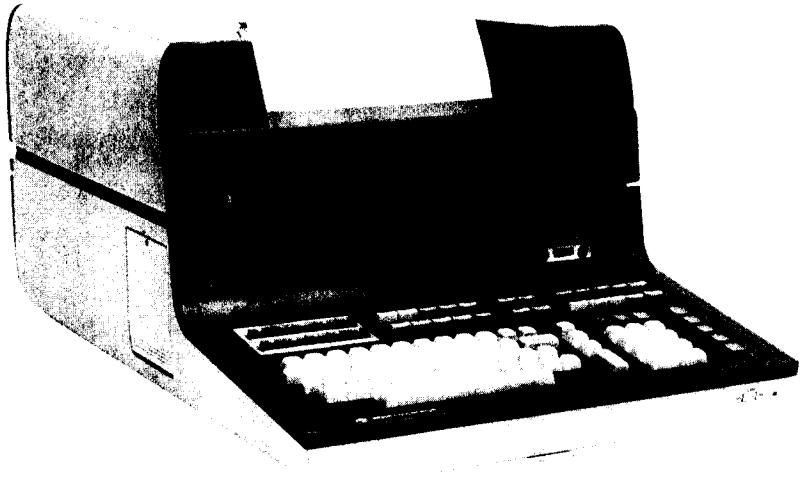
OPERATING MANUAL

STRING VARIABLES ROM
11274B & OPTION 274



9830A CALCULATOR SHOWN WITH 9866A PRINTER

HEWLETT–PACKARD CALCULATOR PRODUCTS DIVISION
P.O. Box 301, Loveland, Colorado 80537, Tel. (303) 667-5000
(For World-wide Sales and Service Offices see rear of manual.)

TABLE OF CONTENTS

PREFACE

The String Variables Read-Only-Memory (ROM) can be purchased as an accessory plug-in block or as an internal modification to the calculator.

The Plug-in Version:

The 11274B String Variables ROM block is installable by the user. It plugs into any of the five slots behind the ROM door on the left side of the calculator.

The Calculator Modification:

The Option 274 String Variables ROM must be installed by qualified HP personnel. When it is installed, a decal showing the option number (Option 274) is attached to the inside of the ROM door.

Should you wish to add the option after you have received your calculator, please order accessory number HP 11274F from the sales office nearest to you (see the back of this manual). The Option 274 will then be installed for you by our field personnel.

Once either version of the ROM (the plug-in block or the internal modification) has been installed, the operation is identical. Therefore, this manual makes no further distinction between the two types of ROM.

Chapter 1

INTRODUCTORY DESCRIPTION

The String Variables Read-Only-Memory (the String ROM) provides additional capabilities to the Model 30, permitting the calculator to process strings (non-numerical data).

With the String ROM, the calculator can use strings of alphanumeric data, instead of just numeric data, in input and output operations; can perform character-by-character comparisons of such strings; and can use the numeric portions as variables in arithmetic calculations. Here are a few applications:

—Conversational Programming

A two-way conversation, in English, between the calculator and the user can be simulated in a program. Replies such as "YES" and "NO" can be accepted by the calculator instead of the numerical codes often used in response to data requests.*

—Text Editing

Variable information such as names and addresses can be inserted into a form letter to be printed on a calculator controlled typewriter. Also, a body of text such as a manuscript can be edited. For example, certain characters or character groups can be located within the text and changed or deleted, or each line of text could be limited to a certain length.

—Language Interpreters

A BASIC program written to teach the calculator another language is called an interpreter. You can write a language interpreter so that the calculator can recognize and act upon statements like "FIND AVERAGE WEIGHT" or "WHAT IS THE TOTAL COST?". For example, a carpet salesman might find an interpreter very useful if it could recognize terms such as "PRICE PER SQ. YD.", "TOTAL AREA" and "TOTAL PRICE AT $6.00 PER SQ. YD.".

*BASIC statements without strings:

```
10 DISP "DO YOU WISH TO CONTINUE";
20 DISP "(1 FOR YES, 0 FOR NO)"
30 INPUT A
40 IF A=1 THEN 100
```

BASIC statements with strings:

```
10 DISP "DO YOU WISH TO CONTINUE";
20 INPUT A$
30 IF A$="YES" THEN 100
```

EQUIPMENT SUPPLIED

An Operating Manual, -hp- Part Number 09830-90002, is supplied with each String Variables ROM.

INSPECTION PROCEDURE

Refer to Appendix A in the 9830A Calculator Operating and Programming Manual for the procedures used to verify the operation of ROM's.

OTHER REQUIREMENTS

It is assumed that you are already familiar with BASIC programming and with the operating procedures for the HP 9830A Calculator.

Chapter 2

CHARACTERISTICS OF STRINGS

A string is an array, named in a DIM or a COM statement, which can contain alphabetic characters and symbols as well as numerics. A one-dimensional array of up to 255 characters or the null string (no characters) can be used. For example:

"ABCDEF"
"12345"
"A82X?"
""

Strings can be assigned values, and String Functions can be executed, either from the keyboard or from a program.

STRING NAMES

A string name consists of a single letter (A through Z) followed by the dollar sign ($), such as A$, B$, Z$.

STRING DEFINITION AND STORAGE SPACE

To reserve storage space for strings longer than 1 character, a DIM or a COM statement is used. The number of characters specified for a string, the array size, must be an integer from 1 to 255, as:

```
10 DIM A$[100],B$[20]
```

Until a string is defined in a DIM statement it is assumed to have a length of 1. (All strings must be initialized in the programming mode before being used in the calculator mode.) The string length given in the DIM statement is the maximum number of characters which may be assigned to the string. It is usually helpful to dimension a string array so that it is larger than is necessary.

The space required (in words) to store a string is one half the number of characters reserved in the DIM statement plus three. For example, a string with a 100 character length takes up 53 words of calculator memory.

SUBSTRINGS

A 'substring' is a part of a string. It can be a single character, or a contiguous set of characters within a string, referenced by the subscripted string name. A single subscript specifies the first character of a substring and implies that all characters following are also part of the substring. Alternatively, a second subscript can be used as a limiter to specify the last character to be included in the substring. Expressions may be used as subscripts.

SUBSTRINGS
(continued)

Example

Program Statement:

```
A$="SEE DICK RUN"
B$=A$[5]
B$=A$[5,8]
```

Result Field:

SEE△DICK△RUN
DICK△RUN
DICK

A '△' represents spaces or blanks within a string.

NOTE

To gain a better understanding of strings, you may wish to write simple programs containing a DIM statement, a string value statement, and a DISP or PRINT statement. To see the result contained in A$, for instance, you could use PRINT A$. To see only a substring, you might use PRINT A$(5,8). A string or substring of up to 72 characters can be printed with a PRINT statement.

When two subscripts are used to specify a substring, the second subscript is generally greater than or equal to the first. If the first subscript exceeds the second subscript an error will occur, unless the first exceeds the second by exactly one, in which case the null string is assumed.

Example

Program Statement:

```
A$="ABANDON SHIP"
B$=A$[5,6]
B$=A$[5,5]
B$=A$[5,4]
B$=A$[5,3]
```

Result Field:

ABANDON△SHIP
DO
D
"" (the null string)
ERROR

STRING BOUNDARIES

The logical boundaries of a string are its first and last elements containing assigned values. The value of a single subscript, or of the first subscript, if two subscripts describe a substring, must be no more than the logical string length plus one. When the logical right boundary is exceeded by more than one, or the logical left boundary is negative or zero, an error will occur.

Example

Program Statements:

```
A$="LOOK OUT BELOW!"
B$=A$[15]
B$=A$[16]
B$=A$[17]
B$=A$[0]
```

Result Field:

LOOK△OUT△BELOW!
!
"" (the null string)
ERROR
ERROR

Chapter 3

RULES OF STRING MODIFICATION

A string or substring can be modified by another string or substring. For example, part of a string can be changed, or an insertion can be made. The string to be modified is called the destination string. For the statement A$=B$, then, A$ is a destination string, and B$ is a modifying string.

The characteristics (length, content) of the destination string after modification depend not only upon the characteristics of the modifying string, but also upon the number of subscripts given for the destination string.

NO SUBSCRIPT

When the destination string has no subscript, the entire string is replaced by the modifying string or substring, and its characteristics after modification are the same as those of the modifying string or substring.

Example

Program Statements:	Result Field:
A$="1234"	1234
B$="ABCDEFGH"	ABCDEFGH
A$=B$	ABCDEFGH
A$=B$[3]	CDEFGH
A$=B$[2,4]	BCD

ONE SUBSCRIPT

When the destination string has one subscript, the indicated substring is replaced by the modifying string or substring.

If the indicated destination substring is shorter than the modifying string or substring, the modification causes the destination string to be lengthened.

Example

Program Statements:	Result Field:
A$="ABCDE"	ABCDE
A$[4]="XYZ"	ABCXYZ
A$[3]="ANDON SHIP"	ABANDON△SHIP

ONE SUBSCRIPT
(continued)

If the indicated destination substring is the null string (that is, when the subscript is equal to the logical length of the destination string plus one), the modifying string is attached at the end of the destination string.

Example

Program Statement:	Result Field:
A$="ABC"	ABC
A$[4]="XYZ"	ABCXYZ

If the indicated destination substring is longer than the modifying string or substring, the modification causes the destination string to be shortened.

Example

Program Statement:	Result Field:
A$="ABCDEF"	ABCDEF
A$[3]="12"	AB12
A$[3]=A$[3,2]	AB

If the indicated destination substring is equal in length to the modifying string or substring, the modification will not affect the length of the destination string.

Example

Program Statement:	Result Field:
A$="ABCDEF"	ABDCEF
A$[4]="XYZ"	ABCXYZ
A$[2]="12345"	A12345

TWO SUBSCRIPTS

When the destination string has two subscripts, the indicated substring is replaced by the modifying string or substring.

The leftmost character in the modifying string or substring is moved to the leftmost position in the indicated destination substring. Then the next adjacent character is moved, and so forth, until the indicated destination substring is filled. If the modifying string or substring is shorter than the indicated destination substring, the remainder of the destination substring is filled with spaces. If the modifying string is longer than the indicated destination substring, the remainder of the modifying string or substring is truncated.

Example

Program Statement:	Result Field:
A$="AIR MAIL"	AIR△MAIL
A$[5,6]="FO"	AIR△FOIL
A$[1,4]="TIN"	TIN△FOIL
A$[5,8]="CANSOK"	TIN△CANS

When the destination string has two subscripts, its length after modification will either be greater than before, or remain unchanged. When the value of the second subscript is greater than the logical length of the destination string, the modification results in a lengthened string.

Example

Program Statement:	Result Field:
A$="JOHN"	JOHN
A$[5,13]=" IS TALL."	JOHN△IS△TALL.
A$[12,18]="ENTED."	JOHN△IS△TALENTED.△
A$[5,20]=" HAS FIVE CHILDREN."	JOHN△HAS△FIVE△CHILDR

NOTES

Chapter 4

STRING FUNCTIONS

A string function returns a numerical value to an expression, in the same way that any other function, such as the square root function, does.

String functions enable you to determine the length, and analyze the content, of a string. This is useful when strings of different length or content are processed at the same time; for instance, when entering strings from the keyboard.

The following syntax notations are used:

'string name' can be a string or substring. A literal value, enclosed in quotes, is not permitted.

'string' can be a string, substring or literal value enclosed in quotes.

parentheses ( ) are required when shown.

NOTE

Shifted string variables are not equal to their corresponding unshifted values thus, (SHIFT) (A) does not equal (A). However, the 9866A printer and the calculator display will print both codes as a capital "A".

THE LEN FUNCTION

LEN (string name)

The LEN function obtains the length of a string or substring. The logical length is found, which is not necessarily equal to the reserved length defined by the DIM statement.

Example

Program Statement:	Result Field:
`A$="ABCD"`	
`B=LEN(A$)`	ABCD 4
`A$[LEN(A$)+1]="EFG"`	ABCDEFG
`A$[LEN(A$)]="!"`	ABCDEF!

AND, OR NOT

Although logical operators cannot be used with string variables directly (see page 5-1), the LEN function *may* be used with logical operators, as shown below:

```
200 X= NOT LEN(B$)
320 IF LEN(R$) OR  NOT LEN(C$) THEN 400
800 GOSUB LEN(P$) OF 820,840,860
```

THE POS FUNCTION

POS (string name, string)

The POS function determines the position of a substring within a string. If the second string is not a part of the first, the value of the function is zero. If the second string is found to be part of the first, the value of the function is the position in the first string at which the second string starts.

Example

Program Statement:	Result Field:
`A$="ABCD"`	ABCD
`B=POS(A$,"C")`	3
`B$="BC"`	BC
`B=POS(A$,B$)`	2
`A$[POS(A$,"D")]="!"`	ABC!
`B=POS(A$,"@")`	0

THE VAL FUNCTION

VAL (string name)

The characters in a string are not recognized as numeric data and therefore they cannot normally be used in computations. The VAL function allows the numerical value of a string of digits to be used in computations. The string is converted into a number by the same rules used in an INPUT statement. The first character must be a digit; decimal points, plus or minus sign and E-notation are also allowed. Also, numerical data entries can be combined logically with input text.

Example

Program Statement:	Result Field:
`A$="ABC123X5"`	ABC123X5
`B=VAL(A$[4])`	123
`B=2*VAL(A$[5])`	46
`B=VAL(A$[4])*VAL(A$[POS(A$,"X")+1])`	615

Chapter 5

STRING COMPARISONS

The IF statement allows comparison of strings or substrings. All of the relational operators allowed in numerical comparisons apply to string comparisons, as follows:

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

The following string comparison could be included in a program to allow communication between the calculator and the user.

(The logical operators – AND, OR, NOT – cannot be used with string variables.)

```
10 DIM A$[10]
20 DISP "DO YOU WISH TO CONTINUE";
30 INPUT A$
40 IF A$="YES" THEN 90
50 STOP
```

When the data request (?) is displayed the user can enter "YES", "NO" or another appropriate reply.

In some cases, such as in alphabetic sequencing problems, it is useful to compare strings for conditions other than "equal to" and "not equal to". For example, to arrange several different strings in alphabetical order, the following type of string comparison could be included in a program.

```
10 DIM A$[20],B$[20],Z$[20]
20 INPUT A$,B$
30 IF A$<B$ THEN 70
40 Z$=B$
50 B$=A$
60 A$=Z$
70 PRINT A$;" IS LESS THAN ";B$
```

Within the calculator memory, each character contained in a string is represented by a standard ASCII* octal code, as shown in Table 5-1. When two string characters are compared, the smaller of the two characters is the one whose octal code is smaller. For example, "2" (octal code 062) is smaller than "R" (octal code 122).

Strings are compared character by character from left to right until a difference is found, (thus "ANT" is smaller than "BEE"). If one string ends before a difference is found, the shorter string is considered smaller. For example, "STEVE" is smaller than both "STEVE△" and "STEVEN".

* ASCII – American Standard Code for Information Interchange.

Table 5-1. ASCII Characters and Their Octal Equivalents

OCTAL CODE	ASCII CHARACTER	OCTAL CODE	ASCII CHARACTER	OCTAL CODE	ASCII CHARACTER
040	△ (blank)	076	>	134	\
041	!	077	?	135	]
042	"	100	@	136	↑
043	#	101	A	137	←
044	$	102	B	141	a
045	%	103	C	142	b
046	&	104	D	143	c
047	' (apost.)	105	E	144	d
050	(	106	F	145	e
051	)	107	G	146	f
052	*	110	H	147	g
053	+	111	I	150	h
054	, (comma)	112	J	151	i
055	-	113	K	152	j
056	.	114	L	153	k
057	/	115	M	154	l
060	0	116	N	155	m
061	1	117	O	156	n
062	2	120	P	157	o
063	3	121	Q	160	p
064	4	122	R	161	q
065	5	123	S	162	r
066	6	124	T	163	s
067	7	125	U	164	t
070	8	126	V	165	u
071	9	127	W	166	v
072	:	130	X	167	w
073	;	131	Y	170	x
074	<	132	Z	171	y
075	=	133	[	172	z

Chapter 6

INPUT AND OUTPUT OF STRINGS

As mentioned in Chapter 2, the standard input and output statements are used with strings. Here we will briefly mention the general rules which apply to the use of input and output statements, but most of the discussion will be about special rules which apply to string input and output.

THE INPUT STATEMENT

The INPUT statement allows string values to be entered from the calculator keyboard during program execution. You may enter up to 80 characters at a time into a string, or, for longer strings, you may enter several substrings of up to 80 characters:

```
10 DIM A$[160]
20 INPUT A$[1,80]
30 INPUT A$[81,160]
```

After the first 80 characters are entered, a second data request (?) will appear on the display so that the second 80 characters may be entered. Up to 255 characters can be entered in a string using this method.

Strings and numeric variables may be mixed in the INPUT statement:

```
10 DIM A$[40],B$[40]
20 INPUT A$,A,B$,B
```

Notice that the variable A and the string A$ can be used in the same program. When a data request (?) appears, the strings and variables can be entered all at once, with strings enclosed in quotes and fields separated by commas, e.g., ?"TODAY'S RECEIPTS", 512.52, "PAID OUT", 272.12. Avoid keying in leading blanks (i.e., spaces) before the value of the numeric variable or before the string variable within a quote field.

Alternatively, when the total length of the data entry is greater than 80 characters, each item can be entered separately. (In this case, the quotes would be optional):

```
?TODAY'S RECEIPTS
?512.52
?PAID OUT
?272.12
```

During data entry, the word STOP is recognized as a string value. Therefore, to exit from a programmed INPUT loop, the calculator END key is used. Otherwise, a test can be

THE INPUT STATEMENT
(continued)

included in your program so that when a certain input condition is met, the loop is satisfied:

```
10 DIM A$[10]
20 INPUT A$
30 IF A$="STOP" THEN 50
40 GOTO 20
50 END
```

In the above example, data requests (?) will continue until "STOP" is entered.

THE READ STATEMENT

The READ statement is used with the DATA statement. The DATA statement may contain a mixture of string and numeric variables, but all string values must be enclosed in quotes.

```
10 DIM A$[10],B$[10],C$[10]
20 READ A,A$,B,B$,C$
30 DATA 64,"TEMP",29.5,"PRESSURE","OCT. 16"
```

If string values are found when numeric data is expected, or numeric data is found when strings are expected, an error will occur.

THE PRINT STATEMENT

The PRINT statement can print up to 72 characters on the output printer or typewriter. Any characters beyond 72 in the string or substring will not be printed. For longer strings, you may print several substrings of up to 72 characters each.

A mixture of strings, numeric variables and constants may be included in a PRINT statement:

```
10 DIM A$[20],B$[20]
20 READ A,A$,B$
30 DATA 56,"TEMPERATURE","BAROMETER"
40 PRINT A$,"TODAY IS",A,B$,29.5
```

THE WRITE STATEMENT

The WRITE statement allows strings to be output to other output devices, as well as to the primary printer.

WRITE statements which do not reference a FORMAT statement restrict their outputs to a maximum line-length of 72 characters. This is because they behave exactly like PRINT statements. Thus,

```
WRITE (15,*)A$
```

is equivalent to

```
PRINT A$
```

Strings and numeric variables can be mixed in a WRITE statement. However, unlike numeric variables, the string variables do not reference the corresponding FORMAT specifications. In program 2 (below), for example, the format specifications (2F5.1) are ignored by the strings and used only by the numeric variables.

The following two sets of instructions, which include WRITE statements with a mixture of strings, numeric variables and constants, have the same printout (as shown):

1.
```
10 A=56
20 FORMAT 2F5.1
30 WRITE (15,20)"TEMPERATURE",A," BAROMETER",29.5
```

2.
```
10 DIM A$[20],B$[20]
20 A$="TEMPERATURE"
30 B$=" BAROMETER"
40 A=56
50 FORMAT 2F5.1
60 WRITE (15,50)A$,A,B$,29.5
```

```
TEMPERATURE 56.0 BAROMETER 29.5
```

When the WRITE statement does reference a FORMAT statement, the length of any output is no longer restricted to 72 characters maximum. For example, in program 2 (above) the WRITE statement in line 60 references the FORMAT in line 50; therefore, the output could have had more than 72 characters (assuming of course that A$ and B$ had originally been dimensioned large enough, and assuming a large enough printer).

THE WRITE STATEMENT
(Continued)

None of the FORMAT statement specifications affect string length; the simple fact that the FORMAT statement is referenced is sufficient to lift the 72-character restriction. This is illustrated in the following program, where the WRITE statement contains no numeric variables so would not normally need to reference a FORMAT statement.

This program can output a 160-character string on one line (assuming that the column-width of the printing device allows at least 160 characters). Since there is no other FORMAT statement for the WRITE statement (line 40) to reference, the FORMAT in line 45 has been added. The F2.0 in the FORMAT statement is a 'dummy' specification in that it does not affect the printout but is needed to complete the FORMAT syntax. (The 'dummy' specification does not have to be F2.0 – E8.1, B, or any other allowable specification would be just as effective.)

```
10 DIM A$[160]
20 INPUT A$[1,80]
30 INPUT A$[81,160]
40 WRITE (15,45)A$
45 FORMAT F2.0
50 END
```

THE DISP STATEMENT

The DISP statement allows the 32 character display to be used as an output device:

```
10 DIM A$[20],B$[20]
20 A$="ABANDON "
30 B$="SHIP"
40 DISP A$;B$
```

Chapter 7

SAVING STRING VARIABLES

When programs are reproduced into memory from the tape cassette by means of a LINK statement, all variables currently in memory are saved. If LOAD (or LOAD KEY) is used to reproduce the program, however, any string variables currently in memory are lost, including those strings defined in a COM statement.† (Numeric variables defined in the COM are, of course, retained). Thus, if LOAD (or LOAD KEY) must be executed, then those strings which must be saved should first be stored on a cassette, so that they can be loaded back into memory afterwards.

Data is stored on the tape cassette with the STORE DATA command. The general syntax has an optional "array" parameter, but this parameter applies only to numeric arrays and cannot be used for storing strings. Therefore, when storing string data, the syntax is:

STORE DATA file

Thus to store strings on tape, the string names must first be specified in a COM statement. All variables specified in COM are stored on the indicated file when the above syntax is used. For instance, to store A$ and B$ on file 3 of your tape cassette, your program could be as follows:

```
1 COM A$[80],B$[80]
2 INPUT A$
3 INPUT B$
4 STORE  DATA 3
5 END
```

Of course, had numeric variables been specified in COM, they too would have been stored in file 3.

The command used to load the strings back into memory must parallel the STORE DATA command; so the syntax is:

LOAD DATA file

When the LOAD DATA command is executed, the memory must contain a suitable COM statement, capable of properly accepting the new data.

†The Advanced Programming I ROM (Option 279 or 11279B) has a TRANSFER statement which can convert a string to a numeric array, and vice versa. This adds considerable flexibility to string manipulation; for example, strings defined in a COM statement can be saved, in numerical form, when a LOAD command is executed, and strings effectively longer than 255 characters can be used.

NOTES

APPENDIX

ERROR MESSAGES

INDICATION	MEANING
ERROR 70	Incomplete IF statement.
ERROR 71	Illegal string function syntax.
ERROR 72	Logical string length exceeded.
ERROR 73	Operation is on non-contiguous string. Substring requested is beyond the logical boundary for the string and is undefined.
ERROR 74	Maximum string length exceeded. Additional string length must be specified in the DIM statement. If DIM (255) has already been specified, program modification may be required.
ERROR 75	Illegal DATA encountered during READ statement execution. Character data found; numeric data expected.
ERROR 76	VAL function argument non-numeric.
ERROR 77	Illegal characters entered during INPUT statement execution. Character data found; numeric data expected.

SAMPLE APPLICATIONS

INSERT CHARACTERS IN A STRING

To insert "XYZ" in "ABCDEF" giving "ABCXYZDEF", you would use program statements as follows:

Program Statement:	Result Field:
`A$="ABCDEF"`	ABCDEF
`A$[7]=A$[4]`	ABCDEFDEF
`A$[4,6]="XYZ"`	ABCXYZDEF

DELETE CHARACTERS FROM A STRING

To delete "CD" from "ABCDEF", contained in A$, giving "ABEF", you would simply use A$(3)=A$(5). Suppose you are to find and delete any occurrences of the combination /* in a string entered from the keyboard. Here is such a program:

```
10 DIM A$[100]
20 DISP "ENTER TEXT";
30 INPUT A$
40 IF A$[1,4]="STOP" THEN 100
50 IF POS(A$,"/*")=0 THEN 80
60 A$[POS(A$,"/*")]=A$[POS(A$,"/*")+2]
70 GOTO 50
80 PRINT A$
90 GOTO 20
100 END
```

When the above program is executed, input will be checked for /* substrings until STOP is found. Notice that the POS function is used here, since exact positions of the characters cannot be predicted.

It is often useful to remove blanks from input text with this technique.

CONCATENATION (LINKING) OF STRINGS

To link "123" to "ABC", contained in A$, giving "ABC123", you would use A$(4)="123". If the length of A$ is unknown, such as when A$ is entered from the keyboard, the LEN function can be used as follows.

```
10 DIM A$[50],B$[50]
20 PRINT "ENTER NAME"
30 INPUT A$
40 PRINT A$
50 A$[LEN(A$)+1]=" IS "
60 PRINT "ENTER DESCRIPTION OF THIS PERSON"
70 INPUT B$
80 B$[LEN(B$)+1]="."
90 A$[LEN(A$)+1]=B$
100 PRINT A$
110 END
```

CONVERSATION WITH A USER

If you have written a program, you will know which data is to be given and the required order of input from the keyboard. However, your program may be used by others who are not familiar with the operation of your program. To help them, you can make your program "conversational" by using strings. The program can inform the user of the data required, and can accept answers such as "YES" and "NO", rather than numeric codes which might have little or no meaning to the user. Here is an example of conversational programming:

```
10 DIM A$[50]
20 PRINT "WHAT ANIMAL HAS A LONG TRUNK ";
30 PRINT "AND LARGE FLAPPING EARS?"
40 INPUT A$
50 IF POS(A$,"ELEPHANT")>0 THEN 130
60 IF POS(A$,"EL")=0 THEN 110
70 IF POS(A$[POS(A$,"EL")],"NT")=0 THEN 110
80 PRINT A$,"IT SHOULD BE ELEPHANT,"
90 PRINT "MAYBE YOU HAD TROUBLE SPELLING IT."
100 GOTO 140
110 PRINT A$,"NO, THE ANSWER SHOULD BE ELEPHANT"
120 GOTO 140
130 PRINT A$,"RIGHT, IT'S THE ELEPHANT"
140 PRINT "WHAT ANIMAL IS RAISED FOR ITS WOOL?"
150 INPUT A$
160 IF POS(A$,"SHEEP")>0 THEN 190
170 PRINT A$," NO, SHEEP ARE RAISED FOR THE WOOL."
180 GOTO 200
190 PRINT A$,"  RIGHT! SHEEP ARE RAISED FOR THE WOOL"
200 END
```

Notice in the above example that if the word elephant is not found, a misspelling of the word, (e.g. elefant) is accepted. Also, the POS fuction is used so that if the substring is found anywhere in the entered character string, such as in the plural form or as part of a sentence, the response is accepted as a correct response.

PRINTING A FORM LETTER

A form letter like the one below might be used by a publishing company in processing magazine subscriptions. The letter is printed using the following program.

```
DEAR MR. JACKSON,

YOUR SUBSCRIPTION TO BUSINESS VIEWS
IS ABOUT TO EXPIRE...

YOU NEED NOT MISS A NEWSY WORD OR
NOVEL IDEA THIS MONTH OR ANY OTHER!

SIMPLY RETURN THE ENCLOSED FORM AND
YOU'LL CONTINUE TO ENJOY BUSINESS VIEWS
WITHOUT INTERRUPTION.

                    CORDIALLY,
```

permitted, because they make things easier to read, but are not required. 'Price' must be the first item entered. It may be dollars, as $43.99, it may be cents, as 89 cents, or it may be just a number, as 9.98. 'Quantity' can be entered as: 'doz' or 'dozen', as 8 doz or 8 dozen, 'lb', as 5lb, 'oz', as 20 oz, 'lb,oz', as 1 lb 4 oz, or it may stand alone.

Here are some sample entries: 50 cents per 5 lbs; $5.00/doz; $.43/1 lb 4 oz.

```
10 DIM A$[80],B$[80],C$[25]
20 DISP "ENTER PRICE PER QUANTITY";
30 INPUT A$
40 IF POS(A$,"STOP")>0 THEN 470
50 C$="EACH"
60 X=0
70 A=VAL(A$[POS(A$,"$")+1])
80 P=POS(A$,"PER")
90 IF P=0 THEN 120
100 B$=A$[P+3]
110 GOTO 150
120 P=POS(A$,"/")
130 IF P=0 THEN 20
140 B$=A$[P+1]
150 B$[LEN(B$)+1]=" "
160 GOSUB 410
170 IF I#1 THEN 200
180 X=1
190 GOTO 210
200 X=VAL(B$)
210 IF POS(B$,"LB")=0 THEN 250
220 C$="PER OZ"
230 X=X*16
240 B$=B$[POS(B$,"LB")+2]
250 Z=POS(B$,"OZ")
260 IF Z=0 THEN 350
270 IF POS(B$,"DOZ")>0 THEN 350
280 C$="PER OZ"
290 IF POS(A$,"LB")=0 THEN 350
300 GOSUB 410
310 IF I#1 THEN 340
320 X=X+1
330 GOTO 350
340 X=X+VAL(B$)
350 IF POS(B$,"DOZ")=0 THEN 370
360 X=X*12
370 PRINT A$,"-----",A/X;C$
380 GOTO 20
390 REM THIS SUBROUTINE CHECKS FOR NUMERIC DATA
400 REM BEFORE A VAL FUNCTION IS EXECUTED.
410 FOR I=1 TO LEN(B$)
420 IF B$[I,I]>"9" THEN 460
430 IF B$[I,I]>"0" THEN 450
440 IF B$[I,I]#" " THEN 460
450 NEXT I
460 RETURN
470 END
```

THE DIM STATEMENT

```
10 DIM A$[255]
```

STRING MODIFICATION

Program Statement:	Result Field:
`20 A$="JOHN"`	JOHN
`30 A$[5]=" IS TALL."`	JOHN∧IS∧TALL.
`40 A$[1,4]="THAT"`	THAT∧IS∧TALL.
`50 A$[9]=A$[10]`	THAT∧IS△ALL.

THE LEN FUNCTION

LEN (string name) Obtains the length of a string or substring.

THE POS FUNCTION

POS (string name, string) Determines the position of a substring within a string.

THE VAL FUNCTION

VAL (string name) Converts the digits in a string to a number.

STRING COMPARISONS

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

THE USE OF STRING FUNCTIONS

Program Statement:	Result Field:
`60 A$="STOP THE TRAIN!"`	STOP△THE△TRAIN!
`70 IF POS(A$,"STOP")=0 THEN 90`	STATEMENT 80
`80 A$[LEN(A$)+1]=" NOW!"`	STOP△THE△TRAIN!△NOW!
`90 A$="542 PEOPLE"`	542△PEOPLE
`100 X=VAL(A$)`	542