#31

dBASE II User's Guide

Victor Institute
3240 S. Higuera Suite A
San Luis Obispo, CA 93401

First VICTOR printing May, 1983.

ISBN 0-88182-000-8 Printed in U.S.A.

CONTENTS

EXHIBITS

CHAPTERS

IMPORTANT SOFTWARE DISKETTE INFORMATION

For your own protection, do not use this product until you have made a backup copy of your Software diskette(s). The backup procedure is described in the user's guide for your computer.

Please read the **DISKID** file on your new software diskette. **DISKID** contains important information including:

- The part number of the diskette assembly.
- The software library disk number (for internal use only).
- The date of the **DISKID** file.
- A list of files on the diskette, with version number, date, and description for each one.
- Configuration information (when applicable).
- Notes giving special instructions for using the product.
- Information not contained in the current manual, including updates, any known bugs, additions, and deletions.

To read the **DISKID** file onscreen, follow these steps:

1. Load the operating system.
2. Remove your system diskette and insert your new software diskette.
3. Enter—

   TYPE DISKID

4. The contents of the **DISKID** file is displayed on the screen. If the file is large (more than 24 lines), the screen display will scroll. Type ALT-S to freeze the screen display; type ALT-S again to continue scrolling.

INTRODUCTION

dBASE II is a powerful and versatile database management program that can perform valuable tasks for you. dBASE II can—

- ▶ Manage small and medium-sized databases made up of files that you create with easy to remember English-like commands.
- ▶ Work as a programming language that uses command files (programs) you create with dBASE II commands.
- ▶ Work as an interactive computer language, using your input from the keyboard.
- ▶ Work as a text editor to modify the command files you write in dBASE II.
- ▶ Generate and print business forms that you create with full-screen editing commands.
- ▶ Generate and print reports containing data from your databases. dBASE II automatically does multiplication, division, sub-totals, and totals every time you make a report.

Using dBASE II, you will soon be able to create complete database systems. You can add, delete, edit, display, and print data from your database, with a minimum of data duplication on file. dBASE II gives you a large measure of program/data independence; when you change your data you don't have to change your programs, and vice versa.

In short, dBASE II will solve many of your data management problems in one software package.

HOW TO USE THIS MANUAL

Two volumes explain the use of dBASE II—the User's Guide and the Reference Manual. New users can get started quickly with the practice exercises in the User's Guide; the Reference Manual is a complete description of dBASE II for new and experienced users alike. The two volumes are not meant to stand alone, although you may find that one suits your needs best.

USER'S GUIDE

The User's Guide is a training manual or tutorial for the dBASE II database management program. It describes operations in the order you use them to create a database. General formats are presented for the dBASE II commands, and their functions and uses are explained in the text. Then hands-on exercises show you how to use the commands to create and manipulate sample data files. Enter the commands as you read the instructions, and you can immediately operate dBASE II on your computer.

The introductory User's Guide does not fully describe every command, and does not discuss all the advanced commands. If you feel that you don't understand a command or expression, look it up in the Reference Manual.

REFERENCE MANUAL

The Reference Manual describes dBASE II operations in functional groups such as files, expressions, and command syntax. Chapters 9-12 are a glossary of detailed descriptions of dBASE II commands in alphabetical order, with examples that supplement those in the User's Guide.

MANUAL CONVENTIONS

This User's Guide uses the following conventions:

▶ In text, all-uppercase is used for the names of commands, files, and data fields.

▶ In the sample entries and in command formats, what you type in at the keyboard is boldface.

— dBASE II commands and keywords are all-uppercase.

— The names of variables, such as files and data fields, are all-lowercase.

This style—uppercase for dBASE II keywords and lowercase for user-supplied variables—is also used in the working accounting system in Appendix A.

▶ The hands-on exercises are illustrated as screen displays with a green background.

— In the screen displays, everything you type (including commands, keywords, names of files and data fields) is all-lowercase. (You can enter commands in any combination of upper- and lowercase.)

— dBASE II replies, except for actual data, are all-uppercase, just as they are on your computer.

Command formats use the following typographic conventions:

▶ Square brackets [ ] enclose optional parts of commands.

▶ Angle brackets < > in commands enclose general terms that you replace with real information. For example, for <filename> in a command format, you enter the name of a database file when you type in the command.

▶ The bracketed word <enter> means press the Return or Enter key on your keyboard. Do not type the word "enter" or the angle bracket symbols.

FIRST, BACKUP YOUR dBASE II DISKETTE

Before you begin the User's Guide exercises or any other work with dBASE II, make a copy of the dBASE II distribution diskette. (Use the DCOPY utility.) Store the original diskette in a safe place and use the copy.

BACKUP YOUR WORK

In working with dBASE II or any computer program, it is essential that you keep current backups of the files you create. You should backup your work frequently, at least once or twice a day, as insurance against accidental loss or damage. For short sessions on your computer, one backup per session may be enough. In general, however, make backups much more frequently than that.

Compare the cost of doing the backups with the cost of the loss of your data; you can rewrite disks, so the cost of backups is low. What's your entire accounting database worth? The importance of keeping backups can't be overemphasized.

HOW TO CALL UP dBASE II ON YOUR SYSTEM

dBASE II is ready for use with your system, and you can begin using the program immediately. First load your operating system. Then insert the copy of your dBASE II diskette into either drive and log onto that drive. If you are using the CP/M-86 operating system, clear the disk system by entering ALT-C. Then, at the system prompt A> or B>, for both MS-DOS and CP/M-86, type—

```
dBASE
```

followed by a Return. You can use any combination of upper- and lowercase to enter dBASE II commands.

In CP/M-86, dBASE II asks for the date (see Reference Chapter 1). If you enter a date, dBASE II records it in your files as the latest access (when you last added to or deleted from the file). The access date is useful for keeping track of updates. If you want dBASE II to ignore the date, just press <enter> or Return.

In MS-DOS, dBASE II does not display a date request because the operating system asks for the date when you load the system.

dBASE II then loads into memory, displays a sign-on message and shows its prompt, a dot (.), to indicate that it is ready to accept commands. From this point you can manipulate databases with the interactive dBASE II commands or you can enter and run programs with the programming function of dBASE II. To learn how to create a database file, turn to User's Guide Chapter 1.

CREATING AND WORKING WITH A DATABASE

This chapter leads you through the steps to create a database file and enter data into it. The basic dBASE II commands are described, and example exercises give you practice using the commands.

For most commands, this User's Guide explains one function that enables you to do a few more things with your database. The User's Guide does not cover all that a command can do. To find out a command's complete potential, look it up in Reference Chapters 9 through 12.

CREATING A DATABASE (CREATE) 1.1

Before you create a new database, you should know the categories of information you will record about the items in the database. In dBASE II, the information categories are called **data fields;** the items or individual listings in the database are called **records.** (See Reference Chapter 2.)

First you must tell dBASE II the name of each data field, what type of data each field will contain, how long each field is, and how many decimal places to allow for numeric data. This information—the names and the lengths of the data fields, and the data type of each field—is called the **record structure.**

In the first practice exercise you will create a database of names for a mailing list. The records in the mailing list file will contain these five data fields:

NAME (15 characters maximum)
ADDRESS (20 characters maximum)
CITY (20 characters maximum)
STATE (2 characters)
ZIP CODE (5 characters)

To begin creating a database, type—

CREATE

followed by a Return. dBASE II responds with—

```
ENTER FILENAME:
```

Because this practice database is a list of names, enter the file name NAMES. Then press Return, and dBASE II creates an empty file called NAMES.DBF. dBASE II supplies the .DBF, which is the default file extension for database files (see Reference Chapter 2).

```
. create
ENTER FILENAME: names
ENTER RECORD STRUCTURE
AS FOLLOWS:
     FIELD      NAME,TYPE,WIDTH,DECIMAL PLACES
     001
```

ENTERING THE RECORD STRUCTURE

dBASE II displays the field number (001), and you enter the name, type, width, and decimal places for the data field.

A field name can be up to 10 characters long, and can be upper- and/or lowercase. The name must start with a letter and can contain digits and embedded colons, but cannot contain spaces. Abbreviate as little as possible to keep the meaning of the field name clear.

Next specify the type of data with a single letter: C for Character, N for Numeric, and L for Logical. All fields in this example contain character data.

Field width can be any length up to 254 characters. In numeric fields with decimal places, the decimal point counts as a character position.

You know what names to give the data fields, the type of data each field will contain, and the field widths. Now type the information into the record structure. Here's what the screen looks like when you're finished:

```
. create
ENTER FILENAME: names
ENTER RECORD STRUCTURE
AS FOLLOWS:
     FIELD     NAME,TYPE,WIDTH,DECIMAL PLACES
     001       name,c,15
     002       address,c,20
     003       city,c,20
     004       state,c,2
     005       zip code,c,5
BAD NAME FIELD
     005       zip:code,c,5
     006       <return>
```

Notice that the example contains an error at field 5—a space was used in the field name. dBASE II told you what the error was and gave you a chance to correct it.

Notice also that the data type for ZIP:CODE was specified as character, even though you may think of zip codes as numbers. In dBASE II the ZIP:CODE field is not numeric because numeric fields can be added by a command such as TOTAL. Adding up ZIP:CODE would be a waste of time. You can still use the relational operators (greater than, less than, equal or not equal to) with character data, so you can do ZIP:CODE sorting anyway.

When dBASE II asks you for the specifications for a sixth field, hit Return to end the structure input. dBASE II saves the structure, then asks if you want to enter data in it. The newly created NAMES.DBF database is immediately ready for data entry, so type y.

1

ENTERING DATA INTO A NEW DATABASE

After you type y in response to INPUT DATA NOW? the screen clears, and dBASE II displays the record number and all the field names in the upper left corner of the screen. The cursor is at the first character position of the first field.

```
RECORD 00001

NAME    :                 :
ADDRESS:                       :
CITY    :                      :
STATE   :  :
ZIP:CODE:      :
```

Colons indicate the length of each data field. When you fill a field or hit Return, the cursor jumps down to the next field. You can move the cursor up to a previous field by holding the ALT (Alternate-function) key down and pressing the letter E once: ALT-E. When you finish with the last field, dBASE II presents the next empty record.

Enter the following names and addresses. You'll be using this file to demonstrate 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 you can't correct by backspacing and retyping, read the next section on EDIT.

If you accidentally get back to the dBASE II dot prompt, type—

```
USE Names
APPEND
```

Then continue with your entries. (USE and APPEND are explained later in this chapter and in the Reference.)

To stop entering data after you've entered the last ZIP:CODE, hit <enter> or Return while you are on the first character of the first field of the next record. If you have typed in some data or moved the cursor in the ninth record, enter ALT-Q to discard that record.

Then dBASE II 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, type QUIT followed by a Return.

You must type QUIT every time you end a dBASE II session to close all files properly. Unless you exit dBASE II with QUIT, you may damage your database.

MODIFYING DATA (EDIT)

Errors made while entering data in your database can be corrected quickly and easily in the full-screen EDIT mode.

To enter EDIT mode, type—

```
USE <filename>
EDIT <n>
```

where: filename is the name of the database you want to edit, and n is the number of the record you want to edit.

To try full-screen editing with the NAMES database, type—

```
USE NAMES
EDIT 3
```

```
RECORD 00003

NAME    :CLINKER, DUANE :
ADDRESS:789 Charles Dr.      :
CITY    :Los Angeles         :
STATE   :CA:
ZIP:CODE:90036:
```

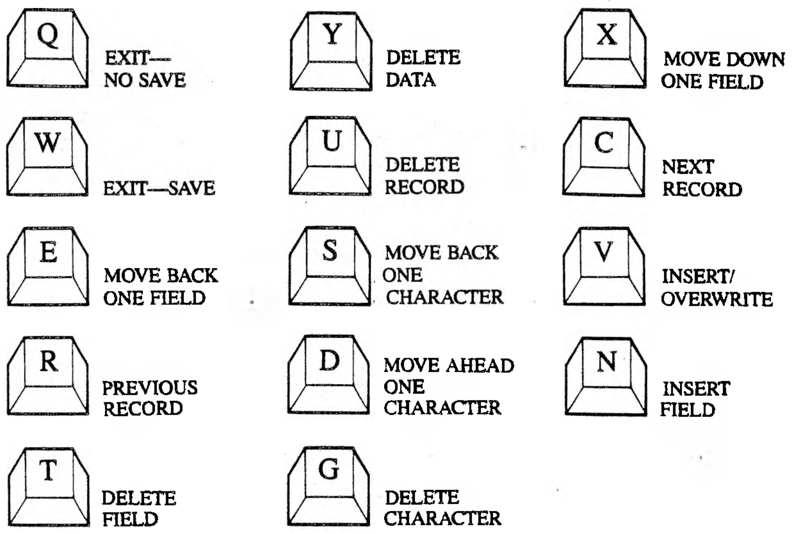
Full-Screen Editing Features

dBASE II brings up the entire record. You can type over the data in the record, and you can use the full-screen editing commands to change the data. The full-screen editing commands are Alternate-function characters, such as ALT-C (move to the next record) and ALT-R (move to the previous record).

Turn to Chapter 7 of the Reference for a summary of the full-screen editing commands. Try some of them, such as ALT-D and ALT-A to move the cursor forwards and backwards. Notice that some Alternate characters have different functions when you use them in different modes (EDIT mode, APPEND mode, MODIFY mode).

Exhibit 1a illustrates the keyboard labels of the full-screen editing commands. To use one of these commands, hold down the ALT key while pressing the appropriate key.

Exhibit 1a: Full-Screen Editing Commands



To exit full-screen editing and save the changes you made, enter ALT-W.

To abort full-screen editing, enter ALT-Q. ALT-Q abandons the changes on the screen when you exit and returns you to the interactive dBASE II dot prompt.

1.2 COMMANDS AND THE ERROR CORRECTION DIALOG (USE, LIST, DISPLAY)

1

dBASE II commands are easy to learn and remember; they are English-like, so learning a new command is like increasing your vocabulary (and your repertoire) by another word. The general form for most dBASE II commands is—

```
VERB SCOPE NOUN CONDITION
```

The first word in a dBASE II command is a verb that names the action to be taken; examples are LIST, DISPLAY, and USE. The scope limits the range of records affected by the verb's action; examples are ALL or the NEXT 3. The noun, which is the object of the verb's action, may be a file, a field, a record, or a variable. The condition specifies the nouns acted on, and is usually a dBASE II expression in a FOR or WHILE phrase.

You can type in commands, in any combination of upper- and lowercase, when dBASE II displays its dot prompt (.). Command lines can be up to 254 characters long. To extend a command beyond the 80-character screen line, type a semicolon (;) as the last character on the line (no space after it). The ; tells dBASE II to use the next line as part of the command.

Here are three basic commands that you will use often.

To tell dBASE II which database you want to work with, type—

```
USE <filename>
```

To look at the record you are on, type—

```
DISPLAY
```

To see all the records in the database, type—

```
LIST
```

LIST scrolls the file; you can stop and start the scrolling with ALT-S.

You can abbreviate dBASE II commands to the first four letters of the command. If you use more letters, all must be correct. For example, DISPLAY, DISP and DISPLA are valid commands; DISPY is not.

After you enter a command, dBASE II scans the command line and prompts you with error messages when mistakes are detected. The error correction dialog gives you a chance to make corrections without retyping the entire line.

To try the error correction dialog, type EDUT 3.

```
. edut 3
***UNKNOWN COMMAND
edut 3
CORRECT AND RETRY (Y/N)?y
CHANGE FROM:u
CHANGE TO   :i
edit 3
MORE CORRECTIONS (Y/N)?n
```

dBASE II repeats the unrecognized command. If you press the Return key, dBASE II discards the command line and gives you the dot prompt again. To correct and retry the command, enter y to begin the error correction dialog.

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

In response to CHANGE TO: type in the replacement for the material you want changed, followed by <enter> or Return.

This example changed only a single letter in a short command, but you'll find the error correction feature most useful for testing and debugging long command lines.

Tip: Use the ERASE command to erase the screen and position the dot prompt at the upper left corner of the screen so that you can start new commands with a clean slate.

1

1.3 USING 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 define exactly what the commands do.

Conditional expressions, such as those beginning with FOR and WHILE, restrict the action of the command to the records or fields that meet the given condition. (The use of expressions is fully explained in Chapter 2.)

In dBASE II expressions, you use "operators" to define the conditions that the data must meet. There are several kinds of operators, described in Chapter 2 and in Reference Chapter 3. The relational operators in expressions are listed in Exhibit 1b.

Exhibit 1b: Relational Operators

OPERATOR	MEANING
<	Less than.
>	Greater than.
=	Equal to.
<=	Less than or equal to.
>=	Greater than or equal to.
<> or #	Not equal to.

The relational operators mean exactly what the explanation in Exhibit 1b says. They compare two parts of an expression and 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.

The LIST command displays the records in the database (you can stop and start scrolling with ALT-S). The full form of the command is—

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

The optional OFF shuts off the display of record numbers.

If you include a FOR phrase, dBASE II lists only the records for which the expression is true.

Now try using LIST with FOR to select files for display. Type the following, using single quotes around the character data (for more on data types, see Chapter 2):

```
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, dBASE II compares only that part. You don't have to give Mr. Green's full name, for example; although you might have used it if your database contained several GREENs.

1

```
. use names
. list
00001  ALAZAR, PAT     123 Crater Rd.     Everett      WA 98206
00002  BROWN, JOHN     456 Minnow Pl.     Burlington   MA 01730
00003  CLINKER, DUANE  789 Charles Dr.    Los Angeles  CA 90036
00004  DESTRY, RALPH   234 Mahogany St.   Deerfield    FL 33441
00005  EMBRY, ALBERT   345 Sage Avenue    Palo Alto    CA 94303
00006  FORMAN, ED      456 Boston St.     Dallas       TX 75220
00007  GREEN, TERRY    567 Doheny Dr.     Hollywood    CA 90046
00008  HOWSER, PETER   678 Dusty Rd.      Chicago      IL 60631

. list off
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 Avenue    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

. list for zip:code='9'
00001  ALAZAR, PAT     123 Crater Rd.     Everett      WA 98206
00003  CLINKER, DUANE  789 Charles Dr.    Los Angeles  CA 90036
00005  EMBRY, ALBERT   345 Sage Avenue    Palo Alto    CA 94303
00007  GREEN, TERRY    567 Doheny Dr.     Hollywood    CA 90046

. list for zip:code<'8'
00002  BROWN, JOHN     456 Minnow Pl.     Burlington   MA 01730
00004  DESTRY, RALPH   234 Mahogany St.   Deerfield    FL 33441
00006  FORMAN, ED      456 Boston St.     Dallas       TX 75220
00008  HOWSER, PETER   678 Dusty Rd.      Chicago      IL 60631

. list for name='GREEN'
00007  GREEN, TERRY    567 Doheny Dr.     Hollywood    CA 90046
```

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

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

LIST FILES shows the names of the database files (files with extension .DBF) on the logged drive. LIST FILES ON <drive> shows the database files on the drive you name (no colon after the drive name).

```
. use names
. list structure
STRUCTURE FOR FILE:  NAMES.DBF
NUMBER OF RECORDS: 00010
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
FLD    NAME      TYPE  WIDTH  DEC
001    NAME       C     015
002    ADDRESS    C     020
003    CITY       C     020
004    STATE      C     002
005    ZIP:CODE   C     005
**TOTAL**             00063

. list files
DATABASE  FILES    #RCDS   LAST UPDATE
NAMES     DBF      00010   00/00/00
MIND      DBF      00007   00/00/00
KEYFILE   DBF      00211   00/00/00
CHECKS    DBF      00783   00/00/00
TEMP      DBF      00010   00/00/00
MONEYOUT  DBF      00000   00/00/00
ORDERS    DBF      00000   00/00/00
```

LOOKING AT YOUR DATA RECORDS (DISPLAY) 1.4

The DISPLAY command is similar to LIST. Its full form is—

```
        [ALL       ]
DISPLAY [Record n] [OFF][FOR <expression>]
        [NEXT n    ]
```

The three options following the keyword DISPLAY—ALL, Record n, or NEXT n—are the scope of the command. Specifying Record n displays only record number n; NEXT n displays the next n records, including the current record. DISPLAY ALL is the same as LIST, except that LIST scrolls the records in the database up the screen, while DISPLAY ALL shows you the database in groups of 15 records (you press any key to display the next 15 records).

Type the following:

DISPLAY ALL
DISPLAY Record 3
DISPLAY NEXT 4

```
. display all
00001  ALAZAR, PAT      123 Crater Rd.    Everett       WA 98206
00002  BROWN, JOHN      456 Minnow Pl.    Burlington    MA 01730
00003  CLINKER, DUANE   789 Charles Dr.   Los Angeles   CA 90036
00004  DESTRY, RALPH    234 Mahogany St.  Deerfield     FL 33441
00005  EMBRY, ALBERT    345 Sage Avenue   Palo Alto     CA 94303
00006  FORMAN, ED       456 Boston St.    Dallas        TX 75220
00007  GREEN, TERRY     567 Doheny Dr.    Hollywood     CA 90046
00008  HOWSER, PETER    678 Dusty Rd.     Chicago       IL 60631

. display record 3
00003  CLINKER, DUANE   789 Charles Dr.   Los Angeles   CA 90036

. display next 4
00003  CLINKER, DUANE   789 Charles Dr.   Los Angeles   CA 90036
00004  DESTRY, RALPH    234 Mahogany St.  Deerfield     FL 33441
00005  EMBRY, ALBERT    345 Sage Avenue   Palo Alto     CA 94303
00006  FORMAN, ED       456 Boston St.    Dallas        TX 75220
```

As with LIST, you can include an optional FOR clause to select specific data by using logical expressions.

The DISPLAY command can also be used like the LIST command for system functions. DISPLAY STRUCTURE is the same as LIST STRUCTURE, and DISPLAY FILES is the same as LIST FILES.

You can use the word LIKE and the wild-card characters * and ? to LIST or DISPLAY file names or data fields. The general form to use is—

DISPLAY FILES LIKE <wild-card>

For example, DISPLAY FILES LIKE *.COM ON B displays all the .COM files on drive B. See your operating system User's Guide for details on wild-card characters.

POSITIONING YOURSELF IN THE DATABASE (GO or GOTO and SKIP) 1.5

dBASE II has an invisible record pointer, an imaginary "finger" that marks your position in the database. When you first open a file with the USE command, the pointer is on the first record. You can move from record to record with the GO and SKIP commands. For practice, 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     WA 98206

. go bottom
. display
00008  HOWSER, PETER    678 Dusty Rd.     Chicago     IL 60631

. goto 5
. display
00005  EMBRY, ALBERT    345 Sage Avenue   Palo Alto   CA 94303

. 8
. display
00008  HOWSER, PETER    678 Dusty Rd.     Chicago     IL 60631
```

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, GO, or even just <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.

To see how SKIP works, type the following:

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

```
 . display
 00008  HOWSER, PETER   678 Dusty Rd.    Chicago     IL 60631

 . skip-3
RECORD:00005

 . display
 00005  EMBRY, ALBERT   345 Sage Avenue  Palo Alto   CA 94303

 . skip
RECORD:00006

 . display
 00006  FORMAN, ED      456 Boston St.   Dallas      TX 75220
```

USING THE INTERACTIVE ? COMMAND (?) 1.6

With the ? command you can use your computer in the interactive calculator mode. You type a question mark and a space, followed by the quantity or mathematical function you want evaluated. dBASE II provides the answer on the next line. The ? command answers a mathematical operation to the same number of decimal places as the maximum in the numbers entered. Using ?? puts the answer on the same line with the command. Type the following:

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

```
. ? 73/3.0000
  24.3333
. ? 73.00/3
  24.33
. ? 73/3
  24
```

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

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

```
. use names
. 6
. ? zip:code
75220
. ? name
FORMAN, ED
. ? state
TX
. skip
RECORD:00007
. ? name
GREEN, TERRY
. go bottom
. ? city
Chicago
```

Chapter 2 shows you how to use the ? to access other dBASE II functions, and how to display CRT prompts from a command file.

1.7 ADDING RECORDS TO A DATABASE (APPEND, INSERT)

To add data records to a database, first choose the file you want to expand by typing USE <filename>. Then type APPEND:

USE Names
APPEND

```
. use names
. append

RECORD #:00009

NAME    :                 :
ADDRESS:                      :
CITY    :                     :
STATE   :  :
ZIP:CODE:      :
```

dBASE II displays a new blank record (with the number that follows the last existing record in the file) and the fields for that database. If you fill in the record, dBASE II adds it to the end of the file (appended).

APPEND shows field lengths with colons. The cursor is at the first position where you can enter data. If you fill the entire field, the cursor automatically moves down to the next field. If your data does not fill the field, hit <enter> or Return to move to the next field.

If there is no data to be entered in a field, use <enter> to move the cursor to the next field. Character fields are automatically filled with blanks; numeric fields show a zero. When entering numeric data with no digits after the decimal, you do not need to type the decimal. dBASE II automatically puts in the decimal point and the number of following zeros specified in the record structure.

You can insert records into a specific location in a database (to keep the file alphabetical, for example) by using the INSERT command—

```
INSERT [BEFORE] [BLANK]
```

Entering INSERT inserts a blank record just after the current record. Specifying BEFORE inserts the record just before the current record. In either case, dBASE II prompts you for data input as with the APPEND and CREATE commands. If you specify BLANK, an empty record is inserted and there are no prompts.

For practice with APPEND and INSERT, 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      (Then enter the data for the first name.)
APPEND             (Then enter the data for the last names.)
```

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

To exit the APPEND mode, position the cursor at the start of a new record, then hit <enter>. If you are in the middle of a record, ALT-Q will let you exit and the current record will be lost.

In APPEND and INSERT modes, you can exit from inside a record; ALT-W saves what you have entered and returns you to the command mode.

1.8 CLEANING UP A DATABASE (DELETE, RECALL, PACK)

You can delete records in both EDIT mode and command mode. In EDIT, you use ALT-U to mark a record for deletion. In the interactive command mode, you can mark the current record for deletion by typing DELETE. Rather than erasing the data, DELETE and ALT-U mark each record with an asterisk. You can see the asterisks when you LIST or DISPLAY the deleted records. dBASE II ignores these records, and does not use them in any processing.

To mark 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 scope of the deletions conditional, use—

DELETE [scope] [FOR <expression>]

where: expression is a condition or set of conditions that must be met. (See Chapter 2 for more on expressions.)

You can easily recover records marked for deletion. To restore records, use the following command:

RECALL [scope] [FOR <expression>]

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

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

```
PACK
```

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

BE CAREFUL: Once you use PACK, 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 next screen shows the first few records in NAMES.DBF as you perform these commands.

```
. list
00001  ALAZAR, PAT     123 Crater Rd.     Everett      WA 98206
00002  BROWN, JOHN     456 Minnow Pl.     Burlington   MA 01730
00003  CLINKER, DUANE  789 Charles Dr.    Los Angeles  CA 90036
00004  DESTRY, RALPH   234 Mahogany St.   Deerfield    FL 33441
00005  EDMUNDS, JIM    392 Vicarious Way  Atlanta      GA 30328

. delete record 2
00001 DELETION(S)
. delete record 4
00001 DELETION(S)
. list
00001  ALAZAR, PAT     123 Crater Rd.     Everett      WA 98206
00002 *BROWN, JOHN     456 Minnow Pl.     Burlington   MA 01730
00003  CLINKER, DUANE  789 Charles Dr.    Los Angeles  CA 90036
00004 *DESTRY, RALPH   234 Mahogany St.   Deerfield    FL 33441
00005  EDMUNDS, JIM    392 Vicarious Way  Atlanta      GA 30328

. recall record 4
00001 RECALL(S)
. list
00001  ALAZAR, PAT     123 Crater Rd.     Everett      WA 98206
00002 *BROWN, JOHN     456 Minnow Pl.     Burlington   MA 01730
00003  CLINKER, DUANE  789 Charles Dr.    Los Angeles  CA 90036
00004  DESTRY, RALPH   234 Mahogany St.   Deerfield    FL 33441
00005  EDMUNDS, JIM    392 Vicarious Way  Atlanta      GA 30328

. pack
PACK COMPLETE, 00004 RECORDS COPIED
. list
00001  ALAZAR, PAT     123 Crater Rd.     Everett      WA 98206
00002  CLINKER, DUANE  789 Charles Dr.    Los Angeles  CA 90036
00003  DESTRY, RALPH   234 Mahogany St.   Deerfield    FL 33441
00004  EDMUNDS, JIM    392 Vicarious Way  Atlanta      GA 30328
```

You can use DELETE to erase entire files. Type DELETE FILE <drive:filename>. Watch out, though—unlike records, deleted files are not recoverable. The data is gone forever.

Before going on to Chapter 2, please CREATE these two files; you will need them for other examples.

```
. create
ENTER FILENAME: moneyout
ENTER RECORD STRUCTURE
AS FOLLOWS:
FIELD   NAME,TYPE,WIDTH,DECIMAL PLACES
001     Check:Date,C,7
002     Check:Nmbr,C,5
003     Client,C,3
004     JobNumber,N,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:
FIELD   NAME,TYPE,WIDTH,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
```

2

ADDING EXPRESSIONS TO COMMANDS

Chapter 1 introduced expressions, and Chapter 2 explains in detail how to use expressions in dBASE II commands. Also Chapter 2 introduces dBASE II operators and more commands. You'll get practice using both expressions and operators as you work your way up to developing command files.

If you're still uncertain about how to write expressions after you've finished this chapter, read a beginning programming text. Most texts discuss expressions in the first chapters. Also, read Reference Chapter 3.

Understanding and using expressions may be the most important part of learning how to use dBASE II effectively. Expressions give you the fine control you need to manipulate your data quickly and easily. And just as important, in order to write application command files, you must first master handling expressions. Then you must learn only two more things about programming—how to make decisions and how to repeat a sequence of commands, both covered in Chapter 3.

USING EXPRESSIONS FOR SELECTION AND CONTROL 2.1

If you check the descriptions of commands in Reference Chapters 9-12, you'll see that many dBASE II commands can be modified in the form—

COMMAND [FOR <expression>]

The <expression> states the conditions the data must meet in order to be affected by the command.

Expressions give you a flexibility that you do not get with other database management systems. Experienced programmers can write a program (a dBASE II command file) 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.

To take advantage of this power, you need to understand how to work with expressions and operators, and 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, but experience is the best teacher. Go through the examples and key them into your computer; you can change names as you go to reflect your own needs.

2.2 CONSTANTS AND VARIABLES (STORE)

Expressions help select and manipulate the data in your database (see DISPLAY). The quantity that you manipulate in the database or in the expression can be either a constant or a variable.

CONSTANTS

Constants are data items that do not change, no matter where they appear in a database or within computer operations. They are literal values—they are exactly what they represent. Examples of constants 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 they are handled differently from other constants.

Strings are a collection of characters (including spaces, digits and symbols) that you can handle, modify, manipulate and otherwise use as data. A substring is a portion of any specific string.

If you want a character or collection of characters (such as words, phrases, or other strings) to be treated as a string constant, you must enclose it in single or double quotes or in square brackets. Then dBASE II understands that it is to deal with the characters as a constant.

Here are the rules for working with character strings:

1. Character strings that appear in expressions must be enclosed in matching single or double quote marks or square brackets.

2. Character strings may contain any of the printable characters (including the space).

3. If you want to use the ampersand (&) as a character, you must place it between two spaces because it is also the dBASE II Macro function (described in Reference Chapter 4).

To see how to use characters as constants, get dBASE II up on your computer and USE NAMES. Then type—

```
? 'Name'
? Name
```

In response to the first "What is...?" (the ? command), the computer responds with NAME because that is the value of the constant. When you eliminate the single quotes, the computer first checks to see if the word is a command. It isn't, so then the computer checks to see if it is the name of a variable.

VARIABLES

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. Now type the following:

```
SKIP 3
? Name
```

```
. use names
. ? 'name'
Name
. ? name
ALAZAR, PAT
. skip 3
RECORD:00004
. ? name
DESTRY, RALPH
```

Now type USE followed by a Return. Since you do not specify a file name, dBASE II closes all files.

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

Variables can 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, and with a maximum total of 1536 characters for all the variables. You can think of dBASE II's memory variables as 64 pigeon-holes where you can temporarily tuck data 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, you might store the date in a variable called DATE. During the session, you 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.PRG in Appendix A).

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

STORE <expression> TO <memory variable>

For some practice with memory variables, 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

2

```
. 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?
```

In the first line, you have to use double quotes around the character string (a constant) because you need 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

```
. store 99 to variable
  99
. store 33 to another
  33
. store variable/another to third
  3
. store '99' to constant
  99
. ? variable/another
           3
. ? variable/3
          33
. ? constant/3
***SYNTAX ERROR***
        ?
? CONSTANT/3

. display memory
MESSAGE                      (C)    How's it going so far?
HOURS                        (N)    10
PAY:RATE                     (N)    17.35
VARIABLE                     (N)    99
ANOTHER                      (N)    33
THIRD                        (N)     3
CONSTANT                     (C)    99
**TOTAL**    07 VARIABLES USED     00054 BYTES USED
```

Entering data into a variable automatically tells dBASE II the data type—digits store as numeric data, digits inside quotes store as characters. You cannot mix data types in dBASE II operations. A syntax error occurred in the previous example because a character string was divided by a number.

The last command in the previous example 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
```

When naming variables, try to make the meaning obvious.

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 than if you invented a completely new and different name.

dBASE II OPERATORS 2.3

Operators are manipulations that you tell dBASE II to perform on your data. Some of them will be familiar; others may take practice.

ARITHMETIC OPERATORS

Arithmetic operators should be the most familiar. They generate arithmetic results, as listed in Exhibit 2a.

Exhibit 2a: Arithmetic Operators

OPERATOR	MEANING
( )	Parentheses for grouping.
*	Multiplication.
/	Division.
+	Addition.
–	Subtraction.

dBASE II evaluates arithmetic operators in a sequence of precedence. The order is—parentheses; multiply and divide; add and subtract. Operators with equal precedence 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

Relational operators make comparisons, and then generate logical results based on whether the comparison is True or False. The relational operators are listed in Exhibit 1b.

Practice using arithmetic and relational operators in expressions by typing 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    FL 33441
00004  EDMUNDS, JIM    392 Vicarious Way   Atlanta      GA 30328
00008  HOWSER, PETER   678 Dusty Rd.       Chicago      IL 60631
00010  JENKINS, TED    210 Park Avenue     New York     NY 10016

. list for address <> '123'
00002  CLINKER, DUANE  789 Charles Dr.     Los Angeles  CA 90036
00003  DESTRY, RALPH   234 Mahogany St.    Deerfield    FL 33441
00004  EDMUNDS, JIM    392 Vicarious Way   Atlanta      GA 30328
00005  EMBRY, ALBERT   345 Sage Avenue     Palo Alto    CA 94303
00006  FORMAN, ED      456 Boston St.      Dallas       TX 75220
00007  GREEN, TERRY    567 Doheny Dr.      Hollywood    CA 90046
00008  HOWSER, PETER   678 Dusty Rd.       Chicago      IL 60631
00009  INDERS, PER     321 Sawtelle Blvd.  Tucson       AZ 85702
00010  JENKINS, TED    210 Park Avenue     New York     NY 10016

. list for name = 'howser'
00008  HOWSER, PETER   678 Dusty Rd.       Chicago      IL 60631
```

LOGICAL OPERATORS

The logical operators generate logical results (True or False). They are listed in Exhibit 2b in the order of precedence within an expression (.NOT. is applied before .AND., and so on).

Exhibit 2b: Logical Operators

OPERATOR	MEANING
( )	Parentheses for grouping.
.NOT.	Boolean not (unary operator).
.AND.	Boolean and.
.OR.	Boolean or.
$	Substring logical operator.

For example, the command:

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

This LIST command would display all the October, 1979 records for costs billed against job numbers 730 and 731 in MONEYOUT.DBF, if it had records in it. Note how the command line was extended with semicolons. When the example command is entered, 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 to display the record. Parentheses function as logical operators just as they do in arithmetic expressions.

Because of the first .AND., dBASE II displays records only when the conditions in both parenthetical statements are true—when JOBNUMBER is 730 or 731. Then dBASE II must 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 skips to the next record, where it performs the same evaluation.

Try using some logical operators with NAMES.DBF. Type the following (the word ALL is optional):

```
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      TX 75220
00008  HOWSER, PETER   678 Dusty Rd.       Chicago     IL 60631
00009  INDERS, PER     321 Sawtelle Blvd.  Tucson      AZ 85702

. display all for name < 'f'
00001  ALAZAR, PAT     123 Crater Rd.      Everett     WA 98206
00002  CLINKER, DUANE  789 Charles Dr.     Los Angeles CA 90036
00003  DESTRY, RALPH   234 Mahogany St.    Deerfield   FL 33441
00004  EDMUNDS, JIM    392 Vicarious Way   Atlanta     GA 30328
00005  EMBRY, ALBERT   345 Sage Avenue     Palo Alto   CA 94303

. display all for address > '400' .and. address <  '700'
00006  FORMAN, ED      456 Boston St.      Dallas      TX 75220
00007  GREEN, TERRY    567 Doheny Dr.      Hollywood   CA 90046
00008  HOWSER, PETER   678 Dusty Rd.       Chicago     IL 60631

. display all for address > '400' .or. address <  '700'
00001  ALAZAR, PAT     123 Crater Rd.      Everett     WA 98206
00002  CLINKER, DUANE  789 Charles Dr.     Los Angeles CA 90036
00003  DESTRY, RALPH   234 Mahogany St.    Deerfield   FL 33441
00004  EDMUNDS, JIM    392 Vicarious Way   Atlanta     GA 30328
00005  EMBRY, ALBERT   345 Sage Avenue     Palo Alto   CA 94303
00006  FORMAN, ED      456 Boston St.      Dallas      TX 75220
00007  GREEN, TERRY    567 Doheny Dr.      Hollywood   CA 90046
00008  HOWSER, PETER   678 Dusty Rd.       Chicago     IL 60631
00009  INDERS, PER     321 Sawtelle Blvd.  Tucson      AZ 85702
00010  JENKINS, TED    210 Park Avenue     New York    NY 10016
```

Notice that the conditional expression in the last DISPLAY command selected all the records instead of a few. To guard against this kind of non-selective "selection," think through your expressions carefully.

THE $ SUBSTRING LOGICAL OPERATOR

The $ substring logical operator has powerful search capabilities. The format is—

<substring> $ <string>

The $ operator searches for the substring on the left within the string on the right. Either or both terms may be string variables or 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"
```

```
. use names
. list for 'ee' $ name
00007  GREEN, TERRY    567 Doheny Dr.    Hollywood     CA90046
. list for '7' $ address
00002  CLINKER, DUANE  789 Charles Dr.   Los Angeles   CA90036
00007  GREEN, TERRY    567 Doheny Dr.    Hollywood     CA90046
00008  HOWSER, PETER   678 Dusty Rd.     Chicago       IL60631
. list for 'ca' $ state
00002  CLINKER, DUANE  789 Charles Dr.   Los Angeles   CA90036
00005  EMBRY, ALBERT   345 Sage Avenue   Palo Alto     CA94303
00007  GREEN, TERRY    567 Doheny Dr.    Hollywood     CA90046

. ? 'oo' $ 'hollywood'
.T.

. go 5
. display
00005  EMBRY, ALBERT   345 Sage Avenue   Palo Alto     CA94303
. ? state $ 'CALIFORNIA'
.T.
```

If you include the state as well as the city in a database's ADDRESS field, you could use the substring function in an expression to select the state you want. For example, to call out names within a specific state, you could type the following:

```
COMMAND FOR 'XX' $ Address
```

where: XX is the abbreviation for the state you want.

STRING OPERATORS

String operators generate string results, concatenating or joining character strings as listed in Exhibit 2c.

Exhibit 2c: String Operators

OPERATOR	MEANING
+	Concatenates strings exactly.
–	Concatenates strings and moves blanks.

Both + and – join two strings. The plus sign joins the strings exactly as they are found. The minus sign moves the trailing blanks in each string to the end of the output string. The blanks are not eliminated, but they do not show up between the strings being joined.

To see how concatenation works, 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.
```

To eliminate trailing blanks, you can use the TRIM function. The format is—

```
STORE TRIM(<variable>) TO <variable>
```

As an example, to eliminate the blanks following the characters of the name, you could have typed STORE TRIM(Name) TO Name.

To eliminate all of the trailing blanks in the example, type STORE TRIM(Name – Address) TO Example.

2.4 CHANGING AN EMPTY DATABASE STRUCTURE (MODIFY)

WARNING: The MODIFY STRUCTURE command destroys the 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. MODIFY destroys any data in the database, so don't use it after you've entered data. Later you'll learn a way to do so, safely.

You'll work with the empty file MONEYOUT.DBF created at the end of Chapter 1. A possible 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
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     Check:Date      C      007
002     Check:Nmbr      C      005
003     Client          C      003
004     JobNumber       N      003
005     Name            C      020
006     Descrip         C      020
007     Amount          N      009     002
008     Bill:Date       C      007
009     Bill:Nmbr       C      007
010     Hours           N      006     002
011     Emp:Nmbr        N      003
**TOTAL**                    00091

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

In MODIFY mode, dBASE II erases the screen and lists the first 16 (or fewer) fields in the database. For this example, use ALT-X to move down one field. Then 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. ALT-W saves the changes to the database structure on disk and resumes normal dBASE II operation (the dot prompt displays). ALT-Q quits MODIFY mode without making the changes. ALT-Q does not destroy the original database, but it is always safest to have a backup file (see the next section).

2.5 DUPLICATING DATABASES AND STRUCTURES (COPY)

You can easily duplicate a file without going back to the operating system. Type the following:

USE Names
COPY TO Temp
USE Temp
DISPLAY STRUCTURE
LIST

```
. use names
. copy to temp
00010 RECORDS COPIED
. use temp
. 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      015
002   ADDRESS     C      020
003   CITY        C      020
004   STATE       C      002
005   ZIP:CODE    C      005
**TOTAL**             00063

. list
00001  ALAZAR, PAT     123 Crater Rd.      Everett      WA 98206
00002  CLINKER, DUANE  789 Charles Dr.     Los Angeles  CA 90036
00003  DESTRY, RALPH   234 Mahogany St.    Deerfield    FL 33441
00004  EDMUNDS, JIM    392 Vicarious Way   Atlanta      GA 30328
00005  EMBRY, ALBERT   345 Sage Avenue     Palo Alto    CA 94303
00006  FORMAN, ED      456 Boston St.      Dallas       TX 75220
00007  GREEN, TERRY    567 Doheny Dr.      Hollywood    CA 90046
00008  HOWSER, PETER   678 Dusty Rd.       Chicago      IL 60631
00009  INDERS, PER     321 Sawtelle Blvd.  Tucson       AZ 85702
00010  JENKINS, TED    210 Park Avenue     New York     NY 10016
```

WARNING: When you COPY to an existing file name, dBASE II writes over the file and destroys the old data.

COPY TO TEMP created a new database called TEMP.DBF, identical to NAMES.DBF, with the same structure and data.

The COPY command can be expanded even further—

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

With an expanded COPY command, you can copy only the structure or just part of the structure to another file. Type the following:

USE Names
COPY STRUCTURE TO Temp
USE Temp
DISPLAY STRUCTURE

```
. use names
. copy structure to temp
. use temp
. display structure
STRUCTURE FOR FILE:  TEMP.DBF
NUMBER OF RECORDS: 00000
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
FLD    NAME      TYPE  WIDTH  DEC
001    NAME       C     015
002    ADDRESS    C     020
003    CITY       C     020
004    STATE      C     002
005    ZIP:CODE   C     005
**TOTAL**             00063
```

You can copy a portion of the structure by listing only the fields you want in the new database. Type—

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

```
. use names
. copy structure to temp fields name, state
. use temp
. display structure
STRUCTURE FOR FILE:  TEMP.DBF
NUMBER OF RECORDS: 00000
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
FLD    NAME      TYPE  WIDTH  DEC
001    NAME       C     015
002    STATE      C     002
**TOTAL**             00018
```

Advanced programmers can use COPY to give their programs access to a database structure. Type—

USE Names
COPY STRUCTURE TO New EXTENDED
USE New
LIST

```
. use names
. copy to new structure extended
00005 RECORDS COPIED
. use new
. display structure
STRUCTURE FOR FILE: NEW.DBF
NUMBER OF RECORDS: 00005
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
FLD        NAME      TYPE   WIDTH  DEC
001    FIELD:NAME     C       010
002    FIELD:TYPE     C       001
003    FIELD:LEN      N       003
004    FIELD:DEC      N       003
**TOTAL**                   00018
. list
00001      NAME             C      15     0
00002      ADDRESS          C      20     0
00003      CITY             C      20     0
00004      STATE            C       2     0
00005      ZIP:CODE         C       5     0
```

The NEW.DBF database records describe the NAMES database structure, and an application program has direct access to this information (see REVIEW.PRG, Appendix A).

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

CREATE <datafile> FROM <structurefile>

2.6 ADDING AND DELETING FIELDS WITH DATA IN THE DATABASE (COPY, USE, APPEND)

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

MODIFY STRUCTURE used alone destroys the data in your database, but you can use it with COPY and APPEND to add and delete fields at will.

The strategy for changing the structure of a filled database is to copy the structure to a temporary file, and then bring the data from the old file into the new structure.

As an example, use the NAMES and ORDERS files to list the orders placed by a given customer. You could easily make the list if the NAMES file had a customer number field to match the ORDERS file. To do so without destroying the records you already have, type the following:

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

Use the full-screen editing features (see Reference Chapter 7) to move down to the first blank field. Type in the changes in the appropriate columns (name is CUSTNMBR, data type is C, length is 9). Now type ALT-W to save the changes and exit to the dBASE II dot prompt.

DISPLAY STRUCTURE and check that it's right. If it's not right, MODIFY STRUCTURE again. Then you can add the data from NAMES by typing—

```
APPEND FROM Names
```

You could also have changed field sizes because 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     015
002    ADDRESS      C     020
003    CITY         C     020
004    STATE        C     002
005    ZIP:CODE     C     005
006    CUSTNMBR     C     009
**TOTAL**               00072
```

Your new file TEMP.DBF should now have the new field you wanted to add and all of the old data. DISPLAY STRUCTURE, then LIST to make sure that the TEMP file is correct.

If the data is correct, you can finish up by typing—

COPY TO Names
USE Names

The COPY command writes over both the old structure and the old data. After displaying and listing the new NAMES file, you can DELETE FILE Temp.

To summarize, use the following sequence to add or delete fields in a database:

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

```
. use names
. copy structure to temp
. use temp
. modify structure
MODIFY ERASES ALL DATA RECORDS...PROCEED? (Y/N) y

. append from names
00010 RECORDS ADDED

. copy to names
00010 RECORDS COPIED

. use names
. display structure
STRUCTURE FOR FILE: NAMES.DBF
NUMBER OF RECORDS: 00010
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
FLD     NAME    TYPE  WIDTH  DEC
001   NAME       C     015
002   ADDRESS    C     020
003   CITY       C     020
004   STATE      C     002
005   ZIP:CODE   C     005
006   CUSTNMBR   C     009
**TOTAL**            00072
```

CP/M-86, MS-DOS, AND OTHER NON-dBASE II DATA FILES (COPY, APPEND) 2.7

You can change dBASE II information into a form compatible with other processors and systems (such as BASIC, Pascal, FORTRAN, PL/1). dBASE II can also read data files created with these processors.

With CP/M-86 and MS-DOS, the System 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 word processing, 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-86 or MS-DOS.

The System Data Format also allows dBASE II to work with data from CP/M-86 or MS-DOS files. However, the data must match the structure of the database that uses it.

Adding data to an existing file from a system file takes only seconds. For example, if you use a word processor to create a file called NEWDATA.TXT, you can add it to the NAMES.DBF file with the APPEND SDF command. The spacing of the data must match the structure of the database.

The NEWDATA.TXT file might contain the following information:

```
FREITAG, JEAN    54 Munchkin Ave.   Houston   TX77006
GOULD, NICOLE    73 Radnor Way      Radnor    PA19089
PETERS, ALICE    676 Wacker Dr.     Chicago   IL60606
GREEN, FRANK     1 Spicer Ave.      Tampa     FL33622

     (15)              (20)           (20)    (2) (5)
```

Then you can add the data from NEWDATA to NAMES by typing—

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

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

The procedure is similar for non-dBASE II files that use different delimiters. One 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>. For example, 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>]]
```

You can make both APPEND and COPY selective by using a conditional expression, and you can specify the scope of COPY as for other dBASE II commands.

With the APPEND command, any fields used in the FOR expression must exist in the database where the data is being transferred.

RENAMING DATABASE FIELDS (COPY, APPEND) 2.8

APPEND transfers data from one file to another for matching fields. If a field name in the FROM file is not in the file in USE, the data is not transferred from that field.

However, the full form does allow you to transfer only data, and you can use this feature to rename the fields in a database. If you want to rename the CUSTNMBR field to CUSTCODE in NAMES.DBF, 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 CUST-CODE. Don't forget to change the name of the CUSTNMBR field in your ORDERS database so that the fields match.

```
. use names
. copy to temp sdf
00014 RECORDS COPIED
. modify structure
MODIFY ERASES ALL DATA RECORDS...PROCEED? (Y/N) Y

. append from temp.txt sdf
00014 RECORDS ADDED
```

Data in a .TXT file created using the SDF (or DELIMITED) option is kept in columns spaced like the fields were in the original file. While you can edit a .TXT file with your word processor, this can be dangerous. Do not change field positions or sizes. dBASE II saves data 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 Chapter 1).

2.9 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. 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 in the command format.

The data named in the REPLACE command can be specific new information (including blanks), or it can 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 the replacement conditional by specifying your conditions in a FOR phrase.

To show how this works, add some data to both the NAMES and ORDERS database files.

First, USE NAMES then type EDIT 1. Now enter 1001 in the CUSTCODE field, using the full-screen editing features to get into position. Use ALT-C to move to the next record. Customer codes should be entered as four-digit numbers, with the record number as the last two digits (1001, 1002, 1003, and so on).

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

Then enter the following commands:

USE Orders
REPLACE All Amount WITH Qty*Price
LIST

```
. use orders
. replace all amount with qty*price
00004 REPLACEMENT(S)
. list
00001       1012       38567        5       0.83       4.15
00002       1003       83899       34       0.12       4.08
00003       1009       12829        7       0.17       1.19
00004       1012       73833       23       1.47      33.81
```

You'll also find REPLACE useful in command files to fill in a blank record that you have appended to a file. You can use data from memory variables in your program to fill in the blank fields.

You can make changes to a few fields in a large number of records 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; you can list several field names separated by commas. This command finds the first record that meets the conditions in the FOR 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> or Return. 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.

```
. use names
. change field custcode

RECORD: 00001

CUSTCODE:
CHANGE? <sp>
TO      1001

CUSTCODE: 1001
CHANGE?<Return>

RECORD: 00002

CUSTCODE:
CHANGE?
```

2.10 ORGANIZING YOUR DATABASES (SORT, INDEX)

Frequently, data records are entered in no particular order, as in the NAMES database. Later you may want to reorganize your data without re-entering all the records. With dBASE II you can organize a database by SORTing and INDEXing it.

Files can be SORTed in ascending or descending order. The full command is—

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

The fieldname is the key on which the file is sorted and may be character or numeric (not logical). SORT works in ascending order, but you can specify 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 moves only as many records as it must.

To SORT the NAMES file so that the customers are in alphabetical order, type—

```
USE Names
SORT ON Name TO Temp
USE Temp
LIST OFF
COPY TO Names
```

```
. use names
. sort on name to temp
SORT COMPLETE
. use temp
. keep list off
ALAZAR, PAT       123 Crater Rd.      Everett     WA 982061001
CLINKER, DUANE    789 Charles Dr.     Los Angeles CA 900361002
DESTRY, RALPH     234 Mahogany St.    Deerfield   FL 334411003
EDMUNDS, JIM      392 Vicarious Way   Atlanta     GA 303281004
EMBRY, ALBERT     345 Sage Avenue     Palo Alto   CA 943031005
FORMAN, ED        456 Boston St.      Dallas      TX 752201006
FREITAG, JEAN     854 Munchkin Ave.   Houston     TX 770061011
GOULD, NICOLE     73 Radnor Way       Radnor      PA 190891012
GREEN, FRANK      441 Spicer Ave.     Tampa       FL 336221014
GREEN, TERRY      567 Doheny Dr.      Hollywood   CA 900441007
HOWSER, PETER     678 Dusty Rd.       Chicago     IL 606311008
INDERS, PER       321 Sawtelle Blvd.  Tucson      AZ 857021009
JENKINS, TED      210 Park Avenue     New York    NY 100161010
PETERS, ALICE     676 Wacker Dr.      Chicago     IL 606061013
. copy to names
00014 RECORDS COPIED
```

Do not SORT a database to itself. A powerline "glitch" could destroy your entire database. Instead, SORT to a temporary file, then COPY it back to the original file after you've confirmed the data.

A database can also be INDEXed so that it appears to be sorted. INDEXed files allow you to locate records quickly (typically within two seconds with floppy diskettes).

The form of the INDEX command is—

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

The INDEX command 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 the customer database by ZIP code, type—

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

You can INDEX a database on multiple keys. For example, you could index the NAMES database on three keys by typing—

```
INDEX ON Name + CustCode + State TO Compound
```

Numeric fields must be converted to character type by using the STR function (described in more detail later). If CUSTCODE were a numeric field with 5 positions and 2 decimal places, the conversion would be performed like this:

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

To take advantage of the speed built into an INDEX file, you must specify it as part of the USE command—

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

To keep your INDEX files up to date, you must USE them with the database file whenever you work with the database. You can USE up to seven INDEX files at one time with a database.

With the INDEX file in USE, any records that you APPEND will automatically be indexed, except for appended blanks. After the APPEND BLANK command, data that you add with the REPLACE command can be added to the index by using the command INDEX with no qualifiers.

Changes made to key fields when you EDIT, REPLACE or PACK the database are reflected in the index file in USE. Other index files for the same database will not be correct.

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

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

FINDING THE INFORMATION YOU WANT (FIND, LOCATE) 2.11

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).

Simply type—

```
FIND <character string>
```

where: the character string (with no quote marks) is all or part of the contents of a field. The string can be as short as you like, but should be long enough to make it unique. For example, the letters "th" occur 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

```
.use names index zips

.find 10
.display
00013  JENKINS, TED     210 Park Avenue   New York      NY 10016 1010

.find 9
.display
00002  CLINKER, DUANE   789 Charles Dr.   Los Angeles   CA 90036 1002

.display next 3
00002  CLINKER, DUANE   789 Charles Dr.   Los Angeles   CA 90036 1002
00010  GREEN, TERRY     567 Doheny Dr.    Hollywood     CA 90044 1007
00005  EMBRY, ALBERT    345 Sage Avenue   Palo Alto     CA 94303 1005
```

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 key you are looking for, dBASE II displays NO FIND.

FIND can also be used with files INDEXed on multiple keys. A disadvantage of a compound key (which may not be a disadvantage in your application) is that you must use the keys inclusively from the left when you access the data. That is, you can use the FIND command and just the NAME, or the NAME and CUSTCODE, or all three fields, but not the STATE or CUSTCODE alone. To do that, you would either have to use the LOCATE command (described next), or have another file indexed on the STATE or CUSTCODE field as the primary key.

When looking for specific kinds of data, use—

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

Use the LOCATE command when you are looking for specific data in a file that is not indexed on the key you are interested in (for example, you want to LOCATE a state in a file that is indexed on zip codes).

If you want to search the entire database between your pointer and the end of the file, you do not have to specify ALL, or position the pointer at the start of the file (GO TOP) before the LOCATE command. If you are looking for data in a character field, enclose the data in single quotes.

For an example of LOCATE, 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 displays END OF LOCATE or END OF FILE ENCOUNTERED.

```
. use names
. locate for name='gou'
RECORD:00008
. display
00008  GOULD, NICOLE   73 Radnor Way   Radnor  PA 190891012
. locate for zip:code>'8' .and. name < 'G'
RECORD:00001
. display name, zip:code
00001  ALAZAR, PAT      98206
. continue
RECORD:00003
. continue
RECORD:00005
. continue
END OF FILE ENCOUNTERED
```

2.12 SUMMARIZING DATA (REPORT)

In most applications you will want data summaries of records that meet certain specifications. With the REPORT command you can format and print out tabulated data quickly and easily.

First select the database you want the report from, then create the custom format by typing—

USE <database>
REPORT

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

The NAMES and ORDERS databases used as examples so far don't have enough data in them to show you how powerful dBASE II can be, so now you will use MONEYOUT.DBF and other databases that are part of an existing business system. (The entire system is given in Appendix A, including database structures and the command files that run it.)

For some realism in your practice sessions, you can create a database that you would actually use in your business. Enter data in it, then substitute it for MONEYOUT in the examples.

```
. use moneyout
. report
ENTER REPORT FORM NAME: JobCosts
ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE, W=PAGE WIDTH
PAGE HEADING? (Y/N) Y
ENTER PAGE HEADING: COST SUMMARY
DOUBLE SPACE REPORT? (Y/N) n
ARE TOTALS REQUIRED? (Y/N) y
SUBTOTALS IN REPORT? (Y/N) n
COL   WIDTH,CONTENTS
001   10,Check:Date
ENTER HEADING: DATE
002   22,Name
ENTER HEADING: SUPPLIER
003   22,Descrip
ENTER HEADING: DESCRIPTION
004   12,Amount
ENTER HEADING: AMOUNT
ARE TOTALS REQUIRED? (Y/N) y
005   <Return>

      PAGE NO. 00001
```

After you define all the contents of the report, hit <enter> or Return when prompted with the next field number. dBASE II immediately starts the report to show you what you have specified, and goes 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. You can use it again without having to repeat the dialog. The full form of the command is—

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

where: formname is the name of the REPORT form file (extension FRM). If the form file does not exist, dBASE II creates it. If no scope is specified, the scope defaults to ALL.

With an existing form file you can make a report on a selected part of the data by including a FOR phrase. For example, by typing—

REPORT FORM JobCosts FOR Job:Nmbr= 770

you 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 heading in your report by typing—

SET HEADING TO <character string>

The string can be up to 60 characters and spaces, but no quote marks.

You can prepare the entire report as a hardcopy by adding TO PRINT at the end of the command.

The 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) 2.13

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 <expression>]
  [TO <memory variable>]
```

The COUNT 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.

The result of the count can be stored in a memory variable, which is created when the command is executed if it did not already exist.

To get totals, use—

```
SUM <field(s)> [<scope>][FOR <expression>]
  [TO <memory variable(s)>]
```

You can list up to five numeric fields to sum in the database in USE. If more than one field is to be totalled, separate the field names by commas. You can limit the records being totalled by using the scope and/or conditional expressions after the FOR (such as 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
```

2.14 SUMMING DATA AND ELIMINATING DETAILS (TOTAL)

TOTAL works like the subtotal capability in the REPORT command, except that the results are placed in a database rather than being printed out. The form of the TOTAL command is—

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

The database containing the data being TOTALed must be presorted or indexed on the key used in the TOTAL command.

The TOTAL command is useful for eliminating detail and providing summaries. The screen shows what happens with MONEYOUT—

```
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.

TOTAL transfers all the fields if the database named did not exist, but uses the structure of an existing database. One problem with the new database is that only two of the fields contain useful information. This can be handled with one more command line. You can limit the fields in the new database by creating it first, before you enter the TOTAL command—

COPY TO Temp FIELDS Job:Nmbr, Amount

Now when you TOTAL to TEMP, the new database contains only the job numbers and totals. Try it with your database.

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

```
. use moneyout
. index on job:nmbr to jobs
00093 RECORDS INDEXED
. use moneyout index jobs
. total on job:nmbr to temp fields amount for job:nmbr > 699;
                                           .and.  job:nmbr < 800
00025 RECORDS COPIED
. use temp
. list
00011   810129   3148   SML   779   138.00   LETTER FONT   TYPE
810129   2633          0.00   0
00012   810129   3152   SML   782    59.49   MAGIC TOUCH   BACKGROUND
TONE         810129   429         0.00    0
00013   810129   3148   SMM   784    46.00   LETTER FONT   TYPE
810129   3003          0.00   0
00014   810129   3148   DOC   786   251.00   LETTER FONT   TYPE
810129   2764          0.00   0
                             (Partial listing)
```

3

CREATING AND WORKING WITH COMMAND FILES

Once you understand how to write expressions (covered in Chapter 2), you are very close to being able to write programs. There are four basic programming structures you can use to get a computer to do what you want—

- ▶ Sequence
- ▶ Choice/Decision
- ▶ Repetition
- ▶ Procedures

You've already seen that dBASE II processes your commands sequentially in the order you give them. This section explains how you make choices (IF...ELSE), how you can make the computer repeat a sequence of commands (DO WHILE..), and how to use subfiles of commands (procedures).

Then examples will 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) 3.1

The commands introduced so far can accomplish a great deal, yet they only scratch the surface of the capabilities of dBASE II. Its full power emerges when you set up command files. Then the commands you enter once can be repeated over and over.

When you create a command file, you are programming the computer. Programming with English-like commands of dBASE II is simple. Also, dBASE II is a relational database management system in which you work with increments of data and information, rather than bits and bytes.

A dBASE II command file, like a computer program, lists commands that you want performed; command files must have a .PRG file extension. When you run a command file, dBASE II starts at the top of the list and processes the commands one at a time through the end.

Other computer languages operate the same way. In BASIC the sequence is visible because each program line is numbered. In other languages (dBASE II among them), the sequence is implied and the computer processes the first line on the page, then the second line, and so on. Some languages use separators such as colons between command statements; dBASE II uses the carriage return that terminates each command line.

The only time the sequence is not followed is when the computer is specifically told to do something else. Usually, this "branching" is based on stated conditions; the computer must make a decision based on expressions or conditions that you have set up in the command file.

For practice, create a command file.

You use the dBASE II command MODIFY COMMAND to create command files. You can also create command files with a text editor (such as ED in CP/M-86 or EDLIN in MS-DOS) or with a word processor such as WordStar in program mode, but using dBASE II is easy and fast.

Type—

```
MODIFY COMMAND Test
```

This command puts dBASE II into MODIFY COMMAND mode. The screen clears for you to write a command file (a series of commands) using the full-screen editing features (described in Chapter 2 and in Reference Chapter 7).

Enter the following short program:

```
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.'
```

Save the TEST.PRG command file by entering ALT-W to store the file and return to the dot prompt. Then run the TEST program by typing—

```
DO Test
```

OOPS! The TEST.PRG command file crashes if you entered it exactly as shown here. To correct the error (ZipCode should be Zip:Code), type MODIFY COMMAND Test again. dBASE II brings up the command file and you can make the change using full-screen operations.

Once you are writing command files of your own, you'll find the built-in editor (MODIFY COMMAND mode) is one of the most convenient features of dBASE II..

The command file created here is trivial, but it shows you how you can perform a sequence of commands from a file with a single command. Running command files is similar to using .COM files (MS-DOS) or .CMD files (CP/M-86) in your operating system.

To run a dBASE II command file from the operating system (from the A> or B> prompt), type—

```
dBASE <filename>
```

where: filename is the name of a command file, including the .PRG extension.

MAKING CHOICES (IF..ELSE) 3.2

Choices and decisions are made in dBASE II with IF..ELSE..ENDIF. You use these words much as you do in ordinary English: IF I'm hungry, I'll eat, (OR) ELSE I won't. With the dBASE II program, you use the identical construction, but you must use words that the program understands.

SIMPLE DECISIONS

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
```

3

The condition can be a series of expressions (to a maximum of 254 characters) that can be logically evaluated as true or false. Use the logical operators (see Chapter 2) to tie them together. Using the MONEYOUT file, the following decision could be set up:

```
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, dBASE II performs the commands listed between the IF and the ENDIF (in sequence), then goes to the statement following the ENDIF. If the conditions are not met, dBASE II 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
```

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

MULTIPLE CHOICE

Frequently, you must choose from a list of alternatives. An example might be the use of a screen menu to select one of several different procedures that you want to perform.

For multiple choice, use the IF..ELSE..IF construction. This is the same IF..ELSE described earlier, but you use it in several levels (called nesting), as shown here:

```
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 far as needed to choose the one correct set of commands from the list of alternatives. Nested IFs are used frequently in the working accounting system, Appendix A.

Remember that each IF must have a corresponding ENDIF or your program will bomb. Notice that the ENDIFs in the example are followed by numbers or labels. dBASE II does not read the rest of the line after an ENDIF, so you can add identification labels like the ones in the example to help remember which IF the ENDIF matches.

3.3 REPEATING A PROCESS (DO WHILE..)

Repetition is one of the major advantages of a computer. It can perform a task over and over without getting bored or making mistakes. Repetition 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. Remember that these commands must eventually change the conditions, or the loop will continue forever.

When you know how many times you want the process repeated, you can use the IF..ELSE..ENDIF structure like this:

```
STORE 1 TO Index                 * Start counter at 1
DO WHILE Index < 11              * Process 10 records
   IF Item 303 ' '               * 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.PRG
   STORE Index+1 TO Index        * Increase counter by 1
ENDDO ten times
```

In this example, if there is data in the ITEM field, dBASE II performs the instructions in the command file PROCESSA.PRG, 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, dBASE II proceeds through the DO WHILE instructions again. When the counter passes 10, dBASE II skips the loop and performs the next instruction after the ENDDO.

The LOOP instruction is used to stop a sequence and cause dBASE II to go back to the start of the 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 dBASE II back to the DO WHILE Index < 11. The record with the blank is not counted, since the program bypasses the command line that adds 1 to the counter.

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

PROCEDURES (SUBSIDIARY COMMAND FILES) 3.4

Being able to create standard procedures that can be used in many programs simplifies computer programming. In BASIC, you can store standard procedures in command files that can be called by other command files.

PROCESSA is another command file (with a .PRG extension). 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, you can call out additional procedures which can themselves call other files. Up to 16 command files can be open at a time, so if a file is in USE, dBASE II can open up to 15 other files. Some dBASE II commands use additional files in their operation (REPORT, INSERT, COPY, SAVE, RESTORE and PACK use one additional file; SORT uses two additional files).

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). Ending a command file with the RETURN command is not strictly necessary, but it is good programming practice.

Notice that the command lines are indented in the examples; each IF is indented the same amount as the ELSE and/or ENDIF that matches it. You don't have to indent your command files, but you'll find that indenting keeps the program logic clear, especially when you have nested structures within other structures. Using all uppercase for the dBASE II commands, and both upper- and lowercase for the variables helps, too.

DATA INPUT DURING A RUN (WAIT, INPUT, ACCEPT) 3.5

For many applications, the command files have to obtain additional data from the operator, rather than using only what is in the databases. Your 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.

You can use the WAIT, INPUT, or ACCEPT commands to get data from the operator. The action of each command suits it for slightly different situations.

The form of the WAIT command is—

WAIT [TO memory variable]

WAIT halts command file processing and displays a WAITING prompt until the operator inputs a single character from the keyboard. 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 (such as <enter>, a control character, and so on), a blank is entered into the variable.

The form of the INPUT command is—

INPUT [' prompt '] TO memory variable

INPUT accepts any data type from the keyboard to a named memory variable, creating that variable if it did not exist. If you include the optional prompting message (in single or double quotes, but both delimiters the same), it appears on the screen 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, as always, must be entered in quotes or square brackets.

The form of the ACCEPT command is—

ACCEPT [' prompt '] TO memory variable

The ACCEPT command accepts character data without the need for delimiters. ACCEPT is 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 because it does not require quotes. ACCEPT 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, and can be used like ACCEPT.

3.6 FORMATTING SCREEN AND PRINTER DISPLAYS (@..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 appear just below the last line already on the screen.

Another command—the @ command—lets you position your prompts and get your data from any position you select on the screen—

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

The @ command positions the prompt (entered in quotes or square brackets) at the screen coordinates you specify. The coordinates are the row (or line) and column on the CRT, with 0,0 being the upper left "home" position. If you specify 9,34 as the coordinates, the prompt starts on row 10 in column 35.

The SAY part of the command is optional; it tells dBASE II what to say (or display) at the coordinates you name. Without a SAY phrase, the @ command erases the line (or portion of the line) on the screen. To see how @ with SAY displays a prompt, and how @ without SAY erases the line beginning at the column you name, 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 with SAY can show the value of an expression with one or more variables. The form is—

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

To see how @ evaluates and displays an expression (doing string concatenation in this example), type the following in dBASE II:

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

The @ command can be expanded further to display the value of variables you are using (such as memory variables or field names in a database) at whatever screen position you specify. The GET phrase displays the current value of a variable.

Using the @ command, you can display a label (or prompt) with the SAY phrase, and fill in specific data with the GET phrase. The format is—

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

To see how the GET phrase works, type the following:

```
ERASE
USE Names
@ 5,0 SAY 'Customer Name' GET Name
@ 10,0 SAY "Home Address" GET Address
@ 15, 0 SAY 'State' GET State
@ 15,10 GET Zip:Code
```

Stay in dBASE II. There's more to come.

Notice that the preceding example displayed the values of the variables (with and without prompts) at different places on the screen.

After you format the screen with @..SAY..GET, you (or an operator) can input or correct data in the variables on the screen. First, type—

```
READ
```

The cursor positions itself on the first field you entered. You can type in new data, or leave it the way it was (with the variable you named in the GET phrase) by hitting <enter> or Return. When you leave this field, the cursor moves 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. However, it is much more powerful than either because it allows you to enter many variables.

With the formatting facility of the @ command, you can design input forms so that the screen looks like the paper forms used before. Any new data entered will be stored in the database.

A database may have a dozen or more 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.

The NAMES file is not the best example (see the accounting system in Appendix A) but it shows how to selectively get data into a database with a large structure.

For more practice with command files, create a file called TRIAL.PRG with the following command 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 the operating system, type dBASE Trial (type DO Trial at the dot prompt). Enter data into several records as TRIAL.PRG runs. After you finish, LIST the file to see what you've added.

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

You can customize the screen by placing prompts and variable input fields wherever you want them. However, you must use the ERASE or CLEAR GETS command after every 64 GETs. Use the CLEAR command if you do not want to change the screen.

3.7 WORKING WITH MULTIPLE DATABASES (SELECT PRIMARY/SECONDARY)

To begin working with a database file, you type USE <filename> to tell dBASE II which file you're interested in. Then you proceed to enter data, edit, and so forth. If you want to work on a different database, USE NewFile. dBASE II closes the first file and opens the second one. You can use any number of files this way, both from your terminal and in command files, but you can USE only one file at a time. You can also close a file without opening a new one by typing USE with no filename.

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 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 in the PRIMARY area when you first USE a file. 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 working 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; use the P. prefix for variables you need from the PRIMARY area.

As an example, the SELECT SECONDARY command is used in the NAMETEST.PRG file in the accounting system in Appendix A. 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.PRG, DEPTRANS.PRG, and PAYROLL.PRG files.

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

SOME USEFUL COMMANDS AND FUNCTIONS 3.8

- 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 ALT-B to scroll right. Use ALT-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 CP/M-86 operating system bit map. Please read the detailed description in the Reference section before using it.
- NOTE or * allows you to insert comments in a command file; the comments are not displayed when the command file is executed. You can insert 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 you to store comments in a command file; they are 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 directory. Do NOT try to rename a file in USE.
- You can use the ? command to call out the following functions:

  # is the record number function; it provides the value of the current record number.

  * is the deleted record function; it 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 otherwise.

3.9 ABOUT WRITING COMMAND FILES

Briefly, here's the approach to use to begin your own programming:

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 wnat to do, describe the details in modified English. Many texts call it "pseudocode." All this means is that you use English terms that are somewhat similar to the instructions that the computer program understands.

You might write your outline like this:

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

Add a bit more detail, and 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 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 outline 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 <= '801201' TO PRINT
GOTO TOP                * Go to the first record
DO WHILE .NOT. EOF      * Repeat for the entire file
    IF Check:Nmbr+'  '  * If 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 "top-down, step-wise refinement" can be applied to this procedure, meaning "Start at the top, then divide and conquer."

At this stage in our example, we haven't done the SUMMARY.FRM file or the WRITECHECK.PRG and UPDATE.PRG files, but it doesn't matter. 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.

You can still test a partial program like this by using what programmers call stubs. For the command files that you've named in the program, enter three items: a message that lets you know the program reached it, WAIT, and RETURN. When dBASE II goes to these procedure files,the message will display, then dBASE II returns and continues with the rest of the program after you hit any key.

USING FUNCTIONS AND CREATING FORMATS

By now you should be writing command files that can perform useful work for you. This chapter introduces more functions and commands, and explains how to display and print out your data in exactly the format you want.

FUNCTIONS 4.1

Functions are special-purpose operations that you can use in dBASE II expressions; functions perform tasks that are difficult or impossible using regular arithmetic, logical and string operations. dBASE II functions fall into these same three categories, based on the results they generate.

You call up a function by typing ? followed by a space and the function itself; the function consists of a key phrase or symbol plus the variable or expression you want the function to evaluate. Functions can be called from the keyboard or within command files. You must include parentheses around the variable, string, or expression in the function, as shown in Exhibit 4a.

Don't worry about memorizing the functions now. Scan the descriptions so that you know where to look when you need one of them in a command file. See Reference Chapter 3 for more about functions.

Exhibit 4a: dBASE II Functions

!(<variable/string>)

Lower- to uppercase function—changes all the characters from 'a'..'z' in a string or string variable to uppercase. Any other characters in the string are unaffected. The ! function is used in the accounting system (Appendix A) to put keyboard inputs in a standard form in the files; a standard form makes searching for data simpler, since all the data is stored in uppercase, regardless of how it was entered.

$(<exp/variable/string>, <start>, <length>)

Substring function—selects characters from a string or character variable, starting at the specified position and continuing for the specified length.

As an example, if you have a variable called DATE whose value is 810823, the function $(Date,5,2) gives you 23. To convert these numerals to a number, you can use VAL($(Date,5,2)).

An example of the $ function is in the DATETEST.PRG file in Appendix A, 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 Chapter 2.

@(<variable1/string1>, <variable2/string2>)

Substring search function. You might think of this as "Where is string1 AT in string2?" The @ function 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.

&

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 variable (must be character data). The & is useful when you must use complex expressions 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.PRG file in Appendix A, the & is used to get the name of the required database:

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

The & can also be used as an abbreviation of a command. If you STORE 'Delete Record' to D, the command &D 5 then deletes record 5 when the program runs.

If the Macro symbol is not followed by a valid string variable, it is skipped.

See Reference Chapter 4 for more information on Macro substitution.

CHR (<number>)

The character function—yields the ASCII character equivalent of the number. For example, ? CHR(27) + "E" clears the screen, CHR(27) + "p" produces reverse video and ? CHR(27) + "q" cancels it.

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.

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

File function—verifies the existence of a file. It yields a True value if the file exists on the disk, False if it does not. For 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. For example, FILE("B:" + Database) tells you whether the file name stored in the memory variable Database is on drive B (see REPORTMENU.PRG in Appendix A).

INT(<variable/exp>)

Integer function—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. For example, INT(123.86) yields 123, while INT(-123.86) yields -123.

For variables and expressions, the expression is first evaluated, then an integer is formed from the results. A call to a variable yields a truncated integer formed from the current value of that variable. For example, if you are on record 7 of MONEYOUT.DBF, a call to INT(Amount) produces 2333, the integer part of $2,333.75.

The integer function rounds a value to any number of decimal places. INT(value*10 + 0.5)/10 rounds 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 10s. For 3 places, use 1000, and so on.

LEN(<variable/string>)

String length function—tells you how many characters are in the string you name. LEN can be useful when the program must 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).

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

Integer to string function—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.

The STR function is used in Appendix A to simplify displays. Numbers are converted to strings, then concatenated with (joined to) other strings of characters for displays.

TRIM

Trim function—eliminates the trailing blanks in the contents of a string variable. This is done by typing:

```
STORE TRIM (<variable>) TO <newvariable>
```

TYPE(<exp>)

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

VAL(<variable/string/substring>)

String to integer function—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. With MONEYOUT.DBF, VAL(Job:Nmbr) yields the numeric value of the contents of the job number field, since all Job Numbers are stored as characters. You can also use VAL with the substring operator: VAL($(<string>)).

4.2 CHANGING THE WORKING ENVIRONMENT

You can change the dBASE II working environment (the way dBASE II interacts with your system setup) using simple SET commands from the keyboard or within command files. Examples of parameters you can change are—

- CRT screen controls (such as output of @ command formats to the screen, full-screen operations, echo of output to the screen, display of colons around formatted variables, and dual intensity display);
- Printer settings (such as left margin, and form feeds and special headings for the REPORT command);
- Advanced programming features (such as linking PRIMARY and SECONDARY files, and creating an output file from CRT display).

You can adjust these parameters back and forth "on the fly," or set them up once at the beginning of your command file. In many applications, the defaults will be just what you need.

You use the SET command to change parameters in your command files or in interactive commands. All the forms of the SET command are listed and described in Exhibit 12a of the Reference.

MERGING RECORDS FROM TWO DATABASES (UPDATE) 4.3

You can transfer data from one database file to another with the following command:

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

You must presort both databases on the key field before the UPDATE.

dBASE II rewinds both files to the beginning, then compares key fields. If they are identical, data from the FROM database 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.

The UPDATE command can be used to keep inventory current. In the accounting system in Appendix A, it is used in PAYROLL.PRG and CHECK-STUB.PRG. It's useful and worth experimenting with.

MERGING ENTIRE DATABASES (JOIN) 4.4

JOIN is one of the most powerful commands in dBASE II. It combines 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> FOR <expression> [FIELD <list>]
```

JOIN 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 the newfile. This process is repeated until all records from the files have been compared.

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 the next example.)

To use the command, enter 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 sequence 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.)

4.5 FULL-SCREEN EDITING AND FORMATTING (@..SAY..GET..PICTURE)

dBASE II has powerful formatting commands that position information precisely where you want it. You saw this formatting facility in action in Chapter 3 when you were introduced to the @ command—

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

The @ command positions prompts and variables (and their values) at the specified location on the screen. When you list a series of @ commands, and then follow them with READ, you can control the format of the entire screen.

To refresh your memory, you might create and run the following command file fragment:

```
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>]
```

followed by a READ in the command file. This command form displays only the colons marking the field length for the variable.

With SET SCREEN ON, you do not have to enter line numbers in numerical order. It's good practice, however, always to enter them in order since they must be in order for PRINT formatting.

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

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

During a READ, you fill in the optional PICTURE phrase using the format symbols listed in Exhibit 9a of the Reference.

For example, 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, because the format characters are 9s (see Reference Chapter 9).

4.6 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. That is, the lines must be in increasing order (print line 7 before line 9, and so on). On any given line, the columns must be in order (print column 15 before column 63, and so on).

As in the SCREEN mode, the GET phrase can be used to output the current value of a variable that you name, the result of an expression, or a literal string prompt message.

The optional USING phrase specifies which characters are printed as well as where they appear on the page. The symbols used in USING phrases are listed in Exhibit 9a of the Reference.

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

SETTING UP AND PRINTING A FORM

To set up a form, use measurements based on your printer spacing (lines per inch vertically and characters per inch horizontally).

The "Outgoing Cash Menu" used in an earlier command file might have another selection item called "4 = Write checks". The next example shows how to do part of the WRITECHECK command file.

To start with, you must input the date. The following command lines accept the date to a variable called MDATE, and check 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)) <> 83
      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 preceding commands set the value of MDATE to eight 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 = 83. dBASE II performs these three steps:

1. 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).

2. The VAL function converts this to an integer.

3. 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 the 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
```

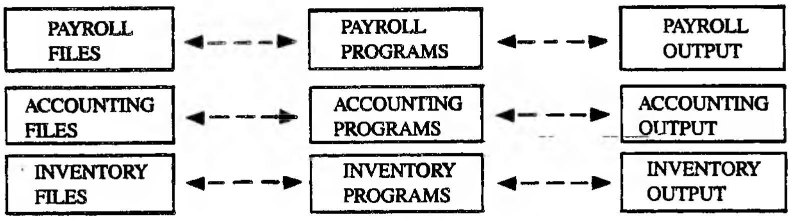
You can check this out on your screen before you print it, then switch from SCREEN to PRINT modes with the SET command. The values for the variables are provided elsewhere in your command file.

Longer forms are no problem—simply start numbering over again every 24 lines (use 0 to 23).

DATABASE BASICS

A database management system (DBMS) like dBASE II is considerably different from a simple file handling system. File handling systems are usually configured like the schematic diagram in Exhibit 5a.

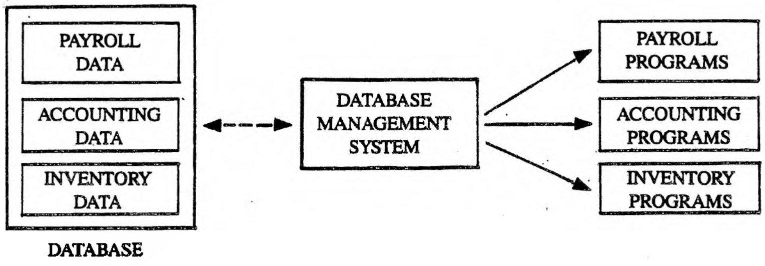
Exhibit 5a: A File Handling System



In a system like the one outlined in Exhibit 5a, 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, you would have to write a new program and it wouldn't necessarily work: data might 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.

In contrast to a file handling system, 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 Exhibit 5b.

Exhibit 5b: A Database Management System



Data is monitored and manipulated by the DBMS, not the individual applications programs. Every applications system has access to all the data. In a file handling system, this kind of access 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, you would have to write a new program and set up new files. Using a DBMS, you write a new access program, but you do not have to restructure the data because the DBMS takes care of that task.

In a file handling system, if you add a new kind of data to a record (for example, salary history in a personnel file) you have to modify all your file handling programs. With a DBMS, however, additions and changes have no effect on the programs that don't need to use the new information; unaffected programs don't see it and don't know that it's there.

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

In a hierarchical system, the relationships between the data elements are maintained with sets, linked lists, and pointers telling the system where to go next. Hierarchical systems tend to become extremely complex and difficult to maintain. 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; the relation between data elements can be represented in a two-dimensional table like the one in Exhibit 5c.

Exhibit 5c: Data in a Relational Database

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 Freight, Inc.	Shipping	97.00	SPT-233

Each row across Exhibit 5c represents an entry in the database; in a DBMS, each row is a record. Each column represents a single item of data for a record; in a DBMS a column is 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 the same type. Each record (row) is unique, and the order of records doesn't matter. In real databases, records don't get any more complicated than the ones in Exhibit 5c, but they do get larger.

DATABASE ORGANIZATION (SORT, INDEX) 5.1

Once you've got your database set up, you'll want to access your data in an ordered manner. With some databases, the order in which you enter the data is the order in which you want to get your information out. In most cases, however, you'll want the information you extract to be organized differently.

With dBASE II you can organize data using the SORT command or the INDEX command (see Chapter 2).

The SORT command moves entire records around to set up a new database arranged in ascending or descending order on any field that you specify (name, ZIP code, state, and so on). The field on which you SORT the database is called the key.

One drawback of sorting is that you may want to access the database on one field for one application, and on another field for a different application. Another drawback is that any new records you add are not in the SORT order; if you want to maintain the order, you must SORT the database again every time you enter data.

Also, finding data in a sorted database is relatively slow, since dBASE II must search the database sequentially.

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

With an indexed database, only the keys are organized, with pointers to the record to which they belong. dBASE II uses a structure called B-trees for indexes. A B-tree is similar to a binary tree, but uses storage more efficiently and is a great deal faster.

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.

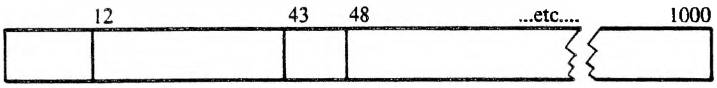
New entries to a database are automatically added to the index file(s) being used.

Another advantage of indexed databases is the rapid location of data that you are interested in. A FIND command (described in Chapter 2) typically takes 2 seconds with a medium to large indexed database.

dBASE II limits you to 65,535 records per file, but with the memory and mass storage capabilities of your computer, the record capacity is really no limitation at all.

A dBASE II record can be as large as 32 fields and 1000 characters long (whichever comes first). The character size of a record is illustrated in Exhibit 5d.

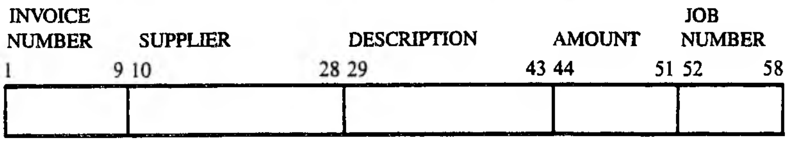
Exhibit 5d: Capacity of a Record in Characters



You can think of the capacity of a record as a 1000 character-long strip that you can segment any way you want (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 maximum per field). Or you can have one record only one character (and one field) long. Or anything in between.

In Exhibit 5c, each record has five fields and the total record length is 58 characters, as illustrated in Exhibit 5e.

Exhibit 5e: Character Size of a Sample Record



DATA TYPES

Each field must contain a single type of data, and in dBASE II data types are—

- Character: all the printable ASCII characters, including the integers, symbols and spaces.
- Numeric: positive and negative numbers as large as 1.8×10^{63} and as small as 1.0×10^{-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.

5

FIELD NAMES

Each data field has a name—the name can be up to 10 characters (no spaces) long, must start with a letter, and can include digits and an embedded colon.

Here are some examples of valid and invalid field names:

A	(Valid)
A123456789	(Valid)
Job:Number	(Valid: upper- and lowercase okay)
A123,B456	(Illegal comma)
Reading:	(Illegal: colon not embedded)

Use enough characters to make the name meaningful. JOB:NMBR is a lot better than NO. and infinitely better than J.

dBASE II File Extensions: Conventions for dBASE II file names and extensions are described in Reference Chapter 2.

dBASE II Expressions: The use of expressions is covered in Chapter 2 and in Reference Chapter 3.

dBASE II Operators: Reference Chapter 3 describes the use of dBASE II operators (arithmetic, logical, relational, and string operators).

dBASE II Functions: See Reference Chapter 3 and User's Guide Chapter 4 for a summary of the dBASE II functions.

SUMMARY: COMMANDS GROUPED BY FUNCTION 5.3

The following abbreviations are used in this summary:

<exp>	=	expression
<var>	=	variable
<str>	=	string
<coord>	=	coordinates

5

Angled brackets < > enclose items that you specify when you enter the command (such as <filename> and <exp>).

Square brackets [..] enclose optional items. In some cases, options are nested (i.e., they themselves have other options).

For an alphabetical list of dBASE II commands, see Reference Chapters 9-12 or Reference Appendix B.

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

Combined, these two commands create a new file with the same structure as an old file.

USE <oldfile>
COPY TO <newfile> STRUCTURE EXTENDED

Combined, these commands 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 and opens the new.

USE

Closes all files.

RENAME <oldname> TO <newname>

Changes name of old file. You must not rename an open file.

COPY TO <filename>

Creates a backup file.

CLEAR

Closes all files and erases 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). You can use LIST instead.

DISPLAY FILES LIKE <wildcard> [ON <d>]

Shows other types of files on drives.

QUIT

Closes both active areas, all files, and terminates dBASE II operation.

ORGANIZING DATABASES

SORT ON <key> TO <newfile>

Generates a database sorted on the key.

INDEX ON <key> TO <newfile>

Creates an index file for the database in USE. You can use multiple keys for both commands.

REINDEX

Rebuilds index file(s) using original key(s).

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 a record so it is not used by dBASE II.

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. Make multiple changes to a database.

@ <coord> GET <var>

Formats console screen or printer output.

READ

Displays the formatted variable, and 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 you to change your command files without using your text editor.

USING VARIABLES

(Up to 64 memory variables plus any number of field names are allowed.)

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 after any key is pressed.

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.

TEXT
ENDTEXT

Displays all text between TEXT and ENDTEXT.

SEARCHING

SKIP [±<exp>]

Moves forward or backward a specific number of records.

GO[TO] <number>, GO TOP, GO BOTTOM

Moves 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 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 .PRG 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.

A

A WORKING ACCOUNTING SYSTEM

This appendix contains an accounting system of command files like ones that you can create following the instructions in this User's Guide. The system includes some programming techniques that you may find useful in your own data management.

This appendix illustrates the use of dBASE II commands, and demonstrates how to set up menus, how to find data and merge files, and how to set up the inputs to the system. You will find a number of interesting solutions to the problems of 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 throughout.

To create your own databases, of course, you start by using the CREATE command. Database structures that you might find useful are listed at the beginning of the appendix. COSTBASE.DBF, for example, started life as MONEYOUT.DBF, but field names and sizes are changed (with and without data in the database).

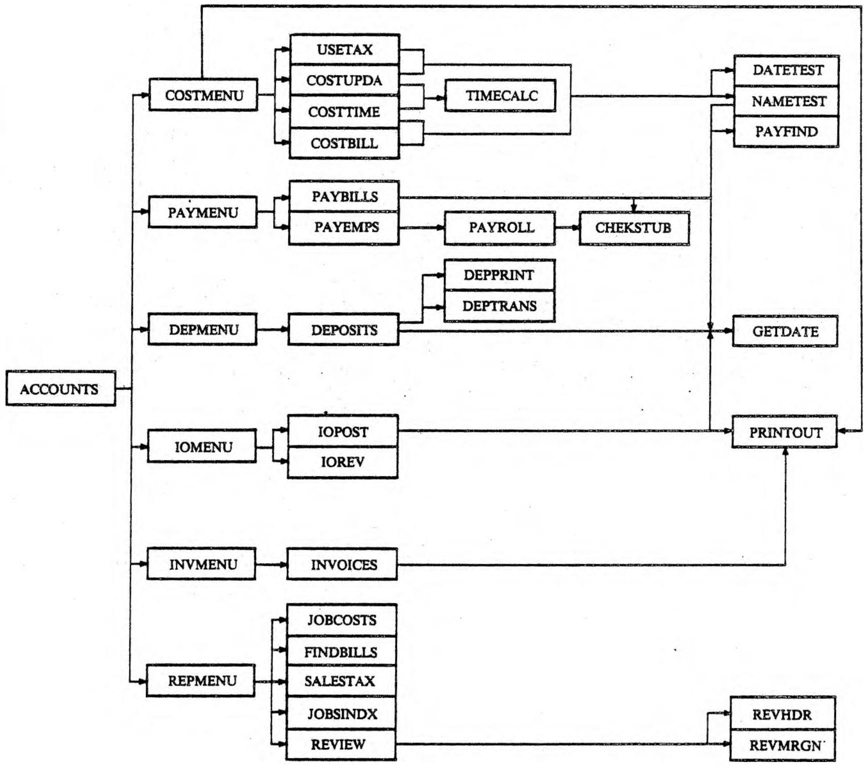
Check the database structures, then see how they are used in the programs. The field names and their individual structures are the same for all the databases to allow for file merges and other uses. Data from one database will fit into corresponding fields in another; with common names the transfer is straightforward.

These command files were written using the procedures recommended in the User's Guide. 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 understands in uppercase when you get to that level.

The indentation and mixture of upper- and lowercase letters makes writing the command files a lot easier because you can see groupings of the structures that you are using.

The diagram on this page shows the calling pattern for the accounting system. Command files (.PRG extension) use the DO command to call other files. In the diagram, arrows represent possible calls; files on the left end of an arrow can call files on the right end of that arrow. For example, the control module for the system (ACCOUNTS.PRG) can call any of six command files—COSTMENU, PAYMENU, DEPMENU, IOMENU, INVMENU, or REPMENU. Each of these files can call other command files.

Calling Sequence for Files in the Accounting System



CONTENTS

```
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         NAME            C     020
007         DESCRIP         C     020
008         BILL:DATE       C     007
009         BILL:NMBR       C     007
010         HOURS           N     006     002
011         EMP:NMBR        N     003
** TOTAL **                     00091
(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         NAME            C     020
006         DESCRIP         C     020
007         AMOUNT          N     009     002
008         BILL:DATE       C     007
009         BILL:NMBR       C     007
010         HOURS           N     006     002
011         EMP:NMBR        N     003
** TOTAL **                     00093

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         INV:DATE        C     006
005         TAXABLE         N     009     002
006         SALES:TAX       N     009     002
007         TAXFREE         N     009     002
008         PO:NMBR         C     008
009         DESCRIP         C     027
010         MORE            L     001
** TOTAL **                     00082
(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
005        SALES:TAX       N    009    002
006        TAXFREE         N    009    002
007        AMOUNT          N    009    002
008        AMT:RCD         N    009    002
009        DATE:RCD        C    007
** TOTAL **                   00070
(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        PAY:NMBR        C    007
004        DEPOSIT         N    009    002
005        INV:NMBR        C    006
006        COMMENTS        C    021    002
** TOTAL **                   00071

STRUCTURE FOR FILE:  B:CHECKFIL.DBF
FLD          NAME        TYPE WIDTH  DEC
001        CHECK:DATE      C    006
002        CHECK:NMBR      C    005
003        AMOUNT          N    009
004        BILL:NMBR       N    007
005        NAME            C    020
006        EMP:NMBR        N    003
007        CLIENT          C    003
008        JOB:NMBR        N    003
009        DESCRIP         C    020
010        BALANCE         N    009
** TOTAL **                   00087


STRUCTURE FOR FILE:  B:INSERTS.DBF
FLD          NAME        TYPE WIDTH  DEC
001        IO:NMBR         C    005
002        MAGAZINE             014
003        ISSUE           C    006
004        CLIENT          C    003
005        JOB:NMBR        N    003
006        AD                   015
007        SPACE           C    013
008        GROSS:COST      N    009    002
009        NET:COST        N    009    002
010        TIMES           C    003
011        IO:DATE         C    006
** TOTAL **                   00087
(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          MARKER           C     001
003          PAYROLL          N     009     002
004          FICA             N     008     002
005          FICASAL          N     009     002
006          FIT              N     009     002
007          SDI              N     007     002
008          SDISAL           N     009     002
009          SIT              N     009     002
010          UISAL            N     009     002
** TOTAL **                       00075

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          QDTSAL           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          SUPPLEIER        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 generates 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:

<u>JOBCOSTS.FRM</u>

```
n
n
y
n
9,BILL:DATE
DATE
22,NAME
SUPPLIER..............
17,DESCRIP
DESCRIPTION......
12,AMOUNT
AMOUNT
Y
```

<u>BILLED.FRM</u>

```
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 be changed in only one location to update the entire system.

NEXTCHECK	(C)	3565	Keep checkbook current in PAYBILLS, PAYROLL,
MBALANCE	(N)	23921.18	and DEPOSITS command files.
THISYEAR	(N)	81	Used by GETDATE, DATECHECK and PAYROLL files.
MINYEAR	(N)	79	
NEXT:IO	(C)	2885	Next insertion order number (IOPOST.PRG).
NEXT:INV	(C)	10623	Next invoice number (INVOICES.PRG).
FICACUT	(N)	0.0665	Entire grouping is used in the PAYROLL file.
FICAEND	(N)	29700.00	Easy to update every year because the values
MAXFICA	(N)	1975.05	are not sprinkled throughout the programs.
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. Used in TIMECALC program.

A

```
************************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


STORE  T TO Accounting
DO WHILE Accounting
   ERASE
   ?
   ?
   ? '1> ENTER BILLS & TIME SHEET         6> REPORTS & PRINTOUTS'
   ? '                                       Job cost & billing summaries'
   ? '2> PAY BILLS & SALARIES                Find & Edit bills by name'
   ? '                                       Review/print databases'
   ? '3> DEPOSITS & CHECKBOOK'
   ?
   ? '4> MEDIA INSERTION ORDERS'
   ?
   ? '5> CLIENT BILLINGS & INVOICES'
   ?
   ?
   ? '                 Pick a number (Q to QUIT)'
   WAIT TO Action
   DO CASE
      CASE Action = "Q"
         ERASE
         QUIT
      CASE Action = "1"
         DO Costmenu
      CASE Action = "2"
         DO Paymenu
      CASE Action = "3"
         DO Depmenu
      CASE Action = "4"
         DO Iomenu
      CASE Action = "5"
         DO Invmenu
      CASE Action = "6"
         DO Repmenu
   ENDCASE
   STORE T TO Accounting
ENDDO Accounting
```

```
*************************COSTMENU COMMAND FILE*************************
*      This is one level down from the Accounts.PRG control module.
* Selections 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


   DO CASE
      CASE 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
```

```
CASE Action = "2"
   ERASE
   @  4,10 SAY 'CHECK ALL THE BILLS BEFORE ENTERING THEM.'
   ?  '         If any of the bills are for items used by'
   ?  '         the agenoy but sales tax was not paid,'
   ?  '         select OPTION 1 from the entry menu.'
   ?  '         <Return> to continue.'
   ?
   WAIT
   DO CostBills
CASE Action = "3"
   DO CostTime
CASE 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;
               !(D), C with   !(Client),;
            Bill:Nmbr WITH !(B:N)
            ?
            ?  'Do you want to edit any other entries (Y or N)?'
            WAIT TO Changing
         ENDIF value out of range
      ENDIF  (eof)
   ENDDO Changing
   RELEASE All
CASE 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.'
```

A

```
                ? '<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
       CASE Action = "6"
          DO CostUpdate
       CASE 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
       OTHERWISE
          RELEASE All
          RETURN
       ENDCASE
    STORE T TO Posting
 ENDDO (POSTING)
```

```
**************************USETAX COMMAND FILE**************************
*       This file accepts inputs for supplier bills when the agency has
* brought 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) <> 'F'
   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

   * Following 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 separate the two types of entries later in the
   * SalesTax.PRG 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
```

A

```
***************************COSTBILL 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

   * Following checks names against a
   *list of supliers 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
```

A

```
*************************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.
*      CostTime is called by Costmenu.
************************************************************************

@  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 statements flags all entry errors,
      * then 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 TO 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 TO TO Entering
         ENDIF
      ENDIF
      IF Hours = 0
         ? '     HOURS must be entered.
         STORE TO TO Entering
      ENDIF

      IF (Emp:nmbr <= 0) .OR. (Emp:nmbr >= 100)
         ? '     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 data field 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

   * Verified match between employee name and number, then computes the
   * amount to be billed for the employee's time based on his salary.
   DO Time Calc

   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
```

A

```
********************  COSTUPDA 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
```

A

```
**************************PAYMENU COMMAND FILE**************************
*    This program is called by the Accounts.PRG 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 Conti~ue
         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, which calls it.
*       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.
*       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 brought in from PayFind.PRG.  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 number 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 (Q)
   ENDDO Confirm the record
ENDIF (Key)

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 unpaid bill meeting the criteria is found, Looking is set to
* False.  Otherwise 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.  Called by Paymenu.
************************************************************************
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
   DO CASE
      CASE Action = "1"
         DO Payroll
      CASE 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
         ELSE
            LOOP
         ENDIF
```

```
          RELEASE All
          ?
          ? 'Do you want to SKIP any employees (Y or N)?'
          WAIT TO Skip
          IF !(Skip) <> 'Y'
             DO Payroll
          ENDIF
          RELEASE Skip
       CASE 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
       OTHERWISE
          RELEASE ALL
          RETURN
       ENDCASE

   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 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, 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.
*      Payroll is called by Payemps.
************************************************************************

****************************************
*** 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
```

A

```
      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 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.PRG
      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 total 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 unemployment 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 _FF

   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 (Paid .OR.*)

   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

      DO CASE
         CASE $(Date,3,2) = '03'
            REPLACE Check:Date WITH '1ST'
         CASE $(Date,3,2) = '06'
            REPLACE Check:Date WITH '2ND'
         CASE $(Date,3,2) = '09'
            REPLACE Check:Date WITH '3RD'
         CASE $(Date,3,2) = '12'
            REPLACE Check:Date WITH '4TH'
      ENDCASE

      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  (EOQ = 'Y')

* 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 (EOM = 'y')

**********************************************************************
******  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

***************************************************************************
****** 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 housekeeping on the checkbook.
* Depmenu is called by Accounts.
*************************************************************

STORE TO 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

   DO CASE
      CASE Action = '1'
         DO Deposits
      CASE 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
      CASE 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
      OTHERWISE
         RELEASE All
         RETURN
   ENDCASE
   ERASE
   STORE T TO Incoming
ENDDO Incoming
```

A

```
***********************  DEPOSITS COMMAND FILE   ************************
* This file records any money coming in a file called Deposits.  If
* 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 session, 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 agaihst 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. It is called by
*  Deposits.
************************************************************************

@  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
```

A

```
          **********  DEPTRANS COMMAND FILE  **********
* Applies deposits from the GetDep file against the matching invoices
* in the Invoices file as payments are received against them.
* Deptrans is called by deposits.
***********************************************************************

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 (Again)
      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 thru 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).  Iomenu
* is called by Accounts.
*************************************************************************

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

   DO CASE
      CASE Action = "1"
         DO IOPost
      CASE 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
```

A

```
            ENDDO Changing
            RELEASE All
         CASE Action = '3'
            DO IOReview
         OTHERWISE
            RELEASE All
            RETURN
      ENDCASE
      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.  Iopost is called by Iomenu.
*************************************************************************

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
   @ 12,0 SAY '  WHIWHIMMGAZSNE ' GET Magaeine
   @ 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 if 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 printout

   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
```

A

```
*********************** 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.
* Ioreview is called by Iomenu.
***********************************************************************

SET TALK OFF
USE B: Inserts

STORE ' ' TO Again
DO WHILE !(Again) <> 'F'
   STORE '    ' TO MClient
   STORE '                ' TO Magazine
   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
```

A

```
         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.PRG file). Invmenu is called by Accounts.
************************************************************************
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
   DO CASE
      CASE Action = '1'
         DO Invoices
      CASE 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
      CASE 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
      OTHERWISE
         RELEASE All
         RETURN
   ENDCASE
   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
* by invoice number and the data is stored in the Invoices file.
*    A printout of items billed and invoice totals is provided.
* Invoices is called by Invemenu.
*************************************************************************

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 adjusted 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.
```

A

```
    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
```

```
               **********  REPMENU COMMAND FILE  **********
* This command file is a called by the ACCOUNTS.PRG 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

      DO CASE
         CASE 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
         CASE Action = '2'
            DO FindBills
         CASE 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 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
      CASE Action = '4'
         DO SalesTax
      CASE Action = '5'
         DO JobsIndx
      OTHERWISE
         RELEASE All
         RETURN
    ENDCASE
    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.
*     Jobcosts is called by Repmenu.
*************************************************************************

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
```

A

```
   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 JobCosts procedure,'
   ? '   please reindex from the next menu.'
   ? '<Return> to continue.'
   WAIT
ENDIF

RELEASE All
RETURN
```

A

```
********************** 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 JobCosts 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.
* Jobsindx is called by Repmenu.
*************************************************************************

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
```

A

```
*********************** FINDBILL 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 theOrest of the entries for the supplier based
* on name only.  Findbills is called by Repmenu.
************************************************************************

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 an 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
      * 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 ~ew 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.PRG 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.
*    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. Review.PRG is called by RepMenu.
************************************************************************


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 the 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
         @ 15,0  SAY CHR(27)+CHR(74)
         READ
      ELSE
         IF First > VAL(Last) .OR. First <= 0 .OR. RecoCnt > VAL(Last);
                              .OR. RecoCnt <= 0
```

A

```
         @ 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
   * Clear to end of screen
   @ 15,0  SAY CHR(27)+CHR(74)
ENDDO (Changing)
?
?
?
?
?
?
?
?
?
?
?
?
?
?
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) = '33'
      @ 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 Mark > 0
                  ENDIF Start > Size
               ENDIF Mark = 1 or Size
            ENDDO Size
         ENDIF Unfinished Y
      ENDIF Size
   ENDDO Unfinished

   IF LEN(Partial) > 0
   *   DO headings
      ? "WE'D DO THE HEADINGS HERE."
      WAIT
   ENDIF

ELSE
   STORE CHR(0) TO Partial
ENDIF Partial

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]+'  ***'
?
```

A

```
? '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
```

A

```
                  * 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 PageMark = 5
               ENDIF Count = 10
            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 opr N)?'
         WAIT TO Reviewing
      ELSE
         STORE 'Y' TO Reviewing
      ENDIF Abort C
   ENDIF Abort Q
   ?
ENDDO Reviewing
USE
DELETE FILE Temp
RELEASE All
RETURN
```

```
*********************** REVHDR COMMAND FILE ****************************
* Used by Review.PRG to print headings for different database listings.
************************************************************************

IF !(Database)='INSERTS
   ? 'IO#  MAGAZINE       ISSUE   JOB   AD             SPACE         '+;
     '   GROSS      NET   X  DATE'
ELSE
   IF !(Database)='BILLINGS'
      ? 'INV3  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.PRG 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
```

```
************************* SALESTAX 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. Salestax is called by
* Repmenu.
***********************************************************************


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'
? '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 printer
      * 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
```

A

```
*********************** TIMECALC COMMAND FILE ****************************
* Verifies that employee name and number match., then calculates billing
* charges for employee time. Called by Costupdate and CostTime.
***************************************************************************

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
ENDDO billing calculations

RELEASE All
RETURN
```

```
*********************** PRINTOUT COMMAND FILE **************************
* This file is used by several other command files.  It prints out a
* listing of the records in a file without the record number.  The
* output 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.PRG file.
*    Printout is called by Iopost and Invoices.
************************************************************************

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
```

A

```
********************* 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.
*  Getdate is called by Deposits, Paybills and Iopost.
************************************************************************

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.
*   Datetest is called by Usetax, Costupda, Costtime, & Costbills.
************************************************************************

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)+'/'+$(&Date,5,2)
            ? CHR(7)
            * The macro symbol is used to get the contents of the
            * date field being checked without creating a new variable.
            IF VAL($(&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.
* Nametest is called by Paybills, Usetax, Costupda, CostTime and
* Costbills.
**************************************************************************

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.                                 '
            @ 12,29 SAT '     A to ADD it to the SUPPLIERS file.            '
            @ 13,20 SAT '     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
```

A

```
            ENDDO (Again)
         ENDIF 0
         SELECT PRIMARY
         IF !(Decision) <> 'E'
            SKIP
         ENDIF
      ENDIF deleted
   ENDDO
   RELEASE All
   RETURN
```

A

```
*********************** CHEKSTUB 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.  Chekstub is called by Payroll and Paybills.
************************************************************************

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
```

INDEX

For each index listing, entries from the User's Guide are given first and are preceded by "UG". Entries from the Reference Manual are preceded by "Ref." and are given last.