





**Important Notice to dBASE II Users**  
Cat. No. 26-5352

This addendum replaces the information about system requirements on page 1 and the information on pages 2 and 3 of the dBASE II Introduction in your dBASE II operator's manual. This software is compatible with various Tandy computers. Before you attempt to run the software, please find your computer's configuration in the list below, and follow the instructions provided for that configuration.

**Minimum Configurations**

- Tandy 1000: 256K memory, 2 disk drives, VM-2 monitor, printer
- Tandy 1200: 256K memory, 1 disk drive with a hard disk, VM-3 monitor, video card, printer
- Tandy 2000: 256K memory, 2 disk drives, VM-1 monitor, printer

**Loading Instructions**

**Tandy 1200/2000 Single-Drive Hard Disk Systems**

1. Start up MS-DOS under hard disk control so that the C> prompt is displayed and you are in the \ directory.
2. If you want to copy your files into a sub-directory, you must first create that directory. If you do not want a directory, skip this step.
  - a. At the C> prompt, type:

**MKDIR \directory name> [ENTER]**

- b. Change to the new directory by typing:

CD \<directory name> [ENTER]

3. Place your dBASE II diskette in Drive A, and at the C> prompt, type:

COPY A:\*. \* C: [ENTER]

4. When the C> prompt reappears, your files have been moved to your hard disk.

#### Tandy 1000 Single- and Double-Drive Systems

1. Turn on your computer, and place an MS-DOS system diskette in Drive A.
2. Press the reset button.
3. At the A> prompt, type:

COPYDOS [ENTER]

**Note:** If you have a single disk drive, when prompted to insert a diskette in Drive B, remove the diskette in Drive A, and replace it with the diskette required for Drive B.

Insert a blank, unformatted Tandy 1000 diskette in Drive B. (First, be sure that the diskette's write-protect notch is not covered.) Close the drive door, and press the space bar to continue. COPYDOS formats the Drive B diskette. It then asks if you want to format another diskette. Press [N].

4. COPYDOS asks you to insert your application program diskette in Drive A and a blank, formatted diskette in Drive B. Remove the system diskette from Drive A, and replace it with your dBASE II diskette. The Drive B diskette is already formatted, so press the space bar to continue.

5. When prompted, remove the dBASE II diskette from Drive A, and insert the system diskette. Press the space bar to continue.

COPYDOS copies the MS-DOS files to the Drive B diskette. Once COPYDOS finishes executing, your Drive B diskette is a bootable dBASE II diskette.

### Tandy 2000 Double-Drive System

Converting a Tandy 1000/1200 diskette so that it runs efficiently on a Tandy 2000 computer is a simple task. The steps for completing the conversion are provided below.

1. Turn on your Tandy 2000 computer.
2. Insert a Tandy 2000 MS-DOS system diskette in Drive A, and press the reset switch.
3. Place a blank, unformatted diskette in Drive B, and type the following command:

**FORMAT B: /S [ENTER]**

4. After the diskette has been formatted, the system is transferred. This makes your diskette bootable. When it is finished, you are prompted to format another diskette. If you wish to format another diskette, type:

**Y [ENTER]**

When you have finished formatting, type:

**N [ENTER]**

5. Remove the system diskette from Drive A, and replace it with your master application diskette. Type:

**COPY A:\*. \* B: [ENTER]**

As each file is copied, the file name is displayed on your screen. When all the files are copied, the A> prompt is displayed.

6. Remove your master application diskette from Drive A, and put it in a safe place. Use the original only to make working copies of your application.

## Backup Instructions for the Sample Program Diskette

### Single-Drive Systems

1. Insert an MS-DOS system diskette in Drive A, and type:

**FORMAT A: [ENTER]**

You are prompted to insert a blank, unformatted diskette in Drive A and press any key. When the formatting is finished, you are prompted to format another diskette. Press [N].

2. Remove the formatted diskette, and replace it with the MS-DOS system diskette. Type:

**DISKCOPY [ENTER]**

You are prompted to switch out the appropriate diskettes in Drive A until the diskette has been successfully copied. You are then prompted to copy another diskette. Press [N].

### Double-Drive Systems

1. Insert an MS-DOS system diskette in Drive A, and type:

**FORMAT B: [ENTER]**

You are prompted to insert a blank, unformatted diskette in Drive B and press any key. When the formatting is complete, you are prompted to format another diskette. Press [N].

2. To copy the diskette, type the following:

**DISKCOPY A: B: [ENTER]**

You are prompted to place the application diskette in Drive A and a blank, formatted diskette in Drive B. When you have done this, press the space bar. When the copy is complete, you are prompted to copy another diskette. Press [N].

### Tandy 1000 Printer Configuration Instructions

Before starting your application, you must assure that the printer you are using is set up to work with your Tandy 1000. To accomplish this, find your printer on the list below, and follow the indicated steps.

If your printer is:	Do this:
IBM-compatible (IBM, Epson, etc.)	No action required
LPVIII DMP120 DMP200 DMP400 DMP420 DMP500 DMP2100	Set NL/CR switch to CR
DWP11 DWP11b DWP410 DWP210	Follow Procedure I
Other	Follow Procedure II

#### Procedure I

1. At the MS-DOS prompt, type:

```
lpinst [ENTER]
```

The screen shows:

```
Does your printer automatically linefeed
after a carriage return?
```

2. Press [Y].



When the printer configuration process is finished, the MS-DOS prompt returns to the screen. The printer configuration procedure need be performed only once. Now, whenever you run the application, the printer configuration is done automatically. (The LPINST command builds a new autoexec.bat file and then appends any existing autoexec.bat file to the newly-built autoexec.bat file.)

#### Procedure II

1. At the MS-DOS prompt, type:

```
lpinst [ENTER]
```

The screen shows:

```
Does your printer automatically linefeed
after a carriage return?
```

2. Press [Y] or [N]. If you are not sure about the answer you should give, run a printout test. If the printer doublespaces lines, return to MS-DOS, run LPINST as instructed above, and answer [Y]. Again, run a printout test to ensure that the printer is performing as expected.

**Note:** If you do not want the system to automatically configure itself for your printer (for example, if you change printers frequently), you can manually reconfigure the system by issuing a simple "MODE" command at the MS-DOS prompt.

To turn off the extra linefeed, type:

```
lf [ENTER]
```

Then, when the system prompt returns to the screen, type:

```
mode lloff [ENTER]
```

To turn on the extra linefeed, type:

lf [ENTER]

Then, when the system prompt returns to the screen, type:

mode lfon [ENTER]

If you reset the computer, the system returns to its default setting, LFON. After you issue the appropriate mode command, your Tandy 1000 is configured to work with your printer.

**Model 2000  
dBASE II  
Cat. No. 26-5352**

**MODE**

The program **MODE** is included on your diskette to permit you to select the video mode in which to run **dBASE II**. Four options are available with the **MODE** utility:

<b>BW</b>	— black and white
<b>COLOR</b>	— color (available only if you have a color graphics board and a color monitor)
<b>40</b>	— 40 characters per line
<b>80</b>	— 80 characters per line

In addition to the four options given above you may enter a question mark (?) to display help documentation.

To select the appropriate video mode you must execute **MODE** before running **dBASE II**. The default mode is set to **BW**, 80 characters. To specify an option other than the default, simply enter **MODE** followed by at least one of the options presented above. For example, you may set color by entering the following at the system prompt:

**MODE COLOR**

**TANDY**

SECRET  
11 20 00  
SECRET

SECRET

SECRET

SECRET

SECRET

SECRET

SECRET

SECRET

**Model 2000**  
**dBASE II**  
**Cat. No. 28-5352**

**SAMPLE DATABASES AND COMMAND FILES**

The sample program diskette which is included with the product, contains sample databases and command files to be used as examples of some of the applications for dBASE II. These files are intended to be useful as they are and are internally documented so that you can customize them to suit your own purposes. Each set of programs listed below is progressively more involved, so you can start at a level comfortable to you and work up. To utilize these applications, place the dBASE II program diskette in Drive A and the sample program diskette in Drive B. Execute dBASE and at the dot prompt enter:

**SET DEFAULT TO B:**

Each application can then be run by entering DO followed by the name of the main program of each application. For example, the checkbook example may be run by entering:

**DO CHXMENU**

The main program of each group is boldfaced.

**TICKLE FILE MANAGEMENT FILES (CARDFILE)**

<b>CMAIN.PRG</b>	TEX.DBF	CPRINT.PRG	KEYWORD.NDX
APPECARD.FMT	TICKLE.NDX	TICKLE.FRM	EDITCARD.FMT

**CHECKBOOK MANAGEMENT FILES**

<b>CHXMENU.PRG</b>	CHXINPUT.FMT	CHXDEPOS.FMT	CHXBOOK.MEM
CHECKS.DBF	DEPOSITS.DBF		

**INVENTORY MANAGEMENT**

<b>IMAIN.PRG</b>	INVENT.DBF	ISETUP.PRG	I'BYNMBR.NDX
INVMAINT.PRG	I'BYDESC.NDX	INVQUAN.PRG	INV'CONS.FRM
INVREAD.PRG	INV'PRTR.FRM	INVRPRT.PRG	

**PERSONNEL MANAGEMENT FILES**

<b>EMAIN.PRG</b>	EMPLOYEE.DBF	ESETUP.PRG	EMPSCRN.FMT
EMP-ENTR.PRG	EMP-UPD.FMT	EMP-UPD.PRG	EMP-PRT.FRM
EMP-TERM.PRG	EMP-CONS.FRM	EMP-RPRT.PRG	E'BYNMBR.NDX
EMPLOYEE.MEM	E'BYNAME.NDX		

**TANDY**



## **dBASE II Technical Notes**

- \*\*\* If a database field is defined with a width greater than 70, the 71st character is not shown when a LIST or DISPLAY command is executed.

To view the 71st character, use EDIT or BROWSE.

- \*\*\* Attempts to produce a report from a database file containing no data results in extraneous characters being displayed or data from another database file shown in the report.

Execute reports from a command file. In the command file precede the report command with the conditional statement:

```
IF # < > 0
```

If this conditional statement is met, there is data in the file. If there is no data in the file, # is equal to 0 and no report is printed.

- \*\*\* When utilizing the JOIN command to create a single database with fields from separate databases, the secondary database is left open and displays the message:

```
FILE CURRENTLY IN USE
```

To close the secondary database, at the dot prompt, enter:

```
SELECT SECONDARY
```

When the dot prompt is again displayed, type:

```
USE
```

- \*\*\* Any MODIFY STRUCTURE command following a MODIFY COMMAND entry results in extraneous characters displayed on the screen.

Perform a USE <database> immediately before executing the MODIFY STRUCTURE command.

- \*\*\* In BROWSE, ctl-B pans to the right moving the cursor to the next field as it pans. However, ctl-Z pans to the left but does not move the cursor as it pans.

- \*\*\* PACK does not release disk space or update the database file size when deleted records are removed with this command. As subsequent records are added to the database file, the empty space from the PACK is reused.

- \*\*\* EJECT ON/OFF applies only to those line printers that have the capability to perform form feeds, including the LP 2150 and DMP series of printers with the exception of the DMP 100 and DMP 120.

\*\*\* The CLEAR and USE commands do not always close files left open as the result of syntax errors.

If you receive the message FILE CURRENTLY IN USE and the USE or CLEAR command does not correct it, quit dBASE II and begin again.

## **dBASE II Manual Corrections and Additions**

In the manual, DISPLAY STRUCTURE and LIST STRUCTURE cause slightly different results than the example screens illustrate. The header proceeding the display of the database STRUCTURE is:

```
STRUCTURE FOR FILE:   X:< database filename >
.DBF
```

where X is the drive indicator.

### **Part A Changes**

**Contents, Section I** Entering data into your database begins on page 13.

**Page 2, Introduction** You must use a formatted disk to make a backup. Replace the existing note with the following:

Note: If your disk has not been formatted, follow the instructions under "Formatting Data Diskettes" before performing a DISKCOPY.

**Page 12, Creating Databases** After the first sentence, add the following:

A numeric field may be defined with up to nine decimal places.

**Page 13, Creating Databases** There should be a colon between the words ZIP and CODE on the first screen.

**Page 14, Creating Databases** The record number on the screen should be 00003.

**Page 23, Creating Databases** At the bottom of the page, insert ♦? State♦ after name and insert ♦? Name♦ after skip.

**Section II, Contents** Summarizing data and eliminating details begins on page 58.

**Page 32, Expressions** The last name on the screen should be Edmunds, Jim rather than Destry, Ralph.

**Page 34, Expressions** The number of bytes used on the screen is 00061 instead of 00084.

**Page 35, Expressions** The result of the second problem is 0.00713 rather than 0.00644.

**Page 38, Expressions** The record number for Clinker, Duane should be 00002.



**Page 40, Expressions** On the screen, there should be nine spaces after Pat and nine spaces after is.

**Page 41, Expressions** The fields on the screen should appear as follows.

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	

The sentence below the screen should indicate that dBASE II lists the first 22 (or fewer) fields.

**Page 42, Expressions** Delete record 00002 Brown, John from the screen. Record 3 becomes record 2 and record 4 becomes record 3. Add the following as record 4.

00004 EDMUNDS, JIM 392 Vicarious Way  
Atlanta GA 30328

**Page 43, Expressions** The first screen should say:

.copy to temp structure

In the Advanced Programmers section, add ♦**DISPLAY STRUCTURE**♦ before ♦**List**♦.

**Page 44, Expressions** The Records Copied and Number of Records should be 00005. Delete the parenthetical phrase (see Review. CMD, Section VI) from the first paragraph.

**Page 49, Expressions** Replace the last paragraph with the following information.

A CustCode should be added to each record in the Names database starting with the second record. The first record remains unchanged for a subsequent example of an alternate method of making updates to the database.

To update the second record, ♦**USE Names**♦ then type ♦**EDIT 2**♦. Use the full screen editing features of dBASE to position the cursor beside the CUSTCODE prompt and enter ♦**1002**♦. To move to the next record, type ♦**cti-C**♦. Continue entering customer codes as four digit numbers with the record number as the last two digits (1003, 1004, 1005, and so on).

**Page 50, Expressions** Add .F. to the number in the last column of the first and last records on the screen.

**Page 52, Expressions** Delete record 2 from the screen. Record 3 becomes record 2 and record 4 becomes record 3. The CustCode of the new record 2 is 1002. The CustCode of the new record 3 is 1003. Insert the following record 4.

00004 EDMONDS, JIM 392 Vicarious Way  
Atlanta GA 30328 1004

The CustCode of record 9 is 1014. The CustCode of record 14 is 1013.

**Page 54, Expressions** On the screen, Clinker, Duane is record 00002, and the CustCode for this record is 1002.

**Page 55, Expressions** On the screen, the number after Alazar, Pat should be 98206 followed by record 2 rather than 3.

**Page 56, Expressions** The screen should show L = 66,W = 80 after Page Width. At the bottom of the page, delete the apostrophes from around the last 770 you type.

**Page 57, Expressions** On the screen, add 00/00/00 under the page number line. Change the first paragraph to the following:

An additional heading may be added to the report at runtime by typing **◆SET HEADING TO character string◆** prior to executing the REPORT command. The character string may be up to 60 characters and spaces with no quotation marks.

**Page 58, Expressions** On the screen, 00014 bytes are used instead of 00012 bytes. Insert the following after the first sentence in the second paragraph from the bottom:

If the database named in the TOTAL command already exists, the key field and the fields being totaled must be defined in the structure of the summary database. If the named database does not exist, all fields will be transferred from the database in use to create a new database.

If you have already created a database file named TEMP in a previous example in this manual, please remove TEMP before continuing with this example by entering:

**DELETE FILE TEMP**

**Section III,Contents** Delete TEXT and .FMT from the commands list.

**Page 62, Command Files** In the first sentence, use cti-W to get back to the dBASE II prompt. To rename the main dBASE file, type:

**A> ◆RENAME DBASE.COM DO.COM◆**

At the bottom of the page, delete the apostrophes around the 730. Change Supplier to Name.

**Page 64, Command Files** In the example, insert SKIP after STORE Index + 1 TO Index.

**Page 73, Command Files** Delete the RESET paragraph from the page.

**Page 89, Organization** In the fourth paragraph on the page, the last sentence should show DBMS rather than DMBS.

## Part B

**Contents** Section 4.1 Functions begins on page 9. The section heading Rules of Commands should be Rules to Operate By.

**Page 1, dBASE** Delete the text beginning at the second paragraph to the sentence "The sign-on message . . .".

**Page 40, ACCEPT** The example use 22 bytes and 37 bytes respectively.

**Page 48, BROWSE** The up arrow, or cti-E, backs up to the previous data field. The down arrow, or cti-X, advances to the next data field. The right arrow, or cti-D advances to the next character. The left arrow, or cti-S, backs up to the previous character.

**Page 88, INPUT** In the example, the word EVERYTHING is misspelled.

**Page 89, INPUT** A total of 59 bytes is used in the example.

**Page 97, LOCATE** When using LOCATE with the NEXT clause, the message END OF FILE ENCOUNTERED is displayed if the expression cannot be found within the scope of the NEXT.

**Page 98, LOCATE** The message END OF FILE ENCOUNTERED is displayed instead of END OF FILE.

**Page 123, REPORT** This example has 12 records instead of the 16 indicated.

**Page 126, REPORT** Delete the games on 07/07/82 and 07/23/82. The scores are changed to the following: List = 112; Anna = 135; Wayne = 91; game total = 338. Delete the Report Form Orders example.

**Page 128, RESET** Delete this page.

**Page 129, RESTORE** The total bytes used in the examples are 43, 0, and 43 respectively.

**Page 131, SAVE** The total bytes used in the examples are 43, 0, and 43 respectively.

**Page 142, STORE** Delete EOF (L) .T. from the example. This example uses 43 bytes.

**Page 143, SUM** In the example, 7 bytes are used rather than the 6 bytes indicated.

**Page 146, TOTAL** Create a database named CALLS with fields PART:NO and AMOUNT as shown in the example.

Page 10: Continued from the previous page.

Page 11: Continued from the previous page.

Page 12: Continued from the previous page.

Page 13: Continued from the previous page.

Page 14: Continued from the previous page.

Page 15: Continued from the previous page.

Page 16: Continued from the previous page.

Page 17: Continued from the previous page.

Page 18: Continued from the previous page.

Page 19: Continued from the previous page.

Page 20: Continued from the previous page.

Page 21: Continued from the previous page.

Page 22: Continued from the previous page.

Page 23: Continued from the previous page.

Page 24: Continued from the previous page.

Page 25: Continued from the previous page.

Page 26: Continued from the previous page.

**dBASE<sup>®</sup>**  




TERMS AND CONDITIONS OF SALE AND LICENSE OF RADIO SHACK AND TANDY COMPUTER EQUIPMENT AND SOFTWARE PURCHASED FROM A RADIO SHACK COMPANY-OWNED COMPUTER CENTER, RETAIL STORE OR FROM A RADIO SHACK FRANCHISEE OR DEALER AT ITS AUTHORIZED LOCATION.

**LIMITED WARRANTY**

**I. CUSTOMER OBLIGATIONS**

- A. CUSTOMER assumes full responsibility that this computer hardware purchased (the "Equipment") and any copies of software included with the Equipment or licensed separately (the "Software") meets the specifications, capacity, capabilities, versatility, and other requirements of CUSTOMER.
- B. CUSTOMER assumes full responsibility for the condition and effectiveness of the operating environment in which the Equipment and Software are to function, and for its installation.

**II. RADIO SHACK LIMITED WARRANTIES AND CONDITIONS OF SALE**

- A. For a period of ninety (90) calendar days from the date of the Radio Shack sales document received upon purchase of the Equipment, RADIO SHACK warrants to the original CUSTOMER that the Equipment and the medium upon which the Software is stored is free from manufacturing defects. THIS WARRANTY IS ONLY APPLICABLE TO PURCHASES OF RADIO SHACK AND TANDY EQUIPMENT BY THE ORIGINAL CUSTOMER FROM RADIO SHACK COMPANY-OWNED COMPUTER CENTERS, RETAIL STORES AND FROM RADIO SHACK FRANCHISEES AND DEALERS AT ITS AUTHORIZED LOCATION. The warranty is void if the Equipment's case or cabinet has been opened, or if the Equipment or Software has been subjected to improper or abnormal use. If a manufacturing defect is discovered during the stated warranty period, the defective Equipment must be returned to a Radio Shack Computer Center, a Radio Shack retail store, participating Radio Shack franchisee or Radio Shack dealer for repair, along with a copy of the sales document or license agreement. The original CUSTOMER'S sole and exclusive remedy in the event of a defect is limited to the correction of the defect by repair, replacement, or refund of the purchase price, at RADIO SHACK'S election and sole expense. RADIO SHACK has no obligation to replace or repair expendable items.
- B. RADIO SHACK makes no warranty as to the design, capability, capacity, or suitability for use of the Software, except as provided in this paragraph. Software is licensed on an "AS IS" basis, without warranty. The original CUSTOMER'S exclusive remedy, in the event of a Software manufacturing defect, is its repair or replacement within thirty (30) calendar days of the date of the Radio Shack sales document received upon purchase of the Software. The defective Software shall be returned to a Radio Shack Computer Center, a Radio Shack retail store, participating Radio Shack franchisee or Radio Shack dealer, along with the sales document.
- C. Except as provided herein, no employee, agent, franchisee, dealer, or other person is authorized to give any warranties of any nature or on behalf of RADIO SHACK.
- D. EXCEPT AS PROVIDED HEREIN, RADIO SHACK MAKES NO EXPRESS WARRANTIES, AND ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE IS LIMITED IN ITS DURATION TO THE DURATION OF THE WRITTEN LIMITED WARRANTIES SET FORTH HEREIN.
- E. Some states do not allow limitations on how long an implied warranty lasts, so the above limitations may not apply to CUSTOMER.

**III. LIMITATION OF LIABILITY**

- A. EXCEPT AS PROVIDED HEREIN, RADIO SHACK SHALL HAVE NO LIABILITY OR RESPONSIBILITY TO CUSTOMER OR ANY OTHER PERSON OR ENTITY WITH RESPECT TO ANY LIABILITY, LOSS OR DAMAGE CAUSED OR ALLEGED TO BE CAUSED DIRECTLY OR INDIRECTLY BY EQUIPMENT OR SOFTWARE SOLD, LEASED, LICENSED OR FURNISHED BY RADIO SHACK, INCLUDING BUT NOT LIMITED TO, ANY INTERRUPTION OF SERVICE, LOSS OF BUSINESS OR ANTICIPATORY PROFITS OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OR OPERATION OF THE EQUIPMENT OR SOFTWARE IN NO EVENT SHALL RADIO SHACK BE LIABLE FOR LOSS OF PROFITS OR ANY INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY BREACH OF THIS WARRANTY OR IN ANY MANNER ARISING OUT OF OR CONNECTED WITH THE SALE, LEASE, LICENSE, USE OR ANTICIPATED USE OF THE EQUIPMENT OR SOFTWARE.
- NOTWITHSTANDING THE ABOVE LIMITATIONS AND WARRANTIES, RADIO SHACK'S LIABILITY HEREUNDER FOR DAMAGES INCURRED BY CUSTOMER OR OTHERS SHALL NOT EXCEED THE AMOUNT PAID BY CUSTOMER FOR THE PARTICULAR EQUIPMENT OR SOFTWARE INVOLVED.
- B. RADIO SHACK shall not be liable for any damages caused by delay in delivering or furnishing Equipment and/or Software.
- C. No action arising out of any claimed breach of this Warranty or transactions under this Warranty may be brought more than two (2) years after the cause of action has accrued or more than four (4) years after the date of the Radio Shack sales document for the Equipment or Software, whichever first occurs.
- D. Some states do not allow the limitation or exclusion of incidental or consequential damages, so the above limitations or exclusions may not apply to CUSTOMER.

**IV. RADIO SHACK SOFTWARE LICENSE**

- RADIO SHACK grants to CUSTOMER a non-exclusive, paid-up license to use the Software on one computer, subject to the following provisions:
- A. Except as otherwise provided in this Software License, applicable copyright laws shall apply to the Software.
- B. Title to the medium on which the Software is recorded (cassette and/or diskette) or stored (ROM) is transferred to CUSTOMER, but not title to the Software.
- C. CUSTOMER may use Software on one host computer and access that Software through one or more terminals if the Software permits this function.
- D. CUSTOMER shall not use, make, manufacture, or reproduce copies of Software except for use on one computer and as specifically provided in this Software License. Customer is expressly prohibited from disassembling the Software.
- E. CUSTOMER is permitted to make additional copies of the Software only for backup or archival purposes or if additional copies are required in the operation of one computer with the Software, but only to the extent the Software allows a backup copy to be made. However, for MSDOS Software, CUSTOMER is permitted to make a limited number of additional copies for CUSTOMER'S own use.
- F. CUSTOMER may resell or distribute unmodified copies of the Software provided CUSTOMER has purchased one copy of the Software for each one sold or distributed. The provisions of this Software License shall also be applicable to third parties receiving copies of the Software from CUSTOMER.
- G. All copyright notices shall be retained on all copies of the Software.

**V. APPLICABILITY OF WARRANTY**

- A. The terms and conditions of this Warranty are applicable as between RADIO SHACK and CUSTOMER to either a sale of the Equipment and/or Software License to CUSTOMER or to a transaction whereby RADIO SHACK sells or conveys such Equipment to a third party for use to CUSTOMER.
- B. The limitations of liability and Warranty provisions herein shall inure to the benefit of RADIO SHACK, the author, owner and/or licensor of the Software and any manufacturer of the Equipment sold by RADIO SHACK.

**VI. STATE LAW RIGHTS**

- The warranties granted herein give the original CUSTOMER specific legal rights, and the original CUSTOMER may have other rights which vary from state to state.



**(This page is intentionally left blank)**







**dBASE II®**

**Assembly Language  
Relational Database  
Management System**



**USER MANUAL**



**dBASE II®**

**Computer Program and User Manual**

**Copyright © 1981, 1983 Ashton-Tate as an unpublished work;**

**Confidential, Proprietary Material**



**All Rights Reserved**

**Licensed to Tandy Corporation**

This program and manual are the property of Ashton-Tate and embody proprietary, confidential and trade secret information of Ashton-Tate. The dBASE II program and manual are protected by trade secret and copyright laws. The dBASE II program shall be used only in conformance with the terms of the Limited Warranty and Software License contained in this manual. Use of the dBASE II program other than as expressly authorized is prohibited and a violation of the copyright laws.

Violation of copyright laws or misappropriation of trade secrets can result in criminal prosecution, fines and imprisonment in addition to civil damages.

Reproduction of any portion of this manual without express written permission from Tandy Corporation and Ashton-Tate is prohibited. While reasonable efforts have been taken in the preparation of this manual to assure its accuracy, Tandy Corporation and Ashton-Tate assume no liability resulting from any errors or omissions in this manual, or from use of the information contained herein.



## DEFINITION OF PRODUCT


When you purchase dBASE II you should receive the following:

*MS™DOS System Diskette with dBASE II Programs.*

*dBASE II User Manual.*

*Diskette of sample databases.*

*One set of three function key templates.*





**(This page is intentionally left blank)**



# PART A CONTENTS

## Introduction and Installation:

Introduction .....	1
System Requirements .....	1
dBASE II Specifications .....	1

## Disk Preparation

Making a 'Working' and 'Backup' Copy of the dBASE II diskette .....	2
Formatting Data Diskettes .....	3
Installing dBASE II on your Hard Disk .....	3
Backing Up files from Hard Disk .....	4
Restoring files to Hard Disk .....	4
File naming conventions .....	4
Typographic conventions used in this manual .....	6
Bringing up dBASE II .....	7

## Section I:

How to CREATE a database .....	11	CREATE
Entering data into your new database .....	12	
Modifying data in a database .....	14	EDIT, BROWSE
Full Screen Editing Features .....	15	
An introduction to dBASE II to commands and the error correction dialog .....	16	USE, DISPLAY, LIST
Expanding commands with expressions .....	17	LIST
Looking at your data records .....	20	DISPLAY
Positioning yourself in the database .....	21	GO, GOTO, SKIP
The interactive ? command .....	23	?
Adding more records to a database .....	24	APPEND, INSERT
Cleaning up a database .....	26	DELETE, RECALL, PACK
Section I Summary .....	28	

## Section II:

Using expressions for selection and control .....	31	
Constants and variables .....	31	STORE
dBASE II operators .....	35	
Logical operators .....	36	
Substring logical operator .....	38	
String operators .....	39	
Changing an empty database structure .....	40	MODIFY
Duplicating databases and structures .....	41	COPY
Adding and deleting fields with data in the database .	44	COPY, USE, MODIFY
Dealing with MS-DOS and "foreign files" .....	47	COPY, APPEND
Renaming database fields .....	48	COPY, APPEND
Modifying data rapidly .....	49	REPLACE, CHANGE
Organizing your databases .....	51	SORT, INDEX
Finding the information you want .....	53	FIND, LOCATE
Getting information out of all that data .....	55	REPORT
Automatic counting and summing .....	57	COUNT, SUM
Summarizing data and eliminating details .....	58	TOTAL
Section II Summary .....	60	

### Section III:

Setting up a command file (writing your first program) . . . . .	61	MODIFY COMMAND <file>
Making choices and decisions . . . . .	62	IF..ELSE..ENDIF
Repeating a process . . . . .	64	DO WHILE..
Procedures (subsidiary command files) . . . . .	65	DO <file>
Entering data interactively during a run . . . . .	65	WAIT, INPUT, ACCEPT
Placing data and prompts exactly where you want them . . . . .	66	@..SAY..GET
A command file that summarizes what we've learned . . . . .	69	
Working with multiple databases . . . . .	72	SELECT PRIMARY/ SECONDARY
Generally useful system commands and functions . . . . .	73	
A few words about programming and planning your command files . . . . .	74	

### Section IV:

Expanding your control with functions . . . . .	77	
Changing dBASE II characteristics and defaults . . . . .	80	SET..
Merging records from two databases . . . . .	82	UPDATE
JOINING entire databases . . . . .	82	JOIN
Full screen editing and formatting . . . . .	83	SET FORMAT TO SCREEN
Formatting the printed page . . . . .	85	@..SAY..GET..PICTURE..
Setting up and printing a Form . . . . .	86	SET FORMAT TO PRINT
Time to regroup . . . . .	87	@..SAY..USING

### Section V:

Database Basics . . . . .	89
A brief introduction to database organization . . . . .	90
dBASE II Records, Files and Data Types . . . . .	91
dBASE II OPERATIONS SUMMARY . . . . .	94
dBASE II FUNCTION SUMMARY . . . . .	95
dBASE II COMMAND SUMMARY . . . . .	96
Commands grouped by what you want done . . . . .	100
100 File structure	
101 File operations	
102 Organizing database	
102 Combining databases	
102 Editing, updating, changing data	
103 Using variables	
104 Interactive input	
104 Searching	
104 Output	
105 Programming	

## Introduction

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

- Create complete database systems.
- Easily add, delete, edit, display and print data from your database, with a minimum of data duplication on file.
- Gain a large measure of program/data independence, so that when you change your data you don't have to change your programs, and vice-versa.
- Generate reports from one or more databases, automatically do multiplication, division, sub-totals, totals and other data manipulation every time you use them.
- Use the full-screen editing capability to set up a screen format, so that you see exactly what you're going to get, and enter data by simply "filling in the blanks."

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

## Systems Requirements

dBASE II requires the following hardware and software environment:

- Tandy Model 2000 with MS™-DOS operating system.
- 256K bytes minimum of memory
- One or more mass storage devices (usually floppy disk drives).
- Optional text printer (for some commands).

## dBASE II Specifications

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

MS is the trademark of Microsoft Corporation.

## INTRODUCTION . . . 2

### Disk Preparation

If you have a Model 2000 configured with floppy diskette drives, follow the disk preparation instructions described below on Making a 'Working' and 'Backup' Copy of the dBASE II Diskette, and in the section on Formatting Data Diskettes. Hard disk users should skip to the section on Installing dBASE II on your Hard Disk and continue reading the next two sections on Backing Up and Restoring Hard Disk files.

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

#### Making a 'Working' and 'Backup' Copy of the dBASE II diskette

In order to protect yourself against accidental loss of any portion of your new product, you should never use your distribution disk copy of dBASE II to run the dBASE II program. Instead, you should employ it solely for the purpose of making 'working' and 'backup' copies of the program.

After you have made a working and backup copy of the program following the directions given below, store your dBASE II distribution disk away in a safe place. If a bad power surge should ever strike — and we certainly hope it never will — your dBASE II Master will always be there on your shelf to turn a potential disaster into a mere inconvenience.

To make your 'working' and 'backup' copies of dBASE II, you may simply copy the contents of your dBASE II distribution diskette onto blank disks using your disk copy utility, DISKCOPY.

To carry out this procedure, first 'boot' your system with your dBASE II distribution diskette in drive A. When your system prompt A> appears on the screen, enter the following command:

```
DISKCOPY A: B: <enter>
```

**Note:** Here and elsewhere '<enter>' denotes your console's ENTER key.

As soon as your system's DISK COPYing utility has been taken into memory, your computer will prompt you to place the 'source' of the copy operation into drive A, and the 'target' of the copy operation in drive B.

Place a blank disk in drive B, and enter any console character to execute the copy.

**Note:** If your blank disk has not been formatted, your system will automatically format it before executing the copy operation.

When the copy operation has been successfully completed, your system will ask you if you wish to make another copy. Enter Y to make another copy.

When your system prompts you to place the 'source' in drive A and the 'target' in drive B, simply replace the disk in drive B with another blank disk (leave the distribution disk in drive A), and press any key to execute the copy operation once again.

When your system finishes the copy operation, and asks you if you wish to make another copy, enter N to exit the DISKCOPY program and return to the A>.



Remove the dBASE II distribution disk from drive A, return it to its protective envelope, and store it away in a safe place. Remove the new disk from drive B.

At this point mark one of your new copies of dBASE II as your 'working' copy, and the other as your dBASE II 'backup'. Place your 'working' copy in drive A, and put aside your 'backup' for safekeeping.

For more information about your system's DISKCOPYing utility, see your DOS system manual.

### Formatting Data Diskettes

If you wish to save your data on separate diskettes, you must format the blank diskette before you can use it. To format a blank diskette or reformat an old diskette, type:

```
FORMAT [drive]:
```

Where drive specifies the drive in use. Immediately after you enter the FORMAT command, MS-DOS displays a format prompt followed by instructions to insert a diskette. Insert a blank diskette or a diskette to be reformatted and press any key. As the diskette is formatted, the dashed formatting line turns to periods to indicate the format has not encountered errors. A question mark in place of a dash indicates that the corresponding track on the disk contains flawed sectors. This causes no problems, however, because FORMAT locks out the flawed sectors so that MS-DOS never writes to them. When the FORMAT is complete, the screen shows the total disk space, the number of bytes in bad sectors, and the number of bytes available on the disk. You are prompted to format another diskette. If you wish to format another disk press <Y><enter>. When you have finished formatting, press <N><enter> to the format question.

### Installing dBASE II on your Hard Disk

In what follows, we will assume that you have already formatted your hard disk according to the instructions in your Introduction to Model 2000 Manual. Moreover, we will assume that disk drive C is your hard disk drive and disk drive A is one of your floppy disk drives. (If your configuration is different simply enter the appropriate drive name in the place of drive A and C below.)

To place the contents of your dBASE II distribution diskette onto your hard disk, simply 'boot' your system, and when your system prompt C> appears on the screen, place the distribution diskette in disk drive A. Now enter:

```
COPY A:DBASE*. * C:/V <enter>
```

**Note:** Here and elsewhere '<enter>' denotes your console's ENTER key.

This tells your computer to copy all dBASE II files on A (the distribution disk) to hard disk drive C. (DBASE\*. \* means 'all dBASE files'. '/V' tells your computer to check over all the copied files and verify that they are error free.)

When your system prompt reappears, take the distribution disk out of drive A, return it to its protective envelope, and store it in a safe place.

## INTRODUCTION . . . 4

### Backing Up files from Hard Disk

Loss of information stored on hard disk, although not likely, can be disastrous — simply because of the amount of information. Therefore, you should always keep and update floppy diskette copies of hard disk information. Make copies immediately, then store them in a safe place.

To make copies, first use the **FORMAT** command to organize a blank diskette into a filing system in which you can put disk files. Then use the **BACKUP** command to copy files from the hard disk to the formatted diskettes.

To copy *all* files from the hard disk to a formatted diskette in Drive A, type:

```
BACKUP C:\ A:/S<enter>
```

Whenever the **BACKUP** command fills a diskette, it prompts you to insert another. Follow the prompts until **BACKUP** is finished.

Often you may want to copy only those files modified since the last backup. To do so, type:

```
BACKUP *.* A:/M<enter>
```

All the files in the current directory that have been modified since last backup are copied to the diskette in Drive A. (See the MS-DOS Commands Reference Manual for an explanation of “current directory.”)

### Restoring Files to Hard Disk

If you keep a complete set of backup diskettes, restoring your hard disk system is straightforward. Simply insert a backup diskette into Drive A and type:

```
RESTORE A: C: \ /S<enter>
```

**RESTORE** copies all files from the backup diskette to the hard disk. The **RESTORE** command works only with diskettes that have been created using the **BACKUP** command.

### File Naming Conventions

A file name is the way that a unique file is identified on a disk drive. File names may have up to three parts. These are:

1. The drive name (optional).
2. The file name itself.
3. The file extension (optional).

Example: d:filename.ext

The drive name is simply the letter from “A” to “D” which identifies the disk drive containing the file you wish to create or access. Drive names must be separated from the file name itself by a colon (:). This ‘part’ of the file name is optional, and may be omitted if you are addressing the ‘logged’, i.e., currently active, disk drive. If the drive name is not used, the following colon must be omitted.

The maximum length of the file name itself is 8 characters. You may create or access your file names using upper or lower case characters, since your operating system will convert all file name entries to upper case before reading them or writing them to disk.

The file extension simply provides both you and your operating system or program with a way of identifying and handling files individually or as a group according to their function.

For example, files containing programs which can be executed by your operating system should have the extension “.COM.” On the one hand, this enables your operating system to identify which files can and cannot be opened and processed as operating system programs (or command files). On the other hand, this enables you to quickly identify all your operating system programs when you look over your disk file directory.

The essential dBASE II disk files are:

DBASE.COM  
 DBASEOVR.COM  
 DBASEMSG.TXT (the help file)

**Note:** For dBASE II, the executable program name is DBASE.COM. Since the “.COM” extension identifies the dBASE II file as a program which can be opened and processed by your operating system, you may initiate the dBASE program by simply entering “dBASE” in response to your system prompt (e.g., “A>”). It is not necessary to include the “.COM” extension in your command, since your system will assume that the called file is a “.COM” file unless otherwise instructed.

The maximum length of the file extension is 3 characters, and must be separated from the file name by a period (.). The period must be omitted when the extension is not used.

Command files written in the dBASE program language for execution by dBASE II use the extension “.PRG.” Their extension thus distinguishes them from your operating systems programs.

The other file name extensions used by dBASE II are:

DBF = database files	FRM = report form files
TXT = external files (delimited)	NDX = index files
FMT = screen format files	MEM = memory files

Your file names and extensions should not contain spaces or any of the following characters, since your operating system employs them in special ways:

/ < > . , ; : = ? \* [ ]

#### Multiple file manipulation using your system's wildcards:

The “?” and “\*” are your operating system's “wildcards”. Used in the file name portion of certain commands (e.g. copying a file with COPY), “?” and “\*” will allow you to specify several or even all the files on a disk as the object of a command's action. This is done by ‘masking’ portions of the file name specified in the command so that — from your computer's ‘point of view’ — it ‘matches’ the names of all the files you wish to affect.

## INTRODUCTION . . . 6

**Note:** "Wildcards" cannot be used when creating file names. They are only used to designate groups of existing files.

One or more "?" in the name or extension portion of a file specified as the object of a command will tell your computer to act as though the names of all files on the 'logged' disk drive have 'matching' characters in the positions in which the "?"s appear.

The "\*" tells your computer to act as though the names of all files on the logged disk drive have 'matching' characters from the position in which the \* appears in the designated file name to the end of the portion of the file name in which it is used. Separate "\*"s must be used to mask characters in the file name and the file name extension ("\*.\*)" is used to designate all the files on the logged drive.)

**Note:** Wildcards may not be used in the drive name portion of the file name.

### File Name examples:

valid names without wildcards:

A:TEST.DBF    B:FRED.LET    C:JANE.TXT    COPY.COM    TEST.PRG

invalid names:

T:TEST    FRED,MARY.DAT    A:A:READ.ME    COPY/COM

valid names with wildcards:

TE\*.DAT    B:TEST.\*    FRED?.DAT    A:\*.COM    C:DB\*.\*

For more information about file naming conventions and "wild cards," see your operating system manual.

### Typographic conventions used in this manual:

**Lowercase** in the screen representations (in squares) indicates material that you type in.

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

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

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

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

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

### Bringing up dBASE

1. Turn on your Model 2000.
2. Floppy Disk Users: Insert your dBASE II diskette into Drive A, the lower floppy disk drive.

Be sure the diskette's read/write window (oval-shaped window) points into the drive and the diskette's label faces up. After fully inserting the diskette, turn the drive latch so that it blocks the drive opening.

3. Press the RESET switch.
4. MS-DOS loads into memory. This loading takes about five seconds. Then the screen shows a startup/copyright message. The message ends with this prompt:

Enter new date:

If the screen is blank or shows an error message instead of the startup message...

- The diskette may be in backwards. Remove the diskette — even if the drive light is on — and correctly insert it.
- Adjust the video controls that are on the front of the monitor.
- Press RESET to reload MS-DOS.

If your screen is still blank, it may be because you're using a monitor other than the one in the normal configuration. To let the computer know your configuration is different, press <F12>, and without releasing <F12>, press RESET.

5. Enter the current date in the *mm-dd-yy* format.

For example, for June 14, 1984, type:

6-14-84 <ENTER>

The screen shows the time elapsed since you loaded MS-DOS and asks for the new time:

Current time is 0:00:11.59

Enter new time:

You can skip this question by pressing <ENTER>. If you want to set the time, enter it in the 24-hour format, *hh:mm:ss.cc*. (*cc* represents hundredths of a second.) Include as much or as little of the time as you want.

## **INTRODUCTION...8**

For example, type:

14:30 <ENTER>

for 2:30 p.m.

6. After you enter the time, MS-DOS displays its system prompt, which is A> if you are operating under floppy disk control or C> if you are operating under hard disk control. Your Model 2000 is now under the control of MS-DOS and is ready for use.
7. Bring up dBASE II by typing:

dBASE <enter>

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

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

It will only take a few minutes.

Additional information concerning the various facets of dBASE II may be accessed by using the HELP facility on the dBASE II System Disk. Once in dBASE II type HELP dBASE.



**(This page is intentionally left blank)**





**(This page is intentionally left blank)**





**Section I:**

How to CREATE a database .....	11	CREATE
Entering data into your new database .....	13	
Modifying data in a database .....	14	EDIT, BROWSE
Full Screen Editing Features .....	15	
An introduction to dBASE II commands and the error correction dialog .....	16	USE, DISPLAY, LIST
Expanding commands with expressions .....	17	LIST
Looking at your data records .....	20	DISPLAY
Positioning yourself in the database .....	21	GO, GOTO, SKIP
The interactive ? command .....	23	?
Adding more records to a database .....	24	APPEND, INSERT
Cleaning up a database .....	26	DELETE, RECALL, PACK
Section I Summary .....	28	

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



**(This page is intentionally left blank)**



**How to CREATE a database**

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

NAME: up to 20 characters long  
 ADDRESS: up to 22 characters long  
 CITY: up to 20 characters long  
 STATE: 2 characters long  
 ZIP CODE: 5 characters

First, type `*CREATE*`.

dBASE II responds with: ENTER FILENAME:

Enter a filename starting with a letter and up to 8 characters long (limited by MS DOS), no colons, no spaces. Since this is a file of names, let's call it something that makes sense to a human being: type `*Names*`.

When you hit `<enter>`, dBASE II creates a file called `<NAMES.DBF>`. The part of the name after the period is the MS-DOS file name extension, and is short for data base file (Section V, File Types)

In a database management system, each one of the items that we want to enter into a single related grouping is called a *field* and the grouping is called a *record* (Section V, Database Basics). In our example, each record will have 5 fields. dBASE needs to know the name of each field, what type of data it will contain, how long it is and how many decimal places if the data is numeric.

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

Field names can be up to 10 characters long, and may be entered in upper and/or lowercase. The name must start with a letter and cannot contain spaces, but can contain digits and embedded colons. Don't abbreviate any more than you have to: the computer will understand what you mean, but people might not.

The type of data is specified by a single letter: C for Character, N for Numeric and L for Logical. In this case, all fields contain character data.

## CREATING DATABASES . . . 12

Field width can be any length up to 254 characters. If the field is numeric and decimal places are specified, remember that the decimal point also takes one character position.

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

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

001     name,c,20
002     address,c,22
003     city,c,20
004     state,c,2
005     zip code,c,5
BAD NAME FIELD
005     zip:code,c,5
006     <return>
```

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

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

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

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

The <Names.DBF> database is immediately ready for data entry, so type \*y\*. Next we'll tell you how to enter the data.

**Entering data into your new database**

The record number and all the fields will be displayed starting in the upper left-hand corner of the screen, with the cursor at the first character position of the first field.

```

RECORD 00001
NAME      :
ADDRESS   :
CITY      :
STATE     :
ZIP CODE  :
  
```

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

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

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

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

## CREATING DATABASES . . . 14

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

```
♦USE Names♦  
♦APPEND♦
```

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

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

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

If you want to stop now, simply type ♦QUIT♦.

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

**Note:** Later, you'll be told how to set up FORMAT FILES (.FMT extension) that allow you to format the screen any way you want it. When a format file is being used, the fields and prompts will appear on the screen wherever you want them, rather than as the automatic listing you've just seen. With a format file, you'll also be able to use only fields that you select, rather than all the fields in the database. This applies to the CREATE, APPEND, EDIT and INSERT commands.

### Modifying data with EDIT and BROWSE

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

```
♦USE Names♦  
♦EDIT <number>♦
```

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

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

```
RECORD 00009                                DELETED  
  
NAME      :CLINKER, DUANE                    :  
ADDRESS   :789 Charles Dr.                  :  
CITY      :Los Angeles                       :  
STATE     :CA                                :  
ZIP:CODE  :90036
```

If you mark a record for deletion by using **^ctl-U^**, "DELETED" appears at the top of the screen. Pressing **^ctl-U^** again removes the word and "un-deletes" the record. If you **^LIST^** (pp. 16 and 18) or **^DISPLAY^** (pp. 16 and 20) your database, you will see an asterisk next to all records marked for deletion.

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

To exit gracefully and save the changes made so far, use **^ctl-W^**.

BROWSE FIELDS [< field list >]

is one of the most powerful dBASE II commands for editing and displaying data.

BROWSE shows up to 19 records, displaying as many fields from each record as will fit within your screen "window." **^ctl-B^** moves the window right one field; **^ctl-Z^** moves the window left.

*Fields* (not records) longer than 80 characters "wrap" around your screen so that you always see the entire field. If you specify a list of fields (separated by commas), only those fields will be displayed.

The FULL SCREEN and EDIT commands (next page) will move your cursor around within fields, field to field, and record to record. You can make any changes you want to within the database and these will be saved if you exit with **^ctl-W^**, ignored if you exit with **^ctl-Q^**.

#### Full Screen Features: (not applicable on command line)

down arrow or **ctl-X** moves cursor down to the next field.

up arrow or **ctl-E** moves cursor back to the previous field.

right arrow or **ctl-D** moves cursor ahead one character.

left arrow or **ctl-S** moves cursor back one character.

<INSERT> or **ctl-V** toggles between overwrite and insert modes.

<DELETE> or **ctl-G** deletes the character under the cursor.

<BACKSPACE> deletes the character to the left of the cursor.

<PRINT> or **ctl-P** toggles your printer ON and OFF.

**ctl-W** saves any changes made and resumes normal dBASE II operation.

**ctl-Q** quits and returns to normal dBASE II operation without making changes, even in the MODIFY mode.

**^EDIT^** **^BROWSE^** functions: (Do not use **ctl-R** or **ctl-U** in **^APPEND^** mode)

**ctl-C** writes the record to disk and advances to the next record.

**ctl-R** writes the record to disk and backs up to the previous record.

**ctl-U** toggles the record deletion mark on and off.

## CREATING DATABASES ... 16

### •BROWSE• functions:

ctrl-B pans the window right one field.

ctrl-Z pans the window left one field.

### •MODIFY• functions:

ctrl-T deletes current field, moves all the lower fields up.

ctrl-Y blanks current field to the right of cursor position, leaves all fields where they were.

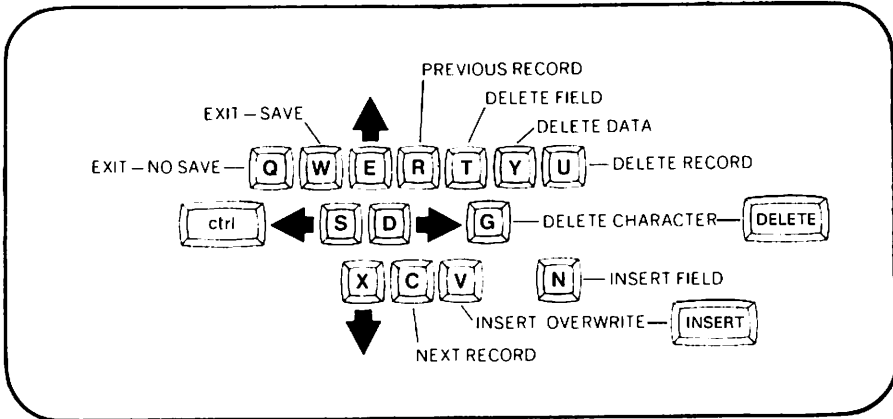
ctrl-N moves fields down one position to make room for insertion of a new field at the cursor position.

### •APPEND• functions:

ctrl-C writes the record to disk and advances to the next record.

<Enter> when cursor is at the initial position of a new record resumes normal dBASE II operation.

ctrl-Q erases the record and resumes normal dBASE II operation.



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

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

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

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

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



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

The command line is scanned and you are prompted with error messages when mistakes are detected. You get a second chance to make corrections without having to retype the entire line.

Type \*EDUT 3\*.

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

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

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

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

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

### Expanding commands with expressions and relational operators (LIST)

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

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

## CREATING DATABASES . . . 18

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

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

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

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

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

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

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

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

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

```

.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 be used to provide you with system information.

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

♦LIST FILES♦ shows the names of the database (.DBF) files on the logged-in drive. ♦LIST FILES ON <drive>♦ shows the database files on another drive (do NOT use the usual MS-DOS colon).

. use names  
. list structure  
STRUCTURE FOR FILE: NAMES.DBF  
NUMBER OF RECORDS:00008  
DATE OF LAST UPDATE:00/00/00  
PRIMARY USE DATABASE

FLD	NAME	TYPE	WIDTH	DEC
001	NAME	C	020	
002	ADDRESS	C	022	
003	CITY	C	020	
004	STATE	C	002	
005	ZIP:CODE	C	005	
**TOTAL**			00070	

DATABASE	FILES	#RCDS	LAST UPDATE
NAMES	DBF	00008	00/00/00

**Looking at data with DISPLAY**

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

```

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

```

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

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

- ♦DISPLAY ALL♦
- ♦DISPLAY Record 3♦
- ♦DISPLAY Next 4♦

```

.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\*, the optional FOR clause can be used to select specific data by using logical expressions.

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

```

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

```

BOTH \*LIST\* and \*DISPLAY\* can show you specific types of files on a drive using the MS-DOS "wild cards." \*DISPLAY FILES LIKE \*.COM ON B\*, for example, would display all the ".COM" files on drive B.

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

### Positioning commands (GO or GOTO and SKIP)

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

```

*USE Names*
*GO TOP*
*DISPLAY*
*GO BOTTOM*
*DISPLAY*
*GOTO 5*
*DISPLAY*
*8*
*DISPLAY*

```

## CREATING DATABASES . . . 22

```

.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 <number>♦ (or ♦GO <number>♦) (And you can even eliminate the GO and just specify the record number).

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

```

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

```

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

### The interactive ? command

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

Type the following:

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

```
. ? 73/3.0000
    24.33333
. ? 73.00/3
    24.33
. ? 73/3
    24
. STORE 73/3 to s
    24
```

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

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

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

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

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

**Adding more data with the APPEND and INSERT commands**

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

```
*USE Names*
*APPEND*
```

```
. use names
. append

RECORD # 00009

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

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



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

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

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

\*INSERT [BEFORE] [BLANK]\*

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

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

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

The sequence of commands is:

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

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

To exit the \*APPEND\* mode, at the beginning of a new, blank record hit <enter>.

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

With both the APPEND and INSERT commands, you'll later learn how to set up a FORMAT FILE (.FMT extension) to show only selected fields from the database and any special prompt that you might want to include.

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

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

To delete the current record, type \*DELETE\*.

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

To make the deletions conditional, expand the command to:

\*DELETE [scope] [FOR <expression> ]\*

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

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

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

To restore the records, use the following command:

\*RECALL [scope] [FOR <expression> ]\*

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

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

\*PACK\*.

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

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

To see how these commands work, type the following:

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

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

```
. list
00001 ALAZAR, PAT          123 Crater Rd.          Everett          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
```

**Section I Summary**

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

You can now \*CREATE\* a new database and start entering data in minutes.

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

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

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

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

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

```
. create
ENTER FILENAME: 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      CustNnbr.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
```



**(This page is intentionally left blank)**



## Section II:

Using expressions for selection and control .....	31	
Constants and variables .....	31	STORE
dBASE II operators .....	35	
Logical operators .....	36	
Substring logical operator .....	38	
String operators .....	39	
Changing an empty database structure .....	40	MODIFY
Duplicating databases and structures .....	41	COPY
Adding and deleting fields with data in the database .	44	COPY, USE, MODIFY
Dealing with MS-DOS and "foreign files" .....	47	COPY, APPEND
Renaming database fields .....	48	COPY, APPEND
Modifying data rapidly .....	49	REPLACE, CHANGE
Organizing your databases .....	51	SORT, INDEX
Finding the information you want .....	53	FIND, LOCATE
Getting information out of all that data .....	55	REPORT
Automatic counting and summing .....	57	COUNT, SUM
Summarizing data and eliminating details .....	57	TOTAL
Section II Summary .....	60	

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

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

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



**(This page is intentionally left blank)**





## Using expressions for selection and control

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

As you saw, they are a powerful way to extend the commands and manipulate your data quickly and easily. Many dBASE II commands can be modified in the form:

♦ <COMMAND> [FOR <expression>] ♦

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

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

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

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

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

## Constants and variables (STORE)

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

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

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

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

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

```

♦dBASE♦
♦USE Names♦
♦?'Name'♦
♦? Name♦
    
```

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

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

```

♦SKIP 3♦
♦? Name♦
    
```

```

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

Now type ♦USE♦. Since we do not specify a file name, the computer simply closes all files.

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

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

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

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

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

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

\*STORE <expression> TO <memory variable>\*

Type the following:

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

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

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

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

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

```
. 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 00084 BYTES USED
```

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

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

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

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

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

```

♦DISPLAY MEMORY♦
♦RELEASE Another♦
♦DISPLAY MEMORY♦
♦RELEASE ALL♦
♦DISPLAY MEMORY♦

```

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

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

### **dBASE II operators**

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

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

```

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

```

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

```

17/33*72.00 + 8 = 45.09           (divide, multiply then add)
17/(33*72.00000 + 8) = 0.00644    (multiply, add then divide)
17/33*(72.00 + 8) = 41.21        (divide, add then multiply)

```

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

```

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

```

Type the following:

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

```
. LIST FOR Zip:Code<='70000'
00003 DESTRY, RALPH      234 Mahogany St.      Deerfield      FL 33441
00004 EDMUNDS, JIM      392 Vacarious 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.   Tuscon        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
```

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

- ( ) : parentheses for grouping
- .NOT. : Boolean not (unary operator)
- .AND. : Boolean and
- .OR. : Boolean or
- \$ : substring logical operator  
(substring search)

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

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

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

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

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

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

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

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

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

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

. USE Names

. DISPLAY all FOR Zip:Code > '5'.AND.Zip:Code < '9'

00006	FORMAN, ED	456 Boston St.	Dallas	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 Dohney 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 what happened with the last command: all the records were displayed. If you're not familiar with logical operators, this kind of non-selective "selection" will have to be guarded against.

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

♦ <substring> \$ <string> ♦

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

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



```

. USE Names
. LIST FOR 'EE' $ Name
00007 GREEN, TERRY           567 Doheny Dr           Hollywood           CA 90044
. LIST FOR '7' $ Address
00003 CLINKER, DUANE         789 Charles Drive       Los Angeles         CA 90038
00007 GREEN, TERRY           567 Doheny Dr.          Hollywood           CA 90044
00008 HOWSER, PETER          678 Dusty Rd.           Chicago              IL 60631
. LIST FOR 'CA' $ State
00003 CLINKER, DUANE         789 Charles Drive       Los Angeles         CA 90038
00005 EMBRY, ALBERT          345 Sage Avenue         Palo Alto           CA 94303
00007 GREEN, TERRY           567 Doheny Dr.          Hollywood           CA 90044
.? 'oo' $ 'Hollywood'
.T
. go 5
. display
00005 EMBRY, ALBERT          345 Sage Avenue         Palo Alto           CA 94303
.? State $ "California"
.T

```

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

```
♦ <COMMAND> FOR 'XX' $ Address ♦
```

*String operators generate string results.*

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

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

```

♦ USE Names ♦
♦ ? Name + Address ♦
♦ ? Name - Address ♦
♦ ? 'The name in this record is ' + Name:
  - ' and the address is ' + Address ♦

```

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

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

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

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

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

**Changing an empty database structure (MODIFY)**

**Warning:** the **<MODIFY>** command will destroy your database. Please follow instructions carefully.

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

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

```

>USE MoneyOut
>LIST STRUCTURE (page 19)
>MODIFY STRUCTURE
>
(in response to the question)
    
```

```

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

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

```

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

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

### Duplicating databases and structures (COPY)

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

```

♦USE Names♦
♦COPY TO Temp♦
♦USE Temp♦
♦DISPLAY STRUCTURE♦
♦LIST♦

```

```
. 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	020	
002	ADDRESS	C	022	
003	CITY	C	020	
004	STATE	C	002	
005	ZIP:CODE	C	005	
**TOTAL**			00070	

```
. 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 90044
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 filename, the file is written over and the old data is destroyed.

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

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

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

- \*USE Names\*
- \*COPY TO Temp STRUCTURE\*
- \*USE Temp\*
- \*DISPLAY STRUCTURE\*

```

.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             020
002          ADDRESS       C             022
003          CITY          C             020
004          STATE         C             002
005          ZIP:CODE       C             005
**TOTAL**                                00070
    
```

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

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

```

.use names
.copy to temp structure field 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             020
002          STATE         C             002
**TOTAL**                                00023
    
```

**For Advanced Programmers:** COPY can also be used to give your program access to a database structure. Type:

- ◆ USE Names ◆
- ◆ COPY TO New STRUCTURE EXTENDED ◆
- ◆ USE New ◆
- ◆ LIST ◆

## EXPRESSIONS ... 44

```
. use Names
. copy to New structure extended
00006 RECORDS COPIED
. use New
. display structure
STRUCTURE FOR FILE: NEW.DBF
NUMBER OF RECORDS: 00006
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           20           0
00002      ADDRESS       C           22           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.COMD, Section VI).

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

```
♦CREATE <datafile> FROM <structurefile>♦
```

### Adding and deleting fields with data in the database

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

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

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

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

```

♦USE Names♦
♦COPY TO Temp STRUCTURE♦
♦USE Temp♦
♦MODIFY STRUCTURE♦
♦y

```

(in answer to the prompt)

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

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

```

♦APPEND FROM Names♦

```

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

```

. display structure
STRUCTURE FOR FILE TEMP.DBF
NUMBER OF RECORDS.00010
DATE OF LAST UPDATE.00/00/00
PRIMARY USE DATABASE
  FLD      NAME      TYPE      WIDTH      DEC
001      NAME      C          020
002      ADDRESS   C          022
003      CITY      C          020
004      STATE     C          002
005      ZIP CODE   C          005
006      CUSTNMBR  C          009
**TOTAL**                                00079

```

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

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

```

♦COPY TO Names♦
♦USE Names♦

```

## EXPRESSIONS . . . 46

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

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

- \*USE <oldfile>\*
- \*COPY TO <newfile> STRUCTURE\*
- \*USE <newfile>\*
- \*MODIFY STRUCTURE\*
- \*APPEND FROM <oldfile>\*
- \*COPY TO <oldfile>\*

```
. use names
. copy to temp structure
. 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	020	
002	ADDRESS	C	022	
003	CITY	C	020	
004	STATE	C	002	
005	ZIP:CODE	C	005	
006	CUSTNMBR	C	009	
**TOTAL**			00079	



### Dealing with MS-DOS and "foreign" data files (more on COPY and APPEND)

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

With 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 MS-DOS.

The System Data Format also allows dBASE II to work with data from MS-DOS files. However, *the data must match the structure of the database that will be using it.*

If we use a word processor to create a file called <NewData.TXT> we could add it to the <Names.DBF> file. NOTE: the spacing of the data must match the structure of the database. If the <NewData.TXT> file contained the following information:

FREITAG, JEAN	854 Munchkin Ave.	Houston	TX 77006
GOULD, NICOLE	73 Radnor Way	Radnor	PA 19089
PETERS, ALICE	676 Wacker Dr.	Chicago	IL 60606
GREEN, FRANK	441 Spicer Ave.	Tampa	FL 33622
(20)	(25)	(20)	(2) (5)

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

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

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

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

```
*COPY TO Temp DELIMITED*
```

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

## EXPRESSIONS ... 48

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

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

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

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

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

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

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

### Renaming database fields with COPY and APPEND

As we said earlier, ♦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 in that field will not be transferred.

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

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

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

```

. use names
. copy to temp sdf
00015 RECORDS COPIED
. modify structure
MODIFY ERASES ALL DATA RECORDS...PROCEED?
(Y/N) Y

. append from temp.TXT sdf
00015 RECORDS ADDED

```

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

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

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

### Modifying data rapidly (REPLACE, CHANGE)

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

```

*REPLACE [scope] < field> WITH < exp> [, < field> WITH < exp> ,... ]
                                [FOR < expression> ]

```

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

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

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

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

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

## EXPRESSIONS ... 50

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

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

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

```
. 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.F.
00003      1009      12829      7      0.17      1.19.F.
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. Data from memory variables in your program is frequently used to fill in the blank fields.

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

\*CHANGE [scope] FIELD <list> [FOR <expression>]\*

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

Once you have looked at all the listed fields within a record, you are represented 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 all field custcode

RECORD: 00001

CUSTCODE
CHANGE? * * (ENTER A SPACE TO CHANGE AN EMPTY FIELD)
TO      1001

CUSTCODE: 1001
CHANGE? <enter>

RECORD: 00002

CUSTCODE
CHANGE?

```

**Reminder:** If you want to make changes to a relatively small number of records, the BROWSE command (page 16) may be what you need.

### Organizing your databases (SORT, INDEX)

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

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

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

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

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

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

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

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

```
. use names
. sort on name to temp
. SORT COMPLETE
. use temp
. list
```

00001	ALAZAR, PAT	123 Crater Rd.	Everett	WA 98206 1001
00002	BROWN, JOHN	456 Minnow Pl.	Burlington	MA 01730 1002
00003	CLINKER, DUANE	789 Charles Dr.	Los Angeles	CA 90036 1003
00004	DESTRY, RALPH	234 Mahogany St.	Deerfield	FL 33441 1004
00005	EMBRY, ALBERT	345 Sage Avenue	Palo Alto	CA 94303 1005
00006	FORMAN, ED	456 Boston St.	Dallas	TX 75220 1006
00007	FREITAG, JEAN	854 Munchkin Ave.	Houston	TX 77006 1011
00008	GOULD, NICOLE	73 Radnor Way	Radnor	PA 19089 1012
00009	GREEN, FRANK	441 Spicer Ave.	Tampa	FL 33622
00010	GREEN, TERRY	567 Doheny Dr.	Hollywood	CA 90044 1007
00011	HOWSER, PETER	678 Dusty Rd.	Chicago	IL 60631 1008
00012	INDERS, PER	321 Sawtelle Blvd.	Tucson	AZ 85702 1009
00013	JENKINS, TED	210 Park Avenue	New York	NY 10016 1010
00014	PETERS, ALICE	676 Wacker Dr.	Chicago	IL 60606

```
. copy to names
00014 RECORDS COPIED
```

**Note:** You may not SORT a database to itself. Instead, sort to a temporary file, then \*COPY\* it back to the original file name after you've confirmed the data.

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

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

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

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

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

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

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

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

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

\*USE < database name> INDEX < index filename>\*

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

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

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

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

### Finding the information you want (FIND, LOCATE)

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

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

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

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

```

.use names index zips

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

.find 9
.display
00003 CLINKER, DUANE       789 Charles Dr.          Los Angeles       CA 90036 1003

.display next 3
00003 CLINKER, DUANE       789 Charles Dr.          Los Angeles       CA 90036 1003
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 *identical* key that you are looking for, dBASE II displays NO FIND.

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

When looking for specific kinds of data, use

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

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

If you want to search the entire database between your pointer and the end of the file, you do not have to specify the scope. If you do want to search the entire file, either specify "all", or first position the pointer at the start of the file (GO TOP). If you are looking for data in a character field, the data should be enclosed in single quotes. Type the following:

```

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



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

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

```

♦CONTINUE♦
♦CONTINUE♦
♦CONTINUE♦

```

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

```

. use names
. locate for Name='GOU'
RECORD: 00008
. display
00008  GOULD, NICOLE           73 Radnor Way           Radnor           PA 19089 1012

. locate for Zip Code > '8' and Name < 'G'
RECORD: 00001
. display Name, Zip Code
00001  ALAZAR, PAT           98208

. continue
RECORD: 00003
. continue
RECORD: 00005
. continue
END OF FILE ENCOUNTERED

```

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

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

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

```

♦SET EJECT OFF♦
♦USE < database > ♦
♦REPORT♦

```

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

## EXPRESSIONS ... 56

You can try this with the files you've created on the demonstration disk, but the < NAMES > and < ORDERS > databases that we've used as examples so far don't have enough data in them to really show you how powerful dBASE II can be. For our examples from here on we will be using < MoneyOut.DBF > and other databases that are part of an existing business system.

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

```
. 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          < enter >
```

PAGE NO. 00001

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

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

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

By typing

```
*REPORT FORM JobCosts FOR Job:Nmbr='770' *
```

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

. REPORT FORM JobCosts FOR Job:Nmbr=770

PAGE NO 00001

COST SUMMARY			
DATE	SUPPLIER	DESCRIPTION	AMOUNT
810113	LETTER FONT	TYPE	177.00
810113	ABLE PRINTER	MAILER	805.00
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**			1,840.10

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

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

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

**Automatic counting and summing (COUNT, SUM)**

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

For counting use:

\*COUNT [scope] [FOR conditions] [TO memory variable]\*

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

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

To get totals, use:

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

## EXPRESSIONS ... 58

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

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

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

### Summarizing data and eliminating details (TOTAL)

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

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

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

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

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

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

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

\*COPY TO Temp FIELDS Job:Nmbr, Amount\*

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

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

```
. USE MoneyOut
. INDEX ON Job Nmbr TO Jobs
0093 RECORDS INDEXED
. USE MoneyOut INDEX Jobs
. TOTAL ON Job.Nmbr TO Temp FIELDS AMOUNT FOR JOB Nmbr> 699.
                                         AND Job Nmbr< 800

00028 RECORDS COPIED
. USE Temp
. LIST
00011  810128  3145  SML  778  138.00  LETTER FONT  TYPE
810129  2633      0.00   0
00012  810128  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)
```

**Section II Summary**

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

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

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

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

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

### Section III:

Setting up a command file (writing your first program) .....	61	MODIFY COMMAND <file>
Making choices and decisions .....	62	IF..ELSE..ENDIF
Repeating a process .....	64	DO WHILE..
Procedures (subsidiary command files) .....	65	DO <file>
Entering data interactively during a run .....	65	WAIT, INPUT, ACCEPT
Placing data and prompts exactly where you want them .....	66	@..SAY..GET
A command file that summarizes what we've learned .....	69	
Working with multiple databases .....	72	SELECT PRIMARY/ SECONDARY
Generally useful system commands and functions .....	73	
A few words about programming and planning your command files .....	74	

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

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

- Sequence
- Choice/Decision
- Repetition
- Procedures

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

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



**(This page is intentionally left blank)**





### Setting up a command file (writing your first program)

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

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

To set up a command file, you list the commands you want performed in a MS-DOS file with a <.PRG> extension to its name, using a text editor or word processor.

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

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

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

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

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

```
♦MODIFY COMMAND Test♦
```

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

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

```
♦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.'♦
```

## COMMAND FILES ... 62

When you're finished, use `clt-W` to get back to the dBASE II prompt. Now type:

```
. *Do Test*
```

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

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

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

**Tip:** You may want to rename the main dBASE file to `<DO.COM>`, so that you can type `*DO <filename>*` whether you're in your system or in dBASE II. To do this with MS-DOS, type: `A> *REN DO.COM=dBASE.COM*`

### Making choices and decisions (IF..ELSE)

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

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

```
IF Job:Nmbr = '730' .AND. Amount > 99.99;
  .OR. Supplier = 'MAGIC TOUCH';
  .OR. Bill:Date > '791231'
  do this command
  [cmd2]
  [...]
ENDIF
```

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

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

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

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

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

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

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

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

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

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

**Repeating a process (DO WHILE..)**

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

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

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

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

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

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

In this example, if there is data in the <Item> field, the computer performs whatever instructions are in another command file called ProcessA.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, the computer proceeds through the DO WHILE instructions again. When the counter passes 10, the computer skips the loop and performs the next instruction after the ENDDO.

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

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

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

**Procedures (subsidiary command files)**

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

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

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

```

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

```

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

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

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

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

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

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

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

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. The following commands can do this.

♦WAIT [TO memory variable]♦

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

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

♦INPUT ['prompt'] TO memory variable♦

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

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

♦ACCEPT ['prompt'] TO memory variable♦

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

**Tips on which to use when:**

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

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

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

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

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

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

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

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

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

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

```
♦ERASE♦
♦@ 20,30 SAY 'What?'♦
♦@ 5,67 SAY 'Here...'♦
♦@ 11,11 SAY "That's all."♦
♦@ 20, 0♦
♦@ 5, 0♦
♦@ 11,16♦
```

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

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

Type the following in dBASE II:

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

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

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

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

## COMMAND FILES . . . 68

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

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

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

```
♦READ♦
```

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

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

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

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

Our <Names> file is not a good example, but we can use it to show how to selectively get data into a database with a large structure.

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

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



```

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

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

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

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

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

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

#### A command file that summarizes what we've learned

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

```

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

```

SET TALK OFF
USE MoneyOut
ERASE
    
```

## COMMAND FILES . . . 70

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

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

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

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

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

```

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

```

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

```

STORE 1 TO X
DO WHILE X < 100
    STORE X + 1 to X
ENDDO
ERASE
RETURN

```

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

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

1. Using ERASE frequently is good housekeeping that's easy to do.
2. Using indentation helps make the operation of the program clearer. That's also why we used upper- and lowercase letters. The computer sees them all as uppercase, but this way is much easier for us humans.
3. The "?" can be used to space lines on the display and to show character strings (in quotes or brackets).
4. The WAIT command waits for a *single* character before letting the program move on. The input then *must* be treated as a character, the way we did in the nested IF's by putting quotes around the values we were looking for.

5. The INPUT command waits for and accepts any data type, but characters and strings must be in single or double quotes or square brackets. When you have an apostrophe in your message, use the double quote or square brackets to define the string or the computer gets confused.
6. You don't have to predefine variables. Just make up another name whenever you need one (up to a maximum of 64 active at any one time).
7. Logical values can be treated in shorthand. "IF Decision" in the program worked as if we had said: "IF Decision = T".
8. The RETURN at the end of the program isn't necessary, but was tacked on because you *would* need it if this were a sub-procedure in another command file. That's how the computer knows that it should go back where it came from, rather than just quitting.

### Working with multiple databases (PRIMARY, SECONDARY SELECT)

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

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

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

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

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

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

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

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

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

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

### Generally useful system commands and functions

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

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

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

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

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

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

RENAME <oldfile> TO <newfile> changes file names in the MS-DOS directory. *Do NOT try to rename files in USE.*

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

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

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

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

**A few words about programming and planning your command files**

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

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

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

**Briefly, here's the approach:**

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

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

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

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

You might write your program outline like this:

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

Adding a bit more detail, it looks like this:

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

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

```

USE CostBase
* Print a hardcopy summary for December, 1980.
REPORT FORM Summary FOR Bill.Date >= '801201' .AND:
    Bill.Date <= '801231' TO PRINT
GOTO TOP
DO WHILE .NOT. EOF
    IF Check.Nmbr=' '
        DO WriteCheck
        DO Update
    ENDIF
SKIP
ENDDO

```

- \* Go to the first record
- \* Repeat for the entire file
- \* If the invoice isn't paid,
- \* write a check, then
- \* update the records
- \* Go to the next record

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

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

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

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

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



**(This page is intentionally left blank)**





**Section IV:**

Expanding your control with functions .....	77	
Changing dBASE II characteristics and defaults .....	80	SET..
Merging records from two databases .....	82	UPDATE
JOINING entire databases .....	82	JOIN
Full screen editing and formatting .....	83	SET FORMAT TO
		SCREEN @..SAY..GET..
		PICTURE..
		SET FORMAT TO PRINT
Formatting the printed page .....	85	
@..SAY..USING..		
Setting up and printing a Form .....	86	
Time to regroup .....	87	

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

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



**(This page is intentionally left blank)**



### Expanding your control with functions

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

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

**Note:** the parentheses shown below *must* be used.

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

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

!(*< variable/string >*)

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

TYPE(*< expression >*)

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

INT(*< variable/expression >*)

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

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

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

The integer function can also be used to round a value to any number of decimal places. INT(value\*10 + 0.5)/10 rounds a value to the nearest decimal place because of the

## FUNCTIONS . . . 78

order of precedence of operations (parentheses, then integer, then divide). To round to two places, use "100" in place of the "10"s. For 3 places, use "1000," etc.

VAL(< variable/string/substring > )

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

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

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

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

LEN(< variable/string > )

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

\$( < expression/variable/string > , < start > , < length > )

is the *substring function*. It selects characters from a string or character variable, starting at the specified position and continuing for the specified length. The <length> must be a string literal when used with the INDEX on SAY commands.

As an example, if we had a variable called <Date> whose value was '810823', the function \$(Date.5.2) would give us '23'. To convert these numerals to a number, we could use VAL\$(Date.5.2).

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

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

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

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

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

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

CHR(< number >)

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

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

&

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

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

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

FILE(< "filename" variable expression >)

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

TRIM (< string >)

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

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

RANK (< string >)

returns the decimal value of the first character of a string. This function corresponds to the ASC function common in many versions of BASIC.



SET DEBUG	<u>ON</u> <u>OFF</u>	sends output from the ECHO and STEP commands to the printer only sends ECHO and STEP output to the screen
SET BELL	<u>ON</u> <u>OFF</u>	enables bell when field is full disables bell
SET COLON	<u>ON</u> <u>OFF</u>	uses colons to delimit input variables on the screen disables the colons
SET CONFIRM	<u>ON</u> <u>OFF</u>	waits for <enter> before leaving a variable during full screen editing leaves the variable when the field is full
SET CARRY	<u>ON</u> <u>OFF</u>	carries data from the previous record forward to the new record when in APPEND shows a blank record in APPEND mode
SET INTENSITY	<u>ON</u> <u>OFF</u>	enables dual intensity for full screen operations disables dual intensity
SET LINKAGE	<u>ON</u> <u>OFF</u>	permits databases to be linked for display with up to 64 fields and up to 2000 bytes per displayed record. The P. or S. prefix must be used when field names are similar in both databases disables linkage
SET EXACT	<u>ON</u> <u>OFF</u>	requires that all characters in a comparison between two strings match exactly allows different length strings. E.g., 'ABCD' = 'AB' would be True (Also affects FIND command)
SET ESCAPE	<u>ON</u> <u>OFF</u>	allows the <escape> key to abort command file execution disables the <escape> key

♦SET ALTERNATE TO <filename> ♦creates a file with a .TXT extension for saving everything except full-screen displays that goes to your CRT screen. To start saving, type ♦SET ALTERNATE ON♦.

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

To stop, type♦ SET ALTERNATE OFF♦. This also terminates when you QUIT dBASE II.

**Merging records from two databases (UPDATE)**

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

```

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

**Note:** Both databases must be presorted or indexed on the "key" field if the optional RANDOM command is not used.

Without the RANDOM command, both files are "rewound" to the beginning, then key fields are compared. If they are identical, then data from the FROM data base is either *added* numerically to data in the USE file, *or* is used to *replace* data in the use file for the fields specified in the field list. When "key" fields do not match, those records are skipped. This command can be used to keep inventory updated, for example.

If the fields in both databases use the same names, you can simply list the fields in which data is to be replaced. If the databases use different field names, you can use the second form of the REPLACE option to specify which field in the USE database is to be replaced by which field in the FROM database.

If RANDOM is used, the database being updated must be indexed on the "key" field, but the FROM <file> records can be in any order. As each record is read from the FROM <file>, a FIND command locates the correct record in the database being updated.

**JOINing entire databases**

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

```

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

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

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



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

To use the command, use this sequence of instructions:

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

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

**Full screen editing and formatting (TEXT/ENDTEXT, @..SAY..GET..PICTURE, .FMT files)**

To display or print a large block of text, the text can be bracketed like this:

**TEXT**

Any text you want can be entered here and will be sent directly to the screen or printer without any command processing. The only exception is; lines that are continued using the semicolon. Processing resumes after the command: ENDTEXT

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

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

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

## FUNCTIONS . . . 84

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

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

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

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

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

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

```
@ 5.1 SAY "Today's date is" GET Date PICTURE '99/99/99'
```

would display:

```
Today's date is:  /  /  :
```

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

The GET function symbols are:

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

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

If you use only @...SAY...GET commands and comments (preceded by an asterisk), you can save this as a format file with an .FMT extension. You can then use these format files for custom input screens for your databases, rather than using the dBASE II default which

simply lists all the fields in the database. The .FMT format file can include special instructions, etc. that help the operator enter the correct data, and the names of only the fields in which data is to be entered.

To use a .FMT format file, type:

```
SET FORMAT TO < filename >
```

Now when you use the APPEND, EDIT or INSERT commands, the format in the named file will be displayed on the screen.

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

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

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

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

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

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

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.

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

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

The command @ 10.50 SAY Hours \*Rate USING '\$\$\$\$\$.99' could be used in both the screen and the printer modes since it has no GET phrase. For Hours=8 and Rate=12.73, it would print or display \$\$\$101.84, useful for printing checks that are more difficult to alter.

**Setting up and printing a form**

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

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

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

```
ERASE
SET TALK OFF
STORE "          " TO MDate
STORE T TO NoDate
DO WHILE NoDate
  @ 5,5 SAY "Set date MM/DD/YY" GET MDate PICTURE "99/99/99"
  READ
  IF VAL$(MDate,1,2) < 1:
    .OR. VAL$(MDate,1,2,) > 12:
    .OR. VAL$(MDate,4,2,) < 1:
    .OR. VAL$(MDate,4,2,) > 31:
    .OR. VAL$(MDate,7,2,) < > 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 above first sets the value of MDate to 8 blanks, then the @..SAY command displays:

Set date MM/DD/YY:    /    /    :

When the date is entered, it is checked by the IF to see whether the month is in the range 1-12, day is in the range 1-31, and year=83. This is done in three steps:

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

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

The printout for the check itself could be the next portion of the program. Using the measurements of our checks, this is the list of commands:

```
@ 8.3 SAY Script* A character variable that prints the amount in script. This is filled in by
    * another procedure called Chng2Script. We stubbed this for now like this:
    *   STORE 'Script Stub' TO Script
    *   RETURN
@ 11.38 SAY Vendr:Nmbr
@ 11.50 SAY MDate
@ 11.65 SAY Amount
@ 13.10 SAY Vendor
@ 14.10 SAY Address
@ 15.10 SAY City:State
@ 15.35 SAY ZIP
@ 17.10 SAY Who
```

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

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

### Time to regroup

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

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

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

You might want to check through the command files in the example programs. Most of the dBASE II commands have been used, and the files work the way they are set up. Each program is intended to have a useful purpose now, and is well-documented so you can learn the purpose of each. Also, each is intended to be modified as you grow and learn in dBASE. The first command file is usually the main menu for the system, with sub-files selected by pressing a number.

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

## FUNCTIONS . . . 88

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

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

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

Comments are sprinkled throughout the files for documentation, although in many cases the programs are almost self-documenting because so many of the dBASE II commands are similar to English equivalents. You are invited to play with these programs at your leisure, using the rules and techniques we have already discussed.



**(This page is intentionally left blank)**



**(This page is intentionally left blank)**





## Section V:

Database Basics .....	89
A brief introduction to database organization .....	90
dBASE II Records, Files and Data Types .....	91
dBASE II OPERATIONS SUMMARY .....	94
dBASE II FUNCTION SUMMARY .....	95
dBASE II COMMAND SUMMARY .....	96
Commands grouped by what you want done .....	100
100 File structure	
100 File operations	
102 Organizing database	
102 Combining databases	
102 Editing, updating, changing data	
103 Using variables	
104 Interactive input	
104 Searching	
104 Output	
105 Programming	



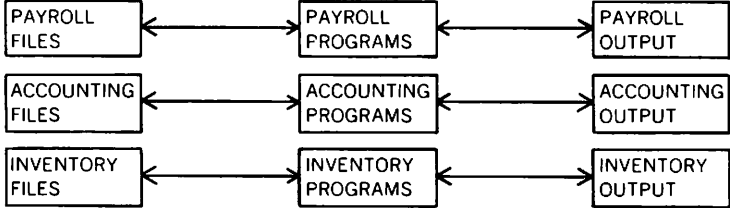
**(This page is intentionally left blank)**



**Database Basics**

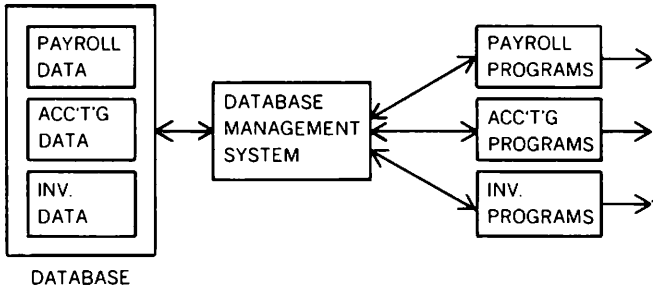
A *database management system (DBMS)* like *dBASE II* is considerably different from a file handling system.

A *file handling system* is usually configured like this:



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

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



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

To generate a new processing system in a file handling system, a new program and new files must be set up. Using a DBMS, a new access program is written, but the data does not have to be restructured: the DBMS takes care of it.

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

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

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

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

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

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

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

**A brief introduction to database organization**

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

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

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

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

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

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

*INDEXING* is a way around these problems.

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

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

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

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

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

### **dBASE II Capacities**

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

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

## **ORGANIZATION...91a**

### **Determining Disk Space**

It is often useful to determine how much free storage space is left on a diskette. This can be accomplished by using your CHKDSK.COM operating system utility program.

To discover the free space remaining on a 'logged' (active) disk drive containing the CHKDSK program, simply enter:

```
CHKDSK <enter>
```

in response to your system prompt.

Your system will then report the amount of space available on the disk, the amount of space occupied by the files on disk, and the amount of free and occupied space in your system's memory. All space is reported in "xxxx" number of bytes. (One byte is roughly equivalent to one character of information.)

While 'logged' onto a drive containing a CHKDSK program, you may discover the same information for any other disk drive by simply including the name of the drive you wish to 'check' in the command.

For example, CHKDSK B: <cr> tells your system to present you with the information for the disk in drive B. (The ":" must be included after the drive name.)

**Note:** You may discover the amount of space occupied by individual files on any disk drive by simply asking your system to present you with the file directory for that drive.

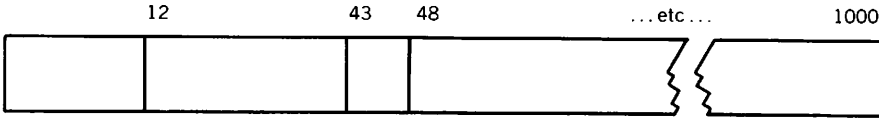


(This page is intentionally left blank)



## ORGANIZATION . . . 92

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



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

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

Invoice Number	Supplier	Description	Amount	Job Number
1      9 10	28 29	43 44	51 52	58

### Data Types

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

#### Character

All the printable ASCII characters, including the integers, symbols and spaces.

#### Numeric

Positive and negative numbers as large as  $1.8 \times 10^{63}$  down to numbers as small as  $1.0 \times 10^{-63}$ . Accuracy is to ten digits, or down to the penny for dollar amounts as high as \$99,999,999.99.

#### Logical

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

### Field Names

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



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

**Tip:** Use as many characters as it takes to make the name meaningful. 'Job:Nmbr' is a lot better than 'No.' and infinitely better than 'J'. Using a maximum of nine characters will make handling memory variables much easier (discussed later).

**Another tip:** Once you get into setting up Command files, You'll find it useful to use capital letters for words that dBASE II understands and upper and lowercase for fields, variables and other items that you control. You'll appreciate this the first time you go back into a command file to make changes.

### **dBASE II File Types**

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

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

#### **DBF Database files:**

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

#### **FRM Report form files**

These files are automatically created by dBASE II when you go through the REPORT dialog. They contain headings, totals, column contents, etc. They can be modified using a word processor or text editor, but we recommend not using this practice: make your changes using dBASE II.

#### **PRG Command files**

These files contain a sequence of dBASE II statements to perform frequently-used functions, and can be as complex as a complete payroll system. These are created using a text editor or word processor or MODIFY COMMAND.

#### **NDX Index files**

These are automatically created by the INDEX command. Indexing provides very rapid location of data in larger databases.

## ORGANIZATION . . . 94

### .MEM Memory files

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

### .TXT Text output files

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

### .FMT format files

These files may only contain @...SAY statements and comments preceded by an asterisk. Use them to format screens with the APPEND, INSERT and EDIT commands.

## DBASE II OPERATIONS SUMMARY

### *Arithmetic Operators* (generate arithmetic results: p. 35)

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

### *Relational Operators* (generate logical results: p. 35)

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

### *Logical Operators* (generate T/F logical results: p. 36)

( )	: parentheses for grouping
.NOT.	: Boolean not (unary operator)
.AND.	: Boolean and
.OR.	: Boolean or
\$	: substring logical operator (p. 38) (is string1 in string2?)

### *String Operators* (generate string results: p. 39)

+	: string concatenation (joining)
-	: string concatenation with blank shift-right

**dBASE II FUNCTION SUMMARY**

# record number (p. 73)

\* deleted record (p. 73)

EOF end of file (p. 74)

!( < variable/string > ) convert to uppercase (p. 77)

TYPE(< expression > ) data type (p. 77)

INT(< variable/expression > ) integer function (p. 77)

VAL(< variable/string/substring > ) string to integer (p. 78)

STR(< expression/variable/number > . < length > . < decimals > ) integer to string (p. 78)

LEN(variable/string) string length (p. 78)

\$(expression/variable/string> . < start > . < length > ) substring select (p. 78)

@(< variable1/string1 > . < variable2/string2 > ) substring search (p. 78)

CHR(< number > ) number to ASCII (p. 79)

&(< variable > ) macro substitution (p. 79)

FILE("< filename"/var/exp > ) file exists? (p. 79)

TRIM (< string > ) trailing blanks (p. 80)

RANK(< string > ) ascii value of character (p. 80)

**dBASE II COMMAND SUMMARY**

The following abbreviations are used in this summary:

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

The symbols < . . > bracket items that are to be specified by the user. Square brackets [ . . ] enclose optional items. In some cases, options are nested (themselves have other options).

? < exp [,list] >

Display an expression (or list separated by commas) (p. 23)

@ < coord > [ SAY < exp > [ USING 'picture' ] ] [ GET < var > [ PICTURE 'picture' ] ]

Format console screen or printer output (p. 67)

ACCEPT ['prompt'] TO < var >

Input a character string from the console, no quotes (p. 65)

APPEND [BLANK]

APPEND FROM < filename > [ SDF ] [ FOR < exp > |  
| DELIMITED ] [ FOR < exp > ]

Add data to a database (pp. 25, 48, 68)

CANCEL

Abort a command file execution

CHANGE [scope] FIELD < list > [ FOR < exp > ]

Make multiple changes to a database (p. 50)

CLEAR

Reset dBASE data files and memory variable environment (p. 73)

CONTINUE

Continue a LOCATE command (p. 55)

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

Copy data from a database to another file (pp. 42, 47, 48)

COPY TO < filename > STRUCTURE EXTENDED

Creates a new DBF file whose records show the structure of the old file. (See also  
CREATE < newfile > FROM < oldfile > ) (p. 42)

COUNT [scope] [ FOR < exp > ] [ TO < var > ]

Counts records that satisfy some condition (p. 57)

CREATE |< filename> |

**Make a new database (p. 11)**

CREATE < newfile> FROM < oldfile>

**Creates < newfile> with structure determined by the data in the records of < oldfile>. (See also COPY STRUCTURE EXTENDED) (p. 43)**

DELETE [scope] |FOR < exp> |

**Mark specified records for deletion (p. 26)**

DELETE FILE < filename>

**Erase a file from the system (p. 26)**

DISPLAY [scope] |FOR < exp> | [OFF]

**Show data based upon request (pp. 16, 20)**

DISPLAY [scope] |< field> |.list|

**Shows only the selected field(s) (p. 20)**

DISPLAY STRUCTURE

**Show structure of the database in USE (p. 21)**

DISPLAY MEMORY

**Show the contents of the memory variables (p. 32)**

DISPLAY FILES [ON disk drive]

**Show a disk directory (p. 21)**

DISPLAY STATUS

**Show current files open, index files and keys, and all SET parameters.**

DO < filename>

**Execute a command file (p. 62)**

DO WHILE < exp>

**Perform a group of commands repeatedly (p. 64)**

EDIT

**Alter the data in a database (p. 14)**

EDIT |number|

**Presents a specific record for editing (p. 14)**

EJECT

**Do a form feed on the printer**

ELSE

**Alternate execution path in an IF command (p. 62)**

## ORGANIZATION . . . 98

ENDDO

Terminator for DO WHILE command (p. 64)

ENDIF

Terminator for an IF command (p. 62)

ENDTEXT

Terminator for a TEXT command (p. 83)

ERASE

Clear console screen (pp. 17, 67)

FIND < key >

Locate a record in an indexed database based upon key value (no quotes needed for character keys) (p. 53)

GO or GOTO [RECORD], or [TOP], or [BOTTOM], n

Position to a given place in a database (p. 22)

HELP [< command verb > |

Give a short explanation of a dBASE command

IF < exp >

Conditional execution command (p. 62)

INDEX ON < key > TO < filename >

Create an index file for the database in USE (p. 52)

INPUT ['prompt'] TO < var >

Accept user inputs into memory variables. User prompt string is optional (p. 66)

INSERT [BEFORE]

[BLANK]

Add a new record to a database among other records (p. 25)

JOIN TO < filename > FOR < exp > [FIELDS < list > |

Create a database composed of matching records from two other databases (p. 82)

LIST

Show data records (pp. 16, 17)

LOCATE [scope] [FOR < exp > |

Find the record that matches a condition (p. 54)

LOOP

Escape mechanism for DO WHILE groups (p. 64)

NOTE or \*

A command file comment that is not displayed when the command file is run (p. 73)

MODIFY COMMAND <filename>

Permits modification of a file directly from dBASE II (p. 73)

MODIFY STRUCTURE

Alter the structure of a database. Destroys all data in the database (p. 40)

PACK

Eliminates records marked for deletion (p. 26)

QUIT

Terminate dBASE

READ

Enter full screen editing of a formatted screen. Accepts data into GET commands (p. 68)

RECALL [scope] [FOR <exp>]

Unmark records that have been marked for deletion (p. 26)

RELEASE [<var> [,list]] or [ALL]

Eliminate unwanted memory variables (p. 34)

REMARK

A comment that is shown on the screen when the command file is run (p. 73)

RENAME <oldfile> TO <newfile>

Give a file a new name (p. 73)

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

Alter data in a database. Make sure that you have a backup, because dBASE II will do precisely what you ask it to do, even if it's not exactly what you had in mind (p. 49)

REPORT [scope] [FORM <filename>] [TO PRINT] [FOR <exp>]

Generate a report (p. 56)

RESET

Tell MS DOS that a diskette swap may have occurred

RESTORE FROM <filename>

Remember SAVED memory variables. Destroys all existing memory variables

RETURN

Terminate a command file and return to calling file

SAVE TO <filename>

Write memory variables to a file for future use

## ORGANIZATION . . . 100

SELECT [PRIMARY] or [SECONDARY]  
Switch working areas (p. 72)

SET parameter [ON], or [OFF], or [TO < condition, filename> ]  
Dynamically reconfigure dBASE operation (p. 80)

SKIP ± < exp/number >  
Move forward or backwards in the database (p. 22)

SORT ON < key > TO < filename > [ASCENDING]  
[DESCENDING]  
Generate a database that is sorted on a field (p. 51)

STORE < exp > TO < var >  
Place a value into a memory variable (p. 33)

SUM [scope] < field [,list] > [TO < var [,list] > ] [FOR < exp > ]  
Total fields in a database (p. 57)

TEXT  
Display a block of text without special formatting until ENDTEXT encountered. (p. 83)

TOTAL TO < filename > ON < key > [FIELDS < field > [,list] >  
Generate a database with sub-totals for records (p. 58)

UPDATE FROM < filename > ON < key > [ADD < field [,list] > ]  
[REPLACE < field [,list] > ]  
Modify a database with data from another database (p. 82)

USE < filename > [INDEX < filenames > ]  
Open a database file for future operations (p. 16)

USE  
Close all previously opened database files.

WAIT [TO < var > ]  
Pause in program operation [for input] (p. 66)

## dBASE II COMMANDS GROUPED FUNCTIONALLY

### FILE STRUCTURE:

CREATE  
Defines an entirely new file structure

CREATE < newfile > FROM < oldfile >  
Creates a new file whose structure is described in the records of the old file



USE <oldfile>

COPY TO <newfile> STRUCTURE

These two commands combined create a new file with the same structure as an old file

USE <oldfile>

COPY TO <newfile> STRUCTURE EXTENDED

Create a new file that contains the structure of the old file as data

CREATE <newfile> FROM <oldfile>

Creates a new file whose structure is defined by the records in the old file

DISPLAY STRUCTURE

LIST STRUCTURE

Both show the structure of the file in USE

MODIFY STRUCTURE

Changes file names, sizes, and overall structure, but destroys data in the database

**TO CHANGE STRUCTURE WITH DATA IN THE DATABASE:**

USE <oldfile>

COPY TO <newfile>

USE <newfile>

MODIFY STRUCTURE

APPEND FROM <oldfile>

COPY TO <oldfile>

USE <oldfile>

DELETE FILE <newfile>

**TO RENAME FIELDS WITH DATA IN THE DATABASE:**

USE <oldfile>

COPY TO <newfile> SDF

MODIFY STRUCTURE

APPEND FROM <newfile> .TXT SDF

DELETE FILE <newfile>

**FILE OPERATIONS:**

USE <filename>

Opens a file

USE <newfile>

Closes the old file, opens a new file

USE

Closes all files

## ORGANIZATION . . . 102

RENAME <oldname> TO <newname>

Must NOT rename an open file

COPY TO <filename>

Creates a backup copy

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), can use LIST instead

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

Shows other types of files on drives

QUIT

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

### ORGANIZING DATABASES:

SORT ON <key> TO <newfile>

INDEX ON <key> TO <newfile>

Can use multiple keys for both commands

### Combining Databases

COPY TO <newfile>

Creates a duplicate of the file in USE

APPEND FROM <otherfile>

Adds records to the file in USE

UPDATE FROM <otherfile> ON <key>

Adds to totals or replaces data in the file in USE. Both files must be sorted on the <key>.

JOIN

Creates a third file from two other files

### EDITING, UPDATING, CHANGING DATA:

DISPLAY, LIST, BROWSE

Let you examine the records

**DELETE**

Marks record so it is not used

**RECALL**

Unmarks record

**PACK**

Erases deleted records

**EDIT**

Lets you make changes to specific records

**REPLACE < field WITH data>**

Global replacement of data in fields, can be conditional as with most dBASE II commands

**CHANGE..FIELD**

Edit based on field, rather than record

**@ < coord> GET < var>****READ**

Displays the variable, lets you change it

**INSERT [BEFORE][BLANK]**

Inserts a record in a database

**UPDATE FROM < otherfile> ON < key>**

Adds to totals or replaces data in file in USE from another file

**MODIFY COMMAND < filename>**

Allows changes to your command files without having to go through your text editor

**USING VARIABLES:**

(Allowed up to 64 memory variables plus any number of field names.)

**LIST MEMORY, DISPLAY MEMORY**

Both show the variables, their data types and their contents, returns the contents of a character memory variable (i. e., provides a *literal* character string)

**STORE < value> TO < var>**

Sets up or changes variables

**RELEASE < var>**

Cancels the named variable

**SAVE MEMORY TO < filename>**

Stores memory variables to the named file (with .MEM extension)

## ORGANIZATION . . . 104

RESTORE FROM < filename >

Reads memory variables back into memory (destroys any other existing memory variables)

### INTERACTIVE INPUT:

WAIT

Stops screen scrolling, continues with any key

WAIT TO < var >

Accepts character to memory variable

INPUT ['prompt'] TO < var >

Accepts any data type to a memory variable (creates it if it did not exist), character input must be in quotes

ACCEPT ['prompt'] TO < var >

Same as INPUT, but no quotes around character input

@ < coord > SAY ['prompt'] GET < var > [PICTURE]

READ

Displays memory variable, replaces it with new input

### SEARCHING:

SKIP [+ < exp > |

Moves forward or backward a specific number of records

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

Move you to a specific record, the first record, or the last record in the database

FIND < str >

Works with indexed file in USE, very fast

LOCATE FOR < exp >

CONTINUE

Searches entire database

### OUTPUT:

?. DISPLAY, LIST

Show expressions, records, variables, structures

REPORT [FORM < formname > |

Creates a custom format for 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)

TEXT

Displays a block of text without special formatting with ? or @ <coord> SAY command.

**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

DO CASE

CASE <expression>

<commands>

CASE <expression>

<commands>

OTHERWISE

<commands>

Multiple choices.

ENDCASE



**(This page is intentionally left blank)**



## PART B CONTENTS

1.0	Using dBASE .....	1
2.0	System Requirements .....	4
3.0	dBASE Files .....	5
3.1	Database Files .....	5
3.2	Memory Files .....	6
3.3	Command Files .....	6
3.4	Report Form Files .....	7
3.5	Text Output Files .....	7
3.6	Index Files .....	7
3.7	Format Files .....	7
4.0	Expressions .....	8
4.1	Functions .....	8
4.2	Operations .....	14
5.0	Macro Substitution .....	18
6.0	Interfacing with Non-dBASE Processor .....	19
7.0	Classes of Commands .....	20
8.0	Full Screen Operation .....	23
9.0	Command Rules .....	25
9.1	Symbol Definitions .....	25
9.2	Rules of Commands .....	27
10.0	Machine Language Commands .....	31
	Appendices	
	A) List of Commands .....	151
	B) Limitations and Constraints .....	154
	C) Error Messages .....	155
	D) Set Color To .....	159



**(This page is intentionally left blank)**





## 1.0 USING dBASE

To execute the dBASE program, place the dBASE distribution diskette (or preferably, a copy of that diskette) into any available disk drive. Set that drive to be the default drive (e.g. if the disk is placed into the "B" drive, type in "B:" followed by a carriage return) and then type in the following line:

```
dBASE
```

The program will then be loaded into memory, and will start execution with a date request:

```
ENTER DATE AS MM/DD/YY OR RETURN FOR NONE:
```

This date will be posted on any database that is altered during the following run and will also be printed in REPORT headings for any report generated in that run. The date is checked for calendar accuracy. **WARNING:** The calendar check is not valid for February 29 in the years 1900 and 2100. A slash or any special character (except a period) may be used to delimit the numbers.

### Examples of valid dates:

```
1.1.81
02 02 82
3/17/83
```

Then the sign-on message is displayed:

```
*** dBASE II   VER 02.XX.XX***
```

The period on the second line is the dBASE prompt, indicating that dBASE is ready to accept commands. Commands to dBASE are generally imperative sentences: a verb possibly followed by phrases that give further direction about the action to be taken. dBASE scans each line completely before executing any part of it. If dBASE detects an error in the command then the user is notified via error messages on the console. Generally, the user may correct the erroneous command and re-issue rather than re-enter the entire command. When dBASE detects an error that it can't describe explicitly, it assumes that the error is a syntax error and displays the erroneous line with a question mark at the beginning of the phrase that caused the confusion.

## **dBASE . . . 2**

### **Error recovery examples:**

```
. DISPRAY MEMORY
*** UNKNOWN COMMAND
DISPRAY MEMORY
CORRECT AND RETRY? Y
CHANGE FROM :PR
CHANGE TO :PL
DISPLAY MEMORY
MORE CORRECTIONS? (cr)
```

erroneous command echoed  
Yes, correct  
change the letters PR  
to PL  
after the change  
enter = no more changes

```
. STORE (2+2 TO X
*** SYNTAX ERROR ***
?
STORE (2+2 TO X
CORRECT AND RETRY? Y
CHANGE FROM :+2
CHANGE TO :+2)
STORE (2+2) TO X
MORE CORRECTIONS? N
4
```

the string (2 + 2 is indicated

N(o) more changes  
the result

```
. SUM TO X
NO EXPRESSION TO SUM
SUM TO X
CORRECT AND RETRY? N
```

explanation

no change, abort this command

The program can also be executed in the following manner:

```
DBASE <filename>
```

This will load dBASE into memory, access a command file <filename>, and begin immediate execution of the command file. This form is especially useful when using dBASE in a SUBMIT file or when using the chaining option of the dBASE QUIT command.

**CONTROL CHARACTERS**

<PRINT> — Toggles print switch (see also SET PRINT command)

ctl-U — Deletes current line

ctl-X — Deletes current line (except in full screen edit)

<BACKSPACE> — Deletes last character entered

<ESC> — Escapes from certain possibly long-running commands. I.e. DISPLAY, COUNT, DELETE, INPUT, LIST, LOCATE, RECALL, REPLACE, SKIP, AND SUM. Also ESC serves as an escape from ACCEPT, INPUT, REPORT (dialogue), and WAIT. In all cases, ESC returns control to the interactive monitor and displays a dot prompt.

When in a command file execution, dBASE checks for an ESC character before starting every command line.

**Note:** This escape capability can be disabled by the SET ESCAPE OFF command.

## REQUIREMENTS . . . 4

### 2.0 SYSTEM REQUIREMENTS

In order for dBASE to operate properly, a system with the following attributes should be available.

- a) Tandy Model 2000 with MS-DOS operating system;
- b) 256K bytes (or more) of memory including MS-DOS;
- c) Optional printer.

### 3.0 dBASE FILES

Basically, a file is a collection of information residing on a mass storage device that contains the user's data. The information can be stored to or retrieved from the file. Files can be grouped into six types, each one either concerned with a particular operation of or created by dBASE.

All dBASE files have a name field of eight characters and a file type of three characters. Listed below are the default file types used by dBASE. For each command that accesses a file, the type field may be left off and dBASE will assume the default type for that command. For instance, if a database file already has DBF as its type, then it need not be specified in any of the file manipulation commands.

DATABASE FILES	— .DBF
MEMORY FILES	— .MEM
COMMAND FILES	— .PRG
REPORT FORM FILES	— .FRM
TEXT OUTPUT FILES	— .TXT
INDEX FILES	— .NDX
FORMAT FILES	— .FMT

Any legitimate MS DOS filename may be used to refer to dBASE files. Remember, if during an access of any file, the type is not supplied by the user, dBASE will assume the above file types.

### 3.1 DATABASE FILES (.DBF)

Databases are what dBASE is all about. dBASE's database files consist of a structure record and zero to 65535 data records. The structure record is essentially a map of the data record format. The structure can contain up to thirty-two different entries. Each entry in the structure refers to a field of data in the data records. The structure holds the following data:

- \* The name of the data fields
- \* The type of data within data fields
- \* The size of the data fields
- \* The position of the data within records

**DATA FIELD NAME** — The name may be up to 10 characters long. In all operations during a dBASE run, the data fields will be referenced by this name. Field names are alphanumeric (plus colons) by nature. However, fields must begin with a letter and colons must be embedded in the name. Some examples follow.

## FILES . . . 6

### Examples of data field names:

A	
A123456789	
ABC:DEF	
A:B:C:D:E	
ABCD:	invalid, colon not embedded
ABC,DEF	invalid, comma is illegal

**DATA TYPE** — dBASE allows three types of data to be used to specify the contents of the data fields. They are: character strings ('ABCD'), numeric quantities (2 or 5\*18), and logical values (true/false).

**FIELD SIZE** — This is the number of character positions (width) needed to contain the data that will be placed into this field. Character string fields and numeric fields may be from 1 to 254 positions in length. The count for a numeric field should include the decimal point. Logical fields are always one position in length. Also, for numeric fields, the number of positions to the right of the decimal point may also be contained in the structure.

Once the structure has been defined, the user can enter data values into the fields for as many records as are desired. Usually, there is only one structured data file available to the user at any given time (this is referred to as the USE file or the file in USE). There is, however, a way to use two databases at one time. See the commands SELECT and JOIN.

### 3.2 MEMORY FILES (.MEM)

Memory files are static files of memory which are divided into variables similar to data fields. These variables are known as memory variables and are limited to 64 in number.

The values of memory variables are independent of the database in use. That is, the record position of the file in USE has no bearing on the variables in the memory file. Memory variables are used to contain constants, results of computations, and symbolic substitution strings (see Section 5), etc. The rules of naming, typing, and sizing of memory variables are identical to those of data fields described above.

The SAVE command will write all current memory variables to a memory file; and the RESTORE command will read a saved memory file back into the memory variables.

### 3.3 COMMAND FILES (.PRG)

A command file contains a sequence of dBASE command statements. This provides the user with a method of saving a set of frequently used command sequences which then allows one to more easily manipulate database files.

Command files may be created and modified by text editors and/or word processors, although dBASE now has the capability to create/edit command files itself with the MODIFY COMMAND. Command files are started by the DO command. Command files may contain any dBASE commands, however, one should be careful since some of the commands (CREATE, INSERT, APPEND) require user inputs beyond the command file contents.

Command files may be nested, i.e. command files may contain DO commands which are then executed. Again, care should be exercised in that dBASE allows, at most, 16 files to be open at any given time. Therefore, if there is a file in USE, only 15 command files may be nested. Certain commands also use work files (e.g. SORT uses 2 additional files; REPORT, INSERT, COPY, SAVE, RESTORE, and PACK each use one additional file). For instance, if a SORT command is issued from the lowest command file in a nest, then only 13 levels of command file could be used (i.e. the USE file, 2 SORT work files and 13 command files = 16). Whenever a command file issues the RETURN command or whenever the end-of-file is encountered on a command file, the command file is closed and its resources are available for other commands.

### 3.4 REPORT FORM FILES (.FRM)

The REPORT command either generates a new form file or uses an existing form file. The form file contains instructions to the report generator on titles, headings, totaling, and column contents. Form files are constructed by dBASE as part of the REPORT dialog. They can be modified by text editors or word processors; however, it is fairly easy to make mistakes and this practice is discouraged except for advanced dBASE users.

### 3.5 TEXT OUTPUT FILE (.TXT)

The text output files are created when the "SET ALTERNATE TO <filename>" and "SET ALTERNATE ON" commands have been specified. See SET command for more details. Also, the COPY and APPEND commands assume a text (.TXT) file whenever the SDF (System Data Format) or DELIMITED options are used.

### 3.6 INDEX FILES (.NDX)

Index files are generated by the INDEX command of dBASE. They contain keys and pointers to records of a database file. Indexing is a dBASE technique that gives rapid location of data in a large database. See the INDEX command for more information.

### 3.7 FORMAT FILES (.FMT)

A format file contains only "@" statements and "\*" comments. It is identified by the "SET FORMAT TO <filename>" command and is activated by subsequent READ commands. Like command files (which format files resemble), format files are created and modified by any good text processor or the MODIFY COMMAND capability. Format files are not, however, necessary. "@"'s and "\*"s statements are usually built into the command file that needs them.

#### 4.0 EXPRESSIONS

An expression in dBASE is a group of simple items and operators that can be evaluated to form a new simple value. For example "2 + 2" is an expression that can be evaluated to the value "4". Expressions are not necessarily always numeric in nature. The expression 'abc'+ 'def' can be evaluated to the value 'abcdef' (character string concatenation), or the expression 1 > 2 can be evaluated to the logical (Boolean) value of ".F." (false).

Expressions in dBASE are formed from the following components:

- \* Database field variables
- \* Memory variables
- \* Constants within the commands (literals)
- \* Functions
- \* Operations

**VARIABLES** — A variable in dBASE is any data field whose value may change. There are two types of variables: data field names, (the information contained in the field of a dBASE database record is subject to change any time the database is repositioned or edited); and memory variable names (which contain information which is subject to change whenever a STORE, RESTORE, COUNT, SUM, WAIT, ACCEPT, INPUT, etc. command is issued).

There are three types of variables:

- \* Character strings
- \* Numeric quantities
- \* Logical values

**CONSTANTS** — A constant (or literal) is a data item which has an invariant, self-defined value. For instance, 1, 'abc', and .T. are constants which have a constant value regardless of the position of the database or any memory variable commands. They are literals since they ARE the value they represent (as opposed to variables which are names representing a value). The values they represent are, respectively: a numeric one, a character string (containing the letters "a", "b", and "c"), and a logical (Boolean) value of TRUE (".T.").

Character string constants must be enclosed in single quotes ('), double quotes ("), or in square brackets ([,]). If a character string contains one of these "delimiters", then it should be enclosed in a pair of one of the other ones. For example, the strings 'abc[def]ghi' and labc'def'ghi) are valid character strings while 'abc'def'ghi' is not.

Logical constants (true/false) are represented by "T", "t", "Y", or "y" for true values (denoting true or yes) and "F", "f", "N", or "n" for false values (denoting false or no).



## 4.1 FUNCTIONS

Functions are special purpose operations that may be used in expressions to perform things that are difficult or impossible using regular expressions. In dBASE, there are three basic types of functions: numeric, character, and logical. The function type is based on the type of value that functions generate.

### Integer Function

INT(< numeric expression > )

This function evaluates a numeric expression and discards the fractional part (if any) to yield an integer value. The value of the INT function is the truncated value of the numeric expression within.

#### Examples:

```
. ? INT(123.456)
123
. STORE 123.456 TO X
123.456
. ? INT(X)
123
```

### Record Number Function:

#

The value of the record number function is the integer corresponding to the current record number.

#### Examples:

```
. ? #
4
. SKIP
. ? #
5
```

(assuming that a database is in USE and is positioned at record number 4)

## EXPRESSIONS ... 10

### String Function:

STR(< numeric expression> , < length> , [< decimals> ])

This function evaluates a numeric expression and yields a character string. The value of the STR function is a character string of length < length> . If < decimals> are specified, it is the number of digits to the right of the decimal point. All specifiers may be constants, variables, or expressions.

**Caution:** When this function is used to generate a key for indexing, the specifiers **MUST** be literals.

### Example:

```
. ? STR(123.456,9,3)
123.456
```

### Substring Function

\$(< char expression> , < start > , < length> )

This function forms a character string from the specified part of another string. The value of the substring function is a character string of length < length> filled with characters from the character expression starting with character position < start> for < length> characters. < start> may be constants, variables or expression. However, when used with the INDEX or SAY commands, < length> must be a literal.

If < length> is longer than the < char expressions> , or if between the < length> and < start> the < char expression> "runs out" of characters, then the result will be only those characters that are there. See the following examples.

**Caution:** When the function is used to generate a key for indexing, the specifiers **MUST** be literals.

### Examples:

```
. ? $('abcdefghi',3,3)
cde
. store 3 TO m
3
. store 3 TO n
3
. ? $('abcdefghi',m,n)
cde
. ? $('abcdefghi',6,7)
fghi
```

**String to Numeric Function**

VAL(&lt;char string&gt;)

This function forms an integer from a character string made of digits, signs, and up to one decimal point. The length of the integer is equal to the number of characters in the string. If the character string begins with numeric characters but also contains non-numeric characters, then the value generated by the VAL function equals the leading numeric characters.

Another way to convert character numbers into numerics is the use of "&" (see 5.0 Macros). The "&" will convert the string into a numeric (including the decimal) when the substitution is encountered.

**Examples:**

```
. ? VAL('123')
123
. ? VAL('123xxx')
123
. ? VAL('123.456')
123
. STORE '123.456' TO NUM
123.456
. ? 14 + &NUM
137.456
```

**Rank Function**

RANK (&lt;string&gt;)

This function returns the ASCII value of the first character of the <string>. This function corresponds to the ASC function in many versions of the BASIC language.

**Example:**

```
. ? RANK('A')
65
```

**Length Function**

LEN(&lt;char string&gt;)

This function yields an integer whose value is the number of characters in the named string.

**Example:**

```
. STORE 'abc' TO STRING
. ? LEN(STRING)
3
```

## EXPRESSIONS . . . 12

### Deleted Record Function

\*

This is a logical function which is .TRUE. if the current record has been marked for deletion, and .FALSE. otherwise.

#### Example:

```
. ? *  
.T.
```

(assuming that a database is in USE and that its current record has been deleted using the DELETE command)

### End-of-File Function

EOF

This is a logical function which is .TRUE. if the end of file has been reached for the file in USE (the current record will be the last record in the database).

#### Examples:

```
. ? EOF  
.F.  
. GOTO BOTTOM  
. ? EOF  
.F.  
. SKIP  
. ? EOF  
.T.
```

(assuming that a database is in USE and is not positioned at the last record).

### Substring Search Function

@ (<char string 1>,<char string 2>)

This function yields an integer whose value is the character position in <char string 2> which begins a substring identical to <char string 1>. If string 1 does not occur in string 2 then the @ function will be of value zero. Note: the @ function is similar to the substring operator "\$" except that it tells where the first string is found in the second string, and can well be pronounced "where is string 1 AT in string 2".

#### Example:

```
. ? @ ('def','abcdefghi')  
4
```

**Upper Case Function**

!( < char string expression > )

This function yields the same string as the character string expression except that all lower case characters are converted to upper case.

**Example:**

```
. ? !('abc')
ABC
```

**Number to Character Function**

CHR( < numeric expression > )

This function yields the ASCII character equivalent of the numeric expression. That is, if the expression were the number 13, then CHR(13) generates a carriage return ASCII character. This function is useful when the user needs to send direct controls to hardware devices, most often printers.

**Example:**

```
. ? 'abcd' + CHR(13) + '_____'
abcd
```

**Date Function**

DATE( )

This function will generate a character string that contains the system date in the format XX/XX/XX. The character string always has a length of 8. Nothing goes between the parentheses, they only indicate a function (to avoid problems with variables named "DATE").

The dBASE system date can be entered at dBASE start-up time or at anytime using the SET DATE TO command. With the SET DATE TO <string> (no quotation marks), no checking is done, so the user can enter a date that is in MM/DD/YY, DD/MM/YY, or YY/MM/DD format.

**Examples:**

```
. ? DATE( )
06/15/81
. STORE DATE( ) TO MEMVAR
06/15/81
. SET DATE TO 4 1 82
. ? DATE( )
04/01/82
```

## EXPRESSIONS ... 14

### File Function

FILE(<string exp>)

This is a logical function which is **.TRUE.** if the <string exp> exists and is **.FALSE.** if it does not.

#### Example:

```
. USE TRACE
. ? FILE('TRACE')
.T.
```

### Type Function

TYPE(<exp>)

This function yields a one-character string that contains a 'C', 'N', 'L', or 'U' if the <exp> is of type Character, Numeric, Logical, or Undefined respectively. (TYPE returns a "U" if the expression is not parsable, even if a variable's parenthetic comments are syntactically correct.)

#### Example:

```
. STORE 1 TO X
. ? TYPE(X)
N
```

### Trim Function

TRIM(<cstring>)

The TRIM function removes trailing blanks from a field. Usually dBASE carries trailing blanks on all variables to avoid column alignment problems on displays.

**Note:** This function must NOT be used in the INDEX command as the key length must be computable for internal dBASE usage.

#### Examples:

```
. STORE 'ABC' TO S
. ? LEN(S)
6
. STORE TRIM(S) TO S
. ? LEN(S)
3
```

## 4.2 OPERATIONS

There are four basic types of operations, arithmetic, comparison, logical and string. The specific operators in each class are listed below, and examples follow for the less familiar ones.

It is important to know that both "sides" of the operators must be the same type. That is, one may only add integers to integers or concatenate characters with characters, adding an integer to a character results in dBASE seeing a syntax error.

```
. STORE 3 TO A
3
. STORE '3' TO B
3
. ? A+B
*** SYNTAX ERROR ***
?
? A+B
CORRECT AND RETRY(Y/N)?
```

This error occurs because numerics and characters are seen differently at the machine level; a numeric 3 is just that—3 hex, while a character 3 has the ASCII value of 33 hex. The program becomes confused, it does not know whether or not an addition is taking place or a concatenation. Using the same variables as in the previous example:

```
. ? A+VAL(B)
6
```

The string '3' has been converted to an integer and the addition performed.

ARITHMETIC OPERATORS (generate arithmetic results)

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

**Examples:**

```
. ? (4+2)*3
18
. ? 4+(2*3)
10
```

An example of use of arithmetic parentheses used for grouping in calculations

COMPARISON OPERATORS (generate logical results)

```
< = less than
> = greater than
= = equal
< > or # = not equal
< = = less than or equal
> = = greater than or equal
$ = substring operator (e.g., if A and B are character strings, A$B will be TRUE if and only if string A is equal to B, or is contained in B)
```

## EXPRESSIONS . . . 16

### Examples:

. ? 'abc'\$'abcdefghi'

.T.

. ? 'abcd'\$'ghijkl'

.F.

. DISPLAY FOR '8080'\$TITLE

An example of the \$ substring operator

Results in all records with '8080' somewhere in the field TITLE being displayed on the screen

### LOGICAL OPERATORS (generate logical results)

.OR. = Boolean or

.AND. = Boolean and

.NOT. = Boolean not (unary operator)

### Examples:

. STORE T TO A

.T.

. STORE F TO B

.F.

. ? A.OR. B

.T.

. STORE .NOT. B TO C

.T.

. ? A.AND. C

.T.

### STRING OPERATORS (generate string result)

+ = string concatenation

- = string concatenation with blanks moved to far right

### Examples:

. STORE 'ABCD ' TO A

ABCD

. STORE 'EFGH' TO B

EFGH

. ? A+B

ABCD EFGH

. STORE 'ABCDE ' TO A

ABCDE

. STORE '1234 67' TO B

1234 67

. ? A-B

ABCDE1234 67

In a string concatenation the two strings are just appended to each other.

In a string concatenation with blank move, the trailing blanks are moved to the end of the string. Leading and embedded blanks are not altered.



**Note:** The lengths of  $A - B$  and  $A + B$  are equal.

**ORDER OF EXECUTION**

Arithmetic, string and logical operators have an order in which they are satisfied. That is, what operation is done before what other operations. The following table indicates the order of precedence for each of the three major operator classes. In each of the "levels" (1, 2, etc.) the order of execution is left-to-right.

Arithmetic operator precedence	String operator precedence	Logical
1) parenthesis, functions	parenthesis, functions	.NOT.
2) unary +, -	relations, \$(substring op)	.AND.
3) *, /	+, - (concatenation)	.OR.
4) +, -		
5) relations		

**Example:**

. ??4 + 2\*3

10

### 5.0 MACRO SUBSTITUTION

Whenever an ampersand (&) followed by the name of a character string memory variable is encountered in a command, dBASE replaces the & and memory variable name with the memory variable's character string. This allows the user to define some parts of a command once and call it out any number of times in various commands.

Macros are useful when complex expressions must be frequently used. They also allow parameter passing within command file nests. All characters between the ampersand and the next special character (including space) are taken as the memory variable name.

If the user desires to append characters to the symbolic substitution, then the memory variable name should be terminated with a period. The period will be removed like the ampersand at substitution time.

If an ampersand is not followed by a valid memory variable name then no expansion is attempted and the ampersand remains in the command line.

#### Examples:

```
. ACCEPT ""Enter data disk drive letter""to DR  
. STORE DR + ':DATAFILE' TO DR  
USE &DR
```

(at execution time this becomes  
USE B:DATAFILE if "B" was entered in  
response to the ACCEPT)

```
. STORE 'DELETE RECORD' TO T  
&T 5
```

(at execution time will be DELETE  
RECORD 5)

## 6.0 INTERFACING WITH NON-dBASE PROCESSORS

dBASE can read data from files which were created by processors other than dBASE (e.g. BASIC, FORTRAN, PASCAL) and can generate files which can be accepted by other processors.

The APPEND command has the ability to read standard ASCII text files (using the MS DOS convention of a line of text followed by a carriage return and line feed) by specifying the SDF (System Data Format) option. Similarly, the COPY command generates standard ASCII format files when the SDF option is used. Unless explicitly overridden, the file types of files created with the SDF and DELIMITED options will be .TXT.

Some processors and languages read and write files in a delimited format. In this form all fields are separated by commas and character strings are enclosed in quotes. dBASE can APPEND and COPY these files when the DELIMITED keyword is included in the command. If the DELIMITED feature is used, SDF is assumed.

Since some processors use single quotes and some use double quotes to delimit character strings, APPEND will accept either. The COPY command normally generates single quotes but will output any character as defined by the WITH phrase of the DELIMITED clause. It is strongly recommended that only single and double quotes be used.

A special case occurs when a "," is used in the WITH phrase for a COPY. All trailing blanks in character strings and leading blanks in numerics are trimmed. Also, character strings will not be enclosed with quotes or any other character.

### Examples:

- . USE <filename>.DBF
- . COPY TO <filename>.TXT DELIMITED WITH "
  
- . USE <filename>.DBF
- . APPEND FROM <filename>.TXT SDF

### 7.0 CLASSES OF COMMANDS

During the normal use of dBASE, various commands are used in combination to accomplish a particular task. Such groups are shown below. Some dBASE commands are patterned after the structured constructs that most "modern" computer languages use. These commands are in the COMMAND FILE class of commands. There are some special rules that control the use of these commands, which are expounded upon in section 9.0.

**CREATION OF FILES** — the following commands create database files and associated files:

- \* CREATE — create new structured database files
- \* COPY — copy existing databases to create copies
- \* MODIFY — alters database structures
- \* REPORT — create a report form file
- \* SAVE — copy the memory variables to mass storage
- \* INDEX — creates an index file
- \* REINDEX — realigns an old index file
- \* JOIN — outputs the JOIN of two databases
- \* TOTAL — outputs a database of totaled records

**ADDITION OF DATA** — the following commands add new data records to databases:

- \* APPEND — add data at end of a file
- \* CREATE — allows addition of data at creation
- \* INSERT — insert data into a file

**EDITING OF DATA** — the following commands edit the data within a database:

- \* CHANGE — edit columns of fields
- \* BROWSE — full screen window viewing and editing
- \* DELETE — marks records for deletion
- \* EDIT — alter specific data fields in a database
- \* PACK — removes records marked for deletion
- \* RECALL — erases mark for deletion
- \* REPLACE — replaces data fields with values
- \* READ — replaces data from user defined full-screen
- \* UPDATE — allows batch updates of a database

**USER ASSISTANCE COMMANDS** — the following commands give user on-line information:

- \* DISPLAY — current information about datafile and system parameters  
STATUS
- \* DISPLAY — show files on currently logged disk drive  
FILES
- \* HELP — explanation of dBASE commands and other information

**DATA DISPLAYING COMMANDS** — the following commands display selected data from a database:

- \* @ — displays user formatted data on the terminal or printer
- \* BROWSE — displays up to 19 records with as many fields as will fit on the screen
- \* COUNT — count the number of records that meet some conditional expression
- \* DISPLAY — displays records, fields, and expressions
- \* READ — displays data and prompting information in full-screen mode
- \* REPORT — format and display a report of data
- \* SUM — compute and display the sum of an expression over a group of database records
- \* ? — displays an expression list
- \* TEXT — displays a block of text data

**POSITIONING COMMANDS** — the following commands position the current record pointer to records as directed:

- \* CONTINUE — positions to next record with conditions specified in the LOCATE command
- \* FIND — positions to record corresponding to a key on indexed files
- \* GOTO — position to a specific record
- \* LOCATE — find a record that fits a condition
- \* SKIP — position forwards or backwards

**FILE MANIPULATING COMMANDS** — the following commands affect entire database files:

- \* APPEND — append dBASE files or files in System Data Format (SDF)
- \* COPY — copy databases to other databases or SDF files
- \* DELETE — delete files
- \* DO — specifies a command file from which subsequent commands are to be taken
- \* RENAME — rename a file
- \* SELECT — switches between USE file
- \* SORT — create a copy of a database which is sorted on one of the data fields
- \* USE — specifies the database file to be used for all operations until another USE is issued

**MEMORY VARIABLE COMMANDS** — the following commands manipulate the memory variables:

- \* ACCEPT — stores a char string into memory variables
- \* COUNT — stores counts into memory variables
- \* DISPLAY — can display memory variables
- \* INPUT — stores expressions into memory variables
- \* RESTORE — retrieves sets of stored memory variables
- \* SAVE — save the memory variables to a file
- \* STORE — stores expressions into memory variables
- \* SUM — stores sums into memory variables
- \* WAIT — accepts a single keystroke into a memory variable

## COMMANDS ... 22

COMMAND FILE COMMANDS — the following commands assist in the control and usage of command files:

- \* ACCEPT — allows input of character strings into memory variables
- \* CANCEL — cancels command file execution
- \* DO — causes command files to be executed and allows structured loops in command files
- \* IF — allows conditional execution of commands
- \* ELSE — alternate path of command execution within IF
- \* ENDDO — terminator for DO WHILE command
- \* ENDIF — terminator for IF command
- \* INPUT — allows input of expressions into memory variables
- \* LOOP — skips to beginning of DO WHILE
- \* MODIFY — allows editing of command files
- \* RETURN — ends a command file
- \* SET — sets dBASE control parameters
- \* WAIT — suspends command file processing

DEVICE CONTROLLING COMMANDS — the following commands control peripheral devices like printers and terminals:

- \* EJECT — ejects a page on the list device
- \* ERASE — clears the terminal

**8.0 FULL SCREEN OPERATION**

The following are cursor control keys for full screen operation:

- left arrow.  
or ctl-E      — Backs up to previous data field.
- right arrow  
or ctl-X      — Advances to next data field.
- ctl-S          — Backs up one character in data field.
- ctl-D          — Advances one character in data field.
- ctl-Y          — Clears out rest of current field to blanks.
- <INSERT>  
ctl-V          — Switches (toggles) between overwrite and insert modes.
- <DELETE>  
or ctl-G       — Deletes character under cursor.
- <BACKSPACE>— Deletes character to left of cursor.
- ctl-Q          — Aborts full screen and returns to normal dBASE control. Changes to database variables are abandoned.

When in EDIT:

- ctl-U          — Switches (toggles) the current record between being marked for deletion and unmarked.
- ctl-R          — Writes current record back to disk and displays previous record i.e. backs up a record.
- ctl-C          — Writes current record back to disk and displays next record i.e. advances to next record.
- ctl-W          — Writes current record to disk and exits screen edit mode.

When in MODIFY

- ctl-N          — Moves all lines down one to make room for an insertion of a new line.
- ctl-T          — Deletes the line where the cursor is and moves all lower lines up.
- ctl-C          — Scrolls page down.
- ctl-R          — Scrolls line up; scrolls page up if at top of screen.
- ctl-W          — Writes data to the disk and resumes normal operations.
- ctl-Q          — Exits without saving changes.

## FULL SCREEN . . . 24

When in APPEND, CREATE, or INSERT:

ctl-C or ctl-R — Write current record to disk and proceed to next record.

Carriage return when no changes have been made and cursor is in initial position — terminate operation and resume normal dBASE operations.

When in BROWSE:

ctl-U — Switches (toggles) the current record between being marked for deletion and unmarked.

ctl-R — Writes current record back to disk and displays previous record i.e. backs up a record.

ctl-C — Writes current record back to disk and displays next record i.e. advances to next record.

ctl-W — Writes current record to disk and exits screen edit mode.

ctl-Z — Pans the window left one field.

ctl-B — Pans the window right one field.



## 9.0 COMMANDS

The explicit definitions of the dBASE commands are in this section. The user should familiarize him/herself with the information in this front material before reading the rest of the command information.

### 9.1 SYMBOL DEFINITIONS

Understanding what the special symbols in the general formats of the dBASE commands really mean is vitally important. Not only does it help in understanding just what the form of the command really is, it helps to show the potential of each command. Please read the following table thoroughly.

< commands > or < statements >

Means any valid dBASE statements; it also means *whole* statements. An IF without an ENDIF, (or a DO WHILE without an ENDDO), is only half of a statement, while a REPORT is a whole statement in itself.

< char string > or < cstring >

Means any character string; character strings are those characters that are enclosed in single quotes ('), doublequotes ("), or square brackets ([ ]).

< delimiter >

Means any special character; special characters are those characters from the keyboard that are punctuation marks like any of the following: "()\*=,@'.

< exp >

Means an expression; an expression can be created by tacking together numbers, functions, field names or character strings in any meaningful manner. "4 + 8," and "doc = '3' .or. doc = '4'," are both expressions as well as "\$('abc'+&somestr,n,3) = 'abcdefg'."

< exp list >

Means a list of expressions separated by commas; usually simple expressions are used. Two of the examples in the previous paragraph are rather complicated, the first one could be considered as simple.

< field >

Means any record field name; in one of the examples that are in the following commands, one of the databases has field names like ITEM, COST, DATE, etc.

< field list > or < list >

Means a list of record field names separated by commas.

< file > or < file name >

Means any filename; these are file names that must obey the rules for file names that were stated in section 3.0

## RULES ... 26

### < form file >

Means the name of a report form file; see section 3.4 and the REPORT command for the how and why of this type of file.

### < index file >

Means the name of the file where indexing information is placed; see section 3.6 and the INDEX command for the how and why of this type of file.

### < key >

Means a field name on which the database will be indexed. There may be several indexes for any given database, each on different (or on a combination of) keys. Keys may be <expressions> or field names. See the INDEX command for more information.

### < memvar >

Means any memory variable; memory variables are those variables that are created by STOREs or by use of a command that saves some value for later use (ACCEPT, INPUT, etc.) There is a maximum of 64 memory variables allowed in dBASE.

### < memvar list >

Means a list of memory variables separated by commas.

### < n >

Means a literal; literals are numbers which are not gotten from memory variables or calculations. "4 + 8" is not a literal, while "4" and "9876" are literals.

### < scope >

Means a specification of the scope of the command; scope means how much does the command cover. There are three values that < scope > may take on.

### ALL

Means all the records in the file. ALL means that the file is rewound and whatever the command ALL, the records in the file are searched for compliance. ALL is the default for some of the commands. For other commands, the default will be the current record (especially for the more potentially destructive commands like DELETE). Each command description tells what is the default scope. In the case of using a FOR phrase in any of the commands, ALL will be the default.

### NEXT n

Means the next n records, including the current record; NEXT also begins with the record currently being pointed at. An n must have a literal value, that is, it must not be a memory variable or an expression.

### RECORD n

Means only record n; again, n must not be a memory variable or an expression—it must be a literal before it will work.

**FOR <exp>**

Any record so long as some logical expression has a true value. Unless otherwise specified, the presence of a FOR clause causes ALL records to be scanned (with a rewind of the database). It is exclusive of the WHILE clause.

**WHILE <exp>**

Operate on the next block of records, as long as some logical expression (<exp>) has a true value. For the command to execute at all, the expression must be true for the current record. The command stops when the first false expression is encountered. The presence of a WHILE clause implies NEXT 65534 unless otherwise specified and does not rewind the database. It is exclusive of the FOR clause.

There are other special symbols used in the command formats. These are special to the command and will be explained in the body of the command.

**9.2 RULES TO OPERATE BY**

As with all command "languages," there are a set of rules which must be followed to successfully operate the program. The following rules are to be used in translating the general format of the commands into the more useful specific forms.

1. The verb of any command must be the first non-blank character of the command line; the phrases may follow in any order. A verb is an action word; CREATE, APPEND, REPORT, SET, DISPLAY and ERASE are all examples of verbs—they cause a specific action. Phrases are equivalent to adverbs; they more fully describe the action. FOR, NEXT, and WITH are examples of words that begin phrases. All of these example words are referred to as "keywords."
2. Any number of blanks may be used to separate words and phrases. Remember though, blanks are counted in the 254 limit described in Rule #3.
3. All commands must be less than 254 characters in length (even after a macro expansion).
4. Commands and keywords can be abbreviated to the first four (or more) characters. E.g. DISPLAY STRUCTURE could be input as DISP STRU or DISPL STRUCT or etc. Just remember that the abbreviation must also be spelled correctly up to the point where it ends.
5. Either upper or lower case letters may be used to enter commands, keywords, field names, memory variable names, or file names.
6. Parts of the commands are optional, that is, some parts of the commands may be left off when the command is used. Square brackets ( | ) are used in the command formats to show which phrases are the optional constructs that may be left off. These are the phrases which are used to modify the action of commands. The upper case words are the keywords and they must be entered whenever the phrase that contains them is used.

## RULES... 28

7. A reserved word is a keyword that will generate an error if it is used for something other than what it is supposed to be. There are no reserved words in dBASE. However, certain field names and file names can cause difficulty, e.g., a command file named WHILE will be incorrectly interpreted as a DO WHILE statement by the DO command processor; ALL as a field name cannot be used in a number of commands. (Also STATUS, STRUCTURE, FILE, RECORD). In general, it is a good practice to avoid the use of dBASE keywords as field names or file names.
8. dBASE statements in a command file must nest correctly. To nest something means that one statement must fit inside another statement. This is especially important to proper execution of the IF-ELSE-ENDIF and the DO WHILE-ENDDO groups. Indenting a command file will show if the statements are correctly nested. dBASE does not catch nesting errors; it will, however, execute the command file in an unknown manner. On the following page are examples of how to correctly nest these two statements.

```
DO WHILE .NOT. EOF
    statements
IF A .AND. B
    more statements
ELSE
    DO WHILE A <= 57
        some more statements
    ENDDO
    even more statements
ENDIF
    infinitely more statements
ENDDO
```

This is the correct way to nest. The IF-ELSE-ENDIF statement is totally within the DO WHILE-ENDDO statement, just as the second DO WHILE-ENDDO statement is totally within the ELSE part of the IF-ELSE-ENDIF. It would be just as easy to show more levels of nesting, since dBASE allows many more levels to exist.

```

DO WHILE .NOT. EOF
.
. statements
.
IF something changes values
.
ENDDO
.
. more statements
.
ENDIF
    
```

This is an example of a NO NO. The ENDDO crossed over the boundary of the IF-ENDIF group; that is, the two statements do not nest properly. The command file that holds these statements will not work as expected and dBASE will not explain why.





**(This page is intentionally left blank)**



## 10.0 MACHINE LANGUAGE COMMANDS

SET CALL TO <address>

Sets the decimal address that will be called by a dBASE CALL command.

CALL [<memvar>]

Performs a machine language call to the address set by a SET CALL TO or the default address if no SET CALL has been done. There are about 254 bytes of stack available, the BX register pair points to the first byte of the <memvar>. It is most important that no attempt be made to lengthen or shorten a character string. Control can be passed back to dBASE with a RET instruction.

LOAD [<file>]

This command loads a file that is assumed to be a .HEX file in INTEL HEX format into memory.

POKE <address>.<data byte>[,<data byte> ...]

Places data directly into memory

PEEK (<address>)

A function that returns a number corresponding to the unsigned binary value of the byte at <address>.



**(This page is intentionally left blank)**





?

```
? [<exp list> |
?? [<exp list> ]
```

This command is a specialized form of the DISPLAY command; it is equivalent to DISPLAY OFF <exp>. It can be used to show the value of an expression or list of expressions. The question mark command (possibly pronounced "what is") can use memory variables, database fields, constants, or functions. A "?" with no expression spaces down a line on the output. This feature is particularly useful in command files to "open up" the displays.

The second form of this command "??" behaves like a single "?" except that no line feed or carriage return is done before the expression is printed. This can be used in command files to output more than one expression to the same output line.

If a SET RAW ON command is in effect, then no spaces will be placed between items in a list, otherwise one space separates items.

#### Examples:

```
. USE EXAMPLE
. 4
. ? #
4
. ? NAME
CHANG. LEE
. ? 5+9
14
```

Following is a sample command file that uses the ? to space out the display. The command file is set up to be executed with the command: "DBASE D:FILE." The dBASE response to the command file follows the command file.

```
SET DEFAULT TO D
USE trace INDEX trace
DISP STRU
?
ACCEPT "Enter today's date." TO dte
SET DATE TO &dte
RELEASE dte
RETURN
```

?...34

STRUCTURE FOR FILE: TRACE.DBF

NUMBER OF RECORDS: 02359

DATE OF LAST UPDATE: 10/08/82

PRIMARY USE DATABASE

FLD	NAME	TYPE	WIDTH	DEC
001	UP	C	024	
002	TRFLD	C	005	
003	DOC	C	024	
004	DESCR	C	080	
005	NATURE	C	010	
006	STATUS	C	006	
007	TESTED	C	004	
** TOTAL **			00154	

Enter today's date:10 14 81

•

@

```
@ <coordinates> [SAY <exp> [USING <format>]]
      [GET <variable> [PICTURE <format>]]
```

This command works with the SET FORMAT TO, ERASE, EJECT, CLEAR, GETS and READ commands and is a most powerful way to display specific, formatted information on the screen or the printer. The way an "@" is interpreted changes according to how the SET FORMAT TO command is used. Also whether or not one of the other commands has an effect also depends on the SET command. All combinations are discussed below.

The <coordinates> are an "x,y" pair and may take on one of two meanings, either they are screen coordinates or they are printer coordinates. The "x,y" denotes line (x) and column (y). On the Model 2000, the screen oriented coordinates have an "x" range of 0-23, and a "y" range of 0-79, that is 24 lines by 80 columns. dBASE uses the 0th line for messages to the user and the user should avoid using it. The printer oriented coordinates have both an "x" and a "y" range of 0-254. For either of these two meanings, the coordinates can be any literal, numeric memory variable, or numeric expression. The SET FORMAT command is used to choose between either of these two meanings.

The <coordinates> can also be relative addresses on the screen. That is, the new <coordinates> can reference the ending location on the screen or printer. A "\$ + <expression>" construct is used to signal relative addressing and means the <coordinate> where the last @-command left off plus the value of the expression. If an @ 10,10 say 'HI' had been issued, then @ \$,\$+2 say 'THERE' will generate "HI THERE" on line 10 starting in column 10. Either the row or column can be relatively addressed. Only positive offsets are allowed (e.g. \$ - <expression> is NOT permitted).

When a SET FORMAT TO SCREEN command has been issued (which is the default), the "@" command causes data to be displayed on the screen. A coordinate pair of 0,0 means the first character location on the upper left corner of the display. (This is frequently referred to as the home position.) The pair 10,15 means the 11th line and the 16th column of the display. Again the 0th line on the screen should not be used. "@" commands may be issued in any order to the screen. That is, one may SAY something to line 15 before one SAYS something to line 10. Likewise columns may be filled in any order.

When a SET FORMAT TO PRINT command has been issued, the "@" command will cause data to be printed on the printer. The coordinate pair 0,0 refers to the upper left hand corner of the paper. "@" commands to the printer must be output in order. Much paper will be wasted if this is not done. The user may like to pretend that a typewriter is being used (indeed, it is). All commands to line 5 must precede commands to line 6; also, all commands to column 10 must precede commands to column 20, etc. If this is not done, a page eject will occur before the new line is printed.

When the SET FORMAT TO SCREEN has been issued, an ERASE will clear the screen of all information that was previously on it, will release all the GETs (see below), and will reset the coordinates to 0,0. When the SET FORMAT TO PRINT has been issued, an EJECT will do a page feed and reset the coordinates to 0,0.

The SAY phrase is used to display an expression that will not be altered by subsequent editing via the READ command. The USING subphrase is used to format the expression emitted by the SAY phrase. Formatting directives are explained below. It is a good thing to always use the USING subphrase. dBASE will take liberties with the expression if there is no USING.

SAY phrases may be used on either the screen or the printer. GETs however, will only be recognized when the SET FORMAT TO SCREEN command has been issued.

The GET phrase displays the current value of a field variable or memory variable. The variable must exist prior to issuing of the GET and is subject to later editing by the READ command. The PICTURE phrase may be used with a GET phrase to allow special formatting and validation of the data as it is entered (see the READ command for further information). If no PICTURE clause is given, then the data type (character, numeric or logical) forms an implicit PICTURE.

If the data type of the field variable or memory variable in the GET is logical, then the data validation allows only the characters 'T', 'F', 'Y', 'N', and their lower case equivalents to be entered.

A maximum of 64 GETs can be active at any given time. Either the ERASE command or the CLEAR GETS command may be used to release the existing GETs.

When SET FORMAT TO SCREEN is in effect and if neither a SAY or a GET phrase is given, then the remainder of the line indicated by the coordinates is cleared to spaces. Thus @ 10,0 will clear the entire 11th line.

When the SET FORMAT TO SCREEN is in effect, a READ must be issued in order to "fill" the GETs. (See the READ command). However when SET FORMAT TO PRINT is in effect, "@" commands require no subsequent READ commands to complete their action.

Not needing a READ to print allows the user to directly format the output for any pre-printed material (such as checks, purchase orders, etc.) in a most convenient manner. The user need only to remember that "@" commands must be issued as if one were typing on a typewriter.

In using the SET FORMAT TO PRINT capability, it is often necessary to print out more than one item. The ability to substitute memory variables for the coordinate values is important. The following example is from a command file that generates a special report form for a special task.

```

SET FORMAT TO PRINT
GOTO TOP
STORE 7 TO CNTR
DO WHILE .NOT. EOF
  IF CNTR >= 50
    EJECT
    STORE 7 TO CNTR
  ENDIF
  @ CNTR,12 SAY P USING 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  @ CNTR,48 SAY D USING 'XXXXXXXXXX'
  @ CNTR,64 SAY P1 USING 'XXXXXXXXXXXXXXXXXXXX'
  @ CNTR,88 SAY U USING 'XXXXXXXXXX'
  @ CNTR,104 SAY P2 USING 'XXXXXXXXXXXXXXXXXXXX'
  IF RCD < > 0
    @ CNTR,130 SAY RCD USING '9999'
  ENDIF
  STORE CNTR + 1 TO CNTR
  SKIP
ENDDO
RETURN

```

In this command file, a maximum of 57 lines will be printed on the printer before a page eject is done. The purpose here was to print out most of the fields of a database (and selectively print out one of the fields). Care must be taken to make sure enough room is given to the SAY phrase to emit the variable. If the USING is shorter than the variable or the field, the variable or field is truncated. The <format> for the USING (the 'XXX...X') strings are explained in the table below.

Also, in the SET FORMAT TO PRINT mode, if the coordinates of the next "@" allow information to be printed on the same line but start it in a column that has already been printed, the printer may not output the proper information. In fact, the printer may go to the extreme right and print (in one square) all the information in the rest of the line. In the SET FORMAT TO SCREEN mode, the old information will be written over by the new information.

The last form of the SET FORMAT command is: SET FORMAT TO <format file>. When this command is in effect and when a READ command has been issued, the "@" commands are READ from the predesigned <format file>. In this manner, the user may design the screen into a format for more specialized purposes. It is important to note here that the use of format files is not necessary for use of "@"s, since "@"s may reside in command files. See READ for more information.

#### Formats:

Both the USING and PICTURE clauses have a format as their objective. The format is a series of characters that indicate which characters appear on the screen or page. The following table defines the characters and their functions:

Format character	SAY function	GET function
#	causes the next number to be output	allows only a digit (1, 2,...,8, 9, 0) and the characters ".", "+," "-", and " " (a space) to be entered
9	same as #	same as #
X	outputs the next character	allows any character to be entered
A	outputs the next character	allows only alpha to be entered
\$ or *	outputs either a digit or a \$ or * instead of leading zeros	output as is
!	no effect	converts lowercase alpha characters to uppercase

**Example:**

```
. @ 5,1 SAY 'ENTER PHONE NUMBER' GET PNO PICTURE '(999)999-9999'
```

The message 'ENTER PHONE NUMBER' would be displayed, followed by '(bbb) bbb-bbbb' (b indicates a blank), assuming that the value of PNO was all blanks prior to issuance. When (and if) the READ command is issued, only digits can be entered. The value of PNO after the READ command might well be '(213)555-5555' after editing. All of the non-functional characters in the PICTURE format are inserted into the variable. In this example, the parentheses, minus sign and the blank are non-functional.

```
. @ 10,50 SAY HOURS*RATE USING '$$$$$$.99'
```

This "@" command could be used with either the screen or the printer since it has no GET phrase. It might well be used to print payroll checks. The dollar signs will be printed as long as there are leading zeros in the item to be printed. If hours=40 and rate = 12.50, then '\$\$\$\$500.00' will be displayed. This feature is known as floating dollar and is valuable for printing checks that cannot be easily altered in value.

When commas are used in the integer part of a picture, they are replaced by the picture character in front of them, if there are no significant digits in the item to the left of where the comma would otherwise be placed.

```
. @ 10,50 SAY HOURS*RATE USING '$$$,$$$$.99'
```

Would output \$\$\$,\$500.00 and specifically not output \$\$\$,500.00.

Normally, a number of "@" commands are issued then, if any GET phrases were included, a READ command is issued to allow editing or data entry into the GET variables. In the following example, the screen is formatted with several "@"s and a database is filled with information according to these "@"s. The last record in the database will have a "0" in the field "name"; this is the record that will be deleted, since it is not necessary.

```

SET FORMAT TO SCREEN
USE F:EXAMPLE
ERASE
DO WHILE NAME # '0'
  APPEND BLANK
  @ 5.0 SAY "ENTER NEXT NAME":
      GET NAME PICTURE 'XXXXXXXXXXXXXXXXXXXXX'
  @ 6.0 SAY "ENTER TELEPHONE NUMBER":
      GET TELE:EXTSN PICTURE 'XXXXX'
  @ 6.40 SAY "ENTER MAIL STOP":
      GET MAIL.STOP PICTURE 'XXXXXXXXXX'
  READ
ENDDO
GOTO BOTTOM
DELETE
PACK
LIST
RETURN

```

The following commands affect the operation of the "@" command:

- SET INTENSITY ON/OFF (default is ON) affects the screen intensity of GETs, and SAYs.
- SET BELL ON/OFF (default is ON) affects the bell alarm when invalid characters are entered or a data boundary is crossed.
- SET COLON ON/OFF (default is ON) affects whether GET variables are bounded by colons.
- SET DEBUG ON/OFF (default is OFF) allows easier debugging of "@" commands by shifting ECHO and STEP messages to the printer.
- SET SCREEN ON/OFF (default is ON) allows use of full screen operations.
- SET FORMAT TO SCREEN/PRINT/<format file> determines device destination of output (SCREEN or PRINTER). SET FORMAT TO <format file> establishes a format file as the source of "@" commands for the READ command. SCREEN is the default value.
- READ enters the editing mode so that GET variables can be altered.

## ACCEPT ... 40

### ACCEPT

ACCEPT ["<string> "] TO <memvar>

This construct permits the entry of character strings into memory variables just as the INPUT command, but without the necessity of enclosing them in the quote marks required by the INPUT command. ACCEPT makes a memory variable of the type 'character' out of whatever is entered; INPUT determines the data type from the syntax of the entry and makes a memory variable of that type.

The <memvar> is created, if necessary, and the input character string is stored into <memvar>. If "<string>" is present, it is displayed on the screen, followed by a colon, as a prompt message before the input is accepted. If a carriage return is entered in response to an ACCEPT request, <memvar> will receive a single space character. Either single quotes, double quotes, or square brackets may be used to delimit the prompt string, however, both the beginning and ending marks must correspond.

#### Examples:

```
. ACCEPT "ENTER PERSONS NAME" TO NAM  
ENTER PERSON'S NAME:John Jones
```

```
. ACCEPT "ENTER PERSON'S NAME" TO NAM2  
ENTER PERSON'S NAME: Dave Smith
```

```
. DISP MEMO  
NAM          (C)          John Jones  
NAM2         (C)          Dave Smith  
** TOTAL **           02 VARIABLES USED           00020 BYTES USED
```

```
. ACCEPT TO ANY  
:ANY CHARACTERS
```

```
. DISP MEMO  
NAM          John Jones  
NAM2         Dave Smith  
ANY          ANY CHARACTERS  
** TOTAL **           03 VARIABLES USED           00034 BYTES USED
```





## APPEND . . . 42

When APPENDING in the full-screen mode, the SET CARRY ON command will cause all of the data from the previous record to be carried over to the next record. Changes can then be made. This is especially useful if successive records have a lot of common data.

If a SET FORMAT TO <file> is in effect, then APPEND will use the @-commands from the format file to form the full-screen and allow complete control of the screen and the data that will be appended. Otherwise, APPEND displays all fields in tabular form.

The APPEND command is especially useful when it is necessary to expand/contract fields or add/delete fields from an existing database. Using the CREATE command, set up a new database containing the desired structure and then APPEND the old database to the new. Fields which appear only in the new database will be blank filled.

### Examples:

```
. USE EXAMPLE
```

```
. DISPLAY STRUCTURE
```

```
STRUCTURE FOR FILE: EXAMPLE
```

```
NUMBER OF RECORDS: 00005
```

```
DATE OF LAST UPDATE: 12/31/82
```

```
PRIMARY USE DATABASE
```

FLD	NAME	TYPE	WIDTH	DEC
001	NAME	C	020	
002	TELE:EXTSN	C	005	
003	MAIL:STOP	C	010	
** TOTAL **			00036	

```
. DISPLAY ALL
```

00001	NEUMAN, ALFRED E.	1357	123/456
00002	RODGERS, ROY	2468	180/103
00003	CASSIDY, BUTCH	3344	264/401
00004	CHANG, LEE	6743	190/901
00005	POST, WILEY	1011	84/13B

```
. APPEND
```

```
RECORD 00006
```

```
NAME: LANCASTER, WILLIAM J
```

```
TELE:EXTSN: 6623
```

```
MAIL:STOP: 170/430
```

```
RECORD 00007
```

```
NAME: NORRIS, R. "BOB"
```

```
TELE:EXTSN: 8093
```

```
MAIL:STOP: 427/396
```

RECORD 00008

NAME: (cr)

. DISPLAY ALL OFF NAME, TELE:EXTSN	
NEUMAN, ALFRED E.	1357
RODGERS, ROY	2468
CASSIDY, BUTCH	3344
CHANG, LEE	6743
POST, WILEY	1011
LANCASTER, WILLIAM J.	6623
NORRIS, R. "BOB"	8093

. APPEND FROM DUPE3  
00007 RECORDS ADDED

. DISPLAY ALL		
00001 NEUMAN, ALFRED E.	1357	123/456
00002 RODGERS, ROY	2468	180/103
00003 CASSIDY, BUTCH	3344	264/401
00004 CHANG, LEE	6743	190/901
00005 POST, WILEY	1011	84/13B
00006 LANCASTER, WILLIAM J	6623	170/430
00007 NORRIS, R. "BOB"	8093	427/396
00008 NEUMAN, ALFRED E.	1357	
00009 RODGERS, ROY	2468	
00010 CASSIDY, BUTCH	3344	
00011 CHANG, LEE	6743	
00012 POST, WILEY	1011	
00013 LANCASTER, WILLIAM J.	6623	
00014 NORRIS, R. "BOB"	8093	

. APPEND BLANK

. REPLACE NAME WITH 'RINEHART, RALPH'

00001 REPLACEMENT(S)

. DISPLAY

00015 RINEHART, RALPH

APPEND . . . 44

. DISPLAY ALL NAME, 'ex =', TELE:EXTSN

00001	NEUMAN, ALFRED E.	ex = 1357
00002	RODGERS, ROY	ex = 2468
00003	CASSIDY, BUTCH	ex = 3344
00004	CHANG, LEE	ex = 6743
00005	POST, WILEY	ex = 1011
00006	LANCASTER, WILLIAM J.	ex = 6623
00007	NORRIS, R. "BOB"	ex = 8093
00008	NEUMAN, ALFRED E.	ex = 1357
00009	RODGERS, ROY	ex = 2468
00010	CASSIDY, BUTCH	ex = 3344
00011	CHANG, LEE	ex = 6743
00012	POST, WILEY	ex = 1011
00013	LANCASTER, WILLIAM J.	ex = 6623
00014	NORRIS, R. "BOB"	ex = 8093
00015	RINEHART, RALPH	ex =

. USE B:SHOPLIST

. DISP STRU

STRUCTURE FOR FILE: B:SHOPLIST.DBF

NUMBER OF RECORDS: 00009

DATE OF LAST UPDATE: 06/22/82

PRIMARY USE DATABASE

FLD	NAME	TYPE	WIDTH	DEC
001	ITEM	C	020	
002	NO	N	005	
003	COST	N	010	002
** TOTAL **			00036	

. CREATE

FILENAME: NEWSHOP

ENTER RECORD STRUCTURE AS FOLLOWS:

FIELD	NAME,TYPE,WIDTH, DECIMAL PLACES
001	ITEM, C, 25
002	NO,N,5
003	COST,N,10,2
004	NEED:DATE,C,8
005	(cr)

INPUT NOW? N

. USE NEWSHOP

. APPEND FROM B:SHOPLIST

00009 RECORDS ADDED

```
. LIST
00001 BEANS                5          0.75
00002 BREAD LOAVES        2          0.97
00003 T-BONE              4          3.94
00004 PAPER PLATES        1          0.86
00005 PLASTIC FORKS       5          0.42
00006 LETTUCE             2          0.53
00007 BLEU CHEESE         1          1.96
00008 MILK                 2          1.30
00009 CHARCOAL            2          0.75
```

```
. REPLACE ALL NEED:DATE WITH ' 7/4/82'
00009 REPLACEMENT(S)
```

```
. LIST
00001 BEANS                5          0.75          7/ 4/82
00002 BREAD LOAVES        2          0.97          7/ 4/82
00003 T-BONE              4          3.94          7/ 4/82
00004 PAPER PLATES        1          0.86          7/ 4/82
00005 PLASTIC FORKS       5          0.42          7/ 4/82
00006 LETTUCE             2          0.53          7/ 4/82
00007 BLEU CHEESE         1          1.96          7/ 4/82
00008 MILK                 2          1.30          7/ 4/82
00009 CHARCOAL            2          0.75          7/ 4/82
```

(The following example demonstrates the DELIMITED file append. This file could have been created by a number of different versions of BASIC)

```
'BARNETT, WALT',31415,6
'NICHOLS, BILL',76767,17
'MURRAY, CAROL',89793,4
'WARD, CHARLES A.',92653,15
'ANDERSON, JAMES REGINALD III',11528', 16
```

(Append the file into a dBASE-structured database)

```
. USE ORDERS
```

```
. DISP STRU
STRUCTURE FOR FILE:  ORDERS.DBF
NUMBER OF RECORDS:  00008
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
```

FLD	NAME	TYPE	WIDTH	DEC
001	CUSTOMER	C	020	
002	PART:NO	C	005	
003	AMOUNT	N	005	
** TOTAL **			00031	

**APPEND . . . 46**

```
. LIST
00001 SWARTZ, JOE          31415          13
00002 SWARTZ, JOE          76767          13
00003 HARRIS, ARNOLD       11528          44
00004 ADAMS, JEAN          89793          12
00005 MACK, JAY            31415           3
00006 TERRY, HANS         76767           5
00007 JUAN, DON            21828           5
00008 SALT, CLARA          70296           9
```

```
. APPEND FROM DELIM.DAT DELIMITED
00005 RECORDS ADDED
```

```
. LIST
00001 SWARTZ, JOE          31415          13
00002 SWARTZ, JOE          76767          13
00003 HARRIS, ARNOLD       11528          44
00004 ADAMS, JEAN          89793          12
00005 MACK, JAY            31415           3
00006 TERRY, HANS         76767           5
00007 JUAN, DON            21828           5
00008 SALT, CLARA          70296           9
00009 BARNETT, WALT        31415           6
00010 NICHOLS, BILL        76767          17
00011 MURRAY, CAROL        89793           4
00012 WARD, CHARLES A.     92653          15
00013 ANDERSON, JAMES REGI 11528          16
```

(The following examples demonstrate an APPEND FROM <file> FOR <exp>. Note that the fields in the FOR are in the USE file also.)

```
. USE CHECKS
. DISP STRU
STRUCTURE FOR FILE: CHECKS.DBF
NUMBER OF RECORDS: 00013
DATE OF LAST UPDATE: 10/18/82
PRIMARY USE DATABASE
FLD      NAME              TYPE      WIDTH      DEC
001      NUMBER             N         005
002      RECIPIENT             C         020
003      AMOUNT              N         010          002
004      HOME                L         001
005      OUTGOING             L         001
** TOTAL **                      00038
```

```
. LIST
00001      1 Phone Company      104.89 .F. .T.
00002      2 Gas Company        4.14 .F. .T.
00003      3 Electricity          250.31 .F. .T.
00004      4 Grocery Store      1034.45 .F. .T.
00005      34 Me                561.77 .T. .F.
00006      6 Bank, service charge  4.00 .T. .T.
00007      7 Doctor Doolittle     100.00 .T. .T.
00008      8 Pirates                101.01 .F. .T.
00009      9 Car Repair Man      500.01 .T. .T.
00010     10 Me                561.01 .T. .F.
00011     11 Tupperware         50.02 .F. .T.
00012     12 Me                561.77 .T. .F.
00013     13 Me                750.03 .T. .F.
```

```
. USE MONTH
. DISP STRU
STRUCTURE FOR FILE: MONTH.DBF
NUMBER OF RECORDS: 00003
DATE OF LAST UPDATE: 10/18/82
PRIMARY USE DATABASE
```

FLD	NAME	TYPE	WIDTH	DEC
001	NUMBER	N	005	
002	AMOUNT	N	010	002
003	HOME	L	001	
** TOTAL **			00017	

```
. LIST
00001      29                14.89 .T.
00002      16                764.09 .T.
00003      78                97.96 .T.
```

```
. APPEND FROM CHECKS FOR HOME
00006 RECORDS ADDED
```

```
. APPEND FROM CHECKS FOR OUTGOING
*** SYNTAX ERROR ***
```

?

```
APPEND FROM CHECKS FOR OUTGOING
CORRECT AND RETRY (Y/N)? N
```

**That last append was to show what would happen if the FOR field was not in the USE file.**

## BROWSE... 48

### BROWSE

BROWSE [FIELDS < field list >]

The BROWSE command is one of the most powerful dBASE commands for data editing and viewing. The data from up to 19 records is displayed onto the screen (fewer if fields are greater than 80 characters). As many fields as will fit are put on each line. The screen should be considered as a window into a database. You can scroll backwards and forwards through the records and you can pan left and right through the fields of the database. Any data can be edited with the standard full-screen editing method (see section 8 for additional information).

If no <field list> is supplied, then BROWSE will show all fields in the same order as the structure.

This is a summary of the full-screen control keys that will work in BROWSE:

left arrow, or ctl-E, backs up to the previous data field;  
right arrow, or ctl-X, advances to the next data field;

ctl-D advances to the next character;  
ctl-S backs up to the last character;

<DELETE> or ctl-G deletes the character under the cursor;  
<BACKSPACE> deletes the character before the cursor;

ctl-Q exits without saving the changes;  
ctl-W exits and saves the changes;

ctl-B pans the window right one field;  
ctl-Z pans the window left one field;

ctl-C writes the current record and advances one record;  
ctl-R writes the current record and backs up one record;

ctl-U switches (toggles) the current record between being marked for deletion and not being marked.

#### Example:

. BROWSE

. BROWSE FIELDS NAME, ADDRESS, ZIP, COMPANY



**CANCEL**

CANCEL

Cancel a command file execution and return to the normal keyboard interpretive mode.

**Example:**

```
INPUT 'IS JOB DONE (Y/N)' TO X
IF X
  CANCEL
ENDIF
```

This is a fragment from a command file. The INPUT command asks for a yes/no answer. If the answer is yes ('Y', 'y', 'T', or 't') then the IF X line of the command file will be satisfied (since X will be logically .TRUE.) and the CANCEL command will be executed.

## CHANGE... 50

### CHANGE

CHANGE [<scope>] FIELD <list> [FOR <exp>]

CHANGE is a command that allows the user to make a number of alterations to a database with minimum effort. All database fields that are referenced in the list are presented to the user in the order given by <list>. The user has the opportunity of entering new data, modifying the data or skipping to the next field. When the <list> has been exhausted, CHANGE will proceed to the next record as specified in the <scope>. The default scope is the current record.

A field can be deleted in its entirety by typing a control-Y (followed by <enter> in response to the CHANGE? message. The CHANGE command can be aborted by typing an ESCAPE character.

#### Example:

```
. USE CARDS  
. CHANGE FIELD DATE
```

```
RECORD: 00001
```

```
DATE: 08/19/81  
CHANGE? 81  
TO      82
```

```
DATE: 08/19/82  
CHANGE? (cr)
```

**CLEAR**

## CLEAR [GETS]

If the GETS (or GET) keyword is used, then all of the GETs that are pending (i.e. a GET set up by the @ command) are cleared and the screen is left intact. This is opposed to the ERASE command which clears pending GETs and also erases the screen.

If there is no GETS keyword, then this command resets dBASE II. All databases in USE are closed and un-used, all memory variables are released, and the PRIMARY work area is re-selected.

This command gives dBASE II a "clean slate." For instance: if a command file finished executing and left dBASE in the SECONDARY state, then executing a new command file that assumes that the PRIMARY state was selected will cause unknown things to happen.

CLEAR should be used at the beginning of a command file to give the command file a known state.

**Example:**

```
.CLEAR
```

**CONTINUE... 52**

**CONTINUE**

This command is used with the LOCATE command. LOCATE and CONTINUE may be separated by other commands, however there are limitations. See the LOCATE command for more information.



**COPY**

COPY TO <file> [<scope>] [FIELD <fieldlist>] [FOR <exp>] [SDF]  
 [WHILE <exp>] [DELIMITED [WITH <delimiter>]]

COPY TO <file> STRUCTURE [EXTENDED]  
 [FIELD<fieldlist>]

This command copies the database in USE to another file. The <file> may be in dBASE format or in the System Data Format (if the SDF option is specified).

If the STRUCTURE clause is specified, then only the structure of a dBASE file in USE is copied to the "TO" file.

If a list of fields is supplied following a FIELD clause, then only those data fields are copied TO the file. For the COPY STRUCTURE FIELD <list>, only the structure of the listed fields is copied TO the file. In either case, the new structure will be made up of only those fields specified by the FIELD clause. No FIELD clause specifies that all fields will be copied.

If the SDF clause is specified, then the file in USE is copied to another file without the structure. This new file will be in ASCII standard format. This allows the generation of files which can be input to processors other than dBASE. The STRUCTURE and SDF clauses are mutually exclusive.

If the DELIMITED keyword is also in the command, then the output file will have all of its character string type fields enclosed in quotes and the fields will be separated by commas. This is the converse of a delimited APPEND. By default, the DELIMITED type of COPY uses single quotes as delimiters to mark character string fields. The WITH sub-phrase of the DELIMITED phrase allows any character to be the delimiter. If a "," is used as the delimiter, then the character fields will have trailing blanks trimmed, the numeric fields will have the leading blanks trimmed, and the character strings will not be enclosed in quotes. The APPEND command will only respond to single and double quotes.

If either the DELIMITED or SDF option is used; then the output <file> name will default to a .TXT extension, otherwise the output file will default to a .DBF extension.

The "TO" file is created if it does not exist, and will destroy any existing file of the same name.

COPY TO <file> STRUCTURE [EXTENDED] [FIELD] <fieldlist> ]

Inclusion of the EXTENDED clause in this command copies the structure of the database file in USE to the specified database as records. The structure may then be examined during program execution.

**Example:**

```
. USE VTESTDB
. DISP STRU
```

STRUCTURE FOR FILE: C:\VTESTDB.DBF  
NUMBER OF RECORDS: 00008  
DATE OF LAST UPDATE: 12/20/82  
PRIMARY USE DATABASE

FLD	NAME	TYPE	WIDTH	DEC
001	CHAR	C	010	
002	LOG	L	001	
003	NUMDEC	N	010	002
004	NUMINT	N	010	
** TOTAL **			00032	

. COPY STRUCTURE EXTENDED TO TEST  
00004 RECORDS COPIED

. USE TEST  
. DISP STRU

STRUCTURE FOR FILE: C:\TEST.DBF  
NUMBER OF RECORDS: 00004  
DATE OF LAST UPDATE: 12/20/82  
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	CHAR	C	10	0
00002	LOG	L	1	0
00003	NUMDEC	N	10	2
00004	NUMINT	N	10	0

(Modifications could be made here to the data (which is a structure)).

**Examples:**

```
. DISPLAY ALL OFF NAME, TELE:EXTSN
NEUMAN, ALFRED E.          1357
RODGERS, ROY              2468
CASSIDY, BUTCH           3344
CHANG, LEE                6743
POST, WILEY              1011
LANCASTER, WILLIAM J.    6623
NORRIS, R. "BOB"         8093
```

. DISPLAY STRUCTURE  
 STRUCTURE FOR FILE: EXAMPLE  
 NUMBER OF RECORDS: 00007  
 DATE OF LAST UPDATE: 12/14/82  
 PRIMARY USE DATABASE

FLD	NAME	TYPE	WIDTH	DEC
001	NAME	C	020	
002	TELE:EXTSN	C	005	
003	MAIL:STOP	C	010	
** TOTAL **			00036	

. COPY TO DUPE  
 00007 RECORDS COPIED

. COPY TO DUPE2 FOR TELE:EXTSN < '8000'  
 00006 RECORDS COPIED

. USE DUPE2

. DISPLAY ALL

00001	NEUMAN, ALFRED E.	1357	123/456
00002	RODGERS, ROY	2468	180/103
00003	CASSIDY, BUTCH	3344	264/401
00004	CHANG, LEE	6743	190/901
00005	POST, WILEY	1011	84/13B
00006	LANCASTER, WILLIAM J.	6623	170/430

. USE EXAMPLE

. COPY FIELD NAME, TELE:EXTSN TO DUPE3  
 00007 RECORDS COPIED

. USE DUPE3

STRUCTURE FOR FILE: DUPE3  
 NUMBER OF RECORDS: 00007  
 DATE OF LAST UPDATE: 12/20/82  
 PRIMARY USE DATABASE

FLD	NAME	TYPE	WIDTH	DEC
001	NAME	C	020	
002	TELE:EXTSN	C	005	
** TOTAL **			00026	

. DISPLAY ALL

00001	NEUMAN, ALFRED E.	1357
00002	RODGERS, ROY	2468
00003	CASSIDY, BUTCH	3344
00004	CHANG, LEE	6743
00005	POST, WILEY	1011
00006	LANCASTER, WILLIAM J.	6623
00007	NORRIS, R. "BOB"	8093

. USE EXAMPLE

. COPY NEXT 4 TO DUPE5

00004 RECORDS COPIED

. USE DUPE5

. DISPLAY ALL

00001	NEUMAN, ALFRED E.	1357	123/456
00002	RODGERS, ROY	2468	180/103
00003	CASSIDY, BUTCH	3344	264/401
00004	CHANG, LEE	6743	190/901

(The delimited COPY)

. USE ORDERS

. DISP STRUCTURE

STRUCTURE FOR FILE: ORDERS.DBF

NUMBER OF RECORDS: 00012

DATE OF LAST UPDATE: 07/01/82

PRIMARY USE DATABASE

FLD	NAME	TYPE	WIDTH	DEC
001	CUSTOMER	C	020	
002	PART:NO	C	005	
003	AMOUNT	N	005	
** TOTAL **			00031	

. LIST

00001	SWARTZ, JOE	31415	13
00002	SWARTZ, JOE	76767	13
00003	HARRIS, ARNOLD	11528	44
00004	ADAMS, JEAN	89793	12
00005	MACK, JAY	31415	3
00006	TERRY, HANS	76767	5
00007	JUAN, DON	21821	5
00008	SALT, CLARA	70296	9
00009	BARNETT, WALT	31415	6
00010	NICHOLS, BILL	76767	17
00011	MURRAY, CAROL	89793	4
00012	WARD, CHARLES A.	92653	15



. COPY TO DELIM.DAT DELIMITED  
00012 RECORDS COPIED

'SWARTZ, JOE	' , '31415' ,	13
'SWARTZ, JOE	' , '76767' ,	13
'HARRIS, ARNOLD	' , '11528' ,	44
'ADAMS, JEAN	' , '89793' ,	12
'MACK, JAY	' , '31415' ,	3
'TERRY, HANS	' , '76767' ,	5
'JUAN, DON	' , '21828' ,	5
'SALT, CLARA	' , '70296' ,	9
'BARNETT, WALT	' , '31415' ,	6
'NICHOLS, BILL	' , '76767' ,	17
'MURRAY, CAROL	' , '89793' ,	4
'WARD, CHARLES A.	' , '92653' ,	15

## COUNT ... 58

### COUNT

```
COUNT [< scope> ] [FOR< exp> ] [TO< memvar> ]  
      [WHILE< exp> ]
```

Count the number of records in the USE file. If the FOR clause is invoked, then only the number of records which satisfy the expression are counted. If the TO clause is included, the integer count is placed into a memory variable. The memory variable will be created if it did not exist prior to this command. Count will count deleted records if DELETED is set OFF, and ignore them if DELETED is set ON.

dBASE responds with the message:

```
COUNT = xxxxx
```

### Examples:

```
. USE INVNTY
```

```
. DISPLAY STRUCTURE
```

```
STRUCTURE FOR FILE: INVNTY
```

```
NUMBER OF RECORDS: 00010
```

```
DATE OF LAST UPDATE: 10/23/82
```

```
PRIMARY USE DATABASE
```

FLD	NAME	TYPE	WIDTH	DEC
001	ITEM:NO	N	006	
002	CLASS:NO	N	003	
003	VENDOR:NO	N	005	
004	DESCR	C	013	
005	UNIT:COST	N	007	002
006	LOCATION	C	005	
007	ON:HAND	N	004	
008	SOLD	N	004	
009	PRICE	N	007	002
** TOTAL **			00055	

```
. DISPLAY ALL
```

00001	136928	13	1673	ADJ. WRENCH	7.13	189	9	0	9.98
00002	221679	9	1673	SM. HAND SAW	5.17	173	4	1	7.98
00003	234561	0	96	PLASTIC ROD	2.18	27	112	53	4.75
00004	556178	2	873	ADJ. PULLEY	22.19	117	3	0	28.50
00005	723756	73	27	ELEC. BOX	19.56	354	6	1	29.66
00006	745336	13	27	FUSE BLOCK	12.65	63	7	2	15.95
00007	812763	2	1673	GLOBE	5.88	112	5	2	7.49
00008	876512	2	873	WIRE MESH	3.18	45	7	3	4.25
00009	915332	2	1673	FILE	1.32	97	7	3	1.98
00010	973328	0	27	CAN COVER	0.73	21	17	5	0.99

. COUNT  
COUNT = 00010

. COUNT FOR ITEM:NO> 500000  
COUNT = 00007

. COUNT FOR 'ADJ'\$DESCR  
COUNT = 00002

. GOTO TOP

. COUNT FOR PRICE< 10 NEXT 6  
COUNT = 00003

. GOTO TOP

. COUNT NEXT 6 FOR PRICE< 10  
COUNT = 00003

. USE B:SHOPLIST

. LIST

00001	BEANS	5	0.75
00002	BREAD LOAVES	2	0.97
00003	T-BONE	4	3.94
00004	PAPER PLATES	1	0.86
00005	PLASTIC FORKS	5	0.42
00006	LETTUCE	2	0.53
00007	BLEU CHEESE	1	1.96
00008	MILK	2	1.30
00009	CHARCOAL	2	0.75

. DISPLAY STRUCTURE

STRUCTURE FOR FILE: B:SHOPLIST.DBF  
NUMBER OF RECORDS: 00009  
DATE OF LAST UPDATE: 12/10/82  
PRIMARY USE DATABASE

FLD	NAME	TYPE	WIDTH	DEC
001	ITEM	C	020	
002	NO	N	005	
003	COST	N	010	002
** TOTAL **			00036	

. COUNT TO XX FOR COST> 1  
COUNT = 0003

. ? XX  
3

## CREATE . . . 60

### CREATE

```
CREATE [<filename>]  
      [<filename> FROM <FILENAME>]
```

A new dBASE structured file is CREATED. The user provides the structure, field names, and file name for the database file.

If not supplied in the command, the user is first prompted for the <filename> to be used by the message:

FILENAME:

The user enters a valid filename with the following added restriction: the filename may contain no special characters other than those normally used by MS DOS for special purposes (such as B: to denote disk drive "B").

dBASE is now ready to accept the structure of the data base from the user. The following message is displayed:

```
ENTER RECORD STRUCTURE AS FOLLOWS:  
FIELD      NAME,TYPE,WIDTH,DECIMAL PLACES  
001
```

The user now enters field names and associated structure information. A field name is a character string up to 10 characters long which consists of alphabetic letters, numeric digits, and colons. Field names must begin with an alphabetic character. CREATE will check for and prevent duplicate field names. Fields may be any of three types: character string, numeric, or logical. The type field is specified by one character, as:

- C — character string
- N — numeric
- L — logical

The width refers to the length of the field; for instance, a character string may be 20 characters long (i.e. its width is 20). Numeric data may be either integer or decimal. The width of integers is the maximum number of digits that they may be expected to contain. For decimal numbers, two widths are required; the first is the maximum number of digits that the decimal number is expected to contain (including the decimal point), the second width is the number of digits which are to be allowed on the right side of the decimal point. Logical data may only be of length 1.

The CREATE <filename> FROM <filename> command creates a new file by reading the structure from the records of the FROM file. The contents of the FROM file may be input by the COPY STRUCTURE EXTENDED command, or by the usual data entry methods.

**Example:**

```
. USE TEST
. DISP STRU
```

```
STRUCTURE FOR FILE:  C:TEST.DBF
NUMBER OF RECORDS:  00004
DATE OF LAST UPDATE: 12/14/82
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	CHAR	C	10	0
00002	LOG	L	1	0
00003	NUMDEC	N	10	2
00004	NUMINT	N	10	0

(Modifications could be made here to the data [which is a structure])

```
. CREATE TEST2 FROM TEST
. USE TEST2
. DISP STRU
```

```
STRUCTURE FOR FILE:  C:TEST2.DBF
NUMBER OF RECORDS:  00000
DATE OF LAST UPDATE: 12/14/82
PRIMARY USE DATABASE
```

FLD	NAME	TYPE	WIDTH	DEC
001	CHAR	C	010	
002	LOG	L	001	
003	NUMDEC	N	010	002
004	NUMINT	N	010	
** TOTAL **			00032	

```
. CREATE
FILENAME: EXAMPLE
ENTER RECORD STRUCTURE AS FOLLOWS:
FIELD          NAME,TYPE,WIDTH, DECIMAL PLACES
001            NAME,C,20
002            TELE:EXTSN,C,5
003            MAIL:STOP,C,10
004            (cr)
INPUT DATA NOW?  Y
```

CREATE... 62

RECORD 00001

NAME NEUMAN, ALFRED E.  
TELE:EXTSN: 1357  
MAIL:STOP: 123/456

RECORD 00002

NAME: RODGERS, ROY  
TELE:EXTSN: 2468  
MAIL:STOP: 180/103

RECORD 00003

NAME: CASSIDY, BUTCH  
TELE:EXTSN: 3344  
MAIL:STOP: 264/401

RECORD 00004

NAME CHANG, LEE  
TELE:EXTSN 6743  
MAIL:STOP 190/901

RECORD 00005

NAME POST, WILEY  
TELE:EXTSN: 1011  
MAIL:STOP: 84/13B

RECORD 00006

NAME: (cr)

. DISPLAY STRUCTURE  
NO DATABASE FILE IN USE, ENTER FILENAME: EXAMPLE  
STRUCTURE FOR FILE: EXAMPLE  
NUMBER OF RECORDS: 00005  
DATE OF LAST UPDATE: 12/14/82  
PRIMARY USE DATABASE

FLD	NAME	TYPE	WIDTH	DEC
001	NAME	C	020	
002	TELE:EXTSN	C	005	
003	MAIL:STOP	C	010	
** TOTAL **			00036	

. DISPLAY ALL

00001	NEUMAN, ALFRED E.	1357	123/456
00002	RODGERS, ROY	2468	180/103
00003	CASSIDY, BUTCH	3344	264/401
00004	CHANG, LEE	6743	190/901
00005	POST, WILEY	1011	84/13B



## DELETE... 64

### DELETE

```
DELETE [<scope>] [FOR <exp> ]  
      [WHILE <exp> ]  
DELETE FILE <filename>
```

All records which are within <scope> (and which satisfy the FOR expression if present) are marked for deletion. The default scope is the current record only. Records are not physically deleted until a PACK operation; however records marked for deletion will not be copied, appended, or sorted. The RECALL operation may be used to revive records marked as deleted. Records which are marked for deletion can be displayed. The mark of deletion appears as an asterisk between the record number and the first field. See also the SET DELETED ON/OFF command.

In the second form, the file named <filename > will be removed from the disk drive where it resides (if possible), and the space it was occupying will be released to the operating system for reassignment. If, however, the <filename> is currently in use, the file will not be deleted.

#### Examples:

```
. LIST  
00001 136928 13 1673  ADJ. WRENCH      7.13 189   9  0  9.98  
00002 221679  9 1673  SM. HAND SAW    5.17 173   4  1  7.98  
00003 234561  0   96  PLASTIC ROD     2.18  27  112 53  4.75  
00004 556178  2  873  ADJ. PULLEY    22.19 117   3  0 28.50  
00005 723756 73   27  ELECT. BOX     19.56 354   6  1 29.66  
00006 745336 13   27  FUSE BLOCK     12.65  63   7  2 15.95  
00007 812763  2 1673  GLOBE          5.88 112   7  3  7.49  
00008 876512  2  873  WIRE MESH      3.18  45   7  3  4.25  
00009 915332  2 1673  FILE           1.32  97   7  3  1.98
```

```
. DELETE RECORD 2  
00001 DELETION(S)
```

```
.5
```

```
. DELETE NEXT 3  
00003 DELETION(S)
```



. LIST

00001	136928	13	1673	ADJ. WRENCH	7.13	189	9	0	9.98
00002	*221679	9	1673	SM. HAND SAW	5.17	173	4	1	7.98
00003	234561	0	96	PLASTIC ROD	2.18	27	112	53	4.75
00004	556178	2	873	ADJ. PULLEY	22.19	117	3	0	28.50
00005	*723756	73	27	ELECT. BOX	19.56	354	6	1	29.66
00006	*745336	13	27	FUSE BLOCK	12.65	63	7	2	15.95
00007	*812763	2	1673	GLOBE	5.88	112	5	2	7.49
00008	876512	2	873	WIRE MESH	3.18	45	7	3	4.25
00009	915332	2	1673	FILE	1.32	97	7	3	1.98

. RECALL ALL

00004 RECALL(S)

. LIST

00001	136928	13	1673	ADJ. WRENCH	7.13	189	9	0	9.98
00002	221679	9	1673	SM. HAND SAW	5.17	173	4	1	7.98
00003	234561	0	96	PLASTIC ROD	2.18	27	112	53	4.75
00004	556178	2	873	ADJ. PULLEY	22.19	117	3	0	28.50
00005	723756	73	27	ELECT. BOX	19.56	354	6	1	29.66
00006	745336	13	27	FUSE BLOCK	12.65	63	7	2	15.95
00007	812763	2	1673	GLOBE	5.88	112	5	2	7.49
00008	876512	2	873	WIRE MESH	3.18	45	7	3	4.25
00009	915332	2	1673	FILE	1.32	97	7	3	1.98

. DISP FILES ON B

DATABASE FILES	# RCDS	LAST UPDATE
SHOPLIST .DBF	00007	06/06/82
SHOPSAVE .DBF	00007	06/05/82

. DELETE FILE B: SHOPSAVE

FILE HAS BEEN DELETED

. DISPLAY FILES ON B

DATABASE FILES	# RCDS	LAST UPDATE
SHOPLIST .DBF	00007	06/06/82



**Examples:**

. USE B:INVENTORY

. DISPLAY STRUCTURE

STRUCTURE FOR FILE: B:INVENTORY.DBF

NUMBER OF RECORDS: 00008

DATE OF LAST UPDATE: 12/18/82

PRIMARY USE DATABASE

FLD	NAME	TYPE	WIDTH	DEC
001	ITEM	C	020	
002	COST	N	010	002
003	PART:NO	C	005	
004	ON:HAND	N	005	

\*\* TOTAL \*\*

00041 BYTES

(note: total includes 1 overhead byte)

. DISPLAY ALL ITEM, PART:NO, COST\*ON:HAND, \$(PART:NO,1,2) FOR :

COST > 100 .AND. ON:HAND > 2 OFF

TANK, SHERMAN	89793	404997.00	89
TROMBONES	76767	15076.12	76
RINGS, GOLDEN	70296	1000.00	70

. DISPLAY MEMORY

CLIENT:NAM (C) DANGLEMEYER, PRENTICE

BUDGET (N) 123456.70

EF:STATUS (L) .T.

\*\* TOTAL \*\*

03 VARIABLES USED

00027 BYTES USED

. DISPLAY FILES ON B: LIKE \*.FRM

TEST.FRM ADMIN.FRM ORDERS.FRM

. DISPLAY FILES

DATABASE FILES		# RCDS	LAST UPDATE
TEST	.DBF	00077	00 00 00
ADRECS	.DBF	00073	09 23 81
HISTSTR	.DBF	00000	06 29 81
TMPADMIN	.DBF		

NOT A dBASE II DATABASE

The last .DBF file in the list above is the file that is not a dBASE database.

Only representative examples of DISPLAY are given here; refer to other commands for other examples.

**DO**

- a. DO < file>
- b. DO WHILE < exp>  
    < statements>  
    ENDDO
- c. DO CASE

In case a. < file> is opened and read. The file in this case is known as a **COMMAND FILE**. It consists entirely of dBASE commands. The input is interpreted and executed as keyboard commands are. DO's can be stacked up to 16 deep (i.e. command files can contain DO commands which invoke other command files). Control is released by a command file with an end-of-file or by the RETURN command. If the current command file was called by a command file, control will be given back to the higher level command file. If, during the execution of a command file, a CANCEL command is encountered, all command files are closed and the keyboard is made the source for future commands.

In case b. if the < exp> evaluates as a logical TRUE, the statements following the DO are executed until an ENDDO statement is encountered. If the < exp> evaluates to a logical FALSE, control is transferred to the statement following the ENDDO statement.

**Note:** < statements> refers to entire statements. The DO WHILE statement ends with an ENDDO. Statements must nest properly; if there is an IF "inside" a DO WHILE, then an ENDDO may not occur before the ENDIF. See section 9.2 Rule 8 for more information.

**Examples:**

```
DO ACCNTPAY

DO WHILE .NOT.EOF
  DISPLAY NAME
  .
  .
  .
  SKIP
ENDDO
```

CASE is an extension of the DO command and takes the form shown above. There is no limit to the number of CASE phrases that a DO CASE may contain. The OTHERWISE phrase is optional.

DO CASE is a structured procedure. The individual CASEs in the construct could be viewed as the exceptions to the rule that defines the OTHERWISE. If some condition needs some special processing, then the condition would be a CASE and all other conditions would be the OTHERWISE. OTHERWISE may also be viewed as the default condition. See the first example below.

How dBASE handles the DO CASE construct may best be explained as a series of IFs. That is, dBASE will execute the DO CASE as if it were a list of IF-ENDIFs.

DO CASE	IF ITEM='ORANGES'
CASE ITEM='ORANGES'	any statements
any statements	ELSE
CASE ITEM='APPLES'	IF ITEM='APPLES'
any statements	any statements
OTHERWISE	ELSE
any statements	any statements
ENDCASE	ENDIF
	ENDIF

Thus, dBASE will examine the <exp>s in the individual CASEs and the first one that is true will have the statements after it executed. When dBASE reaches the next phrase beginning with a "CASE," it will exit to the ENDCASE. This means that if more than one CASE is true, only the first one will be executed.

If the OTHERWISE clause is present and none of the CASEs are true, then the <statements> in the OTHERWISE clause will be executed. If there is no OTHERWISE clause and none of the CASEs are true, then the DO CASE will be exited with none of the <statements> executed at all.

Any statements that are placed between the "DO CASE" and the first "CASE" will not be executed.

#### Examples:

```
DO CASE
  CASE ITEM = "BROWN"
    < statements > that process BROWN
  CASE ITEM = "JONES"
    < statements > that process JONES
  CASE ITEM = "SMITH"
    < statements > that process SMITH
  OTHERWISE
    < statements > that process all the other names
ENDCASE
```

In the case above, all the expressions were for the same field name. This is not necessary. An <exp> may contain anything and the series of CASEs need not have a tight relationship.

## DO... 70

```
DO CASE
  CASE TODAY = "MONDAY"
    <statements> for MONDAY
  CASE WEATHER = "RAIN"
    <statements> for RAIN
  CASE CITY = "LOS ANGELES"
    <statements> for LOS ANGELES
ENDCASE
```

Of course, if it is a rainy Monday in Los Angeles only the CASE for MONDAY will be executed.

CASEs need not be all character strings as in these two examples. Any expression will work.

```
DO CASE
  CASE 3 = 2 + 1
    <statements> for addition
  CASE .NOT. A
    <statements> for boolean logic
  CASE "A"$"ABCDEF"
    <statements> for string logic
  OTHERWISE
    <statements>
ENDCASE
```

**EDIT**

EDIT [n]

The EDIT command allows the user to selectively change the contents of the data fields in a database.

Editing can be done by either "EDIT" or "EDIT n" (n represents the record to be edited). If n is not present, then dBASE will ask for the record number to be edited. When EDIT is used, dBASE responds with:

ENTER RECORD # :

See section 8, full-screen operations, for a description of control keys and cursor movement.

If a SET FORMAT TO < file > is in effect, then EDIT will use the @- commands from the format file to form the full-screen and allow complete control of the screen and the data that will be appended. Otherwise, EDIT displays all fields in tabular form.

The editing of a data field can be aborted by entering a ctl-Q character. This discards any editing done and restores the data field to its original contents.

If an INDEXed file is being EDITed and the index clause was USEd, the dBASE will adjust the index if the key field is altered. If more than one index file is associated with the database, then the un-USEd files will be unaffected by the edit.



**{This page is intentionally left blank}**







**(This page is intentionally left blank)**



## EJECT... 74

### EJECT

EJECT

This command causes the printer to do a form feed (eject the page) if either PRINT is SET ON or FORMAT is SET TO PRINT. When using the @ command to do direct page formatting, the EJECT command also zeros the line and column registers. See also the SET EJECT ON/OFF command.

#### Example:

. EJECT

**ERASE**

ERASE

This command clears the screen and places the cursor in the upper left corner of the screen. When using the @ command with the SET SCREEN ON in effect, ERASE clears memory of prior @ command GETS and PICTURES.

**Example:**

```
. ERASE
```

**FIND**

FIND < char string > or '< char string >'

This command causes dBASE to FIND the first record in an indexed database (in USE) whose key is the same as < char string >. FIND allows very rapid location of records within an indexed database. A typical FIND time is two seconds on a floppy diskette system.

FIND operates only on databases that have previously been indexed (see the INDEX command description). If the INDEX command used a character string expression as the key, then FIND will operate when it is given only the first few characters of the key. The found record will be the first one whose key has the same order and number of characters as the < char string >. For example: a record whose key is 'SMITH, JOHN' could be found by the statement 'FIND SMI' provided that there are no other keys starting with 'SMI' proceeding SMITH, JOHN in the index. FIND will always find only the first record whose key is the same as < char string >. Even if the record pointer is moved down further in the file, a subsequent FIND on the same key will find the FIRST record.

If the index was created with a numeric key, then the found record will be the first record whose key is arithmetically equal to the object of the FIND.

**Note:** For indexes keyed on both characters and numbers, the FIND object is a character string with or without quote delimiters. Quote marks only become necessary for character strings if the original key had leading blanks. In that case, the exact number of leading blanks should be inside the quotes.

If a memory variable is desired as a FIND object, it must be placed after the FIND command by means of an &-macro replacement, e.g. FIND &NAME where NAME is a character string memory variable. Numeric memory variables must first be converted to a string by means of the STR function before they can be "macro-ized." See Section 5 for a discussion on macros.

Once a record in a database has been located by means of the FIND command, it can be processed just as any other database record. That is, it can be interrogated, altered, used in calculations, etc. dBASE commands that cause movement of the database (e.g. LIST, REPORT, COPY, etc.) will process the found record first and proceed to the next record in sequence, based upon the key.

If no record exists whose key is identical to the < char string > then the message: "NO FIND" will be displayed on the screen and the record number function "#" will give the value of zero.

If a second record with the same key is wanted, then a SKIP or a LOCATE FOR <exp> should be used. The SKIP will not know when there is no longer a match, the LOCATE (as long as the key was used in the expression) will be able to find additional matches.

SET EXACT ON will cause FIND to get a 'hit' only if there is a character for character match for the ENTIRE key (except for trailing blanks).

SET DELETE ON will cause FIND to NOT find any records that have been marked for deletion by the DELETE command.

**Examples:**

. USE SHOPLIST INDEX SHOPINDX

. LIST

00001	Beans	5	0.75
00007	Bleu cheese	1	1.96
00002	Bread loaves	2	1.06
00009	Charcoal	2	0.75
00006	Lettuce	2	0.53
00008	Milk	2	1.30
00004	Paper plates	1	0.94
00005	Plastic forks	5	0.42
00003	T-Bone steak	4	4.33

. FIND Bread

. DISPLAY

00002	Bread loaves	2	1.06
-------	--------------	---	------

. DISPLAY NEXT 3

00002	Bread loaves	2	1.06
00009	Charcoal	2	0.75
00006	Lettuce	2	0.53

. FIND P

. DISPLAY

00004	Paper plates	1	0.94
-------	--------------	---	------

. FIND Plas

. DISPLAY

00005	Plastic forks	5	0.42
-------	---------------	---	------

. FIND P

. DISPLAY

00004	Paper plates	1	0.94
-------	--------------	---	------

FIND will work in the file indexed with a multiple key, if the key includes all the spaces between the parts of the key.

**FIND . . . 78**

. LIST

00001	Flying High	Bird, I. M.	IMB001	02/29/04
00005	Nesting Procedures	Bird, I. M.	IMB002	09/25/06
00002	Diving	Fish, U. R.	URF001	12/30/23
00008	Nursing	Knight and Gale	KG001	08/04/44
00010	Vacationing in Europe	Knight and Gale	KG002	06/24/42
00004	101 Ways to Tie a Knot	Lynch, I.	IL001	04/01/00
00003	How to Survive a Crash	Lynch, M.	ML001	01/01/30
00007	Even Primes	Sladek, L	LS001	12/01/73
00009	Even More Primes	Sladek, L	LS002	04/24/73
00006	Thinking Big	Tim, Tiny	TT001	05/07/42

. FIND "Bird, I.M. IMB002"

. DISP

00005	Nesting Procedures	Bird, I. M.	IMB002	09/25/06
-------	--------------------	-------------	--------	----------

. FIND "Lynch, M."

. DISP

00003	How to Survive a Crash	Lynch, M.	ML001	01/01/30
-------	------------------------	-----------	-------	----------

. FIND "Sladek,L LS002"

. DISP

00009	Even More Primes	Sladek, L	LS002	04/24/73
-------	------------------	-----------	-------	----------

**GO or  
GOTO**

- a. GOTO RECORD < n >
- b. GOTO TOP
- c. GOTO BOTTOM
- d. < n >
- e. GOTO < memvar >

This command is used to reposition the record pointer of the database.

In either case a or d, the current-record pointer is set to record number < n >. Case d is a shorthand method for case a.

In cases b and c, the file in USE is rewound/unwound (TOP/BOTTOM) and the first/last record in the file is pointed to by the current-record pointer. When the file in USE has been INDEXed, then first/last record is not necessarily the first/last physical record in the database but rather is first/last according to the key used to index the database.

Case e can be used to position to a record number contained in a memory variable.

**Examples:**

```
. USE SHOPLIST

. GOTO RECORD 6
6

. DISPLAY
00006 LETTUCE                2                0.53

. GOTO TOP

. DISPLAY
00001 BEANS                   5                0.75

. GOTO BOTTOM

. DISPLAY
00009 CHARCOAL               2                0.75
```

**GOTO ... 80**

. LIST

00001	BEANS	5	0.75
00002	BREAD LOAVES	2	0.97
00003	T-BONE	4	3.94
00004	PAPER PLATES	1	0.86
00005	PLASTIC FORKS	5	0.42
00006	LETTUCE	2	0.53
00007	BLEU CHEESE	1	1.96
00008	MILK	2	1.30
00009	CHARCOAL	2	0.75

. STORE 4 TO RECORDNO

4

. GOTO RECORDNO

. DISP

00004	PAPER PLATES	1	0.86
-------	--------------	---	------





**HELP**

HELP <command verb>

The HELP command displays a brief synopsis of the dBASE commands and their syntax.

This command looks for the file DBASEMSG.TXT on the disk. DBASEMSG.TXT is a MS DOS type file that may be modified by a text editor to pass any information to the user.

**Example:**

```
. HELP CREATE
```

## IF ... 82

### IF

```
IF <exp>  
  <commands>  
{ELSE  
  <commands> }  
ENDIF
```

The IF command allows conditional execution of other commands usually in command files. When the <exp> evaluates to TRUE, the commands following the IF are executed. When the expression evaluates to FALSE, the commands following the ELSE are executed. If no ELSE is specified, all commands are skipped until an ENDIF is encountered. IF commands may be nested to any level.

**Note:** <commands> refers to whole command statements. The IF command begins with IF and ends with ENDIF. Statements must nest properly, an IF with a DO WHILE in the true (or false) path must not end before the ENDDO. See section 9.8 Rule 8 for more information.

### Examples:

```
IF STATUS='MARRIED'  
  DO MCOST  
ELSE  
  DO SCOST  
ENDIF
```

```
IF X=1  
  STORE CITY+STATE TO LOCATION  
ENDIF
```

**INDEX**

INDEX ON <expression> TO <index file name>  
INDEX

The INDEX command causes the current file in USE to be indexed on the <expression>. The <expression> is known as the "key." This means that a file will be constructed by dBASE (the <index file>) that contains pointers to the records in the USE file. The index file is made in such a way that the USE database appears to be sorted on the key for subsequent operations. The file in use is not physically changed. Sorting will be in an ascending order. A descending sort may be done on an expression that is a numeric. See below for an example.

Indexing allows very rapid location of database records by specifying all or part of the key by means of the FIND command. (See FIND). A database need not be indexed unless the application being worked would be enhanced by it. An indexed database can be used later with or without the indexing feature.

Usually, the INDEX command need only be done once for any given file. For instance, the APPEND command will automatically adjust the index file when new records are added.

If an indexed database is reUSED (in a later dBASE run or later in the same run that did the original INDEX operation), then a special form of the USE command must be used (i.e. USE <database filename> INDEX <index filename>).

Any number of index files may be constructed for any database, however, only the USED index files will be automatically updated by the APPEND, EDIT, REPLACE, READ or BROWSE commands.

An indexed file can be packed with the PACK command and the database, as well as the index file, will be properly adjusted. However if more than one index file is associated with the PACKed database, then that database must be REINDEXed on those keys.

**Warning:** The TRIM function must NOT be used as part of an index key. Also, if the \$ or STR functions are used as part or all of a key, they must have literal numbers (not variables or expressions) as their length parameters (e.g. INDEX ON \$(NAME,N,5)+STR(AMOUNT,5) TO NDXFILE instead of INDEX ON \$(NAME, N,N+5)+STR(AMOUNT, SIZEVAR) TO NDXFILE).

The command INDEX alone (with no parameters) will index the current record to the index files in use. This allows an index to be built which does not include all the records in a database. See also the REINDEX command.

# INDEX... 84

## Examples:

. USE SHOPLIST

. LIST

00001	Beans	5	0.75
00002	Bread loaves	2	1.06
00003	T-Bone steak	4	4.33
00004	Paper plates	1	0.94
00005	Plastic forks	5	0.42
00006	Lettuce	2	0.53
00007	Bleu cheese	1	1.96
00008	Milk	2	1.30
00009	Charcoal	2	0.75

. DISPLAY STRUCTURE

STRUCTURE FOR FILE: SHOPLIST.DBF

NUMBER OF RECORDS: 00009

DATE OF LAST UPDATE: 07/03/83

PRIMARY USE DATABASE

FLD	NAME	TYPE	WIDTH	DEC
001	ITEM	C	020	
002	NO	N	005	
003	COST	N	010	002
** TOTAL **			00036	

. NOTE CREATE INDEX FILE SHOPINDEX

. INDEX ON ITEM TO SHOPINDEX

. NOTE NOW LIST IN INDEX ORDER

. LIST

00001	Beans	5	0.75
00007	Bleu cheese	1	1.96
00002	Bread loaves	2	1.06
00009	Charcoal	2	0.75
00006	Lettuce	2	0.53
00008	Milk	2	1.30
00004	Paper plates	1	0.94
00005	Plastic forks	5	0.42
00003	T-Bone steak	4	4.33

. NOTE INDEXING ALLOWS FIND COMMAND

. FIND Milk

. DISPLAY

00008 Milk	2	1.30
------------	---	------

. FIND Be

. DISPLAY

00001 Beans	5	0.75
-------------	---	------

. SKIP

RECORD: 00007

. DISPLAY

00007 Bleu cheese	1	1.96
-------------------	---	------

. SKIP -1

RECORD: 00001

. DISPLAY

00001 Beans	5	0.75
-------------	---	------

. NOTE REGULAR USE COMMAND DOES NOT INCLUDE INDEX FILE

. USE SHOPLIST

. LIST

00001 Beans	5	0.75
00002 Bread loaves	2	1.06
00003 T-Bone steak	4	4.33
00004 Paper plates	1	0.94
00005 Plastic forks	5	0.42
00006 Lettuce	2	0.53
00007 Bleu cheese	1	1.96
00008 Milk	2	1.30
00009 Charcoal	2	0.75

. NOTE ALTERNATE FORM OF USE COMMAND RECALLS INDEX FILE

. USE SHOPLIST INDEX SHOPINDX

INDEX ... 86

. LIST

00001	Beans	5	0.75
00007	Bleu cheese	1	1.96
00002	Bread loaves	2	1.06
00009	Charcoal	2	0.75
00006	Lettuce	2	0.53
00008	Milk	2	1.30
00004	Paper plates	1	0.94
00005	Plastic forks	5	0.42
00003	T-Bone steak	4	4.33

. USE BOOKS

. DISP STRU

STRUCTURE FOR FILE: BOOKS.DBF

NUMBER OF RECORDS: 00010

DATE OF LAST UPDATE: 10/18/82

PRIMARY USE DATABASE

FLD	NAME	TYPE	WIDTH	DEC
001	TITLE	C	025	
002	AUTHOR	C	015	
003	CAT:NUM	C	006	
004	ARR:DTE	C	008	
** TOTAL **			00055	

. INDEX ON AUTHOR + CAT:NUM TO BOOKS

00010 RECORDS INDEXED

. LIST

00001	Flying High	Bird, I. M.	IMB001	02/29/04
00005	Nesting Procedures	Bird, I. M.	IMB002	09/25/06
00002	Diving	Fish, U. R.	URF001	12/30/23
00008	Nursing	Knight and Gale	KG001	08/04/44
00010	Vacationing in Europe	Knight and Gale	KG002	06/24/42
00004	101 Ways to Tie a Knot	Lynch, I.	IL001	04/01/00
00003	How to Survive a Crash	Lynch, M.	ML001	01/01/30
00007	Even Primes	Sladek, L	LS001	12/01/73
00009	Even More Primes	Sladek, L	LS002	04/24/73
00006	Thinking Big	Tim, Tiny	TT001	05/07/42

example of descending sort

. LIST

00001	MOW LAWN	5
00002	WALK DOG	9
00003	WALK CAT	1
00004	WATER GRASS	7
00005	PAINT TRIM	4
00006	INSTALL SPRINKLERS	6

. INDEX ON — PRIORITY TO SCHEDULE  
00006 RECORDS INDEXED

. LIST		
00002	WALK DOG	9
00004	WATER GRASS	7
00006	INSTALL SPRINKLERS	6
00001	MOW LAWN	5
00005	PAINT TRIM	4
00003	WALK CAT	1

## INPUT ... 88

### INPUT

INPUT ["<cstring>"] TO <memvar>

This construct permits the entry of expression values into memory variables, and can be used within command files as a means for the user to enter data at the command file's bidding. <memvar> is created, if necessary, and the expression is stored into <memvar>. If <cstring> is present, it is displayed on the screen as a prompt message before the input is accepted.

The type of the <memvar> is determined from the type of data that is entered. If a delimited character string is entered, the <memvar> will be of character type. If a numeric expression is entered, <memvar> will be of numeric type. If a T or Y (for True or Yes) is entered, <memvar> will be a logical variable with the value TRUE; if an F or N (for False or No) is entered, <memvar> will be a logical variable with the value FALSE. The function TYPE may be used to explicitly determine the type of the entry.

Either single or double quote marks may be used to delimit the prompt string, however, both the beginning and ending marks must be the same.

INPUT should be used to enter numeric and logical data only. The ACCEPT command is a more convenient way to enter character strings.

#### Examples:

```
. INPUT TO X
:3
3
```

```
. INPUT TO Z
:23/17.000+X
4.352
```

```
. INPUT 'PROMPT USER FOR INPUT' TO Q
PROMPT USER FOR INPUT: 12345
12345
```

```
. INPUT 'ENTER T IF EVERYTHING IS OKAY' TO LOG
ENTER T IF EVERYTHING IS OKAY: T
.T
```

```
. INPUT "ENTER A CHAR STRING" TO CHAR
ENTER A CHAR STRING: 'CHAR STRING MUST BE QUOTE DELIMITED'
CHAR STRING MUST BE QUOTE DELIMITED
```



. DISP MEMO

X	(N)	3	
Z	(N)	4.352	
Q	(N)	12345	
LOG	(L)	.T.	
CHAR	(C)	CHAR STRING MUST BE QUOTE DELIMITED	
** TOTAL **		05 VARIABLES USED	00054 BYTES USED

. INPUT 'ENTER ANY LOGICAL' TO LOG2

ENTER ANY LOGICAL :y

.T.

## INSERT... 90

### INSERT

INSERT [BEFORE] [BLANK]

This command allows records to be INSERTed into the middle of a database. Only one record at a time may be inserted into the database with the INSERT command.

The BEFORE phrase is used to cause insertion before the record currently pointed at, otherwise the new record will be placed just after the current record. Unless the BLANK phrase is used, the user will be prompted for input values as with the APPEND and CREATE commands. If the BLANK phrase is specified, then an empty record is inserted.

If the CARRY is SET ON then the information in the previous record is carried over to the new record.

If a SET FORMAT TO <file> is in effect, then INSERT will use the @-commands from the format file to form the full-screen and allow complete control of the screen and the data that will be appended. Otherwise, INSERT displays all fields in tabular form.

INSERTs into a large non-indexed database take a long time to complete and should be avoided unless necessary. INSERTs into an indexed file, no matter what size, are identical to APPENDs.

#### Examples:

. USE SHOPLIST

. LIST

00001	BEANS #303 CAN	5	0.69
00002	BREAD LOAVES	2	0.89
00003	T-BONE STEAK	4	3.59
00004	LETTUCE	2	0.49
00005	MILK (1/2 GAL)	2	1.19
00006	CHARCOAL, 5# BAGS	2	0.69

. GOTO RECORD 4

. INSERT

RECORD 00005

ITEM: BLEU CHEESE  
NO: 1  
COST: 1.79

```
. LIST
00001 BEANS #303 CAN          5          0.69
00002 BREAD LOAVES           2          0.89
00003 T-BONE STEAK           4          3.59
00004 LETTUCE                 2          0.49
00005 BLEU CHEESE             1          1.79
00006 MILK (1/2 GAL)         2          1.19
00007 CHARCOAL, 5# BAGS      2          0.69
```

. GOTO RECORD 4

. INSERT BEFORE

RECORD 00004

```
ITEM:    PAPER PLATES
NO:      1
COST:    .79
```

```
. LIST
00001 BEANS #303 CAN          5          0.69
00002 BREAD LOAVES           2          0.89
00003 T-BONE STEAK           4          3.59
00004 PAPER PLATES           1          0.79
00005 LETTUCE                 2          0.49
00006 BLEU CHEESE             1          1.79
00007 MILK (1/2 GAL)         2          1.19
00008 CHARCOAL, 5# BAGS      2          0.69
```

. 4

. DISPLAY

```
00004 PAPER PLATES           1          0.79
```

. INSERT BLANK

```
. LIST
00001 BEANS #303 CAN          5          0.69
00002 BREAD LOAVES           2          0.89
00003 T-BONE STEAK           4          3.59
00004 PAPER PLATES           1          0.79
00005
00006 LETTUCE                 2          0.49
00007 BLEU CHEESE             1          1.79
00008 MILK (1/2 GAL)         2          1.19
00009 CHARCOAL, 5# BAGS      2          0.69
```

**INSERT ... 92**

. 5

. REPLACE ITEM WITH 'PLASTIC FORKS' AND NO WITH 5 AND COST WITH .39

00001 REPLACEMENT(S)

. LIST

00001	BEANS #303 CAN	5	0.69
00002	BREAD LOAVES	2	0.89
00003	T-BONE STEAKS	4	3.59
00004	PAPER PLATES	1	0.79
00005	PLASTIC FORKS	5	0.39
00006	LETTUCE	2	0.49
00007	BLEU CHEESE	1	1.79
00008	MILK (1/2 GAL)	2	1.19
00009	CHARCOAL, 5# BAGS	2	0.69



**JOIN**

JOIN TO <file> FOR <expression> [FIELDS <field list>]

This is one of the most powerful commands in dBASE. It allows two databases to be JOINED together to form a third database whenever some criterion is met.

The two databases used are the primary and secondary USE files. First the SELECT PRIMARY command is issued. Then the JOIN command is issued. JOIN then positions dBASE to the first record of the primary USE file and evaluates the FOR expression for each record in the secondary USE file. Each time that the expression yields a TRUE result, a record is added to the new database. When the end of the secondary USE file is reached, the primary USE file is advanced one record, the secondary USE file is 'rewound' and the process continues until the primary USE file is exhausted.

If the FIELDS phrase is omitted then the output database will be comprised of all the fields in the primary USE file's structure and as many of the secondary USE file's fields as will fit before exceeding the 32 field limit of dBASE.

If the FIELDS phrase is supplied, then those fields, and only those fields, that are in the field list will be placed in the output database.

This command takes a lot of time to complete if the contributing databases are large. And if the joining criterion is too loose, causing many joinings per primary record, then there is the potential for causing a JOIN that dBASE cannot complete. For example, suppose that the primary and secondary USE files each contain 1000 records, and that the expression is always true, a million records should be output by the JOIN into a database whose size would exceed the dBASE maximum of 65,535 records.

**Example:**

```
. USE INVENTORY
```

```
. DISPLAY STRUCTURE
```

```
STRUCTURE FOR FILE: INVENTORY.DBF
```

```
NUMBER OF RECORDS: 00008
```

```
DATE OF LAST UPDATE: 12/14/82
```

```
PRIMARY USE DATABASE
```

FLD	NAME	TYPE	WIDTH	DEC
001	ITEM	C	020	
002	COST	N	010	002
003	PART:NO	C	005	
004	ON:HAND	N	005	
** TOTAL **			00041	

**JOIN ... 94**

```
. LIST
```

00001	TIME STITCH	9.99	24776	1
00002	WIDGET	1.67	31415	18
00003	GADGET, LARGE	16.33	92653	7
00004	TANK, SHERMAN	134999.00	89793	3
00005	SINK, KITCHEN	34.72	21828	77
00006	TROMBONES	198.37	76767	76
00007	RINGS, GOLDEN	200.00	70296	5
00008	#9 COAL	22.00	11528	16

. SELECT SECONDARY

. USE ORDERS

. DISPLAY STRUCTURE

STRUCTURE FOR FILE: ORDERS.DBF

NUMBER OF RECORDS: 00008

DATE OF LAST UPDATE: 12/21/82

PRIMARY USE DATABASE

FLD	NAME	TYPE	WIDTH	DEC
001	CUSTOMER	C	020	
002	PART:NO	C	005	
003	AMOUNT	N	005	
** TOTAL **			00031	

. LIST

00001	SWARTZ, JOE	31415	13
00002	SWARTZ, JOE	76767	13
00003	HARRIS, ARNOLD	11528	44
00004	ADAMS, JEAN	89793	12
00005	MACK, JAY	31415	3
00006	TERRY, HANS	76767	5
00007	JUAN, DON	21828	5
00008	SALT, CLARA	70296	9

. SELECT PRIMARY

JOIN TO ANNOTATE FOR PART:NO=S.PART:NO;  
FIELD CUSTOMER, ITEM, AMOUNT, COST

**use the inventory file to  
add names to the orders**

. USE ANNOTATE

. DISPLAY STRUCTURE

STRUCTURE FOR FILE: ANNOTATE.DBF

NUMBER OF RECORDS: 00008

DATE OF LAST UPDATE: 12/21/82

PRIMARY USE DATABASE

FLD	NAME	TYPE	WIDTH	DEC
001	CUSTOMER	C	020	
002	ITEM	C	020	
003	AMOUNT	N	005	
004	COST	N	010	002
** TOTAL **			00056	

. LIST

00001	SWARTZ, JOE	WIDGET	13	1.67
00002	MACK, JAY	WIDGET	3	1.67
00003	ADAMS, JEAN	TANK, SHERMAN	12	134999.00
00004	JUAN, DON	SINK, KITCHEN	5	34.72
00005	SWARTZ, JOE	TROMBONES	13	198.37
00006	TERRY, HANS	TROMBONES	5	198.37
00007	SALT, CLARA	RINGS, GOLDEN	9	200.00
00008	HARRIS, ARNOLD	#9 COAL	44	22.00

. USE INVENTORY

(join customer names with part numbers with insufficient inventory to satisfy orders so that the customers can be notified, for instance)

. JOIN TO BACKORDR FOR PART:NO=S.PART:NO.AND.ON:HAND< AMOUNT;  
FIELD CUSTOMER,ITEM

. USE BACKORDR

. LIST

00001	ADAMS, JEAN	TANK, SHERMAN
00002	SALT, CLARA	RINGS, GOLDEN
00003	HARRIS, ARNOLD	#9 COAL

**LIST**

- a. LIST [< scope> | [FOR < exp> ] |< exp list> ] [OFF]  
[WHILE < exp> ] [FIELDS < field list> ]
- b. LIST STRUCTURE
- c. LIST MEMORY
- d. LIST FILES [ON < disk drive> ] [LIKE < skeleton> ]
- e. LIST STATUS

LIST is the same as DISPLAY, except the scope defaults to ALL records and WAIT does not wait for a go-ahead after 15 record groups. Notice however that LIST STRUCTURE, LIST FILES and LIST MEMORY commands work exactly as the DISPLAY command.

With DELETED SET ON, LIST will show only non-deleted records.



**LOCATE**

LOCATE [<scope>] [FOR <exp>]  
[CONTINUE]

This command causes a search of database records in the USE file for the first record whose data fields allow the <exp> to be TRUE. When the expression is satisfied, the following message is displayed:

RECORD n

The CONTINUE command may be used to continue the search. Other dBASE commands may be issued between the LOCATE and the CONTINUE. This does, however, limit the number of the characters in the FOR <exp> to 128 instead of 254. See CONTINUE.

If the expression cannot be found, the message END OF FILE is displayed, and the database is left positioned at the last record in the file. If the NEXT clause (see scope, section 9.1) is used with this command and the expression cannot be found within the scope of the NEXT, the message END OF LOCATE is displayed, and the database is left positioned at the last record scanned.

**Note:** a LOCATE will work faster on a file that is USED without an INDEX file.

**Examples:**

. USE SHOPLIST

. LIST1

00001	BEANS #303 CAN	5	0.69
00002	BREAD LOAVES	2	0.89
00003	T-BONE STEAK	4	3.59
00004	PAPER PLATES	1	0.79
00005	PLASTIC FORKS	5	0.39
00006	LETTUCE	2	0.49
00007	BLEU CHEESE	1	1.79
00008	MILK (1/2 GAL)	2	1.19
00009	CHARCOAL, 5# BAGS	2	0.69

. LOCATE FOR COST>.70

RECORD: 00002

. CONTINUE

RECORD: 00003

**LOCATE . . . 98**

. DISP ITEM  
T-BONE STEAKS  
. CONTINUE  
RECORD: 00004

. CONTINUE  
RECORD: 00007

. CONTINUE  
RECORD: 00008

. CONTINUE  
END OF FILE



**LOCK ( ) UNLOCK**

Concurrent users may use the LOCK function and UNLOCK command to prevent records from simultaneous manipulation in more than one work partition.

The lock function locks onto the current record so that it may not be manipulated by commands originating from another work partition. No entry should be placed between ()s, since \* is assumed. LOCK may be activated with commands "?" and "IF." (Function returns a logical True if record was available and consequently locked by command; returns a False if record was already locked by another user.)

The UNLOCK command releases a record that has been LOCKed to use by other operators.

**Example:**

```
IF LOCK ( )
    <commands>
    UNLOCK
ELSE
    <alternative commands>
ENDIF
```

**(This page is intentionally left blank)**



**LOOP**

## LOOP

This command is used within the body of a DO WHILE to skip the commands following the LOOP, and still allow the reappraisal and possible reexecution of the body of the DO WHILE. LOOP is used to shorten DO WHILE loops which, if large, can be time consuming or may contain commands which are to be skipped at times. LOOP acts much as an ENDDO command, it will backup to the DO WHILE that matches it in nesting depth.

Use of loops in a DO WHILE is not always a good programming practice. The following example was done a second time without use of the LOOP capability.

**Example 1:**

```

STORE 1 TO INDEX
DO WHILE INDEX < 10
  STORE INDEX+1 TO INDEX
  IF ITEM=' '
    SKIP
    LOOP
  ENDIF
DO PROCESS
ENDDO

```

Anytime that ITEM is equal to blanks then skip to the next record and go back to the DO WHILE

**Example 2:**

```

STORE 1 TO INDEX
DO WHILE INDEX < 10
  STORE INDEX + 1 TO INDEX
  IF ITEM = ' '
    SKIP
  ELSE
    DO PROCESS
  ENDIF
ENDDO

```

## MODIFY ... 100

### MODIFY

- a. MODIFY STRUCTURE
- b. MODIFY COMMAND [<file>]

Form a. of this command allows the user to modify the structure of a dBASE file. Any changes are permitted. Fields can be added, deleted, or have their parameters (e.g. name, type, length, number of decimals) changed. (MODIFY STRUCTURE checks for syntactically correct field names.)

MODIFY acts upon the database currently in USE. The existing structure is displayed on the screen, changes are made directly on the screen in the same way as full-screen editing is done with two exceptions: CTL-N inserts a blank line wherever the cursor is, CTL-T deletes the line that the cursor is on. The other control keys behave as described in section 9.

**Note:** The MODIFY STRUCTURE command deletes ALL data records that were in the USE file prior to the MODIFY. In order to modify a structure and keep its data, first COPY the structure to a work file, USE the work file, make the modifications, and finally APPEND the old data to the work file. The original database and the work file may be RENAMED if it is necessary to restore their original names. See the example below.

Form b. of this command allows minor full-screen editing of command files (or anything else). If the <command file> is omitted, then the user is prompted for it. If the file doesn't exist, it is created. After a command file has been edited, MODIFY COMMAND will rename the file type of the old copy to.BAK and save the new copy with the type .CMD.

When in MODIFY COMMAND, the CTL-N and CTL-T editing functions work as described in a previous paragraph. CTL-Q will abort all changes to the command file, CTL-W will write the changes back to the disk and to the rename that was described above.

There are some significant restrictions to this form of the command: 1) lines can only be 77 or fewer characters long (including the carriage return/line feed pair); 2) TAB characters are converted to single spaces; 3) the cursor can only be backed up in a file about 4000 bytes; 4) there is no search or block move capability as are in some text editors.

Full-screen cursor controls are the same for MODIFY COMMAND *except* for the following commands:

- ctl-N inserts a blank line wherever the cursor is;
- ctl-T deletes the line the cursor is on and moves up the lower lines;
- ctl-W writes the changes made to the file back on the disk and exits MODIFY COMMAND;
- ctl-Q aborts any changes made to the command file;
- ctl-R scrolls one line down; and
- ctl-C scrolls one page up.

**Example:**

- . NOTE — AN EXAMPLE OF HOW TO MODIFY A STRUCTURE WITHOUT
- . NOTE — LOSING THE INFORMATION IN THE FILE
- . USE INVNTRY
- . COPY TO WORK
- . USE WORK
- . MODIFY STRUCTURE
- . APPEND FROM INVNTRY
- . DELETE FILE INVNTRY
- . USE
- . RENAME WORK TO INVNTRY

## NOTE... 102

### NOTE

- a. NOTE [<any characters>]
- b. \* [<any characters>]

This command allows comments to be placed into a command file. Unlike the **REMARK** command, the content of this command is not echoed onto the output device.

### Example:

NOTE — last modification : 4 july 1976

\* — last modification spelled doom's day



**PACK**

## PACK

This command purges all records marked for deletion by the DELETE command. Once the PACK command has been issued, nothing can bring back deleted records.

After a PACK, dBASE updates all INDEX files currently used with a database. This results in a much speedier PACK and index update process.

An alternate method to the PACK is to COPY the old file to a new file. DELETED records will not be copied. Then the old file may be deleted (or saved as a back-up) and the new file renamed.

PACK will not reduce amount of disk space reserved for that file by MS DOS. To recover the space, use a COPY TO <file name> and then delete the source file. This is a limitation of the MS DOS operating system not of dBASE II.

**Examples:**

```
. USE B:SHOPSAVE
```

```
. LIST
```

00001	BEANS	5	0.75
00002	BREAD LOAVES	2	0.97
00003	T-BONE	4	3.94
00004	PAPER PLATES	1	0.86
00005	PLASTIC FORKS	5	0.42
00006	LETTUCE	2	0.53
00007	BLEU CHEESE	1	1.96
00008	MILK	2	1.30
00009	CHARCOAL	2	0.75

```
. DELETE RECORD 8
```

```
00001 DELETION(S)
```

```
. LIST
```

00001	BEANS	5	0.75
00002	BREAD LOAVES	2	0.97
00003	T-BONE	4	3.94
00004	PAPER PLATES	1	0.86
00005	PLASTIC FORKS	5	0.42
00006	LETTUCE	2	0.53
00007	BLEU CHEESE	1	1.96
00008	*MILK	2	1.30
00009	CHARCOAL	2	0.75

## PACK . . . 104

```
. PACK  
PACK COMPLETE, 00008 RECORDS COPIED
```

```
. LIST  
00001 BEANS 5 0.75  
00002 BREAD LOAVES 2 0.97  
00003 T-BONE 4 3.94  
00004 PAPER PLATES 1 0.86  
00005 PLASTIC FORKS 5 0.42  
00006 LETTUCE 2 0.53  
00007 BLEU CHEESE 1 1.96  
00008 CHARCOAL 2 0.75
```

A PACK need not always be done, for example, suppose some records must be deleted but it is necessary for them to remain in the database. These records will not be COPY'd, APPENDED, or SORTed, but they may be COUNTed. It becomes important to know whether or not the record being processed is deleted or not. The following example is a partial command file that would skip over a record that has been deleted and continue processing with the next record.

```
DO WHILE .NOT. EOF  
  LOCATE FOR NATURE = "TLM"  
  IF .NOT. *
```

```
    commands
```

```
  ENDIF  
  CONTINUE  
ENDDO
```

This operation might be avoided entirely by using the SET DELETED ON command.

**QUIT**

QUIT

This command closes all database files, command files, and alternate files and returns control to the operating system. The message **\*\*\* END RUN dBASE \*\*\*** is displayed.

## READ

READ [NOUPDATE]

This command enters the full-screen mode for editing and/or data entry of variables identified for and displayed by an "@" command with a GET phrase. The cursor can be moved to any of the GET variables. Changes made to those variables on the screen are entered into the appropriate database fields or memory variables.

If the SET FORMAT TO <format file> command has been issued, then READ will cause all of the "@" commands in the format file to be executed, thus formatting the screen, allowing editing of all GET variables. Notice that this technique is a tailorable substitute for the EDIT command when in the interactive mode.

When in the SET FORMAT TO SCREEN mode, an ERASE command is used to clear the screen. A series of "@" commands may then be issued to format the screen. Then a READ command would be given which would allow editing.

If a second or later series of "@" commands is issued after a READ command, then READ will place the cursor on the first GET variable following the last READ. In this way, the screen format and the specific variables edited can be based on decisions made by the user in response to prior READ commands.

Variables to be used with the "@" commands and edited using the READ command must be either in the USE file as field names or must be character string memory variables. Memory variables must be predefined before the "@" command is issued. If necessary, store as many blanks as you want the maximum length of the memory variable to be in order to initialize the memory variable (e.g. STORE ' ' to MEMVAR).

See section 8 for cursor control and data entry instructions.

The SET SCREEN ON command must be in effect (this is the default condition if full-screen operations were enabled when dBASE II was installed).

The NOUPDATE option can be used to avoid index updating. dBASE checks for altered index keys after READ commands and updates index files as needed. If multiple indexes are used, this process can take time. If it is known that no keys are altered by the READ command (e.g. only memory variables or non-key fields are READ) then the index update process can be skipped by use of the NOUPDATE phrase.

**Example:**

```

.
.
STORE ' ' TO PTYPE
STORE ' ' TO ACCT
ERASE
@ 5.0 SAY 'Enter a C for cash payment'
@ 6.0 SAY ' ' or a D for deferred payment'
@ 8.10 GET PTYPE
READ
IF PTYPE='D'
    @ 10.10 SAY 'Enter acct no.' GET ACCT PICTURE '999-99-9999'
    READ
ENDIF
.
.
.

```

In this command file fragment, the screen is cleared and the first two "@" commands are put up. The cursor will be between two colons that mark the screen location of the variable PTYPE. Since the first STORE set the size of PTYPE at 1 character, any entry by the user will fill PTYPE and exit the first READ command.

If a "D" was entered by the dBASE operator, then the "@" command that asks for an account number will be done. Notice that ACCT was defined long enough in the STORE to include the two dashes that the PICTURE phrase in the "@" will enter.

```

USE CHECKS
SET FORMAT TO SCREEN
ACCEPT "Option" TO CHOICE
IF CHOICES'Aa'
    ERASE
    DO WHILE NUMBER # 0
        APPEND BLANK
        @ 5.0 SAY "Enter next Number";
            GET NUMBER PICTURE '99999'
        @ 6.0 SAY "Enter Recipient";
            GET RECIPIENT PICTURE 'XXXXXXXXXXXXXXXXXXXXXXXXXXXX'
        @ 7.0 SAY "Enter Amount";
            GET AMOUNT PICTURE '9999999999'
        @ 8.5 SAY "Is it back yet?";
            GET HOME
        @ 8.30 SAY "Are you paying out?";
            GET OUTGOING
    READ
    ENDDO
ENDIF

```

In the last example, a file was used and altered directly, the choice being left up to the operator on whether or not to add new records to the database in question.

Refer to the "@" command for more details.

**RECALL ... 108**

**RECALL**

RECALL [< scope> ] [FOR < exp> ]  
                  [WHILE < exp> ]

This command removes the mark-for-deletion from the records that were marked by the DELETE command.

**Examples:**

. USE DUPE3

. LIST

00001	NEUMAN, ALFRED E.	1357
00002	RODGERS, ROY	2468
00003	CASSIDY, BUTCH	3344
00004	CHANG, LEE	6743
00005	POST, WILEY	1011
00006	LANCASTER, WILLIAM J	6623

. 3

. DELETE NEXT 3  
00003 DELETION(S)

. LIST

00001	NEUMAN, ALFRED E.	1357
00002	RODGERS, ROY	2468
00003	*CASSIDY, BUTCH	3344
00004	*CHANG, LEE	6743
00005	*POST, WILEY	1011
00006	LANCASTER, WILLIAM J	6623

. RECALL RECORD 4  
00001 RECALL(S)

. LIST

00001	NEUMAN, ALFRED E.	1357
00002	RODGERS, ROY	2468
00003	*CASSIDY, BUTCH	3344
00004	CHANG, LEE	6743
00005	*POST, WILEY	1011
00006	LANCASTER, WILLIAM J	6623

. RECALL ALL  
00002 RECALL(S)

. LIST

00001	NEUMAN, ALFRED E.	1357
00002	RODGERS, ROY	2468
00003	CASSIDY, BUTCH	3344
00004	CHANG, LEE	6743
00005	POST, WILEY	1011
00006	LANCASTER, WILLIAM J	6623

## REINDEX... 110

### REINDEX

#### REINDEX

This command behaves exactly as the INDEX on <exp> to <index file name> command EXCEPT that the key expression and TO file need not be entered. REINDEX uses the keys from the index files currently in use and completely rebuilds the files. Index files must have been specified by a USE command or a SET INDEX TO command.

. USE BOOKS INDEX AUTHORS

. REINDEX



**RELEASE**

RELEASE [<memvar list>]  
    [ALL]  
    [ALL LIKE <skeleton>]  
    [ALL EXCEPT <skeleton>]

This command releases all, subsets of all, or selected memory variables and makes the space that they consumed available for new memory variables. If ALL is specified, then all memory variables will be deleted.

The <skeleton> is similar to <skeleton> in DISPLAY FILES. Question marks mask any single character, asterisks match up to ten subsequent characters.

E.g. RELEASE ALL LIKE MEM??X would release memory variables MEMABX, MEMFFX, and MEMRRX but not MEMAB, MEMABY, or MAMABX.

RELEASE ALL EXCEPT MEM\* would release MAM, MAMABX, and ME but would not release MEM, MEMXXXXX, or MEMABX.

## REMARK...112

### REMARK

REMARK [any characters]

This command allows the display of any characters. The contents of this command are displayed on the output device when this command is encountered.

#### Examples:

```
.REMARK ***** REMARK TEST *****  
***** REMARK TEST *****
```

**RENAME**

RENAME < original file name> TO < new file name>

This command allows the changing of the name of a file in a MS-DOS directory. If no file type (the 3 characters following a file name) is given then dBASE assumes that a database's name is being used and assigns the type .DBF to the named files. See section 4 for more detail concerning dBASE use of file types.

**Example:**

```
. RENAME INVENMAC TO INVENOLD  
. RENAME D:REPORT.FRM TO REPORT.BAK  
. RENAME TYPELESS. TO TYPED.TYP
```

## REPLACE . . . 114

### REPLACE

```
REPLACE [<scope>] <field> WITH <exp> [,<field> WITH <exp>]  
      [FOR <exp>] [NOUPDATE]  
      [WHILE <exp>]
```

This command is used to replace the contents of specified data fields of the file in USE with some new data. This command is contrasted with the STORE command in that REPLACE changes only field variables, while the STORE command changes only memory variables.

If <scope> is not supplied in the command then REPLACE acts only on the current record.

If a REPLACE is done on an index key and the index is in USE, then the index file will be adjusted by deleting the old index entry and re-entering the new entry in its proper place. Un-USED index files will not be affected. When a REPLACE is done on an index key, the altered record will "shift places" in the file, the new "next record" will not be the same as the old "next record." The key should not be REPLACEd with a NEXT n as the <scope>.

The NOUPDATE option can be used to avoid index updating. dBASE checks for altered index keys after REPLACE commands and updates index files as needed. If multiple indexes are used, this process can take time. If it is known that no keys are altered by the REPLACE command (e.g. only memory variables or non-key fields are REPLACEd) then the index update process can be skipped by use of the NOUPDATE phrase.

**Note:** If you are using PRIMARY and SECONDARY databases, you can REPLACE a field *only* in the currently SELECTED database.

#### Examples:

```
. USE SHOPLIST
```

```
. NOTE INFLATION CAUSES 10% PRICE INCREASE
```

```
. LIST
```

00001	BEANS #303 CAN	5	0.69
00002	BREAD LOAVES	2	0.89
00003	T-BONE STEAKS	4	3.59
00004	PAPER PLATES	1	0.79
00005	PLASTIC FORKS	5	0.39
00006	LETTUCE	2	0.49
00007	BLEU CHEESE	1	1.79
00008	MILK (1/2 GAL)	2	1.19
00009	CHARCOAL, 5# BAGS	2	0.69

```
. REPLACE ALL COST WITH COST *1.1
```

```
00009 REPLACEMENT(S)
```

. LIST

00001	BEANS #303 CAN	5	0.75
00002	BREAD LOAVES	2	0.97
00003	T-BONE STEAKS	4	3.94
00004	PAPER PLATES	1	0.86
00005	PLASTIC FORKS	5	0.42
00006	LETTUCE	2	0.53
00007	BLEU CHEESE	1	1.96
00008	MILK (1/2 GAL)	2	1.30
00009	CHARCOAL, 5# BAGS	2	0.75

. USE B:SHOPLIST

. COPY TO B:SHOPWORK

00009 RECORDS COPIED

. LIST

00001	BEANS	5	0.75
00002	BREAD LOAVES	2	0.97
00003	T-BONE	4	3.94
00004	PAPER PLATES	1	0.86
00005	PLASTIC FORKS	5	0.42
00006	LETTUCE	2	0.53
00007	BLEU CHEESE	1	1.96
00008	MILK	2	1.30
00009	CHARCOAL	2	0.75

. GOTO TOP

. REPLACE NEXT 5 COST WITH COST \*1.1 FOR COST &gt; .75

00003 REPLACEMENT(S)

. LIST

00001	BEANS	5	0.75
00002	BREAD LOAVES	2	1.06
00003	T-BONE	4	4.33
00004	PAPER PLATES	1	0.94
00005	PLASTIC FORKS	5	0.42
00006	LETTUCE	2	0.53
00007	BLEU CHEESE	1	1.96
00008	MILK	2	1.30
00009	CHARCOAL	2	0.75

. USE CHECKS

. DISP STRU

REPLACE... 116

STRUCTURE FOR FILE CHECKS.DBF

NUMBER OF RECORDS: 00016

DATE OF LAST UPDATE: 10/18/82

PRIMARY USE DATABASE

FLD	NAME	TYPE	WIDTH	DEC
001	NUMBER	N	005	
002	RECIPIENT	C	020	
003	AMOUNT	N	010	002
004	HOME	L	001	
005	OUTGOING	L	001	
** TOTAL **			00038	

. LIST

00001	1 Phone Company	104.89 F. T.
00002	2 Gas Company	4.15 F. T.
00003	3 Electricity	250.30 F. T.
00004	4 Grocery Store	1034.45 F. T.
00005	134 Me, salary	561.77 T. F.
00006	6 Bank (sc)	4.00 T. T.
00007	7 Doctor Doolittle	100.00 T. T.
00008	8 Pirates	100.00 F. T.
00009	9 Car Repair Man	500.01 F. T.
00010	10 Me	561.77 T. F.
00011	11 Tupperware	50.02 T. T.
00012	12 Me	561.77 T. F.
00013	13 Me	750.03 T. F.
00014	234 Peter Rabbit	14.00 F. T.
00015	237 Golden Goose	650.00 F. T.
00016	30 Me	561.77 T. F.

. 11

. REPLACE HOME WITH F

00001 REPLACEMENT(S)

. DISPLAY

00011	11 Tupperware	50.02 F. T.
-------	---------------	-------------

**REPORT**

```
REPORT [FORM <form file>] [<scope>] [TO PRINT] [PLAIN] [FOR <exp>]
                                           [WHILE <exp>]
```

**REPORT** is used to prepare reports (either on the screen or on paper) by displaying data from the file in **USE** in a defined manner. Reports may have titled columns, totaled numeric fields, and displayed expressions involving data fields, memory variables, and constants.

The **FOR** phrase allows only that information which meets the conditions of the **<exp>** to be reported; the **TO PRINT** phrase sends the report to the printer as well as the screen; and the **<scope>** of the report defaults to **ALL** unless otherwise specified.

The first time the **REPORT** command is used (for a new report) a **FORM** file is built. **dBASE** prompts the user for specifications of the report format and automatically generates the **FORM** file. Subsequent reports can use the **FORM** file to avoid respecification of the report format. **REPORT** strips high-order bits in form files. If the **FORM** phrase of the command is omitted the user will be prompted for the name of the form file.

The following example of a form file has almost all the options specified. The user may control the number of spaces to indent the lines in the body of the report with the '**M**' option (default is 8 spaces); the number of lines per page is changed with the '**L**' option (default is 57 lines); and the location of the page heading is controlled with the '**W**' option (the page width, default is 80 characters) since it is only used for centering the page heading.

The **REPORT** command will process a maximum of 24 fields.

```
. REPORT FORM SHOPFORM
ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE, W=PAGE WIDTH M=5,W=65
PAGE HEADING? (Y/N) Y
ENTER PAGE HEADING: Shopping List for Picnic
DOUBLE SPACE REPORT? (Y/N) N
ARE TOTALS REQUIRED? (Y/N) Y
SUBTOTALS IN REPORT? (Y/N) N
COL      WIDTH,CONTENTS
001      23,ITEM+'...'
ENTER HEADING: Item;=====
002      10,NO
ENTER HEADING: > Number;=====
ARE TOTALS REQUIRED? (Y/N) Y
003      10,COST
ENTER HEADING: > Cost/Item;=====
ARE TOTALS REQUIRED? (Y/N) N
004      10,NO*COST
ENTER HEADING: > Cost;=====
ARE TOTALS REQUIRED? (Y/N) Y
005      (cr)
```

## REPORT ... 118

REPORT asks for the width of the field to be printed and the contents of the field. The width asked for here has no relationship to the actual width of the field to be printed out, for instance, in the first column above, ITEM is in a column that is 23 characters wide, in the data base ITEM is actually only 20 characters wide. One should also note that the string '...' is being concatenated to the contents of the field ITEM. This accounts for the extra 3 characters in the report. This also means that if the report column is less in length than the field that should go into it, dBASE will wrap the field to fit. An 80 character field would generate 2 lines if it were put into a 50 character column.

The contents of the columns may be fields from a database, a memory variable, literals, or expressions. Note that in column 1 in the form on the previous page, there is a concatenated string. Each record reported from the database in use will have three periods concatenated to the end of the string (*the database will remain unchanged*). Column 4 contains the product of NO and COST. Column 4 has no field equivalent to it in the database. (The fields are, left to right, named ITEM, NO, and COST).

. LIST			
00001	BEANS	5	0.75
00002	BREAD LOAVES	2	1.06
00003	T-BONE	4	4.33
00004	PAPER PLATES	1	0.94
00005	PLASTIC FORKS	5	0.42
00006	LETTUCE	2	0.53
00007	BLEU CHEESE	1	1.96
00008	MILK	2	1.30
00009	CHARCOAL	2	0.75

Returning to the FORM file (the questions on what should go into the report), note that there are some special characters used in the headings. For page headings, column headings, and character strings, a semicolon (;) will break the heading or string at the semicolon and resume the display on the next line. If a heading or string is too long to fit within the number of spaces allowed for it, it will be broken at the last blank (if possible) and resumed on the next line. The other significant characters are "<," and ">." In column headings, if the title is preceded with a "<" then the title will be left-justified in the column. Likewise a ">" will right-justify the title.

Other options in REPORT include totaling, subtotaling, and summary reports. In summary reports, detail records are not displayed, just totals and subtotals. Totaling and subtotaling are done only on fields that are numeric in nature. See the report examples.

Finally, pressing <enter> will end the report form and begin displaying the report. A copy will be printed on the printer if the TO PRINT phrase was included in the initial command.

Other dBASE commands that affect the operation of report are the "SET EJECT OFF," "SET HEADING TO" and "SET DATE TO" commands. Before REPORT prints out its information, it does a page eject. This capability may be suppressed with the SET EJECT OFF command. The SET HEADING TO command allows an additional heading to be added



to the report at run time. This command has an effect for the duration of one session. (The heading must be set each time a new dBASE run is initiated.) The same is for the SET DATE TO command. The date of the report may be changed or omitted by use of this command. See the SET command for more information.

There comes a time, when this capability is no longer adequate, special forms must be used, more flexibility is desired with the report format, retrieving the data from the database requires more complex methods than REPORT will handle, etc. The "@" and the SET FORMAT TO PRINT commands will give the user more power over the form of the report. See the "@" command for more information and examples.

**REPORT ... 120**

**Example 1:**

. USE SHOPLIST

. REPORT FORM SHOPFORM

PAGE NO. 00001

**Shopping List for Picnic**

Item	Number	Cost/Item	Cost
BEANS	5	0.75	3.75
BREAD LOAVES	2	1.06	2.12
T-BONE	4	4.33	17.32
PAPER PLATES	1	0.94	0.94
PLASTIC FORKS	5	0.42	2.10
LETTUCE	2	0.53	1.06
BLEU CHEESE	1	1.96	1.96
MILK	2	1.30	2.60
CHARCOAL	2	0.75	1.50
** TOTAL **	24		33.35

. SET HEADING TO 4 July 1976

. REPORT FORM SHOPFORM

PAGE NO. 00001

4 July 1976

**Shopping List for Picnic**

Item	Number	Cost/Item	Cost
BEANS	5	0.75	3.75
BREAD LOAVES	2	1.06	2.12
T-BONE	4	4.33	17.32
PAPER PLATES	1	0.94	0.94
PLASTIC FORKS	5	0.42	2.10
LETTUCE	2	0.53	1.06
BLEU CHEESE	1	1.96	1.96
MILK	2	1.30	2.60
CHARCOAL	2	0.75	1.50
** TOTAL **	24		33.35

**Example 2:**

This example shows use of the subtotalling capabilities of dBASE. When the report form is created the subtotalling is done on the field PART:NO. This could be done if it was necessary to know not only who the part was ordered by but also how many of each part must be made (or bought).

. USE ORDERS INDEX ORDERS

. LIST

00003	HARRIS, ARNOLD	11528	44
00013	ANDERSON, JAMES REGI	11528	16
00007	JUAN, DON	21828	5
00001	SWARTZ, JOE	31415	13
00005	MACK, JAY	31415	3
00009	BARNETT, WALT	31415	6
00008	SALT, CLARA	70296	9
00002	SWARTZ, JOE	76767	13
00006	TERRY, HANS	76767	5
00010	NICHOLS, BILL	76767	17
00004	ADAMS, JEAN	89793	12
00011	MURRAY, CAROL	89793	4
00012	WARD, CHARLES A.	92653	15

. REPORT

ENTER REPORT FORM NAME: ORDERS  
 ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE, W=PAGE WIDTH W=65  
 PAGE HEADING? (Y/N) Y  
 ENTER PAGE HEADING: ORDERS LISTED BY PART NUMBER  
 DOUBLE SPACE REPORT? (Y/N) N  
 ARE TOTALS REQUIRED? (Y/N) Y  
 SUBTOTALS IN REPORT? (Y/N) Y  
 ENTER SUBTOTALS FIELD: PART:NO  
 SUMMARY REPORT ONLY? (Y/N) N  
 EJECT PAGE AFTER SUBTOTALS? (Y/N) N  
 ENTER SUBTOTAL HEADING: Orders for part number  
 COL            WIDTH,CONTENTS  
 001            20,CUSTOMER  
 ENTER HEADING: < CUSTOMER NAME  
 002            10,AMOUNT  
 ENTER HEADING: > QUANTITY ORDERED  
 ARE TOTALS REQUIRED? (Y/N) Y  
 003

PAGE NO. 00001

ORDERS LISTED BY PART NUMBER

CUSTOMER NAME	QUANTITY ORDERED
Orders for part number 11528	
HARRIS, ARNOLD	44
ANDERSON, JAMES REGI	16
** SUBTOTAL **	60
*Orders for part number 21828	
JUAN, DON	5
**SUBTOTAL **	5
*Orders for part number 31415	
SWARTZ, JOE	13
MACK, JAY	3
BARNETT, WALT	6
** SUBTOTAL **	22
*Orders for part number 70296	
SALT, CLARA	9
** SUBTOTAL **	9
*Orders for part number 76767	
SWARTZ, JOE	13
TERRY, HANS	5
NICHOLS, BILL	17
** SUBTOTAL **	35
*Orders for part number 89793	
ADAMS, JEAN	12
MURRAY, CAROL	4
** SUBTOTAL **	16

```

*Orders for part number 92653
WARD, CHARLES A.                15
** SUBTOTAL **                  15

** TOTAL **                      162

```

**Example 3:**

Suppose some of your colleagues and yourself started playing cards for points to see who would buy lunch for everyone on the next holiday. In the interest of Fair Play, you decide to keep a running total on the score. All sorts of information could be dug out of the database (like who could lose his shirt if he weren't careful). The following database could be an example of such a game.

```
. DISP STRU
```

```

STRUCTURE FOR FILE: CARDS.DBF
NUMBER OF RECORDS: 00016
DATE OF LAST UPDATE: 09/17/82
PRIMARY USE DATABASE

```

FLD	NAME	TYPE	WIDTH	DEC
001	DATE	C	008	
002	LISA	N	003	
003	ANNA	N	003	
004	WAYNE	N	003	
** TOTAL **			00018	

## REPORT . . . 124

. REPORT

ENTER REPORT FORM NAME: CARDS

ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE, W=PAGE WIDTH W=40

PAGE HEADING? (Y/N) Y

ENTER PAGE HEADING: Hearts Scores

DOUBLE SPACE REPORT? (Y/N) N

ARE TOTALS REQUIRED? (Y/N) Y

SUBTOTALS IN REPORT? (Y/N) N

COL WIDTH,CONTENTS

001 10,DATE

ENTER HEADING: Date of;Game

002 6,Lisa

ENTER HEADING: Score;Lisa

ARE TOTALS REQUIRED? (Y/N) Y

003 6,ANNA

ENTER HEADING: Score;Anna

ARE TOTAL REQUIRED? (Y/N) Y

004 6,WAYNE

ENTER HEADING: Score;Wayne

ARE TOTALS REQUIRED? (Y/N) Y

005 5,LISA+ANNA+WAYNE

ENTER HEADING: Game;Total

ARE TOTALS REQUIRED? (Y/N) Y

006 (cr)

(Note — the last column in the report form is a totaling of the scores in each of the records, that is, the sum of Lisa's, Wayne's and Anna's scores. It is not necessary for the column in the report to exist in the database before it may be used, the field "LISA+ANNA+WAYNE" does not exist in the database "CARDS." This would be an example of how an expression may be placed in a report.)

PAGE NO. 00001

Hearts Scores

Date of Game	Score Lisa	Score Anna	Score Wayne	Game Total
05/26/82	29	75	53	157
05/27/82	45	48	63	156
05/28/82	50	56	74	180
05/29/82	86	24	72	182
06/05/82	43	12	75	130
06/12/82	42	9	27	78
06/26/82	84	35	63	182
07/06/82	33	71	26	130
08/19/82	37	55	38	130
09/15/82	19	57	54	130
09/16/82	15	7	108	130
09/17/82	59	13	58	130
** TOTAL **	542	462	711	1715

A report may also cover just a few of the records in a file. Such as:

. GOTO RECORD 7

. REPORT NEXT 4 FORM CARDS

PAGE NO. 00001

Hearts Scores

Date of Game	Score Lisa	Score Anna	Score Wayne	Game Total
07/07/82	40	63	27	130
07/09/82	55	41	60	156
07/13/82	40	63	54	157
07/23/82	38	69	23	130
** TOTAL **	173	236	164	573

A report may also ask for information which would meet certain criteria. Such as:

. REPORT FORM CARDS FOR WAYNE < 50

**REPORT . . . 126**

PAGE NO. 00001

Date of Game	Hearts Scores				Game Total
	Score Lisa	Score Anna	Score Wayne		
06/12/82	42	9	27	78	
07/06/82	33	71	26	130	
07/07/82	40	63	27	130	
07/23/82	38	69	23	130	
08/19/82	37	55	38	130	
** TOTAL **	190	267	141	598	

REPORT FORM ORDERS WHILE CUSTOMER > = "M"

PAGE NO. 00001  
12/13/82

CUSTOMER	PART	AMOUNT
MACK, JAY	31415	3
MURRAY, CAROL	89793	4
NICHOLS, BILL	76767	17
SALT, CLARA	70296	9
SWARTZ, JOE	31415	13
SWARTZ, JOE	76767	13
TERRY, HANS	76767	5
WARD, CHARLES A.	92653	15

**PLAIN** is an extension of the command **REPORT**. This allows for a **dBASE** report to be created in such a manner that it may be inserted into a report generated by a wordprocessor.

The clause **PLAIN** causes page numbers and the date at the top of each page in the report to be suppressed. Page headings are inserted into the **dBASE** report only at the beginning of the report. If it is desired to suppress the page ejects between reports then the **SET EJECT OFF** must still be used.

**Examples:**

- . USE TRACE INDEX DOC
- . NOTE POSITION THE DATABASE AT THE FIRST RECORD FOR THE REPORT
- . 304



. REPORT FORM TABLES PLAIN WHILE DOC = "3-280-T"  
 ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE, W=PAGE WIDTH  
 PAGE HEADING? (Y/N) Y  
 ENTER PAGE HEADING: TABLES  
 DOUBLE SPACE REPORT? (Y/N) N  
 ARE TOTALS REQUIRED? (Y/N) N  
 COL WIDTH,CONTENTS  
 001 20,\$(DOC,7,17)  
 ENTER HEADING: TABLE  
 002 40,DESCR  
 ENTER HEADING: REQUIREMENT  
 003 (cr)

TABLE

TABLE	REQUIREMENT
Table 1	GLL Telemetry Modes
Table 2	Allowable combinations of R/T and Record Formats
Table 2.3.2	Bus User Codes
Table 3	GLL Bit rate allocation
Table 4	Header Format
Table 5	Format Identification
Table 6	Commutation Map Identifier Assignment
Table 7	S/C Clock Progression
Table A2.2.1	Eng data layout
Table A2.2.2	Fixed-Area Structure/Position Identifiers
Table A2.2.3	Variable Area Pocket Structure/Position Identifier
Table A2.2.4	CDS Fixed area Measurement Sampling Time
Table A2.2.8	Engr Measurements

## RESET ... 128

### RESET

RESET [<drive>]

The RESET command is used to reset the MS-DOS bit map after a diskette has been swapped. Normally, if a diskette is swapped, MS-DOS will not allow writes to take place until after a warm or soft boot has taken place.

If <drive> is not specified, then the entire system will be reset. Unfortunately, neither dBASE nor the operating system can determine which diskettes you may have swapped and I/O errors may result if a disk that has open files is replaced. If <drive> is specified, dBASE checks to see whether any of its files are open on that drive, checks empty drives, and will prevent I/O errors. The <drive>-less form should therefore not be used and is maintained for compatibility reasons only. Do not swap and RESET the drive which contains the dBASE system command files.

Issuing a RESET command when no disk swap has taken place has no effect.

**RESTORE**

RESTORE FROM <file> [ADDITIVE]

This command reads a file of memory variables. The file must be built using the SAVE TO <file> command. All memory variables which were defined previous to the RESTORE command are deleted by this command.

If the ADDITIVE phrase is included, then memory variables already defined are not released, and as many variables as will fit are added from the FROM file.

**Examples:**

```
. DISPLAY MEMORY
ONE      (N)      1.0000
ALFABET  (C)      ABCDEFGHIJKL
CHARS    (C)      ABCDEFGHIJKL NEW STUFF
** TOTAL **      03 VARIABLES USED      00042 BYTES USED
```

```
. SAVE TO MEMFILE
```

```
. RELEASE ALL
```

```
. DISPLAY MEMORY
** TOTAL **      00 VARIABLES USED      00000 BYTES USED
```

```
. RESTORE FROM MEMFILE
```

```
. DISPLAY MEMORY
ONE      (N)      1.0000
ALFABET  (C)      ABCDEFGHIJKL
CHARS    (C)      ABCDEFGHIJKL NEW STUFF
** TOTAL **      03 VARIABLES USED      00042 BYTES USED
```

## **RETURN ... 130**

### **RETURN**

#### RETURN

This command is used inside a command file to return control to the command file which called it (or to the keyboard if the user called the command file directly). Encountering an end of file on a command file is equivalent to a RETURN command.

Command files usually have a RETURN command as their last executable line, and may be used to protect lines of text or programmer notes from processing.

**SAVE**

SAVE TO <file> [ALL LIKE <skeleton> to <memory filename>]

This command stores all currently defined memory variables to a file. These memory variables may be restored by the RESTORE command.

The <skeleton> is similar to <skeleton> in DISPLAY FILES. Question marks mask any character, asterisks match any subsequent characters.

E.g. SAVE TO <file> ALL LIKE MEM??X would save memory variables MEMABX, MEMFFX, and MEMRRX but not MEMAB, MEMABY, or MAMABX.

The SAVE ALL LIKE <skeleton> to <memory filename> saves all variables whose names are selected by the LIKE <skeleton>.

**Examples:**

```
. DISPLAY MEMORY
ONE      (N)      1.0000
ALFABET  (C)      ABCDEFGHIJKL
CHARS    (C)      ABCDEFGHIJKL NEW STUFF
** TOTAL **      03 VARIABLES USED      00042 BYTES USED
```

```
. SAVE TO MEMFILE
```

```
. RELEASE ALL
```

```
. DISPLAY MEMORY
** TOTAL **      00 VARIABLES USED      00000 BYTES USED
```

```
. RESTORE FROM MEMFILE
```

```
. DISPLAY MEMORY
ONE      (N)      1.0000
ALFABET  (C)      ABCDEFGHIJKL
CHARS    (C)      ABCDEFGHIJKL NEW STUFF
** TOTAL **      03 VARIABLES USED      00042 BYTES USED
```

RELEASE ALL LIKE M\* (release all beginning with M).

SAVE ALL LIKE ?DATE TO DATES (same MDATE, BDATE, ZDATE, etc.).

## SELECT... 132

### SELECT

```
SELECT [PRIMARY  ]  
      [SECONDARY]
```

This command causes dBASE to select one of the two possible database areas for future operations. This permits the dBASE user to do operations on two databases at a time, such as using the data from one database to update the data in another database, or comparing the data in two databases, or any of a number of other multi-database operations.

When dBASE is initiated, the PRIMARY area is active. PRIMARY will stay active until a SELECT SECONDARY instruction is given. The secondary area will then be active until a SELECT PRIMARY command is encountered. A different database may be USE'd in each of the areas. This permits the (nearly) concurrent usage of two databases at once. There is no effect if a SELECT SECONDARY is entered when the secondary area is already selected or vice versa with the primary area.

When both database areas have databases in USE, field variables can be extracted from either area. That is to say, any expression can use variables from either database region. If the field names in both regions are the same for a desired variable, then the variable can be prefixed with a "P." or "S." to denote which database it is to come from.

dBASE commands that cause movement of the database (i.e. GOTO, SKIP, REPORT, SORT, COPY, LIST, DISPLAY and others) affect only the currently selected database. The SET LINKAGE ON command will allow all sequential commands (those that have a <scope> parameter) to perform positioning on both the secondary and the primary databases. (See the SET command.) The REPLACE command will only affect variables in the currently selected database. The DISPLAY STRUCTURE command will display the structure of the currently selected database only.

#### Examples:

```
. USE SHOPLIST
```

```
. LIST
```

00001	Beans	5	0.75
00002	Bread loaves	2	1.06
00003	T-Bone steak	4	4.33
00004	Paper plates	1	0.94
00005	Plastic forks	5	0.42
00006	Lettuce	2	0.53
00007	Bleu cheese	1	1.96
00008	Milk	2	1.30
00009	Charcoal	2	0.75

```
. NOTE NOW OPEN ANOTHER DATABASE IN THE SECONDARY AREA
```

. SELECT SECONDARY

. USE SHOPCOST

. LIST

00001	800104	31.38
00002	800111	45.69
00003	800118	51.18
00004	800124	48.19
00005	800201	55.82
00006	800209	12.04
00007	800229	12.04

. SELECT PRIMARY

. SUM COST

12 04

. SELECT SECONDARY

. APPEND

RECORD 00008

DATE : 800303

AMOUNT : 12.04

RECORD 00009

DATE : (cr)

. SUM AMOUNT

268.38

. NOTE EITHER DATABASE'S VARIABLES CAN BE ACCESSED

. DISP OFF COST,AMOUNT,ITEM,DATE

0.75

12.04

Charcoal

800303

. NOTE THE SAME DATABASE CAN BE USED IN BOTH AREAS

. USE SHOPLIST

. NOTE BUT ONE MUST BE CAREFUL SINCE THE VARIABLE NAMES ARE IDENTICAL

. NOTE IN BOTH DATABASES





8. LINKAGE      ON      All sequential commands (LIST, REPORT, SUM, i.e. those that allow a <scope>) will increment the PRIMARY and SECONDARY record pointers simultaneously, but does not force them to be equal.
- OFF      Makes PRIMARY and SECONDARY database pointers independent.
9. COLON        ON      Bounds GET data items with colons in @ commands.
- OFF      Removes colons.
10. BELL        ON      Bell rings whenever illegal data is entered or data field boundaries are crossed.
- OFF      Bell is turned off.
11. ESCAPE     ON      An escape character (1B Hex) aborts execution of command files.
- OFF      Escape key will not interrupt processing.
12. EXACT       ON      Requires that character strings match completely (except for trailing blanks) in expressions and the FIND command.
- OFF      Matches will be made on the basis of the length of the second string, e.g. "ABCDEF" = "ABC" is true.
13. INTENSITY   ON      Full-screen operations will use dual intensity screen characters (normal and inverse video on some terminals).
- OFF      Dual intensity will not be used.
14. DEBUG       ON      Output from the ECHO and STEP commands will be sent to the printer so that full-screen commands may be checked out without the screen becoming cluttered.
- OFF      No extra output on the printer.
15. CARRY       ON      Data from the previous record will be carried-over when APPENDING records in the full-screen mode.
- OFF      No carrying will be done.
16. CONFIRM     ON      dBASE will not skip to next field in full-screen editing until a control key (like return) is typed.
- OFF      dBASE will skip to next field any time too many characters are entered.

## SET...136

- |             |            |   |
|-------------|------------|---|
| 17. EJECT   | <u>ON</u>  | REPORT command will eject a page before beginning a new report.   |
|             | OFF        | The page eject will be suppressed.  |
| 18. RAW     | ON         | Spaces are left off.  |
|             | <u>OFF</u> | Places spaces between fields when the DISPLAY and LIST commands are used without the fields list.   |
| 19. DELETED | ON         | Makes dBASE ignore all records marked for deletion. Records marked for deletion cannot be located with the FIND commands nor processed by any command that allows the NEXT phrase (e.g. LIST, LOCATE, COUNT, etc.) except when a command addresses a deleted record by record number. |
|             | <u>OFF</u> | Records marked for deletion can be LOCATED and DISPLAYed (but not COPYed or APPENDED).  |

### Form b parameters and their formats:

1. SET HEADING TO <string>

This form of the SET command saves the <string> internally and prints the string as part of the report header line. The <string> can be up to 60 characters long. (See REPORT for an example.)

2. SET FORMAT TO [SCREEN]  
[PRINT]  
[<format file>]

The first two forms of this SET parameter determine where the output of "@" commands will go. The last form determines where @ commands are READ from. The format file can only contain @ SAY, ?, READ, and GET commands. It must also have the extension .FMT. (See the "@" and READ commands.)

3. SET DEFAULT TO <drive>

This SET command makes the specified disk drive into the default drive. dBASE will assume that inexplicit file names are on this disk drive. This allows command files to be written in such a way (conveniently) that referenced files may be on any drive in the system. This can also be done with &-macros for further generality in disk drive assignment. In the interactive mode of dBASE, this SET command permits implicit file names.

When a default drive has been set, ALL inexplicit file names are set to the dBASE default. This includes form files, command files, memory files, format files, index files, text files as well as database files.

The parameter <drive> may or may not have the colon (:) attached, that is, both "B" and "B:" are acceptable forms of specifying which drive is wanted.

**Note:** This SET command does not affect the MS DOS default drive in any way. The dBASE initial default drive is the same as the MS DOS default drive, the SET DEFAULT redefines dBASE's internal default only while within dBASE.

**Example:**

```
. SET DEFAULT TO B:
```

```
. USE DATEVSYSR
```

(dBASE will access the 'B' drive for this database and all subsequent data and command files.)

```
4. SET ALTERNATE TO [<file>]
```

This form of the SET ALTERNATE command is part of a two step process to write everything that is normally written onto the screen, onto a disk file as well. This includes output that dBASE generates as well as all inputs typed onto the console. This form identifies and opens the receiving disk file. If the <file> existed on the disk prior to this command, it will be overwritten. A subsequent SET ALTERNATE ON begins the echo process.

**Example:**

```
SET ALTERNATE TO B:PRINTFLE
```

```
SET ALTERNATE ON
```

```
.
```

```
.
```

```
any commands
```

```
.
```

```
.
```

```
SET ALTERNATE TO anyfile
```

Everything which appears on the screen or printer will be copied onto (in this example) B:PRINTFLE.TXT, which can be word processed, printed, or saved.

```
5. SET DATE TO mm/dd/yy
```

The format may be yy/mm/dd, dd/mm/yy, etc; m, d, or y may be any numeral. The system date can be set or reset at any time with this command. It however does not perform date/calendar validation like the date request when dBASE is first started.

```
SET DATE TO 12.10.76
```

## SET ... 138

6. SET INDEX TO <index file> [, <index file>, ... <index file>]

SET INDEX TO identifies and sets up as many as seven index files to be used for future operations. If an index file is currently in USE when this command is issued then the old index file is closed and the new one established.

**Note:** when the new index is set up, the database is left positioned where it was, but the index does not point anywhere. A FIND command or GOTO must be issued to set the index pointer before any commands that have a NEXT clause are issued.

The first index file named is considered as the Master Index. All FINDs use only this index and the database will be in the Master Index order (when skipping).

A SET INDEX TO command (with no index files) will release all indexes and the database will be a sequential file.

7. SET MARGIN TO n

This form of the SET command allows the user to control the left margin when a report is printed. All lines to be printed will be offset by n spaces. The n parameter must be a literal number in the range 1 to 254.

8. SET F <n> TO <'new value'>

This form of the SET command allows the user to redefine the function keys. <n> indicates the number of the function key, e.g., F1, F2, F3, etc. <'new value'> denotes the dBASE command word or words you wish to store in the buffer maintained for the specified function key. Whenever the function key is depressed, the entered command will be executed.

Command words must be entered in quotes (as indicated), and must be followed by a ";" to indicate a carriage return, unless you wish to follow function key output with a direct console entry. If you wish to place more than one dBASE command in a key, all but the last command must be followed by an ";". <'new value'> may be up to 30 characters in length.

### Example:

SET F1 TO 'HELP;' — Executes HELP <enter> to access main help file entry.  
SET F3 TO 'DISPLAY RECORD;' — DISPLAY RECORD output at "." prompt. User must enter desired record number and <enter> to execute command.

At installation function key are assigned the following values (which you may change at any time):

Key	Value	Key	Value
F1	'HELP;'	F6	'LIST STAT;'
F2	'DISPLAY;'	F7	'LIST MEMO;'
F3	'LIST;'	F8	'CREATE;'
F4	'LIST FILES;'	F9	'APPEND;'
F5	'LIST STRU;'	F10	'EDIT #;'

**SKIP**

SKIP [+< exp>]  
 [-< exp>]

This command causes the current record pointer to be advanced or backed up relative to its current location.

**Example.**

. USE INVENTORY1

. LIST

00001	136928	13	1673	ADJ. WRENCH	7.13	189	9	0	9.98
00002	221679	9	1673	SM. HAND SAW	5.17	173	4	1	7.98
00003	234561	0	96	PLASTIC ROD	2.18	27	112	53	4.75
00004	556178	2	873	ADJ. PULLEY	22.19	117	3	0	28.50
00005	723756	73	27	ELECT. BOX	19.56	354	6	1	29.66
00006	745336	13	27	FUSE BLOCK	12.65	63	7	2	15.95
00007	812763	2	1673	GLOBE	5.88	112	5	2	7.49
00008	876512	2	873	WIRE MESH	3.18	45	7	3	4.25
00009	915332	2	1673	FILE	1.32	97	7	3	1.98
00010	973328	0	27	CAN COVER	0.73	21	17	5	0.99

. 5

. SKIP -2

RECORD: 00003

. SKIP

RECORD: 00004

. SKIP 3

RECORD: 00007

**SORT**

SORT ON < field> TO < file> [ASCENDING ]  
[DESCENDING]

This command allows the user to sort data files to another file which is different from the original file. The file in USE is sorted on one of the data fields and may be sorted into ascending or descending order. Notice that the USE file remains in USE and is unaltered, and that deleted records are ignored.

While the SORT command allows only one key, a database may be sorted on several keys by cascading sorts: sort on the most minor key first and progress toward the major key. dBASE will only disturb the order of records when necessary. The collating sequence for character fields is the ASCII code. ASCENDING is assumed if neither ASCENDING or DESCENDING is specified.

The sort uses the ASCII collating sequence. This means that the string 'SMITH' is "smaller" than 'Smith' (the expression " 'SMITH' < 'Smith' " would be TRUE).

The INDEX command is contrasted with the SORT command in this way: INDEX, when done, performs nearly all of SORTs duties. Also, INDEX generally allows greater freedom and greater speed than SORT.

Note that SORT performs a physical rewrite of the datafile, so you should have at least as much disk capacity available as the file currently occupies.

. USE SHOPLIST

. LIST

00001	BEANS #303 CAN	5	0.75
00002	BREAD LOAVES	2	0.97
00003	T-BONE STEAKS	4	3.94
00004	PAPER PLATES	1	0.86
00005	PLASTIC FORKS	5	0.42
00006	LETTUCE	2	0.53
00007	BLEU CHEESE	1	1.96
00008	MILK (1/2 GAL)	2	1.30
00009	CHARCOAL, 5# BAGS	2	0.75

. SORT ON ITEM TO SORTFILE

SORT COMPLETE

. USE SORTFILE

. LIST

00001	BEANS #303 CAN	5	0.75
00002	BLEU CHEESE	1	1.96
00003	BREAD LOAVES	2	0.97
00004	CHARCOAL, 5# BAGS	2	0.75
00005	LETTUCE	2	0.53
00006	MILK (1/2 GAL)	2	1.30
00007	PAPER PLATES	1	0.86
00008	PLASTIC FORKS	5	0.42
00009	T-BONE STEAKS	4	3.94

## STORE... 142

### STORE

STORE <exp> TO <memvar>

This command computes the value of an expression and stores the value into a memory variable. If the memory variable did not exist before this command was issued then dBASE will create the memory variable automatically.

Note that STORE will alter only memory variables. Use the REPLACE command to change database field variables.

. RELEASE ALL

. STORE 1 TO ONE

1

. STORE 'ABCDEFGHJKLM' TO ALFABET  
ABCDEFGHJKLM

. STORE ALFABET+' NEW STUFF' TO CHARS  
ABCDEFGHJKLM NEW STUFF

. STORE ONE\*1.0000 TO ONE  
1.0000

. DISPLAY MEMORY

EOF	(L)	.T.	
ONE	(N)	1.0000	
ALFABET	(C)	ABCDEFGHJKLM	
CHARS	(C)	ABCDEFGHJKLM NEW STUFF	
** TOTAL **		04 VARIABLES USED	00042 BYTES USED



**SUM**

```
SUM < field> [< field>] [TO < memvar list>] [< scope>] [FOR < exp>]
                                     [WHILE < exp>]
```

The SUM command adds numeric expressions involving the USE file according to the <scope> and FOR clauses. Up to 5 expressions may be simultaneously summed. If the TO clause is present, the sums are also stored into memory variables (memory variables will be created if they didn't exist prior to the issuance of the SUM command). The default scope of SUM is *all* non-deleted records.

```
. USE SHOPLIST
```

```
. LIST
```

00001	BEANS #303 CAN	5	0.75
00002	BREAD LOAVES	2	0.97
00003	T-BONE STEAKS	4	3.94
00004	PAPER PLATES	1	0.86
00005	PLASTIC FORKS	5	0.42
00006	LETTUCE	2	0.53
00007	BLEU CHEESE	1	1.96
00008	MILK (1/2 GAL)	2	1.30
00009	CHARCOAL, 5# BAGS	2	0.75

```
. SUM COST
```

```
11.48
```

```
. SUM COST FOR NO=1
```

```
2.82
```

```
. SUM COST, NO
```

```
11.48 24
```

```
. SUM COST TO MSUM
```

```
11.48
```

```
. ? MSUM
```

```
11.48
```

```
. DISPLAY MEMORY
```

```
MSUM      (N)      11 48
** TOTAL **      01 VARIABLES USED      00006 BYTES USED
```

```
. ? MSUM*1.10
```

```
12.6280
```

```
. SUM NO*COST,NO,COST,COST/NO
```

```
31.53      24      11.48      5.81
```

## TEXT ... 144

### TEXT

TEXT

All lines following the TEXT command are copied directly to the screen or printer, according to the SET status. Macros are not expanded but continued lines are combined. The copy continues until an ENDTEXT command is encountered.

The TEXT command is a convenient way of incorporating blocks of text into command files without the need of multiple @ SAY or ? commands.

#### Example:

TEXT

Anything that appears within the TEXT/ENDTEXT bounds is copied to the output without being processed by the dBASE command interpreter. The only exception is input lines continued with a semicolon.

ENDTEXT

A TEXT/ENDTEXT command assists in generating form letters and other block data applications. This command can save multiple ?, ??, or @ SAY commands.

#### Example:

? "Dear           " + Name

TEXT

This is an example of the very handy TEXT command in dBASE.

ENDTEXT

? "

Yours truly,"

**TOTAL**

```
TOTAL ON <key> TO <database> [FIELDS <list>] [FOR <exp>]
                               [WHILE <exp>]
```

The TOTAL command is similar to the subtotal capability in the REPORT command except that the subtotals are placed into a database instead of printed. This allows condensation of data by eliminating detail and summarizing.

**Note:** The USE database must be either presorted by the key or indexed on the key.

If the TO database was defined (if it existed and had a structure), then its structure will be left intact and used to decide which fields will be totaled arithmetically.

If the TO database did not exist prior to this TOTAL command, then the structure from the USE database will be copied to the TO file.

This command is most selective when the TO database exists and the FIELD phrase is included in the command. In this case, only the numeric fields in the FIELDS are totaled. In any other configuration of this command, all numeric fields are totaled.

TOTAL can also be used to remove duplicate records from a database since a non-numeric field in the FIELDS list is not totaled (naturally) and is not flagged as an error.

**Example:**

```
. USE ORDERS INDEX ORDERS

. DISP STRU
STRUCTURE FOR FILE:  ORDERS.DBF
NUMBER OF RECORDS:  00008
DATE OF LAST UPDATE: 12/14/82
PRIMARY USE DATABASE
FLD          NAME                TYPE      WIDTH      DEC
001          CUSTOMER            C          020
002          PART:NO             C          005
003          AMOUNT              N          005
** TOTAL **                                00031

. LIST
00003 HARRIS, ARNOLD              11528      44
00007 JUAN, DON                  21828      5
00001 SWARTZ, JOE                31415     13
00005 MACK, JAY                  31415      3
00008 SALT, CLARA                70296      9
00002 SWARTZ, JOE                76767     13
00006 TERRY, HANS                76767      5
00004 ADAMS, JEAN                89793     12
```

TOTAL... 146

(Imagine that the warehouse needs to know how many of each item to bring out. By totaling on the quantity as long as the part numbers are the same, a database is generated that contains part numbers and the number needed.)

(The database CALLS has already been defined.)

. TOTAL ON PART:NO TO CALLS  
00006 RECORDS COPIED

. USE CALLS

. DISP STRU  
STRUCTURE FOR FILE: CALLS.DBF  
NUMBER OF RECORDS: 00006  
DATE OF LAST UPDATE: 12/21/82  
PRIMARY USE DATABASE

FLD	NAME	TYPE	WIDTH	DEC
001	PART:NO	C	005	
002	AMOUNT	N	005	
** TOTAL **			00011	

. LIST

00001	11528	44
00002	21828	5
00003	31415	16
00004	70296	9
00005	76767	18
00006	89793	12

(Note: two orders totaled)

(Note: two other orders totaled)

**UPDATE**

```
UPDATE FROM <database> ON <key> [ADD <field list>] [RANDOM]
      [REPLACE [ <field list> ] ]
      [<field> WITH <from-field>]
```

The UPDATE command revises the USE file by using data from a second database to modify the USE database. Updated items can be either summed or replaced in entirety. A record is updated if a match occurs when a specified key field in the USE database is compared to a key field in the FROM database. These fields are supplied with the ON phrase.

If the RANDOM phrase is used, then <key> is assumed to be a *single* field in the FROM <file> that can match indexes in the USE database (the USE database must be indexed). Records in the FROM file can be in any order. As each record is read from the FROM file, an internal FIND is done to find the record in the USE database.

If the RANDOM option is not used, then the USE database must be either pre-sorted by the key or indexed on the key. The FROM database must be pre-sorted by the key. Both databases are 'rebound' and a record is read. If the keys match, the add or replace action takes place as directed. If the key in the USE file is smaller (in sort sequence) than the key in the FROM database, then no action takes place, and the record is skipped and left unchanged. Similarly, if the FROM key is smaller, no updates happen and that record is skipped.

**Example:**

```
. USE INVUPDAT

. DISPLAY STRUCTURE
STRUCTURE FOR FILE: INVUPDAT.DBF
NUMBER OF RECORDS: 00003
DATE OF LAST UPDATE: 12/18/82
PRIMARY USE DATABASE
FLD          NAME          TYPE          WIDTH          DEC
001          PART:NO       C             005
002          ON:HAND       N             005
003          COST          N             010             002
** TOTAL **
                                00021

. LIST
00001          21828          77             35.88
00002          70296          0              250.00
00003          89793          2             134999.00
```

(Notice that the database is sorted on the "key" PART:NO.)

**UPDATE . . . 148**

. USE INVENTORY INDEX INVENTORY

. DISPLAY STRUCTURE

STRUCTURE FOR FILE: INVENTORY.DBF

NUMBER OF RECORDS: 00008

DATE OF LAST UPDATE: 12/18/82

PRIMARY USE DATABASE

FLD	NAME	TYPE	WIDTH	DEC
001	ITEM	C	020	
002	COST	N	010	002
003	PART:NO	C	005	
004	ON:HAND	N	005	
** TOTAL **			00041	

. DISP ALL

00008	#9 COAL	22.00	11528	16
00005	SINK, KITCHEN	34.72	21828	77
00001	TIME STITCH	9.99	24776	1
00002	WIDGET	1.67	31415	18
00007	RINGS, GOLDEN	200.00	70296	5
00006	TROMBONES	198.37	76767	76
00004	TANK, SHERMAN	134999.00	89793	5
00003	GADGET, LARGE	16.33	92653	7

(Again notice that the database is indexed on the "key" PART:NO.)

. UPDATE ON PART:NO FROM INVUPDAT ADD ON:HAND REPLACE COST

. LIST

00008	#9 COAL	22.00	11528	16
00005	SINK, KITCHEN	35.88	21828	154
00001	TIME STITCH	9.99	24776	1
00002	WIDGET	1.67	31415	18
00007	RINGS, GOLDEN	250.00	70296	5
00006	TROMBONES	198.37	76767	76
00004	TANK, SHERMAN	134999.00	89793	7
00003	GADGET, LARGE	16.33	92653	7

(Note—the two new Sherman tanks were added to the database and the cost of the golden rings and the kitchen sinks were replaced with the new prices.)

**USE**

USE [< database file> ]

USE < database file> INDEX < index file> [, < index file> ... < index file> ]

**Example:**

```
. USE DATABASE INDEX NAME,CITY,PART:NO,SALESMAN
```

The USE command specifies which (pre-existing) database file is to be the file in USE. If there was a USE file prior to this command, the old file is closed. If a filename is not specified in the command, then the previous USE file is closed.

The second form of USE is to specify a database for operation and an associated index file (which was previously created by the INDEX command) and permits subsequent index operations such as FIND and indexed sequential file access.

Up to seven index files may be USED with any one database at the same time. The first index file named is considered as the Master Index. All FINDs use only this index and the database will be in the Master Index order (when skipping). All of the named index files will be automatically updated anytime their keys are modified (by APPEND, EDIT, REPLACE, READ, or BROWSE commands).

**Examples:**

```
. USE EXAMPLE
```

```
. USE TRACE INDEX TRACE
```

## WAIT ... 150

### WAIT

WAIT [TO < memvar > ]

This command causes dBASE to cease operations until any character is entered from the keyboard, the message WAITING is displayed on the screen. If the TO clause is specified, then the single keystroke that releases dBASE from the wait-state will be entered into the memory variable.

The TO option is most useful when only a single character is required to direct the action of a command file process e.g. menu selections. Notice that a carriage return is not necessary to "send" the character as in the ACCEPT and INPUT commands.

If any non-printable character (i.e. RETURN, LINE FEED, or any other control character) is typed as the response to a WAIT TO command, the value of the memory variable is set to a blank.

#### Example:

```
. RELEASE ALL
```

```
. WAIT TO ACTION  
WAITING 1
```

```
. DISP MEMO  
ACTION (C)  
** TOTAL **
```

```
1  
01 VARIABLES USED 00002 BYTES USED
```



## APPENDIX A      LIST OF COMMANDS

? <exp> [, <exp> ]  
 @ < coordinates> [SAY <exp> [USING '<picture>']]  
                   [GET <variable> [PICTURE '<picture>']]  
 ACCEPT ["<cstring>"] TO <memvar>  
 APPEND [FROM <file> [SDF] [DELIMITED] [FOR <exp> ]] or [BLANK]  
 BROWSE [FIELDS <field list>]  
 CANCEL  
 CHANGE FIELD <list> [<scope>] [FOR <exp>]  
 CLEAR [GETS]  
 CONTINUE  
 COPY TO <file> [<scope>] [FIELD <list>] [FOR <exp>] [SDF]  
           [DELIMITED [WITH <delimiter>]] or [STRUCTURE]  
 COUNT [<scope>] [FOR <exp>] [TO <memvar>]  
 CREATE [<filename>]  
 DELETE [<scope>] [FOR <exp>]  
 DELETE FILE <file>  
 DISPLAY [<scope>] [FOR <exp>] [<exp list>] [OFF] [FIELDS <field list>]  
 DISPLAY STRUCTURE  
 DISPLAY MEMORY  
 DISPLAY FILES [ON <disk drive>] [LIKE <skeleton>]  
 DO <file>  
 DO WHILE <exp>  
 EDIT  
 EJECT  
 ELSE  
 ENDDO  
 ENDIF  
 ENDTEXT  
 ERASE  
 FIND <key>  
 GO or GOTO [RECORD], or [TOP], or [BOTTOM], <n>  
 HELP [command]  
 IF <exp>  
 INDEX  
 INDEX ON <char string expression> TO <index file name>  
 INPUT ["<cstring>"] TO <memvar>  
 INSERT [BEFORE] [BLANK]  
 JOIN TO <file> FOR <expression> [FIELDS <field list>]  
 LIST  
 LOCATE [<scope>] [FOR <exp>]  
 LOOP  
 MODIFY STRUCTURE  
 MODIFY COMMAND <command file>  
 NOTE or \*  
 PACK

## APPENDIX A... 152

QUIT  
READ [NO UPDATE]  
RECALL [<scope>] [FOR <exp>]  
REINDEX  
RELEASE [<memvar list>] [ALL [LIKE <skeleton>]]  
REMARK  
RENAME <current file name> TO <new file name>  
REPLACE [<scope>] <field> WITH <exp> [AND <field> WITH <exp>]  
REPORT [<scope>][FORM <form file>] [TO PRINT] [FOR <exp>]  
RESET  
RESTORE FROM <memfile> [ADDITIVE]  
RETURN  
SAVE TO <file> [ALL LIKE <skeleton>]  
SELECT [PRIMARY or SECONDARY]  
SET <parm> [ON] [OFF]  
SET ALTERNATE TO <file>  
SET DEFAULT TO <drive>  
SET DATE TO <string>  
SET FORMAT TO <format file name>  
SET HEADING TO <string>  
SET INDEX TO <index file list>  
SET MARGIN TO <n>  
SKIP <+/-> [<n>]  
SORT ON <field> TO <file> [ASCENDING], or [DESCENDING]  
STORE <exp> TO <memvar>  
SUM <field> [<scope>] [TO <memvar list>] [FOR <exp>]  
TEXT  
TOTAL TO <file> ON <key variable> [FIELDS <field list>]  
UPDATE FROM <file> ON <key variable> [ADD <field list>] [REPLACE <field list>]  
USE <file> [INDEX <index file name>]  
WAIT [TO <memvar>]

**FUNCTIONS:**

@(&lt; string1&gt; .&lt; string2&gt; )

\*

#

'(&lt; char string&gt; )

\$(&lt; char string&gt; .&lt; start&gt; .&lt; length&gt; )

CHR(&lt; numeric expression&gt; )

DATE( )

EOF

FILE(&lt; file&gt; )

INT(&lt; numeric expression&gt; )

LEN(&lt; char string&gt; )

LOCK( )

RANK(&lt; string&gt; )

STR(&lt; numeric expression&gt; .&lt; width&gt; [.&lt; decimals&gt; ])

VAL(&lt; char string&gt; )

TRIM(&lt; char string&gt; )

TYPE(&lt; exp&gt; )

AT function

deleted record function

record number function

upper case function

substring function

numeric to ASCII

system date function

end-of-file function

existence function

integer function

length function

lock function

ASCII value function

string function

value function

trims strings

supplies data type

APPENDIX B      LIMITATIONS AND CONSTRAINTS

number of fields per record .....	32 max
number of characters per record .....	1000 max
number of records per database .....	65535 max
number of characters per character string .....	254 max
accuracy of numeric fields .....	10 digits
largest number .....	$1.8 \times 10^{63}$ approx
smallest number .....	$1.0 \times 10^{-63}$ approx
number of memory variables .....	64 max
number of characters per command line .....	254 max
number of expressions in SUM command .....	5 max
number of characters in REPORT header .....	254 max
number of characters in index key .....	99 max
number of pending GETS .....	64 max
number of files open at one time .....	16 max



**APPENDIX C      ERROR MESSAGES**

**BAD DECIMAL WIDTH FIELD**

**Redefine Decimal.**

**BAD FILE NAME**

**Syntax error in filename.**

**BAD NAME FIELD**

**Redefine Name.**

**BAD TYPE FIELD**

**Must be C, N, or L.**

**BAD WIDTH FIELD**

**Redefine size of data field.**

**CANNOT INSERT — THERE ARE NO RECORDS IN DATABASE FILE**

**Use the APPEND command instead.**

**CANNOT OPEN FILE**

**Internal error, contact dealer for support.**

**COMMAND FILE CANNOT BE FOUND**

**Check spelling.**

**DATA ITEM NOT FOUND**

**DATABASE IN USE IS NOT INDEXED**

**FIND is only permitted on indexed databases.**

**DIRECTORY IS FULL**

**The MS-DOS disk directory cannot hold anymore files.**

**DISK IS FULL**

**No space left on disk.**

**END OF FILE FOUND UNEXPECTEDLY**

**The database in USE is not in the correct format. If all records are correct and present, then PACK and re-INDEX the database.**

**"FIELD" PHRASE NOT FOUND**

**Rewrite command line.**

**FILE ALREADY EXISTS**

**FILE DOES NOT EXIST**

## APPENDIX C . . . 156

FILE IS CURRENTLY OPEN

Type a **USE** or **CLEAR** command to close the file.

FORMAT FILE CANNOT BE OPENED

FORMAT FILE HAS NOT BEEN SET

**SET** the appropriate format file.

ILLEGAL DATA TYPE

ILLEGAL GOTO VALUE

Must be  $> 0$  and  $< 65535$

ILLEGAL VARIABLE NAME

Only alphanumerics and colons are allowed in variable and field names.

INDEX DOES NOT MATCH DATABASE

**dBASE** cannot match the key with the database. Try another index file.

INDEX FILE CANNOT BE OPENED

Check spelling or **INDEX** the database.

JOIN ATTEMPTED TO GENERATE MORE THAN 65,534 RECORDS

The **FOR** clause allows too many joined output records, make it more stringent.

KEYS ARE NOT THE SAME LENGTH

MACRO IS NOT A CHARACTER STRING

**&macros** must be character strings.

MORE THAN 5 FIELDS TO SUM

Sum is limited to 5 fields.

NESTING LIMIT VIOLATION EXCEEDED

NO EXPRESSION TO SUM

NO "FOR" PHRASE

Rewrite command with correct syntax.

NO "FROM" PHRASE

Rewrite command with correct syntax.

## NO FIND

More a diagnostic type message than an error message. dBASE couldn't find the key.  
Record # has been set to 0.

## NON-NUMERIC EXPRESSION

## NONEXISTENT FILE

## NUMERIC FIELD OVERFLOW

Increase numeric field width, if possible, to a maximum of 10.

## "ON" PHRASE NOT FOUND

Rewrite command with correct syntax.

## OUT OF MEMORY FOR MEMORY VARIABLES

Reduce the number or size of memory variables.

## RECORD LENGTH EXCEEDS MAXIMUM SIZE (OF 1000)

Reduce size of some fields or create a second database on a common key.

## RECORD NOT IN INDEX

Index file was not updated after a record was added. Reindex.

## RECORD OUT OF RANGE

Record number greater than number of records in database. The Record doesn't exist.

## SORTER INTERNAL ERROR, NOTIFY SCDP

Internal error, contact dealer for support.

## SOURCE AND DESTINATION DATA TYPES ARE DIFFERENT

Check that both datatypes are numeric or both character

## \*\*\* SYNTAX ERROR \*\*\*

## SYNTAX ERROR IN FORMAT SPECIFICATION

## SYNTAX ERROR, RE-ENTER

## "TO" PHRASE NOT FOUND

Rewrite command with correct syntax.

## TOO MANY CHARACTERS

Shorten the command line.

## TOO MANY FILES ARE OPEN

There is a maximum of 16 files allowed to be open at one time.

## **APPENDIX C . . . 158**

### **TOO MANY MEMORY VARIABLES**

**There is a maximum of 64 memory variables.**

### **TOO MANY RETURNS ENCOUNTERED**

**Probably an error in the structure of a command file.**

### **"WITH" PHRASE NOT FOUND**

**Rewrite command with correct syntax.**

### **UNASSIGNED FILE NUMBER**

**Internal error, contact dealer for support.**

### **\*\*\* UNKNOWN COMMAND**

**Check spelling.**

### **VARIABLE CANNOT BE FOUND**

**Need to create the variable, or check the spelling.**



## APPENDIX D SET COLOR TO

If you have a Model 2000 with a color monitor, the SET COLOR command allows you to manipulate the screen colors. The syntax for this command is

```
SET COLOR TO <n1,n2>
```

To change the screen colors you must be in color graphics mode by executing the MS-DOS Mode Command and before bringing up dBASE II, then in dBASE II enter:

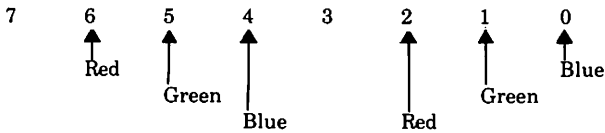
```
SET INTENSITY ON
SET COLOR TO <n1,n2>
```

n1 = a highlighted color, 1-255

n2 = a normal intensity color, 1-255

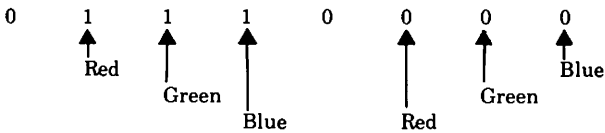
The lower four bits set the color of the characters on the screen (foreground), the upper four bits the color of the screen behind the characters (background). If all lower or upper bits are off (0) the foreground or background will be black.

Bit Pattern

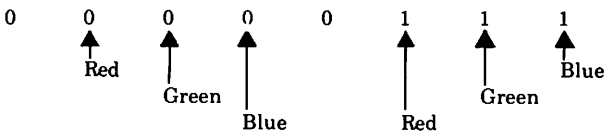


The default value for n1 is 112. The binary equivalent of 112 is 01110000, so the lower four bits are all set off (black). Therefore, the characters will be black and the background will be white (Red + Green + Blue) in the high intensity mode. The default value for n2 is 7, which has a binary value of 00000111. The higher bits are off (black), yielding a black background behind the characters. The foreground characters are white (Red + Green + Blue).

Default value of n1 = 112. Binary equivalent is 01110000.



Default value of n2 = 7. Binary equivalent is 00000111.



## APPENDIX D . . . 160

Here are some sample byte values for various colors. These are sample values for n1 and n2. These all result in a black background.

0-Black (Binary 00000000)  
1-Blue (Binary 00000001)  
2-Green (Binary 00000010)  
3- Blue + Green (Cyan) (Binary 00000011)  
4-Red (Binary 00000100)  
5-Red + Blue (Magenta) (Binary 00000101)  
6-Red + Green (Yellow) (Binary 00000110)  
7-Red + Green + Blue (White) (Binary 00000111)

**Example:**

SET COLOR TO 4, 1

Sets high intensity color to red, normal intensity to blue. The background color is black.

FOR MORE INFORMATION, CONSULT YOUR OPERATING SYSTEM MANUAL.

## INDEX

This index is designed so that you can find the desired information with a minimum of effort. The **boldface** page numbers are the best entry points for any topic. These **boldface** numbers point to the most comprehensive coverage for any subject matter.

! format character, B38  
 ! lowercase-to-uppercase function, **A77, A95, B13**  
 " delimiter, B88  
 # format character, B38  
 # not equal operator, A94, B15  
 # record number function, A73, A95, B9  
 \$ format character, A85, B38  
 \$ substring function, A78, A95, B10  
 \$ substring logical operator, A36, A38, A94  
 & macro substitution function, A79, A95, **B18**  
 ' delimiter, B88  
 ( ) parentheses for grouping, A35, A94, B15  
 \* comment, A73, B7, B102  
 \* deleted record function, A73, A95, B12  
 \* format character, B38  
 \* multiplication, A35, A94, B15  
 + addition, A35, A94, B15  
 + string concatenation, A39, A94, B16  
 - string concatenation, A39, A94, B16  
 - subtraction, A35, A94, B15  
 . dbASE prompt, A10, B1  
 .AND. Boolean and, A36, A94, B16  
 .DBF database file name extension, A93  
 .DBF database files, B5  
 .FMT format files, A14, A84, B7  
 .FRM report form file name extension, A56, A93  
 .FRM report form files, B7  
 .MEM memory files, A94, B6  
 .NDX index files, A93, B7  
 .NOT. Boolean not, A36, A94, B16  
 .OR. Boolean or, A36, A94, B16  
 .PRG command files, A93, B6  
 .TXT text output file, A48, A94, B7, B19  
 / delimiter for date, B1  
 / division, A35, A94, B15  
 9 format character, A84, A85, B38  
 < less than, A18, A35, A94, B15  
 <...> brackets, A6  
 <= less than or equal to, A18, A35, A94, B15  
 < > not equal to, A35, A94, B15  
 <enter> A3  
 = equal to, A18, A35, A94, B15  
 > greater than, A18, A35, A94, B15  
 >= greater than or equal to, A18, A35, A94, B15  
 ? command, interactive, A23, A66, B21, **B33**  
 ? spacing lines, A71  
 ?? command, B33  
 @ command, **A66-68, B35**  
     data display command, B21  
     and screen or printer, B38  
     and PRINT, A85  
     and READ, B106  
     and TEXT, A83  
 @ statement, B7  
 @ substring search function, A78, A95, B12

— A —

A — format character, A84, B38  
abbreviations for commands and keywords, B27  
abort of full-screen edit, A15  
ACCEPT, B40  
    command file command, B22  
    and entering data, A65-66  
    and GET, A68  
    and INPUT, B88  
accounting programs, A89  
addition of data, A24, B20  
addition of fields, A44  
addition operator, A94  
addition sign, A35, B15  
ADDITIVE phrase and RESTORE, B129  
advanced programmers, A43  
ALL — definition, B26  
ALL LIKE and SAVE, B131  
ALTERNATE parameter, A81, B134  
APPEND:  
    cursor control, A16, B24  
    exit from, A25  
    FROM, example, B46  
    functions, A16  
APPEND BLANK, A68, B41  
APPEND, B41-47, B100  
    adding data, A24, B20  
    and COPY, B19  
    dot prompt, A14  
    entering data, A12  
    foreign data files, A47-48  
    with INDEX, B149  
    and INSERT, B90  
    MODIFY STRUCTURE, A44  
    renaming database fields, A48  
    SDF files, B21  
Appendix A — List of Commands, B151-153  
Appendix B — Limitations and Constraints, B154  
Appendix C — Error Messages, B155-158  
Appendix D — Set Color To, B159-160  
arithmetic operators, A94, B15  
    definition, A35  
    precedence of, A35, B17  
ascending sort, B140  
ASCII collating sequence, B140  
at sign, A67-68, B35  
automatic counting, A57  
automatic summing, A57

## — B —

**B\*-trees indexes, A91**  
 backspace — deletion of last character, B3  
 backup copy of dBASE II, A2  
**BASIC, interfacing with, A47, B19**  
 basic programming structures, A, Section III, Contents  
**BEFORE phrase, B90**  
**BELL, A81, B39, B135**  
**BLANK:**  
     in APPEND command, B41  
     in INSERT command, A25, B90  
**Boolean AND, A36, A94, B16**  
**Boolean NOT, A36, A94, B16**  
**Boolean OR, A36, A94, B16**  
**BOTTOM, GOTO, B79**  
 brackets, square, A6, A34  
**BROWSE, B48**  
     changing records, A51  
     cursor control, B24  
     modifying data, A15, A73, B20, **B48**  
     panning on screen, B24  
     screen message for, A14  
**BROWSE FIELDS:**  
     command, A15, B48  
     example, B48  
 built-in editor, A62

## — C —

**CALL, B31**  
**CANCEL, B22, B49**  
 cancellation of command, B49  
**CARRY, B90**  
**CARRY parameter, B135**  
**CARRY SET ON and INSERT, B90**  
**CASE, DO, B68**  
**CHANGE, A49-51, B20, B50**  
 changes in data fields, B71  
 changing data, commands for, A102-103  
 changing file name, B113  
 char string — definition, B25  
 character — definition, A32  
 character data type, A77, A92  
 character data with ACCEPT, A66  
 character deletion, B3  
 character string:  
     and ACCEPT, B40  
     constant, A31, B8  
     definition, A31  
     in expressions, A34  
     field, B5  
     field in CREATE, B60  
 character string constant - definition, B8  
**CHR:**  
     number-to-ASCII character function, A79, A95, **B13**  
 classes of commands, B20

clause:

- BLANK, B41
  - DELIMITED, B41
  - FIELDS, B145
  - FOR, B27, B143
  - NEXT, B97
  - OTHERWISE, B69
  - PICTURE, B37
  - SDF, B41, B53
  - STRUCTURE, B53
  - USING, B37
  - WHILE, B27, B145
- CLEAR, A73, B35, B51
- CLEAR GETS, A69, B36, B51
- clear screen, A17, A34, B22
- closing of database files, B105
- collating sequence, ASCII, B140
- COLON, B39, B135
- command file, A88, B6
- commands, B22
  - and DO command, B68
  - editing, B100
  - indentation for readability, A65, A71
  - name extension, A93
  - nested, B7
  - planning for, A74
  - procedures in, A64
  - programming for, A74
  - sample, A69
  - setup, A61
  - subsidiary, A64
- command file examples, A69-71
- command summary, A96
- changing data, A102
  - editing data, A102
  - interactive input, A104
  - output, A104
  - programming, A105
  - searching, A104
  - updating data, A102
  - using variables, A103
- commands in functional groups, A100
- commands, language rules, B27
- commands, List of, B151-152
- comments, A73, A88, E7
- comparison, B15-16
- concatenation — definition, A39
- conditional execution, B82
- CONFIRM parameter, B135
- CONSOLE parameter, B134
- constant:
- definition, A31, B8
  - storage of, A31
- constraints of dBASE II, B154
- CONTINUE and LOCATE, B52, B97
- CONTINUE, A55, B21, B52, B97
- control characters, B3
- control keys in BROWSE, A73, B48
- control keys, cursor, B23-24
- control, expressions for, A31
- conventions, typographic, A6

- COPY, A41-46, B53-57**
  - and creation of files, B20
  - file manipulation, B21
  - and foreign data files, A47-49, B19
  - and PACK, B103
  - portion of structure, A43
  - renaming database fields, A48
  - and SDF files, B19, B21
  - and STRUCTURE EXTENDED, B60
  - and TOTAL, A59
- COPY DELIMITED, A47-48, B53, B56-57**
- copy of dBASE II disk, A2
- correction dialog, error, A16, B2
- COUNT, B58-59**
  - automatic, A57
  - data displaying, B21
  - memory variable, B21
  - WHILE, B58
- counting, automatic, A57
- CREATE, B60-63**
  - and APPEND, B42
  - and classes of commands, B20
  - FROM, B60
  - how to, A11
  - and INSERT, B90
  - summary, A28, A87
- creation of database, A11
- creation of files, B20
- creation of structure, B60
- cstring — definition, B25
- ctl-B in BROWSE, A16, A73, B48
- ctl-C:
  - in BROWSE, A15, B48
  - in EDIT function, A15
- ctl-D:
  - in BROWSE, B48
  - in full-screen edit, A15
- ctl-E:
  - in data entry, A13
  - in BROWSE, A15, B48
  - in full-screen edit, A15
- ctl-G:
  - in BROWSE, A15, B48
  - in full-screen edit, A15
- ctl-N, B100
  - in MODIFY function, A16
- ctl-P — print switch toggle, A15, B3
- ctl-Q:
  - in APPEND function, A16
  - in BROWSE, B48
  - in data entry, A14
  - in EDIT function, A15
  - in full-screen edit, A15
  - in MODIFY, A41, B100
- ctl-R:
  - in APPEND function, A16
  - in BROWSE, A15, B48
  - in EDIT function, A14-15

## INDEX . . . 166

ctl-S:  
  in BROWSE, A15, B48  
  in full-screen edit, A15  
  and LIST, A16  
  to stop scrolling, A16  
ctl-T in MODIFY function, A16, B100  
ctl-U:  
  current-line deletion, B3  
  in BROWSE, A15, B48  
  in EDIT function, A15  
ctl-V:  
  in BROWSE, A15  
  in full-screen edit, A15  
ctl-W:  
  in BROWSE, A15, B48  
  in dBASE editor, A62  
  in EDIT, A15  
  exit in INSERT or APPEND mode, A25  
  in MODIFY, A41, B100  
ctl-X:  
  current-line deletion, B3  
  in BROWSE, A15, B48  
  in full-screen edit, A15  
  in MODIFY, A41  
ctl-Y in MODIFY function, A16  
ctl-Z in BROWSE, A16, A73, B48  
current-record pointer, B79, B139  
current-line deletion, B3  
cursor:  
  control keys, B23-24  
  controls, full-screen, B100  
  movement, A13

### — D —

data:  
  addition of, A24, B20  
  changes in fields, B71  
  display, A21, B117  
  editing, A14  
  editing commands, A102-103  
  entry, A12  
  field name — definition, B5  
  field replacement, B114  
  files, foreign, A47  
  input checking, A86  
  item — definition, B8  
  interactive entry, A65  
  modification, A14, A49  
  record format, B5  
  search for, A54  
  standard format, A47-49  
  summary of, A59  
  system format, B19  
  type function, A77  
  types, A11, B92, B6



**database:**  
 basics, A89  
 cleanup, A26  
 creation of, A11  
 field names, TIP, A35  
 file name extension, A93  
 files, B5  
 indexed, A52  
 management system, A89  
 modification of structure, A40  
 organization of, A51, A90-91  
 organization summary, A102  
 records per file, A1  
 renaming of fields, A48  
 structure, A87-88

**databases:**  
 combination of, A102  
 duplication of, A41  
 joining of, A82

**date, entering, A7**  
**DATE() function, B13**  
**dates, valid, B1**

**dBASE II:**  
 distribution diskette, B1  
 files, B5  
 commands, A16  
 defaults in characteristics, A80-81  
 specifications, A1

**DBMS — definition, A89**  
**DEBUG parameter, B39, B135**  
**decimal number, width of field for, B60**  
**decision-making, A62**  
**defaults in form file, B117**

**definitions:**  
**ALL, B26**  
 arithmetic operators, A35  
 char string, B25  
 character, A31  
 character string, A31  
 character string constant, A31-32, B8  
 command, B25  
 command length, A1, B25  
 command verb, B27  
 concatenation, A39  
 constant, A31, B8  
 cstring, B25  
 cstring and ACCEPT, B40  
 data field name, B5  
 data item, B8  
 DBMS, A89  
 delimiter, B25  
 exp, B25  
 exp list, B25  
 expression, B8  
 field, A11, A90, B25  
 field list, B25  
 file, B5, B25  
 file name, A93  
 FOR, B27  
 form file, B26

function, B9  
 index file, B26  
 key, B26  
 length of command, A1, B27  
 literal, B8  
 literal value, A31  
 logical constant, B8  
 macros, B18  
 memory variable, A32  
 memvar, B26  
 memvar list, B26  
 n, B26  
 NEXT n, B26  
 operators, A35-37  
 record, A11, A90  
 RECORD n, B26  
 relational operators, A35  
 reserved word, B28  
 scope, B26  
 statement, B25  
 string, A31  
 string constant, A32  
 substring, A31  
 symbols, B25-27  
 variables, A32, B8  
 WHILE, B27  
 DELETE, B64-65  
   and COPY, A46  
   and database cleanup, A26  
   editing of data, B20  
   file manipulation, B21  
   and PACK, B64, B103  
   and RECALL, A26, B108  
 DELETE FILE, B64  
 DELETE NEXT, B64  
 DELETE RECORD, A15, A26, B64  
 deleted record function, A73, B12  
 deleted record mark, A15  
 deletion of fields, A44  
 DELIMITED:  
   and APPEND, B41  
   and COPY, A48  
   keyword, B41, B53  
   option, B19  
 delimiter — definition, B25  
 delimiters, INPUT, B88  
 delimiters, string, A34  
 descending sort, B140  
 device-controlling commands, B22  
 dialog, error correction, A17, B1  
 disk, copy of dBASE II, A2  
 DISPLAY, A16, A20, B66-67  
   and EDIT, A15  
   and FIELDS, B66  
   memory variables, B21  
   selected data, B21  
   vs LIST, B96  
 DISPLAY FILES, B66  
 DISPLAY MEMORY, B66  
 DISPLAY OFF, B33, B66

DISPLAY STATUS, B66  
 DISPLAY STRUCTURE, A45, A97, B66-67, B132  
 distribution diskette, dBASE, B1  
 division operator, A94  
 division sign, A35, B15  
 DO CASE, B68  
 DO, B6, B21-22, B68-70  
 DO WHILE, A64, B68-70, B82  
 DO WHILE loop, A64, B99  
 dot prompt, A14  
 double quotation mark delimiter, B88  
 duplicate record removal, B145  
 duplication of databases, A41  
 duplication of structures, A41

— E —

ECHO parameter, B134  
 EDIT:  
   and BROWSE, A14, A73  
   command, A53, B20, B71  
   cursor control, B23  
   functions, A15  
   modification of data with, A14, A28  
 edit, abort of full-screen, A15  
 editing:  
   and built-in editor, A62  
   commands for, A102-103  
   data, B20  
   full-screen features, A12, A15, A83  
 EJECT, B74  
   and @, B35  
   device control, B22  
   and long forms, A87  
   and REPORT, A55  
 EJECT parameter, B136  
 ELSE, A62-63, B22, B82  
 END CASE, B69-70  
 ENDDO, A64, A69, B22, B68, B99  
   and command files, B22  
   and DO, B68  
   and LOOP, B99  
 ENDF, A62-63, B22, B82  
 END OF FILE — message, B97  
 end of file and RETURN, B130  
 end-of-file function, A74, B12  
 END OF LOCATE — message, B97  
 END RUN dBASE — message, B105  
 ENDTEXT, A83, A98, B144  
 EOF end-of-file function, A74, A95, B12  
 equal to sign, A18, A35, A94, B15  
 ERASE, B75  
   and @ command, B35, B75  
   and command files, A69  
   or CLEAR GETS, A69, B51  
   clear screen, A17, A35, B22  
   housekeeping, A71  
   and READ command, B106  
 error correction dialog, A17, B1

## INDEX . . . 170

error message(s):  
  list of, B155-158  
error recovery examples, B2  
ESC — escape from long-running commands, B3  
ESCAPE key, A50  
ESCAPE parameter, B135  
EXACT parameter, B135  
EXCEPT IN RELEASE command, B111  
execution, conditional, B82  
execution order of operators, B17  
exit APPEND mode, A25  
exit INSERT mode, A25  
exp — definition, B25  
exp list — definition, B25  
expansion:  
  of commands with expressions, A17  
  of commands with relational operators, A17  
  of control, A77  
expression(s):  
  in character strings, A34  
  definition, B8  
  expansion of commands with, A17  
  for selection and control, A31  
  in SUM command, B143  
extensions, file name, A93  
EXTENDED, COPY STRUCTURE, B53-54, B60

— F —

field(s):  
  addition of, A44  
  characters per, A1  
  definition, A11, A90, B25  
  deletion of, A44  
  display in BROWSE, A73  
  extraction of variable, B132  
  length, B60  
  list and BROWSE, A15  
  list — definition, B25  
  per record, A1  
  size, B6  
field names, A92, B60  
  characters for, B27  
  specification, A11, B60  
FIELDS:  
  and BROWSE, B48  
  and DISPLAY, B66  
  and JOIN, B93  
  and LIST, B96  
  and TOTAL, B145  
file — definition, B5, B25  
FILE function, A79, A95, B14  
file handling system, A89  
file manipulating commands, B21  
file name(s):  
  change, B113  
  definition, A93, B25  
  extension:  
    .DBF, A93, B5, B113  
    .FMT, A14, A25, A94, B7

.FRM, A56, A93, B7  
 .MEM, A94, B6  
 .NDX, A93, B7  
 .PRG, A61, A93, B6  
 .TXT, A48, A94, B7  
 extensions, A93-94, B5  
 RENAME, B113  
 file(s):  
   command, A88  
   creation of, B20  
   dBASE, B5  
   FORM, B117  
   LIST, B96  
   operations, A101  
   records per, A91  
   structure commands, A100-101  
 FILES, DISPLAY, A20, B66  
 FILES, FORMAT, A14, A25  
 FILES, LIST, A19  
 FIND, A53  
   and indexed files, A91  
   features, B76-78  
   positioning command, B21  
   and SET, B138  
 first/last record, B79  
 FOR clause:  
   definition, B27  
   and COUNT, B58  
   and RECALL, B108  
   and REPLACE, B114  
   and SUM, B143  
 foreign data files, A47  
 form feed command, B74  
 FORM file, B117  
 form file — definition, B26  
 FORM phrase, B117  
 FORMAT FILES, A14, A25  
 FORMAT TO SCREEN, B39  
 format(s), B37  
   characters, B38  
   files, A84, B7  
   printed page, A85  
   for report, A56  
   system data format, B41, B53  
 formatting, full-screen, A83  
 forms, printing of, A86  
 forms setup, A86  
 FROM, APPEND, B41  
 full-screen:  
   and APPEND, B42  
   and BROWSE, B48  
   cursor controls, A15, B23, B100  
   and EDIT, B71  
   features, A15  
   and formatting, A83  
   and MODIFY COMMAND, B100  
   operation without, A12-13  
   and READ, B106  
 function(s):  
   character, A79

data type, A77  
DATE(), B13  
definition, B9  
deleted record, A73, B12  
end of file, A74, B12  
FILE, A79, B14  
integer, A77, B9  
integer to string, A78  
length, B11  
list of, A95, B153  
lowercase to uppercase, A77  
macro substitution, A79  
number to character, B13  
RANK, A80, A95, B11  
record number, A73, B9  
string, A77  
string length, A78  
string to integer, A78  
string to numeric, B11  
substring, A78, B10  
substring search, A78, B12  
summary, A95  
TRIM, A40, A80, A95, B14, B83  
TYPE, A77, A95, B14  
uppercase, B13  
functional groups, commands in, A100

— G —

GET, A66, A83-85, B35, B39, B106  
GETS keyword in CLEAR command, B51  
GO BOTTOM, A22, B79  
GO, A21-22, B79  
GO TOP, A22, B79  
GOTO, A22, B21, B79  
greater-than-or-equal-to-sign, A18, A35, A94, B15  
greater-than sign, A35, A94, B15

— H —

hardware environment, A1  
HELP, A98, B81  
hierarchical DBMS, A90

— I —

IF, B22, B82  
IF..ELSE..ENDIF, A62-63, B82  
IF..ELSE, nesting with, A63  
indentation for readability, A65, A71, A88  
INDEX, A51-55, B83-87  
    and database organization, A91  
    and EDIT, B71  
    and file creation, A20, B20  
    and FIND, B83  
    and LOCATE, B97  
    and PACK, B103  
    and REINDEX, B110  
    and SORT, B140  
    and TOTAL, A58  
    and USE, B149

INDEXed file and EDIT, B71  
 index file(s), B7  
   definition, B26  
   name extension, A93  
 index key and REPLACE, B114  
 index pointer, B138  
 indexed database:  
   advantages, A91  
   and FIND, A54, B76  
   and LOCATE, A54  
 indexes, B\*-trees, A91  
 indexing, A91  
 INPUT, A66, A72, B22, B88-89  
 input commands, interactive, A104  
 input data checking, A86  
 INSERT, A24, A25, B20, B24, B90-92  
 INSTALL and modification, A8  
 INSTALL program, A3  
 installation of dBASE II, A3  
 INT — integer function, A77, A95, B9  
 INTENSITY, B39  
 INTENSITY parameter, B135  
 interactive ? command, A23  
 interactive data entry, A65  
 interactive input commands, A104  
 international date, B13, B137

— J —

JOIN, A82-83, B6, B20, B93-95  
 joining of databases, A82, B93-95

— K —

key — definition, B26  
 keyword(s):  
   abbreviations for, B27  
   DELIMITED, B41, B53  
   GETS, B51  
   lowercase letters for, B27  
   uppercase letters for, B27  
 key, sort, A91

— L —

last/first record, B79  
 LEN string length function, A78, A95, B11  
 length:  
   character string, A1  
   of command — definition, B27  
   command line, A1  
   field, B60  
   index key, A1  
   report header, A1  
 less-than-or-equal-to sign, A18, A35, A94, B15  
 less-than sign, A18, A35, A94, B15  
 LIKE in RELEASE command, B111  
 limitations of dBASE II, B154  
 line deletion, B3  
 LINKAGE parameter, B135  
 list — definition, B25

LIST, A15, A17-19, B96  
  LIST FILES, A19, B96  
  LIKE in LIST FILES command, B96  
LIST MEMORY, A34, B96  
list of fields and BROWSE, A15  
LIST STATUS, B96  
LIST STRUCTURE, A19, B96  
literal — definition, B8  
literal value — definition, A31  
LOAD, B31  
LOCATE, A53-55, B97-98  
  and CONTINUE, B52  
  and FIND, B76  
  and INDEX, B97  
  positioning, B21  
logical constant — definition, B8  
logical operations, A36, A72, A92, A94,  
  B6, B8, B14, B16, B60  
logical operator precedence, A36, B17  
LOOP, B22, B99  
loop termination, A64  
loop, DO WHILE, B99  
lowercase letters:  
  for commands, B27  
  for field names, A93  
  for keywords, B27  
lowercase-to-uppercase function, A77

— M —

macro substitution function, B18  
mark for deletion, removal of, B108  
memory files, A94, B6  
memory variable(s):  
  alteration, B142  
  characters for names, B27  
  commands, B21  
  and COUNT, A58  
  definition, A32, B6  
  and field names, A35  
  as FIND object, B76  
  RELEASE OF, B111  
  saving of, B131  
  and SUM, A58  
  values of, A68  
MEMORY, DISPLAY, B66, B96  
memvar, B26, B40  
memvar list — definition, B26  
menus, multiple choice in, A63  
merge, records, A82  
MODIFY COMMAND, A40-41, A73, B7,  
  B20, B22, B100  
MODIFY cursor control for, B23  
MODIFY functions, A16  
MODIFY STRUCTURE, A44, B100  
MS-DOS operating system, B4  
multiple choice in menus, A63  
multiple databases, A72  
multiplication operator, A35, A94, B15



## — N —

n — definition, B26  
 names, field, A92, B60  
 naming of variables, A35  
 nested command files, B7  
 nesting of dBASE statements, B28  
 nesting with IF..ELSE, A63  
 NEXT clause, B97  
 NEXT n — definition, B26  
 not-equal-to sign, A35, A94, B15  
 NOTE, B102  
 NOUPDATE option, B106, B114  
 number-to-ASCII function, A79  
 number-to-character function, A79, B13  
 number, largest and smallest, A1  
 numeric accuracy, A1  
 numeric data types, A92, B60  
 numeric expression, adding of, B143  
 numeric field, B6  
 numeric field in CREATE, B60

## — O —

operating system, B4  
 operation(s), B14
 

- arithmetic, B14-15
- comparison, B14-15
- file, A101
- logical, B14, B16
- string, B14, B16

 PACK, A26, B64  
 RECALL, A26, B64  
 operators:
 

- arithmetic, A94
- arithmetic — definition, A35
- definition, A35
- execution order of, B17
- logical, A36, A94
- relational, A12, A94
- string, A39

 option, SDF, B53  
 option, TO, B150  
 organization of databases, A51, A90  
 OTHERWISE clause, B69  
 output, commands for, A104

## — P —

PACK, A26-27, B103-104
 

- and editing, B20
- and INDEX, A53, B83

 PACK operation, A26, B64  
 page formatting, A85, B74  
 page length, A87  
 panning on screen, B24, B48  
 parameter:
 

- ALTERNATE, B134
- BELL, B135
- CALL, B31
- CARRY, B135

**COLON, B135**  
**CONFIRM, B135**  
**CONSOLE, B134**  
**DATE, B137**  
**DEBUG, B135**  
**DEFAULT, B136**  
**DELETED, B136**  
**ECHO, B134**  
**EJECT, B136**  
**ESCAPE, B135**  
**EXACT, B135**  
**FORMAT, B136**  
**HEADING, B136**  
**INDEX, B138**  
**INTENSITY, B135**  
**LINKAGE, B135**  
**MARGIN, B138**  
**PRINT, B134**  
**RAW, B136**  
**SCREEN, B134**  
**STEP, B134**  
**TALK, B134**  
**parentheses for grouping, A35, A36-37, A94, B15**  
**payroll program process, A89**  
**PEEK, B31**  
**period, dBASE prompt, A14, B1**  
**phrase:**  
    **ADDITIVE, B129**  
    **BEFORE, B90**  
    **BLANK, B41, B90**  
    **FIELDS, B93, B145**  
    **FOR, B27**  
    **FORM, B117**  
    **GET, B36, B106**  
    **PICTURE, A84**  
    **SAY, A83, A84-85, B35-36**  
    **USING, A85**  
    **WHILE, B27**  
**PICTURE clause, A84, B37**  
**PLAIN clause with REPORT, B117, B126-127**  
**planning for command files, A74**  
**pointer, current record, B139**  
**pointer, index, B138**  
**POKE, B31**  
**positioning commands, A21, B21**  
    **GO, A21**  
    **GO BOTTOM, A21**  
    **GO TOP, A21**  
    **GOTO, A21**  
    **SKIP, A22**  
**precedence of arithmetic, string, logical operators, A35, B17**  
**PRIMARY in SELECT command, A72**  
**PRINT parameter, B134**  
**print switch toggle, B3**  
**printed page format, A85-86**  
**printer instructions, A87**  
**printing of forms, A85**  
**procedures in command files, A64-65**  
**process, repetition of, A64**  
**programming, A61**  
    **for command files, A74**

commands for, A105  
 prompt, dot, dBASE, A14, B1  
 pseudocode, A74, A88

— Q —

question-mark command, A23, A66, B33  
 quit APPEND mode, A25  
 QUIT, A14, B105  
 quit INSERT mode, A25  
 quotation marks, A34

— R —

RANDOM phrase, B147  
 RANK function, B11  
 RAW parameter, B136  
 READ, B106-107  
   and @, B35, B39  
   data entry, A68  
   editing data, B20  
   display data, B21  
 RECALL, A26, B20, B64, B108-109  
 RECORD n — definition, B26  
 record(s), A91-92  
   characters per, A1  
   per database file, A1  
   definition, A11, A90  
   fields per, A1  
   format, data, B5  
   GOTO, B79  
   merge, A82  
   number function, A73, B9  
   purge of, B103  
   structure, B5  
 refinement, top-down & step-wise, A75  
 REINDEX, B110  
 relational DBMS, A90  
 relational operators, A12, A17, A35, A94  
 RELEASE, A58, B111  
 RELEASE ALL, A34, A58, B111  
 REMARK, A73, B112  
 removal of duplicate record, B145  
 RENAME, A73, B21, B113  
 renaming of database fields, A48  
 repetition of process, A64  
 REPLACE, A49-51, B114-116  
   and editing data, B20  
   and INDEX, A53  
   modifying data, A49-51  
   and SELECT, B132  
   and SORT, A53  
   and STORE, B142  
   and UPDATE, A82, B147  
 REPORT, A55-57, B117-127  
   creation of, B20  
   data display, B21  
   form file name extension, A93  
   form files, B7

format, A55  
 header length, A1  
 preparation, B117  
 and TOTAL, B145  
 requirements, system, A1, B4  
 reserved word — definition, B28  
 RESTORE, B21, B129, B131  
 RETURN, B130  
 and command file, A65, B22  
 and DO, B68  
 return to dBASE II, A50  
 rules, command language, B27-29

— S —

SAMPLE.PRG, A69  
 SAVE, B20-21, B129, B131  
 SAVE TO and RESTORE, B129  
 SAY, A66, A84, B35-36, B38  
 scope — definition, B26  
 screen, clear, A17, B22  
 screen message for CREATE, A11  
 SCREEN parameter, B39, B134  
 scrolling, ctrl-S to stop, A16  
 SDF:  
 and APPEND, B41  
 clause, B53  
 and COPY, B53  
 and foreign data files, A47-48  
 option, B19, B53  
 search for data, A54  
 search, substring, A36  
 searching, commands for, A104  
 SECONDARY in SELECT command, A72  
 SELECT, A72, B6, B21, B132-133  
 SELECT PRIMARY, B93, B132  
 SELECT SECONDARY, B132  
 sequence of commands, A61  
 SET commands, A80-81, B134-138  
 setup, forms, A86  
 sign-on message, B1  
 single drive disk copying, A2  
 single quotation mark delimiter, B88  
 SKIP, A22, B21, B76, B139  
 software environment, A2  
 SORT, A51-53, A91, B21, B140  
 SORT and TOTAL, A59  
 sort key, A91  
 specifications, dBASE II, A1  
 square brackets, A5, A6, A34  
 statement — definition, B25  
 statement, @, B7  
 STATUS, DISPLAY, B66, B96  
 STEP parameter, B134  
 step-wise refinement, A75  
 storage of constants and variables, A31-32, B131  
 STORE, B142  
 constants and variables, A31, A32-33  
 memory variable command, B21  
 and REPLACE, B114

**STR** integer-to-string function, A78, A95, B10

string:

- concatenation, A39, A94, B16
- concatenation with blank shift-right, A94
- constant — definition, A32
- definition, A31
- delimiters, A34
- function, B10
- length function, A78
- operations, B15
- operator precedence, B17
- operators, A39, A94, B16

string-to-integer function, A78, B11

**STRUCTURE**:

- clause, B53
- with COPY, A42, A48
- DISPLAY, A21, B67
- LIST, A19, B96
- MODIFY, B100-101

structure(s):

- creation of, B60
- database, A87
- duplication of, A42
- record, B5

stubs as partial programs, A75

subsidiary command files, A64-65

substring:

- definition, A31
- function, A78, B10
- logical operator, A36, A38, A94
- operator, B15
- search function, A36, A78, B12

subtotal capability, B121, B145

subtraction operator, A94

subtraction sign, A35, B15

**SUM**, A57-58, B21, B143

summary:

- command, A96
- of data, A58
- database organization, A102
- function, A95

summing, automatic, A57

symbol — definition, B25

**SYNTAX ERROR** — error message, B1

system:

- data format, B19
- requirements, A2, B4

system data format:

- and APPEND, B41
- clause, B53
- and COPY, B53
- and foreign data files, A47-48
- option, B19, B53

— T —

**TALK** parameter, A80, B134

terminal cursor X-Y positioning of, A67

termination of dBASE II session, A14

**TEXT**, A83, A100, B144

## INDEX . . . 180

text output file, A94, B7  
top-down refinement, A75  
TOP, GOTO, B79  
TOTAL, A58-59, B20, B145-146  
TRIM function, A40, A79, B14, B83  
TYPE — data type function, A77, B14  
typographic conventions, A6

### — U —

UPDATE, B147-148  
  defaults, A82  
  editing data, B20  
  and RANDOM, A82, B147  
updating data, commands for, A102-103  
uppercase function, B13  
uppercase letters for commands and keywords, B27  
USE, B149  
  entering data, A13  
  file manipulation command, B21  
  USING clause, A85, B36-37

### — V —

VAL string-to-integer function, A78, A95  
variables:  
  commands for using, A103-104  
  definition, A32, B8  
  GET, B39  
  memory, A58, A67, B6  
  naming of, A33  
  storage of, A33  
verb in command, B27

### — W —

WAIT, B150  
  in classes of commands, B22  
  in command files, A66  
  as program test, A75  
  and single character, A71  
what is . . . ? command, A23, A66, B21, B33  
WHILE:  
  and COUNT, B58  
  definition, B27  
  DO, B68  
  RECALL, B108  
  and REPLACE, B114  
width of field, B60  
word separation, B27

### — X —

X format character, A85, B38





















**RADIO SHACK, A DIVISION OF TANDY CORPORATION**

**U.S.A.: FORT WORTH, TEXAS 76102  
CANADA: BARRIE, ONTARIO L4M 4W5**

---

**TANDY CORPORATION**

**AUSTRALIA**

**91 KURRAJONG ROAD  
MOUNT DRUITT, N.S.W. 2770**

**BELGIUM**

**PARC INDUSTRIEL DE NANINNE  
5140 NANINNE**

**U. K.**

**BILSTON ROAD WEDNESBURY  
WEST MIDLANDS WS10 7JN**