C++ Stream Output

Introduction

Many high level languages have input and output routines built into the language itself. However, in C and C++, I/O facilities are technically not part of the language itself. To get output, programmers must use I/O functions in libraries that are provided by compiler writers. Fortunately, these libraries are standardized and programmers can expect to see pretty much the same I/O functions in the various C++ compilers.

iostream

C++ compilers provide a header file, "`iostream.h`", that contains various function prototypes, including the operator "<<", which is called the *insertion* operator or *put to* operator. This operator can handle the output of all built-in C++ data types and can be overloaded to handle output of user-defined types. When a program that uses "iostream" is executed, an object in *ostream* called *cout* is created. *cout* stands for *console output*, and by default is connected to the standard output device which is normally the screen. C++ also has one called *cerr*, which stands for *console error*. A programmer can then output the value of a variable by using the << operator to send the value of the variable to *cout*. The object *cout* will respond by displaying the value of the variable on the console screen.

To use this header file one needs to "include" this file at the top of the program. This is done with the statement "`#include <iostream.h>`". If one is using Visual C++ .NET then the line is "`#include <iostream>`" followed by the line "`using namespace std;`", note in .NET some header files do not need the **.h** after the file name.

Examples

```
cout << 123;            // output "123" (no leading/trailing blanks)
cout << "Hello World!"; // output "Hello World!"
cout << 'A';            // output "A"
```

Floating point types have sensible default formatting. For example,

```
cout << 1.23;         // output "1.23"
cout << 0.23;         // output "0.23"
cout << 10.0003;      // output "10.0003"
cout << 1.99999999;   // output "2"
cout << 1.23456789;   // output "1.234568"
cout << 123456789.0;  // output "1.234568e+09"
```

IO Manipulators

If you wish to depart from the *ostream* defaults, you can control output by using *I/O manipulators*. Here are some simple ones that are defined in header file "iostream.h".

endl	: Sends a newline character to stdout and flushes output buffer.
flush	: Flushes the output buffer.
dec	: Sets flag that causes subsequent integer output to be in base 10 (decimal)
hex	: Sets flag that causes subsequent integer output to be in base 16 (hexadecimal)
oct	: Sets flag that causes subsequent integer output to be in base 8 (octal)

A library with header file named "`iomanip.h`" contains several additional manipulators. These allow one to do such things as control field widths and floating point format. Unlike the manipulators in *iostream.h*, these require parameters to specify their actions. Some of the more useful of these are described below.

setw

The *setw* manipulator is used to set field widths. Some examples follow:

```
cout << setw(5) << 123;     // output: "  123"
cout << setw(6) << "Hello"; // output: " Hello"
cout << setw(1) << 123;     // output: "123"
cout << setw(3) << 1 << 8;  // output: "  18"
```

Note that if the width provided by *setw* is to small, it is ignored. The last example shows that *setw* only affects the next item sent to *cout*. However, the effects of the other manipulators described above are permanent, for example, once *hex* output is specified, it remains in effect until decimal or octal output is specified.

setfill

The default "fill" character is the blank, i.e. if *setw* is called with a width larger than the number of columns needed, the extra places are filled with blanks. *setfill* allows you to use any character for the filling.

```
cout << setfill('X') << setw(5) << 123;  // output: "XX123"
```

setprecision

This manipulator puts an upper bound on the number of digits displayed to the right of the decimal point.

```
cout << setprecision(2) << 1.234;    // output: "1.23"
cout << 0.1234;     // output: "0.12" setprecision(2) still in effect
```

The examples below show that *setprecision* does not force the display of digits to the right of the decimal point.

```
cout << setprecision(2) << 1234567890;   // output: "1.23e+09"
cout << setprecision(2) << 1.999;        // output: "2"
```

setiosflags

Sending *setprecision(2)* to *cout* only sets the maximum number of displayed digits after the decimal point to be 2. In order to force the display of digits to the right of the decimal point, an additional manipulator, *setiosflags()*, is needed. This function allows access to a bit string that controls input and output. A typical call to *setiosflags* would be of the form "*setiosflags(constant)*", where *constant* is a constant defined in *iostream.h*.

Using setiosflags to Control Floating Point Output

The chart below gives a summary of some of the more useful calls to *setiosflags()*.

Call	Effect
`setiosflags(ios::fixed)`	Turns off scientific notation, forces fixed point display.
`setiosflags(ios::scientific)`	Use scientific notation.
`setiosflags(ios::showpoint)`	Forces display of decimal point and the display of digits specified by setprecision.
`setiosflags(ios::left)`	Left justify output.
`setiosflags(ios::right)`	Right justify output.

Examples (these assume default iosflags settings are in effect at first)

```
cout << 1234567890;          //Default: scientific notation "1.23e+09"
cout << 1.999;               //Default: don't always show decimal point "2"
cout << setprecision(2) << 1.9;                  //output: "1.9"
cout << setiosflags(ios::fixed) << 1234567890;   //output: "1234567890"
cout << setiosflags(ios::showpoint) << 1.999     //output: "2.000000"
     << setprecision(3) << 1.9;                  //output: "1.900"
```

From the examples above, we see that a combination of *setw*, *setprecision* and *setiosflags* must be used in order to completely control the formatting of floating point numbers. Below is a program segment that shows how to force the display of the decimal point and two digits to the right of the decimal point.

```
cout << setiosflags(ios::fixed) << setiosflags(ios::showpoint)
     << setprecision(2);
cout << 12.0;               //output: "12.00"
cout << 1234567890.0;       //output: "1234567890.00"
cout << 1.999;              //output: "2.00"
```

The calls

```
cout << setiosflags(ios::fixed)      and
cout << setiosflags(ios::showpoint)
```

can be combined and made into the single call

```
cout << setiosflags(ios::fixed | ios::showpoint);
```

or

```
cout << setf(ios::fixed | ios::showpoint);
```

Some ostream Member Functions

In the examples below, where the prototype only is given, these member functions are called with an o*stream object* followed by *dot operator* then *member name*, example is "`cout.put(Ch)`". Also assume that the following declarations have been made.

```
char Ch;
char Str[100];
```

`ostream& **put**(char Ch)`

Writes a character to the appropriate output device or file. For example, "`cout.put('A');`" will write the letter 'A' to the output screen. Another example, if *FileOut* is a member of *ofstream*, then "`FileOut.put(Ch);`" writes the character stored in Ch to the file associated with the variable *FileOut*.

`ostream& **write**(const char *Str, int NumberofChars)`

Writes *NumberofChars* characters from the string *Str* to the associated output device, either standard output device , i.e. the monitor, or an output file. For example, "`cout.write(Str, 20);`" writes the first 20 characters of *Str* to the screen.

`ostream& **flush**()`

When a file is opened for output, a file buffer is set up in main memory. Writes to the file are actually sent to the file buffer. When the buffer fills up, it is written to the disk drive or other secondary storage device. Flush forces an immediate write of the information in the buffer to the disk.