Unit 10 Standard Input and Output

Structure:

10.1 Introduction

In the previous unit you studied about files, streams, command line arguments and preprocessor directives. In this unit you are going to study I/O stream hierarchy and simple programs using these streams. You are also going to learn the methods to format the output of the programs

In C++, I/O is done with "streams." An input stream such as *cin* is a source of data that can be read into variables. An output stream such as *cout* is a place where data can be sent for display, storage, or communication. A file is a collection of data saved on a disk drive. Data in a file differs from data in variables because a file can exist before the program is run and can persist after the program ends. To read data from a file or to save data in a file, a program just has to use the right type of stream.

Objectives:

After studying this unit you should be able to:

- explain C++ iostream library and its organization
- describe programming using I/O streams

- explain ios class functions and flags
- discuss manipulators

10.2 Understanding the C++ *iostream* Library

The iostream library is an object-oriented library that provides input and output functionality using streams.

10.2.1 Standard Input/Output Stream Library

A stream is sequence of bytes. It is a continuous flow of data elements that are transmitted or intended for transmission in a defined format. It works either as a source from where the data can be obtained or as a destination for the output sent. Source stream that provides data to the program is called input stream. Destination stream that receives output from program is called output stream. A stream can basically be represented as a source or destination of characters of indefinite length. Figure 10.1 depicts the I/O stream library structure.

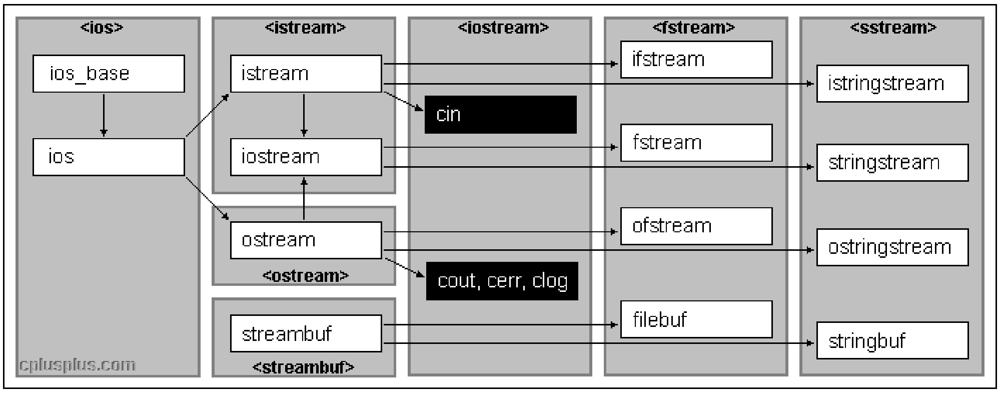

Figure 10.1: Stream library

To operate with streams we have at our disposal the standard iostream library which provides us the following elements:

- **Basic class templates** – The base of the iostream library is the hierarchy of class templates. Most of the functionality of the library in the type independent way is provided by the class templates. There are two template parameters in class templates i.e. the *char* type(charT) and traits. The *char type* describes the type of elements to be manipulated and *traits* parameter provides additional characteristics specific for that type of elements. In the class hierarchy the class templates have the same name as their char type instantiations but they have the prefix basic in their names. For example, the class template which istream is instantiated from is called basic_istream, the one from which fstream is instantiated is called basic_fstream, and so on. The ios_base is an exception which is by itself is type-independent.

- **Class template instantiations –** The library contains two standard sets of instantiations of the complete iostream class template hierarchy i.e. narrow oriented which manipulates the elements of char type and wide oriented that manipulates the wchar_t type elements. The few narrow oriented classes are ios, istream and ofstream. The names of narrow oriented classes and relationship amongst them is shown in figure 10.1. The wide oriented instantiation classes has the same naming convention as narrow oriented instantiation only difference is that in wide oriented instantiation the name of classes and objects are prefixed with character 'w'. For example wios, wistream, wostream.
- **Standard objects** – Many objects that performs standard input and output operations are declared in <iostream> library. These are of two types i.e. narrow oriented objects and wide oriented objects. Example of narrow oriented objects are – cin, cout, cerr and clog. And wide oriented objects are declared as win, wout, wcerr and wclog.
- **Types –** The iostream classes barely use fundamental types on their member's prototypes. They generally use defined types that depend on the traits used in their instantiation. For the default char and wchar_t instantiations, types - streampos, streamoff and streamsize are used to represent positions, offsets and sizes, respectively.
- **Manipulators –** The global functions that are used in conjunction with insertion (<<) and extraction (>>) operators performed on iostream objects are known as manipulators. The formatting settings and properties of streams are modified by manipulators. Examples of manipulators are- endl, hex and scientific.

10.2.2 Organization

The library and its hierarchy of classes are split in different files:

- As you have observed in the programs that we don't include <ios>, <istream>, <ostream>, <streambuf> and <iosfwd> files directly in the programs. They describe the base classes of the hierarchy and are automatically included by other header files of the library that contain derived classes.

- <iostream> It contains the declarations of the objects that performstandard input and output operations(including cin and cout). <fstream> the file stream classes (like the template basic_ifstream or the class ofstream) as well as the internal buffer objects used with these (basic_filebuf) are defined in this file. These classes manipulate the files using streams.
- <sstream>: The classes which are defined in file manipulates string objects as if they were streams.
- <iomanip> Declares some standard manipulators with parameters to be used with extraction and insertion operators to modify internal flags and formatting options.

10.2.3 Elements of the iostream Library

Classes

ios_base	Base class with type-independent members for the standard stream classes
ios_base	Base class with type-dependent members for the standard stream classes
istream	Input stream
ostream	Output Stream
iostream	Input/Output Stream
ifstream	Input file stream class
ofstream	Output file stream
fstream	Input/output file stream class
istringstream	Input string stream class
ostringstream	Output string stream class
stringstream	Input/output string stream class
tsreambuf	Base buffer class for streams
filebuf	File stream buffer
stringbuf	String stream buffer

Objects

Cin	Standard input stream
Cout	Standard output stream
Cerr	Standard output stream for errors
Clog	Standard output stream for logging

Types

Fpos	Stream position class template
Streamoff	Stream offset type
Streampos	Stream position type
Streamsize	Stream size type

Manipulators

Boolalpha	Alphanumerical bool values
Dec	Use decimal base
Endl	Insert newline and flush
Ends	Insert null character
Fixed	Use fixed-point notation
Flush	Flush stream buffer
Hex	Use hexadecimal base
Internal	Adjust field by inserting characters at an internal position
Left	Adjust output to the left
Noboolalpha	No alphanumerical bool values
Noshowbase	Do not show numerical base prefixes
Noshowpoint	Do not show decimal point
Noshowpos	Do not show positive signs
Noskipws	Do not skip whitespaces
Nounitbuf	Do not force flushes after insertions
Nouppercase	Do not generate upper case letters
Oct	Use octal base
Resetiosflags	Reset format flags
Right	Adjust output to the right
Scientific	Use scientific notation
Setbase	Set basefield flag
Setfill	Set fill character

Setiosflags	Set format flags
Setprecision	Set decimal precision
Setw	Set field width
Showbase	Show numerical base prefixes
Showpoint	Show decimal point
Showpos	Show positive signs
Skipws	Skip whitespaces
Unitbuf	Flush buffer after insertions
Uppercase	Generate upper-case letters
Ws	Extract whitespaces

Self Assessment Questions

1. The _______ library is an object-oriented library that provides input and output functionality using streams.
2. A _________ continuous flow of data elements that are transmitted or intended for transmission in a defined format.
3. The base of the iostream library is the hierarchy of _________.
4. clog stands for____________.

10.3 Basic Programming using Streams

10.3.1 Basic Stream Concepts

In C++, the file stream classes are designed with the idea that a file should simply be viewed as a stream or array of uninterpreted bytes. For convenience, the "array" of bytes stored in a file is indexed from zero to *len*-1, where *len* is the total number of bytes in the entire file.

Each open file has two "positions" associated with it:

- The current reading position, which is the index of the next byte that will be read from the file. This is called the "get pointer" since it points to the next character that the basic get method will return.
- The current writing position, which is the index of the byte location where the next output byte will be placed. This is called the "put pointer" since it points to the location where the basic put method will place its parameter.

  These two file positions are independent, and either can point anywhere at all in the file.

Creating Streams

The streams need to be created before they are used in the programs. The statements to create streams are similar to the variable declaration. These statements are written at the top of the program with variable declaration. The example to create a stream is shown below:

ifstream in_stream;

ofstream out_stream;

Before we can use an input or output stream in a program, we must create it. Statements to create streams look like variable declarations, and are usually placed at the top of programs or function implementations along with the variable declarations. So for example the statements

ifstream in_stream; This statement creates a stream called "in_stream" *belonging to the class (like type) "ifstream" (input-file-stream).*

ofstream out_stream;- This statement creates a stream called "out_stream" belonging to the class "ofstream" (output-file-stream).

However, the analogy between streams and ordinary variables (of type "int", "char", etc.) can't be taken too far. We cannot, for example, use simple assignment statements with streams (e.g. we can't just write "in_stream1 = in_stream2").

Connecting and Disconnecting Streams to Files

Consider that a file named “test.txt” exists shown in figure 10.2. The diagrammatic representation of this file is shown below:

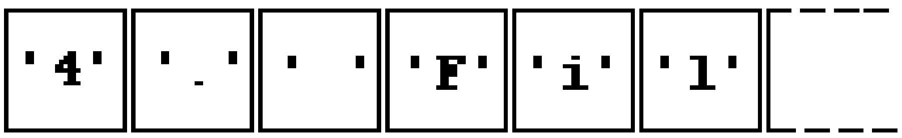

Figure 10.2: test.txt

Having created a stream, we can connect it to a file using the *member function* "open(...)". The function "open(...)" has a different effect for ifstreams than for ofstreams (i.e. the function is polymorphic).

To connect the ifstream "in_stream" to the file "test.txt", we use the following statement:

in_stream.open("Test.txt");

This connects "in_stream" to the beginning of "test.txt". Diagrammatically, we end up in the following situation as shown in figure 10.3.

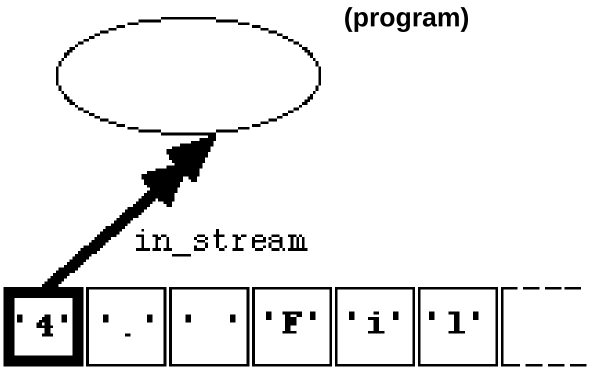

Figure 10.3: opening input stream

To connect the ofstream "out_stream" to the file "test.txt", we use an analogous statement:

out_stream.open("Test.txt");

Although this connects "out_stream" to "test.txt", it also deletes the previous contents of the file, ready for new input. Diagrammatically, we end up as shown in figure 10.4.

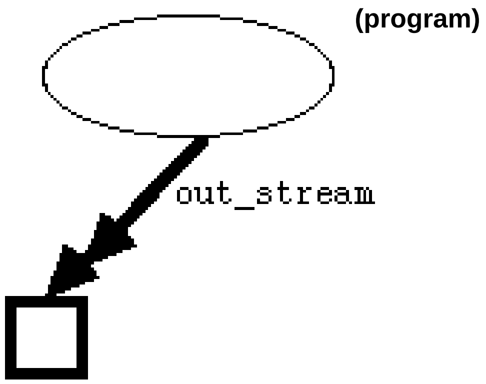

Figure 10.4: opening output stream

To disconnect, connect the ifstream "in_stream" to whatever file it is connected to, we write:

in_stream.close();

Diagrammatically, the situation changes from that of Figure 10.3 to figure 10.5:

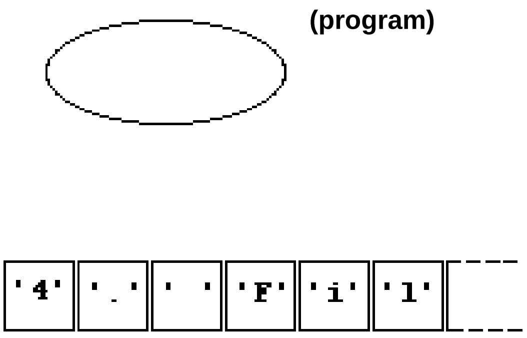

Figure. 10.5: closing an input stream

The statement:

out_stream.close();

has a similar effect, but in addition, the system will "clean up" by adding an "end-of-file" marker at the end of the file. Thus, if no data has been output to "Test.txt" since "out_stream" was connected to it, we change from the situation in Figure 10.4 to figure 10.6.

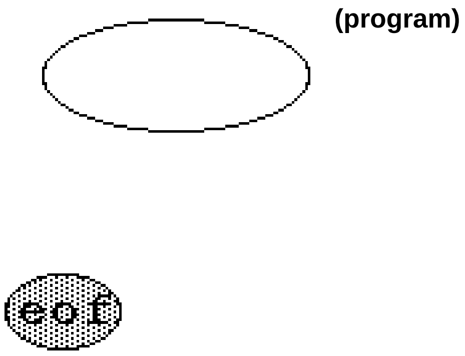

Figure 10.6: closing an output stream

In this case, the file "Test.txt" still exists, but is *empty*.

10.3.2 Your Very First Program

The first program, will create a file, and put some text into it.

```
#include <fstream>
using namespace std;
int main() {
   ofstream SaveFile("cpp-home.txt");
   SaveFile << "Hello World, from www.cpp-home.com and Loobian!";
   SaveFile.close();
   return 0;
}
```

The above program will create cpp-home.txt file and will write the sentence - Hello World, from www.cpp-home.com and Loobian into it. The file named cpp-home.txt will be created in the directory from where we execute the file containing the program.

Here is what every line means:

#include <fstream>
You need to include this file in order to use C++'s functions for File I/O. In this file, are declared several classes, including ifstream, ofstream and fstream, which are all derived from istream and ostream.

ofstream SaveFile("cpp-home.txt");

- ofstream means "output file stream". The handle for a stream to write in a file is created by it of the handle. You can pick whatever you want.
- ("cpp-home.txt"); - It opens the file named cpp-home.txt. The file will be created of it doesn't already exist. This file is placed in the directory from where we are executing the program. *ofstream* is a class. So, an object of this class is created by the following statement -ofstream SaveFile("cpp-home.txt");. Whatever arguments we have mentioned in the brackets are passed to the constructor. So, to summarize: we create an object from class ofstream, and we pass the name of the file we want to create, as an argument to the class' constructor.

SaveFile << “Hello World, from www.cpp-home.com and Loobian!”;

This statement writes the text in cpp-home.txt file. As you already know that SaveFile is a handle to the opened file stream. So, as you can see in statement above you have to write the handle name, and write the text in inverted commas after <<. Sometimes if it required to pass variable instead of text then you should just pass it as a regular use of the cout <<. For example, SaveFile << variablename;

SaveFile.close();

You should close the stream when you finish using it. The above statement closes the stream. *SaveFile* is an object from class ofstream, and this class (ofstream) has a function namd close that closes the stream. So, to close the stream you need to write the name of the handle, dot and close(). Once you have closed the file, you can’t access it anymore, until you open it again.

That’s the simplest program, to write in a file.

10.3.3 Reading a File

You saw how to write into a file. Now, when we have cpp-home.txt, we will read it, and display it on the screen. But let’s first look at one of the simplest and a most effective technique.

```
#include <fstream.h>
void main()
 { //the program starts here
   ifstream OpenFile("cpp-home.txt");
    char ch;
   while(!OpenFile.eof())
     {
        OpenFile.get(ch);
        cout << ch;
      }
    OpenFile.close();
}
```

You should already know what the first line is. Hence, we shall move to next important statement.

ifstream OpenFile(“cpp-home.txt”);

ifstream means “input file stream”. In the previous program, it was ofstream, which means “output file stream”. The previous program is to write a file, that’s why it was “output”. But this program is to read from a file, that’s why it is “input”. *OpenFile* is the object from class ifstream, which will handle the input file stream. And in the inverted commas, is the name of the file to open.

Note that there is nothing to check whether the file exists. This will be covered in a short while.

char ch;

An explanation for this statement is redundant.

while(!OpenFile.eof())

As you have already studied, when the end of the file is encountered then the function eof() returns a non-zero value. So we use a while loop that will run until the end of file is encountered. So, we will get through the whole file, so that we can read it.

OpenFile.get(ch);

OpenFile is the object from class ifstream. This class contains the get() function. So, as long as we have an object we can use this function. The get() function extracts a single character from the stream and returns it. In the program shown above the get() function takes the variable name as parameter in which the character which is read is placed. So, after calling OpenFile.get(ch) it will read one character from the stream OpenFile, and will put this character into the variable ch. This function if called second time will read the next character. It will not read the same character again and every time we loop, we read one character and put it into ch.

cout << ch;

Explanation for this statement is redundant.

OpenFile.close();

As we have opened the file stream, we need to close it. Use the close() function, to close it. Just as in the previous program.

When you compile and run this program, the output will be:
Hello World, from www.cpp-home.com and Loobian!

Self Assessment Questions

5. A_________ is a stream or array of uninterpreted bytes.
6. The statements to create streams are written at the top of the program with variable declaration. (True/False).
7. The statement ifstream in_stream; creates a stream called "in_stream" belonging to the class "ifstream". (True/False).

10.4 Formatted Console I/O Operations

C++ provides various console I/O functions for formatting the outputs. Formatting means displaying the outputs in a readable and comprehensible manner**.** Formatting of outputs can be achieved in two ways as mentioned below:

1. Using ios class functions and flags
2. Using Manipulators

10.4.1 ios Class Fucntions and Flags

The ios class contains many member functions that would help us to format the output in a number of ways. The most important ones among them are shown in table 10.1 below:

Table 10.1: ios format functions

Function	**Task**
width()	It specifies the required field size for displaying an output value
precision()	It specifies the number of digits to be displayed after the decimal point of a float value
fill()	It specifies a character that is used to fill the unused portion of a field.
self()	It specifies the format flags that can control the form of output display(like left justification and right justification)
unself()	It clears the flags specified

The special functions that can be included in I/O statements to alter the format parameters of a stream are known as manipulators. Table 10.2 shows some of important manipulator functions that are used frequently.

The header file iomapin should be included in the program to access these manipulators.

Table 10.2: Manipulators

Manipulators	**Equivalent ios function**
setw()	width()
setprecision()	precesion()
setfill()	fill()
setiosflags()	setf()
resetiosflags()	unsetf()

In addition to these functions supported by the C++ library, it is possible to create our own manipulator functions to provide any special output formats. We will discuss, in this section, how to use the predefined formatting functions and how to create new ones.

Now we will study ios format functions one by one

1. **Defining field Width: width()**
   You can use width() function to define the width of a field necessary for the output of an item. Because it is a member function, you have to use an object to invoke it. The example is shown below:
   ```
   cout.width(w);
   ```
   Here w is the filed width(number of columns). The output will be printed in the field of w characters wide at the right end of the filed. The width() function can specify the field width for only one item i.e. the item that follows immediately. It will revert back to the default after printing one item

   For example:

   ```
   cout.width(5);

   cout<<543<<12<<"\n";
   ```

   The result of the above statement will be:

		5	4	3	1	2

The value 543 is printed right-justified in the first five columns. The specification width(5) does not retain the setting for printing the number 12. This can be improved as follows:

```
cout.width(5);
cout<<543;
cout.width(5);
cout<<12<<"\n";
```

The result of the above statement will be:

		5	4	3				1	2

2. **Setting Precision: precision()**

The floating numbers are printed with six digits after decimal point by default. But you can also explicitly specify the number of digits to be printed after decimal while printing floating point numbers. You can achieve this by using precision member function. The syntax is as follows:

```
cout.precision(d);
```

Here d is the number of digits to the right of the decimal point.

For example:

```
cout.precision(3);
cout<<sqrt(2)<<"\n";
cout<<3.14159<<"\n";
cout<<2.50032<<"\n";
```

The output of the above statements will be:

```
1.141      (truncated)
2.142      (rounded to the nearest zeros)
2.5        (no trailing zeros)
```

We can set different values to different precision as follows:

```
cout.precision(3);
cout<<sqrt(2)<<"\n";
cout.precision(5);
cout<<3.14159;<<"\n";
```

3. Filling and Padding: fill()

Many of the times we print the values using much larger field widths than required by the values. And by default the unused spaces of field are filled with white spaces. You can fill this unused portion of field by some desired characters. It is used in the following form:

cout.fil(ch);

Here ch represents the character to used to fill the unused spaces. The example is shown below:

cout.fill('*');

cout.widht(10);

cout<<5250<<"\n";

The result of the above statements will be:

*	*	*	*	*	*	5	2	5	0

This kind of padding is used in the financial institutions like banks while printing cheques so that one can change the amount easily.

4. Formatting Flags, Bit-fields and self()

You have studied that when the function width is used, the value is printed right-justified by default in the field width created. But usually we print the text left-justified. The self() function of ios class is used to perform this task. The self function can be used as follows:

cout.self(arg1, arg2);

The arg1 is the formatting flags defined in the ios class. The formatting flag specifies the format action required for the output. Another ios constant, arg2, known as bit field specifies the group to which the formatting flag belongs. The table 10.3 shows the bit fields, flags and their format actions. There are three bit fields and each has a group of format flags which are mutually exclusive. Examples:

cout.self(ios::left, ios::adjustfield);

cout.self(ios::scientific, ios::floatfield);

It should be noted that the first argument should be one of the group members of the second argument.

Table 10.3: Flags and bit fields for self() function

Format required	Flag(arg1)	Bit-field(arg2)
Left-justified output	ios::left	ios::adjustfield
Right-justified output	ios::right	ios::adjustfield
Padding after sign or base	Ios::internal	ios::adjustfield
Scientific notation	ios::scientific	ios::floatfield
Fixed point notation	ios:fixed	ios::floatfield
Decimal base	ios::dec	ios::basefield
Octal base	ios::oct	ios::basefield
Hexadecimal base	ios::hex	ios::basefield

Let us consider the following code:

```
cout.fill(‘*’);
cout.self(ios::left, ios::adjustfield);
cout.width(15);
cout<<”table 1”<<”\n”;
```

The output will be:

T	A	B	L	E		1	*	*	*	*	*	*	*	*

5. Displaying trailing zeros and plus sign

If the following numbers 10.75, 25.00 and 15.50 are printed using field width of eight positions, with two digits precision, then the output will be as follows:

			1	0	.	7	5
						2	5
				1	5	.	5

It is to be noted that the trailing zeros in the second and third items have been truncated. But there are many situations in which we need to print the trailing zeros like the list of prices of items, employees’ salary statement. So the self() function with the flag ios::showpoint can be used as a single argument to achieve this task. For example:

```
cout.self(ios::showpoint)
```

This statement will print the trailing zeros and trailing decimal points. Under default precision, the value 3.25 will be displayed as 3.250000 as the default precision assumes a precision of six digits.

Similarly, we can print a plus sign before a positive nume3 using the following statement:

```
cout.self(ios::showpos);              //show + sign
```

For example consider the following statements:

```
cout.self(ios::showpint);
cout.self(ios::showpos);
cout.precision(3);
cout.slef(ios::fixed, ios::floatfield);
cout.self(ios::internal, ios::adjustfield);
cout.width(10);
cout<<275.5<<"\n";
```

The output of the above statements will be:

+			2	7	5	.	5	0	0

10.4.2 Managing output with Manipulators

Manipulators are the functions which are used to manipulate the output formats and are provided in header file *iomapin.* They provide the same features as that of the ios member functions and flags. The most commonly used manipulators are shown in table 10.4 below. Their meaning and equivalents are also shown in the table. You have to include the file *iomapin* in the program to access these manipulators.

Table 10.4: Manipulators and their meanings

Manipulator	Meaning	Equivalent
setw(int w)	Set the field width to w	width()
setprecision(int d)	Set the floating point precision to d	precision()
setfill(int c)	Set the fill character to c	fill()
setiosflags(long f)	Set the format flag f	setf()
resetiosflags(long f)	Clear the flag specified by f.	unsetf()
endl	Insert new line and flush the stream	"\n"

Some examples of manipulators are as follows:

```
cout << setw(10) << 12345;
```

The above statement prints the value 12345 right-justified in a field width of 10 characters. To make the output left-justified, the statement can be modified as follows:

```
cout << setw(10) << setiosflags(ios::left) << 12345;
```

We can format two or more outputs by one statement. For example:

```
cout << setw(5) << setprecision(2) << 1.2345 << setw(10) << setprecision(4) << sqrt(2) << setw(15) << setiosflags(ios::scientific) << sqrt(3) << endl;
```

The above statement will print all the three values in one line with the field size 5, 10 and 15 respectively. You can see that each output is controlled by different sets of format specifications.

It is possible to jointly use the manipulators and the ios functions in a program. The following segment of code is valid:

```
cout.self(ios::showpoint);
cout.self(ios::showpos);
cout << setprecision(4);
cout << setiosflags(ios::scientific);
cout<<setw(10) << 123.45678;
```

The difference in the way manipulators are implemented as compared to the ios member functions is that the ios function returns previous format state which can be used later, if necessary. Whereas, the manipulator does not return the previous format state. If you want to save the old format states, you should use trhe ios member fucntions rather than the manipulators.

```
Example: cout.precision(2);             //previous state
         int p=cout.precision(4);       //current state
```

When we execute the above statements, it is observed that p will hold the value of 2(previous state) and the new format state will be 4. The previous format state can be restored as follows:

```
cout.precision(p);       //p=2
```

Self Assessment Questions

8. Formatting of outputs can be achieved in two ways these are __________ and ___________.
9. The _________ is used function to define the width of a field necessary for the output of an item.
10. _________ are the functions which are used to manipulate the output formats and are provided in header file ________.
11. ________ function specifies a character that is used to fill the unused portion of a field.
12. The resetios flags(long f) clears the flag specified by f. (True/False).

10.5 Summary

- A stream is a sequence of bytes. It is a continuous flow of data elements that are transmitted or intended for transmission in a defined format. It works either as a source from where the data can be obtained or as a destination for the output sent.
- To operate with streams we have to our disposal the standard iostream library which provides us the following elements: Basic class templates, Class template instantiations, Standard objects, Types, Manipulators.
- In C++, the file stream classes are designed with the idea that a file should simply be viewed as a stream or array of uninterpreted bytes.
- The current reading position, which is the index of the next byte that will be read from the file.
- The current writing position, which is the index of the byte location where the next output byte will be placed.
- The streams need to be created before they are used in the programs. The statements to create streams are similar to the variable declaration.
- The streams need to be created before they are used in the programs. The statements to create streams are similar to the variable declaration. These statements are written at the top of the program with variable declaration.
- C++ provides various console I/O functions for formatting the outputs. Formatting means displaying the outputs in more readable and

comprehensible manner. Formatting of outputs can be achieved in two ways i.e. using ios class functions and flags and using Manipulators.

- The ios class contains many member functions that would help us to format the output in a number of ways. The most important ones among them are :width(), precision(), fill(), self(), unself()
- Manipulators are the functions which are used to manipulate the output formats and are provided in header file iomapin.

10.6 Terminal Questions

1. Explain the standard Input/Output iostream library.
2. List and explain the elements of iostream library.
3. Discuss in detail the basic stream concepts.
4. Discuss the process involved in reading a file with the help of an example.
5. Explain all the ios functions used for formatting output with examples.
6. Explain that how manipulators are used to manage the output.

10.7 Answers

Self Assessment Questions

1. Iostream
2. Stream
3. Class templates
4. Standard output stream for logging
5. File
6. True
7. True
8. using ios class functions, flags and manipulators
9. width()
10. manipulators , iomapin
11. fill()
12. True

Terminal Questions

1. A stream is sequence of bytes. It is a continuous flow of data elements that are transmitted or intended for transmission in a defined format. It works either as a source from where the data can be obtained or as a destination for the output sent. For more details refer section 10.2.
2. Classes, objects and types are the major division under elements of iostream library. For more details refer section 10.2.
3. In C++, the file stream classes are designed with the idea that a file should simply be viewed as a stream or array of uninterpreted bytes. For more details refer section 10.3.
4. The first program, will create a file, and put some text into it.

   ```
   #include <fstream>
   using namespace std;
   int main() {
    ofstream SaveFile("cpp-home.txt");
    SaveFile << "Hello World, from www.cpp-home.com and Loobian!";
    SaveFile.close();
    return 0;
   }
   ```

   The above program will create cpp-home.txt file and will write the sentence -Hello World, from www.cpp-home.com and Loobian into it. For more details refer section 10.3.2.
5. The ios class contains many member functions that would help us to format the output in a number of ways. The important ios class format functions are as follows: width(), precision(), fill(), self(), unself(). For more details refer section 10.4.1.
6. Manipulators are the functions which are used to manipulate the output formats and are provided in header file iomapin. They provide the same features as that of the ios member functions and flags.
   For more details refer section 10.4.2.