C++ Notes: I/O Manipulators

Manipulators are the most common way to control output formating.

#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
General output		
`endl`	now	Write a newline ('\n') and flush buffer.
`setw(`*`n`*`)`	next	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 `left` and `right` to justify the data appropriately in the field. Output is right justified by default. Equivalent to cout.width(*n*); To print a column of right justified numbers in a seven column field: `cout << setw(7) << n << endl;`
`width(n)`	next	Same as `setw(n)`.
`left`	next	Left justifies output in field width. Only useful after `setw(n)`.
`right`	next	Right justifies output in field width. Since this is the default, it is only used to override the effects of `left`. Only useful after `setw(n)`.
`setfill(`*`ch`*`)`	all	Only useful after `setw`. If a value does not entirely fill a field, the character *ch* will be used to fill in the other characters. Default value is

		blank. Same effects as `cout.fill(ch)` For example, to print a number in a 4 character field with leading zeros (eg, 0007): `    cout << setw(4) << setfill('0') << n << endl;`
Floating point output		
`setprecision(`*`n`*`)`	all	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 `fixed` for that effect. Equivalent to cout.precision(*n*);
`fixed`	all	Used fixed point notation for floating-point numbers. Opposite of `scientific`. If no precision has already been specified, it will set the precision to 6.
`scientific`	all	Formats floating-point numbers in scientific notation. Opposite of `fixed`.
`bool` output		
`boolalpha` **`noboolalpha`**	all	Uses alphabetic representation (`true` and `false`) for `bool` values. Turned off with `noboolalpha`.
Input		
`skipws` **`noskipws`**	all	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, `cin >> noskipws;` turns whitespace skipping off for all subseqent `cin` input.
`ws`	now	Reads and ignores whitespace at the current position.
Other		
		showpoint, noshowpoint, uppercase, nouppercase, dec, oct, hex, setbase(*8*|*10*|*16*), showbase, noshowbase, ends, showpos, noshowpos, internal, flush, unitbuf, nounitbuf, 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. 0.1000000014901161
F. ******3445
G. ******34******45
```

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