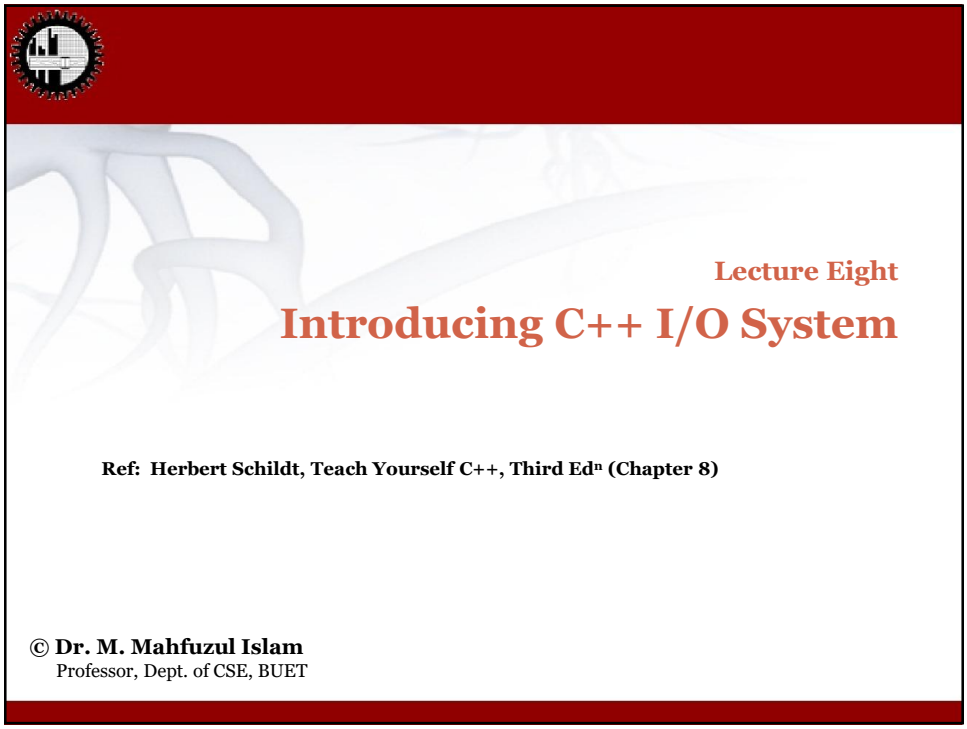
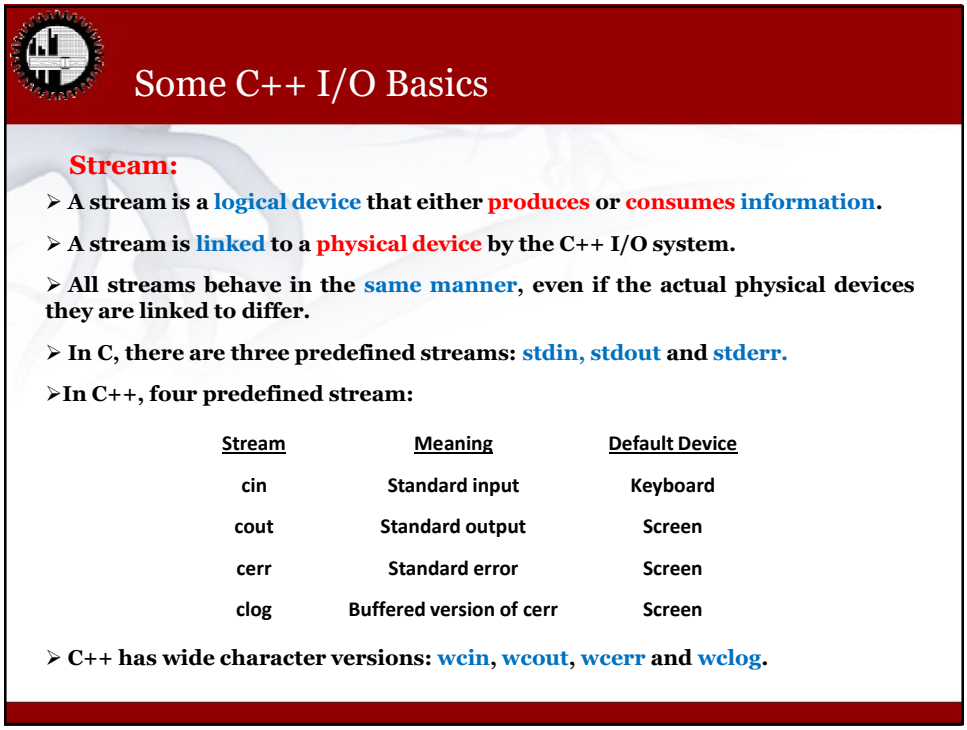


Lecture Eight

Introducing C++ I/O System

Ref: Herbert Schildt, Teach Yourself C++, Third Edn (Chapter 8)

© Dr. M. Mahfuzul Islam
Professor, Dept. of CSE, BUET



Some C++ I/O Basics

Stream:

- A stream is a logical device that either produces or consumes information.
- A stream is linked to a physical device by the C++ I/O system.
- All streams behave in the same manner, even if the actual physical devices they are linked to differ.
- In C, there are three predefined streams: stdin, stdout and stderr.
- In C++, four predefined stream:

Stream	Meaning	Default Device
cin	Standard input	Keyboard
cout	Standard output	Screen
cerr	Standard error	Screen
clog	Buffered version of cerr	Screen

- C++ has wide character versions: wcin, wcout, wcerr and wclog.

Some C++ I/O Basics

Template Class and I/O Classes:

- In the header file <iostream>, a complicated set of class hierarchies is defined.
- The I/O classes begin with a system of template classes. A template class defines the form of a class without fully specifying the data upon which it operates.
- C++ has two versions of template class: one for 8-bit character and another for wide character.
- C++ I/O system is build upon two related but different template class hierarchies: basic_streambuf and basic_ios.

Template Class	8-bit character based class
basic_streambuf	streambuf
basic_ios	ios

Some C++ I/O Basics

- basic_streambuf supplies basic, low-level input and output operations and provides the underlying support for the entire I/O system. basic_streambuf is needed for advanced I/O programming.
- basic_ios: A high level class that provides formatting, error checking and status information related to stream I/O.

basic_ios is used as a base of several derived classes, including basic_istream, basic_ostream and basic_iostream

Template Class	8-bit character based class
basic_istream	istream
basic_ostream	ostream
basic_iostream	iostream
basic_fstream	fstream
basic_ifstream	ifstream
basic_ofstream	ofstream

Including <iostream> in a program, thus, allow access to ios class.

Formatting I/O

- Each stream has associated with a set of format flags that control the way the information is formatted.
- The ios class declares a bitmask enumeration called fmtflags, in which the following values are defined to set or clear format flags.

Collective value	Individual value	Purpose
adjustfield	Left	Output is left justified.
	Right	Output is right justified.
	internal	A numeric value is padded to fill a field by inserting spaces between any sign or base character.
	not set	Default output is right justified.
basefield	Oct	Display output in Octal.
	Dec	To return output to Decimal.
	Hex	Display output in Hexadecimal.
floatfield	Scientific	Display floating point values in scientific notation.
	Fixed	Display floating point values in normal notation.

Formatting I/O

boolalpha	Booleans can be input or output using the keyword true or false.
skipws	Leading whitespace characters (spaces, tabs and newlines) are discarded when input is being performed in a stream.
showbase	The base of numeric values are displayed. For example, if the conversion base is hexadecimal, 1F is displayed as 0x1F.
showpoint	A decimal point and tailing zeros to be displayed for all floating point output - whether needed or not.
showpos	A leading plus sign (+) to be displayed before positive values. This only affects on decimal output.
unitbuf	The buffer is flashed after each insertion operation.
uppercase	The character are displayed uppercase.

Formatting I/O

- To set or clear one or more format flags, the function setf() and unsetf() are used. Both are the members of ios. The general format are:

```
fmtflags setf(fmtflags flags);
void unsetf(fmtflags flags);
```

An example to set the showbase and hex flags for cout:

```
cout.setf( ios::showbase | ios::hex);
```

As showbase is an enumerated constant within the ios class, the scope resolution operator is used to tell compiler the class name; otherwise, showbase will not be recognized.

- The member function of ios class flags() returns the current setting of each format flag. Its prototype is:

```
fmtflags flags();
```

Example: `ios::fmtflags f = cout.flags();`

The second form of flags() function set all format flags, its prototype is as follows:

```
fmtflags flags(fmtflags f);
```

- For some compilers, the dec flag overrides the other flags, so it is necessary to turn it off when turning on either hex or oct.

Formatting I/O

```
#include <iostream>
using namespace std;

int main(){
    // display using default setting
    cout << 123.23 << " hello " << 100 << '\n';
    cout << 10 << ' ' << -10 << '\n';
    cout << 100.0 << "\n\n";

    cout.unsetf( ios::dec );
    cout.setf(ios:: hex | ios::scientific);
    cout << 123.23 << " hello " << 100 << '\n"
    cout.setf( ios::showpos );
    cout << 10 << ' ' << -10 << '\n';
    cout.setf( ios::showpoint | ios::fixed);
    cout << 100.0;

    return 0;
}
```

OUTPUT:

```
123.23 hello 100
10  -10
100
1.232300e+02 hello 64
A fffffff6
+100.000000
```

Formatting I/O

➢**The member function of ios class flags() returns the current setting of each format flag. Its prototype is:**

fmtflags flags();

Example: ios::fmtflags f = cout.flags();

The second form of flags() function set all format flags, its prototype is as follows:

fmtflags flags(fmtflags f);

➢**For some compilers, the dec flag overrides the other flags, so it is necessary to turn it off when turning on either hex or oct.**

```
#include <iostream>
using namespace std;

void showflags();
```

```
int main(){
    showflags();
    cout.setf(ios::oct | ios::showbase | ios::fixed);
    showflags();
    ios::fmtflags f = ios::showpos | ios::showbase
                    | ios::oct | ios::right;
    cout.flags(f);              // set flags
    showflags();

    return 0;
}
```

Formatting I/O

```
void showflags(){
ios::fmtflags f = cout.flags();    //get flag settings

if ( f & ios::skipws ) cout << "skipws on\n";
else cout << "skipws off\n";
if ( f & ios::left ) cout << "left on\n";
else cout << "left off\n";

if ( f & ios::right ) cout << "right on\n";
else cout << "right off\n";

if ( f & ios::internal ) cout << "internal on\n";
else cout << "internal off\n";

if ( f & ios::dec ) cout << "dec on\n";
else cout << "dec off\n";

if ( f & ios::oct ) cout << "oct on\n";
else cout << "oct off\n";

if ( f & ios::hex ) cout << "hex on\n";
else cout << "hex off\n";

if ( f & ios::showbase ) cout << "showbase on\n";
else cout << "showbase off\n";
```

```
 if ( f & ios::showpoint ) cout << "showpoint on\n";
else cout << "showpoint off\n";

if ( f & ios::showpos ) cout << "showpos on\n";
else cout << "showpos off\n";

if ( f & ios::uppercase ) cout << "uppercase on\n";
else cout << "uppercase off\n";

if ( f & ios::scientific ) cout << "scientific on\n";
else cout << "scientific off\n";

if ( f & ios::fixed ) cout << "fixed on\n";
else cout << "fixed off\n";

if ( f & ios::unitbuf ) cout << "unitbuf on\n";
else cout << "unitbuf off\n";

if ( f & ios::boolalpha ) cout << "boolalpha on\n";
else cout << "boolalpha off\n";

cout << '\n';
}
```

Formatting I/O

OUTPUT:

skipws on	skipws on	skipws on
left off	left off	left off
right off	right off	right on
internal off	internal off	internal off
dec on	dec on	dec on
oct off	oct on	oct on
hex off	hex off	hex off
showbase off	showbase on	showbase on
showpoint off	showpoint off	showpoint off
showpos off	showpos off	showpos on
uppercase off	uppercase off	uppercase off
scientific off	scientific off	scientific off
fixed off	fixed on	fixed on
unitbuf off	unitbuf off	unitbuf off
boolalpha off	boolalpha off	boolalpha off

Using width(), precision() and fill()

➢**In addition to the formatting flags, there are 3 member functions of ios for setting format parameters: width(), precision() and fill().**

Prototypes of these 3 functions are as follows:

```
streamsize width(streamsize w);        // default minimum size
streamsize precision(streamsize p);    //default six digits
char fill(char ch);                    // default spaces
```

In all cases, the new value set to the given parameter (i.e., w, p or ch) and the old value is returned.

✓**streamsize is defined in ios as some form of integer.**

Using width(), precision() and fill()

- **Field width is set before each output statement.**

```
#include <iostream>
using namespace std;

int main(){
  cout.width(10);
  cout << "hello" << '\n';

  cout.fill('%');
  cout.width(10);
  cout << "hello" << '\n';

  cout.setf(ios::left);
  cout.width(10);
  cout << "hello" << '\n';

  cout.width(10);
  cout.precision(10);
  cout << 123.234567 << '\n';

  cout.width(10);
  cout.precision(6);
  cout << 123.234567 << '\n';

  double x;
  cout.precision(4)
  cout << "   x    sqrt(x)   x^2\n\n";
  for( x =2.0; x<=5; x++){
      cout.width(7);
      cout << x << "   ";
      cout.width(7);
      cout << sqrt(x) << "   ";
      cout.width(7);
      cout << x*x << " \n ";
  }
  return 0;
}
```

OUTPUT:

```
     hello
%%%%%hello
hello%%%%%
123.234567
123.235%%%
```

x	sqrt(x)	x*x
2	1.414	4
3	1.732	9
4	2	16
5	2.236	25

Using I/O Manipulators

- **The second way of formatting I/O information is using special functions called *manipulators*.**
- **To access manipulators that takes parameters, such setw(), <iomanip> must be included. This is not necessary if the manipulator does not take arguments.**
- **If the manipulator does not take an argument, it is not followed by parenthesis, because it is the address of the manipulator that is passed to the overloaded << operator.**
- **Manipulators are easier to use and allow more compact code to be written.**
- **Specific format flags of ios class can be set using manipulator using setiosflags() and clear them using resetiosflags().**

Using I/O Manipulators

List of manipulators:

Manipulator	Purpose	Input/Output
boolalpha	Turn on boolalpha flag	Input/output
dec	Turn on dec flag	Input/output
endl	Output a newline character and flushes the stream	Output
ends	Output a null	Output
fixed	Turn on fixed flag	Output
flush	Flushes a stream	Output
hex	Turn on hex flag	Input/output
internal	Turn on internal flag	Output
left	Turn on left flag	Output
noboolalpha	Turn off boolalpha flag	Input/output
noshowbase	Turn off showbase flag	Output
noshowpoint	Turn off showpoint flag	Output
noshowpos	Turn off showpos flag	Output
noskipws	Turn off skipws flag	Input
nounitbuf	Turn off unitbuf flag	Output
nouppercase	Turn off uppercase flag	Output
oct	Turn on oct flag	Input/output

Using I/O Manipulators

resetiosflags(fmtflags f)	Turn off the flags specified in f	Input/output
right	Turn on right flag	Output
scientific	Turn on scientific flag	Output
setbase(int base)	Set the number base to base	Input/output
setfill(int ch)	Set the fill character to ch	Output
setiosflags(fmtflags f)	Turn on the flags specified in f	Input/output
setprecision(int p)	Set number of digits of precision	Output
setw(int w)	Set the field width to w	Output
showbase	Turn on showbase flag	Output
showpoint	Turn on showpoint flag	Output
showpos	Turn on showpos flag	Output
skipws	Turn on skipws flag	Input
unitbuf	Turn on unitbuf flag	Output
uppercase	Turn on uppercase flag	Output
ws	Skips leading white space	Input

Using I/O Manipulators

```
#include <iostream>
#include <iomanip>
using namespace std;

int main(){
    cout << hex << 100 << endl;
    cout << oct << 10 << endl;
    cout << setfill('X') << setw(10);
    cout << 100 << " hi " << endl;

    double x;
    cout << setprecision(4);
    cout << "      x    sqrt(x)    x^2\n\n";
    for( x = 2.0; x <= 5.0; x++ ){
        cout << setw(7) << x << "   ";
        cout<<setw(7)<<sqrt(x) << "   ";
        cout << setw(7) << x*x << endl;
   }
   return 0;
}
```

OUTPUT:

```
64
12
XXXXXXX144 hi
```

x	sqrt(x)	x*x
2	1.414	4
3	1.732	9
4	2	16
5	2.236	25

Creating your own Inserters

- **In C++, the output operation is called an insertion and the << is called the insertion operator.**
- **All insertion function or inserter has the following general form:**

```
ostream &operator << (ostream &stream, class-name ob){
    //body of inserter
    return stream;
}
```

- **The first parameter is a reference to output stream and the second parameter is the object to be displayed. The function returns an output stream.**
- **An inserter cannot be a member of a class, because for an operator member function the left operand must be an object which is implicitly passed through this pointer.**
- **An inserter can be a friend function.**
- **An inserter should be written as general as possible.**

Creating your own Inserters

Inserter using friend function.

```
#include <iostream>
using namespace std;

class coord {
     int x, y;
public:
    coord (int i = 0,  int j = 0) { x = i; y = j; }
    friend ostream &operator << ( ostream
                                  &stream, coord ob);
};

ostream &operator << ( ostream &stream,
                                      coord ob){
    stream<<ob.x<< ', '<<ob.y<<'\n';
    return stream;
}

int main(){
    coord a(1, 1), b(10, 23);
    cout << a << b;
    return 0;
}
```

Without using friend function.
Variable x and y must be public.

```
#include <iostream>
using namespace std;

class coord {
public:
    int x, y;
    coord (int i = 0,  int j = 0) { x = i; y = j; }
};

ostream &operator << ( ostream &stream,
                                      coord ob){
    stream<<ob.x<< ', '<<ob.y<<'\n';
    return stream;
}

int main(){
    coord a(1, 1), b(10, 23);
    cout << a << b;
    return 0;
}
```

Creating Extractor

- **In C++, the input operation is called an extraction and the >> is called the extraction operator.**
- **All insertion function or inserter has the following general form:**

```
istream &operator >> (istream &stream, class-name ob){
    //body of inserter
    return stream;
}
```

The first parameter is a reference to input stream and the second parameter is the object to be displayed. The function returns an input stream.

- **An extractor cannot be a member of a class, because for an operator member function the left operand must be an object which is implicitly passed through this pointer.**
- **An extractor can be a friend function.**
- **An extractor should be written as general as possible.**

Creating Extractor

```
#include <iostream>
using namespace std;

class inventory {
      char item[40];
      int onhand;
      double cost;
public:
      inventory (char *i, int o, double c ) {
            strcpy(item, i);
            onhand = o;
            cost = c;
      }
      friend ostream &operator << ( ostream
                              &stream, inventory ob);
      friend istream &operator >> ( istream
                             &stream, inventory &ob);
};

ostream &operator << ( ostream &stream,
                                    inventory ob){
    stream << ob.item << ': ' << ob.onhand;
    stream << " on hand at $" << ob.cost << '\n';
    return stream;
}
```

```
istream &operator >> ( istream &stream,
                                    inventory &ob){
      cout << "Enter the item name: ";
      stream >> ob.item;
      cout << "Enter number on hand: ";
      stream >> ob.onhand;
      cout << "Enter cost: ";
      stream >> ob.cost;

      return stream;
}

int main(){
       inventory ob("hammer", 4, 12.55);

      cout << ob;
      cin >> ob;
      cout << ob;

      return 0;
}
```

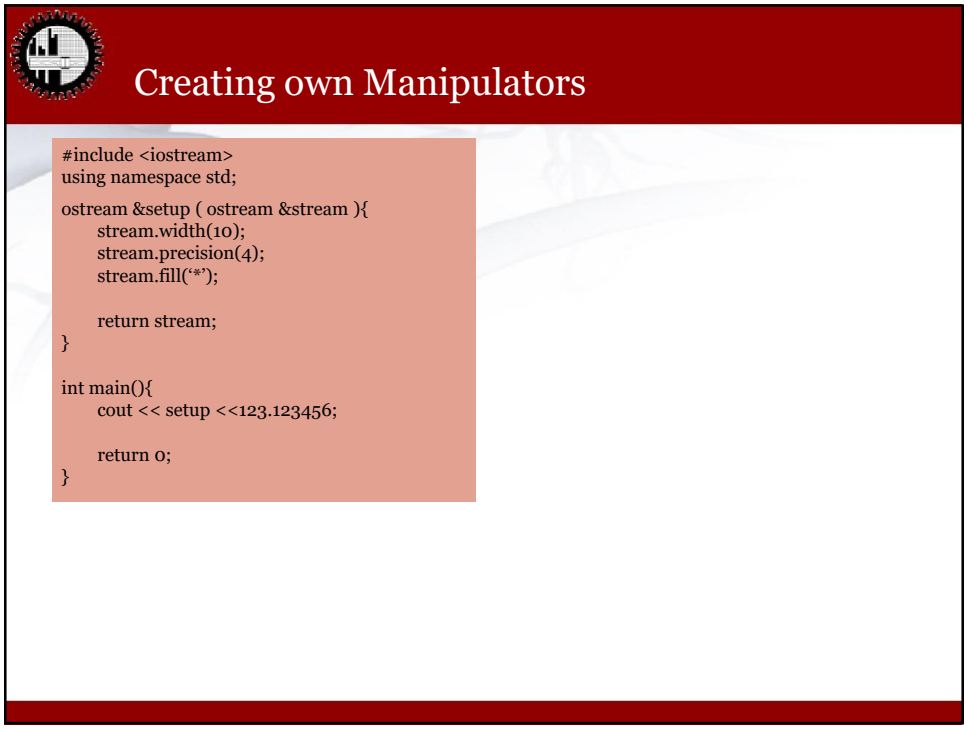
Creating own Manipulators

- **Custom manipulators are important for two reasons: (1) consolidate a sequence of several separate I/O operations. This simplifies the source code and prevents accidental errors; and (2) when need to perform I/O operations on a nonstandard device.**
- **Two types of manipulators: (1) those that operate on input stream; and (2) those that operate on output stream.**
- **Two types of manipulators: (1) manipulators taking arguments; and (2) manipulators not taking arguments.**
- **General form of all parameterless manipulator output function:**

```
ostream &manip-name(ostream &stream){
    // body
    return stream;
}
```

General form of all parameterless manipulator input function:

```
istream &manip-name(istream &stream){
    // body
    return stream;
}
```



Creating own Manipulators

```
#include <iostream>
using namespace std;

ostream &setup ( ostream &stream ){
      stream.width(10);
      stream.precision(4);
      stream.fill('*');

      return stream;
}

int main(){
      cout << setup <<123.123456;

      return 0;
}
```