Formatted Output - Printf

The functions described in this section (printf and related functions) provide a convenient way to perform formatted output. You call printf with a format string or template string that specifies how to format the values of the remaining arguments.

Unless your program is a filter that specifically performs line or character-oriented processing, using printf or one of the other related functions described in this section is usually the easiest and most concise way to perform output. These functions are especially useful for printing error messages, tables of data, and the like.

Formatted Output Basics

The printf function can be used to print any number of arguments. The template string argument you supply in a call provides information not only about the number of additional arguments, but also about their types and what style should be used for printing them.

Ordinary characters in the template string are simply written to the output stream as-it-is, while conversion specifications introduced by a `%' character in the template cause subsequent arguments to be formatted and written to the output stream. For example:

```
int pct = 37;
char filename[] = "foo.txt";
printf ("Processing of `%s' is %d%% finished.\nPlease be patient.\n",
        filename, pct);
```

produces output like:

```
Processing of `foo.txt' is 37% finished.
Please be patient.
```

This example shows the use of the `%d' conversion to specify that an int argument should be printed in decimal notation, the `%s' conversion to specify printing of a string argument, and the `%%' conversion to print a literal `%' character.

There are also conversions for printing an integer argument as an unsigned value in octal, decimal, or hexadecimal radix (`%o', `%u', or `%x', respectively); or as a character value (`%c').

Floating-point numbers can be printed in normal, fixed-point notation using the `%f' conversion or in exponential notation using the `%e' conversion. The `%g' conversion uses either `%e' or `%f' format, depending on what is more appropriate for the magnitude of the particular number.

You can control formatting more precisely by writing modifiers between the `%' and the character that indicates which conversion to apply. These slightly alter the ordinary behavior of the conversion. For example, most conversion specifications permit you to specify a minimum field width and a flag indicating whether you want the result left or right-justified within the field.

The specific flags and modifiers that are permitted and their interpretation vary depending on the particular conversion. They're all described in more detail in the following sections. Don't worry if all this seems excessively complicated at first; you can almost always get reasonable free-format output without using any of the modifiers at all. The modifiers are mostly used to make the output look "prettier" in tables.

Output Conversion Syntax

This section provides details about the precise syntax of conversion specifications that can appear in

a printf template string.

Characters in the template string that are not part of a conversion specification are printed as-it-is to the output stream. Multibyte character sequences are permitted in a template string.

The conversion specifications in a printf template string have the general form:

```
% flags width [ . precision ] type conversion
```

For example, in the conversion specifier `%-10.8ld', the `-' is a flag, `10' specifies the field width, the precision is `8', the letter `l' is a type modifier, and `d' specifies the conversion style. (This particular type specifier says to print a long int argument in decimal notation, with a minimum of 8 digits left-justified in a field at least 10 characters wide.)

In more detail, output conversion specifications consist of an initial `%' character followed in sequence by:

- Zero or more flag characters that modify the normal behavior of the conversion specification.

- An optional decimal integer specifying the minimum field width. If the normal conversion produces fewer characters than this, the field is padded with spaces to the specified width. This is a minimum value; if the normal conversion produces more characters than this, the field is not truncated. Normally, the output is right-justified within the field.

The GNU library's version of printf also allows you to specify a field width of `*'. This means that the next argument in the argument list (before the actual value to be printed) is used as the field width. The value must be an int. Other C library versions may not recognize this syntax.

- An optional precision to specify the number of digits to be written for the numeric conversions. If the precision is specified, it consists of a period (`.') followed optionally by a decimal integer (which defaults to zero if omitted).

The GNU library's version of printf also allows you to specify a precision of `*'. This means that the next argument in the argument list (before the actual value to be printed) is used as the precision. The value must be an int. If you specify `*' for both the field width and precision, the field width argument precedes the precision argument. Other C library versions may not recognize this syntax.

- An optional type modifier character, which is used to specify the data type of the corresponding argument if it differs from the default type. (For example, the integer conversions assume a type of int, but you can specify `h', `l', or `L' for other integer types.)

- A character that specifies the conversion to be applied.

The exact options that are permitted and how they are interpreted vary between the different conversion specifiers. Following are the basic printf conversions:

Character	**Argument Type; Printed As**
d, i	int; decimal number
o	int; unsigned octal number (without a leading zero)
x, X	int; unsigned hexadecimal number
u	int; unsigned decimal number
c	int; single character
	char *; print character from the string until a '\0' or the number of

s	characters given by precision
f	double; [-]m.dddddd, where the number of d's is given by the precision (default 6)
e, E	double; [-]m.dddddde+xx or [-]m.ddddddE+xx, where the number of d's is given by the precision (default 6)
g, G	double; use %e or %E if the exponent is less than -4 or greater than or equal to the precision; otherwise use %f
p	void *; pointer
%	no argument is converted; print a %