7...

Managing Console I/O Operations

Objectives...

- To understand C++ streams and C++ stream classes.
- To learn about Unformatted I/O operations and Formatted console I/O operations.
- To study Output formatting using manipulators.
- To understand user defined manipulators.

7.1 INTRODUCTION

- C++ defines its own object oriented I/O system; it has its own object oriented way of handling data input and output. These Input/Output operations are carried out using different stream classes and their related functions.
- Until now we have used the object of the istream class (cin) and object of ostream class (cout); with operators >> and << respectively. For formatting purpose we use different streams and their functions in C++.

7.2 C++ STREAMS [S-18, 19, W-22]

- A stream is sequence of bytes.
- Stream acts either as a source from which the input data can be obtained or as a destination to which the output data can be sent.
- In C++ to manage the data flow we have different stream classes, (Refer Fig. 7.1)
- Fig. 7.1 shows stream classes for console I/O.

1. **istream class:** It is a derived class of `ios` and hence, inherits the properties of `ios`. It defines input functions such as `get( )`, `getline( )` and `read( )`. In addition, it has an overloaded member function, stream extraction operator >>, to read data from a standard input device to the memory items.
2. **ostream class:** It is a derived class of `ios` and hence inherits the properties of `ios`. It defines output functions such as `put( )` and `write( )`. In addition, it has an overloaded member function, stream insertion operator <<, to write data from

memory to a standard output device. cerr object is of ostream class used for displaying error messages.

3. **iostream class:** It is derived from multiple base classes, istream and ostream, which are inturn inherited from class ios. It supports both input and output stream operations.

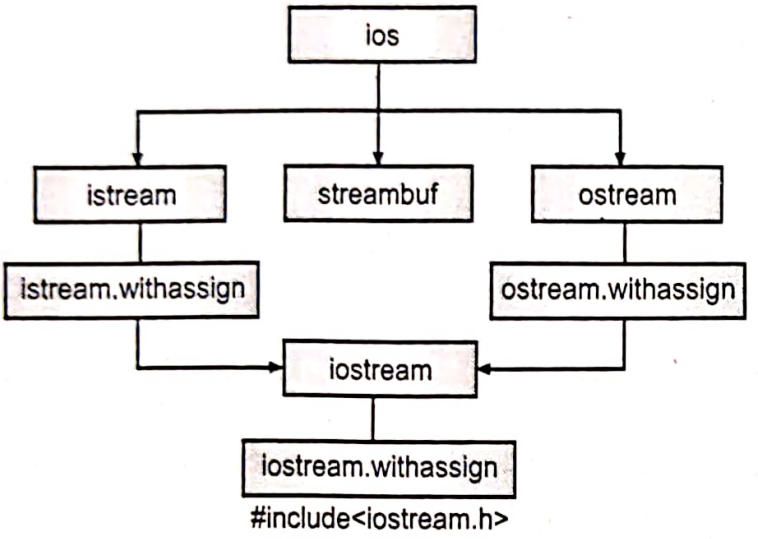

Fig. 7.1: Stream class for console I/O

- The classes' istream_withassign, ostream_withassign and iostream_withassign add the assignment operators to their parent classes.

7.3 C++ STREAM CLASSES [S-22]

- The Input/Output system in C++ is designed to work with a wide variety of devices including terminals, disk and tape drives.
- The Input/Output system supplies an interface to the programmer that is independent of the actual device being accessed.
- This interface is known as stream. C++ Input/Output occurs in a stream of bytes.
- A stream is simply a sequence of bytes. It is a one way transmission path that is used to connect a file stored on a physical device such as a disk or CD-ROM, to a program.
- It acts either as a source from which the input data can be obtained or as a destination to which the output data can be sent.
- The source stream, which provides data to the program, is called the **input stream** and the destination stream that receives output from the program is called the **output stream**.
- In input operation, the bytes flow from a device (For example, a keyboard, a disk drive, a network connection) to main memory.
- In output operation, bytes flow from main memory to a device (For example, a display screen, a printer, a disk drive, a network connection). In other words, a program extracts the bytes from an input stream and inserts bytes into an output stream as shown in Fig. 7.2.
- The data in the input stream can come from the keyboard or any other storage device.
- Similarly, the data in the output stream can go to the screen or any other storage devices.

- Stream acts as an interface between the program and the Input/Output device. Therefore, a C++ program handles data independent of the devices used.

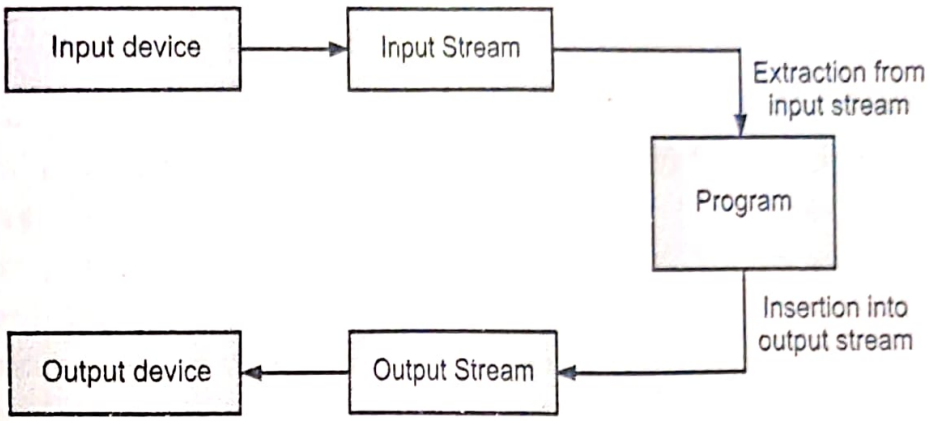

Fig. 7.2: Data Streams in C++

7.4 UNFORMATTED I/O OPERATIONS

7.4.1 Unformatted I/O Operators

- The <<, >> operators for shifting are overloaded in ostream and istream class respectively.
- The general format for reading the data from the keyboard is:

```
cin >> var1, >> var2 >> ..... >> varn;
```

Where, var1 and var2 are any, C++ predefined variables.

- This statement in the program will make the compiler to wait for the input data from the keyboard.
- The operator >> reads the data character by character and assigns it to the indicated location.
- The reading for the variable will be terminated at the encounter of a white space or a character that does not match with destination type.
- The general form for displaying data on screen is:

```
cout << data1 << data2 << ..... <<datan;
```

7.4.2 Unformatted I/O Functions

- Unformatted I/O functions are listed below:

1. get(): [W-18, 22]

- The class istream defines a member function get() which is used to handle single character Input/Output operations at a time.
- There are two types of get() functions.
  - `get(char *);`
  - `get(void);`
  - We can use any one of the prototype to read the data from a keyboard, it is similar to that of `cin` object.

- Only the difference is that `cin` will terminates when encounters space, tab or new line character where as with get() function. We can read these all.
- `get(char *)`: Versions assigns the inputted character to its argument.
- `get (void)`: Type returns the input character.

Program 7.1: Program for get function.

```
#include<iostream>
using namespace std;
int main()
{
    char c;
    cout <<"Enter a character:";
    cin.get(c);
    cout <<"Entered character is:";
    cout <<c;
    return (0);
}
```

Output:

```
Enter a character:A
Entered character is:A
Press any key to continue . . .
```

- We can write the same program by making a use of get(void) version as.

```
#include<iostream>
using namespace std;
int main()
{
    char c;
    cout <<"Enter character:";
    c = cin.get();
    cout<<"Entered character is:";
    cout << c;
    return(0);
}
```

Output:

```
Enter a character:A
Entered character is:A
Press any key to continue . . .
```

- The value returned by the function get() is assigned to the variable c.

2. put(): [W-18, 22]

- The class ostream defines the member function put() which is used to output a line of text on a screen character by character.
- The function put(), outputs one character at a time.
- **Syntax:**

```
cout.put(char);
```

- **Example:**

```
cout.put('s');      //Displays the character s,
cout.put(ch);       //Displays the contents of a variable ch.
```

- The variable ch must contain character value. We can also use number as an argument to the put() function like:

```
cout.put(65);       //Displays character A. Since, the ASCII value of character is A is 65.
```

Program 7.2: Program for put function.

```
#include<iostream>
using namespace std;
int main()
{
    char c;
    cout <<"Enter a character:";
    cout.put(c);
    cin.get(c);
    cout <<"Entered character is:";
    cout <<c;
    return (0);
}
```

Output:

```
Enter a character:M
Entered character is:M
Press any key to continue . . .
```

---

3. getline(): [S - 18]

- We can read line of text more efficiently by making a use of getline() function.
- The getline() function reads a whole line of text that ends with a newline character.
- **Syntax:**

```
cin.getline(line, size);
```

- This statement calls the built in function getline() which reads character input into the variable line.
- This function will stop further as soon as it encounters a new line character "\n" or after reading size-1 character.

- This function reads the newline character but it does not save it. Consider the following example:

```
char title[20];
cin.getline(title, 20);
```

And suppose the entered value is: `programming with C++ is fun`

- Then output would be `programming with C++`
- As we are mentioning the size as 20 so, getline() function will read only 20 characters including white spaces.
- We can also read the string using the overloaded operator >> with object `cin` as,

```
cin >> title;
```

- But `cin` can only read the strings that do not contain any white spaces, which means `cin` can read only single word at a time not a series of words line "programming with C++ is fun".

Program 7.3: Program for getline function.

```
#include<iostream.h>
#include<conio.h>
int main()
{
    int size = 40;
    char title [50];
    cout << "Enter title:";
    cin.getline(title, size);
    cout<<"Entered title is:";
    cout <<"\n" << title;
    getch();
    return(0);
}
```

Output:

```
Enter title: Computer
Entered title is:Computer
```

---

4. write():

- Write function is used to write a set of characters into the file.
- The write() is a member function of ostream class it is used to display entire line on the screen.
- **Syntax:**

```
cout.write (line, size);
```

variable of which contents have to be display and size is number of
; One important thing to note about write() function is, it does
displaying the characters automatically when NULL character is
is greater than the length of line, then it displays beyond the

5. read():

- The read function is used to read a set of characters form the file.
- **Syntax:**

```
fileObject_name.read((char *) and variable_name;
size of (variable_name));
```

Program 7.4: Program for write function.

```
#include<iostream.h>
#include<conio.h>
int main()
{
    char name [10];
    cout << "Enter name:";
    cin >> name;
    cout<<"Entered Name is:";
    cout.write (name, 10);
    getch();
    return(0);
}
```

Output:

```
Enter name: Mahesh
Entered Name is: Mahesh
Press any key to continue . . .
```

6. peek():

It reads particular char from file.

7. putback():

It places the control back to previous character obtained by get().

8. ignore():

It skips number of characters while reading or writing in a file.

9. sizeof():

It gives size of memory required for storage.

For example: `sizeof(c);` where c is a character so it requires 1 byte of storage.

7.5 FORMATTED CONSOLE I/O OPERATIONS [S-18, 22]

- There are two types of functions are available in C++ for formatting the output.
- These are:
  1. IOS class functions.
  2. Manipulators.
- The IOS class contains large number of functions that would help us to format the output in number of ways.

- The IOS class functions are:
  - `width()` : To specify the required field size for displaying an output value.
  - `precision()` : To specify the number of digits to be displayed after the decimal point.
  - `fill()` : To specify a character i.e. used to fill the unused portion of a field.
  - `setf()` : To specify format flags that can control the form of output display.
  - `unsetf()` : To clear the flags specified.
- The functions used for formatting are listed below:

1. Field width:

- width( ) function is used to define the width of a field necessary for the output of an item.
- The width function sets the field width. As width is the member function, you have to use an object to invoke it.
- **Syntax:**

```
cout.width (w);
```

Where w is the field width.

- Width function can also be used with input function. It is given by,

```
cin.width (w);
```

- For example,

```
cout.width (6);
cout<<241<<18<<"\n"
```

Will give the output:

			2	4	1	1	8

- The value 241 is printed right justified in first six columns. The specification width (6) does not retain the setting for printing the number 18.
- This can be written as:

```
cout.width (6);
cout<<241
cout.width (5);
cout<<18
```

This will give the output as:

			2	4	1	.	.	.	1	8

Program 7.5: Program for the use of width function for character array.

```
#include<iostream.h>
int main ( )
{
   int w = 4;
```

```
    char string [10];
    cout<<"Enter a sentence:\n";
    cin.width(5);
    while (cin>>string)
    {
        cout.width (w++);
        cout<<string<<endl;
        cin.width (5);
    }
    return 0;
}
```

Output:

```
Enter a sentence:
ABC LMN PQR XYZ
ABC
LMN
PQR
XYZ
Press any key to continue . . .
```

2. Setting Precision:

- We can control the precision of floating point numbers i.e. the number of digits to the right of the decimal point.
- This can be done either by using **setprecision()** stream manipulator or **precision()** member function.
- The precision member function with no arguments return the current precision setting.
- **Syntax:**

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

Where, d is the number of digits to the right of the decimal point.

- For example:

```
cout.precision(4);
cout<<sqrt (2)<<"\n";
cout<<3.14159<<"\n";
cout<<6.40000<<"\n";
```

- The outputs will be,

```
1.4112      (truncated value)
3.1416      (rounded to nearest value)
6.4         (no trailing zeros)
```

- We can also combine the field specification with the precision setting.

```
cout.precision (3);
cout.width (6);
cout<<2.24689;
```

- The output will be:

	2	.	2	4	7

3. Filling and Padding:

- Filling and Padding are used to fill the unused positions of the field.
- We can use fill ( ) function to fill the unused position by any desired character.
- **Syntax:**

```
cout.fill (ch);
```

Where, ch represents the character which is used for filling the unused positions.

- Consider the following example:

```
cout.fill ("*");
cout.width(10);
cout<<4280<<"\n";
```

The output will be:

*	*	*	*	*	*	4	2	8	0

- These characters are inserted as padding.

4. Formatting flags, Bit fields:

- The set flag function, setf() is used for printing a left-justified number or for getting a floating point number printed in the scientific notation.
- **Syntax:**

```
cout.setf (arg1, arg2);
```

- The arg1 is one of the formatting flags defined in class ios.
- The formatting flags specify the format action required for the output.
- Arg2 is another ios constant, which is also known as bit field, specifies the group to which formatting flag belongs.
- There are three bit fields and each has a group of format flags which are mutually exclusive.
- Flags and bit fields for setf( ) functions are given in following table.

Table 7.1: Flags and bit fields for setf( ) functions

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 indicator	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

- Consider the following example:

```
cout.fill ('*');
cout.self (ios:: left, ios::adjustfield);
cout.width (12);
cout<<"TONY"<<"\n";
```

- The output will be:

T	O	N	Y	*	*	*	*	*	*	*	*

- When, we print the number trailing with zero, they will be truncated.
- For example, if we want to print 312.42, 29.00, 18.20, with seven positions and three digit precision, the output will be:

		3	1	2	.	4		2

						2	9

				1	8	.	2

- To remove such problems, i.e. if we want to display trailing zeros, we can use the function **setf ()** with the flag ios:: showpoint as a single argument. It is written as:

```
cout.setf (ios:: trailing zero);
```

- This function will display trailing zeros and trailing decimal point. Similarly, we can also display plus sign before the number. The function used for this is,

```
cout.self (ios:: for sign);
```

- Consider the following example:

```
cout.setf (ios::showpoint);
cout.setf (ios::showpos);
cout.precision (3);
cout.setf (ios::fixed, ios::floatfield);
cout.setf (ios::internal, ios::adjustfield);
cout.width (10);
cout<<100.5 << "\n";
```

- The output will be:

+			1	0	0	.	5	0	0

- The flags showpoint and showpos do not have any bit fields and therefore are used as single arguments in setf().

- This is possible because the setf() has been declared as an overloaded function in class IOS. Some other flags that do not have bit field are showbase, uppercase, skipws, unitbuf, stdio.

  `ios:: showbase` helps to use base indicator on output.

  `ios:: uppercase` helps to use uppercase letters.

  `ios:: skipws` skips white space on input.

  `ios:: unitbuf` flush all streams after insertion.

  `ios:: stdio` flushes stdout and stdin after insertion.

7.6 OUTPUT FORMATTING USING MANIPULATORS [S-22]

- Manipulators are special functions that can be included in the I/O statements to alter the format parameters of a stream.
- To access these manipulators, the file **iomanip.h** has to be included in the program.
- Manipulators perform formatting tasks. They provide the same features as IOS member functions and flags. But some manipulators are more convenient to use. We can concatenate several manipulators together.

Table 7.2: Manipulators and equivalent I/O functions

Manipulators	Equivalent I/O function
setw()	width()
setprecision()	precision()
setfill()	fill()
setioflags()	setf()
resetioflags()	unsetf()

1. Setting of field width:

- setw() is used to define the width of field necessary for the output for a variable.
- For example: `setw(6);` This function will reserve 6 digits for a number.

```
setw(6);
cout << 1234;
```

Output:

		1	2	3	4

2. Setting precision for float numbers:

- We can control number of digits to be displayed after decimal point for float numbers by using setprecision() function.

```
setprecision(2);
```

- This function will display only two digits after a decimal point.
- **For example:**

```
setprecision(2);
cout<<3.14159;
```

Output: `3.14`

3. Filling of unused positions:

- Filling of unused positions of the field can be done by using setfill().

```
setfill('*');
```

- Here, unused positions will be filled by using * sign.

```
setw(6);
setfill ('*');
cout << 1234;
```

Output:

*	*	1	2	3	4

4. setbase():

- It is used to show the base of a number, for example:

```
setbase(10)
```

7.6.1 User Defined Manipulators

- The general form for creating a manipulator without any argument is,

```
ostream and manipulator (ostream and output)
{
    - - - - - - - - - -
    - - - - - - - - - -     //code
    - - - - - - - - - -
    return output
}
```

Here, manipulator is the name of manipulator under creation. Consider the following example which defines a manipulator called money that displays 'dollor'.

```
ostream and money (ostream and output)
{
    output <<"Dollors ($)";
    return output;
}
```

- Now the statement,

```
cout << 400 << money;
```

Will display the output as,

```
400 Dollors ($)
```

Summary

- A stream is sequence of bytes.
- Stream acts either as a source from which the input data can be obtained or as a destination to which the output data can be sent.
- **istream class:** It is a derived class of *ios* and hence, inherits the properties of *ios*. It defines input functions such as *get( )*, *getline( )* and *read( )*.
- **ostream class:** It is a derived class of *ios* and hence inherits the properties of *ios*. It defines output functions such as *put( )* and *write( )*.
- **iostream class:** It is derived from multiple base classes, istream and ostream, which are inturn inherited from class *ios*.

- The source stream, which provides data to the program, is called the **input stream** and the destination stream that receives output from the program is called the **output stream**.
- The <<, >> operators for shifting are overloaded in ostream and istream class respectively.
- The class istream defines a member function get() which is used to handle single character Input/Output operations at a time.
- The class ostream defines the member function put() which is used to output a line of text on a screen character by character.
- The getline () function reads a whole line of text that ends with a newline character.
- Write function is used to write a set of characters into the file.
- The read function is used to read a set of characters form the file.
- width ( ) function is used to define the width of a field necessary for the output of an item.
- We can use fill ( ) function to fill the unused position by any desired character.

Check Your Understanding

1. The stream that supplies data to a program is called ..... stream.
   (a) input (b) output
   (c) I/O (d) none
2. The stream that receives data from the program is called ..... stream.
   (a) input (b) output
   (c) I/O (d) none
3. The read() and write() functions handle data in ..... form.
   (a) text (b) binary
   (c) character (d) none
4. The ios::ate mode allows us to write data anywhere in the file.
   (a) true (b) false
5. The functions write( ) and get( ) are similar.
   (a) true (b) false
6. The output pointer is called put pointer.
   (a) true (b) false
7. The programmer must create cin, cout, cerr, clog object explicitly.
   (a) true (b) false

Answers

1. (a)	2. (b)	3. (b)	4. (b)	5. (b)	6. (a)	7. (b)

Practice Questions

Q.I Answer the following questions in short:

1. What is a Stream? Enlist various Stream Classes.
2. What are the operators used by unformatted I/O?
3. What is meant by Manipulators?
4. List various unformatted I/O operations.
5. Differentiate between put() and write() function.

Q.II Answer the following questions:

1. Describe various unformatted I/O operations.
2. Explain the following:
   (i) getline()
   (ii) setw()
3. Explain setf() in detail.
4. Write short note on: User Defined Manipulators.
5. Explain the following function:
   (i) read
   (ii) write
6. What are the IOS class function? Explain them.
7. Write short notes on:
   (i) Streams
   (ii) Manipulators.

Q.III Define the term:

1. Stream
2. Manipulators

Previous Exam Questions

April 2018

1. What is stream concept in C++? **[2 M]**

Ans. Refer to Section 7.2.

2. What is the difference between cin and getline( )? **[2 M]**

Ans. Refer to Section 7.4.2.

3. Explain any four formatted input/output functions. **[4 M]**

Ans. Refer to Section 7.5.

October 2018

1. Write uses of get( ) and put( ). **[2 M]**

Ans. Refer to Section 7.4.2.

April 2019

1. What is stream? Explain a stream class hierarchy. **[4 M]**

Ans. Refer to Section 7.2.