LOC

Format **LOC (<file number>)**

Purpose With random access files, returns the record number which will be used by the next GET or PUT statement if that statement is executed without specifying a record number. With sequential files, returns the number of file sectors (128-byte areas) which have been read or written since the file was opened.

When the specified file is the PX-4's RS-232C interface, the LOC function returns the number of bytes of data in the RS-232C receive buffer.

Remarks This function can be used to control the flow of program execution according to the number of records or file sectors which have been accessed by a program since the file was opened.

Example

```
10 ON ERROR GOTO 160
20 OPEN"R",#1,"A:LOCTEST",5
30 PRINT "OUTPUT"
40 FIELD#1,5 AS A$
50 FOR A=1 TO 20:LSET A$=STR$(A):PRINT STR$(A);:PUT#1,A:NEXT
60 PRINT
70 CLOSE
80 OPEN"R",#1,"A:LOCTEST",5
90 PRINT "INPUT"
100 FIELD #1,5 AS A$
110 IF LOC(1)>10 THEN 150
120 GET#1
130 PRINT A$;
140 GOTO 110
150 ERROR 230
160 IF ERR=230 THEN PRINT:PRINT "INPUT PAST LIMIT"
170 END

OUTPUT
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
INPUT
 1    2    3    4    5    6    7    8    9    10   11
INPUT PAST LIMIT
Ok
```


LOCATE

Format **LOCATE [<X>],[,[<Y>][,<cursor switch>]]**

Purpose Moves the cursor to the specified position.

Remarks This statement moves the cursor to the screen position whose horizontal character coordinate is specified by <X> and whose vertical character coordinate is specified by <Y>. The value specified for <X> must be in the range from 1 to Xmax and that specified for <Y> must be in the range from 1 to Ymax, where Xmax and Ymax are determined by the size of the virtual screen.

<cursor switch> is a switch which determines the status of the cursor following execution of the LOCATE statement. Cursor display is turned off if 0 is specified for <cursor switch>, and cursor display is turned on if 1 is specified. Normally, the cursor is not displayed during execution of BASIC programs; however, it can be displayed by executing a LOCATE statement with 1 specified for <cursor switch>. The BASIC interpreter also forcibly sets the cursor switch to 1 whenever it returns to the command input mode. (The cursor is also always displayed during execution of an INPUT or LINE INPUT statement, regardless of the status of the cursor switch.)

MO error (Missing operand) — A required operand was not specified in the statement.

FC error (Illegal function call) — The number specified in one of the statement operands was outside of the prescribed range.

OV error (Overflow) — The number specified in one of the statement operands was outside of the prescribed range.

ClibPDF - www.fastio.com

LOF

Format **LOF (<file no.>)**

Purpose Returns the size of a file.

Remarks When the file specified in <file no.> is a disk file, the LOF function returns the size of that file. When the file is open in the "R" mode, the size of the file is calculated based on the record size specified when the file was opened and the maximum record number which has been written to that file. When the file is open in the "I" or "O" mode, its size is calculated based on the number of records written and a record size of 128 bytes.

If the file specified in <file no.> is the RS-232C interface, the LOF function returns the number of bytes remaining in the receive buffer.

Under CP/M, the size of files is determined in 128-byte units. Therefore, if a record size of less than 128 bytes is used for a random access file, the value returned by the LOF function will be greater than that indicated by the maximum record number output.

NOTE:
When the LOF function is executed against a disk file, the disk must be accessed in order to determine the size of the file. Therefore, when a random access file is used in a retrieval program or the like, execution time will be greatly increased if the LOF function is executed repeatedly. To prevent this, substitute the LOF value into a variable at the beginning of the retrieval loop, then refer to the variable inside the loop instead of the LOF function.

LOG

Format **LOG(X)**

Purpose Returns the natural logarithm of X.

Remarks The value specified for X must be greater than zero. LOG(X) is calculated to the precision of the numeric type of expression X.

To obtain the logarithm to another base the mathematical conversion has to be carried out as in the first example below.

To obtain a number from its logarithm (i.e. its antilogarithm) use EXP (X) as shown in the second example.

Example 1

```
10 CLS
20 INPUT "What base logarithm do you want ";N
30 PRINT:INPUT "What number do you want the log of ";X
40 Z = (LOG(X)/LOG(N)):'THIS IS THE FORMULA FOR CONVERTING
NATURAL LOGS TO OTHER BASES
50 PRINT:PRINT "Log to the base ";N;" of ";X;" is ";Z
60 END
```

```
What base logarithm do you want ? 10
What number do you want the log of ? 100
Log to the base 10 of 100 is 2
Ok
```

```
What base logarithm do you want ? 8
What number do you want the log of ? 34
Log to the base  8  of  34  is  1.69582
Ok
```


ClibPDF - www.fastio.com

Example 2

```
10 CLS
20 INPUT "What is the number of which you want the natural
   log ";X
30 PRINT:PRINT"The log to the base e of ";X;" is ";LOG(X)
40 PRINT:PRINT"The antilog (given by EXP(X)) is ";EXP(LOG(X))
50 END
```

```
What is the number of which you want the natural log ? 23
The log to the base e of  23  is  3.13549
The antilog (given by EXP(X)) is  23
Ok
```

```
What is the number of which you want the natural log ? 657
The log to the base e of  657  is  6.48769
The antilog (given by EXP(X)) is  657
Ok
```


LOGIN

Format **LOGIN <program area no.>[,R]**

Purpose Switches between BASIC program areas.

Remarks A number from 1 to 5 is specified in <program area no.>, indicating one of the five BASIC program areas. When the R option is not specified, executing this command causes the BASIC interpreter to switch to the specified program area and stand by for entry of commands in the direct mode. In this case, the variable area is cleared and all open files are closed.

When the R option is specified, the specified program area is selected and the program in that area is executed immediately, starting with its first line. In this case, the variable area is not cleared and any files which are open at the time of command execution remain open.

FC error (Illegal function call) — A number other than 1 to 5 was specified in <program area no.>.

MO error (Missing operand) — A required operand was not specified in the command.

ClibPDF - www.fastio.com

LPOS

Format LPOS(X)

Purpose Returns the current position of the print head pointer in the printer output buffer.

Remarks The maximum value returned by LPOS is determined by the line width which has been set by the WIDTH LPRINT statement, and does not necessarily correspond to the physical position of the print head. This is especially true if a control character has been sent to the printer; see the program below for an example of this.

X is a dummy argument, and may be specified as any numeric expression.

Example In the example below, at the end of line 100 ten characters have been printed. The position in the buffer is thus 11. Line 20 adds two control characters compatible with EPSON printers to cause the printer to change the print style. The first character is a control character, which is ignored by the LPOS function. The second character is used by the printer but not printed; it is in fact the letter "E". The position in the buffer as returned by LPOS is now 12 because only this character has been added. Line 30 adds another ten characters to the line, and thus LPOS returns a value of 22.

```
10 LPRINT "1234567890";:GOSUB 100
20 LPRINT CHR$(27);CHR$(69);:GOSUB 100
30 LPRINT "1234567890";:GOSUB 100
40 END
100 A = LPOS(X):PRINT"Print head pointer is at position ";A
110 RETURN

12345678901234567890

Ok
run
Print head pointer is at position  11
Print head pointer is at position  12
Print head pointer is at position  22
Ok
```


LPRINT/LPRINT USING

Format **LPRINT [<list of expressions>]**
LPRINT USING <format string>;<list of expressions>

Purpose These statements are used to output data to a printer connected to the PX-4.

Remarks These statements are used in the same manner as the PRINT and PRINT USING statements, but output is directed to the printer instead of the display screen.

See also **PRINT, PRINT USING**

Example 1

```
10 A = 3
20 A$ = "There are "
30 LPRINT A$;A;" vowels in `computer'"
40 END

There are  3  vowels in `computer'
```

Example 2

```
10 'THE LPRINT COMMAND
20 A = 3
30 A$ = "There are "
40 LPRINT A$;A;" vowels in 'computer'"
50 LPRINT
60 'THE LPRINT USING "!" COMMAND
70 LPRINT USING "!";"AAA";"BBB";"CCC"
80 LPRINT
90 'THE LPRINT USING "\   \" COMMAND
100 A$ = "123456"
110 B$ = "ABCDEF"
120 LPRINT USING "\ \";A$;B$
130 LPRINT USING "\   \";A$;B$
140 LPRINT
150 'THE LPRINT USING "&" COMMAND
160 LPRINT USING "&";A$;" = ";B$
170 LPRINT
```


ClibPDF - www.fastio.com

```
180 'THE LPRINT USING "#" COMMAND
190 LPRINT USING "####";1;.12;12.6;12345
200 LPRINT
210 LPRINT USING "###.##   ";123;12.34;123.456;.12
220 LPRINT
225 'THE LPRINT USING "+#" COMMAND
230 LPRINT USING "+#### ";123
240 LPRINT
245 'THE LPRINT USING "#-" COMMAND
250 LPRINT USING "####- ";345;-456
260 LPRINT
265 'THE LPRINT USING "**" COMMAND
270 LPRINT USING "**####.##  ";12.35;123.555;555555.88#
280 LPRINT
290 'THE LPRINT USING "$$" COMMAND
300 LPRINT USING "$$####.##  ";12.35;123.555;555555.88#
310 LPRINT
320 'THE LPRINT USING "$**" COMMAND
330 LPRINT USING "$**####.##  ";12.35;123.555;555555.88#
340 LPRINT
350 'THE LPRINT USING "**$" COMMAND
360 LPRINT USING "**$####.##  ";12.35;123.555;555555.88#
370 LPRINT
380 'THE LPRINT USING "##,.##" COMMAND
390 LPRINT USING "#######,.##";555555.88#
400 LPRINT
405 'THE LPRINT USING "##.##^^^^^" COMMAND
410 LPRINT USING "###.##^^^^  ";123.45;12.345;1234.5
415 LPRINT
425 'USING THE UNDERSCORE
435 LPRINT USING "###_%";123
445 LPRINT
455 'USING OTHER CHARAACTERS
465 LPRINT USING "##/##/##";12;34;56
475 LPRINT
485 LPRINT USING "(###)";123
495 LPRINT
505 LPRINT USING "<###>";123
515 LPRINT
```


LSET/RSET

Format

LSET <string variable> = <string expression>
RSET <string variable> = <string expression>

Purpose

These statements move data into a random file buffer to prepare it for storage in a random access file with the PUT statement.

Remarks

<string variable> is a variable which has been assigned to positions in a random file buffer with the FIELD statement. <string expression> is any string constant or string variable.

If the length of <string expression> is less than the number of bytes which were assigned to the specified variable with the FIELD statement, the LSET (Left SET) statement left-justifies the string data in the variable and the RSET (Right SET) statement right-justifies it. The positions following left-justified data and those preceding right-justified data are padded with spaces.

If the length of <string expression> is greater than the number of bytes assigned to the specified variable, excess characters are truncated from the right end of <string expression> when it is moved into the buffer. (This is true for both the LSET and RSET statements.)

Numeric values must be converted to strings before they can be moved into a random file buffer with the LSET or RSET statements. This is done using the MKI$, MKS$, and MKD$ functions described elsewhere in this Chapter, and Chapter 5.

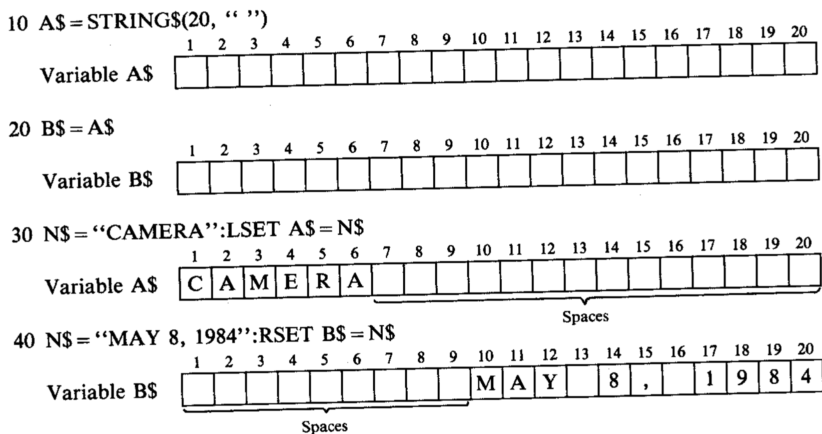
See also

FIELD, GET, OPEN, PUT

ClibPDF - www.fastio.com

Example See Chapter 5 for examples of use of LSET/RSET in a program.

NOTE:
The LSET and RSET statements can also be used to left or right justify a string in a string variable which has not been assigned to a random file buffer. For example, the following program left-justifies character string "CAMERA" in a 20-character field prepared in variable A$ and right-justifies character string "MAY 8, 1984" in variable B$. This procedure can be very useful when formatting data for output.



MENU

Format **MENU**

Purpose Returns BASIC to the start-up menu.

Remarks Executing this command returns the BASIC interpreter to the BASIC menu screen which is displayed at the time of start-up. It also clears all variables and closes any files which are open. This command resets the virtual screen size to 40 columns × 50 lines.

See also **LOGIN, WIDTH**

```
BASIC Verx.x (C) 1983 Microsoft & EPSON
RETURN to run or SPACE to login.
      [illegible]
       P2:                  0 Bytes
       P3:                  0 Bytes
       P4:                  0 Bytes
       P5:                  0 Bytes
          ###### Bytes Free
```


ClibPDF - www.fastio.com

MERGE

Format **MERGE <file descriptor>**

Purpose Merges a program from a disk device or the RS-232C interface with the program in the currently logged in program area.

Remarks Specify the device name, file name, and file name extension under which the file was saved in <file descriptor>. The device name can be omitted if the file is in the currently active drive (the drive which was logged in under CP/M at the time BASIC was started). If the file name extension is omitted, ".BAS" is assumed.

The file being merged must have been saved in ASCII format. Otherwise, a BF error (Bad file mode) will occur.

If any lines of the program being merged have the same numbers as lines of the program in memory, the merged lines will replace the corresponding lines in memory. Thus, a program brought into the BASIC program area with the MERGE statement may be thought of as an overlay which replaces corresponding lines previously included in the program area.

BASIC always returns to the command level following execution of a MERGE command.

See also SAVE

Examples First type in and save the following programs, making sure they are saved in ASCII format by using the ",A" extension as shown below.

```
10 'PROGRAM "MERGE1" TO BE MERGED WITH "MERGE2" THEN "MERGE3"
20 PRINT "HELLO MUM"
30 PRINT
Ok
SAVE "A:MERGE1",A
Ok
```



```
NEW
Ok
30 'PROGRAM "MERGE3" TO BE MERGED WITH "MERGE1"
40 PRINT "GOODBYE DAD"
50 END
SAVE "A:MERGE3",A
Ok
```

Clear the program with the NEW command, and type in the following program.

```
50 'PROGRAM "MERGE 2" WITH WHICH "MERGE1" WILL BE MERGED
60 PRINT "GOODBYE MUM"
70 END
```

Now type

```
MERGE "A:MERGE1"
```

If the program in memory is listed it will be seen to consist of the lines from both programs as follows:

```
10 'PROGRAM "MERGE1" TO BE MERGED WITH "MERGE2" THEN "MERGE3"
20 PRINT "HELLO MUM"
30 PRINT
50 'PROGRAM "MERGE 2" WITH WHICH "MERGE1" WILL BE MERGED
60 PRINT "GOODBYE MUM"
70 END
```

Now type in the following.

```
MERGE "A:MERGE3"
```

List to see that the result is as follows.

```
10 'PROGRAM "MERGE1" TO BE MERGED WITH "MERGE2" THEN "MERGE3"
20 PRINT "HELLO MUM"
30 'PROGRAM "MERGE3" TO BE MERGED WITH "MERGE1"
40 PRINT "GOODBYE DAD"
50 END
60 PRINT "GOODBYE MUM"
70 END
```


ClibPDF - www.fastio.com

MID$

Format As a statement
MID$ (<string expression 1>,n[,m]) = <string expression 2>

As a function
MID$ (X$,J[,K])

Purpose As a statement, replaces a portion of one string with another. As a function, returns the character string from the middle of string expression X$ which consists of the K characters beginning with the Jth character.

Remarks When MID$ is used as a statement, n and m are integer expressions and <string expression 1> and <string expression 2> are string expressions. In this case, MID$ replaces the characters beginning at position n in <string expression 1> with the characters of <string expression 2>. If the m option is specified, the first m characters of <string expression 2> are used in making the replacement; otherwise, all of <string expression 2> is used. However, the number of characters replaced cannot exceed the length of the portion of <string expression 1> which starts with character n.

For example:

```
10 A$="ABCDEFG" : LET B$="wxyz"
20 MID$(A$,3)=B$
30 PRINT A$
```

will give the result "ABwxyzG" for A$. Whereas,

```
10 A$="ABCDEFG" : LET B$="wxyz"
20 MID$(A$,3,2)=B$
30 PRINT A$
```

will give a value of "ABwxEFG".

When MID$ is used as a function, J and K must be integer expressions in the range from 1 to 255. If K is omitted, or there are fewer than K characters to the right of the Jth character,the string returned consists of all characters to the right of the Jth character. If the value specified for J is greater than the number of characters in X$, MID$ returns a null string.

See also **LEFT$, RIGHT$**

Example **INSERT MID$**

```
10 A$ = "Computers  is great!"
20 B$ = "are"
30 PRINT A$
40 MID$(A$,11) = B$
50 PRINT A$
60 PRINT
70 B$ = "super-duper"
80 MID$(A$,15,5) = B$
90 PRINT A$
100 PRINT
110 B$ = MID$(A$,7,12)
120 PRINT B$
130 END
```

```
Computers  is great!
Computers are great!

Computers are super!

ers are supe
```


ClibPDF - www.fastio.com

MKI$/MKS$/MKD$

Format **MKI$(<integer expression>)**
MKS$(<single precision expression>)
MKD$(<double precision expression>)

Purpose Converts numeric values to string values for storage in random access files.

Remarks Numeric values must be converted to string values before they can be placed in a random file buffer by a LSET or RSET statement for storage with a PUT statement. MKI$ converts integers to 2-byte strings, MKS$ converts single precision numbers to 4-byte strings, and MKD$ converts double precision numbers to 8-byte strings.

Unlike the STR$ function, which produces an ASCII string whose characters correspond to the digits of the decimal representation of a number, these functions convert numeric values to characters whose ASCII codes correspond to the binary coded decimal values with which corresponding values are stored in variable memory. In many instances, less disk space is required for storage of numbers which are converted to strings using the MKI$/MKS$/MKD$ functions.

See also **CVI/CVS/CVD**

For examples of the use of these functions, see the section of Chapter 6 dealing with random access files.

```
FILES "A:
MERGE1   .BAS   MERGE3   .BAS
Ok
NAME "A:MERGE1.BAS" AS "A:CHAIN.BAS"
Ok
```

```
FILES "A:
CHAIN    .BAS   MERGE3   .BAS
Ok
```


MOTOR

Format **MOTOR [<switch>]**

Purpose Controls the motor of the external audio cassette.

Remarks This statement turns on or off the external audio cassette's REM (remote) terminal. The REM terminal is turned on if ON is specified for <switch>, and is turned off if OFF is specified. If <switch> is not specified, execution of this statement reverses the setting of the REM terminal.

ClibPDF - www.fastio.com

MOUNT

Format **MOUNT**

Purpose Reads the microcassette tape directory into memory and enables the tape for use.

Remarks A MOUNT command must be executed before data can be written to or read from a microcassette tape. The MOUNT command prepares the tape for access by reading its directory into memory.

Before executing the MOUNT command, any previously mounted tape must be unmounted by executing the REMOVE command.

IO error (Device I/O error) — The drive does not contain a cassette tape.

DR error (Disk read error) — An error occurred while the microcassette tape was being read.

AC error (Tape access error) — The MOUNT command was executed without unmounting the previous tape with the REMOVE command.

See also **REMOVE**
Section 2.4.5

NAME

Format **NAME <old filename> AS <new filename>**

Purpose Changes the name of a disk device file.

Remarks Both <old filename> and <new filename> are specified as a device name, file name, and extension. The device name may be omitted if the file resides on the disk device which is currently active.

The file name specified in <old filename> must be that of a currently existing file, and that specified in <new filename> must be a name which is not assigned to any other file belonging to the applicable disk device. If <old filename> is not the name of an existing file, an NE error (File not found) will occur; if <new filename> is already assigned to an existing file, an FE error (File already exists) will occur. If the file being renamed has a file name extension, that extension must be specified in <old filename>.

If the NAME command is executed with a file which is already open, there is no guarantee that the file will remain intact. Using the CLOSE command will ensure that all files are closed.

This command changes only the name of the specified file; it does not rewrite the file to another area in the storage medium.

See also **FILES**

ClibPDF - www.fastio.com

NEW

Format NEW

Purpose Deletes the program in the currently logged in program area and clears all variables.

Remarks Enter NEW at the command level to clear the memory before starting to enter a new program. BASIC always returns to the command level upon execution of a NEW command.

An FC error (Illegal function call) will occur when this command is executed if program editing has been disabled by executing a TITLE command with the P (protect) option specified.

See also TITLE

Example

```
LOGIN 1
P1:DEMOPROG     23 Bytes
Ok
NEW
Ok
```

```
LOGIN 1
P1:              0 Bytes
Ok
```


OCT$

Format OCT$(X)

Purpose Returns a string which represents the octal value of X.

Remarks The numeric expression specified in the argument is rounded to the nearest integer value before it is evaluated.

A description of using numbers and numeric variables is given in Chapter 2.

Example

```
10 CLS
20 INPUT "What value do you want to convert ";X
30 PRINT
40 PRINT "The octal value of ";X;" is ";OCT$(X)
50 FOR J = 1 TO 3000:NEXT
60 GOTO 10
```

```
What value do you want to convert ? 23
The octal value of  23  is 27
```

```
What value do you want to convert ? 983
The octal value of  983  is 1727
```


ClibPDF - www.fastio.com

ON COM(n) GOSUB...RETURN

Format **ON COM(n) GOSUB [<line number>]...**
RETURN [<line number>]

Purpose Defines the starting point of the communication trap routine to which processing branches when a communication line interrupt is generated.

Remarks This statement defines the starting point of the communication trap routine to which processing branches when a communication port input interrupt is generated. The n in COM(n) indicates the communication port number, and is specified as a number from 0 to 3. <line number> is the first line of the communication trap routine. Communication interrupts are disabled if 0 is specified for <line number> or the parameter is omitted. Processing is returned to the main routine from the communication trap by the RETURN statement. If RETURN is executed by itself, processing resumes from the point at which it was interrupted; if a line number is specified following RETURN, processing resumes with the first statement in that line.

Since only one communication device (COMn:) can be open at a time, there is no possibility of interrupts being generated from two communication ports simultaneously. The COM(n) ON statement must be executed to allow branching to the communication V trap. Communication interrupts are not generated if COM(n) OFF is executed. If COM(n) STOP is executed, no interrupt is generated but data received is saved; an interrupt is then generated the next time COM(n) ON is executed. Once one interrupted has been generated, other are deferred just as if COM(n) STOP were executed. Unless the COM(n) OFF statement is executed in the communication trap routine, generation of deferred interrupts is automatically reenabled upon return to the main routine in the same manner as when COM(n) ON is interrupted. Communication interrupts are not generated except during execution of a BASIC program. Further, all interrupts are automatically disabled if any error occurs. When using the RETURN <line number> statement, remember that the status of any subroutines (GOSUB) or loops (WHILE or FOR/NEXT) being processed remains unchanged when processing branches to the communication trap.

ON ERROR GOTO

Format **ON ERROR GOTO [<line number>]**

Purpose Causes program execution to branch to the first line of an error processing routine when an error occurs.

Remarks Execution of the ON ERROR GOTO statement enables error trapping; that is, it causes execution of a program to branch to a user-written error processing routine beginning at the program line specified in <line number> whenever any error (such as a syntax error) occurs. This error processing routine then evaluates the error and/or directs the course of subsequent processing. For example, it may be written to check for a certain type of error or an error occurring in a certain program line, then to resume execution at a certain point in the program depending on the result.

If subsequent error trapping is to be disabled, execute ON ERROR GOTO 0. If this statement is encountered in an error processing routine, program execution stops and BASIC displays the error message for the error which caused the trap. It is recommended that all error processing routines include an ON ERROR GOTO 0 statement for errors for which are not provided for in the error recovery procedures. If <line number> is not specified, the effect is the same as executing ON ERROR GOTO 0.

See also **ERROR, RESUME, ERL/ERR, Appendix A**

Example See the program example under ERROR.

NOTE:
BASIC always displays error messages and terminates execution for errors which occur in the body of an error processing routine; that is, error trapping is not performed within an error processing routine itself.

ClibPDF - www.fastio.com

ON...GOSUB/ON...GOTO

Format

ON <numeric expression> GOSUB <list of line numbers>
ON <numeric expression> GOTO <list of line numbers>

Purpose

Transfers execution to one of several program lines specified in <list of line numbers> depending on the value returned when <numeric expression> is evaluated.

Remarks

The value of <numeric expression> determines to which of the line numbers listed execution will branch. If the value of <numeric expression> is 1, execution will branch to the first line number in the list; if it is 2, execution will branch to the second line number in the list; and so forth. If the value is a non-integer, the fractional portion is rounded.

An FC error (Illegal function call) will occur if the value of <numeric expression> is negative or greater than 256.

With the ON...GOSUB statement, each program line indicated in <list of line numbers> must be a line of a subroutine.

See also

GOSUB...RETURN, GOTO

Example 1

```
10 CLS
20 INPUT "Type in a number from 5 to 10 ";X
30 Y = X - 4
40 ON Y GOTO 60,80,100,120,140,160
50 END
60 PRINT X;"- 4 = ";Y;" so this is line 60"
70 GOTO 20
80 PRINT X;"- 4 = ";Y;" so this is line 80"
90 GOTO 20
100 PRINT X;"- 4 = ";Y;" so this is line 100"
110 GOTO 20
120 PRINT X;"- 4 = ";Y;" so this is line 120"
130 GOTO 20
140 PRINT X;"- 4 = ";Y;" so this is line 140"
150 GOTO 20
160 PRINT X;"- 4 = ";Y;" so this is line 160"
170 GOTO 20
```



```
Type in a number from 5 to 10 ? 7
 7 - 4 =  3  so this is line 100
Type in a number from 5 to 10 ? 9
 9 - 4 =  5  so this is line 140
```

Example 2

```
10 CLS
20 INPUT "Type in a number from 1 to 5 ";X
30 ON X GOSUB 50,60,70,80,90
40 GOTO 20
50 PRINT "ONE":RETURN
60 PRINT "TWO":RETURN
70 PRINT "THREE":RETURN
80 PRINT "FOUR":RETURN
90 PRINT "FIVE":RETURN
```

```
Type in a number from 1 to 5 ? 2
TWO
Type in a number from 1 to 5 ? 4
FOUR
Type in a number from 1 to 5 ? 3
THREE
```

NOTE:
Only numeric expressions can be used to control branching with the ON...GOSUB and ON...GOTO statements. However, it is possible to derive numeric values from string values by using functions such as ASC and INSTR$. For example, the following sample program derives numeric results for the ON...GOSUB and ON...GOTO statements based on input of string values.

ClibPDF - www.fastio.com

Example 3

```
10 CLS
20 INPUT "Type in a word beginning with A, B or C ";A$
30 X = ASC(A$)
40 Y = X - 64
50 ON Y GOSUB 70,110,150
60 END
70 PRINT "The ASCII code for A is 65, so line 40 subtracts
80 PRINT "64 from this code to give 1, thus causing the"
90 PRINT "subroutine at line 70 to be executed"
100 RETURN
110 PRINT "The ASCII code for B is 66, so line 40 subtracts
120 PRINT "64 from this to give 2, causing the subroutine"
130 PRINT "at line 110 to be executed"
140 RETURN
150 PRINT "The ASCII code for C is 67, and line 40 subtracts
160 PRINT "64 from this giving 3 so that the subroutine on"
170 PRINT "line 150 is executed."
180 RETURN

Type in a word beginning with A, B or C ? Albatross
The ASCII code for A is 65, so line 40 subtracts
64 from this code to give 1, thus causing the
subroutine at line 70 to be executed
Ok
Type in a word beginning with A, B or C ? Carousel
The ASCII code for C is 67, and line 40 subtracts
64 from this giving 3 so that the subroutine on
line 150 is executed.
Ok
```


OPEN

Format

OPEN "<mode>",[#]<file number>,<file descriptor>, [<record length>]

Purpose

The OPEN statement enables input/output access to a disk device file or other device.

Remarks

Disk device files must be OPENed before any data can be input from or output to such files. The OPEN statement allocates a buffer for I/O to the specified file and determines the mode of access in which that buffer will be used. <mode> is a string expression whose first character is one of the following.

O or o .... Specifies the sequential output mode.
I or i .... Specifies the sequential input mode.
R or r .... Specifies the random input/output mode.

Any mode can be specified for a disk device file, but only the "I" or "O" modes can be specified for devices such as the RS-232C interface or printer.

<file number> is an integer expression from 1 to 15 which specifies the number by which the file is to be referenced in I/O statements as long as the file is open. The value of <file number> is limited to the maximum specified in the /F: option if this option is used when BASIC is started up. Since this is 3 in the default mode, it is necessary to ensure the /F: option is specified before a program is run if more than 3 files are required.

<file descriptor> is a string expression which conforms to the rules for naming files (see Chapter 2).

<record length> is an integer expression which, if specified, sets the record length for random access files. If not specified, the record length is set to 128 bytes.

Disk files and files on the RAM disk can be open for sequential input or random access under more than one file number at a time. However, a given sequential access file can only be opened for sequential output under one file number at a time, and such a

ClibPDF - www.fastio.com

file cannot be open in both the sequential input and sequential output modes concurrently.

Microcassette files can only be open for sequential input, random access, or sequential output under one file number at a time. Further, only one microcassette file can be open at any given time.

The RS-232C interface can be open concurrently in the sequential input mode and the sequential output mode, but cannot be opened in the random access mode.

See also Chapters 5 and 6.

Example For programming examples, see Chapters 5 and 6 and the explanation of EOF.

OPTION BASE

Format **OPTION BASE <base number>**

Purpose Declares the minimum value of array subscripts.

Remarks When BASIC is started, the minimum value of array subscripts is set to 0; however, in certain applications it may be more convenient to use variable arrays whose subscripts have a minimum value of 1. Specifying 1 for the value of <base number> in this statement makes it possible to set the minimum subscript base to one.

Once the subscript base has been set by executing this statement, it cannot be reset until a CLEAR statement has been executed; executing a CLEAR statement restores the option base to 0. Further, OPTION BASE 1 cannot be executed if any values have previously been stored in any array variables. A DD error (Duplicate Definition) will occur if the OPTION BASE statement is executed under either of these conditions.

See also DIM

Example

```
10 CLEAR
20 PRINT "Memory free following CLEAR:";FRE(0)
30 OPTION BASE 0
40 DIM A(5,5,5,5)
50 PRINT "Memory free after DIM A(5,5,5,5) with OPTION BASE
0:";FRE(0)
60 CLEAR
70 OPTION BASE 1
80 DIM A(5,5,5,5)
90 PRINT "Memory free after DIM A(5,5,5,5) with OPTION BASE
1:";FRE(0)

run
Memory free following CLEAR: 7324
Memory free after DIM A(5,5,5,5) with OPTION BASE 0: 2125
Memory free after DIM A(5,5,5,5) with OPTION BASE 1: 4809
Ok
```


ClibPDF - www.fastio.com

OPTION COUNTRY

Format **OPTION COUNTRY <string>**

Purpose Selects the international character set.

Remarks This statement selects the international character set which is used for keyboard input/output, display, and output to the printer. The character set selected by this statement is determined by the first character of the <string> parameter as follows.

"U" or "u"...... U.S.A
"F" or "f"...... France
"G" or "g"...... Germany
"E" or "e"...... England
"D" or "d"...... Denmark
"W" or "w"..... Sweden
"I" or "i" ...... Italy
"S" or "s"...... Spain
"N" or "n"...... Norway

After execution of the OPTION COUNTRY statement, the specified character set is used for all output to the LCD screen or printer. The currency symbol output with the PRINT USING statement is also changed to the corresponding symbol in the applicable character set.

The OPTION COUNTRY statement can only be used with the European version of PX-4 BASIC; if it is executed with the American version, a Syntax error will result.

Country selection can be changed by DIP switch setting or CONFIG.COM of CP/M utility program. See Appendix N and Operating Manual.

OPTION CURRENCY

Format **OPTION CURRENCY <string>**

Purpose Changes the currency symbol.

Remarks This statement specifies the character which is output for the currency symbol when two consecutive currency symbol codes (&H5C with the Japanese version, and &H24 with the export versions) are specified at the beginning of the numeric formatting string in the PRINT[#] USING statement.

The character output for the currency symbol is the first character in <string>, and can be any character which is included in the character set.

ClibPDF - www.fastio.com

OUT

Format **OUT <integer expression 1>,<integer expression 2>**

Purpose Used to send data to a machine output port.

Remarks The data to be output is specified in <integer expression 2> and the port to which it is to be output is specified in <integer expression 1>. Both values must be in the range 0 to 255.

See also **INP**

NOTE:
Use of this statement requires sound knowledge of the PX-4 firmware. Incorrect use may corrupt programs or data held in memory, including BASIC itself.

ClibPDF - www.fastio.com

PCOPY

Format **PCOPY <program area no.>**

Purpose Copies the contents of the currently selected program area to another program area.

Remarks This command copies the contents of the currently selected program area to the program area whose number is specified in <program area no.>. The number specified must be in the range from 1 to 5, and must be the number of an area which does not contain a program. It must also be the number of an area other than the currently selected program area.

Programs which have been saved using the protect save function cannot be transferred between program areas with this command.

FC error (Illegal function call) — A number other than 1 to 5 was specified for <program area no.>; the currently selected program area was specified as the destination area; the specified program area was not empty, or; an attempt was made to PCOPY a program saved using the protect save function.

OM error (Out of memory) — The amount of free memory available was not sufficient to allow the program to be copied.

PEEK

Format **PEEK(J)**

Purpose Returns one byte of data from the memory address specified for J as an integer from 0 to 255.

Remarks As the name suggests, PEEK is a function to look at memory locations and return the value of the contents of the location PEEKed. The contents of the location are not changed by inspecting it. For the beginner learning BASIC, PEEK and the allied command POKE (which allows the contents of a location to be changed) are commands which are difficult to understand, because it is not always easy to see the function of the values used. They can be used in a large number of ways. Also the values are very computer dependent. It is often possible to type many BASIC programs into the PX-4 when they have been written for other computers even if the BASIC is another version of MICROSOFT BASIC. When PEEK and POKE commands are used, they are invariably not directly translatable. For example with many computers it is possible to PEEK and POKE the memory reserved for the screen. This is not possible directly with the PX-4.

The integer value specified for J must be in the range from 0 to 65535.

See also **POKE**

Example If location 4 is PEEKed, the number returned will correspond to the drive which is the default drive when returning to CP/M. This is not the default drive for BASIC. If the value returned is "0" then A: is the default drive, if it is "1" then it is drive "B" and so on.

POINT

Format **POINT (<horizontal coordinate>,<vertical coordinate>)**

Purpose Returns the status of the dot at the specified screen coordinates.

Remarks The POINT function returns the status of the dot at the specified graphic coordinates as a function code. If the dot is set (turned on), the value returned is 7; if the dot is reset (turned off), the value returned is 0. If the specified graphic coordinates are outside the screen, the value returned is −1.

OV error (Overflow) — An argument specified was not in the range from −32768 to 32767.

ClibPDF - www.fastio.com

POKE

Format POKE <integer expression 1>,<integer expression 2>

Purpose Writes a byte of data into memory.

Remarks The address into which the data byte is to be written is specified in <integer expression 1> and the value which is to be written into that address is specified in <integer expression 2>. The value specified in <integer expression 2> must be in the range from 0 to 255.

The complement of the POKE statement is the PEEK function, which is used to check the contents of specific addresses in memory. Used together, the POKE statement and PEEK function are useful for accessing memory for data storage, writing machine language programs into memory, and passing arguments and results between BASIC programs and machine language routines.

See also PEEK

Example An example of using BASIC to POKE a machine code routine is described under the CALL command.

In the example under the PEEK command, it was shown how to find out which drive would be the active drive when exiting to CP/M. This can be altered with the BASIC POKE command. If you have any BASIC programs in memory which you want to save, save them first. In direct mode type "POKE 4,1", then type "SYSTEM" and press [RETURN]. You will be transferred to either the system menu or the CP/M command line. If the menu is active, use ESC to return to the CP/M command line. The active drive should be shown as "B>" which will normally be assigned to one of the ROM sockets.

WARNING:
Since this statement changes the contents of memory, the work area used by BASIC may be destroyed if it is used carelessly. This can result in erroneous operation, so be sure to check the memory map to confirm that the address specified is in a usable area.

POS

Format POS(<file no.>)

Purpose Returns the current position of the print head, cursor, or file output buffer pointer.

Remarks When "0" is specified for <file no.>, this function returns the current horizontal position of the cursor. The value returned ranges from 1 to the number of columns in the virtual screen currently being output.

When a number other than "0" is specified for <file no.>, this function returns the current position of the pointer in the output buffer for the specified file. Here, the file must be one which has been opened in the sequential output mode or the random access mode. The value returned for a file opened in the sequential input mode will have no meaning.

When the value specified for <file no.> is other than "0", the value returned by the POS function will be a number in the range from 1 to 255; immediately after the file is opened or a carriage return is output, the value returned is "1".

If the file specified is "LPT0:", this function returns the same value as the LPOS function.

ClibPDF - www.fastio.com

POWER

Format

1) **POWER OFF [,RESUME]**
2) **POWER** | **<duration>** / **CONT** |

Purpose

Allows the power to be turned off by program and controls the auto power-off function.

Remarks

Executing POWER OFF turns off the power in the restart mode. When the power goes off in the restart mode, BASIC is restarted by a hot start when the power is turned back on.

When POWER OFF, RESUME is executed, the power goes off in the continue mode. When the power is then turned back on (by moving the power switch from ON to OFF, and then back ON again), BASIC program execution resumes with the statement following the POWER statement which turned off the power.

When the power goes off in the restart mode, it is turned back on again if the wake-up time set by a WAKE statement is reached before the power is manually turned back on.

When the wake time is reached after the power has gone off in the continue mode, program execution resumes with the statement following the POWER statement which turned off the power.

POWER <duration> specifies the amount of time which will elapse before the auto shut-off function automatically turns off the power when a certain amount of time passes in the direct mode without anything being entered from the keyboard.

Executing POWER CONT disables the auto power-off function. The auto power-off function can later be reenabled by executing POWER <duration>.

FC error (Illegal function call) — The value specified for <counter value> was outside the prescribed range.

PRESET

Format

PRESET [STEP] (X,Y)[,<function code>]

Purpose

Resets (turns off) the dot at the specified graphic screen coordinates.

Remarks

This statement resets the dot at the graphic screen coordinates specified by (X,Y). When STEP is specified, relative coordinates are used.

When <function code> is specified, this statement sets or resets the dot at the specified position in the same manner as the PSET statement. If <function code> is omitted, the specified dot is reset.

After execution of the PRESET statement, the LRP (last reference pointer) is updated to the values specified for (X,Y).

FC error (Illegal function call) — The number specified in one of the statement operands was outside of the prescribed range.

OV error (Overflow) — The number specified in one of the statement operands was outside of the prescribed range.

ClibPDF - www.fastio.com

PRINT

Format **PRINT [<list of expressions>]**

Purpose Outputs data to the LCD screen.

Remarks Executing a PRINT statement without specifying any expressions advances the cursor to the line following that on which it is currently located without displaying anything.

When a <list of expressions> is included, the values of the expressions are output to the display screen. Both numeric and string expressions may be included in the list. The positions in which items are displayed is determined by the delimiting punctuation used to separate items in the list.

Under BASIC, the screen is divided up into zones consisting of 14 spaces each. When items in <list of expressions> are delimited with commas, each succeeding value is displayed starting at the beginning of the following zone. When items are delimited with semicolons, they are displayed immediately following one another. Including one or more spaces between items has the same effect as a semicolon. Other display formats can be obtained by including the TAB, SPACE$, and SPC functions.

If a semicolon or comma is included at the end of the list of expressions, the cursor remains on the current display line and values specified in the next PRINT statement are displayed starting on the same line. If the list of expressions is concluded without a semicolon or comma, the cursor is moved to the beginning of the next line.

If values displayed by the PRINT statement will not fit on one display line, display is continued on the next line.

See also **LPRINT, PRINT USING, SPACE$, SPC, TAB**

ClibPDF - www.fastio.com

Example

```
10 PRINT 123;456;789
20 PRINT 123,456,789
30 PRINT "123";"456";"789"
40 PRINT "ABC",
50 PRINT "DEF"
60 PRINT "ABC";
70 PRINT "DEF"

run
 123  456  789
 123           456           789
123456789
ABC            DEF
ABCDEF
Ok
```

NOTE:
A question mark (?) may be typed in place of the word PRINT when entering the PRINT statement. BASIC automatically converts question marks encountered during statement execution to PRINT statements.

PRINT USING

Format

PRINT USING <format string>;<list of expressions>

Purpose

Displays string data or numbers using a format specified by <format string>.

Remarks

<format string> consists of special characters which determine the size and format of the field in which expressions are displayed. <list of expressions> consists of the string expressions or numeric expressions which are to be displayed. Each expression in <list of expressions> must be delimited from the one following it by a semicolon.

The characters which make up the <format string> differ according to whether the expressions included in <list of expressions> are string expressions or numeric expressions. The characters and their functions are as follows:

Format strings for string expressions

"!"
Specifies that the first character of each string included in <list of expressions> is to be displayed in a 1-character field.

Example 1

```
10 PRINT USING "!";"Aa";"bB";"Dc"

run
AbD
Ok
```

"\n spaces\"
Specifies that 2+n characters of each string in <list of expressions> is to be displayed. Two characters will be displayed if no spaces are included between the backslashes, three characters will be displayed if one space is included between the backslashes, and so on. Extra characters are ignored if the length of any string in <list of expressions> is greater than 2+n. If the length of the field is greater than that of a string, the string is left-justified in the field and padded on the right with spaces.

Example 2

```
10 A$="1234567"
20 B$="ABCDEFG"
30 PRINT USING "\   \";A$;B$
40 PRINT USING "\         \";A$;B$

run
12345ABCDE
1234567    ABCDEFG
Ok
```

"&"
Specifies that strings included in <list of expressions> are to be displayed exactly as they are.

Example 3

```
10 READ A$,B$
20 PRINT USING "&";A$;" ";B$
30 DATA EPSON,PX-4

run
EPSON PX-4
Ok
```

Format strings for numeric expressions

With numeric expressions, the field in which digits are displayed by a PRINT USING statement is determined by a format string consisting of the number sign (#) and a number of other characters. When the format string consists entirely of # signs, the length of the field is determined by that of the format string.

If the number of digits in numbers being displayed is smaller than the number of positions in the field, numbers are right justified in the field. If the number of digits is greater than the number of # signs, a percent sign (%) is displayed in front of the number and all digits are displayed.

ClibPDF - www.fastio.com

Minus signs are displayed in front of negative numbers, but (ordinarily) positive numbers are not preceded by a plus sign.

The following is an example of use of the # sign in the numeric format string of a PRINT USING statement.

Example 4

```
10 PRINT USING "####  ";1;.12;12.6;12345
20 END

run
   1     0     13  %12345
Ok
```

Other special characters which may be included in numeric format strings are as follows:

"."
A decimal point may be included at any point in the format string to indicate the number of positions in the field which are to be used for display of decimal fractions. The position to the left of the decimal point in the field is always filled (with 0 if necessary). Digits to the right of the decimal point are rounded to fit into positions to the right of the decimal point in the field.

```
10 PRINT USING "###.##  ";123;12.34;123.456;.12
20 END

run
123.00   12.34  123.46    0.12
Ok
```

"+"
A plus sign (+) at the beginning or end of the format string causes the sign of the number (plus or minus) to be displayed in front of or behind the number.

Example 5

```
10 PRINT USING "+####";123;-123
20 END

run
 +123 -123
Ok
```


"–"
A minus sign at the end of the format string causes negative numbers to be displayed with a trailing minus sign.

Example 6

```
10 PRINT USING "####-  ";345;-456
20 END

run
 345   456-
Ok
```

"**"
A double asterisk at the beginning of the format string causes leading spaces to be filled with asterisks. The asterisks in the format string also represent two positions in the display field.

Example 7

```
10 PRINT USING "******.##   ";
12.35;123.555;555555.88#
20 END

run
****12.35  ***123.56  555555.88
Ok
```

"$$"
A double dollar sign at the beginning of the format string causes the dollar sign (or other character selected with the OPTION CURRENCY statement) to be displayed immediately to the left of numbers displayed. The dollar signs in the format string also represent two positions in the display field (one of which is used for display of the dollar sign).

Example 8

```
10 PRINT USING "$$####.##   ";
12.35;123.555;555555.88#
20 END

run
   $12.35    $123.56  %$555555.88
Ok
```


ClibPDF - www.fastio.com

"**$"

Specifying **$ at the beginning of the format string combines the effect of the dollar sign and asterisk. Numbers displayed are preceded by a dollar sign, and empty spaces to the left of the dollar sign are filled with asterisks. The symbols * *$ also represent three positions in the display field (one of which is used for display of the dollar sign).

Example 9

```
10 PRINT USING "**$####.##   ";12.35;
123.555;555555.88#
20 END

run
****$12.35   ***$123.56   $555555.88
Ok
```

","

Including a comma to the left of the decimal point in a format string causes commas to be displayed to the left of every third digit to the left of the decimal point. If the format string does not include a decimal point, include the comma at the end of the format string; in this case, numbers are rounded to the nearest integer value for display.

The comma represents the position of an additional position in the display field, and each comma displayed occupies one position.

Example 10

```
10 PRINT USING "#######,.##";555555.88#
20 END

run
 555,555.88
Ok
```


"^^^^"

Four carets (exponentiation operators) at the right end of the format string cause numbers to be displayed in exponential format. The four carets reserve space for display of E+XX.

The decimal point may also be included in the format string at any position desired. Significant digits are left-justified, and the exponent and fixed point constant are adjusted as necessary to allow the number to be displayed in the number of positions in the field.

Unless a leading + or trailing + or − sign is included in the format string, one digit position to the left of the decimal point will be used to display a + or − sign.

Example 11

```
10 PRINT USING "###.##^^^^  ";
   123.45;12.345;1234.5
20 END

run
 12.35E+01    12.35E+00    12.35E+02
Ok
```

"_"

An underscore mark in the format string causes the following character to be output as a literal together with the number.

Example 12

```
10 PRINT USING "###_%";123
20 END

run
123%
Ok
```


ClibPDF - www.fastio.com

Other characters:
If characters other than those described above are placed at the beginning or end of a format string, those characters will be displayed in front of or behind the formatted number. Operation varies from case to case if other characters are included within the format string; however, in general including other characters in the string has the effect of dividing the string up into sections, with formatted numbers displayed in each section together with the delimiting character.

```
10 PRINT USING "##/##/##";12;34;56
20 PRINT USING "(###)";123
30 PRINT USING "<###>";123
40 END
```

```
Ok
run
12/34/56
(123)
<123>
Ok
```

NOTE:
The formatting characters shown above apply to the ASCII character set. If you select a character set other than ASCII with the Option Country statement some of the formatting characters will be output differently as shown below.

Hex.	Dec.	U.S.A	France	Germany	England	Denmark	Sweden	Italy	Spain	Norway
23H	35	#	#	#	£	#	#	#	₧	#
24H	36	$	$	$	$	$	¤	$	$	¤
5CH	92	\	ç	Ö	\	Ø	Ö	\	Ñ	Ø
5EH	94	^	^	^	^	Ü	Ü	^	^	Ü

Example 13

See listing under **OPTION COUNTRY.**

PRINT # /PRINT # USING

Format

PRINT # <file number>,[<list of expressions>]

PRINT # <file number>, USING <format string>;<list of expressions>

Purpose

These statements write data to a sequential output file.

Remarks

The value of <file number> is the number under which the file was opened for output. The specification of <format string> is the same as that described in the explanation of the PRINT USING statement, and the expressions included in <list of expressions> are the numeric expressions which are to be written to the file.

Both of the formats above write values to the disk in display image format; that is, data is written to the disk in exactly the same format as it is displayed on the screen with the PRINT or PRINT USING statements. Therefore, care must be taken to ensure that data is properly delimited when it is written to the file (otherwise, it will not be input properly when the file is read later with the INPUT # or LINE INPUT # statements).

Numeric expressions included in <list of expressions> should be delimited with semicolons. If commas are used, the extra blanks that would be inserted between display fields by a PRINT statement will be written to the disk.

String expressions included in <list of expressions> must be delimited with semicolons; further, a string expression consisting of an explicit delimiter (a comma or carriage return code) should be included between each expression which is to be read back into a separate variable. The reason for this is that the INPUT # statement regards all characters preceding a comma or carriage return as one item. Explicit delimiters can be included using one of the following formats.

PRINT # 1, <string expression>;",";<string expression>...

PRINT # 1, <string expression>;CHR$(13);<string expression>...

ClibPDF - www.fastio.com

If a string which is to be read back into a variable with the INPUT # statement includes commas, significant leading spaces or carriage returns, the corresponding expression in the PRINT # statement must be enclosed between explicit quotation marks CHR$(34). This is done as follows.

PRINT # 1,CHR$(34);"SMITH, JOHN";CHR$(34);CHR$(34);

"SMITH, ROBERT";CHR$(34);...

This would actually be printed to the disk as

"SMITH, JOHN","SMITH, ROBERT".....

When the LINE INPUT # statement is to be used to read items of data back into variables, delimit string expressions in <list of expressions> with CHR$(13) (the carriage return code) as shown in the example above.

See also **INPUT #, LINE INPUT #, WRITE #,** and Chapter 6.

PSET

Format **PSET [STEP] (X,Y)[,<function code>]**

Purpose Sets (turns on) the dot at the specified graphic screen coordinates.

Remarks This statement sets the dot at the graphic screen coordinates specified by (X,Y). When STEP is specified, relative coordinates are used.

<function code> is a number from 0 to 7 which specifies whether the dot at the specified coordinates is set or reset. If 0 is specified, the PSET statement resets (turns off) the specified dot; if 1 to 7 is specified, the specified dot is set (turned on). If omitted, 7 is assumed.

After execution of the PSET statement, the LRP (last reference pointer) is updated to the values specified for (X,Y).

MO error (Missing operand) — A required operand was not specified in the statement.

FC error (Illegal function call) — The number specified in one of the statement operands was outside of the prescribed range.

OV error (Overflow) — The number specified in one of the statement operands was outside of the prescribed range.

ClibPDF - www.fastio.com