The printf Fact Sheet

Character Escape Sequences

Escape Sequence	Name	Description
`\?`	Question mark	Allows the expression of a literal question mark character.
`\\`	Backslash	Allows the expression of a literal backslash character.
`\’`	Single quote	Allows the expression of a literal single quote character.
`\”`	Double quote	Allows the expression of a literal double quote character.
`\0`	Null	Produces the special character whose ASCII code is zero.
`\a`	Alert (or Bell)	Produces an audible or visible alert on output devices, without affecting the position of subsequent output on the device.
`\b`	Backspace	Moves the output position to the previous position on the current output line.
`\f`	Form feed	Moves the output position to the next page on the output device.
`\n`	New line	Moves the output position to the beginning of the next line.
`\N`	Octal constant (N = octal digits)	Using octal digits (0 1 2 3 4 5 6 7), allows the programmer to specify a character using the octal representation of its ASCII code.
`\xN`	Hexadecimal constant (N = hex digits)	Using hexadecimal digits (0 1 2 3 4 5 6 7 8 9 A B C D E F), allows the programmer to specify a character using the hexadecimal representation of its ASCII code.
`\r`	Carriage return	Moves the output position to the beginning of the current line.
`\t`	Horizontal tab	Moves the output position to the next tab stop on the current line.
`\v`	Vertical tab	Moves the output position to the beginning of the line at the next vertical tab stop.

Format Conversion Specifiers in printf

Format	Name	Description
`%%`	Percent sign	Allows the expression of a literal percent sign character in the output.
`%c`	Character	Specifies a single unsigned character, or the exact number of characters specified by the length modifier.
`%d or %i`	Signed decimal integer	Specifies an optionally signed decimal integer. *Precision* specifies the minimum number of digits to generate. Adding an l (as in %ld or %li) specifies a long integer. Adding an ll specifies a long long int.
`%e or %E`	Decimal scientific notation	Specifies a floating point decimal number in scientific notation. The case of the ‘e’ in the format specifier indicates the case of the ‘e’ output in scientific notation. *Precision* specifies the number of fraction digits to generate.
`%f`	Decimal floating point number	Specifies an optionally signed floating point decimal number. *Precision* specifies the number of fraction digits to generate. %f is used for float or double, since floats are promoted to doubles in the variable argument list.
`%g or %G`	Decimal floating point number	Equivalent to %f or %e, whichever is shorter. The case of the ‘g’ in the format specifier indicates the case of the ‘e’ in the output, if scientific notation is used. *Precision* specifies the maximum number of significant digits to generate.
`%n`	Number of characters written	Places the number of characters written so far into the integer variable whose address is specified in the associated argument.
`%o`	Unsigned octal integer	Specifies an unsigned integer in octal notation. *Precision* specifies the minimum number of digits to generate.
`%p`	Pointer	Specifies the value of a pointer (a memory address) in hexadecimal notation.
`%s`	Character string	Specifies a null-terminated string of characters. *Precision* specifies the maximum number of characters to generate from the string.
`%u`	Unsigned decimal integer	Specifies an unsigned decimal integer. *Precision* specifies the minimum number of digits to generate. Adding an l (as in %lu) specifies an unsigned long integer. Adding an ll specifies an unsigned long long int.
`%x or %X`	Unsigned hexadecimal integer	Specifies an unsigned integer in hexadecimal notation. The case of the ‘x’ in the format specifier indicates the case of the digits A-F in the hexadecimal integer. *Precision* specifies the minimum number of digits to generate.

Gaining More Control over Output

Output may be more finely controlled by adding specifiers and modifiers to the conversion specifiers outlined in the previous table.

- **Minimum field-width specifier** – The default field width printed is the actual width of the data being printed. An integer placed between the % and the data type specifier pads the output with spaces or zeros, to ensure that the output is at least a minimum length. If the output exceeds this length, the full value is output without any padding, so that no data is hidden. By default, spaces are used for padding. Placing a 0 between the % and the minimum field-width specifier pads with zeros instead of spaces. For example, "%5d" prints an integer at least five digits wide, padded on the left with spaces as needed. "%07d" prints an integer at least seven digits wide, padded on the left with zeros as needed.

- **Precision modifier** – An integer, placed after a decimal point following the % or the minimum field-width specifier, provides output control that varies depending on the data type being printed, as follows:
  - By default, floating point formats (%f, %e, %E) print six digits after the decimal place. The precision modifier determines the number of decimal places printed. If it is 0, or a decimal point appears with no number following it, no decimal point appears in the output.
  - For integer and unsigned integer formats, the precision modifier determines the minimum number of digits printed. Leading zeros are added as needed.
  - For %g and %G formats, the precision modifier determines the number of significant digits printed. The default is six significant digits.
  - For %s, the precision modifier indicates the *maximum* number of characters to be printed.

- **Left-justifier** – By default, all fields are right-justified. To force a field to be left-justified, place a minus sign immediately after the % sign. This feature is often useful for printing strings that you want lined up with each other on the left.

- **Short and long modifiers** – By default, output of the integer types assume int or unsigned int. Modifiers may be added to the integer format conversion specifiers (%d, %I, %o, %u, %x, %X) to indicate short, long, and long long sizes.:
  - Adding an h immediately before the format conversion letter prints a short (or unsigned short).
  - Adding an l (lowercase L) immediately before the format conversion letter prints a long (or unsigned long).
  - Adding an ll (two lowercase Ls) immediately before the format conversion letter prints a long long (or unsigned long long).

Dynamic Output Control at Run Time

In a situation where you don't know the minimum field-width or precision at compile time, you can inject them into your format strings at run time with the following technique.

Using asterisks * as placeholders within the format conversion specifier, you can pass the minimum field-width specifier and precision modifier to the format string dynamically at run time. To do this, insert an asterisk where the minimum field-width specifier would go (immediately after the %), and another asterisk where the precision modifier would go (immediately after the decimal point). Then, pass two additional integer arguments to printf, one for the minimum field-width specifier and one for the precision modifier. (Indicate left-justification by passing a negative minimum field-width specifier.) For example, if you want to print the value of double variable x, and you have two other integer variables minfw and precmod containing the minimum field-width specifier and precision modifier, then the call to printf would look like this:

```
printf("%*.*f\n", minfw, precmod, x);
```

In this function call, the value of minfw gets plugged into the first *, the value of precmod gets plugged into the second *, and x is printed using the resulting %f format. Let's say minfw is 15 and precmod is 6. Then the above statement would print x using the format conversion specifier "%15.6f". So, the equivalent printf call would be:

```
printf("%15.6f\n", x);
```

All of these features are available in the printf function (print to console output), the fprintf function (print to a file), and the sprintf function (store output into a string).