

C++ Notes: I/O Manipulators

Manipulators are the most common way to control output formatting.

#include <iomanip>

I/O *manipulators* that take parameters are in the <iomanip> include file.

Default Floating-point Format

Unless you use I/O manipulators (or their equivalent), the default format for each floating-point number depends on its value.

- No decimal point: 1.0000 prints as 1
- No trailing zeros: 1.5000 prints as 1.5
- Scientific notation for large/small numbers: 1234567890.0 prints as 1.23457e+09

I/O Manipulators

The following output manipulators control the format of the output stream. Include <iomanip> if you use any manipulators that have parameters. The Range column tells how long the manipulator will take effect: *now* inserts something at that point, *next* affects only the next data element, and *all* affects all subsequent data elements for the output stream.

Manip.	Rng	Description
<i>General output</i>		
<code>endl</code>	<i>now</i>	Write a newline ('\n') and flush buffer.
<code>setw(n)</code>	<i>next</i>	Sets minimum field width on output. This sets the minimum size of the field - a larger number will use more columns. Applies only to the next element inserted in the output. Use <code>left</code> and <code>right</code> to justify the data appropriately in the field. Output is right justified by default. Equivalent to <code>cout.width(n)</code> ; To print a column of right justified numbers in a seven column field: <pre>cout << setw(7) << n << endl;</pre>
<code>width(n)</code>	<i>next</i>	Same as <code>setw(n)</code> .
<code>left</code>	<i>next</i>	Left justifies output in field width. Only useful after <code>setw(n)</code> .
<code>right</code>	<i>next</i>	Right justifies output in field width. Since this is the default, it is only used to override the effects of <code>left</code> . Only useful after <code>setw(n)</code> .
<code>setfill(ch)</code>	<i>all</i>	Only useful after <code>setw</code> . If a value does not entirely fill a field, the character <i>ch</i> will be used to fill in the other characters. Default value is blank. Same effects as <code>cout.fill(ch)</code> For example, to print a number in a 4 character field with leading zeros (eg, 0007): <pre>cout << setw(4) << setfill('0') << n << endl;</pre>
<i>Floating point output</i>		
<code>setprecision(n)</code>	<i>all</i>	Sets the number of digits printed to the right of the decimal point. This applies to all subsequent floating point numbers written to that output stream. However, this won't make floating-point "integers" print with a decimal point. It's necessary to use <code>fixed</code> for that effect. Equivalent to <code>cout.precision(n)</code> ;
<code>fixed</code>	<i>all</i>	Used fixed point notation for floating-point numbers. Opposite of <code>scientific</code> . If no precision has already been specified, it will set the precision to 6.
<code>scientific</code>	<i>all</i>	Formats floating-point numbers in scientific notation. Opposite of <code>fixed</code> .
<i>bool output</i>		
<code>boolalpha</code> <code>noboolalpha</code>	<i>all</i>	Uses alphabetic representation (<code>true</code> and <code>false</code>) for <code>bool</code> values. Turned off with <code>noboolalpha</code> .
<i>Input</i>		
<code>skipws</code> <code>noskipws</code>	<i>all</i>	For most input values (eg, integers and floating-point numbers), skipping initial whitespace (eg, blanks) is very useful. However, when reading characters, it is often desired to read the whitespace characters as well as the non-spacing characters. The these I/O manipulators can be used to turn whitespace skipping off and on. Eg, <pre>cin >> noskipws;</pre> turns whitespace skipping off for all subsequent <code>cin</code> input.
<code>ws</code>	<i>now</i>	Reads and ignores whitespace at the current position.

<i>Other</i>		
		showpoint, noshowpoint, uppercase, nouppercase, dec, oct, hex, setbase(8/10/16), showbase, noshowbase, ends, showpos, noshowpos, internal, flush, unitbuf, nunitbuf, setiosflags(f), resetiosflags(f)

Example

```
#include <iostream>
#include <iomanip>
using namespace std;
int main() {
    const float tenth = 0.1;
    const float one   = 1.0;
    const float big   = 1234567890.0;

    cout << "A. " << tenth << ", " << one << ", " << big << endl;
    cout << "B. " << fixed << tenth << ", " << one << ", " << big << endl;
    cout << "C. " << scientific << tenth << ", " << one << ", " << big << endl;
    cout << "D. " << fixed << setprecision(3) << tenth << ", " << one << ", " << big << endl;
    cout << "E. " << setprecision(20) << tenth << endl;
    cout << "F. " << setw(8) << setfill('*') << 34 << 45 << endl;
    cout << "G. " << setw(8) << 34 << setw(8) << 45 << endl;

    return 0;
}
```

produces this output (using DevC++ 4.9.8.0):

```
A. 0.1, 1, 1.23457e+009
B. 0.100000, 1.000000, 1234567936.000000
C. 1.000000e-001, 1.000000e+000, 1.234568e+009
D. 0.100, 1.000, 1234567936.000
E. 0.10000000014901161
F. *****3445
G. *****34*****45
```

Lines F and G show the scope of `setw()` and `setfill()`.

[Copyleft](#) 2000-2003 [Fred Swartz](#) Last update 2003-08-24, URL=undefined