ERASE

ERASE

This command clears the screen and places the cursor (if any) 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: that 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).

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 a multiple indexed file if the two keys are placed within quotes.

```
. 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 short-hand 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
```

```
. 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
```

IF

```
IF <exp>
  <commands>
[ELSE
  <commands>]
ENDIF
```

The IF command allows conditional execution of other commands. This command is used in command files. When the <exp>ression 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 DO WHILE. 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
```

See Appendix for further examples.

REINDEX

INDEX

INDEX ON <expression> TO <index file name>

The INDEX command causes the current file in USE to be indexed on the <expression>. <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 specifing 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.

Many times, 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 that one index file is associated with the PACKed database, then that database must 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 S(NAME,N,5)+STR(AMOUNT,5) TO NDXFILE instead of INDEX ON S(NAME,N,N+5)+STR(AMOUNT,SIZEVAR) TO NDXFILE).

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       0.7u

. DISPLAY STRUCTURE
STRUCTURE FOR FILE:   SHOPLIST.DBF
NUMBER OF RECORDS:    00009
DATE OF LAST UPDATE:  07/03/76
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 SHOPINDX

. INDEX ON ITEM TO SHOPINDX

. 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                    1.96

  SKIP -1
    -- 00001

. DISPLAY
00001  Beans                         0.75

. NOTE REGULAR USE COMMAND DOES NOT          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

. 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: 10/16/81
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
```

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 type character. If a numeric expression is entered, <memvar> will be of type numeric. 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

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.

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 STEAKS           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 STEAKS           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 STEAKS           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 STEAKS           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
```

```
. 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 ON expression for each record in the secondary USE file. Each time that the expression yields a TRUE result, a record as 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 a 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 INVENTRY

.DISPLAY STRUCTURE
STRUCTURE FOR FILE:   INVENTRY.DBF
NUMBER OF RECORDS:    00008
DATE OF LAST UPDATE:  00/00/00
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

. 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  THOMBONES                  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:  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

. 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;      use the inventory
FIELD CUSTOMER,ITEM,AMOUNT,COST                file to add names
                                               to the orders

. USE ANNOTATE

. DISPLAY STRUCTURE
STRUCTURE FOR FILE:    ANNOTATE.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       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 INVENTRY

(join customer names with part numbers with insufficent
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

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.

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 expression <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 in 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

. 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

. LOCATE FOR COST>.70
RECORD: 00002

. CONTINUE
RECORD: 00003

. DISP ITEM
T-BONE STEAKS
. CONTINUE
RECORD: 00004
```

```
. CONTINUE
RECORD: 00007

. CONTINUE
RECORD: 00008

. CONTINUE
END OF FILE
```

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 a good programming practice and should be avoided. The following example was done a second time, the second follows the first, without use of the LOOP capability.

Example:

```
STORE 1 TO INDEX
DO WHILE INDEX<10
  STORE INDEX+1 TO INDEX
  IF ITEM='    '              Anytime that ITEM is equal to blanks
    SKIP                      then skip to the next record
    LOOP                      and go back to the DO WHILE
  ENDIF
  DO PROCESS
ENDDO
```

Example 2:

```
STORE 1 TO INDEX
DO WHILE INDEX < 10
  STORE INDEX + 1 TO INDEX
  IF ITEM = '    '
    SKIP
  ELSE
    DO PROCESS
  ENDIF
ENDDO
```

MODIFY

a. MODIFY STRUCTURE
b. MODIFY COMMAND [<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 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 RENAME'd 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 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-o for SuperBrain);
ctl-Q - aborts any changes made to the command file;
ctl-R - scrolls one line down; and
ctl-C - scrolls one page up.
ctl-V - Insert

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

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.

If the file being PACKed is indexed, and the indexed file is in use, then the PACK will adjust the index file at the same time it adjusts the USE file. For large indexed files, doing a PACK on the file without the index and then reindexing is faster.

If the database is indexed by more that one index file, then the other index files must be reINDEXed on those keys since the PACK will (in all probability) have moved records around.

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.

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
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     0.55
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; they will however be COUNTed. It becomes important to know wether 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
```

QUIT

QUIT [TO <com file list>]

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.

If the TO phrase is included, then all the programs in the <com file list> will be executed in sequence by CP/M. This feature lets you to go out of dBASE and chain to other pieces of software.

There is no limit to the number of programs or CP/M commands which can be executed as long as the 254 character limit for any command is not broken. dBASE be reentered an the end of the string of commands. However, it is not required; CP/M will be given control when the string of commands are all finished executing.

Example:

```
. QUIT TO 'DIR B:','PIP PRN:=ALTERMAT.TXT','DBASE CMDFILE'
```

In this example, dBASE is exited, a directory of the B-drive is done, PIP is then called to copy a file to the print device, and dBASE is reentered with a command file (CMLFILE.CMD) taking control immediately.

READ

READ

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 appropiate 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).

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 CHOICE$'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 'XXXXXXXXXXXXXXXXXXXXXXXX'
    @ 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

RECALL [<scope>] [FOR <exp>]

This command removes the mark-for-deletion from the records that were marked by the DELETE command.

Examples:

```
. USE DUPE3

. LIST
 0001  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
```

(This page is left intentionally blank)

(This page is left intentionally blank)

RELEASE

RELEASE [<memvar list>]
        [ALL]

This command releases 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.

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 changeing of the name of a file in the CP/M directory. If no file type (the up to 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

REPLACE [<scope>] <field> WITH <exp> [,<field2> WITH <exp2>] ,etc
[FOR <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>.

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>.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
```

```
STRUCTURE FOR FILE:  CHECKS.DBF
NUMBER OF RECORDS:   00016
DATE OF LAST UPDATE: 10/18/81
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 Tuperware               50.02 .F. .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 Tuperware               50.02 .F. .T.
```

REPORT

REPORT [FORM <form file>] [<scope>] [TO PRINT] [PLAIN]

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. 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.

```
. 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 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 in the database in use will have only as far as the report is concerned (the database will remain unchanged) three periods concatenated to the end of the string. 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      1.00
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 signifigant characters are "<", and ">". In column headings, if the title is preceeded with a "<" then the title will be left-justified in the column. Likewise a ">" will right-justify the title.

Other options in REPORT include totalling, subtotalling, and summary reports. In summary reports, detail records are not displayed, just totals and subtotals. Totalling and subtotalling is done only on fields that are numeric in nature. See the report examples.

Finally a carriage return 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 effect 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 flexability 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.

Examples:

. **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       2.00
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 92553
WARD, CHARLES A.              15
** SUBTOTAL **
                              15

** TOTAL **
                             162
```

Example 3:

Suppose some of your collegues 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 loose his shirt if he didn't be 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: 09/17/81
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 **              0001ö
```

```
. 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 TOTALS 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 totalling 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
05/26/81	29	75	53	157
05/27/81	45	48	63	156
05/28/81	50	56	74	180
05/29/81	86	24	72	182
06/05/81	43	-12	75	130
06/12/81	42	9	27	78
06/26/81	84	35	63	182
07/06/81	33	71	26	130
08/19/81	37	55	38	130
09/15/81	19	57	54	130
09/16/81	15	7	108	130
09/17/81	59	13	58	130
** TOTAL **	715	698	875	2288

A report may also cover just a few of the records in a file. Like:

```
GOTO RECORD 7

REPORT NEXT 4 FORM CARDS
```

PAGE NO. 00001

Hearts Scores

Date of Game	Score Lisa	Score Anna	Score Wayne	Game Total
07/07/81	40	63	27	130
07/09/81	55	41	60	156
07/13/81	40	63	54	157
07/23/81	38	69	23	130
** TOTAL **	173	236	164	573

A report may also ask for inrormation which would meet certain criteria. Like:

```
REPORT FORM CARDS FOR WAYNE < 50

 PAGE NO. 00001

                Hearts Scores

  Date of    Score    Score    Score    Game
   Game      Lisa     Anna     Wayne    Total

 06/12/81      42       9       27       78
 07/06/81      33      71       26      130
 07/07/81      40      63       27      130
 07/23/81      38      69       23      130
 06/19/81      37      55       38      130
 ** TOTAL **
               190     267      141      598
```

```
REPORT FORM NEXT WHILE CUSTOMER >="M"

 PAGE NO. 00001
 12/13/81

      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)
```

TABLES

RESET

RESET

The RESET command is used to reset the CP/M bit map after a diskette has been swapped. Normally, if a diskette is swapped, CP/M will not allow writes to take place until after a warm or soft boot has taken place. RESET attempts to re-open all files which were open prior to the swap. If a file that was open is no longer mounted on an active disk drive, RESET closes the file internally.

WARNING: If a disk is swapped that contains a file with the same name as a file that was previously open, the RESET operation will erroneously not close that file. This condition can be avoided by closing all non-essential files prior to the swap and subsequent RESET command. A USE command with no filename will close the file in USE, a CANCEL command will close any command files that may be open.

Issuing a RESET command when no disk swap has taken place has no effect.

RESTORE

RESTORE FROM <file>

This command reads a file of memory variables. The file must be built using the SAVE MEMORY TO <file> command. All memory variables which were defined previous to the RESTORE command are deleted by this command.

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
```