CHAPTER

30

PRINT USING Procedures

True BASIC normally prints numbers in a form convenient for most purposes. But on occasion you may prefer a more elaborate form. For example, you may want to print financial quantities with two decimal places (for cents) and, possibly, with commas inserted every three digits to the left of the decimal point. PRINT USING provides a way to print numbers in this and almost any other form.

Here is an example of the PRINT USING statement.

```
PRINT USING format$: x, y, z
```

Format$ is a string of characters that contains the instructions to PRINT USING for "formatting" the printing of *x, y,* and *z*. This string is called a *format string*. It may be a string variable (as shown above), a *quoted-string*, or a more general string expression.

PRINT USING also allows one to print strings centered or right-justified, as well as left-justified. (The normal PRINT statement prints both strings and numbers left-justified within each print zone; see Chapter 34, the PRINT statement.)

The function USING$ duplicates the PRINT USING statement almost exactly but returns the result as a string rather than printing it on the screen. For example, the following two statements yield the same output as the preceding PRINT USING statement.

```
LET outstring$ = using$(format$, x, y, z)
PRINT outstring$
```

The USING$ function allows you to modify or save the string *outstring$* before printing it. You can also use this function with WRITE and PLOT TEXT statements. (See Chapter 32, for the USING$ function.)

We will first examine how to format numerical output.

Formatting Numbers

The basic idea of a format string is that the symbol "#" stands for a digit position. For example, let us compare the output resulting from two similar PRINT statements, the first a normal PRINT statement and the second employing USING.

```
PRINT x
PRINT USING "###": x
```

In the following table, the symbol "|" is used to denote the left margin and does not actually appear on the screen.

```
x          PRINT x            PRINT USING "###": x
-          -------            --------------------
1          | 1                |  1
12         | 12               | 12
123        | 123              |123
1234       | 1234             |***
```

We notice several things. Without USING, the number is printed left-justified with a leading space for a possible minus sign, and occupying only as much space as needed. With USING, the format string "###" specifies a *field length* of exactly three characters. The number is printed right-justified in this *field*. If the field is not long enough to print the number properly, asterisks are printed instead, the unformatted value (here, of *x*) is printed on the next line and printing continues on the following line. If all you need to do is to print integer numbers in a column but with right-justification, then the preceding example will suffice.

Printing financial quantities so that the decimal points are aligned is important. Also, you may want to print two decimal places (for the cents) even when they are "0". The following example shows how to do this. (In order to print negative numbers, the format string must start with a minus sign.)

```
x          PRINT x            PRINT USING "-##.##": x
--         -------            ----------------------
1          | 1                |  1.00
1.2        | 1.2              |  1.20
-3.57      |-3.57             |- 3.57
1.238      | 1.238            |  1.24
123        | 123              |******
0          | 0                |   .00
-123       |-123              |******
```

Notice that two decimal places are always printed, even when they consist of zeroes. Also, the result is first rounded to two decimals. If the number is negative, the minus sign occupies the leading digit position. If the number is too long to be printed properly

(possibly because of a minus sign), asterisks are printed instead, the unformatted value is printed on the next line, and printing continues on the following line.

Financial quantities are often printed with a leading dollar sign ($), and with commas forming three-digit groups to the left of the decimal point. The following example shows how to do this with PRINT USING.

```
x                       PRINT USING "$#,###,###.##": x
--                      ------------------------------
0                                  |$       .00
1                                  |$      1.00
1234                               |$   1,234.00
1234567.89                         |$1,234,567.89
1e6                                |$1,000,000.00
1e7                                |*************
```

Notice that the dollar sign is always printed and is in the same position (first) in the field. Also, the separating commas are printed only when needed.

You might sometimes want the dollar sign ($) to *float* to the right, so that it appears next to the number, avoiding all those blank spaces between the dollar sign and the first digit in the preceding example. The following example shows how to do this.

```
x                       PRINT USING "$$$$$$$#.##": x
--                      ---------------------------
0                                  |     $ .00
1                                  |     $1.00
1234                               |   $1234.00
1234567.89                         |$1234567.89
```

Digit positions represented by “$” instead of “#” cannot surround or be next to commas.

In the previous two examples, no negative amounts can be printed since the format string does not start with or contain a minus sign.

The format string can also allow leading zeroes to be printed, or to be replaced by asterisks (*). You might find the latter useful if you are preparing a check-writing program.

```
x                       PRINT USING "$%,%%%,%%%.##": x

--                      ------------------------------
0                                  |$0,000,000.00
1                                  |$0,000,001.00
1234                               |$0,001,234.00
1234567.89                         |$1,234,567.89
```

```
x                       PRINT USING "$*,***,***.##": x
--                      ------------------------------
0                                    |$*********.00
1                                    |$********1.00
1234                                 |$****1,234.00
1234567.89                           |$1,234,567.89
```

You can also format numbers using scientific notation. Because scientific notation has two parts, the *decimal-part* and the *exponent-part*, the format string must also have two parts. The *decimal-part* follows the rules already illustrated. The *exponent-part* consists of from three to five *carets* (^) that must immediately follow the *decimal-part*. The following example shows how.

```
x                       PRINT USING "+#.#####^^^^": x
--                      -----------------------------
0                                    |+0.00000e+00
123.456                              |+1.23456e+02
-.001324379                          |-1.32438e-03
7e30                                 |+7.00000e+30
.5e100                               |+5.00000e+99
5e100                                |************
```

Notice that a leading plus sign (+) in the format string guarantees that the sign of the number will be printed, even when the number is positive. Notice also that the last number cannot be formatted because the exponent part would have been 100, which requires an exponent field of *five* carets. Notice also that if there are more carets than needed for the exponent, leading zeroes are inserted. Finally, notice that trailing zeroes in the decimal part are printed.

Floating Characters

You'll notice that one of the previous examples includes several "$", but that only one of them is actually printed. It is printed just to the left of the left-most non-zero digit, but always within the positions given by the sequence of "$". We say that the sequence of "$" defines a *floating region* and that the spot where the "$" is printed *floats* within this region.

Besides the "$", the plus sign (+) and the minus sign (-) can also define *floating regions*. The rules are:

1. You can use either zero, one, or two different floating characters ("+" and "-" cannot both appear, and neither can commas.)
2. You can repeat the first (or only) floating character an arbitrary number of times, but not the second.

3. Zero to two different floating characters generate a sequence of zero to two characters called a *header*, as follows:

The Floating Header

First	Second	Positive	Negative
$	+	"$+"	"$–"
$	–	"$ "	"$–"
$	none	"$"	error
+	$	"+$"	"–$"
+	none	"+"	"–"
–	$	" $"	"–$"
–	none	" "	"–"
none	none	""	error

   Notice that the *header* contains the same number of characters as the number of different floating characters.

4. The zero to two character *header* will be printed as far to the right as possible within the *floating region*.

5. The numerical value's leading digits can overflow into the *floating region*, thereby "pushing" the *header* to the left.

6. If the numerical value exceeds the total space provided, the entire space is filled with asterisks.

The following example illustrates some of these rules.

```
PRINT x                 PRINT USING "$$$$$$$-#,###.##": x
-------                 ---------------------------------
| 0                                |        $     .00
| 1                                |        $    1.00
|-1                                |        $-   1.00
| 4321.5                           |       $ 4,321.50
|-4321.5                           |       $-4,321.50
| 1.23456789e+7                    |   $ 12345,678.90
|-1.23456789e7                     |   $-12345,678.90
| 1000000000                       | $ 1000000,000.00
|-1000000000                       | $-1000000,000.00
```

Notice that the "$" is never printed outside the *floating region*. A place is allocated for the minus sign. The leading digits of the numerical value can overflow into the *floating region*, which does not (and cannot) contain commas.

Formatting Strings

Strings can also be formatted through PRINT USING or the function USING$, although there are fewer options for strings than for numbers. Strings can be printed in the formatted field either left-justified, centered, or right-justified. As with numbers, if the string is too long to fit, then asterisks are printed, the actual string is printed on the next line, and printing continues on the following line. The following example shows several cases.

```
USING           String to be Printed
string      "Ok"        "Hello"     "Goodbye"
------      ------      -------     ---------
"<####"     |Ok         |Hello      |*******
"#####"     | Ok        |Hello      |*******
">####"     |   Ok      |Hello      |*******
```

Notice that if centering cannot be exact, the extra space is placed to the right.

Any numeric field can be used to format a string, in which case the string is centered. This is especially valuable for printing headers for a numeric table. The following example shows how you can format headers using the same format string we used earlier for numbers.

```
s$                      PRINT USING "$#,###,###.##": s$
-------------           -------------------------------
"Cash"                                |    Cash
"Liabilities"                         | Liabilities
"Accounts Receivable"                 |*************
```

Multiple Fields and Other Rules

A PRINT USING format string can contain several format items. For example, to print a table of sines and cosines, we may want to use:

```
LET format$ = "-#.###  -#.######  -#.######"
PRINT USING format$: x, sin(x), cos(x)
```

The value of *x* will then be printed to three decimals, while the values of the sine and cosine will be printed to six decimals. Notice also that spaces between the format items will give equal spaces between the columns in the printed result.

If there are more format items than there are values (numbers or strings) to be printed, the rest of the format string starting with the first unused format item is ignored. If there are *fewer* format items than values to be printed, the format string is reused, but starting on the next line. Thus,

```
PRINT USING "  -#.#####": 1.2, 2.3, 3.4
```

will yield:

```
1.20000
2.30000
3.40000
```

Literals in Format Strings

We have just seen that spaces between format items in a format string are printed. That is, if there are four spaces, the four spaces are printed. The same is true for more general characters that may appear between format items. The rule is simple: you can use any sequence of characters between format items *except* the special formatting characters. The characters you use will then be printed.

The special formatting characters are:

```
#   %   *   <   >   ^   .   +   -   ,   $
```

The following example illustrates this use.

```
PRINT USING "#.## plus #.## equals #.##": 1.2, 2.3, 1.2+2.3
```

will yield:

```
1.20 plus 2.30 equals 3.50
```

If there are fewer values than format items, the unused format items are ignored, but the last intervening literal string is printed. Thus,

```
PRINT USING "#.## plus #.## equals #.##": 1.2, 2.3
```

will yield

```
1.20 plus 2.30 equals
```

If you need to have one of the special formatting characters appear in the output – for example, if you want to have a final period, as in the last example – you can simply add a one-character field to the format string and add the *quoted-string* "." to the PRINT statement. Thus,

```
LET x = 1.2
LET y = 2.3
PRINT USING "#.## plus #.## equals #.## #": x, y, x+y, "."
```

will yield

```
1.20 plus 2.30 equals 3.50 .
```

True BASIC employs two forms of the PRINT USING and Using$ functions. The first is the version used since version 1.0 of the Language System. This is the default in the professional version.

The other is a completely ANSI-standard version, which is slightly more restrictive. If you wish to use this version, you may include the statement OPTION USING ANSI in your program before the first USING statement that you wish to conform to the ANSI standard.

To switch back to the default version, you may include the statement OPTION USING TRUE before the first USING statement that you wish to conform to the True BASIC specifications.

Exceptions

The following runtime errors can arise:

Exceptions:	8201	Badly formed USING string.
	8202	No USING item for output.
	8203	USING value too large for field. (nonfatal)
	8204	USING exponent too large for field. (nonfatal)