Object Oriented Programming in C++

Segment-8

Course Code: 1221

Prepared by Muhammed Nazmul Arefin

Asst. Lecturer, CSE, IIUC

Contents

- Formatted I/O
- Using width(),precision(), and fill()
- Using I/O manipulators
- Creating own manipulators
- Creating extractor
- Creating own manipulators
- File I/O basics
- Unformatted, binary I/O
- Random access

I/O Basics

- When a C program begins execution, three predefined streams are automatically opened: **stdin, stdout,** and **sterr**
- When C++ program begins, these four streams are automatically opened:

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

The C++ I/O system is built upon two related, but different, template class hierarchies. The first is derived from the low-level I/O class called **basic_streambuf**. This class supplies the basic, low-level input and output operations and provides the underlying support for the entire C++ I/O system. Unless you are doing advanced I/O programming, you will not need to use **basic_streambuf** directly. The class hierarchy that you will most commonly be working with is derived from **basic_ios**. This is a high-level I/O class that provides formatting, error-checking, and status information related to stream I/O. **basic_ios** is used as a base for several derived classes, including **basic_istream**, **basic_ostream**, and **basic_iostream**. These classes are used to create streams capable of input, output, and input/output, respectively.

Formatted I/O

C++ helps you to format the I/O operations like determining the number of digits to be displayed after the decimal point, specifying number base etc.

Example:

If we want to add + sign as the prefix of out output, we can use the formatting to do so:
`stream.setf(ios::showpos)`
If input=100, output will be +100
If we want to add trailing zeros in out output to be shown when needed using the formatting:
`stream.setf(ios::showpoint)`
If input=100.0, output will be 100.000

There are two ways to do so:

1. Using the ios class or various ios member functions.
2. Using manipulators(special functions)

Formatted I/O

1. Formatting using the ios members:

The stream has the format flags that control the way of formatting it means Using this setf function, we can set the flags, which allow us to display a value in a particular format. The ios class declares a bitmask enumeration called **fmtflags** in which the values(showbase, showpoint, oct, hex etc) are defined. These values are used to set or clear the format flags.

Few standard ios class functions are:

- width(): The width method is used to set the required field width. The output will be displayed in the given width
- precision(): The precision method is used to set the number of the decimal point to a float value
- fill(): The fill method is used to set a character to fill in the blank space of a field
- setf(): The setf method is used to set various flags for formatting output
- unsetf(): The unsetf method is used To remove the flag setting

Formatted I/O

```
cout << "--------------------------\n";
cout << "Implementing ios::width\n\n";

char c = 'A';

// Adjusting width will be 5.
cout.width(5);

cout << c <<"\n";


int temp = 10;

// Width of the next value to be
// displayed in the output will
// not be adjusted to 5 columns.
cout<<temp;
cout << "\n--------------------------\n";
```

```
Implementing ios::width

    A
10
```

Formatted I/O

```
cout << "-------------------------\n";
cout << "Implementing ios::width\n\n";

char c = 'A';

// Adjusting width will be 5.
cout.width(5);

cout << c <<"\n";


int temp = 10;

// Width of the next value to be
// displayed in the output will
// not be adjusted to 5 columns.
cout<<temp;
cout << "\n-------------------------\n";
```

```
Implementing ios::width


    A
10
```

```
cout << "\n-------------------------\n";
cout << "Implementing ios::precision\n\n";
cout.setf(ios::fixed);
cout.precision(2);
cout<<3.1422;
cout << "\n-------------------------\n";
```

```
Implementing ios::precision


3.14
```

Formatted I/O

```
cout << "\n-------------------------\n";
cout << "Implementing ios::fill\n\n";
char ch = 'a';

// Calling the fill function to fill
// the white spaces in a value with a
// character our of choice.
cout.fill('*');

cout.width(10);
cout<<ch <<"\n";


int i = 1;

// Once you call the fill() function,
// you don't have to call it again to
// fill the white space in a value with
// the same character.
cout.width(5);
cout<<i;
cout << "\n-------------------------\n";
```

```
Implementing ios::fill

*********a
****1
```

Formatted I/O

```
cout << "\n--------------------------\n";
cout << "Implementing ios::setf\n\n";
int val1=100,val2=200;
cout.setf(ios::showpos);
cout<<val1<<" "<<val2;
cout << "\n--------------------------\n";
```

```
Implementing ios::setf

+100 +200
```

```
cout << "\n--------------------------\n";
cout << "Implementing ios::unsetf\n\n";
cout.setf(ios::showpos|ios::showpoint);
// Clear the showflag flag without
// affecting the showpoint flag
cout.unsetf(ios::showpos);
cout<<200.0;
cout << "\n--------------------------\n";
```

```
Implementing ios::unsetf

200.00
```

Formatted I/O

2. Formatting using Manipulators

The second way you can alter the format parameters of a stream is through the use of special functions called manipulators that can be included in an I/O expression.
The standard manipulators are shown below:

- **boolalpha**: The boolalpha manipulator of stream manipulators in C++ is used to turn on bool alpha flag
- **dec**: The dec manipulator of stream manipulators in C++ is used to turn on the dec flag
- **endl**: The endl manipulator of stream manipulators in C++ is used to Output a newline character.
- **and**: The and manipulator of stream manipulators in C++ is used to Flush the stream
- **ends**: The ends manipulator of stream manipulators in C++ is used to Output a null

Formatted I/O

- **fixed**: The fixed manipulator of stream manipulators in C++ is used to Turns on the fixed flag
- **flush**: The flush manipulator of stream manipulators in C++ is used to Flush a stream
- **hex:** The hex manipulator of stream manipulators in C++ is used to Turns on hex flag
- **internal**: The internal manipulator of stream manipulators in C++ is used to Turns on internal flag
- **left**: The left manipulator of stream manipulators in C++ is used to Turns on the left flag
- **right**: The right manipulator of stream manipulators in C++ is used to Turns on the right flag
- **scientific**: The scientific manipulator of stream manipulators in C++ is used to Turns on scientific flag
- **setbase(int base**): The setbase manipulator of stream manipulators in C++ is used to Set the number base to base

Formatted I/O

- **setfill(int** ch): The setfill manipulator of stream manipulators in C++ is used to Set the fill character to ch
- **setiosflags(fmtflags f)**: The setiosflags manipulator of stream manipulators in C++ is used to Turns on the flag specified in f
- **setprecision(int p)**: The setprecision manipulator of stream manipulators in C++ is used to Set the number of digits of precision
- **setw(int w):** The setw manipulator of stream manipulators in C++ is used to Set the field width to w

Formatted I/O

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

void Example()
{
    // performs same as setf( )
    cout << setiosflags(ios::showpos);
    cout << 123<<"\n";

    // hexadecimal base (i.e., radix 16)
    cout << hex << 100 << endl;

    // Set the field width
    cout << setfill('*') << setw(10) << 2343.0;
}

int main()
{
    Example();
    return 0;
}
```

```
+123
64
*****+2343
```

Creating own inserters

```
// Use a friend inserter for objects of type coord.
#include <iostream>
using namespace std;

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

This program displays the following:

```
1, 1
10, 23
```

Creating own extractor

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

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

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

istream &operator>>(istream &stream, coord &ob)
{
  cout << "Enter coordinates: ";
  stream >> ob.x >> ob.y;
  return stream;
}

int main()
{
  coord a(1, 1), b(10, 23);
```

```
  cout << a << b;

  cin >> a;
  cout << a;

  return 0;
}
```

File input output

1. ofstream

This data type represents the output file stream and is used to create files and to write information to files.

2. **ifstream**

This data type represents the input file stream and is used to read information from files.

3. **fstream**

This data type represents the file stream generally, and has the capabilities of both ofstream and ifstream which means it can create files, write information to files, and read information from files.

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

int main ()
{
    string data1;

    // open a file in write mode.
    ofstream outfile;
    outfile.open("afile.txt");

    cout << "Writing to the file" << endl;
    cout << "Enter your name: ";
    getline(cin,data1);

    // write inputted data into the file.
    outfile << data1 << endl;

    cout << "Enter your age: ";
    cin >> data1;
    cin.ignore();

    // again write inputted data into the file.
    outfile << data1 << endl;

    // close the opened file.
    outfile.close();

    // open a file in read mode.
    ifstream infile;
    infile.open("afile.txt");

    cout << "Reading from the file" << endl;
    infile >> data1;

    // write the data at the screen.
    cout << data1 << endl;

    // again read the data from the file and display it.
    infile >> data1;
    cout << data1 << endl;

    // close the opened file.
    infile.close();

    return 0;
}
```

Thank You