

7.2 Formatting with Manipulators (brief introduction)

Reference: Lippman and Lajoie, sec 20.9

- There are two ways to control the formatting of values written to an output stream:
 - i) By invoking methods of the stream directly,
 - ii) By using manipulators (which in turn invoke methods of the stream).
- ▶ Usually, it is more convenient to use manipulators.
- ▶ We have already seen several manipulators: endl, setw(), hex, dec.


```
cout << setw(5) << i << endl;
```
- ▶ Manipulators are most often used with output streams, but some apply to input streams as well.
 - We will consider only output streams.

- Here are some of the more useful manipulators.

Manipulator	Meaning
<code>endl</code>	Write a newline character, and then flush output buffer.
<code>setw(<i>w</i>)</code>	Display the next value written in field of width <i>w</i> . (<i>w</i> is a minimum width; a wider field will be used, if needed.) <i>Default:</i> 0 (reset to 0 after each value written). Note 0 says: use the smallest width that will hold the value — no padding.
<code>setfill(<i>c</i>)</code>	Use character <i>c</i> as the padding character. (Padding occurs only if <code>setw()</code> has been used to specify a field width larger than needed.) <i>Default:</i> blank.
<code>setprecision(<i>p</i>)</code>	For floating point numbers, display <i>p</i> digits (fixed format) or <i>p</i> digits after the decimal point (scientific format). <i>Default:</i> 6.

<code>left</code> <code>right</code>	Left or right justify value in its field. (No effect unless <code>setw()</code> has been used to specify a field width larger than needed.) <i>Default:</i> right (typically not what we want, for strings).
<code>hex</code> <code>dec</code>	Display integers in hexadecimal or decimal. <i>Default:</i> decimal.
<code>showbase</code> <code>noshowbase</code>	For hexadecimal integers, display the base explicitly (e.g., <code>0x2E4A</code>), or don't display it (e.g., <code>2E4A</code>). <i>Default:</i> Don't display base.
<code>showpoint</code> <code>noshowpoint</code>	For floating point numbers, always show <i>p</i> decimal digits (<i>p</i> = precision), or omit trailing zeroes possibly the decimal point (e.g., <code>2.000000</code> or <code>2.5173000</code> or <code>5.173</code>). <i>Default:</i> Omit trailing zeroes.
<code>fixed</code> <code>scientific</code>	Used fixed point or scientific (exponential) notation for floating point numbers. <i>Default:</i> depends on size of number.

- Some of these manipulators were added to C++ with the ANSI standard.
 - ▶ Some compilers (including g++, version 2.95) don't yet support them.
 - ▶ Here are some substitutes:

<code>left</code>	→	<code>setiosflags(ios::left)</code>
<code>right</code>	→	<code>resetiosflags(ios::left)</code>
<code>showbase</code>	→	<code>setiosflags(ios::showbase)</code>
<code>noshowbase</code>	→	<code>resetiosflags(ios::showbase)</code>
<code>showpoint</code>	→	<code>setiosflags(ios::showpoint)</code>
<code>noshowpoint</code>	→	<code>resetiosflags(ios::showpoint)</code>
<code>scientific</code>	→	<code>setiosflags(ios::scientific)</code>
<code>fixed</code>	→	<code>resetiosflags(ios::scientific)</code>
- *Note:* Except for `setw`, the formatting specified by a manipulator *remains in effect, until cancelled*.
 - ▶ However, it applies only to the specific stream.
 - ▶ For example, `cout << left` does not effect formatting of values written to `cerr`.

Using Manipulators to Format Strings		
Source Code	Output	Comment
<code>cout << "abc" << endl;</code>	abc	Minimum width that will hold value (3).
<code>cout << setw(6) << "abc" << endl;</code>	___abc	<code>setw()</code> specifies field width. (<code>_</code> denotes blank.)
<code>cout << "abc" << endl;</code>	abc	Width reset to 0 after each value printed.
<code>cout << left << setw(6) << "abc" << endl;</code>	abc___	Left justify (pad on right).
<code>cout << setw(6) << "abc" << endl;</code>	abc___	Left justification remains in effect
<code>cout << right << setfill('*') << setw(6) << "abc" << endl;</code>	***abc	Right justify (pad on left). Pad with asterisks.
<code>cout << setw(6) << "abc" << endl;</code>	***abc	Fill character remains in effect.

Using Manipulators to Format Integers	
Source Code	Output
<code>cout << 46 << endl</code>	46
<code><< hex << 46 << endl</code>	2e
<code><< setfill('0') << setw(8) << 46 << endl</code>	0000002e
<code><< showbase << 46 << endl</code>	0x2e
<code><< uppercase << 46 << endl</code>	0X2E
<code><< dec << 46 << endl</code>	46
<code><< nouppercase << noshowbase << setfill(' ');</code>	
<code>for (int i = 1 ; i < 15 ; i += 2) cout << left << setw(5) << i*i*i << right << setw(5) << i*i*i << endl;</code>	1 1 27 27 125 125 343 343 729 729 1331 1331 2197 2197

Using Manipulators to Format Floating Point Numbers	
Source Code	Output
<code>cout << 271.0 << endl << 4.862 << endl << 0.0329856943 << endl << showpoint << 271.0 << endl << 4.862 << endl << 0.0329856943 << endl << setprecision(3) << 271.0 << endl << 4.862 << endl << 0.0329856943 << endl << scientific << 271.0 << endl << 4.862 << endl << 0.0329856943 << endl;</code>	271 4.862 0.0329857 271.000 4.86200 0.0329857 271. 4.86 0.0330 2.710e+02 4.862e+00 3.299e-02
<code>cout << fixed << setprecision(6); double x = 1; for (int n = 1 ; n < 7 ; ++n) { x = 3.1415926 * x; cout << setw(12) << x << endl; } x = 1; for (int n = 1 ; n < 7 ; ++n) { x = 3.1415926 * x; int p = 6 + (x>=10) + (x>=100); cout << setprecision(p) << setw(12) << x << endl; }</code>	3.14159 9.86960 31.0063 97.4091 306.020 961.389 3.14159 9.86960 31.00628 97.40908 306.01966 961.38910

- Manipulators are not really a built-in feature of C++.
- ▶ Rather the C++ library provides the standard manipulators, implementing them via classes.
- ▶ A programmer can define additional manipulators.
- ▶ Here is one way in which the standard library could implement `setw`. (Not the way actually used.)

```
// Definition of the class setw.
class setw {
private:
    unsigned width;
public:
    setw( unsigned w ) { width = w; }
    unsigned getWidth() const {return width; }
};

// Overload operator << for left operand an ostream, right operand
// a setw, so it invokes the ostream's width() method
ostream &operator<<( ostream &os, setw s ) {
    os.width( s.getWidth() );
    return os;
}
```

- ▶ Now `setw(7)` is a temporary object of type `setw`, created by the `setw` constructor.