Formatting – Text and Numbers

Formatting output means controlling how information is printed or written to the screen. Note that it is quite awkward in a language like C++ to produce nicely formatted output. In practice we would usually C++ to produce the desired results in a very simple format, and then transfer that output to another program (ex: word processor or spreadsheet) to format it better.

Nonetheless, some formatting of our program's output can be beneficial. To do so, we will use manipulators. We have already used one manipulator **endl** (from the *iostream* header file) which causes subsequent output to begin in column 1 of the next line.

The following manipulators require the use of the header file *iomanip*. With the exception of the manipulator **setw**, after a manipulator is used, all subsequent input or output is formatted accordingly. The manipulator **setw** only applies to the next item to be output.

Input/Output Manipulators (using *iomanip* header file)

- **setw(n)**

You can display data in table format by using the **setw** function. This manipulator sets the field width to n. In other words, it causes the next output value to be displayed using the # of character positions designated by n.

Example:
```
cout << setw(5) << celsius;          // assigns 5 spaces for the value of celsius
```

The values printed in the field are right-justified by default and if the number is less than the number of spaces assigned, blanks will appear before the number. If the number is too large to fit in the field, cout will still print the entire number. **setw**() only specifies the minimum number of positions in the print field. Any number larger than the minimum will cause **cout** to override the **setw** value.

Example:

```
for ( i = 1; i <=1000; i *=10 )
        cout << setw(4) << i << endl;
```

This code prints the numbers 1, 10 , 100, and 1000 right-justified in a field of width 4:

```
   1
  10
 100
1000
```

NOTE: the **setw** manipulator only works with the value immediately following it. After that, cout goes back to is default method of printing; so you must use **setw**() for each value you want to align.

- **left** and **right**: alignment options with **setw(n)**

Since the default alignment is right-aligned with **setw**, you can use 2 other manipulators **left** and **right** to change the alignment. Both of these tags remain in effect until they are changed.

Example:

cout << **left** << **setw**(10) << name << **setw**(10) << plan << **setw**(5) << group;

and THEN when you want to display values or numbers where you would like them to be right aligned, you would type:

cout << **right** << **setw**(6) << cost;

Example:

```
int a = 5, b = 43, c = 104;
cout << left << setw(10) << "Karen"
     << right << setw(6) << a << endl;
cout << left << setw(10) << "Ben"
     << right << setw(6) << b << endl;
cout << left << setw(10) << "Patricia"
     << right << setw(6) << c << endl;
```

The output is:

```
Karen          5
Ben           43
Patricia     104
```

In other words, the code prints names, left-justified in a field of width 10; and prints numbers, right-justified in a field of width of 6.

- **setprecision(n)**

Floating point values may be rounded to a number of significant digits, which is the total number of digits that appear before and after the decimal point. The manipulator **setprecision**(n) specifies that n digits of precision be used for floating-point numbers. Unlike setw, the precision setting remains in effect until it is changed to some other value. Therefore you can also set the precision by itself prior to outputting floating point values.

If the output is less than the total number of precision digits allotted, it will round the number. However, if the total number of digits is less than the precision setting, it will not add zeros.

Example:

```
double num1 = 34.5278;
cout << num1 << endl;
cout << setprecision(5) << num1 << endl;
cout << setprecision(3) << num1 << endl;
cout << setprecision(2) << num1 << endl;
```

Output:

```
34.5278
34.528
34.5
35
```

Example:

```
double val1= 45.678, val2 = 23.4;
cout << setprecision(4);
cout << setw(8) << val1 << endl;
cout << setw(8) << val2 << endl;
```

Output:

```
  45.68
    23.4
```

- **showpoint**

By default, floating point numbers are not displayed with trailing zeros, and floating-point numbers that do not have a fractional part are not displayed with a decimal point. You can use **showpoint** to always display the decimal point and trailing zeros. **showpoint** is used in combination with **setprecision**( ) for total decimal settings to be displayed.

Example:

```
double x = 123.4, y = 456;
cout << setprecision(6) << showpoint << x << endl;
cout << y << endl;
```

Output:

```
123.400
456.000
```

NOTE: This works with most, but not all, compilers so check to make sure.

- **fixed**

When the **fixed** manipulator is used, all floating point numbers printed are displayed in fixed point notation (d.ddd) , with the number of digits to the right of the decimal point specified by the **setprecision** manipulator.

Example:

```
cout << setprecision(2) << fixed;
double x = 123.4567;
cout << x << endl;
```

Output:

123.46