SECTION V
Strings

BASIC/3000 allows the programmer to manipulate character strings through the use of string literals, variables, arrays, functions, operators, assignment statements, and input/output statements. Many of the uses of strings are enhancements to statements that have already been described, such as READ and PRINT.

LITERAL STRINGS

A literal string is a sequence of up to 255 characters. Each character is represented internally by a number between zero and 255 as defined in the standard ASCII character set (see Appendix A). Some of these characters have graphic representations (they can be printed—A, B, d, %), while others do not (they are nonprinting—return, linefeed). Both types of characters can be included in a literal string, but each is handled differently.

Form

A literal string consists of a series of graphic characters surrounded by quote marks:

"*character string*"

The quote mark cannot be included as a character in the *character string.*

The quote mark and nonprinting characters can, however, be included in a literal string by using the integer numeric equivalent of the character preceded by an apostrophe:

'*integer*

The *integer* may be in the range 0-255, but it is good practice to restrict this form to nonprinting characters and the quote mark (34). Nonprinting characters can be combined with quoted strings in a literal string.

Explanation

Literal strings can include both upper case and lower case letters. When a literal string is printed, each character value is printed literally on the output device. However, when a program is listed, literal strings are listed with all graphic characters except the quote mark in quotes and non-graphic characters represented in the apostrophe form.

Examples

""	*A null string (a string of zero length)*
"BASIC"	
"B "	
'13'1Ø	*Carriage return, line feed*
'13'1Ø"TRIPLE STRING"'7	*The literal ends with a bell*
"A"'124"B"	*The literal is A vertical line B*
'34	*The quote mark*

The apostrophe literal form can be juxtaposed with another apostrophe literal; quoted strings cannot be juxtaposed with one another.

DIM Statement with Strings

Literal strings can be contained in string variables, simple or subscripted. Simple string variables greater than one character in length and every array string variable must be dimensioned in a DIM statement. The purpose of the DIM statement is to reserve storage for strings and arrays and to establish their names and maximum size.

Form

The DIM statement consists of the word DIM followed by a list of variable and array definitions separated by commas.

DIM variable(string size),variable(string size), . . .

where *variable* is the name of a simple string variable specified as a letter followed by a $ or a letter and a digit followed by a $. The *string size* is an integer constant that specifies the maximum number of characters the string can contain.

DIM variable(array size,string size),variable(string size), . . .

The *array size* specifies the total number of elements in the array; the *string size* specifies the maximum number of characters in each element. Only one-dimensional string arrays are allowed. Both array size and string size are integers.

If more than one *variable* is included in a single DIM statement, they must be separated by commas. Simple string and string array variables may be dimensioned in the same DIM statement.

Explanation

If a string variable does not appear in a DIM statement then it is implicitly defined as a one-character simple variable. String arrays must be declared in DIM; there is no implicit size for string arrays as there is for numeric arrays. String variables and elements of string arrays are initialized to the null string.

The DIM statement can also be used to declare numeric arrays in the same or a different statement (see Section III).

Example

```
   10 DIM A$[19],B$[5,12],C5$[10,12]
   20 A$="TITLE OF SECTION IS"
   30 MAT READ B$
   40 MAT READ C5$
   50 DATA "ARRAYS","STRINGS","MESSAGES","FILES","INPUT/OUTPUT"
   60 DATA "A1","A2","A3","A4","A5","B1","B2","B3","B4","B5"
   70 PRINT A$,C5$[5],B$[5]
>RUN
TITLE OF SECTION IS         A5                 INPUT/OUTPUT
```

The entire simple string variable A$ is printed, followed by the 5th element of the string array C5$ and the 5th element of the string array B$.

REDIM Statement with Strings

The purpose of the REDIM statement is to dynamically vary the number of elements in a string array, but not the string size itself.

Form

The form of the REDIM statement is the word REDIM followed by a list of previously dimensioned string array variables, each one followed by a new size specification in parentheses.

REDIM variable(new array size)

The *new array size* must be an integer or an integer expression that results in a value between 1 and the previously defined maximum size of the array. Only the array size may be redimensioned; the string size may not be changed. Simple string variables may not be redimensioned. The *variable* must be a string array that has been dimensioned, either implicitly or in a DIM statement.

Explanation

REDIM changes the size of a one-dimensional string array, but cannot increase the original size. The array must have been previously dimensioned with DIM. Unlike DIM, REDIM is an executable statement and its position in a program has meaning. The REDIM statement can also be used to change the dimensions of numeric arrays in the same or a different statement (see Section III).

Example

The number of elements in the string arrays C5$ and B$ is reduced in a REDIM statement in line 50. Then the values read into the 3rd element of C5$ and the 2nd element of B$ are printed following the value of the simple string A$. The maximum length of the individual elements in C5$ and B$ is not affected by REDIM; these lengths remain as specified in the original DIM statement.

```
  10 DIM A$[20],B$[5,12],C5$[10,2]
  20 A$="TITLE OF SECTION IS"
  30 DATA "ARRAYS","STRINGS","MESSAGES","FILES","INPUT/OUTPUT"
  40 DATA "A1","A2","A3","A4","A5","B1","B2","B3","B4","B5"
  50 REDIM B$[3],C5$[5]
  60 MAT READ B$
  70 RESTORE 40
  80 MAT READ C5$
  90 PRINT A$,C5$[3],B$[2]
>RUN
TITLE OF SECTION IS          A3                 STRINGS
```

String Variable

A string variable (simple or subscripted) is used to hold a string literal. The declared size of a string variable is called its physical length. The maximum length of any string variable is 255 characters. A string variable not mentioned in a DIM statement is a simple variable one character in length.

During execution, each string variable contains strings whose length cannot exceed the variable's physical size. This dynamic length is called the logical length of the variable and is initialized to zero (i.e., the null string) at the beginning of program execution.

Form

A simple string variable is referenced by its name and an optional *substring designator* in parentheses.

string name

string name(first character)

string name(first character,last character)

string name(first character;number of characters)

The *string name* is a letter followed by a $ or a letter and a digit followed by a $. The string name may be followed by a *substring designator* in parentheses.

The *substring designator* consists of one or two numeric expressions, separated by a comma or semicolon. The first expression always specifies the *first character* position of the substring. The ending character is determined by the second expression.

If the two expressions are separated by a comma, the second expression specifies the *last character position*; if they are separated by a semicolon it specifies the *number of characters.* If there is only one expression, the ending character position is the last character of the string.

A string array variable is referenced by the *string name* followed, in parentheses, by a *subscript* and an optional *substring designator* separated by a comma.

string name(subscript)

string name(subscript,first character)

string name(subscript,first character,last character)

string name(subscript,first character;number of characters)

The *subscript* is an integer expression that specifies the element of the array to be selected. Since a string array may have only one dimension, there may be only one subscript value.

The *substring designator* and the *string name* are specified in the same way for string array variables as for simple string variables.

Unlike numeric array variables, a string array variable must not have the same name as a simple string variable.

Explanation

Any string variable, simple or subscripted, can be qualified by a substring designator, which is used to select a part of the string to be extracted.

If the substring is specified by a single expression, the substring equals the rest of the string taken from the position indicated by the expression.

If two expressions are separated by a comma, the substring consists of the characters from the position specified by the first expression to the position specified by the second expression. (Note: the second expression can be one less than the first; this specifies the null string).

If two expressions are separated by a semicolon, the substring consists of the characters in the string variable starting at the position indicated by the first expression and taking the number of consecutive characters specified by the second expression. In this case, the expression may evaluate to zero, giving a null string.

If A$ is a simple variable:

A$(3,5)	is the 3rd through the 5th character of the string.
A$(3;3)	is also the 3rd through the 5th character of the string since, in this case, the second expression follows a semicolon indicating the number of characters rather than the last character.
A$(3;0)	is the null string.
A$(3,2)	is also the null string.
A$	every character in the string is selected.

If B$ is an array variable:

B$(3)	is the entire 3rd string in the string array.
B$(2,3,5)	is the 3rd through 5th characters in the second string of the string array.
B$(2,3;3)	is also the 3rd through 5th characters in the second string; the substring starts at the 3rd character and contains 3 characters.

A string array variable must always be subscripted except in MAT statements (see String MAT Operations, this section).

The subscript and substring designator expressions may be any integer expressions. Suppose the variables I and J are used, with I equal to 5 and J equal to 10:

C$(I)	is the 5th character to the end of the string if C$ is a simple string variable; it is the entire 5th string or element if C$ is a string array variable.
C$(I,J)	is the 5th through 10th character if C$ is a simple string variable; it is the 10th character to the end of the string of the 5th string if C$ is a string array variable.

If a substring extends beyond the logical length of a string variable, it is filled out to the specified size with blanks.

Examples

```
    10 DIM A$[10]
    20 A$="ABCDEFGHIJ"
    30 PRINT "STRING A$=";A$
    40 PRINT "SUBSTRING A$(5)=";A$[5]
    50 PRINT "SUBSTRING A$(2;5)=";A$[2;5]
    60 PRINT "SUBSTRING A$(2,5)=";A$[2,5]
    70 PRINT A$[2,1];"=NULL STRING"
>RUN
STRING A$=ABCDEFGHIJ
SUBSTRING A$(5)=EFGHIJ
SUBSTRING A$(2;5)=BCDEF
SUBSTRING A$(2,5)=BCDE
=NULL STRING
```

In the example above, note the difference between A$(2;5) and A$(2,5). In the example below, each array element is a two character string.

```
    10 DIM B$[10,2]
    20 REM   B$ IS A STRING ARRAY
    30 MAT READ B$
    40 DATA "A1","B2","C3","D4","E5","F6","G7","H8","I9","J0"
    50 PRINT "ARRAY ELEMENT B$(2)=";B$[2]
    60 PRINT "ARRAY ELEMENT B$(2,2)=";B$[2,2]
    70 PRINT "ARRAY ELEMENT B$(2,1;1)=";B$[2,1;1]
>RUN
ARRAY ELEMENT B$(2)=B2
ARRAY ELEMENT B$(2,2)=2
ARRAY ELEMENT B$(2,1;1)=B
```

String Expressions

String expressions consist of one or more source strings (literal strings, string variables, string valued functions) combined from left to right with the concatenate operator (+) to form a single new string value. String expressions can be assigned to string variables or compared with other string expressions to form a numeric expression.

Form

The form is a list of source strings separated by "+"

string

string + *string*. . .

Each source *string* can be either a literal string, a string variable, or a string function.

Explanation

A source string is any entity from which a string value is extracted. The value of the source string is as defined under "String Literals," "String Variables," and "String Functions." An example of a literal string is "BASIC" or '10; of a string variable is A$, C5$(2), B$(2,3), B$(2;2), or A1$(5,3,10); of a string function is CHR$(208).

The "+" character, when used between two source strings, is the concatenate operator. The concatenation of two strings produces a temporary string whose characters are those of the first string immediately followed by those of the second. This temporary string can be used in further concatenation operations, in string comparisons, or it can be assigned to a string variable.

The maximum length of any temporary string is 255 characters. The original operands are unaffected by concatenation.

Legal string expressions:

A$ + B$(2) + '93 + '10'23"ABCD" + C5$(4,3;5)

"BASIC" + C5$(2)

"BASIC"

C5$(2)

Example

```
  10 DIM A$[5],B$[10,10]
  20 LET A$="CON",B$[2]="CATENATION"
  30 PRINT A$+B$[2,1;7]+"E"
>RUN
CONCATENATE
```

String Assignment

The assignment operator (=) can be used to assign a string value (defined by a string expression) to one or more string variables (or substrings of string variables). Several different assignments can appear in one LET statement.

Form

Some forms of LET are

> *LET variable=expression*
>
> *LET variable=variable=. . .=variable=expression*
>
> *LET variable=expression,variable=variable=expression,. . .*

The word LET is entirely optional and can be left off. The *variable* is an entire string variable (simple or subscripted) or part of a string variable (indicated by a substring designator) into which a string value is to be copied. If several variables are separated by equals (=) each is assigned a copy of the value. Numeric assignments as described in Section II can be mixed with string assignments in the same LET statement.

Explanation

The execution of a LET statement proceeds as follows. The subscripts of variables to be assigned values are evaluated from left to right. The expression is then evaluated and assigned to the variables. The same expression is assigned to each variable from left to right. The manner in which each assignment occurs depends upon the number of substring subscripts specified for the destination variable.

If there is no substring designator, the entire variable is replaced by the string value. If the new value will fit entirely into the variable, the logical length of the variable is set to the length of the new value. If the variable is too small, the value is truncated on the right and the logical length of the string is made equal to the physical length.

If there is one substring subscript, this specifies the starting position for the assignment. The entire string value is copied into the variable starting with the indicated position and continuing to the physical end of the variable or the end of the string value, whichever comes first. The part of the variable preceding the subscript is unchanged. The starting subscript must be no more than one greater than the current *logical* length of the variable (i.e., there can be no undefined character positions in the middle of a string variable). If the variable is too small, the value is truncated on the right.

If two substring subscripts are specified, they define a field within the variable into which the string value is stored. If necessary, the value will be truncated on the right or padded out with blanks to fit exactly in the substring specified. The substring for the destination must not extend beyond the physical length of the string variable and all previously mentioned rules must be followed also. The new logical length of the variable is the *larger* of the old logical length or the last position of the substring. Any characters from the old value to the left or right of the substring are unchanged.

Example

```
  10 DIM A$[10]
  20 LET A$="1234567890"
  30 PRINT A$
  40 LET A$[5]="ABCDEF"
  50 PRINT A$
  60 LET A$[7;3]="1234"
  70 PRINT A$
  80 A$[6,8]="X"
  90 PRINT A$
 100 A$=A$[1,4]+"567890"
 110 PRINT A$
>RUN
1234567890
1234ABCDEF
1234AB123F
1234AX  3F
1234567890
```

Note that the literal "1234" in line 60 is truncated to fit in substring A$(7;3).

In line 80, substring A$(6,8) is blank filled since "X" is only one character. The final value of A$ is the same as its original value assigned in line 20.

The example below illustrates variations on assignments to substrings of array elements:

```
  10 DIM A$[3,5]
  20 A$[1]=A$[2]=A$[3]="ABCDE"
  30 LET A$[1,3]=A$[2]
  40 PRINT A$[1],A$[2],A$[3]
  50 LET A$[2,4,5]=A$[3]
  60 PRINT A$[1],A$[2],A$[3]
  70 LET A$[3,2;2]=A$[2]=A$[1,1,1]
  80 PRINT A$[1],A$[2],A$[3]
>RUN
ABABC          ABCDE          ABCDE
ABABC          ABCAB          ABCDE
ABABC          A              AA DE
```

String-Related Functions

There are a number of predefined functions in BASIC/3000 that accept string values as parameters and/or return a string value as their result. (User-defined string functions are described in Section VI.)

CHR$ Function

CHR$(integer expression)

where integer expression given a value in the range 0 to 255 inclusive. The value of CHR$ is the string character that corresponds to the value of the expression in the standard character set (see Appendix A). For example,

```
  10 PRINT CHR$(65)
>RUN
A
```

NUM Function

NUM(string expression)

NUM returns the numeric value of the first character of the string expression according to the standard character code in Appendix A. For example,

```
  10 PRINT NUM("A")
>RUN
 65
```

LEN Function

LEN returns the logical length of the string expression. For example,

```
  10 DIM A$[20]
  20 A$="ABCD"
  30 PRINT LEN(A$)
>RUN
 4
```

POS Function

POS(stringA,stringB)

where *stringA* and *stringB* are any string expressions; POS returns the smallest integer that represents the starting position of a substring in *stringA* that exactly equals *stringB.* If *stringB* is not a substring of *stringA*, then POS equals zero. For example.

```
  10 PRINT POS("12ABC34","ABC")
>RUN
 3
```

WRD Function

WRD(stringA,stringB)

where *stringA* and *stringB* are any string expressions. WRD returns the smallest integer that represents the starting position of a substring in *stringA* that exactly equals the value of *stringB* and is neither immediately preceded nor immediately followed by a letter. If there is no such substring, WRD equals zero. For example,

NOTE: This function is not valid for non-Roman alphabetical character strings. In particular, WRD Function cannot be used for alphabetical characters (such as katakana characters) whose decimal equivalents are greater than 127 in the ASCII collating sequence.

```
  10 PRINT WRD("STRING A$ EQUALS X$","EQUALS")
  20 PRINT WRD("12ABC34","ABC")
  30 PRINT WRD("12ABC34","BC")
>RUN
 11
 3
 0
```

UPS$ Function

UPS$(string expression)

UPS$ returns a string value equivalent to *string* with all lower case Roman letters upshifted. UPS$("abcd%12") is "ABCD%12".

DEB$ Function

DEB$(string expression)

DEB$ returns a string equal to the specified string expression, but with all leading and trailing blanks removed. Embedded blanks remain. For example,

```
  10 PRINT DEB$("     A B      ")
>RUN
A B
```

ROW Function

ROW(string array)

where *string array* is any string array name without subscripts. ROW returns the number of elements in the array since a string array is always one-dimensional. For example,

```
  1Ø DIM B$[15,5]
  2Ø PRINT ROW(B$)
  3Ø REDIM B$[1Ø]
  4Ø PRINT ROW(B$)
>RUN
 15
 1Ø
```

COL Function

COL(string array)

where *string array* is any string array name. COL returns the number of columns in the array. Since string arrays are always one-dimensional, COL always returns a value of 1 for string arrays. For example,

```
  1Ø DIM B$[15,6]
  2Ø PRINT COL(B$)
>RUN
 1
```

DAT Function

DAT$(x,y)

where *x* and *y* are integer expressions which specify the first and last character positions, respectively, of a substring within a full string which defines the current date and time. The full date/time string is structured as follows:

Char 1-3:	Day of the week (SUN, MON, TUE, WED, THU, FRI, SAT)
4-5:	Comma and blank
6-8:	Month of the year (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC)
9:	Blank
10-11:	Day of the month (1 to 31)
12-13:	Comma and blank
14-17:	Year
18-19:	Comma and blank

20-21:	Hour (1 to 12)
22:	Colon
23-24:	Minute (0 to 59)
25:	Blank
26-27:	AM or PM

For example, to print the entire date/time string:

```
  10 PRINT DAT$(1,27)
>RUN
WED, MAY 30, 1973,  2:45 PM
```

To print only the date:

```
  10 PRINT DAT$(6,17)
>RUN
MAY 30, 1973
```

To print only the time:

```
  10 PRINT DAT$(20,27)
>RUN
 2:46 PM
```

Comparing Strings

String expressions can be compared with relational operators to produce a result of true (numeric 1) if the relation holds or false (numeric 0) if the relation does not hold. The relational operators are:

=	Equal
<>	Not equal
<	Less Than
>	Greater Than
<=	Less Than or Equal
>=	Greater Than or Equal

Two strings are equal only if they have the same logical length and each character matches. A string is less than another if its first character that does not match the other is numerically less (according to the standard character code in Appendix A) or it is an initial proper subset of the other (e.g., "AB" < "ABC" but "BA" > "ABC").

A string comparison can appear within a numeric expression, since the result is a number. The string relational operators have the same position in the hierarchy of operators as do the numeric relations. For example, these are string comparisons:

A$=B$

A$=B$ OR C$>=D$

(A$<>"BOB") + 5

See Section II for the meaning and hierarchy of relational operators.

A common use of string comparisons is in IF statements.

Examples

```
  10 DIM A$[10],B$[10]
  20 READ A$,B$
  30 IF A$<B$ THEN PRINT A$;"<";B$
  40 ELSE DO
  50   IF A$=B$ THEN PRINT A$;"=";B$
  60   ELSE PRINT A$;">";B$
  70 DOEND
  80 GOTO 20
  90 DATA "ABC","ABCD","ABC","B"
 100 DATA "ABC","ABC","C",""
>RUN
ABC<ABCD
ABC<B
ABC=ABC
C>
OUT OF DATA IN LINE 20
```

String Input and Output

Values may be read from DATA statements with the READ statement, or they may be input by the user with INPUT, ENTER, or LINPUT statements. They are output with PRINT statements.

READING STRINGS

The READ, DATA, and RESTORE statements can be used with string variables that are simple or subscripted, with or without substrings. The string variable is listed in the READ statement and a corresponding string constant must appear in the DATA statement. A RESTORE statement can be used if the DATA statement is to be read again by a subsequent READ statement. For a full description of READ/DATA/RESTORE statements, see Section II.

String variables can be mixed with numeric variables in READ, but the corresponding constant for each variable must match—string constant for string variable and number for numeric variable. The string constant is assigned to the variable according to the rules defined in String Assignment, this section.

Strings can also be read from files as described in Section VIII, and string arrays can be read with MAT READ as described in Section III.

Examples

```
    10 DIM A$[20],B$[20]
    20 DATA "BOB",'10"JONES"
    30 READ A$[1,3]
    40 READ A$[4;6]
    45 REM..A$="BOB"'10"JONES"
    50 LET B$="HI"
    60 PRINT B$,A$
>RUN
HI             BOB
                  JONES
```

When the PRINT statement is executed, the character for linefeed ('10) is not printed but causes a linefeed.

INPUTTING STRINGS

The INPUT statement can be used to assign string constants to string variables from the terminal.

The rules for entering strings from the terminal are

1. The apostrophe form of a string literal (e.g., '40) is not allowed.
2. All strings must have quote marks around them except the last one entered on each line.
3. If the last string on a line does not begin with a quote, it starts with the first non-blank character (i.e., it can have no leading blanks) and ends with the last non-blank character before the carriage return (i.e., it can have no trailing blanks).

The rules used to assign the value to the variable are those described under "String Assignment."

Examples

```
   10 DIM A$[16],B$[2,5],C$[40]
   20 INPUT A$,B$[1],B$[2],C$
   30 PRINT A$;B$[1];B$[2];C$
>RUN
?"THE VALUE OF B$=","1234 "," 2X5 ", X5=ABC
THE VALUE OF B$=1234  2X5 X5=ABC
```

Note that the last value input is not quoted. Usually only one string per line is input, in which case quotes are not required.

ENTERING STRINGS

The ENTER statement allows one string variable to be assigned a value from the terminal with control over the input operation. See Section II for how this control is exercised.

All characters typed in (including quotes) are assigned to the variable using the rules defined under String Assignment. ENTER does not prompt for input, and does not provide a linefeed after the user's carriage return.

Examples

```
   10 DIM A$[30],B$[10]
   20 PRINT "YOU HAVE 30 SECONDS TO ENTER 30 CHARACTERS"
   30 ENTER 30,T,A$
   40 PRINT '10,T,A$
   50 PRINT "YOU HAVE 15 SECONDS TO ENTER 10 CHARACTERS"
   60 ENTER 15,X,B$
   70 PRINT '10,X,B$
>RUN
YOU HAVE 30 SECONDS TO ENTER 30 CHARACTERS
LET A$ = "VALUE OF A$" , 012345
 18.59          LET A$ = "VALUE OF A$" , 01234
YOU HAVE 15 SECONDS TO ENTER 10 CHARACTERS
A$=LEN(B$)
 6.44           A$=LEN(B$)
```

A common use of ENTER is for testing:

```
   10 DIM A$[30],B$[10]
   20 PRINT "WHO WAS THE FIRST PRESIDENT OF THE U.S.?"
   30 ENTER 30,T,A$
   40 IF A$="GEORGE WASHINGTON" THEN GOTO 80
   50 PRINT '10"SORRY,TRY AGAIN"
   60 ENTER 20,T,A$
   70 IF A$<>"GEORGE WASHINGTON" THEN GOTO 140
   80 PRINT '10"CORRECT,YOU TOOK";T;"SECONDS TO ANSWER"
   90 PRINT "WHAT WAS HIS WIFE'S NAME?"
  100 ENTER 15,X,B$
  110 IF B$<>"MARTHA" THEN GOTO 160
  120 PRINT '10"CORRECT,YOU TOOK";X;"SECONDS TO ANSWER"
  130 END
  140 PRINT '10"SORRY,THE CORRECT ANSWER IS GEORGE WASHINGTON"
  150 END
  160 PRINT '10"SORRY, THE CORRECT ANSWER IS MARTHA"
>RUN
WHO WAS THE FIRST PRESIDENT OF THE U.S.?
FRANKLIN D. ROOSEVELT
SORRY.TRY AGAIN
GEORGE WASHINGTON
CORRECT,YOU TOOK 7.97       SECONDS TO ANSWER
WHAT WAS HIS WIFE'S NAME?
MARTHA
CORRECT,YOU TOOK 4.13       SECONDS TO ANSWER
```

PRINTING STRINGS

Any string expression can be output to the list device (e.g. the terminal) using the PRINT statement. The size of the output field is the number of printing characters in the string value. If the string expression is preceded by a comma, it is printed starting in the next division. Each print line is divided into consecutive divisions. For example, a terminal with default print length of 72 characters will have 4 divisions of 15 characters each and one of 12 (see Print Statement, Section II). If the string expression is preceded by a semicolon, it is printed immediately following the preceding output. The semicolon between items in the print list need not appear if the first item ends with a quoted string, and/or the second item begins with one; in this case a semicolon is inserted automatically.

Strings can be output to the terminal with special formats through the PRINT USING statement (see Section IX, Formatted Output). Strings can be output to files as described in Section VIII, and through the PRINT # USING statement (see Section IX). In addition, the MARGIN statement can be used to change the length of the print line for an ASCII file (see Section VIII).

Examples

```
 10 DIM C$[10],N5$[3,5]
 20 LET C$="XK9-753-20",A=2.5,B=1E-19,N5$[1]="ABCDE"
 30 PRINT A,B,C$
 40 PRINT "BOB"+C$,N5$[1]
 50 PRINT C$+"BOB";N5$[1]
 60 PRINT '10'34"LINE"'34'10'13;"-1"
>RUN
 2.5            1.00000E-19    XK9-753-20
BOBXK9-753-20  ABCDE
XK9-753-20BOBABCDE

"LINE"
-1
```

In the first expression in line 60, the '10 (linefeed) causes a linefeed, the '13 (carriage return) causes a carriage return when the line is printed. The '34 (quote) causes a quote to be printed. The actual quote (") before and after the string LINE in the PRINT statement is not printed.

LINPUT Statement

The LINPUT statement accepts all the characters that a user types in at the terminal and assigns them as a string to a specified string variable.

Form

LINPUT string variable

where the *string variable* is the destination of the input. The variable may be simple or subscripted, or it may be a substring.

Explanation

All characters are accepted including quotes and blanks. Input is terminated by a carriage return. No prompt character is printed.

Examples

```
  10 DIM A$[20]
  20 PRINT "TYPE 20 CHARACTERS:"
  30 LINPUT A$
  40 PRINT A$
  50 PRINT "TYPE 5 CHARACTERS:"
  60 LINPUT A$[10;5]
  70 PRINT A$[10;5]
>RUN
TYPE 20 CHARACTERS:
"ANY CHARACTERS" O.K.
"ANY CHARACTERS" O.K
TYPE 5 CHARACTERS:
&"+ "
&"+ "
```

Because more than 20 characters (the size of A$) were input by the user, the final period in the first input line is truncated. In the second input, quotes are entered as part of the string.

String Array Operations

The String MAT Initialize statement sets every element of a string array to the null string. In addition, a new dimension can be specified for the array, as in REDIM. The form for string array initialization is

MAT string array = NUL$

MAT string array = NUL$ (integer expression)

Each element of the *string array* is set to the null string. If the optional *integer expression* is specified, the number of elements in the array is changed to the number of elements specified by the expression. The element size is not changed.

The String MAT Copy statement copies all of the elements of one string array into another. String elements are truncated if necessary. The form for string array copying is

MAT string arrayA = string arrayB

This causes *string arrayB* to be copied into *string arrayA. String arrayA* is redimensioned to the size of *string arrayB.*

String arrays can be read, input, and printed with the MAT READ, MAT INPUT, and MAT PRINT statements as described for numeric arrays in Section III.

String arrays can also be printed according to a specified format with the MAT PRINT USING statement (see Section IX).

Examples

```
10 DIM A$[10,5],B$[20,5]
20 MAT A$=NUL$
30 MAT B$=A$
```

Each element in string array A$ is set to the null value, then the first ten elements of string array B$ are set to null values. B$ is redimensioned to the size of A$.

In the example below, B$ is redimensioned and each element is set to the null string:

```
10 DIM B$[20,5]
20 MAT B$=NUL$(10)
```

Convert Statement

The CONVERT statement is used to convert a numeric value to a string of characters that represent the number, or vice versa. In the first case, the conversion is identical to that used when listing. The destination string variable should be long enough to contain the result, as in "String Assignment"; if not, the result is truncated. In the second case, the string expression must represent a valid numeric constant. An error in this case terminates the program, unless a label is specified in the CONVERT, in which case, control transfers to that label.

Form

CONVERT numeric expression TO string variable

CONVERT string expression TO numeric variable [,label]

Explanation

The string variable must be long enough to contain the converted numeric expression; if not, the converted expression is truncated. No blanks appear in the string. If the string expression in the second form does not represent a valid numeric constant, control transfers to the optional statement label. If the label is omitted and the string expression is not convertible, the program terminates.

Examples

```
  10 A=10,B=15
  20 DIM A1$[5]
  30 CONVERT A+B TO A1$
  40 PRINT A1$
  50 CONVERT A1$ TO X,100
  60 PRINT X
  70 END
 100 PRINT "ERROR IN CONVERSION FROM STRING"
>RUN
25
 25
```

Since A1$ contains the converted numeric constant, 25, there is no error in line 50. If A1$ contained a value that could not be converted, the message in line 100 would be printed. For instance, if line 50 is changed as follows:

```
>50 CONVERT "A1" TO X,100
>RUN
25
ERROR IN CONVERSION FROM STRING
```