function), then evaluated to see if they are in the correct range.

Don't confuse this with the substring logical operator described in Section II.

@(<variable1/string1>, <variable2/string2>)
is the substring search function. You might think of this as "Where is string1 AT in string2?" When you use this function, it produces the character position at which the first string or character variable starts in the second string or character variable. If the first string does not occur, a value of "0" is returned.

One use for this is to find out where a specific string starts so that you can use the preceding substring function. Another use is to find out if a specific string occurs at all.

(If you only need to know whether one string is in another one, you can use the relational string operator: String1$String2, Section II.)

You'll find these useful in a command file when the computer is searching without operator intervention, and you can't simply step in and look to see where the data is.

CHR(<number>)
yields the ASCII character equivalent of the number. Depending on how your terminal uses the standard ASCII code, ? CHR(12) may clear your screen, CHR(14) might produce reverse video while ? CHR(15) would cancel it. Other functions can be used to control hardware devices, such as a printer. Check your manual--you'll probably find a few interesting features.

To get underlining on your printer, try joining a character string, the carriage return and the underline like this: ? 'string' + CHR(13) + '_'. You could even set up a command file that uses the LEN function to find out how long the string is, then produces that many underline strokes.

&

is the **macro substitution** function. When the symbol is used in front of a memory variable name, dBASE II replaces the name with the value of the memory variable (must be character data). This can be used when a complex expression must be used frequently, to pass parameters between command files, or in a command file when the value of the parameter will be supplied when the program is run.

In the <Reportmenu.CMD> file in Section VI, it is used to get the name of the required database:

```
? 'Which file do you want to review?'
ACCEPT TO Database
USE &Database
```

It could also be used as an abbreviation of a command: ^STORE 'Delete Record' TO D^. The command: ^&D 5^ would then delete record 5 when the program runs.

If the macro command is not followed by a valid string variable, it is skipped. This means that you can use the symbol itself as part of a string without getting an error indication.

FILE(<"filename"/variable/expression>)

yields a True value if the file exists on the disk, False if it does not. If you use a specific file name, use the quote marks. The name of a string variable does not require the quote marks. You can also use any valid string expression: ^FILE("B:"+Database)^ would tell you whether the file name stored in the memory variable <Database> is on drive B (see ReportMenu.CMD in Section VI).

TRIM

eliminates the trailing blanks in the contents of a string variable. This is done by typing:
^STORE TRIM (<variable>) TO <newvariable>

Changing dBASE II characteristics and defaults

dBASE II has a number of commands that control how it interacts with your system setup. You can change these parameters back and forth "on the fly", or set them up once at the beginning of your command file and leave them. In many applications, the defaults will be just what you need.

Parameters are changed in your command files (or interactively) by using the SET command. In the list below, normal default values are underlined.

Once again, there's no need to memorize these. As you work with the established defaults, you can decide if you want to change any of the parameters on the list.

SET TALK ON Displays results from commands on console.
OFF No display.

SET PRINT ON Echoes all output to your 'list' device.
OFF No listing.

SET CONSOLE ON Echoes all output to your console.
OFF Console off.

SET SCREEN ON Turns on full screen operation for APPEND, EDIT, INSERT and CREATE commands.
OFF Turns full screen operation off.

SET FORMAT TO SCREEN sends output of @ commands to the screen

SET FORMAT TO PRINT sends output of @ commands to the printer

SET MARGIN TO <nnn> sets the left-hand margin on your printer ("nnn"<=254)

SET RAW ON DISPLAYs and LISTs records without spaces between the fields
OFF DISPLAYs and LISTs records with an extra space between fields

SET HEADING TO <string> changes the heading in the **REPORT** command

SET ECHO ON All commands in a command file are displayed on your console as they are executed.
OFF No echo

SET EJECT ON enables page feed with **REPORT** command
OFF disables page feed

SET STEP ON Halts after completing each command, for debugging command files.
OFF Normal continuous operation.

SET DEBUG ON sends output from the ECHO and STEP commands to the printer only
OFF sends ECHO and STEP output to the screen

SET BELL ON enables bell when field is full
OFF disables bell

SET COLON ON uses colons to delimit input variables on the screen
OFF disables the colons

SET CONFIRM ON waits for <enter> before leaving a variable during full screen editing
OFF leaves the variable when the field is full

SET CARRY ON carries data from the previous record forward to the new record when in **APPEND**
OFF shows a blank record in **APPEND** mode

SET INTENSITY ON enables dual intensity for full screen operations
OFF disables dual intensity

SET LINKAGE ON permits databases to be linked for display with up to 64 fields and up to 2000 bytes per displayed record. The P. or S. prefix must be used when field names are similar in both databases
OFF disables linkage

SET EXACT ON requires that all characters in a comparison between two strings match exactly
OFF allows different length strings. E.g., 'ABCD'='AB' would be True (Also affects **FIND** command)

SET ESCAPE ON allows the <escape> key to abort command file execution
OFF disables the <escape> key

^SET ALTERNATE TO <filename>^ creates a file with a .TXT extension for saving everything that goes to your CRT screen. To start saving, type **^SET ALTERNATE ON^**.

You can change the file that you are saving to by typing **^SET ALTERNATE TO <newfile>^**.

To stop, type **^SET ALTERNATE OFF^**. This also terminates when your QUIT dBASE II.

Merging records from two databases (UPDATE)

Data can be transferred from one database file to another with the following command:

UPDATE FROM <database> ON <key> [ADD <field list>]
[REPLACE <field list>]

Note: Both databases must be presorted on the "key" field before this command is used.

Both files are "rewound" to the beginning, then key fields are compared. If they are identical, then data from the FROM data base is either added numerically to data in the USE file, or is used to replace data in the use file for the fields specified in the field list. When fields do not match, those records are skipped. This command can be used to keep inventory updated, for example.

In the accounting system in Section VI, it is used in <Payroll.CMD> and <CheckStub.CMD>. It's useful and worth experimenting with.

JOINing entire databases

JOIN is one of the most powerful commands in dBASE II. It can combine two databases (the USE files in the PRIMARY and SECONDARY work areas) to create a third database. The form of the command is:

JOIN TO <newfile> ON <expression> [FIELD <list>]

In operation, the command positions dBASE II on the first record of the primary USE file and evaluates each of the records in the secondary USE file. Each time the "expression" yields a true result, a record is added to "newfile". If you are in the primary area when you issue the JOIN command, prefix variable names from the secondary USE file with S.. If you are in the secondary area, prefix variables from the primary USE file with P.. (See example below.)

When each record in the secondary USE file has been evaluated against the first record of the primary USE file, dBASE II advances to the second record of the primary USE file, then evaluates all of the records from the secondary USE file again. This is repeated until all records from the files have been compared against each other.

Note: This can take a great deal of time to complete if the two databases are very large. It may also not be possible to complete at all if the constraints are too loose. Two files with 1,000 records each would create a JOIN database with 1,000,000 records if the JOIN expression was always true, while dBASE II is limited to 65,535 records in any single database.

To use the command, use this sequence of instructions:

```
USE Inventory
SELECT SECONDARY
USE Orders
JOIN TO NewFile FOR P.Part:Number=Part:Number;
        FIELD Customer,Item,Amount,Cost
```

This creates a new database called <NewFile.DBF> with four fields: Customer, Item, Amount and Cost. The structure of these fields (data type, size) are the same as in the two joined databases. (Notice that the "P." prefix is used to call a variable from the work area not in USE.)

Full screen editing and formatting (@..SAY..GET..PICTURE)

dBASE II has a powerful series of formatting commands that allow you to position information precisely where you want it. You saw this in action in our <Sample.CMD> program, where we used:

@ <coordinates> SAY ['prompt'] GET <variable>

This command was able to position prompts and variables (and their values) at any location we specified on the screen. When we listed a series of commands, then followed them with **READ**, we were able to control the format of the entire screen. You might want to create and run the following command file fragment to refresh your memory:

```
STORE "  "  TO MDate
STORE "   " TO MBalance
STORE "   " TO MDraw
@ 5,5 SAY "Set date MM/DD/YY " GET MDate
@ 10,5 SAY "What is the balance? " GET MBalance
@ 15,5 SAY "How much is requested" GET MDraw
READ
ERASE
@ 5,5 SAY "Should we do an evaluation?" GET MEvaluate
READ
```

The command can also be used without the SAY phrase as **@ <coordinates> GET <variable>** (with a later READ in the command file). This displays only the colons delimiting the field length for the variable.

Tip: In the SCREEN mode the line numbers do not have to be in order, but it's good practice to write them this way since they must be in order for PRINT formatting.

This command can also be expanded for special formatting like this:

@ <coordinates> SAY [expression] GET <variable> [PICTURE <format>]

The optional **PICTURE** phrase is filled in using the format symbols listed below. The command:

```
@ 5,1 SAY "Today's date is" GET Date PICTURE '99/99/99'
```

would display:

Today's date is: / / :

assuming that the Date variable was blank. In this example, only digits can be entered.

The GET function symbols are:

9 or # accepts only digits as entries.
A accepts only alphabetic characters.
! converts character input to uppercase.
X accepts any characters.
$ shows '$' on screen.
* shows '*' on screen.

With this command, you can format your menu and input screens any way you want them, quickly and easily.

Tip: The Osborne series of accounting books, besides describing some fairly sophisticated systems, also includes CRT Mask Layouts for menus and entry formats with coordinates clearly marked. Well worth their price for this alone.

Formatting the printed page (SET FORMAT TO PRINT, @..SAY..USING)

When you **SET FORMAT TO PRINT**, the @-command sends its information to the printer instead of the screen.

The GET and PICTURE phrases are ignored, and the READ command cannot be used.

Data to be printed on checks, purchase orders, invoices or other standard forms can first be organized on the screen with this command, then printed exactly as you see it:

@ coordinates SAY variable/expression/'string' [USING format]

For printing, the coordinates must be in order. The lines must be in increasing order (print line 7 before line 9, etc.). On any given, line the columns must be in order (print column 15 before column 63, etc.).

This command can output the current value of a variable that you name, the result of an expression, or a literal string prompt message.

If the **USING phrase** is included, this command specifies which characters are printed as well as where they appear on the page. the symbols used are:

9 or # prints a digit only.
A prints alphabetic characters only.
X prints any printable character.
**$** prints a digit or a '$' in place of a leading zero.
***** prints a digit or a '*' in place of a leading zero.

The command **10,50 SAY Hours*Rate USING '$$$$$$$.99'** could be used in both the screen and the printer modes since it has no GET phrase. For Hours = 8 and Rate =12.73, it would print or display $$$$101.84, useful for printing checks that are more difficult to alter.

Setting up and printing a form

To set up a form, use measurements based on your printer spacing (ours prints 10 characters per inch horizontally, with 6 lines per inch vertically).

The "Outgoing Cash Menu" that we used in our earlier command file could very well have had another selection item called "4 = Write checks", so we're going to do part of the WriteCheck command file.

To start with, we'll have to input the date. The following command lines accept the date to a variable called MDate, and checks to see whether it is (probably) right:

```
ERASE
SET TALK OFF
STORE "  " TO MDate
STORE T TO NoDate
DO WHILE NoDate
   @ 5,5 SAY "Set date MM/DD/YY" GET MDate PICTURE "99/99/99"
   READ
   IF VAL($(MDate,1,2)) < 1;
        .OR.  VAL($(MDate,1,2)) > 12;
        .OR.  VAL($(MDate,4,2)) < 1;
        .OR.  VAL($(MDate,4,2)) > 31;
        .OR.  VAL($(MDate,7,2)) <> 81
      STORE "  " TO MDate
      @ 7,5 SAY "***** BAD DATE, PLEASE RE-ENTER.  *****"
      STORE T TO NoDate
   ELSE
      STORE F TO NoDate
   ENDIF
ENDDO because we now have a valid date
ERASE
```

In English, the above first sets the value of MDate to 8 blanks, then the @..SAY command displays:

```
Set date MM/DD/YY: / / :
```

When the date is entered, it is checked by the IF to see whether the month is in the range 1-12, day is in the range 1-31, and year=81. This is done in three steps:

- the substring function $ takes the two characters representing the month, day or year (e.g., for month it starts in the 4th position and takes 2 characters)
- the VAL function converts this to an integer
- this integer is then compared against the allowed values

If the value is out of range, MDate is set to blanks again and an error message comes up. When a date within the allowed range is entered, the program continues.

The printout for the check itself could be the next portion of the program. Using the measurements of our

checks, this is the list of commands:

```
@ 8,3 SAY Script * A character variable that prints the
                 * amount in script.  This is filled in
                 * by another procedure called Chng2Scrpt.
                 * We stubbed this for now like this:
                 *  STORE 'Script Stub' TO Script
                 *  RETURN
@ 11,38 SAY Vendr:Nmbr
@ 11,50 SAY MDate
@ 11,65 SAY Amount
@ 13,10 SAY Vendor
@ 14,10 SAY Address
@ 15,10 SAY City:State
@ 15,35 SAY ZIP
@ 17,10 SAY Who
```

You can check this out on your screen before you print it, then switch from SCREEN to PRINT modes with the SET command. The values for the variables are provided elsewhere in your command file.

Longer forms are no problem: a printer page can be up to 255 lines long. To reset the line counter, issue and 'EJECT^ command with the printer selected.

Time to regroup

Because dBASE II is such a powerful system, it has a large number of commands and techniques for dealing with your database needs and allowing you to get more information more easily than any other database system or file handler currently running on micros.

The easiest way to learn the techniques is to go through the examples and key them into your computer, changing names as you go to reflect your needs rather than our examples.

You start by using the CREATE command to create your databases. Besides MoneyOut.DBF, we've created a number of other database structures that you might find useful. They're listed at the beginning of Section VI.

The <Costbase.DBF> file started life as <MoneyOut.DBF>. We've modified it a great deal, changing field names and sizes (with and without data in the database).

We've also added our own spacing between the fields (see fields 5, 7 and 10), rather than using the dBASE II "RAW" default of a space between each field. NOTE: you can NOT enter the hyphen as a field name when you CREATE a file. To enter it, you have to MODIFY STRUCTURE, use ctl-N to insert a blank line, then type in the hyphen and the field size. Ctl-W saves the change (ctl-O with Superbrain).

You may want to check some of the other database structures, then see how they are used in the programs. We tried to keep the field names and their individual structures the same for all our databases to allow for file merges and other uses. Data from one database will fit into corresponding fields in another, and with common names the transfer is straightforward.

You might want to check through the command files in Section VI now. Most of the dBASE II commands have been used, and the files work the way they are set up.

The first command file is the main menu for the system, with sub-files selected by pressing a number. Some of the files get a bit complicated (<Payroll> for example), so you might go to some of the utility programs at the end of Section VI before you try to unravel the rest of the programs.

Writing these command files, we used exactly the procedures that we recommended earlier: first define the problem in a general sense. Gradually keep dropping down in levels of detail, using ordinary English at first, then pseudocode, putting terms that dBASE II would understand in capitals when we finally got to that level.

When we came up with something that had to be done, but we weren't sure how to do it, we simply made up a procedure name for it, then went back to it later.

The indentation and mixture of upper- and lowercase

letters was not done just for this manual: it's the way we work all the time. It makes writing the command files a lot easier because you can see groupings of the structures that you are using.

The identifiers were pulled out of our semi-English pseudocode, modified a bit to fit within the 10 characters allowed, but not enough to destroy the meaning.

Comments are sprinkled throughout the files for documentation, although in many cases the programs are almost self-documenting because so many of the dBASE II commands are similar to English equivalents.

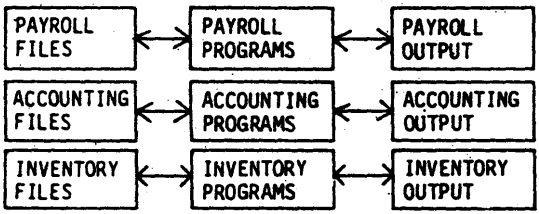
Section V:

Database Basics..... 96
A brief introduction to database organization..... 98
dBASE II Records, Files and Data Types..... 99
dBASE II OPERATION SUMMARY.....102
dBASE II FUNCTION SUMMARY.....103
dBASE II COMMAND SUMMARY.....104
Commands grouped by what you want done.....109
109 File structure
110 File operations
110 Organizing database
110 Combining databases
111 Editing, updating, changing data
111 Using variables
112 Interactive input
112 Searching
112 Output
113 Programming

Database Basics

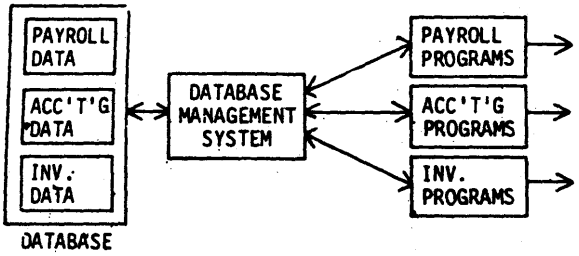
A database management system (DBMS) like dBASE II is considerably different from a file handling system.

A file handling system is usually configured like this:



The payroll programs process the payroll files. The accounting programs process the accounting files. And the inventory programs process the inventory files. To get reports that combine data from different files, a new program would have to be written and it wouldn't necessarily work: the data may be incompatible from file to file, or may be buried so deeply within the other programs that getting it out is more trouble than it's worth.

A database management system integrates the data and makes it much easier to get useful information from your records, rather than just reams of data. Conceptually, a DBMS looks something like this:



Data is monitored and manipulated by the DBMS, not the individual applications programs. All of the applications systems have access to all of the data. In a file handling system, this would require a great deal of duplicated data. Aside from the potential for entry errors, data integrity is extremely hard to maintain when the same data is supposed to be duplicated in different files: it never is.

To generate a new processing system in a file handling system, a new program and new files must be set up. Using a DBMS, a new access program is written, but the data does not

have to be restructured: the DBMS takes care of it.

If a new kind of data is added to a record (salary history in a personnel file, for example), file handling programs have to be modified. With a DBMS, additions and changes have no effect on the programs that don't need to use the new information: they don't see it and don't know that it's there.

Database management systems come in two flavors: hierarchal and relational. These terms refer to how the DBMS keeps track of data.

A **hierarchal system** tends to get extremely complex and difficult to maintain because the relationships between the data elements are maintained with sets, linked lists, and pointers telling the system where to go next. Very quickly, you can end up with lists of lists of lists and pointers to pointers to pointers.

A **relational database management system** like dBASE II is a great deal simpler. Data is represented as it is, and the relation between data elements can be considered a two-dimensional table like this one:

Col.1	Col.2	Col.3	Col.4	Col.5
Invoice Number	Supplier	Description	Amount	Number
2386	Graphic Process	Prints	23.00	BBQ-747
78622	Brown Engraving	Litho plates	397.42	TFS-901
M1883	Air Feeight Inc.	Shipping	97.00	SPT-233

Each row going across the table is called a **record**. Each column is called a **field** of the record. Each entry in the table must be a single value (no arrays, no sets, etc.) All the entries in a column must be of the same type. Each record (row) is unique, and the order of records (rows) doesn't matter.

When we show you more realistic examples later, you'll see that records don't get any more complicated, just larger.

A brief introduction to database organization

Once you've got your database set up, you'll want to access your data in an orderly, ordered manner.

With some databases, the order in which you enter the data will be the order in which you want to get your information out. In most cases, however, you'll want it organized differently.

With dBASE II you can organize data using the **SORT** command or the **INDEX** command. (Both of these are described in more detail in Section II: Organizing your databases.)

The **SORT** command moves entire records around to set up your database in ascending or descending order on any field that you specify (name, ZIP code, etc.). This field is called the key.

One drawback of sorting is that you may want to access the database on one field for one application, on another field for a different application. Another drawback is that any new records added are not in order, and would require a sort every time you entered data if you wanted to maintain the order.

Finding data is also relatively slow, since the sorted database must be searched sequentially.

INDEXING is a way around these problems.

Indexing is a method of setting up a file using only the keys that you are interested in, rather than the entire databases. A key is a database field (or combination of fields) that make up the "subject" of the record. In an inventory system, the part number might be the subject, and the amount-on-hand, cost, location, etc. the descriptive fields. In a personnel database, names or employee numbers would probably make the best keys.

With an indexed database, the keys alone are organized with pointers to the record to which they belong. dBASE II uses a structure called B*-trees for indexes. This is similar to a binary tree, but uses storage much more efficiently and is a great deal faster. A FIND command (described in Section II) typically takes 2 seconds with a medium to large database.

If you need your data organized on several different fields for different applications, you can set up several index files (one for each of the fields) and use the appropriate index file whenever required. You could have index files ordered by supplier name, by customer number, by ZIP code or any other key, all for a single database.

New entries to a database are automatically added to the index file being used.

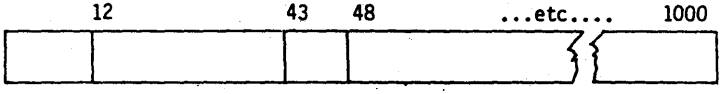
Another advantage of indexed databases is the rapid location of data that you are interested in.

dBASE II Records, Files and Data Types

dBASE II was designed to run on your micro so its scope stops short of infinity, but you'll find that you'll have to work at figuring out how to get to its maximums.

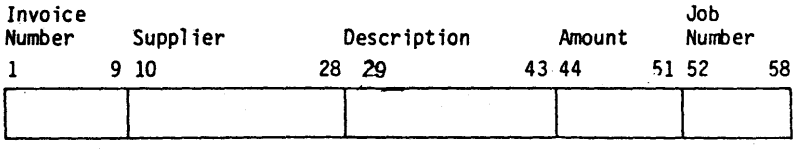
dBASE II limits you to 65,535 records per file, but with the memory and even "mass storage" limitations of a micro, this is really no limitation at all.

A dBASE II record can be as large as 32 fields and 1000 characters long (whichever comes first):



You might want to think of this as a 1000 character long strip that you can segment any way you want to up to the maximums, or shorten if you don't need to use it all. You can have four fields that use the full 1000 characters (254 characters per field maximum). Or a record one character (and field) long. Or anything in between.

In our previous example, each record had five fields and the total record length was 58 characters:



Data Types

As we said earlier, each field must contain a single type of data, and in dBASE II these are:

Character: all the printable ASCII characters, including the integers, symbols and spaces.

Numeric: positive and negative numbers as large as 1.8 x 10<63> down to numbers as small as 1.0 x 10<-63>. Accuracy is to ten digits, or down to the penny for dollar amounts as high as $99,999,999.99.

Logical: these are true/false (yes/no) values that occupy a field one character long. dBASE II recognizes T, t, Y and y as **TRUE**, while F, f, N and n are recognized as **FALSE**.

Field Names

Each field has a name so that dBASE II can recognize it when you want to find it. Field names can be up to 10 characters (no spaces) long, and must start with a letter, but can include digits and an embedded colon:

```
A               (valid)
A123456789      (valid)
Job:Number      (valid:  upper and lowercase okay)
A123,B456       (illegal comma)
Reading:        (illegal:  colon not embedded)
```

> **Tip:** Use as many characters as it takes to make the name meaningful. 'Job:Nmbr' is a lot better than 'No.' and infinitely better than 'J'. Using a maximum of nine characters will make handling memory variables much easier (discussed later).
>
> **Another tip:** Once you get into setting up Command files, you'll find it useful to use capital letters for words that dBASE II understands and upper and lowercase for fields, variables and other items that you control. You'll appreciate this the first time you go back into a command file to make changes.

dBASE II File Types

File names are limited to 8 characters and a 3 character extension after a period. You can use the colon in the file name, but then you'll only be able to manipulate the files through dBASE II: CP/M will store the files and get the names right, but won't recognize them if you ask it to perform a function like PIP. Ten character long filenames aren't a problem: CP/M simply chops them down to eight. If you use upper and lower case letters to name your files, CP/M will change them to capitals, but they'll still show up better in your command files.

A dBASE II file is simply a collection of information of a similar type under a single name, something like a giant file folder. dBASE II operates with the six different file types described below.

DBF Database files: This is where all your data is kept and the extension is automatically assigned by dBASE II when you **CREATE** a new file. Each **.DBF** file can store up to 65,535 records. Do not use a word processor on these files.

.FRM Report form files: These files are automatically created by dBASE II when you go through the **REPORT** dialog. They contain headings, totals, column contents, etc. They can be modified using a word processor or text editor, but we definitely recommend against this practice: make your changes using dBASE II.

.CMD Command files: These files contain a sequence of dBASE II statements to perform functions that you use frequently, and can be a complex as a complete payroll system. These are created using a text editor or word processor.

.NDX Index files: These are automatically created by the **INDEX** command. Indexing provides very rapid location of data in larger databases.

.MEM Memory files: These are automatically created when you **SAVE** the results of computations, constants or variables that you will want later. You can **SAVE** up to 64 items, each up to 254 characters long, then **RESTORE** them the next time you need them.

.TXT Text output files: This file is created when you use the **SET ALTERNATE** command to store everything that goes to the CRT on your disk, too. This feature can be used as a system logging function, and the information can later be edited, printed, and/or saved. They are also created when you **COPY...SDF**.

dBASE II OPERATIONS SUMMARY

Arithmetic operators (generate arithmetic results: p.37)

( ) : parentheses for grouping
* : multiplication
/ : division
+ : addition
- : subtraction

Relational Operators (generate logical results: p. 37)

< : less than
> : greater than
= : equal
<> : not equal
<+ : less than or equal
>= : greater than or equal

Logical Operators (generate T/F logical results: p. 38)

( ) : parentheses for grouping
.NOT. : Boolean not (unary operator)
.AND. : boolean and
.OR. : boolean or
$: substring logical operator (p. 40)
(is string1 in string2?)

String operators (generate string results: p. 41)

+ : string concatenation (joining)
- : string concatenation with blank squash

dBASE II FUNCTION SUMMARY

#	record number (p. 76)
*	deleted record (p. 76)
EOF	end of file (p.76)
!(<variable/string>)	convert to uppercase (p. 80)
TYPE(<expression>)	data type (p. 80)
INT(<variable/expression>)	integer function (p. 80)
VAL(<variable/string/substring>)	string to integer (p.81)
STR(<expression/variable/number>, <length>, <decimals>)	integer to string (p.81)
LEN(<variable/string>)	string length (p. 81)
$(<expression/variable/string>, <start>, <length>)	substring select (p. 82)
@(<variable1/string1>, <variable2/string2>)	substring search (p. 82)
CHR(<number>)	number to ASCII (p. 82)
&	macro substitution (p. 83)
FILE(<"filename"/var/exp>)	file exists? (p. 83)
TRIM	trailing blanks (p. 83)

dBASE II COMMAND SUMMARY

The following abbreviations are used in this summary:
<exp> = expression
<var> = variable
<str> = string
<coord> = coordinates
The symbols <..> bracket items that are to be specified by the user. Square brackets [..] enclose optional items. In some cases, options are nested (themselves have other options).

? <exp [,list]>
Display an expression (or list separated by commas) (p. 25)

@ <coord> [SAY <exp> [PICTURE 'picture']] [GET <var> [PICTURE 'picture']]
Format console screen or printer output (p. 88)

ACCEPT ['prompt'] TO <var>
Input a character string from the console, no quotes (p. 68)

APPEND [BLANK]
APPEND FROM <filename> [SDF] [FOR <exp>]
[DELIMITED [WITH delimiter]]
Add data to a database (pp. 26, 47, 70)

CANCEL
Abort a command file execution

CHANGE [scope] FIELD <list> [FOR <exp>]
Make multiple changes to a database (p. 51)

CLEAR
Reset dBASE file and memory variable environment (p. 76)

CONTINUE
Continue a LOCATE command (p. 55)

COPY [scope] TO <filename> [SDF] [STRUCTURE] [FIELD <list>] [FOR <exp>] [DELIMITED [WITH delimiter]]
Copy data from a database to another file (pp. 43, 47, 49)

COPY TO <filename> STRUCTURE EXTENDED
Creates a new file whose records define the structure of the old file. (see also CREATE <newfile> FROM <oldfile>)

COUNT [scope] [FOR <exp>] [TO <var>]
Counts records that satisfy some condition (p. 58)

CREATE
Make a new database (p. 14)

CREATE <newfile> FROM <oldfile>
Creates <newfile> with structure determined by the data in the records of <oldfile>. (see also COPY STRUCTURE EXTENDED)

DELETE [scope] [FOR <exp>]
Mark specified records for deletion (p. 28)

DELETE FILE <filename>
Erase a file from the system (p. 28)

DISPLAY [scope] [FOR <exp>] [OFF]
Show data based upon request (pp. 20, 23)

DISPLAY [scope] [<field> [,list]]
Shows only the selected field(s)

DISPLAY STRUCTURE
Show structure of the database in USE (p. 23)

DISPLAY MEMORY
Show the contents of the memory variables (p. 35)

DISPLAY FILES [ON disk drive]
Show a disk directory (p. 23)

DO <filename>
Execute a command file (p. 63)

DO WHILE <exp>
Perform a group of commands repeatedly (p. 66)

EDIT
Alter the data in a database (p. 18)

EDIT [number]
Presents a specific record for editing (p. 18)

EJECT
Do a form feed on the printer

ELSE
Alternate execution path in an IF command (p. 64)

ENDDO
Terminator for DO WHILE command

ENDIF
Terminator for an IF command

ERASE
Clear console screen

FIND <key>
Locate a record in an indexed database based upon key value (no quotes needed for character keys) (p. 54)

GO or GOTO [RECORD], or [TOP], or [BOTTOM], n
Position to a given place in a database (p. 24)

IF <exp>
Conditional execution command (p. 64)

INDEX ON <key> TO <filename>
Create an index file for the database in USE (p. 52)

INPUT ['prompt'] TO <var>
Accept user inputs into memory variables. User prompt string is optional (p. 68)

INSERT [BEFORE]
[BLANK]
Add a new record to a database among other records (p. 26)

JOIN TO <filename> ON <exp> [FIELDS <list>]
Create a database composed of matching records from two other databases (p. 87)

LIST
Show data records (pp. 20, 21)

LOCATE [scope] [FOR <exp>]
Find the record that matches a condition (p. 54)

LOOP
Escape mechanism for DO WHILE groups (p. 66)

NOTE or *
A command file comment that is not displayed when the command file is run

MODIFY COMMAND <filename>
Permits modification of a file directly from dBASE II (p. 76)

MODIFY STRUCTURE
Alter the structure of a database. Destroys all data in the database (p. 42)

PACK
Eliminates records marked for deletion (p. 28)

QUIT [TO list of CP/M level commands or .COM files]
Terminate dBASE and execute a program chain. Each command must be in quote marks, and commands must be separated by commas (p. 76)

READ
Enter full screen editing of a formatted screen. Accepts data into GET commands (p. 70)

RECALL [scope] [FOR <exp>]
Unmark records that have been marked for deletion (p. 28)

RELEASE [<var> [,list]] or [ALL]
Eliminate unwanted memory variables (p. 36)

REMARK
A comment that is shown on the screen when the command file is run

RENAME <oldfile> TO <newfile>
Give a file a new name (p. 76)

REPLACE [scope] <field> WITH <exp> [,<field> WITH <exp>...]
Alter data in a database. Make sure that you have a backup, because dBASE II will do precisely what you ask it to do, even if it's not exactly what you had in mind (p. 50)

REPORT [scope] [FORM <filename>] [TO PRINT] [FOR <exp>]
Generate a report (p. 56)

RESET
Tell CP/M that a diskette swap may have occurred

RESTORE FROM <filename>
Remember SAVEd memory variables. Destroys all existing memory variables

RETURN
Terminate a command file and return to calling file

SAVE TO <filename>
Write memory variables to a file for future use

SELECT [PRIMARY] or [SECONDARY]
Switch working areas (p. 75)

SET parameter [ON], or [OFF]
Dynamically reconfigure dBASE operation (p. 84)

SKIP ±<exp/number>
Move forward or backwards in the database (p. 24)

SORT ON <key> TO <filename> [ASCENDING]
[DESCENDING]
Generate a database that is sorted on a field (p. 52)

STORE <exp> TO <var>
Place a value into a memory variable (p.33)

SUM [scope] <field [,list]> [TO <var [,list]> [FOR <exp>]
Total fields in a database (p. 58)

TOTAL TO <filename> ON <key> [FIELDS <field [,list]>
Generate a database with sub-totals for records (p. 59)

UPDATE FROM <filename> ON <key> [ADD <field [,list]>]
[REPLACE <field [,list]>]
Modify a database with data from another database (p. 86)

USE <filename> [INDEX <filename>]
Open a database file for future operations (p. 20)

USE
Close a previously opened database file

WAIT [TO <var>]
Pause in program operation [for input] (p. 68)

dBASE II commands grouped functionally

FILE STRUCTURE:

CREATE defines an entirely new file structure

CREATE <newfile> FROM <oldfile> creates a new file whose structure is described in the records of the old file.

USE <oldfile>
COPY TO <newfile> STRUCTURE
These two commands combined create a new file with the same structure as an old file

USE <oldfile>
COPY TO <newfile> STRUCTURE EXTENDED
Create a new file that contains the structure of the old file as data

CREATE <newfile> FROM <oldfile>
Creates a new file whose structure is defined by the records in the old file.

DISPLAY STRUCTURE
LIST STRUCTURE
Both show the structure of the file in USE

MODIFY STRUCTURE changes file names, sizes, and overall structure, but destroys data in the database

To change structure with data in the database:

```
USE <oldfile>
COPY TO <newfile>
USE <newfile>
MODIFY STRUCTURE
APPEND FROM <oldfile>
COPY TO <oldfile>
USE <oldfile>
DELETE FILE <newfile>
```

To rename fields with data in the database:

```
USE <oldfile>
COPY TO <newfile> SDF
MODIFY STRUCTURE
APPEND FROM <newfile>.TXT SDF
DELETE FILE <newfile>
```

FILE OPERATIONS:

USE <filename> opens a file

USE <newfile> closes the old file

USE closes all files

RENAME <oldname> TO <newname>
Must NOT rename an open file

COPY TO <filename> creates a backup copy

CLEAR closes all files and erase all memory variables

SELECT [PRIMARY][SECONDARY]
allows two files to be independently open at the same time. Data can be transferred with P. and S. prefixes

DISPLAY FILES [ON <d>] lists databases on logged-in drive (or drive specified), can use LIST instead

DISPLAY FILES LIKE <wildcard> [ON <d>] shows other types of files on drives

QUIT closes both active areas, all files, terminates dBASE II operation

ORGANIZING DATABASES:

SORT ON <key> TO <newfile>
INDEX ON <key> TO <newfile>
Can use multiple keys for both commands

COMBINING DATABASES

COPY TO <newfile> creates a duplicate of the file in USE

APPEND FROM <otherfile> adds records to the file in USE

UPDATE FROM <otherfile> ON <key> adds to totals or replaces data in the file in USE. Both files must be sorted on the <key>.

JOIN creates a third file from two other files

EDITING, UPDATING, CHANGING DATA:

DISPLAY, LIST, BROWSE let you examine the records

DELETE marks record so it is not used

RECALL unmarks record

PACK erases deleted records

EDIT lets you make changes to specific records

REPLACE <field **WITH** data> global replacement of data in fields, can be conditional as with most dBASE II commands

CHANGE..FIELD edit based on field, rather than record

@ <coord> **GET** <var>
READ displays the variable, lets you change it

INSERT [BEFORE][BLANK] inserts a record in a database

UPDATE FROM <otherfile> **ON** <key> adds to totals or replaces data in file in USE from another file

MODIFY COMMAND <filename> allows changes to your command files without having to go through your text editor

USING VARIABLES:

(Allowed up to 64 memory variables plus any number of field names.)

LIST MEMORY, DISPLAY MEMORY both show the variables, their data types and their contents

& returns the contents of a character memory variable (i. e., provides a literal character string)

STORE <value> **TO** <var> sets up or changes variables

RELEASE <var> cancels the named variable

SAVE MEMORY TO <filename> stores memory variables to the named file (with .MEM extension)

RESTORE FROM <filename> reads memory variables back into memory (destroys any other existing memory variables)

INTERACTIVE INPUT:

WAIT stops screen scrolling, continues with any key

WAIT TO <var> accepts character to memory variable

INPUT ['prompt'] TO <var> accepts any data type to a memory variable (creates it if it did not exist), character input must be in quotes

ACCEPT ['prompt'] TO <var> same as INPUT, but no quotes around character input

@ <coord> SAY ['prompt'] GET <var> [PICTURE]
READ
displays memory variable, replaces it with new input

SEARCHING:

SKIP [±<exp>] moves forward or backward a specific number of records

GO[TO] <number>, GO TOP, GO BOTTOM
move you to a specific record, the first record, or the last record in the database

FIND <str> works with indexed file in USE, very fast

LOCATE FOR <exp>
CONTINUE
Searches entire database

OUTPUT:

?, DISPLAY, LIST show expressions, records, variables, structures

REPORT [FORM <formname>] creates a custom format for for output, then presents data in that form when called

@ <coord> SAY <var/exp/str> formats output to screen or to printer ([**USING <format>**] can be added to provide PICTURE format for the printer)

PROGRAMMING:

(Programs stored in COMMAND FILES with .CMD extension.)

DO <filename> starts the program

```
IF <conditions>
   perform commands
ELSE
  perform other commands
ENDIF
```

Makes choices, single or multiple (when nested)

```
DO WHILE <conditions>
   perform commands
ENDDO
```

<Conditions> must be changed by something in the loop eventually

(This page intentionally left blank.)

Section VI

A working accounting system

The following pages are command files written by a customer using almost all of the dBASE II commands, following the instructions in this manual.

The system may be fairly elementary from an accountant's viewpoint, but includes some programming techniques that you might find useful.

Besides illustrating individual commands, it shows you how to set up menus, how to find data and merge files, different ways of setting up the inputs to the system, and a number of interesting solutions to some of the problems involved in keeping a cash journal, doing payrolls, and managing databases so that information is available and data integrity is not compromised.

The programs are self-documented, with comments sprinkled liberally throughout.

CONTENTS

	Page
STRUCTURES	117
REPORT FILES	122
SYSTEM CONSTANTS	124
ACCOUNTS	125
COSTMENU	127
USETAX	130
COSTBILLS	133
COSTTIME	135
COSTUPDATE	138
PAYMENU	139
PAYBILLS	140
PAYFIND	144
PAYEMPS	146
PAYROLL	148
DEPMENU	155
DEPOSITS	156
DEPPRINT	158
DEPTRANS	159
IOMENU	160
IOPOST	162
IOREVIEW	165
INVMENU	167
INVOICES	169
REPORTMENU	172
JOBCOSTS	174
JOBSINDX	178
FINDBILLS	179
REVIEW	182
REVHDR	188
REVMRGN	189
SALESTAX	190
TIMECALC	193
PRINTOUT	195
GETDATE	196
DATETEST	197
NAMETEST	198
CHECKSTUB	199

```
STRUCTURE FOR FILE:  B:COSTBASE.DBF
FLD       NAME       TYPE WIDTH   DEC
001       CHECK:DATE    C    007
002       CHECK:NMBR    C    005
003       CLIENT        C    003
004       JOB:NMBR      N    003
005       AMOUNT        N    009    002
006       -             C    001
007       NAME          C    020
008       -             C    001
009       DESCRIP       C    020
010       -             C    001
011       BILL:DATE     C    007
012       BILL:NMBR     C    007
013       HOURS         N    006    002
014       EMP:NMBR      N    003
** TOTAL **                00094
(Indexed on NAME to B:$SUPP.NDX)


STRUCTURE FOR FILE:  B:POSTFILE.DBF
FLD       NAME       TYPE WIDTH   DEC
001       CHECK:DATE    C    007
002       CHECK:NMBR    C    005
003       CLIENT        C    003
004       JOB:NMBR      N    003
005       -             C    001
006       NAME          C    020
007       -             C    001
008       DESCRIP       C    020
009       AMOUNT        N    009    002
010       -             C    003
011       BILL:DATE     C    007
012       BILL:NMBR     C    007
013       HOURS         N    006    002
014       EMP:NMBR      N    003
** TOTAL **                00096
```

```
STRUCTURE FOR FILE:  B:BILLINGS.DBF
FLD       NAME       TYPE WIDTH   DEC
001       INV:NMBR     C    006
002       CLIENT       C    003
003       JOB:NMBR     N    003
004       -            C    001
005       INV:DATE     C    006
006       TAXABLE      N    009    002
007       SALES:TAX    N    009    002
008       TAXFREE      N    009    002
009       -            C    001
010       PO:NMBR      C    008
011       DESCRIP      C    027
012       MORE         L    001
** TOTAL **            00084
(Indexed on INV:NMBR to B:BILLINGS.NDX)


STRUCTURE FOR FILE:  B:INVOICES.DBF
FLD       NAME       TYPE WIDTH   DEC
001       INV:NMBR     C    006
002       CLIENT       C    004
003       INV:DATE     C    007
004       TAXABLE      N    009    002
005       SALES:TAX    N    009    002
006       TAXFREE      N    009    002
007       -            C    001
008       AMOUNT       N    009    002
009       AMT:RCD      N    009    002
010       -            C    001
011       DATE:RCD     C    007
** TOTAL **            00072
(Indexed on INV:NMBR to B:INVOICES.NDX)


STRUCTURE FOR FILE:  B:DEPOSITS.DBF
FLD       NAME       TYPE WIDTH   DEC
001       DEP:DATE     C    007
002       PAYER        C    020
003       -            C    001
004       PAY:NMBR     C    007
005       DEPOSIT      N    009    002
006       -            C    001
007       INV:NMBR     C    006
008       COMMENTS     C    021
** TOTAL **            00073
```

```
STRUCTURE FOR FILE:  B:CHECKFIL.DBF
FLD       NAME        TYPE WIDTH    DEC
001       CHECK:DATE   C    007
002       CHECK:NMBR   C    005
003       AMOUNT       N    009     002
004       -            C    001
005       BILL:NMBR    C    007
006       NAME         C    020
007       EMP:NMBR     N    003
008       -            C    001
009       CLIENT       C    003
010       JOB:NMBR     N    003
011       DESCRIP      C    020
012       BALANCE      N    009     002
** TOTAL **                00089

STRUCTURE FOR FILE:  B:INSERTS.DBF
FLD       NAME        TYPE WIDTH    DEC
001       IO:NMBR      C    005
002       MAGAZINE     C    014
003       -            C    001
004       ISSUE        C    006
005       -            C    001
006       CLIENT       C    003
007       JOB:NMBR     N    003
008       -            C    001
009       AD           C    015
010       SPACE        C    013
011       GROSS:COST   N    009     002
012       NET:COST     N    009     002
013       -            C    001
014       TIMES        C    003
015       IO:DATE      C    006
** TOTAL **                00091
(Indexed on IO:NMBR to B:INSERTS.NDX)
```

STRUCTURE FOR FILE: B:HOLD81.DBF

FLD	NAME	TYPE	WIDTH	DEC
001	CHECK:DATE	C	004	
002	-	C	001	
003	MARKER	C	001	
004	PAYROLL	N	009	002
005	FICA	N	008	002
006	FICASAL	N	009	002
007	FIT	N	009	002
008	SDI	N	007	002
009	SDISAL	N	009	002
010	SIT	N	009	002
011	UISAL	N	009	002
** TOTAL **			00076	

STRUCTURE FOR FILE: B:PERSONNE.DBF

FLD	NAME	TYPE	WIDTH	DEC
001	EMP:NMBR	N	003	
002	NAME	C	020	
003	ADDRESS	C	024	
004	CITY:STATE	C	020	
005	ZIP	C	005	
006	PH:NMBR	C	013	
007	SS:NMBR	C	009	
008	M:S:H	C	001	
009	DEDUCTS	N	002	
010	PAY:RATE	N	007	002
011	FICA	N	008	002
012	YTDFICA	N	008	002
013	FIT	N	009	002
014	YTDFIT	N	009	002
015	SDI	N	007	002
016	YTDSDI	N	007	002
017	SIT	N	009	002
018	YTDSIT	N	009	002
019	NET:PAY	N	009	002
020	QTDSAL	N	009	002
021	YTDSAL	N	009	002
022	PAID	L	001	
023	START:DATE	C	006	
024	RATIO	N	005	003
** TOTAL **			00209	

```
STRUCTURE FOR FILE:  B:SUPPLIER.DBF
FLD       NAME       TYPE WIDTH    DEC
001      SUPPLIER     C    030
002      ADDRESS      C    024
003      CITY         C    016
004      STATE        C    002
005      ZIP          C    005
006      PHONE:NMBR   C    008
007      AREA:CODE    C    003
** TOTAL **              00089
(Indexed on SUPPLIER to B:SUPPLIER.NDX)
```

The agency accounting system uses three standard report forms. The first one is for media and is filled out completely. The remaining two are skeletons, showing only the answers to the questions asked by dBASE II.)

```
ENTER REPORT FORM NAME: Media
ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE, W=PAGE WIDTH
M=50 <enter>
PAGE HEADING? (Y/N) n
DOUBLE SPACE REPORT? (Y/N) n
ARE TOTALS REQUIRED? (Y/N) y
SUBTOTALS IN REPORT? (Y/N) n
COL     WIDTH,CONTENTS
001     6,IO:NMBR
ENTER HEADING: IO #
002     15,MAGAZINE
ENTER HEADING: MAGAZINE
003     7,ISSUE
ENTER HEADING: ISSUE
004     6,CLIENT+STR(JOB:NMBR,3)
ENTER HEADING: JOB #
005     15,AD
ENTER HEADING: AD
006     9,GROSS:COST
ENTER HEADING: $GROSS
ARE TOTALS REQUIRED? (Y/N) Y
007     <enter>
```

The above dialog generate this report form:

PAGE NO. 00001

IO #	MAGAZINE	ISSUE	JOB #	AD	$GROSS
2787	EDN	JAN 7	SPI678	FLAT MAN	3225.00
2788	MICROWAVES	JAN	POM772	FISHERMAN GUNNS	2500.00
2789	MICROWAVES	MAR	POM639	COP: GUNNS	2500.00
2790	ELECTRONICS	JAN	PSS754	NICE LITTLE BK	5900.00
2791	BYTE	FEB	SFT789	BILGE PUMP	2932.00

...(etc.)...

The other two report forms are:

JOBCOSTS.FRM

```
n
n
y
n
9,BILL:DATE
DATE
22,NAME
SUPPLIER.....
17,DESCRIP
DESCRIPTION.....
12,AMOUNT
   AMOUNT
Y
```

BILLED.FRM

```
n
n
Y
n
8,INV:DATE
DATE
8,INV:NMBR
INVOICE
17,DESCRIP
DESCRIPTION
10,TAXABLE
TAXABLE
Y
10,SALES:TAX
SALES TAX
Y
10,TAXFREE
 TAX-FREE
Y
```

SYSTEM CONSTANTS

System constants are kept in a file called B:Constant.MEM. These are called out, used and updated where appropriate from within a number of programs within this accounting system. Constants are kept in a single file so that when any of them change (new tax rates, new year, etc.), they need only be changed in one location to update the entire system.

NEXTCHECK	(C)	3565	Keep checkbook current in PAYBILLS, PAYROLL and DEPOSITS command files.
MBALANCE	(N)	23921.18	
THISYEAR	(N)	81	Used by GETDATE, DATECHECK and PAYROLL files.
MINYEAR	(N)	79	
NEXT:IO	(C)	2885	Next insertion order number (IOPOST.CMD).
NEXT:INV	(C)	10623	Next invoice number (INVOICES.CMD).
FICACUT	(N)	0.0665	Entire grouping is used in the PAYROLL file. Easy to update every year because the values are not sprinkled throughout the programs.
FICAEND	(N)	29700.00	
MAXFICA	(N)	1975.05	
SDICUT	(N)	0.006	
SDIEND	(N)	14900.00	
MAXSDI	(N)	89.40	
UIEND	(N)	6000	
COMPLETED	(C)		
PREVDATE	(C)	810814	
MAXEMPL	(N)	14	Highest employee number. Also used in POSTTIME and TIMECALC programs.

```
   **********  ACCOUNTS COMMAND FILE  **********
* THIS IS THE CONTROL MODULE FOR ALL THE PROCEDURES USED IN THE ACCOUNTING
* FUNCTIONS.  (MENU DRIVEN).  The operator is given a choice of major
* functions.  The menu selection here calls up menus of sub-functions
* to as many levels as necessary.  A package of utility functions is
* also provided as part of the overall system for file maintenance, etc.
*********************************************************

SELECT PRIMARY
CLEAR
SET TALK OFF
SET EJECT OFF
SET MARGIN TO 38
SET RAW ON

*************************************************************************
* This group of commands is hardware-dependent, may be eliminated
* down to the next row of asterisks.
SET PRINT ON
   * Sets (8-1)=7 lines/inch and 89 lines page, Diablo 1650/sheet feeder
   ?? CHR(27)+CHR(30)+CHR(8)
   ?? CHR(27)+CHR(12)+CHR(89)
SET PRINT OFF
* End printer setup commands.
*****************************************************************

STORE T TO Accounting
DO WHILE Accounting
   ERASE
   ?
   ?
   ? '1> ENTER BILLS & TIME SHEETS          6> REPORTS & PRINTOUTS'
   ? '                                         Job cost & billing summaries'
   ? '2> PAY BILLS & SALARIES                  Find & Edit bills by name'
   ? '                                         Review/print databases'
   ? '3> DEPOSITS & CHECKBOOK                  Quarterly Sales Tax'
   ?
   ? '4> MEDIA INSERTION ORDERS'
   ?
   ? '5> CLIENT BILLINGS & INVOICES'
   ?
   ?
   ? '                  Pick a number (Q to QUIT)'
   WAIT TO Action

   IF !(Action) = 'Q'
      * This sequence homes the cursor and erases the screen on the IBM 3101
      STORE chr(27) + chr(76) + chr(27) + chr(89) + chr(0) TO q
      QUIT TO '&q'
   ELSE
      IF Action = '1'
         DO CostMenu
      ELSE
         IF Action = '2'
            DO PayMenu
         ELSE
            IF Action = '3'
               DO DepMenu
            ELSE
               IF Action = '4'
                  DO IOMenu
               ELSE
                  IF Action = '5'
                     DO InvMenu
```

```
                    ELSE
                       IF Action = '6'
                          DO ReportMe
                       ELSE
                          If Action = '7'
                             ERASE
                             @ 5,10 SAY '     HELP file not ready yet.'
                             ? '  <Return> to continue.'
                             WAIT
                          ENDIF 7
                       ENDIF 6
                    ENDIF 5
                 ENDIF 4
              ENDIF 3
           ENDIF 2
        ENDIF 1
     ENDIF Q
     STORE T TO Accountin
ENDDO Accounting
```

```
      **********  COSTMENU COMMAND FILE  **********
* This is one level down from the Accounts.Cmd control module.
* Sele ions are refinements that relate to costs for client-related
* jobs or agency overhead.
*       The main database is called CostBase.Dbf and is kept on disk B.
* Costs are not entered directly into the CostBase, however, because this leads
* to data contamination and all sorts of problems fixing the errors.  Instead,
* supplier bills and agency time sheets are posted into an interim file called
* PostFile.Dbf.  In here, they can be reviewed and edited as necessary.
*       When all the cost entries are confirmed as being correct, they are
* transferred to the CostBase by using the update procedure (selection 5).
*************************************************************************

STORE T TO Posting
DO WHILE Posting
   ERASE
   @  2,20 SAY '    1> UNTAXED ITEMS USED BY AGENCY'
   @  4,20 SAY '    2> ENTER SUPPLIER BILLS'
   @  6,20 SAY '    3> ENTER EMPLOYEE TIME SHEETS'
   @  8,20 SAY '    4> EDIT the POSTFILE'
   @ 10,20 SAY '    5> REVIEW/PRINT the POSTFILE'
   @ 12,20 SAY '    6> UPDATE THE COSTBASE'
   @ 14,20 SAY '    7> WIPE OUT DELETED RECORDS IN POSTFILE'
   @ 16,20 SAY '         <RETURN>'
   WAIT TO Action
   ERASE

   IF Action = '1'
      ERASE
      @  4,10 SAY 'This program accepts bills for items that the agency bought'
      ? '           without paying sales tax, but will use internally, rather'
      ? '           than for a job that will be billed to a client.  This would'
      ? '           include equipment bought out of state and locally bought'
      ? '           materials NOT used in client jobs and NOT taxed.'
      ?
      ? '                  DO NOT ENTER ANY OTHER BILLS!'
      ?
      ? 'Do you want to continue (Y or N)?'
      WAIT TO GoAhead
      IF !(GoAhead) = 'Y'
         DO UseTax
      ELSE
         RELEASE All
      ENDIF
   ELSE
      IF Action = '2'
         ERASE
         @  4,10 SAY '   CHECK ALL THE BILLS BEFORE ENTERING THEM.'
         ? '           If any of the bills are for items used by the agency'
         ? '           but sales tax was not paid, select OPTION 1 from the'
         ? '           entry menu.  <Return> to continue.'
         ?
         WAIT
         DO CostBills
      ELSE
         IF Action = '3'
            DO CostTime
         ELSE
            IF Action = '4'
               STORE "Y" TO Changing
               DO WHILE !(Changing)='Y'
                  USE B:PostFile
                  IF EOF
                     ? 'There are no entries in the POSTING file.'
```

```
            ? '<Return> to continue.'
            WAIT
            STORE "N" TO Changing
         ELSE
            GO BOTTOM
            ERASE
            @  3,10 SAY 'EDITING BILLS ENTERED.'
            @  5,10 SAY 'There are '+STR(#,5)+' file entries.'
            @  6,10 SAY 'Which entry do you want to EDIT?'
            ACCEPT TO Number
            IF VAL(Number) <= 0 .OR. VAL(Number) > #
               ?
               ?
               ? 'Out of range: do you want to continue (Y or N)?'
               WAIT TO Changing
            ELSE
               Edit &Number
               REPLACE Name WITH !(Name), Descrip WITH !(Descrip),;
                  Client WITH !(Client), Bill:Nmbr WITH !(Bill:Nmbr)
               ?
               ? 'Do you want to edit any other entries (Y or N)?'
               WAIT TO Changing
            ENDIF
         ENDIF
      ENDDO Changing
      RELEASE All
   ELSE
      IF Action ='5'
         STORE 'Y' TO Reviewing
         DO WHILE !(Reviewing)='Y'
            USE B:Postfile
            COUNT FOR .NOT. * TO Any
            IF Any = 0
               ? 'No unposted entries in the POSTING file.
               ? '<Return> to continue.'
               WAIT
               STORE "N" TO Reviewing
            ELSE
               ERASE
               ? 'There are '-STR(Any,5)+' unposted entries.'
               ? 'Do you want to print them, too (Y or N)?'
               WAIT TO Output

               IF !(Output)='Y'
                  SET PRINT ON
               ENDIF
               ? '                JOB     NAME                    DESCRIP';
                 +'TION              AMOUNT     DATE    NUMBER'
               ?
               STORE 'OFF' TO Condition
               STORE '0' TO Number
               DO Printout

               ? "That's all the unposted entries."
               ? 'Want to see them again (Y or N)?'
               ? '(To see deleted records, choose "Edit".)'
               WAIT TO Reviewing
            ENDIF
         ENDDO Reviewing
         RELEASE all
      ELSE
         IF Action = '6'
            DO CostUpdate
         ELSE
```

```
                        IF Action = '7'
                           ? 'This destroys all records in the PostFile.'
                           ? 'Do you want to do this (Y or N)?'
                           WAIT TO WipeOut
                           IF !(WipeOut) = 'Y'
                              USE B:PostFile
                              PACK
                           ENDIF
                           RELEASE ALL
                        ELSE
                           RELEASE All
                           RETURN
                        ENDIF 7
                     ENDIF 6
                  ENDIF 5
               ENDIF 4
            ENDIF 3
         ENDIF 2
      ENDIF 1
      STORE T TO Posting
ENDDO Posting
```

```
          ***********  USETAX COMMAND FILE  ***********
* This file accepts inputs for supplier bills when the agency has bought
* an item without paying a use tax on it.
*      The item or items are added to the Invoices file (not Billings),
* then are used by the SalesTax program so that the Quarterly Sales Tax
* report can be prepared by the computer.
*
* A temporary file called GetBills is used for data entry because the operator
* can decide to quit on an incomplete entry, which is marked for deletion.
* When the data is APPENDed to the PostFile, these entries are eliminated (the
* APPEND command does not transfer records marked for deletion).  An entry must
* include at least the name of a supplier and the amount of the bill.  If these
* are not both supplied, the entry is flagged for correction or deletion.
*************************************************************************

ERASE
@  5,20 SAY 'AGENCY USE TAX PROCEDURE'
?
USE B:PostFile
COPY STRUCTURE TO GetBills

USE GetBills
STORE 'Y' TO Bills
DO WHILE !(Bills) <>
   APPEND BLANK
   STORE STR(#,5) to Number
   REPLACE Client WITH 'OFC'
   STORE T TO Entering
   DO WHILE Entering
      ERASE
      @  1,0 SAY 'ENTER ONLY UNTAXED ITEMS NOT USED FOR CLIENT JOBS.'
      @  3,0 SAY '    RECORD NUMBER:' +  Number
      @  4,0 SAY '           CLIENT:' +  Client + ':'
      @  5,0 SAY '       JOB NUMBER' GET Job:Nmbr
      @  6,0 SAY '           AMOUNT' GET Amount
      @  7,0 SAY '      BILL NUMBER' GET Bill:Nmbr
      @  8,0 SAY '        BILL DATE' GET Bill:Date
      @  9,0 SAY '    SUPPLIER NAME' GET Name
      READ
      REPLACE Name WITH !(Name),Descrip WITH 'USE TAX ENTRY';
              Bill:Nmbr WITH !(Bill:Nmbr)
      @   7,17 SAY Bill:Nmbr
      @   9,17 SAY Name
      @  10,17 SAY Descrip

      STORE ' ' TO Getting
      IF Job:Nmbr <= 0  .OR. Job:Nmbr > 99
         @  12,0
         ? '         The JOB NUMBER entry is wrong.'
         ? '         Agency jobs are from 1 through 99.'
         ? '         F if FINISHED,'
         ACCEPT '      <Return> to change.' TO Getting
      ELSE
         IF Amount = 0 .OR. Name <= '
            ?
            ?
            ? '          AMOUNT or NAME missing.'
            ? '          F if  FINISHED,'
            ACCEPT '       <Return> to change.' TO Getting
         ELSE
            @  12,5 SAY '    C to CHANGE.'
            @  13,5 SAY '    F if FINISHED,'
            ACCEPT '       <Return> to continue.' TO Bills
```

```
            IF !(Bills)='C'
               STORE T TO Entering
            ELSE
               STORE F TO Entering
            ENDIF
         ENDIF amount or name
      ENDIF client or job number

      IF !(Getting)= 'F'
         DELETE RECORD &Number
         STORE F TO Entering
         STORE 'F' TO Bills
      ENDIF
   ENDDO Entering
ENDDO Bills

COUNT FOR .NOT. * TO Any
IF Any = 0
   ?
   ? '          No valid entries to add to the files.'
   ? '       <Return> to the menu.'
   WAIT
ELSE

   RESTORE FROM B:Constant
   STORE 'Bill:Date' TO Date
   DO DateTest

   * Checks names against a list of suppliers to catch spelling and
   * abbreviation inconsistencies.
   DO NameTest

   ERASE
   @  3,25 SAY ' *** DO NOT INTERRUPT ***'
   @  5,25 SAY ' UPDATING THE POSTING FILE'
   USE B:PostFile
   APPEND FROM GetBills

   * The following loop transfers the bills just entered into the Invoices
   * file.  The amount of the bill is entered in the "Taxable" column.  The
   * job number is entered into the Invoice Number column.  Since invoice
   * have 5 digits, while job numbers are under 1000, we use this to sepa-
   * rate the two types of entries later in the SalesTax.CMD file.
   * PRIMARY and SECONDARY work areas are used to step through the GetBills
   * file one entry at a time.
   USE GetBills
   SELECT SECONDARY
   USE B:Invoices
   SELECT PRIMARY
   DO WHILE .NOT. EOF
      IF *
         SKIP
      ELSE
         SELECT SECONDARY
         APPEND BLANK
         REPLACE Inv:Nmbr WITH STR(Job:Nmbr,3), Inv:Date WITH Bill:Date,;
                 Taxable WITH P.Amount, Date:Rcd WITH 'USE TAX'
         SELECT PRIMARY
         SKIP
      ENDIF
   ENDDO

ENDIF
```

```
USE
DELETE FILE GetBills
RELEASE All
RETURN
```

```
     ***********  COSTBILLS COMMAND FILE  ***********
* This file accepts inputs for supplier bills.
* A temporary file called GetBills is used for data entry because the operator
* can decide to quit on an incomplete entry, which is marked for deletion.
* When the data is APPENDed to the PostFile, these entries are eliminated (the
* APPEND command does not transfer records marked for deletion).  An entry must
* include at least the name of a supplier and the amount of the bill.  If these
* are not both supplied, the entry is flagged for correction or deletion.
*************************************************************************

ERASE
@ 5,20 SAY 'SUPPLIER BILLS'
USE B:PostFile
COPY STRUCTURE TO GetBills

USE GetBills
STORE 'Y' TO Bills
DO WHILE !(Bills) <> 'F'
   APPEND BLANK
   STORE STR(#,5) to Number

   STORE T TO Entering
   DO WHILE Entering
      ERASE
      @ 1,0 SAY '    RECORD NUMBER: '+Number
      @ 3,0 SAY '           CLIENT' GET Client
      @ 4,0 SAY '       JOB NUMBER' GET Job:Nmbr
      @ 5,0 SAY '           AMOUNT' GET Amount
      @ 6,0 SAY '      BILL NUMBER' GET Bill:Nmbr
      @ 7,0 SAY '        BILL DATE' GET Bill:Date
      @ 8,0 SAY '    SUPPLIER NAME' GET Name
      @ 9,0 SAY '      DESCRIPTION' GET Descrip
      READ
      REPLACE Client WITH !(Client),Name WITH !(Name),Descrip;
              WITH !(Descrip), Bill:Nmbr WITH !(Bill:Nmbr)
      @ 3,17 SAY Client
      @ 8,17 SAY Name
      @ 9,17 SAY Descrip

      STORE ' ' TO Getting
      IF $(Client,1,1) = ' ' .OR. $(Client,2,1) = ' ' .OR. $(Client,3,1) = ' ';
         .OR. Job:Nmbr <= 0
         @ 12,0
         ? '      CLIENT or JOB NUMBER wrong.'
         ? '      F if FINISHED,'
         ACCEPT '   <Return> to change.' TO Getting
      ELSE
         IF Amount = 0 .OR. Name <= '
            ?
            ?
            ? '         AMOUNT or NAME missing.'
            ?
            ? '         F if  FINISHED,'
            ACCEPT '      <Return> to change.' TO Getting
         ELSE
            @ 12,5 SAY '    C to CHANGE,'
            @ 13,5 SAY '    F if FINISHED,'
            ACCEPT '      <Return> to continue.' TO Bills

            IF !(Bills)='C'
               STORE T TO Entering
            ELSE
               STORE F TO Entering
            ENDIF
```

```
         ENDIF amount or name
      ENDIF client or job number

      IF !(Getting)= 'F'
         DELETE RECORD &Number
         STORE F TO Entering
         STORE 'F' TO Bills
      ENDIF
   ENDDO Entering
ENDDO Bills

COUNT FOR .NOT. * TO Any
IF Any = 0
   ? 'No entries to add to the Cost Base.'
   ?. '<Return> to the menu.'
   USE
   WAIT
ELSE

   RESTORE FROM B:Constant
   STORE 'Bill:Date' TO Date
   DO DateTest

   * Checks names against a list of suppliers to catch spelling and
   * abbreviation inconsistencies.
   DO NameTest

   ERASE
   @ 3,25 SAY ' *** DO NOT INTERRUPT ***'
   @ 5,25 SAY ' UPDATING THE POSTING FILE'
   USE B:PostFile
   APPEND FROM GetBills
ENDIF

USE
DELETE FILE GetBills
RELEASE All
RETURN
```

```
      **********  COSTTIME COMMAND FILE  **********
* Accepts time sheet entries for employees using a temporary
* file called GetTime. For data entry.
*      GetTime is used because the operator can decide to quit on an incomplete
*  entry.  In that case, the entry is marked for deletion, and when the data is
* APPENDed to the PostFile, these entries are eliminated (the APPEND command
* does not transfer records marked for deletion).
*     After all entries are made, entries are checked for the
* correct range of employee numbers and to see that hours have
* been entered.   Using GetTime, we can check the entries without
* having to go through the entire PostFile.
*      After verifying that the dates are in the right format and
* checking the names against our Suppliers file, the billing amounts
* are computed.
*     The records are then transferred to the CostFile and the
* temporary file GetTime is deleted.
*****************************************************************

@  0,25 SAY ' TIME SHEETS '

RESTORE FROM B:Constant
USE B:PostFile
COPY STRUCTURE TO GetTime

USE GetTime
STORE 'Y' TO Time
DO WHILE !(Time) <>'F'
   APPEND BLANK
   STORE STR(#,5) TO Number

   STORE T TO Entering
   DO WHILE Entering
      ERASE
      STORE F TO Entering
      @ 1,0 SAY '    RECORD NUMBER: '+Number
      @ 3,0 SAY '      DATE WORKED' GET Bill:Date
      @ 4,0 SAY '           CLIENT' GET Client
      @ 5,0 SAY '       JOB NUMBER' GET Job:Nmbr
      @ 6,0 SAY '     HOURS WORKED' GET Hours
      @ 7,0 SAY ' EMPLOYEE NUMBER' GET Emp:Nmbr
      @ 8,0 SAY '    EMPLOYEE NAME' GET Name
      READ

      REPLACE Check:Nmbr WITH '----', Check:Date WITH Bill:Date,;
              Client WITH !(Client), Name WITH !(Name)
      @ 4,17 SAY Client
      @ 8,17 SAY Name

      * The following sequence of IF statments flags all entry errors, ther
      * gives the operator the choice of fixing them or ending the procedure.

      ?
      IF $(Client,1,1) =' '.OR. $(Client,2,1)=' ' .OR. $(Client,3,1) =' '
         ? '      CLIENT must have three letters.'
         STORE T TO Entering
      ENDIF

      IF Job:Nmbr < 100
         ? '      JOB # is not for a client job.'
         ? '      Is this right (Y or N)?'
         WAIT TO Ask
         IF !(Ask) <> 'Y'
            STORE T TO Entering
         ENDIF
```

```
      ENDIF

      IF .NOT. (Hours > 0)
         ? '      HOURS must be entered.
         STORE T TO Entering
      ENDIF

      IF .NOT.(Emp:Nmbr>0 .AND. Emp:Nmbr<=MaxEmpl)
         ? '      EMPLOYEE # out of range.
         STORE T TO Entering
      ENDIF

      IF $(Name,1,1) = ' '
         ? '      NAME must not start with a blank.'
         STORE T TO Entering
      ENDIF

      IF Entering
         ?
         ?
         ? '      F if FINISHED,'
         ACCEPT '   <Return> to change' TO Time

         * If the operator decides to quit on an incomplete entry, it is
         * marked for deletion so that it is not transferred to the PostFile.
         IF !(Time) = 'F'
            DELETE RECORD &Number
            STORE F TO Entering
         ENDIF
      ELSE
         ?
         ? '      C to CHANGE,'
         ? '      F if FINISHED,'
         ACCEPT '   <Return> to continue' TO Time
         IF !(Time) = 'C'
            STORE T TO Entering
         ENDIF
      ENDIF
   ENDDO Entering
ENDDO Time


COUNT FOR .NOT. * TO Any
IF Any = 0
   ERASE
   @ 3,0 SAY '    No entries to add to the CostFile. '
   ? ' <Return> to the menu. '
   USE
   WAIT
ELSE

   * The test for the date needs the name of the date field to to be tested.
   STORE 'Bill:Date' TO Date
   DO DateTest

   * Checks names against a list of suppliers to catch spelling and
   * abbreviation inconsistencies.
   DO NameTest

   * Verifies match between employee name and number, then computes the amount
   * to be billed for the employee's time based on his salary.
   DO TimeCalc

   ERASE
```

```
   @ 3,25 SAY " *** DO NOT INTERRUPT ***"
   @ 5,25 SAY " UPDATING THE POSTING FILE"
   USE B:PostFile
   APPEND FROM GetTime
ENDIF

DELETE FILE GetTime
RELEASE All
RETURN
```

```
     ********** COSTUPDATE COMMAND FILE **********
*       Records from the COSTFILE are added to  the COSTBASE.
* This step is so critical to data integrity that we: use a password
* to prevent accidental access; verify dates; check the names of suppliers;
* and compute time charges if necessary.  Notice that these are done by
* simply calling the utility command files.
*       The PostFile has all its records marked for deletion after they
* have been posted (can still be recovered).
*************************************************************************

SET TALK OFF

@  4,12 SAY '*************************************************'
@  6,12 SAY 'MAKE CERTAIN EVERYTHING IN THE POSTFILE IS CORRECT'
@  8,12 SAY '    BEFORE ENTERING THE CODE TO CONTINUE'
@ 10,12 SAY '*************************************************'
SET CONSOLE OFF
ACCEPT TO Lock
SET CONSOLE ON

IF !(Lock) <> 'H'
   @ 12,12 SAY '     UNAUTHORIZED ACCESS ATTEMPTED.'
   @ 14,12 SAY 'YOU HAVE 6 SECONDS BEFORE THE EXPLOSION.'
   STORE 1 TO X
   DO WHILE X < 150
      STORE X + 1 TO X
   ENDDO
   RELEASE Lock
   RETURN
ELSE
   ERASE
   @ 5,20 SAY 'Checking bills in the POSTING File:'
   USE B:PostFile
   COUNT FOR .NOT. * TO None
   IF None = 0
      @ 6,20 SAY 'No new entries in the POSTING file.'
      @ 7,20 SAY '<Return> to continue.'
      WAIT
   ELSE
      GO TOP
      RESTORE FROM B:Constant
      STORE 'Bill:Date' TO Date
      DO DateTest
      DO NameTest
      DO TimeCalc
      ERASE
      @ 5,20 SAY ' ***  DO NOT INTERRUPT  ***'
      @ 6,20 SAY 'Posting COSTS to the Costbase.'
      * Save the number of the last record in Costbase
      USE B:CostBase
      GO BOTTOM
      STORE # TO LastReco


      USE B:Costbase INDEX B:$Supp
      APPEND FROM B:PostFile

      USE B:PostFile
      DELETE ALL
   ENDIF
ENDIF

RELEASE All
RETURN
```

```
        **********  PAYMENU COMMAND FILE  **********
* This is a sub-module of the Accounts.CMD file and provides choices
* as to which checks are to be  prepared for posting and printing.
*       Paying salaries has another menu level to allow partial payments
* to selected employees (e.g., leave of absence, when an employee does not
* work a full two week stretch, etc.)
*       The checkbook balance and next check number must be confirmed before
* either of the procedures can be performed.
************************************************************************

RESTORE FROM B:Constant
ERASE
@ 3, 0 SAY 'CHECK NUMBER: '+NextCheck+'          BALANCE: '+STR(MBalance,9,2)
?
? '      Do these match the checkbook?'
? '      C to CONTINUE,'
? '   <Return> to change.'
?
WAIT TO Continue

IF !(Continue) <> 'C'
   RELEASE All
   RETURN
ENDIF

STORE T TO Paying
DO WHILE Paying
     ERASE
     @  5,20 SAY '   1> PAY BILLS
     @  7,20 SAY '   2> PAY SALARIES
     @ 10,20 SAY '      <RETURN>'
     WAIT TO Action

     IF Action = '1'
        USE B:PostFile

        * Can abort if any entries in the Postfile.
        COUNT FOR .NOT. * TO Any
        IF Any = 0
           DO PayBills
        ELSE
           ?
           ? 'The POSTING file has '-STR(Any,5)+' bills in it.'
           ? 'Do you still want to pay bills now (Y or N)?'
           WAIT TO Continue
           IF !(Continue) = 'Y'
              DO PayBills
           ELSE
              RELEASE All
           ENDIF
        ENDIF
     ELSE
        IF Action = '2'
           DO PayEmps
        ELSE
           RELEASE All
           RETURN
        ENDIF 2
     ENDIF 1
     STORE T TO Paying
ENDDO Paying
```

```
      ***********  PAYBILLS COMMAND FILE  ***********
* Before this procedure can be accessed, the check number and balance must
* be verified in the PAYMENU command file.
*       This is one of the longer files, but the individual portions of it are
* not too complicated.  Repetitive procedures in the main loop (controlled
* by the variable "Finished") could have been put in separate command files
* to make this file easier to understand and maintain, but this way it
* minimizes disk accesses and increases speed.
*       This file finds bills to be paid in the CostBase, generates the next
* check number, writes a check in the CheckFil and maintains the checkbook
* balance.
*       The next check number and checkbook balance are recalled from a file
* called Constant.MEM.  The final values for both of these are stored in the
* same file after all the bills have been paid.
*       The date is entered once at the start of the procedure, then
* is automatically inserted into each entry.  The date is checked to
* see that it is in the YYMMDD format, and that the values are within
* possible limits (month from 1 to 12, day from 1 to 31, year=ThisYear).
*       Entries must include at least the name of the party being paid.
*       Balances are automatically computed and shown to the operator.
*       Check numbers are automatically assigned by the computer.
*       If several entries are made against a single check number (the
* operator has this option), these are added and shown as a single
* item in the printout.
*************************************************************************

RESTORE FROM B:Constant
DO GetDate

SELECT PRIMARY
USE B:CostBase INDEX B:$Supp

* Initialize.  "New" is used to determine whether the program should generate
* a new check number or use the old one (where several bills to a single
* supplier are being paid).  "Finished" is the control variable that determines
* whether we should run through the procedure again, or are done paying bills.
STORE 'N' TO New
STORE 'N' TO Finished
DO WHILE !(Finished) <> 'F'
   STORE "C" TO Entering
   DO WHILE !(Entering) = 'C'
      ERASE
      @ 3, 0 SAY 'CHECK NUMBER: '+NextCheck+'      BALANCE: '+STR(MBalance,9,2)
      ? CHR(7)
      @ 4,0
      ACCEPT '    MAKE CHECK TO ' TO MName
      ACCEPT '   INVOICE NUMBER ' TO MBill:Nmbr
      ACCEPT '     ENTER AMOUNT ' TO Temp
      STORE !(MName) TO MName
      STORE !(MBill:Nmbr) TO MBill:Nmbr
      STORE VAL(Temp) TO MAmount
      STORE MAmount*1.00 TO MAmount
      @  6,19 SAY MName
      @  7,19 SAY MBill:Nmbr
      @  8,19 SAY MAmount
      @ 11, 0 SAY '         C to CHANGE,
      ? '      <Return> to continue.'
      WAIT TO Entering
   ENDDO Entering

   IF LEN(MName) > 10
      STORE $(MName,1,10) TO Key
   ELSE
      STORE MName TO Key
```

```
ENDIF

IF Key > ' '
   STORE T TO Looking
   @ 11, 0 SAY "I'M LOOKING, I'M LOOKING!!"
   @ 12,0
   @ 13,0
   STORE 0 TO Start
   FIND &Key
   IF # = 0
      ?
      ? "•  GEE, I CAN'T FIND THE NAME.  Please check the spelling."
      ? "   Or maybe it hasn't been posted to the COSTBASE yet."
      ? '<Return> to continue.'
      WAIT
      ERASE
   ELSE
      DO PayFind
   ENDIF there is an unpaid bill for the supplier

   * "Start" is brougnt in from PayFind.CMD.  If we started at the first
   * entry for a name (had only the name), Start=0.  If we had more than
   * the name, Start contains the record numerber we started on.  Since this
   * could be in the middle of the listing, we use "Counter" so that we can
   * come back to the top of the listing for the name once.
   IF Start > 0
      STORE 0 TO Counter
   ELSE
      STORE 1 TO Counter
   ENDIF

   STORE ' ' TO Confirm
   DO WHILE !(Confirm) <> 'P' .AND. .NOT. Looking
      @  9,0
      ? 'RECORD      NAME                          AMOUNT  BILL #';
        +'   DATE'
      ?
      DISPLAY '    '+Name, Amount, Bill:Nmbr, Bill:Date
      ?
      ? CHR(7)
      ? '            P to PAY this bill,'
      ? '            Q to QUIT without paying,'
      ? '         <Return> to continue.'
      ACCEPT '            ' TO Confirm

      IF !(Confirm) = 'Q'
         IF !(New) = 'S'
            STORE STR(VAL(NextCheck)+1,4) TO NextCheck
         ENDIF
         STORE ' ' TO New
         STORE T TO Looking
      ELSE
         IF !(Confirm) = 'P'
            STORE STR(#,5) TO Found
            REPLACE Check:Date WITH Date, Check:Nmbr WITH NextCheck
            STORE (MBalance-Amount) TO MBalance

            SELECT SECONDARY
            USE B:Checkfil
            APPEND BLANK
            REPLACE Check:Date WITH P.Check:Date, Name WITH P.Name,;
                    Check:Nmbr WITH P.Check:Nmbr, Balance WITH MBalance,;
                    Amount WITH P.Amount, Bill:Nmbr WITH P.Bill:Nmbr
            SELECT PRIMARY
```

```
   ERASE
   @ 3, 0 SAY 'CHECK WRITTEN: '+NextCheck+;
              '     NEW BALANCE: '+STR(MBalance,9,2)
   ?
   DISPLAY 'PAYMENT MADE: '+Check:Date, Name, Amount, Bill:Nmbr,;
           Bill:Date OFF
   ?
   ? '       S for SAME SUPPLIER (Repeats check #)'
   ? CHR(7)
   ACCEPT '     <Return> to continue.' TO New
   IF !(New) <> 'S'
      STORE STR(VAL(NextCheck)+1,4) TO NextCheck
   ELSE
      STORE ' ' TO Confirm
   ENDIF
ENDIF

IF !(New) = 'S' .OR. !(Confirm) <> 'P'
   * If Confirm <> 'P', we rejected the first unpaid bill that was
   * shown.  Rather than going back to the beginning, the loop
   * below SKIPs to the next INDEXed name until we find an unpaid
   * bill, or go beyond the records for the name we are paying.
   *       The same applies if we want to pay another bill to the
   * same supplier (New='S').  Since we are in the file on the name
   * we want we SKIP to the next record until we find an unpaid
   * bill or run out of records for that name.
   *       If we had only the name and started with the first unpaid
   * bill we stop now since we have looked at all the unpaid bills
   * for that supplier.
   *       If we could have entered the list of records for this
   * supplier in the middle (more than the name provided), we look
   * at the unpaid bills between where we are and the end of the
   * list, then go up to the first entry for that name and check
   * all of the unpaid bills that we had previously skipped past.
   * This is controlled by Counter.
   *       After the second FIND in the command file (below), we
   * stop looking when the record number we are on is greater than
   * or equal to the number of the record we start on (Start).

   SKIP
   DO WHILE Check:Nmbr <> ' ' .AND. Name=Key .AND. .NOT. EOF
      SKIP
   ENDDO

   * We enter this loop when we reach the end of the records with
   * names that match the one we are looking for.  If we started
   * with the first unpaid bill, the record number is greater than
   * Start (because Start=0) and Counter=1 (because we set it to
   * that value).  The second IF below is True and we terminate the
   * search.
   *       If Start>0, Counter=0 the first time we run out of
   * records with a matching name, so the program does the ELSE
   * commands below.
   *       Start is still >0 and Count is now 1, so the last term in
   * the first IF applies.  On this second pass when we get to a
   * record number >=Start, we drop into the loop and do the IF to
   * terminate the search because we have now looked at all the
   * unpaid bills for the name we entered.
   IF EOF .OR. Name <> Key .OR. (# >= Start .AND. Start <> 0;
                                 .AND. Counter >0)
      IF (# >= Start  .AND. Counter > 0)
         STORE T TO Looking
         @ 4, 0
         ? chr(27)+chr(74)
```

```
                    ? '     We have now looked at all the entries for '+ MName
                    ? '  <Return> to continue.'
                    ? CHR(7)
                    IF !(New)='S'
                       STORE STR(VAL(NextCheck)+1,4) TO NextCheck
                       STORE 'N' to New
                    ENDIF
                    WAIT
                 ELSE
                    STORE Counter + 1 TO Counter
                    @ 13, 0
                    @ 16, 0 SAY "I'M WORKING AS FAST AS I CAN -- HANG ON! "
                    FIND &Key
                    DO WHILE Check:Nmbr <> ' '
                       SKIP
                    ENDDO
                 ENDIF
              ENDIF
           ENDIF is it the right record
        ENDIF
     ENDDO Confirm the record
  ENDIF

  IF !(New) <> 'S'
     @ 4, 0
     ? chr(27)+chr(74)
     ? '         F if FINISHED,
     ? CHR(7)
     ACCEPT '      <Return> to continue.' TO Finished
  ENDIF
ENDDO Finished

RELEASE MName, MBill:Nmbr, Key, MAmount, Start, Found, Looking, New, Change,;
        Entering, Counter, Temp, Abort, Continue, Finished, Confirm, Date
SAVE TO B:Constant

USE B:Checkfil
COUNT FOR .NOT. * TO Any
ERASE
@ 3,0
IF Any=0
   ? '         No new checks in the checkfile.'
   ? '      <Return> to continue.'
   WAIT
ELSE
   ? 'There are '-STR(Any,5)+' new checks in the CheckFile.'
   ? 'Do you want to print the checkstubs now (Y or N)?'
   ?
   WAIT TO Hardcopy
   IF !(Hardcopy) = 'Y'
      DO NameTest
      DO CheckStub
   ENDIF
ENDIF

RELEASE All
RETURN
```

```
            **********  PAYFIND COMMAND FILE  **********
* This file is called by the PAYBILLS command file after we have found at least
* one cost entry for the supplier that we are looking for.
*      This file now looks for either the first unpaid bill for the supplier
* (if only the name was specified) or looks for a complete match (if more than
* the name was specified.
*      If an an unpaid bill meeting the criteria is found, Looking is
* set to False.  Other wise it remains True.
*      If only the name was used, at this point we are at the first
* unpaid bill for the supplier name.
*      If more than the name was specified for the search, we could be anywhere
* in the indexed list of records for this supplier.  If we do not want to pay
* this particular bill, or we want to pay more bills for this supplier, we use
* a short cut in the PAYBILLS command file so that we do not have to start at
* the first record for the name every time.  To do this, we store the record
* number that we start at to a variable called Start if we have more than the
* name to look for.  Otherwise, Start =0
*****************************************************************************

STORE T TO Looking
IF MBill:Nmbr > ' ' .OR. MAmount > 0
   * If we have more than the name, we first check for the bill number.
   * If this is not found or if the bill has already been paid,
   * the confirming procedure is skipped (Looking set TRUE).
   *      In this case, we may have entered the list of supplier bills in
   * middle of the indexed list.  In a later procedure, we may need to go
   * back to the top and look at the names we skipped.  To do this, if we
   * find a record here, we store its number to "Start".

   IF MBill:Nmbr > '
      DO WHILE Name=Key .AND. .NOT. EOF .AND. Looking
         IF Bill:Nmbr <> MBill:Nmbr
            SKIP
         ELSE
            STORE F TO Looking
         ENDIF
      ENDDO

      * If we're on a new name or the end of the file, Looking is TRUE
      * because we have not found the supplier we were looking for.
      * Otherwise, we have a matching bill number to confirm.
      IF Looking
         ? '   This BILL NUMBER is not in the costbase.'
         ? '<Return> to continue.'
         WAIT
      ELSE
         IF Check:Nmbr <> ' '
            STORE T TO Looking
            ? '   This bill paid on '+Check:Date+', check '+Check:Nmbr
            ? '<Return> to continue.'
            WAIT
         ENDIF
      ENDIF
   ELSE

      * If no bill number, look for the amount and an unpaid bill.
      * If not found, skip the confirmation procedure.
      DO WHILE Name=Key .AND. .NOT. EOF .AND. Looking
         IF Amount <> MAmount .OR. Check:Nmbr <> ' '
            SKIP
         ELSE
            STORE F TO Looking
         ENDIF
      ENDDO
```

```
      * If we're on a new name or the end of the file, Looking is TRUE
      * Otherwise, we have an unpaid bill to confirm.
      IF Looking
         ? '   No unpaid bill for this amount and this supplier.'
         ? '<Return> to continue.'
         WAIT
      ENDIF
   ENDIF

   * If we found a matching record, store its number to Start
   IF .NOT. Looking
      STORE # TO Start
   ENDIF

ELSE
   * If we have only the name, find the next unpaid bill
   DO WHILE Name=Key .AND. .NOT. EOF .AND. Looking
      IF Check:Nmbr <> ' '
         SKIP
      ELSE
         STORE F TO Looking
      ENDIF
   ENDDO

   * If we're on a new name or the end of the file, Looking is TRUE
   * because we did not find the supplier we were looking for.
   * Otherwise, we have an unpaid bill to confirm.
   IF Looking
      ? '   There are no unpaid bills for this supplier.'
      ? '<Return> to continue.'
      WAIT
   ENDIF
ENDIF

RETURN
```

```
********************  PAYEMPS COMMAND FILE **********************
*      Does normal payroll processing or exceptions.
*****************************************************************

SET TALK OFF
STORE T TO Salaries
DO WHILE Salaries
   ERASE
   @  3,20 SAY '   PAYROLL FUNCTIONS          '
   @  6,20 SAY '   1> NORMAL PAYROLL          '
   @  7,20 SAY '   2> PARTIAL PAYMENT(S)      '
   @  8,20 SAY '   3> SKIP EMPLOYEE(S)        '
   @ 10,20 SAY '        <RETURN>'
   WAIT TO, Action

   IF Action = '1'
      DO Payroll
   ELSE
     IF Action = '2'
        ERASE
        ?
        ?
        ?
        ? 'This procedure allows you to pay less than a full salary if'
        ? 'for some reason an employee skipped days of work that are '
        ? 'not to be paid for.  Do you want to continue (Y or N)?'
        WAIT TO Continue
        IF !(Continue) = 'Y'
           RESTORE FROM B:Constant
           USE B:Personne
           ? 'Select the employee number for partial payment:'
           ? '       (Type 0 to quit.)'
           ?
           ?'NO.    NAME                    % OF PAY'
           LIST Name, Ratio*100 FOR .NOT. *
           ?
           INPUT 'Which number(0 to quit)? ' TO Wipe
           STORE INT(Wipe) TO Wipe
           DO WHILE Wipe <> 0
              GO Wipe
              ? 'How many days were worked'
              ? 'since the last regular payday?'
              ? 'Use decimals if needed (1 hour = 0.1333.)'
              ?
              INPUT TO Worked
              STORE Worked/11.0000 TO NewRatio
              REPLACE Ratio WITH NewRatio
              ?
              DISP Name, Ratio*100
              ?
              INPUT 'Next (0 to quit)? ' TO Wipe
              STORE INT(Wipe) TO Wipe
           ENDDO
        ENDIF
        RELEASE All

        ?
        ? 'Do you want to SKIP any employees (Y or N)?'
        WAIT TO Skip
        IF !(Skip) <> 'Y'
           DO Payroll
        ENDIF
        RELEASE Skip
     ELSE
```

```
      IF Action = '3'
         ERASE
         ?
         ?
         ?
         ? 'This procedure allows you to skip a paycheck in the payroll'
         ? 'procedure.  Do you want to continue (Y or N)?'
         WAIT TO Continue
         IF !(Continue) = 'Y'
            RESTORE FROM B:Constant
            USE B:Personne
            ? 'Select the number of the employee to skip:'
            ? '       (Type 0 to quit.)'
            ?'NO.    NAME                SKIP'
            ?
            LIST Name, Paid FOR .NOT. *
            ?
            INPUT 'Which number (0 to quit)? ' TO Wipe
            STORE INT(Wipe) TO Wipe
            DO WHILE Wipe <> 0
               GO Wipe
               REPLACE Paid WITH T
               ?
               ?'NO.    NAME                SKIP'
               ?
               DISP Name, Paid
               ?
               INPUT 'Next? ("0"to quit) ' TO Wipe
               STORE INT(Wipe) TO Wipe
            ENDDO
         ENDIF
         RELEASE All

         ?
         ? 'Do you want to pay a partial salary'
         ? 'to any employees (Y or N)?'
         WAIT TO Part
         IF !(Part) <> 'Y'
            DO Payroll
         ENDIF
         RELEASE Part
      ELSE
         IF Action = '4'
            ? 'Something 4'
            WAIT
         ELSE
            RELEASE All
            RETURN
         ENDIF 4
      ENDIF 3
   ENDIF 2
ENDIF 1
STORE T TO Salaries
ENDDO Salaries
```

```
  ************** PAYROLL COMMAND FILE ******************
* This command file generates payroll check stubs showing all deductions; gets
* the next check number and writes a check in the CheckFile, showing the new
* balance; and stores the salaries and deductions in a database called Hold81.
* This file is used to store the monthly, quarterly and annual FIT, FICA, SDI
* and SIT deductions.  The deductions are not picked up from tax tables because
* there are so few employees.  Instead, they are obtained from the individual
* employee records in the Personnel database.
*      Constants.MEM keeps track of the FICA and SDI percentages and their
* maximums, as well as the the constant for ThisYear.  Changes can be thus
* made in a single spot and will be correct in all the programs in the
* accounting system.
* The file is quite long, but breaks down into simpler modules:
*     I: Get the date and End of Month, Quarter and Year flags.
*    II: Compute all deductions and net pay for an individual employee, then
*        place this in the employee record in Personne.DBF
*   III: Operator verifies deductions and payroll stub is printed.
*    IV: Paycheck is written to the Checkfil and all amounts are placed into
*        the Hold81 summary file.
*     V: When all individuals have been paid, the Hold81 summary file is
*        updated if it is the end of month, quarter or year.
*    VI: Print out the summary file and data so that the physical checkbook
*        can be updated (computer does not print our checks).
*   VII: Delete transient constants, save others back to Constant.MEM for
*        system use.
*******************************************************************************

*****************************************************
*********  I: Get date and pay period flags *********

RESTORE FROM B:Constant
DO GetDate

STORE 'Y' TO GetWhen
DO WHILE !(GetWhen) = "Y"
   ERASE
   @ 1,18 SAY "PAYROLL PROCESSING"
   STORE " " TO EOY
   @ 4,8  SAY 'Want to change the date?' GET Date
   @ 5,8  SAY '(Press <Return> if okay.)'
   READ
   @ 7, 6 SAY "Is it the end of the YEAR?" GET EOY
   @ 7,35 SAY "(Y or N)"
   ? CHR(7)
   READ
   STORE !(EOY) TO EOY
   IF EOY = "Y"
      STORE "Y" TO EOQ
      STORE "Y" TO EOM
   ELSE
      STORE "N" TO EOY
      STORE " " TO EOQ
      @ 8, 3 SAY "Is it the end of the QUARTER?" GET EOQ
      @ 8,35 SAY "(Y or N)"
      ? CHR(7)
      READ
      STORE !(EOQ) TO EOQ
      IF EOQ = "Y"
         STORE "Y" TO EOM
      ELSE
         STORE "N" TO EOQ
         STORE " " TO EOM
         @ 9, 5 SAY "Is it the end of the MONTH?" GET EOM
         @ 9,35 SAY "(Y or N)"
```

```
         ? CHR(7)
         READ
         STORE !(EOM) TO EOM
         IF EOM <> "Y"
            STORE "N" TO EOM
         ENDIF
      ENDIF quarter
   ENDIF year

   ERASE
   @ 4,25 SAY $(Date,1,2)+'/'+$(Date,3,2)+'/'+$(Date,5,2)
   @ 6,0 SAY "End of YEAR: "+EOY+"     End of QUARTER: "+EOQ+;
             "     End of MONTH: "+EOM
   STORE " " TO GetWhen
   ?
   ? '
   @ 8,6 say 'The above information MUST be correct.
   ? CHR(7)
   * 2nd chance at date and flags
   ACCEPT '              Any CHANGES (Y or N)?' TO GetWhen

   STORE 'B:Hold'+STR(ThisYear,2) TO Header
   * Computer now does a date and flag check
   IF !(GetWhen) <> 'Y'
      IF $(Date,5,2)<'26' .AND. EOM = 'Y'
         ?
         ?
         ? "CHECK THE INFO AGAIN.  It's the end of the month, but the"
         ? 'date is '+Date-'.  Do you want to make changes (Y or N)?'
         ? CHR(7)
         WAIT TO GetWhen
      ENDIF
      IF EOY ='Y'
         SELECT SECONDARY
         USE &Header
         GO BOTTOM
         IF  Marker = 'Y'
            ? CHR(7)
            ? 'You blew it--the end of the year has been done!'
            WAIT
            RELEASE All
            STORE T TO Paying
            RETURN
         ENDIF
      ENDIF
   ENDIF
ENDDO GetWhen
RELEASE GetWhen


************************************************************************************
*********** II: Calculate deductions and net pay for each individual ********
* Compute deductions.  Deductions for FICA, FIT, SDI and SIT are  kept in the
* individual employee's Personnel record, rather than getting them from tax
* tables, because there are so few employees.  (You have to decide what should
* and should not be computerized.)  The "YTDxxx" variables are the year-to-date
* totals for these items.  Limits and percentages for FICA and SDI are obtained
* from a file called Constant.MEM.  These are the variables FICACut, FICAMax,
* FICAEnd, SDICut, SDIMax and SDIEnd.

SELECT PRIMARY

USE B:Personne
REPLACE All FICA WITH (Pay:Rate*FICACUT+0.005);
```

```
            SDI WITH (Pay:Rate*SDICUT+0.005)

STORE 0 TO Count
GO TOP
DO WHILE .NOT. EOF
   IF Paid .OR. *
      SKIP
   ELSE
      STORE Count + 1 TO Count

      *** Save the employee record in case the procedure is ended ***
      STORE STR(#,5) TO Payee
      COPY Record &Payee TO Bak

      *** Deductions for partial salary based on number of days worked ***
      *** Ratio is computed in PayMenu.CMD
      IF Ratio < 1.0000
         REPLACE Pay:Rate WITH Pay:Rate*Ratio, FICA WITH FICA*Ratio, FIT;
                 WITH FIT*Ratio, SDI WITH SDI*Ratio, SIT WITH SIT*Ratio
      ENDIF

      * Deductions and totals are computed then stored in the employee record
      * FedTemp, Statemp and EmpTemp are used to carry forward values for
      * salaries subject to FICA, SDI and state uenemployment insurance to
      * Hold81, the summary file.

      IF YTDSAL > FICAEnd
         STORE 0 TO FedTemp
         REPLACE FICA WITH 0
      ELSE
         IF (YTDSal + Pay:Rate) <= FICAEnd
            REPLACE YTDFICA WITH (YTDFICA + FICA)
            STORE Pay:Rate TO FedTemp
         ELSE
            REPLACE FICA WITH (MAXFICA - YTDFICA), YTDFICA WITH MAXFICA
            STORE (FICAEnd - YTDSal) TO FedTemp
         ENDIF
      ENDIF

      IF YTDSal > SDIEnd
         STORE 0 TO StaTemp
         REPLACE SDI WITH 0
      ELSE
         IF (YTDSAL + Pay:Rate) <= SDIEnd
            REPLACE YTDSDI WITH (YTDSDI + SDI)
            STORE Pay:Rate TO StaTemp
         ELSE
            REPLACE SDI WITH (MAXSDI - YTDSDI), YTDSDI WITH MAXSDI
            STORE (SDIEnd-YTDSal) TO StaTemp
         ENDIF
      ENDIF

      * In California, the employer pays an Unemployment Insurance contribution
      * on employee salary up to the amount of UIEnd.  There is nothing
      * deducted from the employee salary for this, so we keep track only of
      * the employer obligation as UISal.

      IF YTDSal > UIEnd
         STORE 0 TO EmpTemp
      ELSE
         IF (YTDSal + Pay:Rate) <= UIEnd
            STORE Pay:Rate TO EmpTemp
         ELSE
            STORE (UIEnd - YTDSal) TO EmpTemp
```

```
   ENDIF
ENDIF

REPLACE Net:Pay WITH (Pay:Rate-FICA-FIT-SDI-SIT)
REPLACE YTDFIT WITH (YTDFIT + FIT)
REPLACE YTDSIT WITH (YTDSIT + SIT)
REPLACE QTDSal WITH (QTDSal + Pay:Rate)
REPLACE YTDSal WITH (YTDSal + Pay:Rate)


*************************************************************************
************** III: Print employee stub *************
ERASE
SET PRINT ON
? '          '+$(Date,3,2)+'/'+$(Date,5,2)+'/'+$(Date,1,2)+': '+Name;
     + '   '+$(SS:Nmbr,1,3)+'-'+$(SS:Nmbr,4,2)+'-'+$(SS:Nmbr,6,4)
? '           GROSS PAY: $'-STR(Pay:Rate,7,2)+'     NET PAY: $';
                         -STR(Net:Pay,7,2)
?
? '                            FICA         FIT         SDI         SIT'
? '          THIS CHECK:    '+STR(FICA,6,2)+'     '+STR(FIT,7,2);
                    +'       '+STR(SDI,5,2)+'    ' +STR(SIT,7,2)
? '           THIS YEAR:   '+STR(YTDFICA,7,2)+'    '+STR(YTDFIT,8,2);
                    +'     '+STR(YTDSDI,6,2) +'    '+STR(YTDSIT,7,2)
? '                         TOTAL SALARY THIS QUARTER: $'-STR(QTDSal,9,2)
? '                            TOTAL SALARY THIS YEAR: $'-STR(YTDSal,9,2)
?
?
?
* Pagefeed after every six employee stubs
IF Count >= 6
   ? CHR(12)
   STORE 0 TO Count
ENDIF
SET PRINT OFF

IF EOQ = 'Y' .AND. Paid
   REPLACE QTDSal WITH 0
ENDIF

*************************************************************
******* IV: Record paycheck in Checkfil and Hold81 **********

* Now a check is "written" in the CheckFil.
SELECT SECONDARY
USE B:Checkfil
APPEND BLANK
REPLACE Check:Nmbr WITH NextCheck, Check:Date WITH Date,;
        Name WITH P.Name, Amount WITH Net:Pay, Emp:Nmbr;
        WITH P.Emp:Nmbr, Client WITH 'OFC', Job:Nmbr WITH 31,;
        Descrip WITH 'SALARY', Balance WITH (MBalance - Amount)
STORE (MBalance - Amount) TO MBalance
STORE STR(VAL(NextCheck)+1,4) TO NextCheck

ERASE
@ 3,25 SAY "** DO NOT INTERRUPT **"
@ 5,25 SAY "UPDATING MASTER RECORD"
? CHR(7)
* We keep an aggregate record of payroll and deductions.  The amounts
* for each employee are added to the amounts already in the last
* record in the file represented by "Header".  (This was set up at the
* start of the "GetWhen" loop earlier, and has the name "B:Hold81" or
* "B:Hold82" or whatever "ThisYear" is.)
*       This last record is either a blank (if this is the first
```

```
      * payroll of the month), or has data from previous salary payments
      * made during the current month.  At the end of the month, quarter and
      * year, totals and a new blank record (except at the end of the year)
      * are added.  This is done in the next loop.

      USE &Header

      * If this is a new year, there are no records in the file so we add a
      * blank record.  Otherwise, we go to the last record in the file.
      IF EOF
         APPEND BLANK
      ELSE
         GO BOTTOM
      ENDIF

      REPLACE Check:Date WITH Date, Payroll WITH (Payroll+Pay:Rate),;
              FICA WITH (FICA+P.FICA), FICASal WITH (FICASal + FedTemp),;
              FIT WITH (FIT + P.FIT), SDI WITH (SDI+P.SDI),;
              SDISal WITH (SDISal + Statemp), SIT WITH (SIT + P.SIT),;
              UISal WITH (UISal + EmpTemp)
      SELECT PRIMARY

      *** Reset the employee record if he was paid for part time.  ***
      *** The Bak file is not deleted here, as each copy command   ***
      *** above wipes out the previous contents.                   ***
      IF Ratio <> 1.0000
         REPLACE Ratio WITH 1.0000
         UPDA FROM Bak ON Emp:Nmbr REPL Pay:Rate,FICA,FIT,SDI,SIT,Net:Pay
      ENDIF
   ENDIF

   SKIP
ENDDO personnel file


*************************************************************************
******* V: Personnel records are reset and Holdxx is updated ********
STORE ' ' TO Completed
REPLACE All Paid WITH F

USE &Header
GO BOTTOM
IF EOM ='Y'
   REPLACE Marker WITH 'M'

   * If it's the end of the quarter, we total the amounts for the
   * previous three months to a new record and mark it with a 'Q'.
   IF EOQ = 'Y'
      STORE STR(#,5) TO Number
      TOTAL ON Marker TO Quarter FOR # >= (VAL(Number)-2)
      APPEND FROM Quarter
      DELETE FILE Quarter

      IF $(Date,3,2) = '03'
         REPLACE Check:Date WITH '1ST'
      ELSE
         IF $(Date,3,2) = '06'
            REPLACE Check:Date WITH '2ND'
         ELSE
            IF $(Date,3,2) = '09'
               REPLACE Check:Date WITH '3RD'
            ELSE
               IF $(Date,3,2) = '12'
                  REPLACE Check:Date WITH '4TH'
```

```
         ENDIF
        ENDIF
      ENDIF
    ENDIF

    REPLACE Marker WITH 'Q'

    * If it's the end of the year, we total all the quarterly amounts to
    * a new record and mark it with a 'Y'.
    IF EOY = 'Y'
       TOTAL ON Marker TO Annual FOR Marker = 'Q'
       APPEND FROM Annual
       REPLACE Marker WITH 'Y', Check:Date WITH 'END'
       DELETE FILE Annual
    ENDIF
  ENDIF

  * If it's the end of a month but not the end of the year, we add a new
  * blank record for next month's payroll records.
  IF EOY <> 'Y'
     APPEND BLANK
  ENDIF
ENDIF

*************************************************************************
******** VI: Print payroll summary, transfer checks to costbase *******
USE B:CheckFil
COUNT FOR .NOT. * TO Any
IF Any=0
   ? '        No new checks written.'
   ? '     <Return> to continue.'
   WAIT
ELSE
   USE &Header
   ERASE
   @ 12,25 SAY "CHECK THE PRINTER, THEN PRESS <RETURN>."
   ? CHR(7)
   WAIT
   ERASE
   SET PRINT ON
   SET MARGIN TO 45
   ? '              MASTER PAYROLL FILE SUMMARY: '+$(Date,3,2) +'/';
          +$(Date,5,2)+'/'+$(Date,1,2)

   ?
   ?
   ?'DATE   PAYROLL    FICA   FICASAL      FIT    SDI   SDISAL       '+;
             'SIT     UISal'

   ?
   LIST OFF
   SET MARGIN TO 38
   ? CHR(12)
   SET PRINT OFF

   ERASE
   @ 3,25 SAY "*** DO NOT INTERRUPT ***"
   @ 5,25 SAY " UPDATING THE COSTBASE"
   ? CHR(7)

   USE B:CostBase INDEX B:$Supp
   APPEND FROM B:Checkfil

   DO CheckStub
```

```
ENDIF

*************************************************************************6
****** VII: Dump transient variables, save necessary ones ********
RELEASE Payee,Number,Date,Ratio,Aborted,Printed,EOY,EOQ,EOM,Any,Header,;
        Count, FedTemp, StaTemp, EmpTemp, Marker, Paying, Salaries
SAVE TO B:Constant

USE
RELEASE All
DELETE FILE Bak
RETURN
```

```
      **********  DEPMENU COMMAND FILE  **********
* Select deposits or perform houskeeping on the checkbook.
*****************************************************

STORE T TO Incoming
DO WHILE Incoming
   ERASE
   @  5,20 SAY '    1> ENTER MONEY COMING IN    '
   @  7,20 SAY '    2> CHANGE OUR CHECK NUMBER  '
   @  9,20 SAY '    3> CHANGE CHECKBOOK BALANCE '
   ?
   ?
   ? '                                  <RETURN>'
   WAIT TO Action

   IF Action = '1'
      DO Deposits
   ELSE
      IF Action = '2'
         RESTORE FROM B:Constant
         ERASE
         @ 5,0 SAY 'This is the next check number' GET NextCheck
         @ 6,0 SAY 'To leave it unchanged, use the <return>.'
         @ 7,0 SAY 'To change it, just type in the new number.'
         READ
         SAVE TO B:Constant
         RELEASE All
      ELSE
         IF Action = '3'
            RESTORE FROM B:Constant
            STORE 'Y' TO Change
            DO WHILE !(Change) = 'Y'
               ERASE
               @ 5,0 SAY'       The current balance is:' GET MBalance
               ? 'To leave it unchanged, use the <return>.'
               ? 'To change it, just type in the new value.'
               READ
               ?
               ?
               ? '      Want to change your mind (Y or N)?'
               WAIT TO Change
            ENDDO
            RELEASE Change
            SAVE TO B:Constant
            RELEASE All
         ELSE
            RELEASE All
            RETURN
         ENDIF 3
      ENDIF 2
   ENDIF 1
   ERASE
   STORE T TO Incoming
ENDDO Incoming
```

```
        ************  DEPOSITS COMMAND FILE  ***********
* This file records any money coming in in a file called Deposits.  If the
* the money is in payment of an invoice, the amount and date of payment are
* entered against that invoice in the Invoice file.
*       The checkbook balance is kept current for each entry.
*       At the end of the sesssion, deposits are printed out individually, then
* the total of deposits plus the new checkbook balance are printed.
*************************************************************************

RESTORE FROM B:Constant

ERASE
@  5,20 SAY '  ENTERING INCOME'
@  7, 5 SAY 'The STARTING BALANCE is '+STR(MBalance,9,2)
?
? '   If this does not match the checkbook,'
? '   <Return> to the main menu to change.'
?
? '   C to CONTINUE.'
?
WAIT TO Continue
IF !(Continue) <> 'C'
   RELEASE All
   RETURN
ENDIF
RELEASE Continue

DO GetDate

SELECT PRIMARY
USE B:Deposits
COPY STRUCTURE TO GetDep

USE GetDep
STORE 'Y' TO Depositing
DO WHILE !(Depositing) <>'F'
   APPEND BLANK
   STORE STR(#,5) TO Number
   REPLACE Dep:Date WITH Date

   ERASE
   * Next loop is used when there has been an error in the entry
   * (defined as no client or no rate).  The operator is shown the
   * previous entries and can make any changes required.
   STORE 'T' TO Incorrect
   DO WHILE !(Incorrect) <> 'F'
      @ 3, 0 SAY ' If a check covers more than one agency invoice,'
      @ 4, 0 SAY ' enter each invoice and amount separately.'
      ?
      @  6,0 SAY ' RECORD NUMBER: '+Number
      @  7,0 SAY '       HOW MUCH' GET Deposit
      @  8,0 SAY 'OUR INVOICE NO' GET Inv:Nmbr
      @  9,0 SAY '    CHECK FROM' GET Payer
      @ 10,0 SAY 'THEIR CHECK NO' GET Pay:Nmbr
      @ 11,0 SAY '       Comments' GET Comments
      ? CHR(7)
      READ

      REPLACE Payer WITH !(Payer), Comments WITH !(Comments)
      @  9,15 SAY Payer
      @ 11,15 SAY Comments

      IF Payer <> '  ' .AND. Deposit > 0
         @ 17,5 SAY '   C to CHANGE,'
```

```
      @ 18,5 SAY '<Return> to continue.'
      ?
      ? CHR(7)
      WAIT TO Depositing
      IF !(Depositing)='C'
         STORE 'T' TO Incorrect
         ERASE
      ELSE
         STORE (MBalance + Deposit) TO MBalance
         @ 17, 5 SAY '   F if FINISHED,'
         ?
         ? '
         ? CHR(7)
         WAIT TO Depositing

         STORE 'F' TO Incorrect
      ENDIF
   ELSE
      @ 15,5 SAY 'CHECK WRITER or AMOUNT missing.'
      ?
      ? '        F if FINISHED,'
      ? '     <Return> to correct the record.'
      ? CHR(7)
      WAIT TO Depositing
      ERASE
      IF !(Depositing)= 'F'
         DELETE RECORD &Number
         STORE 'F' TO Incorrect
      ELSE
         ERASE
         STORE 'T' TO Incorrect
      ENDIF
   ENDIF
 ENDDO Incorrect
ENDDO Depositing

RELEASE Change, Date, NoDate, Depositing, Number, Update, New, Incorrect
SAVE TO B:Constant

COUNT FOR .NOT. * TO Any
ERASE
IF Any = 0
   ? 'No deposits to add to the file.'
   ? 'Press any key to continue.'
   ? CHR(7)
   USE
   WAIT
ELSE
   DO DepPrint

   * The next portion of this program uses the Primary and Secondary work
   * areas to record payments received against agency invoices in the record
   * for that invoice in the Invoices file.  Both work areas are necessary so
   * that we can compare each record in the GetDep file against all of the
   * records in the Invoices file.
   DO DepTrans
   USE B:Deposits
   APPEND FROM GetDep
ENDIF there are deposits to add to the file

DELETE FILE GetDep
RELEASE All
RETURN
```

```
         **********  DEPPRINT COMMAND FILE  **********
* Prints valid deposits in the GetDep file as part of the Deposits file.
*************************************************************************

@  5,10 SAY 'To print the deposits you just entered,'
@  6,10 SAY 'press <Return>.'
? CHR(7)
WAIT
SET PRINT ON
? ' DATE      PAID BY                    AMOUNT    INV #    COMMENTS:'
?
GO TOP
STORE 0 TO Count
DO WHILE .NOT. EOF
   DISPLAY OFF Dep:Date, Payer, Deposit, Inv:Nmbr, Comments FOR .NOT. *
   SKIP
   STORE Count+1 TO Count
   IF Count=10
      STORE 0 TO Count
      WAIT
   ENDIF
ENDDO
SUM Deposit TO Temp
?
? ' The total deposit is ' + STR(Temp,9,2)
?
? ' The final balance is ' + STR(Mbalance,9,2)
?
SET PRINT OFF

RELEASE Count, Temp
RETURN
```

```
          ********** DEPTRANS COMMAND FILE **********
* Applies deposits from the GetDep file against the matching invoices in the
* Invoices file as payments are received against them.
*******************************************************************************

GO TOP
ERASE
DO WHILE .NOT. EOF
   STORE STR(#,5) TO Number
   @  6,20 SAY 'RECORD NUMBER  '+Number
   ? CHR(7) + CHR(27) + CHR(74)

   IF Inv:Nmbr=' ' .OR. *
      SKIP
   ELSE
      @  7,20 SAY 'INVOICE NUMBER '+Inv:Nmbr
      STORE Inv:Nmbr TO Key

      SELECT SECONDARY
      USE B:Invoices INDEX B:Invoices
      FIND &Key
      STORE T TO Again
      STORE 'T' TO Decision
      IF # = 0
         DO WHILE Again
            @  9,15 SAY 'THIS INVOICE NUMBER IS NOT IN THE INVOICE FILE.
            @ 11,15 SAY '      E to EDIT it.
            @ 12,15 SAY '      C to CONTINUE.
            ?
            ? CHR(7)
            WAIT TO Decision

            IF !(Decision) = 'E'
               SELECT PRIMARY
               EDIT &Number
               SELECT SECONDARY
               STORE F TO Again
            ELSE
               IF !(Decision) = 'C'
                  STORE F TO Again
               ELSE
                  STORE T TO Again
               ENDIF C
            ENDIF E
         ENDDO
      ELSE
         REPLACE Amt:Rcd WITH (Amt:Rcd + Deposit), Date:Rcd WITH Dep:Date
      ENDIF 0
      SELECT PRIMARY

      * We do not skip to the next record if the record was edited.  This
      * allows us to run the edited record through the process again.
      IF !(Decision) <> 'E'
         SKIP
      ENDIF
   ENDIF no invoice number or record deleted
ENDDO the transfer
```

```
    ********** IOMENU COMMAND FILE **********
* Selects the appropriate action to be taken with insertion orders
* (instructions from our ad agency to magazine publishers).
*****************************************************************

STORE T TO Inserting
DO WHILE Inserting
   ERASE
   @  7,20 SAY '   1> ENTER INSERTION ORDERS'
   @  9,20 SAY '   2> EDIT INSERTION ORDERS'
   @ 11,20 SAY '   3> REVIEW/PRINT INSERTION ORDERS'
   @ 12,20 SAY '      BY CLIENT & MAGAZINE'
   @ 14,20 SAY '        <RETURN>'
   WAIT TO Action

   IF Action = '1'
      DO IOPost
   ELSE
      IF Action = '2'
         STORE "Y" TO Changing
         DO WHILE !(Changing)='Y'
            USE B:Inserts
            IF EOF
               ? 'There are no entries in the INSERTION ORDER file.'
               STORE "N" TO Changing
            ELSE
               STORE IO:Nmbr TO First
               GO BOTTOM
               STORE IO:Nmbr TO Last
               ERASE
               @  3,15 SAY 'EDITING INSERTION ORDERS '+First+'thru '+Last
               @  5,15 SAY '^W to SAVE, ^Q to CANCEL changes you make.'
               @  6,15 SAY '^R for PREVIOUS, ^C for NEXT record if MORE = T'
               ?
               ?
               ACCEPT 'Which ORDER NUMBER do you want to EDIT?' TO Order
               USE B:Inserts INDEX B:Inserts
               FIND &Order
               IF #=0
                  ?
                  ?
                  ? 'That insertion order is not in the file.'
                  ? 'Do you want to continue (Y or N)?'
                  WAIT TO Changing
               ELSE
                  STORE STR(#,5) TO Number
                  Edit &Number
                  REPLACE Client WITH !(Client), Ad WITH !(Ad),Magazine WITH;
                          !(Magazine)
                  ?
                  ? 'Do you want to edit any other insertion orders (Y or N)?'
                  WAIT TO Changing
               ENDIF
            ENDIF
         ENDDO Changing
         RELEASE All
      ELSE
         IF Action ='3'
            DO IOReview
         ELSE
            RELEASE All
            RETURN
         ENDIF 3
      ENDIF 2
```

```
   ENDIF 1
   STORE T TO Inserting
ENDDO Inserting
```

```
        ********  IOPOST COMMAND FILE  **********
* Gets information for insertion orders (instructions to magazine
* publishers from our ad agency).  Works much like Postbills and
* Posttime.
*****************************************************************

RESTORE FROM B:Constant

DO GetDate

USE B:Inserts
COPY STRUCTURE TO GetInserts
USE GetInserts

STORE ' ' TO New
STORE 'Y' TO Inserting
DO WHILE !(Inserting) <>'F'
   APPEND BLANK
   STORE STR(#,5) TO Number
   REPLACE IO:Date WITH Date, IO:Nmbr WITH Next:IO

   ERASE
   * Next loop is used when there has been an error in the entry
   * (defined as no client or no rate).
   STORE 'T' TO Incorrect
   DO WHILE !(Incorrect) <> 'F'
      ERASE
      @ 4,0 SAY ' INSERTION ORDER: '+IO:Nmbr
      @ 4,30 SAY '              DATE:'+Date
      ?
      @  6,0 SAY '  RECORD NUMBER: '-Number
      IF !(New) = 'S'
         @  7,0 SAY '       OUR CLIENT :' + MClient
      ELSE
         @  7,0 SAY '       OUR CLIENT ' GET MClient
         STORE !(MClient) TO MClient
      ENDIF
      @  8,0 SAY '       JOB NUMBER ' GET Job:Nmbr
      @  9,0 SAY '  AD DESCRIPTION ' GET Ad
      @ 10,0 SAY '  HOW MUCH SPACE ' GET Space
      @ 11,0 SAY '  WHICH MAGAZINE ' GET Magazine
      @ 12,0 SAY '      WHICH ISSUE ' GET Issue
      @ 13,0 SAY 'GROSS SPACE COST ' GET Gross:Cost
      @ 14,0 SAY '   DISCOUNT RATE ' GET Times
      READ

      REPLACE Net:Cost WITH Gross:Cost*0.8500, Client WITH MClient,;
              Ad WITH !(Ad), Magazine WITH !(Magazine), Issue WITH !(Issue)
      @  7,18 SAY Client
      @  9,18 SAY Ad
      @ 11,18 SAY Magazine
      @ 12,18 SAY Issue
      @ 15,0 SAY '  NET SPACE COST ' GET Net:Cost

      IF Client <> '  ' .AND. Gross:Cost > 0 .AND. Job:Nmbr > 99
         @ 18,5 SAY '   C to CHANGE,'
         @ 19,5 SAY '<Return> to continue.'
         ?
         WAIT TO New
         IF !(New)='C'
            STORE 'T' TO Incorrect
         ELSE
            @ 17, 5 SAY ' -F to FINISHED,'
            @ 18, 5 SAY '   S for SAME insertion order,'
```

```
            @ 19, 5 SAY '<Return> for NEXT insertion order.'
            @ 21, 0 SAY '                '
            ACCEPT TO New

            IF !(New) <> 'S'
               IF VAL(Next:IO) < 9999
                  STORE STR(VAL(Next:IO)+1,4) TO Next:IO
               ELSE
                  STORE '1001' TO Next:IO
               ENDIF
            ENDIF
            STORE 'F' TO Incorrect
         ENDIF
         STORE New TO Inserting
      ELSE
         ?
         ?
         ?
         ?
         ?
         ?
         ? '   CLIENT, JOB or RATE missing.'
         ?
         ? '        F if FINISHED,'
         ? '     <Return> to correct the record.'
         ?
         WAIT TO Inserting
         IF !(Inserting)= 'F'
            DELETE RECORD &Number
            STORE 'F' TO Incorrect
         ELSE
            STORE 'T' TO Incorrect
         ENDIF
      ENDIF
   ENDDO Incorrect
ENDDO Inserting

RELEASE Date, NoDate, Inserting, Number, Update, New, Incorrect
SAVE TO B:Constant

COUNT FOR .NOT. * TO Any
ERASE
IF Any = 0
   ? 'No insertions to add to the file.'
   ? 'Press any key to continue.'
   USE
   WAIT
ELSE
   @  5,10 SAY 'To print the insertions you just entered,'
   @  6,10 SAY 'press <Return>.'
   WAIT TO Number
   *"Number" determines the starting record number for the printou

   SET PRINT ON
   ? 'IO # MAGAZINE        ISSUE   JOB   AD              ';
             +'SPACE            GROSS      NET  X  DATE'
   ?

   * "Output" and"Condition" needed in the Printout Command file
   STORE 'Y' TO Output
   STORE 'OFF' TO Condition
   DO Printout

   ERASE
```

```
   @ 5,20 SAY 'UPDATING THE INSERTION ORDER FILE'
   USE B:Inserts INDEX B:Inserts
   APPEND FROM GetInserts
ENDIF

DELETE FILE GetInserts
RELEASE All
RETURN
```

```
      **********  IOREVIEW COMMAND FILE  **********
* Provides insertion order displays and printout.
*      The operator can select all the insertions for the client,
* or can select only those for a particular magazine.
*************************************************************************

SET TALK OFF
USE B:Inserts

STORE ' ' TO Again
DO WHILE !(Again) <> 'F'
   STORE '    ' TO MClient
   STORE '                    ' TO MMagazine
   STORE ' ' TO Hardcopy
   STORE ' ' TO Other
   ERASE
   @  2,11 SAY '     MEDIA SUMMARY:'
   @  4,11 SAY 'ENTER CLIENT CODE' GET MClient
   @  5,11 SAY '   MAGAZINE NAME?' GET MMagazine
   @  6,11 SAY '         P to PRINT' GET Hardcopy
   READ
   IF MClient = ' '
      @  9, 0 SAY ' '
      ? '      CLIENT missing.'
      ? '      F if Finished,'
      ? '  <Return> to continue.'
      WAIT TO Again
   ELSE
      STORE !(MClient) TO MClient
      STORE !(MMagazine) TO MMagazine
      STORE !(Hardcopy) TO Hardcopy
      @ 4,29 SAY MClient
      @ 5,29 SAY MMagazine
      @ 6,29 SAY Hardcopy
      @ 9, 0 SAY ' '
      ?
      ?
      ACCEPT 'Type C to CHANGE any entries' TO Changes
      IF !(Changes) = 'C'
         STORE ' ' TO Again
         ERASE
      ELSE
         IF MMagazine >'
            STORE TRIM(MMagazine) TO MMagazine
            STORE '.AND. Magazine=MMagazine' TO Condition
         ELSE
            STORE CHR(0) TO Condition
         ENDIF

         IF !(Hardcopy) = 'P'
            STORE 'TO PRINT' TO Hardcopy
         ELSE
            STORE CHR(0) TO Hardcopy
         ENDIF Hardcopy

         SET HEADING TO MEDIA SUMMARY FOR &MClient &MMagazine
         REPORT FORM Media &Hardcopy FOR Client=MClient &Condition
         ?
         ? '      F if Finished,'
         ? '  <Return> to continue.'
         WAIT TO Again
         ERASE
      ENDIF okay to do the report
   ENDIF
```

```
ENDDO Again

ERASE
RELEASE All
RETURN
```

```
        ************** INVMENU COMMAND FILE ******************
* Functions are selected by the menu.  This procedure works with two data
* files, BILLINGS and INVOICES.  BILLINGS keeps track of the amount
* billed to a client by individual job number, while INVOICES is a
* summary of the total billed on any given invoice.  This latter file can
* be used to set up an accounts receivable system, as it has fields for
* storing how much has been received in payment against an invoice and
* when that amount was received (filled in by the Deposits.CMD file).
*************************************************************************

ERASE
STORE T TO Invoicing
DO WHILE Invoicing
   @  5,20 SAY '   1> BILL CLIENTS BY JOB'
   @  7,20 SAY '   2> EDIT INVOICES and BILLINGS'
   @  9,20 SAY '   3> REVIEW/PRINT INVOICES and BILLINGS'
   @ 12,20 SAY '        <RETURN>'
   WAIT TO Action

   IF Action = '1'
      DO Invoices
   ELSE
      IF Action = '2'
         STORE 'Y'TO Changing
         DO WHILE !(Changing) = 'Y'
            ERASE
            ? '        J to edit individual job billings,'
            ? '     <Return> to edit the summary invoices.'
            WAIT TO Which
            IF !(Which) = 'J'
               STORE 'Billings' TO Database
            ELSE
               STORE 'Invoices' TO Database
            ENDIF

            USE B:&Database
            STORE Inv:Nmbr TO First
            GO BOTTOM
            STORE Inv:Nmbr TO Last
            ERASE
            @  3,10 SAY 'EDITING '+!(Database)
            @  3,35 SAY First+'thru '+Last
            @  5,10 SAY '^W to SAVE, ^Q to CANCEL changes you make.'
            @  6,10 SAY '^R for PREVIOUS, ^C for NEXT record.'
            @  8,10 SAY 'Which INVOICE NUMBER do you want to EDIT?'
            IF !(Which) = 'J'
               @  9,10 SAY 'This takes you to the FIRST ENTRY for that number.
               @ 10,10 SAY 'Use ^C to look at the rest of them.'
            ENDIF
            ACCEPT TO Invoice

            USE B:&Database INDEX B:&Database
            FIND &Invoice
            IF #=0
               ?
               ?
               ? 'That invoice number is not in the file.'
               ? 'Do you want to continue (Y or N)?'
               WAIT TO Changing
            ELSE
               STORE STR(#,5) TO Number
               Edit &Number
               REPLACE Sales:Tax WITH 0.06*Taxable
               REPLACE Client WITH !(Client)
```

```
            IF !(Which) = 'J'
               REPLACE Descrip WITH !(Descrip),PO:Nmbr WITH !(PO:Nmbr)
            ENDIF
            ?
            ? 'Do you want to edit any other invoices (Y or N)?'
            WAIT TO Changing
         ENDIF
      ENDDO Changing
      RELEASE All
   ELSE
      IF Action ='3'
         ERASE
         @ 4, 0 SAY ' '
         ? '        J to see individual job billings,'
         ? '     <Return> to see the summary invoices.'
         WAIT TO Which
         IF !(Which) = 'J'
            STORE 'Billings' TO Database
         ELSE
            STORE 'Invoices' TO Database
         ENDIF

         USE B:&Database
         STORE 'Y' TO Reviewing
         DO WHILE !(Reviewing)='Y'
            GO BOTTOM
            STORE STR(#,5) TO Last
            ERASE
            @ 5,10 SAY 'The '+!(Database)+' file has '-Last-' entries.'
            @ 7,10 SAY '<Return> to see the entire file, or'
            @ 8,10 SAY 'enter the record number to start on.'
            ACCEPT TO Number

            ? 'Do you want to print the file now (Y or N)?'
            WAIT TO Output

            IF !(Output)='Y'
               SET PRINT ON
            ENDIF

            STORE CHR(0) TO Condition
            DO Printout
            ?
            SET PRINT OFF
            ? 'Do you want to see it again (Y or N)?'
            WAIT TO Reviewing
            ERASE
         ENDDO Reviewing
         RELEASE All
      ELSE
         RELEASE All
         RETURN
      ENDIF 3
   ENDIF 2
ENDIF 1
ERASE
STORE T TO Invoicing
ENDDO Invoicing
```

```
          **********  INVOICES COMMAND FILE  **********
* This file accepts inputs for invoices to clients.  Individual projects
* and items are stored in the Billings data file.  Any number of items
* may be entered using a single invoice number.  Invoice numbers are
* automatically generated by the computer and stored in the Constant.Mem
* file.
*       After all the job billings have been entered, they are summarized b
* invoice number and the data is stored in the Invoices file.
*       A printout of items billed and invoice totals is provided.
*************************************************************************

RESTORE FROM B:Constant

DO GetDate

USE B:Billings
COPY STRUCTURE TO GetCosts

USE GetCosts
STORE ' ' TO Billing
DO WHILE !(Billing) <>'F'
   APPEND BLANK
   STORE STR(#,5) TO Number
   REPLACE Inv:Date WITH Date. Inv:Nmbr WITH Next:Inv

   ERASE
   STORE 'T' TO Entering
   DO WHILE !(Entering) <> 'F'
      ERASE
      @  3, 0 SAY 'INVOICE NUMBER  '+Next:Inv
      @  3,30 SAY '     DATE  '+Inv:Date
      @  5,0 SAY ' RECORD NUMBER: '-Number
      IF !(Billing) = 'S'
         @  7,0 SAY '          CLIENT:'+ MClient
         REPLACE Client WITH MClient
      ELSE
         @  7,0 SAY '          CLIENT ' GET Client
      ENDIF

      @  8,0 SAY '    JOB NUMBER ' GET Job:Nmbr
      @  9,0 SAY 'TAXABLE AMOUNT ' GET Taxable
      @ 10,0 SAY 'TAXFREE AMOUNT ' GET TaxFree
      @ 11,0 SAY '  P. O. NUMBER ' GET PO:Nmbr
      @ 12,0 SAY '   DESCRIPTION ' GET Descrip
      READ

      STORE !(Client) TO MClient
      REPLACE Client WITH MClient,Descrip WITH !(Descrip),;
              PO:Nmbr WITH !(PO:Nmbr)
      @  7,16 SAY Client
      @ 11,16 SAY PO:Nmbr
      @ 12,16 SAY Descrip
      IF Taxable > 0
         REPLACE Sales:Tax WITH 0.06*Taxable
         @ 13,0 SAY '     SALES TAX'GET Sales:Tax
      ENDIF

      IF Job:Nmbr < 100
         @ 16,0 SAY '          JOB not 3 digits.'
      ENDIF

      IF MClient <> '  ' .AND. (Taxable > 0 .OR. TaxFree > 0)
         @ 17,0 SAY '       C to CHANGE this entry.'
         ? '     <Return> to continue.'
```

```
         WAIT TO New
         IF !(New)='C'
            STORE 'T' TO Entering
         ELSE
            @ 16, 0 SAY '         F if FINISHED,             '
            @ 17, 0 SAY '         S for SAME invoice number,'
            @ 18, 0 SAY '      <Return> for NEXT invoice number.'
            @ 19, 0 SAY '
            ACCEPT TO New

            IF !(New) <> 'S'
               STORE STR(VAL(Next:Inv)+3,5) TO Next:Inv
            ENDIF
            STORE 'F' TO Entering
         ENDIF
         STORE New TO Billing
      ELSE
         @ 17,0 SAY '         CLIENT or AMOUNT missing.'
         ?
         ? '         F if FINISHED,'
         ? '      <Return> to correct the record.'
         WAIT TO Billing
         IF !(Billing)= 'F'
            DELETE RECORD &Number
            STORE 'F' TO Entering
         ELSE
            STORE 'T' TO Entering
         ENDIF
      ENDIF
   ENDDO Entering
ENDDO Billing

RELEASE Billing, Entering, MClient, Task, Number, Date, New
SAVE TO B:Constant

PACK

GO TOP
ERASE
IF EOF
   ? 'No invoices to add to the file.'
   ? 'Press any key to continue.'
   WAIT
ELSE
   @ 5,20 SAY '****   DO NOT INTERRUPT   ****'
   @ 7,20 SAY 'UPDATING BILLINGS AND INVOICES'

   * Costs entered are totalled by invoice number to Scratch because several
   * job costs can be entered against each invoice number.  Amounts are adusted
   * for one client who always pays promptly and takes a 2% discount.  Each
   * invoice is totalled.  Temp has only summary data needed for a printout.

   USE B:Invoices
   COPY STRUCTURE TO Scratch

   USE GetCosts
   ERASE
   @  5,10 SAY 'When ready to print the billings you just added,'
   @  6,10 SAY 'press <Return>'
   TOTAL ON Inv:Nmbr TO Scratch FIELDS Taxable, Sales:Tax, TaxFree
   WAIT TO Number

   SET PRINT ON
   ? 'ENTRIES BY JOB NUMBER:'
```

```
   ?
   ? 'INV #  JOB    DATE  TAXABLE      TAX  TAXFREE   P.O.#  DESCRIPTION'
   ?

   * "Output" is needed in the Printout Command file
   STORE 'Y' TO Output
   STORE 'OFF' TO Condition
   DO Printout

   * One of our clients always pays promptly and takes a 2% discount.
   * We do this after the original entries were printed out:
   REPLACE Taxable WITH 0.980*Taxable, TaxFree WITH 0.980*TaxFree, Sales:Tax;
           WITH 0.980*Sales:Tax FOR Client = 'SPI'
   ?
   ? 'Updating the BILLINGS database now.'
   USE B:Billings INDEX B:Billings
   APPEND FROM GetCosts

   USE Scratch
   REPLACE All Amount WITH (Taxable + Sales:Tax + TaxFree)
   COPY TO Temp FIELDS Inv:Date, Inv:Nmbr, Taxable, Sales:Tax,;
                               TaxFree, Amount
   REPLACE Taxable WITH 0.980*Taxable, TaxFree WITH 0.980*TaxFree, Sales:Tax;
           WITH 0.980*Sales:Tax, Amount WITH 0.980*Amount FOR Client = 'SPI'

   USE Temp
   STORE 'Y' TO Output
   SET PRINT ON
   ?
   ?
   ? 'TOTALS BY INVOICE NUMBER:'
   ?
   ? 'DATE   INV#   TAXABLE      TAX   TAXFREE    TOTAL'
   ?

   DO Printout
   ?
   ? 'Updating the INVOICES database now.'
   USE B:Invoices INDEX B:Invoices
   APPEND FROM Scratch
ENDIF

USE
DELETE FILE Scratch
DELETE FILE Temp
DELETE FILE GetCosts
RELEASE All
RETURN
```

```
      **********  REPORTMENU COMMAND FILE  **********
* This command file is a sub-module of the ACCOUNTS.CMD control
* module.  It provides detailed choices that relate to reports
* that the user might choose to see or print from the cost
* database.  The functions are set up as sub-sub-procedures
* under the control of this module.
*************************************************************

ERASE
STORE T TO Reporting
DO WHILE Reporting
     @  3,20 SAY '   1> COSTS BY JOB'
     @  5,20 SAY '   2> FIND & EDIT BILLS'
     @  7,20 SAY '   3> REVIEW A DATABASE'
     @  9,20 SAY '   4> Quarterly Sales Tax Summary'
     @ 11,20 SAY '   5> RE-INDEX THE COSTBASE ON JOB NUMBERS'
     @ 12,20 SAY "      Make sure you won't need the computer"
     @ 13,20 SAY '      for a while: this takes a long time.'

     @ 17,20 SAY '               <RETURN>'
     WAIT TO Action

     IF Action = '1'
        USE B:Postfile
        COUNT FOR .NOT. * TO Any
        IF Any > 0
           @ 15, 0 SAY CHR(27)+CHR(74)
           ? 'There are '+STR(Any,5)+' entries in the Postfile.'
           ? 'Do you still want to do the Job Costs (Y or N).'
           WAIT TO Continue
           IF !(Continue) = 'Y'
              DO JobCosts
           ENDIF
        ELSE
           DO JobCosts
        ENDIF
        RELEASE Any
     ELSE
        IF Action = '2'
           DO FindBills
        ELSE
           IF Action = '3'
              ERASE
              DISPLAY FILES ON B
              ?
              ?
              ? 'Which file do you want to review?'
              ACCEPT TO Database
              IF FILE("B:"+DATABASE) > 0
                 USE B:&Database
                 DO Review
              ELSE
                 * Erases IBM 3101 to end of screen
                 @ 17,0 SAY CHR(27)+CHR(74)
                 @ 17,0 SAY !(Database) + " isn't on the list, is it?  Check ";
                      + 'your spelling, then hit <Return>'
                 ? 'and try again.  Or not, as the case may be.'
                 WAIT
              ENDIF
           ELSE
              IF Action = '4'
                 DO SalesTax
              ELSE
                 IF Action = '5'
```

```
                    DO JobsIndx
                 ELSE
                    RELEASE All
                    RETURN
                 ENDIF 5
              ENDIF 4
           ENDIF 3
        ENDIF 2
     ENDIF 1
  ERASE
  STORE T TO Reporting
ENDDO Reporting
```

```
      **********  JOBCOSTS COMMAND FILE  **********
* Provides summaries of costs by client and job number.  This can
* also be used to summarize all office categories, since they fall
* into these fields.
*      REPORTS ARE BY JOB NUMBER.  Client code is used only in the heading.
* The report is actually prepared based on the job number, so accuracy is
* critical.
*      This file works with a partially indexed costbase, so "Unindexed" is
* used to keep track of how many records are not in the index.  If this gets
* beyond a specific number, the operator is prompted to reindex the Costbase.
*************************************************************************

SET TALK OFF

RESTORE FROM B:Constant
DO GetDate

STORE 0 TO Unindexed
STORE ' ' TO Again
DO WHILE !(Again) <> 'F'
   STORE '    ' TO MClient
   STORE '    ' TO MJob:Nmbr
   STORE ' ' TO Hardcopy
   STORE 'N' TO Number
   ERASE
   @  2,11 SAY ' JOB COST SUMMARY :'
   @  4,11 SAY 'ENTER CLIENT CODE ' GET MClient
   @  5,11 SAY ' ENTER JOB NUMBER ' GET MJob:Nmbr
   @  6,11 SAY '       P to PRINT ' GET Hardcopy
   @  7,11 SAY 'SHOW BILL NUMBERS ' GET Number
   READ
   ?
   IF MClient = ' ' .OR. MJob:Nmbr= '  '
      @  9, 0
      ? '      CLIENT or JOB NUMBER  missing.'
      ? '      F if Finished,'
      ? '  <Return> to continue.'
      WAIT TO Again
   ELSE
      @ 8,0 SAY CHR(27)+CHR(74)
      ACCEPT '    OPTIONAL JOB DESCRIPTION ' TO Message
      STORE TRIM(!(Message)) TO Message
      STORE !(MClient) TO MClient
      STORE !(Hardcopy) TO Hardcopy
      STORE !(Number) TO Number
      @ 4,30 SAY MClient
      @ 6,30 SAY Hardcopy
      @ 7,30 SAY Number
      @ 9,30 SAY Message
      ?
      ?
      ACCEPT 'Type C to CHANGE any entries' TO Changes
      IF !(Changes) = 'C'
         STORE ' ' TO Again
         ERASE
      ELSE
         ERASE
         IF !(Hardcopy) = 'P'
            STORE "TO PRINT" TO Hardcopy
            SET PRINT ON
         ENDIF Hardcopy

         IF Number = 'Y'
            STORE 'Bill #' TO Other
```

```
ELSE
   STORE CHR(0) TO Other
ENDIF

? $(Date,3,2)+'/'+$(Date,5,2)+'/'+$(Date,1,2)+':  COST SUMMARY FOR ';
            +'&MClient-&MJob:Nmbr'
? '            ' + Message
?
? 'DATE    NAME                    DESCRIPTION                    AMOUNT';
   +'  &Other'
?
USE B:CostBase INDEX B:$Jobs
IF Number = 'Y'
   STORE ',Bill:Nmbr' TO Other
ELSE
   STORE CHR(0) TO Other
ENDIF

STORE 0 TO Sum
STORE 0 TO HowMany
STORE 0 TO LineCnt
STORE 0 TO Spacer
FIND &MJob:Nmbr
IF # <> 0
   DO WHILE Job:Nmbr = VAL(MJob:Nmbr) .AND. .NOT. EOF
      DISPLAY Next 1 Bill:Date,Name,Descrip+'    ',Amount &Other OFF
      STORE Sum + Amount TO Sum
      STORE LineCnt + 1 TO LineCnt
      STORE Spacer + 1 TO Spacer

      IF Spacer = 10
         ?
         STORE 0 TO Spacer
      ENDIF

      IF LineCnt = 50
         ? CHR(12)
         STORE 0 TO LineCnt
         STORE 0 TO Spacer
         ? 'DATE    NAME                    DESCRIPTION';
           +'                    AMOUNT'
         ?
      ENDIF
      SKIP
   ENDDO
ENDIF

GO TOP
STORE VAL(Name) TO LastReco
USE B:Costbase
STORE 0 TO Unindexed
GO LastReco
SKIP
DO WHILE .NOT. EOF
   DISPLAY Next 1 Bill:Date, Name, Descrip+'    ', Amount;
           FOR Job:Nmbr = VAL(MJob:Nmbr) OFF
   IF Job:Nmbr = &MJob:Nmbr
      STORE Sum + Amount TO Sum
      STORE LineCnt + 1 TO LineCnt
      STORE Spacer + 1 TO Spacer

      IF Spacer = 10
         ?
         STORE 0 TO Spacer
```

```
      ENDIF

      IF LineCnt = 50
         ? CHR(12)
         STORE 0 TO LineCnt
         STORE 0 TO Spacer
         ? 'DATE     NAME                    DESCRIPTION'
           +'                    AMOUNT'
         ?
      ENDIF
   ENDIF
   STORE Unindexed + 1 TO Unindexed
   SKIP
ENDDO
?
?                                        TOTAL COSTS TO DATE: ' -;
                                                    STR(Sum,9,2)

STORE LineCnt + 2 TO LineCnt
STORE 0 TO Spacer
IF LineCnt = 40
   ? CHR(12)
   STORE 0 TO LineCnt
ELSE
   ?
   ?
   ?
ENDIF

USE B:Billings
? 'BILLED TO DATE FOR &MClient-&MJob:Nmbr'
?
? 'DATE   INV#   DESCRIPTION                    TAXABLE'+;
   '         TAX     TAX FREE'
?
STORE LineCnt + 4 TO LineCnt
STORE 0 TO Sum
STORE 0 TO T
STORE 0 TO S
STORE 0 TO F
DO WHILE .NOT. EOF
   IF Job:Nmbr = &MJob:Nmbr
      DISPLAY Next 1 Inv:Date, Inv:Nmbr, Descrip,STR(Taxable,9,2)+' ';
          STR(Sales:Tax,9,2)+' ',TaxFree FOR Job:Nmbr = &MJob:Nmbr OFF
      STORE T + Taxable TO T
      STORE S + Sales:Tax TO S
      STORE F + TaxFree TO F
      STORE Sum + Taxable + Sales:Tax + TaxFree TO Sum
      STORE LineCnt + 1 TO LineCnt
      STORE Spacer + 1 TO Spacer

      IF Spacer = 10
         ?
         STORE 0 TO Spacer
      ENDIF

      IF LineCnt = 50
         ? CHR(12)
         STORE 0 TO LineCnt
         STORE 0 TO Spacer
         ? 'DATE   INV#  DESCRIPTION   TAXABLE       TAX      TAX FREE'
         ?
      ENDIF
   ENDIF
```

```
            SKIP
         ENDDO
         ?
         ? '                                    SUB-TOTALS : '+ STR(T,9,2) + '  ';
            + STR(S,9,2)+'    ' + STR(F,9,2)
         ?
         ? '                                          TOTAL BILLED TO DATE: ' -;
                                                           STR(Sum,9,2)

         ? CHR(12)
         SET PRINT OFF

         ? '     F if Finished,'
         ? '  <Return> to continue.'
         WAIT TO Again
      ENDIF okay to do the report
   ENDIF
ENDDO Again

IF Unindexed > 50
   ERASE
   @ 5,0
   ? '   There are ' - STR(Unindexed,9) + ' unindexed records'
   ? '   in the Costbase.  To speed up the Job Costs procedure,'
   ? '   please reindex from the next menu.'
   ? '<Return> to continue.'
   WAIT
ENDIF

RELEASE All
RETURN
```

```
   ********************* JOBSINDX COMMAND FILE  ***********************
* Indexes the costbase on job numbers to B:Jobs.NDX.
*      The method of indexing here allows us to use the index to help.
* find job numbers for the Job Costs command files, but allows us to
* do so without having to index the Costbase every time we add a bill.
*      The strategy is: before we index the Costbase on job numbers,
* we first store the number of the last record in a record with a job
* number of zero.  When the file is indexed, this record is at the top
* of the indexed file ($Jobs) so that we can find it whenever we want to.
*******************************************************************************

USE B:Costbase
GO BOTTOM
STORE STR(#,5) TO Temp
GO TOP    '
IF Job:Nmbr = 0
   REPLACE Name WITH Temp
ELSE
   DO WHILE !(Code) <> 'H'
      ? "Uh, Oh--trouble.  Don't touch anything"
      ACCEPT 'and call Hal.' TO Code
   ENDDO
ENDIF

DELETE FILE B:$Jobs.NDX
ERASE
@ 5,0 SAY 'There are ' + Temp + ' records to index.'
SET TALK ON
INDEX ON Job:Nmbr TO B:$Jobs
SET TALK OFF

RELEASE Temp
RETURN
```

```
          ********** FINDBILLS COMMAND FILE **********
* This procedure finds specific bills that we are looking for, then allows
* us to edit them.
*      The bill can be specified by bill number and/or amount.  If you decide
* not to pay a bill that was found specifying more than one item, you will be
* presented the rest of the entries for the supplier based on name only.
*************************************************************************.****

SELECT PRIMARY
USE B:CostBase INDEX B:$Supp

STORE 'N' TO Finished
DO WHILE !(Finished) <> 'F'

   * "Entering" controls a closed loop that allows the operator to change
   * the entry if he or she spots and error.
   STORE "C" TO Entering
   DO WHILE !(Entering) = 'C'
      ERASE
      @ 4,0
      ACCEPT ' NAME OF SUPPLIER ' TO MName
      ACCEPT '   INVOICE NUMBER ' TO MBill:Nmbr
      ACCEPT '     ENTER AMOUNT ' TO Temp
      STORE !(MName) TO MName
      STORE !(MBill:Nmbr) TO MBill:Nmbr
      STORE VAL(Temp) TO MAmount
      STORE MAmount*1.00 TO MAmount
      @  6,19 SAY MName
      @  7,19 SAY MBill:Nmbr
      @  8,19 SAY MAmount
      @ 11, 0 SAY '         C to CHANGE,'
      ? '     <Return> to continue.'

      * OneByOne is used so that we look at the entire listing for a name once
      * If we could have started in the middle of the list and the bill is not
      * the one we want, we go up to the first listing then go through all the
      * entries for the name, one by one.  Used in the last loop in this file.
      IF Bill:Nmbr > '      ' .OR. Amount <> 0
         STORE 0 TO OneByOne
      ELSE
         STORE 1 TO OneByOne
      ENDIF

      WAIT TO Entering
   ENDDO Entering


   STORE T TO Looking
   @ 11, 0 SAY "I'M LOOKING, I'M LOOKING!!"
   @ 12,0
   @ 13,0

   * Now look for a match on the first 10 characters of the name.  This finds
   * the first entry for that supplier, then looks for bill number or amount
   * if we specified them.  If not specified, it skips through all the entries
   * for the name.

   IF LEN(MName) > 10
      STORE $(MName,1,10) TO Key
   ELSE
      STORE MName TO Key
   ENDIF

   FIND &Key
```

```
@ 11, 0
IF # = 0
   ?
   ? "   GEE, I CAN'T FIND THE NAME.  Please check the spelling."
   ? "   Or maybe it hasn't been posted to the COSTBASE yet."
   ? '<Return> to continue.'
   WAIT
   ERASE
ELSE
   * Found at least one entry with a matching name.
   STORE T TO Looking
   IF MBill:Nmbr = ' ' .AND. MAmount = 0
      STORE F TO Looking
   ELSE
      * If we have more than the name, we first check for the bill number.
      IF MBill:Nmbr > ' '
         DO WHILE Name=Key .AND. .NOT. EOF .AND. Looking
            IF Bill:Nmbr <> MBill:Nmbr
               SKIP
            ELSE
               STORE F TO Looking
            ENDIF
         ENDDO

         * If we're on a new name or the end of the file, Looking is TRUE
         * because we have not found the supplier we were looking for.
         * Otherwise, we have a matching bill number to confirm.
         IF Looking
            ? '   This BILL NUMBER is not in the costbase.'
            ? '<Return> to continue.'
            WAIT
         ENDIF
      ELSE

         * If no bill number, look for the amount.
         DO WHILE Name=Key .AND. .NOT. EOF .AND. Looking
            IF Amount <> MAmount
               SKIP
            ELSE
               STORE F TO Looking
            ENDIF
         ENDDO

         * If we're on a new name or the end of the file, Looking is TRUE
         * Otherwise, we have an unpaid bill to confirm.
         IF Looking
            ? '   No bill for this amount and this supplier.'
            ? '<Return> to continue.'
            WAIT
         ENDIF
      ENDIF we have the bill number
   ENDIF we have only the name
ENDIF there is an unpaid bill for the supplier

STORE 'N' TO Changing
DO WHILE !(Changing) <> 'Y' .AND. .NOT. Looking
   @ 12,0
   DISPLAY
   ? CHR(7)
   ? '         E to EDIT this record,'
   ? '         Q to QUIT this supplier,'
   ACCEPT '      <Return> to continue.' TO Changing
   ?
```

```
      IF !(Changing) = 'Q'
         STORE T TO Looking
      ELSE
         IF !(Changing) = 'E'
            STORE STR(#,5) TO Found
            EDIT &Number
            ERASE
         ELSE
            * If the first record is not the one we want, we skip through the
            * rest of the entries for the name.  We first go on from where we
            * were in the listing (if we had more than the name), then go back
            * to the first entry and look at those we had skipped.  If we had
            * only the name, OneByOne = 1 and we go through the list only once.
            SKIP
            IF EOF .OR. Name <> Key
               IF OneByOne = 0
                  FIND &Key
                  STORE 1 TO OneByOne
               ELSE
                  @ 11, 0 SAY CHR(27) + CHR(74)
                  ? "We've gone through all the entries for " + MName+'.'
                  ? '<Return> to continue.'
                  STORE T TO Looking
                  WAIT
               ENDIF
            ENDIF we've gone through the list
         ENDIF is it the right record
      ENDIF
   ENDDO Changing the record
   ?
   ? '        F if FINISHED finding bills,'
   ? '     <Return> to continue.'
   ? CHR(7)
   WAIT TO Finished
ENDDO Finished
```

```
        ********** REVIEW.CMD FILE **********
* This is used to list entries in any .DBF file.  The database must be named in
* the command file calling the procedure.  Records may be listed conditionally,
* with or without the record numbers.
*       Records are listed in groups of 10 with a line space between each group.
* Processing can be continuous, or can stop after every group of 10.
*       The listing can start on a specified record number.
*       The files can be re-listed as many times as desired.
*       Printing is optional.  The "CHR(X)" commands are for a Diablo 1650
* printer.
************************************************************

STORE 'Y' TO Reviewing
DO WHILE !(Reviewing)='Y'
   COPY STRUCTURE EXTENDED TO Temp
   GO BOTTOM
   STORE STR(#,5) TO Last
   ERASE
   ?
   ? 'The '+!(Database)+' database has '-Last+' entries.  They will be shown'
   ? 'in groups of 10 records, 50 records to a page if printed.'
   ? 'Enter new values for defaults or press <Return>:'
   ?
   ? '***  DISPLAY  [Field list] [FOR <expression>] [OFF]  ***'
   ?
   STORE   1 TO First
   STORE   1 TO PageCnt
   STORE VAL(Last) TO RecoCnt
   STORE 'N' TO Pause
   STORE 'N' TO Partial
   STORE 'N' TO Conditions
   STORE 'N' TO Tally
   STORE 'C' TO Changing
   DO WHILE !(Changing) = 'C'
      @  8,10 SAY 'START ON RECORD NUMBER ' GET First
      @  9,10 SAY ' STOP ON RECORD NUMBER ' GET RecoCnt
      @ 10,10 SAY ' START PAGE NUMBERS ON ' GET PageCnt
      @ 11,10 SAY 'PAUSE EVERY 10 RECORDS ' GET Pause
      @ 12,10 SAY '  SHOW SELECTED FIELDS ' GET Partial
      @ 13,10 SAY 'DISPLAY FOR EXPRESSION ' GET Conditions
      @ 14,10 SAY '   SHOW RECORD NUMBERS ' GET Tally

      ?
      ? '     C to CHANGE the defaults,'
      ? '   <Return> to continue.'
      WAIT TO Changing

      IF !(Changing) = 'C'
         * Clear to end of screen on IBM 3101
         @ 15,0  SAY CHR(27)+CHR(74)
         READ
      ELSE
         IF First > VAL(Last) .OR. First <= 0 .OR. RecoCnt > VAL(Last);
                        .OR. RecoCnt <= 0
            @ 15,0  SAY CHR(27)+CHR(74)
            @ 16,0  SAY 'Sorry, wrong number:  '-!(Database)+' contains '+;
                        'records 1 through'+Last+'.'
            ? '<Return> to correct your entry.'
            WAIT
            @ 15,0 SAY CHR(27)+CHR(74)
            STORE 'C' TO Changing
            STORE 1 TO First
            STORE VAL(Last) TO RecoCnt
         ENDIF
```

```
   ENDIF
   * Clears to end of screen on IBM 3101
   @ 15,0  SAY CHR(27)+CHR(74)
ENDDO
?
?
?
?
?
?
?
?
?
?
?
?
?
?
IF !(Partial)= 'Y'
      @ 11,0  SAY CHR(27)+CHR(74)
      @ 11,0 SAY 'The '+!(Database)+' database consists of these FIELDS:'
      USE Temp
      ?
      STORE ' ' TO Choices
      DO WHILE .NOT. EOF
         STORE Choices+TRIM(Field:Name)+', ' TO Choices
         SKIP
      ENDDO
      STORE $(Choices,2,LEN(Choices)-3) TO Choices
   STORE 'Y' TO Unfinished
   DO WHILE !(Unfinished) = 'Y'
      @ 13, 0 SAY Choices

      USE B:&Database
      ?
      ? 'List FIELDS to display (<return> to show all).'
      ?
      ACCEPT '        DISPLAY ' TO Partial
      STORE !(Partial) TO Partial
      STORE Partial TO String
      STORE LEN(String) TO Size

      IF Size =0 .OR. (Size=1 .AND. Partial=' ')
         STORE CHR(0) TO Partial
         STORE 'N' TO Unfinished
      ELSE
         ?
         ? 'Want to change it (Y or N)?'
         WAIT TO Unfinished
         IF !(Unfinished) = 'Y'
            @ 12, 0 SAY CHR(27) + CHR(74)
         ELSE
            @ 10,0 SAY CHR(27) + CHR(74)
            ? '*** Checking fields ['+Partial+'] : '
            ?
            STORE 0 TO F
            STORE 0 TO Counter
            DO WHILE Size >0
               STORE Counter + 1 to Counter
               ?? ' #'+STR(Counter,2)
               STORE @(',', String) TO Mark
               IF Mark = 1 .OR. Mark = Size
                  ? 'Uh, oh--trouble:  comma cannot be at the ';
                        +'start or end of a list of values.'
```

```
                  ? '<Return> and try again.'
                  STORE 0 TO Size
                  STORE 'Y' TO Unfinished
                  WAIT
               ELSE
                  IF Mark > 0
                     STORE (Mark - 1) TO Size
                  ENDIF

                  STORE T TO Blank
                  STORE 1 TO Start
                  DO WHILE Blank .AND. (.NOT. Start > Size)
                     IF $(String, Start, 1)=' '
                        STORE (Start + 1) TO Start
                     ELSE
                        STORE (.NOT. Blank) TO Blank
                     ENDIF
                  ENDDO

                  IF Start > Size
                     ? 'How on earth can I find a blank field?'
                     ? '<Return> and try again.'
                     STORE 0 TO Size
                     STORE 'Y' TO Unfinished
                     WAIT
                  ELSE
                     STORE (F + 1) TO F
                     IF F < 10
                        STORE STR(F,1) TO Suffix
                     ELSE
                        STORE STR(F,2) TO Suffix
                     ENDIF
                     STORE 'FIELD'+Suffix TO Field
                     STORE TRIM($(String,Start,(Size-Start+1) )) TO &Field

                     IF Mark > 0
                        STORE TRIM($(String, (Size + 2))) TO String
                        STORE LEN(String) TO Size
                     ELSE
                        STORE 'N' TO Unfinished
                        STORE 0 TO Size
                     ENDIF
                  ENDIF
               ENDIF
            ENDDO
         ENDIF
      ENDIF
   ENDDO

   IF LEN(Partial) > 0
      DO headings
      ? "WE'D DO THE HEADINGS HERE."
      WAIT
   ENDIF

ELSE
   STORE CHR(0) TO Partial
ENDIF

IF !(Conditions) = 'Y'
   STORE 'Y' TO Unfinished
   DO WHILE !(Unfinished) = 'Y'
      @ 11, 0 SAY CHR(27)+CHR(74)
      @ 11, 0 SAY 'Specify the EXPRESSION or <Return> to skip.'
```

```
      ?
      ? 'DISPLAY &Partial FOR '
      ACCEPT TO Expression
      ?
      ? 'Do you want to change the expression (Y or N)?'
      WAIT TO Unfinished
   ENDDO

   IF Expression > '
      STORE 'FOR '+ Expression TO Conditions
   ELSE
      STORE CHR(0) TO Conditions
   ENDIF
ELSE
   STORE CHR(0) TO Conditions
ENDIF

IF!(Tally) <> 'Y'
   STORE 'OFF' TO Tally
ELSE
   STORE CHR(0) TO Tally
ENDIF

STORE [DISPLAY Next 1 &Partial &Conditions &Tally] TO Command
@ 11, 0 SAY CHR(27)+CHR(74)
@ 11, 0 SAY '***  '+[DISPLAY &Partial &Conditions &Tally]+'  ***'
?
? 'is the command that will be performed on the '+!(Database)+' database.'
? '      C to CHANGE it,'
? '      Q to QUIT with no action,'
? '   <Return> to review the database.'
WAIT TO Abort

IF !(Abort) = 'Q'
   STORE CHR(0) TO Reviewing
ELSE
   IF !(Abort) <> 'C'
      ERASE
      ? 'Enter a one-line heading or press <Return> to skip.'
      ACCEPT TO Message
      STORE !(Message) TO Message
      ?
      STORE 0 TO Count
      STORE 0 TO PageMark
      STORE STR(First,5) TO Number
      GO &Number

      ERASE
      ? 'Do you want to print the listing now (Y or N)?'
      ACCEPT TO Hardcopy

      IF !(Hardcopy)='Y'
         SET PRINT ON
         DO RevMrgn
      ENDIF

      ERASE
      ? Message
      ? 'Page '+ STR(PageCnt,3)

      IF Tally = 'OFF'
          ?? ' starts on Record #'-STR(#,5)
          ?
          IF .NOT.( Partial > ' ' .OR. Conditions > ' ')
```

```
      DO RevHdr
   ENDIF
ENDIF
?
DO WHILE .NOT. EOF .AND. # <= RecoCnt
   &Command

   IF !(Conditions) > CHR(0)
      IF &Expression
         STORE (Count + 1) TO Count
      ENDIF
   ELSE
     STORE (Count + 1) TO Count
   ENDIF
   SKIP

   IF Count=10
      STORE 0 TO Count
      * Inserts a space every ten records, then waits. The printer
      * is turned off so that "WAIT" does not print on the hardcopy.
      ?
      SET PRINT OFF
      IF !(Pause) = 'Y'
         WAIT
      ENDIF

      IF !(Hardcopy) = 'Y
         SET PRINT ON
      ENDIF

      * The following routine prints 50 entries to a page,
      * then moves to the next page and prints a heading

      STORE (PageMark + 1) TO PageMark
      IF PageMark = 5
         ? CHR(12)
         STORE (PageCnt + 1) TO PageCnt

         IF INT(PageCnt/7) = PageCnt/7
            ?
         ENDIF

         ? Message
         ? 'Page '+STR(PageCnt,3)

         IF Tally = 'OFF'
            ?? ' starts on Record #'-STR(#,5)
            ?
            IF .NOT.( Partial > ' ' .OR. Conditions > ' ')
               DO RevHdr
            ENDIF
         ENDIF
         ?
         STORE 0 TO PageMark
       ENDIF
    ENDIF
ENDDO

* Formfeed on Diablo 1650 printer
? CHR(12)
SET PRINT OFF
SET RAW ON
SET MARGIN TO 38
? 'Do you want to see the '+!(Database)+' again (Y or N)?'
```

```
         WAIT TO Reviewing
      ELSE
         STORE 'Y' TO Reviewing
      ENDIF
   ENDIF
   ?
ENDDO Reviewing

USE
DELETE FILE Temp
RELEASE All
RETURN
```

```
  ******************** REVHDR COMMAND FILE ************************
* Used by Review.CMD to print headings for different database listings.
*****************************************************************************

IF !(Database)='INSERTS'
   ? 'IO#  MAGAZINE        ISSUE   JOB    AD              SPACE
    '    GROSS       NET    X  DATE'
ELSE
   IF !(Database)='BILLINGS'
      ? 'INV#  JOB     DATE    TAXABLE       TAX    NO:TAX     PO#    DESCRIPTION'
   ELSE
      IF !(Database)='INVOICES'
         ? 'INV#   CLT   DATE    TAXABLE       TAX    NO:TAX      '+;
                'TOTAL  AMT:RCD   DATE'
      ELSE
         IF !(Database)='COSTBASE'
            ? 'DATE   CHECK  JOB     AMOUNT  NAME                  '+;
                   'DESCRIPTION           DATE    BILL#  HOURS EMP'
         ELSE
            IF !(Database)='DEPOSITS'
               ? 'DATE    RECEIVED FROM         CHECK     AMOUNT  '+;
                                'INV#  COMMENTS'
            ENDIF
         ENDIF
      ENDIF
   ENDIF
ENDIF
```

```
  ******************** REVMRGN COMMAND FILE ************************
* Used by Review.CMD to set margins for different database listings.
*******************************************************************************

IF !(Database)='INSERTS'
   SET MARGIN TO 38
ELSE
   IF !(Database)='COSTBASE'
      SET MARGIN TO 36
   ELSE
      SET MARGIN TO 45
   ENDIF
ENDIF
RETURN
```

```
****************** SALES TAX COMMAND FILE ********************
* This file summarizes the invoice file for a specified period.
*      It shows the invoices and the type of billing (taxable or
* service) along with the totals for the two types and the total
* sales tax liability for the period.
*      It also includes materials and equipment subject to a use tax
* that has not been paid.  These are entered in the invoices database
* when they come in as well as in the Postfile.
*****************************************************************

USE B:Invoices
ERASE
? 'This file summarizes the data you need to prepare the End-of-Quarter'
? 'report to the State Board of Equalization for SALES TAX collected by'
? 'the agency.  It includes use tax on materials bought out of state or'
? 'bought with our resale number without paying a use tax.'

STORE 'C' TO Dating
DO WHILE !(Dating) = 'C'
   STORE 'YYMMDD' TO Start
   STORE 'YYMMDD' TO Finish
   @ 7, 0 SAY 'This summary is for the period FROM ' GET Start
   @ 7,45 SAY '  TO ' GET Finish
   READ
   @ 9,0 SAY ' '
   ? '   C to CHANGE,'
   ? '<Return> to continue.'
   WAIT TO Dating
   @ 7,0
   ? CHR(27) + CHR(74)
ENDDO Dating

ERASE
@ 5,10 SAY '************ DO NOT INTERRUPT ************'
@ 7,10 SAY 'COMPUTING THE QUARTERLY SALES TAX REPORT'
?
COPY TO Temp FIELDS Inv:Nmbr, Inv:Date,Taxable,Sales:Tax,TaxFree,Amount;
 FOR Inv:Date >= Start .AND. Inv:Date <= Finish
USE Temp
SORT ON Inv:Nmbr TO Temp2
USE Temp2
REPLACE Inv:nmbr WITH ' USED' FOR VAL(Inv:Nmbr) < 1000

STORE $(Start,3,2)+'/'+$(Start,5,2)+'/'+$(Start,1,2) TO Start
STORE $(Finish,3,2)+'/'+$(Finish,5,2)+'/'+$(Finish,1,2) TO Finish

@  5,0
SET MARGIN TO 45
SET PRINT ON
STORE 1 TO PageCnt
? 'SALES TAX SUMMARY FROM '+Start+' TO '+Finish+':  Page '+STR(PageCnt,3)
?
? 'INV#     DATE      TAXABLE          TAX     SERVICE         TOTAL'
?
STORE 0 TO Count
STORE 0 TO PageMark
GO TOP
DO WHILE .NOT. EOF
   DISPLAY Inv:Nmbr,Inv:Date,Taxable,Sales:Tax,TaxFree,'  '+STR(Amount,9,2) OFF
   STORE (Count + 1) TO Count
   SKIP
   IF Count=10
      STORE 0 TO Count
      * Inserts a space every ten records, then waits.  The printen
```

```
      * is turned off so that "WAIT" does not print on the hardcopy.
      ?
      * The following routine prints 50 entries to a page,
      * then moves to the next page and prints a heading

      STORE (PageMark + 1) TO PageMark
      IF PageMark = 5
         STORE 0 TO PageMark
         ? CHR(12)
         STORE (PageCnt + 1) TO PageCnt

         * Compensates for an offset caused by the 7 lines/inch printing
         IF INT(PageCnt/7) = PageCnt/7
            ?
         ENDIF

         ? 'SALES TAX SUMMARY FROM ' + Start + ' TO ' + Finish+':  Page ' +;
                STR(PageCnt,3)
         ?
         ? 'INV#    DATE      TAXABLE         TAX    SERVICE        TOTAL'
         ?
      ENDIF
   ENDIF
ENDDO
?
SET PRINT OFF
?
? '              COMPUTING TOTALS NOW.'
?
REPLACE All Inv:Nmbr WITH ' ' FOR VAL(Inv:Nmbr) > 1000
TOTAL ON Inv:Nmbr TO Other
USE Other
REPLACE All Inv:Date WITH 'TOTAL'
REPLACE All Inv:Nmbr WITH 'SALES' FOR Inv:Nmbr = '
SUM Taxable TO Used FOR Inv:Nmbr = ' USED'
SUM Amount TO Sold
STORE Sold + Used TO Gross
SUM Sales:Tax TO Collected
SUM TaxFree TO Service
STORE Collected + Service TO Exempt
STORE Gross - Exempt TO Subject
STORE 0.06*Subject + 0.005 TO Payable

* Print totals of all the invoices
GO TOP
SET PRINT ON
DO WHILE .NOT. EOF
   DISPLAY Inv:Nmbr,Inv:Date,Taxable,Sales:Tax,TaxFree,'  '+STR(Amount,9,2) OFF
   STORE Count + 1 TO Count
   SKIP
ENDDO
IF PageMark > 3
   * Formfeed if not enough room to print the following list
   ? CHR(12)
ENDIF
?
?
? 'ENTER THE FOLLOWING DATA ON THE BOARD OF EQUALIZATION FORM:'
?
* The following segment is not the final, but the state auditor is in right now
* and I've got to get the info out to him and to the state for this month.
* The final version will include all lines in the form, to allow for changes
* in the way we do our business.  Obviously, this is also the place to
* print the form if you want to do that.  Since the form is used only once
```

```
* every three months, we won't automate it entirely.

? '       LINE  1>     TOTAL GROSS SALES:  ' + STR(Sold,9,2)
? '       LINE  2>    SUBJECT TO USE TAX:  ' + STR(Used,9,2)
? '       LINE  3>    TOTAL TRANSACTIONS:  ' + STR(Gross,9,2)
?
? '       LINE  9>    SALES TAX INCLUDED:  ' + STR(Collected,9,2)
? '       LINE 10>  ADVERTISING SERVICES:  ' + STR(Service,9,2)
? '       LINE 11>       TOTAL EXEMPTIONS:  ' + STR(Exempt,9,2)
? '       LINE 12>  SUBJECT TO STATE TAX:  ' + STR(Subject,9,2)
? '       LINE 13>   AMOUNT OF STATE TAX:  ' + STR(0.05*Subject+0.005,9,2)
? '       LINE 14>  SUBJECT TO LOCAL TAX:  ' + STR(Subject,9,2)
?
? '       LINE 19>   AMOUNT OF LOCAL TAX:  ' + STR(0.01*Subject+0.005,9,2)
?
? '       LINE 21>            TOTAL TAXES:  ' + STR(Payable,9,2)
?
? '       LINE 28> TOTAL DUE AND PAYABLE:  ' + STR(Payable,9,2)
? CHR(12)
SET MARGIN TO 38
SET PRINT OFF

RELEASE All
USE
DELETE FILE Temp
DELETE FILE Temp2
DELETE FILE Other
RETURN
```

```
     **********  TIMECALC COMMAND FILE **********
* Verifies that employee name and number match, then
* calculates billing charges for employee time.
************************************************************

SET TALK OFF
ERASE
SELECT PRIMARY
RESTORE FROM B:Constant

GO TOP
DO WHILE .NOT. EOF
   ERASE
   @  4,20 SAY '      ** DO NOT INTERRUPT **
   @  5,20 SAY '      PROCESSING TIME CHARGES

   IF * .OR. Job:Nmbr = 31 .OR. Check:Nmbr <> '---'
      SKIP
   ELSE
      REPLACE Client WITH !(Client),Name WITH !(Name)
      STORE STR(#,4) TO Number
      @  7,20 SAY '          Record # '+Number
      @  8,20 SAY '          '+Name
      ? CHR(7)
      IF Emp:Nmbr<=0 .OR. Emp:Nmbr>MaxEmpl .OR. Hours = 0
         ERASE
         REPLACE Hours WITH Hours*1.00
         REPLACE Emp:Nmbr WITH Emp:Nmbr*1
         @ 4,0 SAY ' '
         DISPLAY
         @ 6,3 SAY 'HOURS='
         @ 6,18 SAY '=EMPLOYEE NUMBER'
         ?
         ? 'Press ANY KEY to correct the EMPLOYEE NUMBER,'
         ? 'or press H to correct the HOURS.'
         WAIT TO Decision
         IF !(Decision) <> 'H'
            @ 6,14 GET Emp:Nmbr
         ELSE
               @ 6,8 GET Hours
         ENDIF
         READ
       ELSE
          SELECT SECONDARY
          USE B:Personne
          STORE T TO Looking
          DO WHILE Looking .AND. .NOT. EOF
             IF $(Name,1,10)=$(P.Name,1,10)
                IF Emp:Nmbr=P.Emp:Nmbr
                   SELECT PRIMARY
                   * Formula optimistically assumes 65 billable hours out
                   * of 75 hours possible in two weeks.  Eff. mult.=3.23
                   REPLACE Amount WITH Pay:Rate*2.8*Hours/65
                   SELECT SECONDARY
                   STORE F TO Looking
                ELSE
                   SELECT PRIMARY
                   STORE T TO Fixing
                   DO WHILE Fixing
                      ERASE
                      @ 4,0 SAY ' '
                      DISPLAY
                      @ 6,16 SAY '=EMPLOYEE NUMBER'
                      ?
```

```
                    ? 'The correct Employee Number is'
                    ?? S.Emp:Nmbr
                    ?? ' for '+S.Name
                    ? 'Press ANY KEY to change the EMPLOYEE NUMBER' -
                    ? 'press N to change the NAME.'
                    WAIT TO Choice
                    IF !(Choice) <> 'N'
                       @ 6,12 GET Emp:Nmbr
                       READ
                       STORE F TO Fixing
                    ELSE
                          @ 5,25 GET Name
                          REPLACE Name WITH !(Name)
                          READ
                          STORE F TO Fixing
                    ENDIF Employee number
                    ERASE
                 ENDDO Fixing
                 SELECT SECONDARY
                 GO TOP
              ENDIF Numbers match
           ELSE
              SKIP
           ENDIF

           IF EOF
              ERASE
              SELECT PRIMARY
              @ 4,0 SAY ' '
              DISPLAY
              @ 6,16 SAY '=EMPLOYEE NUMBER'
              ?
              ? 'This name is not listed in the Personnel file,'
              ? 'so time charges were not calculated.'
              ? 'Press any key to change the name, or write the'
              ? 'record number down and press D to DELETE.'
              WAIT TO Change
              IF !(Change)<> 'D'
                 @ 5,25 GET Name
                 REPLACE Name WITH !(Name)
                 READ
                 SKIP-1
              ELSE
                 ERASE
                 DELETE
                 DISPLAY
                 ?
                 ? 'THIS RECORD HAS BEEN DELETED.'
                 WAIT
              ENDIF Change
              SELECT SECONDARY
           ENDIF no name
        ENDDO Looking
        SELECT PRIMARY
        SKIP
     ENDIF
  ENDIF deleted
NDDO billing calculations

RELEASE All
RETURN
```

```
         **********  PRINTOUT COMMAND FILE  **********
* This file is is used by several other command files.  It prints out a
* listing of the records in a file without the record number.  The
* outut is spaced every 10 records and the printer is positioned back
* at the left margin after the printout.
*      The calling command file determines where the printout starts by
* specifying a value for the variable "Number".
*      This does not show the record numbers.  To do so, use the
* Review.Cmd file.
*************************************************************************

IF VAL(Number) > 0
   GOTO RECORD &Number
ELSE
   GO TOP
ENDIF

STORE 0 TO Count
DO WHILE .NOT. EOF
   IF *
      SKIP
   ELSE
      DISPLAY &Condition
      SKIP
      STORE Count+1 TO Count
      IF Count=10
         STORE 0 TO Count
         * Spaces one line every 10 records, then waits.  Turns the printer
         * off so that "WAIT" does not print.
         ?
         SET PRINT OFF
         WAIT
         IF !(Output)= 'Y'
            SET PRINT ON
         ENDIF
      ENDIF
   ENDIF
ENDDO
* The next 2 lines reposition the printer at the
* left margin.
?
SET PRINT OFF

RELEASE Count, Output
RETURN
```

```
      **********  GETDATE COMMAND FILE  **********
* Confirms that the date is entered as YYMMDD by checking to see that
* the entries for each item are in the correct range.  The year is
* checked against a constant stored in the B:Constant.MEM file.
*************************************************************************

STORE "T" TO NoDate
DO WHILE !(NoDate) <> 'F'
   ERASE
   STORE 'YYMMDD' TO Date
   @ 5,10 SAY "Enter TODAY'S date" GET Date
   ? CHR(7)
   READ

   IF VAL($(Date,1,2)) <> ThisYear;
         .OR. VAL($(Date,3,2)) < 1 .OR. VAL($(Date,3,2)) > 12;
         .OR. VAL($(Date,5,2)) < 1 .OR. VAL($(Date,5,2)) > 31
      @ 10,25 SAY 'DATE ERROR'
      STORE 0 TO X
      DO WHILE X < 50
         STORE X + 1 TO X
      ENDDO
   ELSE
      ?
      @ 10,0 SAY '   C to CHANGE the date,'
      ? '<Return> to continue.'
      WAIT TO Change
      IF !(Change) <> 'C'
         STORE 'F' TO NoDate
      ENDIF
   ENDIF
ENDDO NoDate

RELEASE NoDate, Change, X
RETURN
```

```
    **********  DATETEST COMMAND FILE **********
* This file verifies the Bill:Date and Check:Date to see that they are
* in the right format.  If incorrect, the operator may edit them.
*********************************************************
                 *

ERASE
GO TOP

* The variable DATE brings in the NAME of the date field to be checked
* from the command files where this is used.
DO WHILE .NOT. EOF
   @  6,30 SAY ' VERIFYING '+Date+' '

   IF *
      SKIP
   ELSE
      IF &Date <> '
         STORE STR(#,5) TO Found
         STORE T TO NoDate
         DO WHILE NoDate
            @  8,30 SAY ' RECORD '+Found
            @  9,30 SAY ' '+$(&Date,1,2)+'/'+$(&Date,3,2)+'/'+$(&Late,5,2)
            ? CHR(7)
            * The macro symbol is used to get the contents of the date field
            * being checked without creating a new variable.
            IF VAL($(&Date,1,2)) > ThisYear .OR. VAL($(&Date 1,2)) < MinYear;
                  .OR. VAL($(&Date,3,2)) < 1 .OR. VAL($(&Date,3,2)) > 12;
                  .OR. VAL($(&Date,5,2)) < 1 .OR. VAL($(&Date,5,2)) > 31
               ?
               ?
               ? '                              DATE ERROR: Must be YYMMDD '
               ACCEPT 'Enter new Date' TO Temp
               REPLACE &Date WITH Temp
               ERASE
            ELSE
               STORE F TO NoDate
               SKIP
            ENDIF
         ENDDO NoDate
         RELEASE Temp, NoDate
      ELSE
         SKIP
      ENDIF date is not blank

      * Delay to allow date being checked to be read (quickly)
      STORE 0 TO X
      DO WHILE X < 5
         STORE (X + 1) TO X
      ENDDO

   ENDIF deleted or posted
ENDDO
RELEASE All
RETURN
```

```
         **********  NAMETEST COMMAND FILE  **********
* Checks names in the file in USE against the Suppliers file and gives
* the operator the options of editing, adding them to the Suppliers file
* or ignoring them.  If a name is edited, it is presented again.
*************************************************************************
GO TOP
DO WHILE .NOT. EOF
   IF *
      SKIP
   ELSE
      STORE STR(#,5) TO Number
      STORE !(Name) TO Name
      ERASE
      @  4,25 SAY 'CHECKING NAMES '
      @  6,25 SAY 'RECORD '+Number
      @  7,25 SAY Name
      ? CHR(7)
      STORE $(Name,1,10) to Key
      SELECT SECONDARY
      USE B:Supplier INDEX B:Supplier
      FIND &Key
      STORE T TO Again
      STORE 'T' TO Decision
      IF # = 0
         DO WHILE Again
            @  9,20 SAY 'THIS SUPPLIER NAME IS NOT IN THE SUPPLIERS FILE.
            @ 11,20 SAY '      E to EDIT it.
            @ 12,20 SAY '      A to ADD it to the SUPPLIERS file.
            @ 13,20 SAY '      C to CONTINUE.
            ?
            WAIT TO Decision
            IF !(Decision) = 'A'
               APPEND
               SKIP-1
               REPLACE Name WITH !(Name),Address WITH !(Address),City WITH;
                  !(City)
               STORE F TO Again
            ELSE
               IF !(Decision) = 'E'
                  SELECT PRIMARY
                  EDIT &Number
                  REPLACE Name WITH !(Name)
                  SELECT SECONDARY
                  STORE F TO Again
               ELSE
                  IF !(Decision) = 'C'
                     STORE F TO Again
                  ELSE
                     STORE T TO Again
                  ENDIF C
               ENDIF E
            ENDIF A
         ENDDO
      ENDIF 0
      SELECT PRIMARY
      IF !(Decision) <> 'E'
         SKIP
      ENDIF
   ENDIF deleted
ENDDO
RELEASE All
RETURN
```

```
*************** CHECKSTUB COMMAND FILE *****************
* Prints out check numbers, amounts, and balances from the CheckFile when
* SALARIES and BILLS are paid.  When more than one bill is paid by a
* single check, the program totals all the bills against that check if they
* are entered in consecutive order (which they are in the two command files)
* Records are marked for deletion, but can be reviewed and retrieved.
*********************************************************************

ERASE
@ 5,10 SAY '*** DO NOT INTERRUPT ***'
@ 6,10 SAY ' JUST GETTING ORGANIZED'

USE B:Checkfil
TOTAL ON Check:Nmbr TO Scratch FOR .NOT. *
COUNT FOR .NOT. * TO Entries

USE Scratch
UPDATE FROM B:Checkfil ON Check:Nmbr REPLACE Balance
COUNT FOR .NOT. * TO Checks

IF Entries > Checks
   USE B:Checkfil
   ERASE
   @ 4,0 SAY ' '
   SET PRINT ON
   ? '                    THESE INDIVIDUAL BILLS WERE PAID:'
   ?
   ? '          Date    Check  Name                          Amount
   ?
   LIST '          '+Check:Date, Check:Nmbr, Name, Amount, Bill:Nmbr OFF;
        FOR  .NOT.*
   ?
   ?
   ?
ENDIF

USE Scratch
STORE 'Y' TO Doing
DO WHILE !(Doing)='Y'
   ERASE
   SET PRINT ON
   ? '                  MAKE THE FOLLOWING ENTRIES IN THE CHECK BOOK:'
   ?
   ? '          Date    Check  Name                          Amount     Balance'
   ?
   LIST '          '+Check:Date, Check:Nmbr, Name, Amount, Balance OFF
   ?
   ?
   ?
   SET PRINT OFF
   ? 'Do you want to print it again (Y or N)?'
   WAIT TO Doing
ENDDO

SET PRINT ON
?.CHR(12)
SET PRINT OFF

USE B:CheckFil
DELETE All

DELETE FILE Scratch
RELEASE Doing, Checks, Entries
RETURN
```