Discipline Courses-I
Semester-I
Paper: Programming Fundamentals
Unit-IV
Lesson: File

Lesson Developer: Ritu Singhal
College/Department: I.P College, University of Delhi

Table of Contents

1.1 Text I/O

You have done a number of programs in earlier chapters, have you noticed what is the program doing? It is taking data from some source, processing that data and giving you the result i.e. output at the destination. Accepting data, processing and giving output, is the work of a processor. So a program is like a data processor (as it processes data). How do we get data? Data may be obtained from many sources, for example keyboard, file and many more. Similarly output can go to many destinations like monitor, files or printer. So handling Input/Output is a complex task**.**

C++ considers each source of data as an input entity and result (processed data) as an output entity. Since all input devices are different, their way of accepting data is also different; C++ uses same input operations for each input device and same output operations for all types of output devices. All I/O operations are handled by Streams.

1.1.1 Learning objectives

After reading this topic you should be able to:

- Define Input and Output Entities.
- Define Standard input/output functions by using Unformatted and formatted I/O operations.
- Use operator >>, operator <<, getch(), putch(), getline(), write() functions for unformtted operations.
- Format I/O operations by using ios class functions and flags, manipulators.

1.1.2 Input and Output Entities

Input entity: Any source from that we can accept data is known as an input entity.

Output entity: The result of any program that we obtain after processing the data is known as an output entity.

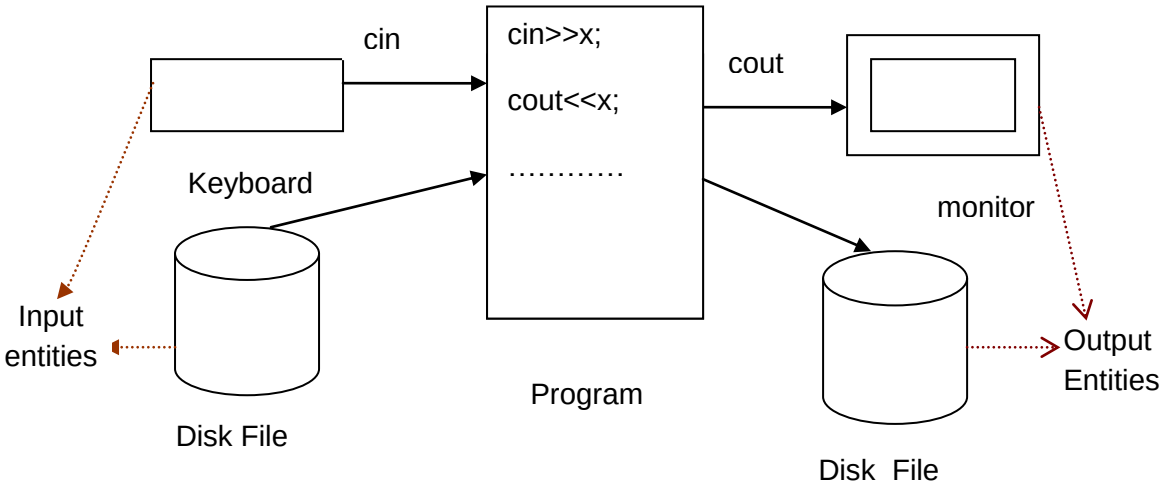

Figure 1.1 Program and I/O Entities Interface

In Figure 1.1, the interfacing of program with input and output entities is shown.

Here, we talk about two common input entities, keyboard and files and two output entities monitor and files. Files come in both input and output entities. So let us discuss files first.

Files

A File is a collection of related data or information usually stored at one place. Place may be hard disk, floppy disk, pen drive, CD, DVD or any other secondary storage device.

When computer reads data from a file, the data from the secondary storage device moves to main memory. This data movement uses a special area known as buffer. A buffer is a temporary storage area that hold the data during swapping. The activities buffer do to handle data are taken care by the device drivers (software that supplied by supplier to install a particular device) or access methods provided by operating system.

Standard Input

Keyboard is standard input device through which user can enter the data for a program.

Standard Output

Monitor is standard output for a program. By default, output of a program is displayed on monitor.

Standard Error

When we run a program, there may be any error. The type and reason of the error should be communicated to the user. This communication is performed through Monitor. So monitor is also treated as standard error device. we can see all the errors on monitor and correct them.

1.1.3 Streams

A **stream** is a sequence of bytes. A stream is an abstract representation of an input data or output data. 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. So, programmer need not know the operation of each data source or destination that may be attached to the computer.

Input stream: The source stream that provides data to the program is called the input stream.

Output stream: The destination stream that receives output from the program is called the output stream.

A program takes the bytes from an input stream and inserts them into an output stream. Keyboard or any storage device provides data for input stream and the data in the output stream can go to the screen or storage device.

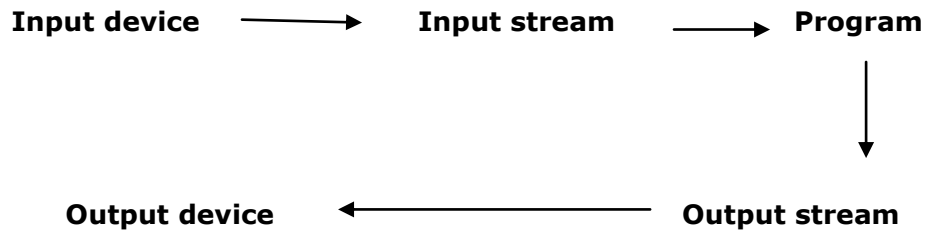

Figure 1.2 Flow of data using stream

So, a stream acts as an interface between the program and I/O devices. And because of these I/O stream C++ program handles data independent of the device implementation.

C++ uses stream and stream classes to implement its I/O operations with the console and files.

Following tasks are performed for sending or receiving the data:

1. In first step streams are created. To create a stream, you should aware of three things :
   a. Whether it is input stream or output stream i.e. type of the stream.
   b. I/O entity that stream use for data transmission.
   c. Transfer mode i.e. text mode or binary mode (details are given in later topics).
2. After creation, stream must be connected to required source or destination.
3. After completion (I/O no longer needed), it should be disconnected from the stream.

c++ defines following standard streams to complete the execution. The task of disconnecting standard streams to standard I/O entities is handled by the operating

system. When a program stops execution, the entities are disconnected from the corresponding streams.

1.1.3.1 Standard Streams

When we execute C++ program, the following streams are opened and closed when the program complete the execution.

I Standard input stream

II Standard output stream

III standard error stream

IV standard logging stream

Standard input stream

It is represented by an object cin of istream class (from which input is received). This stream by default receives input from keyboard, but by using redirection (>>) the stream can take input from a disk file or other input device.

cin : Input stream connected to the standard input device (keyboard).

For example cin >> val;

Standard output stream

It is represented by an object cout of ostream class (to which output is sent). This stream by default sends output to computer screen, but by using redirection(<<) stream output can be sent to a disk file or some other output device.

cout: Output stream connected to standard output device (computer screen).

For example,

```
int val = 45;
cout << "Value = " << val << endl;
```

Output is :

Value = 45

Standard error stream

It is represented by an object cerr of ostream class (stream to which error messages are sent). This stream sends all the error messages to the screen. It is unbuffered i.e. the

error displayed on the console immediately after it is written. The redirection does not work here.

Standard error is where you should display error messages.

For example :

cerr << "Can't open input file input.txt!" << endl;

Standard logging stream

It is represented by an object clog of ostream class.It is a fully buffered version of cerr, therefore, it does not display on the console until the buffer is full.

1.1.4 Standard input/output functions

C++ contains a number of input and output functions. You have used objects cin and cout earlier for input and output of data. It becomes possible by overloading operators. I/O functions that we can use in our program are:

- Overloaded operator >>
- Overloaded operator <<
- get() function
- put() function
- getline() function
- write() function

Let us discuss them in detail:

1.1.4.1 Input using Overloaded operator >> :

The syntax for reading the data from the input device using operator >> (known as extraction operator) is :

```
cin>>VariableName;
```

Do you know what will happen?

This statement will cause the compiler to stop the execution and wait for the input from keyboard. The operator >> reads the data character by character and assigns it to the variable. The reading of the data will be terminated either on pressing enter key or white space or a character that is different from data type of variable to be read terminates the input.

The operator >> is overloaded in the istream class.

Here are some examples:

a)
```
int a;
cin>>a;
```

if we give input 345p, then it read 345 as it is integer and leave 'p' and end this statement. So value of variable a is 345.

b)
```
int a,b,c;
cin>>a>>b>>c;
```

Suppose we input 1 4 9 then value of variables will be a=1, b=4, c=9.

1.1.4.2 Output using Overloaded operator << :

This operator is overloaded in the ostream class.

The syntax for writing the data to the output device using insertion operator (<<) is:

```
cout<<VariableName;
```

Here operator sends the variable to the output device. For example:

```
int sum=9;
cout<<sum;
```

Here operator<< will send the value 9 to computer screen.

If we want to send more data to output stream, then can use either separate statements or cascade operator<< as given below:

```
cout<<var1<<var2<<var3;
```

For example

```
int  area = 5;
```

cout<<"Area of triangle :- "<<area<<" mts ";

This code will produce the output

```
Area of triangle :- 5 mts
```

1.1.4.3 get() function :

The get() function reads one character, including at a time, including white space character from the input stream. It is overloaded in istream.

There are two types of get() function-

1) `int get(void);` // it return a character

2) `void get(char *);` // it reads character into variable

For example :

```
char c;
cin.get(c);    // input a character from keyboard and assign it to c
```

or

```
char c;
c = cin.get();   // input a character from keyboard and assign it  to c
```

The details for stream classes for console operations are given below in table 1.1.

1.1.4.4 put() function :

The put() function sends one character at a time to the output stream. It is overloaded in ostream class. It can be used with standard output stream object or with user defined object of the ostream class.

Syntax is :

```
void put(char);
```

For example

```
cout.put('A');     // display the character A
```

or

```
char ch='A';
cout.put(ch);      // display the character A
```

Value addition: Source code

Heading text Count number of characters in a text.

```
/* Program that accepts a text and terminates by newline character. It counts
number of characters in the text  */

#include<iostream.h>
 int main()
 {
    int  count = 0;                                  // initialize count to 0
    char ch;
    cout<<"enter the text :  "<<endl;
    ch = cin.get();                                  // input character
    while ( ch != '\n' )                             // check for newline character
    {
              cout.put(ch);                       // prints a character
              count++;                             // increments one character
              ch= cin.get();                      // reads one character
     }
     cout<<"\n Number of characters =  "<<count<<endl;//print total character
```

```
    getchar();
 }
```

Output is :

```
enter the text :
computer science
computer science
Number of characters =  16
```

Source: self

1.1.4.5 getline() function :

As the name implies, getline() function reads a line of text at a time.

Syntax is :

```
void getline( line, size );
```

This function reads characters into variable line. We can terminate either by reading newline character '\n' or by reading size-1 characters. The newline character is read but not saved and it is replaced by null character.

For example

```
char text[15];
cin.getline(text,15);
```

If we input

```
Computer Deptt < press enter>
```

then output will be

```
Computer Deptt
```

Here size of text variable is 15. So input will be terminated after 14 characters (size – 1). In this example complete text will be printed.

If we input

```
computer science < press enter>
```

then output will be

```
computer scien
```

Here size of text variable is 15. So input will be terminated after 14 characters (size – 1).

1.1.4.6 write() function

Write () function displays a line at a time.

Syntax is:

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

line is the name of string to be displayed.

The size is the number of characters to be displayed.

If size is more than the number of character then write() function sends the character beyond the line otherwise it will send the characters equal to the Size.

Value addition: Program2 Source code

Heading text : Demonstrate getline() and write( ) function

```
// Program accept line as an input and print the same using write()
#include<iostream.h>
#include<string.h>
void main()
{
    char line[60];
     int length;
     cout<<"enter the line :  \n";
     cin.getline(line,60);
     length = strlen(line);
     cout<<"\nline that you entered  : \n";
     cout.write(line, length );
     cout<<"\n";
     getchar();
}
```

Output is :

```
enter the line :
Hard work is the key to success.

line that you entered  :
Hard work is the key to success.
```

Source: Self

1.1.5 Formatted I/O operations

Now, You are aware of I/O functions, but the output that you want to print is not formatted. If you want output in proper format you have to adjust as per your requirement again and again. And many times you do not have any control. We can control input, as we enter data ourselves or by initializing the values, but we cannot control the format of output. To solve the problem, C++ provides us a number of features to format the output.

These features are :

- ios class functions and flags
- Manipulators

1.1.5.1 ios class functions and flags:

Following functions provide us very efficient way of formatting:

- width()
- precision()
- fill()
- setf()
- unsetf()

Value addition: Table 1

Heading text : Summary of ios class functions and flags

Function	Purpose
width( )	Specifies the minimum number of columns(field size) to display a value. By default, displayed values are right justified.
precision( )	Specifies number of digits to be displayed after decimal point. The fractional part is round off to specified number of digits.
fill( )	Specifies a character to be used to fill the unused area of a display if the number of columns mentioned is more than the character in an output. By default, fill character is white space.
setf( )	Sets the format flags that do left and right justification.
unsetf( )	Clear the specified flags

Source:

Let us discuss them in detail:

1.1.5.1.1 Defining Field width using width( ) function:

Width( ) function is used to define the width of a field for the output of an item. It is a member function of the ios class.

Syntax is :

int width(int);

It is invoked on an object of output stream.

cout.width(w);

→As width( ) is a member function we have to use an output object to invoke it.

It specifies the width for the item that follows it immediately. After displaying that item ,the default width activates.

→ Default width is the number of columns that is equal to the number of characters in the output item.

For example

```
int a = 123;
int b = 55
int c = 90;
cout.width(4);
cout<<a;
cout.width(5);
cout<<b<<c;
```

output will be

	1	2	3				5	5	9	0

→ C++ never truncate the values, if specified width is smaller than the size of the value, C++ expands the width of field to fit the value.

1.1.5.1.2 precision( ) function

In floating point numbers six digits are printed after decimal point that make presentation of data quiet odd. So if we want to set number of digits after decimal point than we can use precision( ) function. This function is a member function of ios class.

Syntax is :

int precision(int);

As it is a member function , an object of ios class will be used to invoke it.

```
cout.precision(d);  // d is number of digits right to decimal point
```

→ precision() function retains the setting till you reset it.

For example

```
float x = 34.24432;
cout<<"value of  x = "<<x<<endl;
cout.precision(2);
cout<<"value of  x = "<<x;
```

Output will be :

```
value of  x = 34.24432
value of  x = 34.24
```

1.1.5.1.3 fill() function

When mentioned width for printing the value is large and the value is small then unused fields are filled with white spaces by default. But if we want to fill that space with some other character then we can use fill( ) function.

It fills unused fields with the character we desire.

Syntax is :

```
char fill( char );
```

Here char argument specify the fill character.

As fill( ) is a member function of the ios class so it is invoked through an object of output stream.

```
cout.fill(char);
```

For example

```
int a = 234;
cout.fill('#');
cout.width(8);
cout<<a<<endl;
```

Output will be

#	#	#	#	#	2	3	4

1.1.5.1.4 setf() function

When we print a value using width() function then printed value is right justified. But generally we want value to be left justified. This facility is provided by setf() function. The setf() function provides a number of formatting facilities.

The setf( ) function is a member function of ios class and is invoked through an object of output stream.

Syntax is :

long setf (long setbits, long field);

long setf (long setbits);

Here first argument setbits is one of the flags defined in ios class. It specifies action required for the output. Second argument known as bit field is an ios constant, specifies the group to which the formatting flag belongs.

setf stands for **set flags.** It sets flag and bit fields.

Following table shows **flags** and **bit fields** for setf( ) function and their related actions.

Flag	Field	Action on output
Left Right Internal	Adjustifield	left justified output. Padding occurs before the sign or the base indicator. Right justified output. Padding occurs after the number. Padding occurs between the sign or the base indicator and the number when the field width is more than the required value.
Scientific Fixed	Floatfield	Float values are output using exponential normalized floating point notation. Float values are output using ordinary floating-point notation (without exponent).
Doc oct hex	Basefield	Integer value is output in the decimal base Integer value is output in the octal base Integer value is output in the hexadecimal base

Table 1.1 Flags that have bit fields for setf( ) function

Following table shows **flags** for setf( ) function and their related actions.

Flag	Action on output
Showbase	Show base indicator on output
Showpoint	Show decimal point and trailing zeros in the output
showpos	Output '+' before positive numbers. By default, If the number is positive then the sign character is not printed
uppercase	Use uppercase letters when the integer number is output using hex base.
skipws	Skip white space on input.
unitbuf	Write output stream immediately i.e. flush stream after insertion.

Table 1.2 Flags without bit fields for setf( ) function

Examples of the flags with and without field are given below:

Justify the Output

We can justify output to left, right or internal as per user's requirement.

Left justify prints the output to left side.
Right justify prints the output to right side.
Internal justify prints sign first then fill the extra space with padding and finally print the value.

for example:

Code is:

```
int x=1200;
int y=-456;
cout.setf(ios::right, ios::adjustfield);
cout.width(12);
cout.fill('*');
```

```
cout <<y<<endl;
cout.setf(ios::left, ios::adjustfield);
cout.fill('#');
cout <<x<<endl;
cout.setf(ios::internal, ios::adjustfield);
cout.fill('^');
cout <<y<<endl;
```

Output is :

*	*	*	*	*	*	*	*	-	4	5	6
1	2	0	0	#	#	#	#	#	#	#	#
-	^	^	^	^	^	^	^	^	4	5	6

Use basefield of your choice

You can use numeric flag for your choice and can have octal and hexadecimal values. Turn on the corresponding flag that you want. The choice for entering data can be any the following formats: decimal, octal, hexadecimal.

For Decimal data : Value does not start with a 0.
For Octal data : Value starts with 0.
For Hexadecimal data : Value starts with 0x or 0X.

For example if you want 25 in these three formats, then result will be like this :

In Decimal : 25
In Octal : 031
In hexadecimal : 0X19

Value addition: Source Code

This program prints the given integer in three different bases: decimal, octal and hexadecimal.

```
#include<iostream.h>
int main()
{
   int a;
   cout<<" Enter the any integer :  ";
   cin>>a;
   cout<<"Value of "<<a<<" in three bases is given below\n "<<endl;
  cout.setf(ios::showbase);     // showbase to indicate base
  cout.setf(ios::uppercase);    // activate uppercase
```

```
        cout.setf(ios::dec, ios::basefield);  // set base to decimal
        cout<<"  In decimal form      :";
        cout.width(6);
        cout<<a<<endl<<endl;
        cout.setf(ios::oct, ios::basefield);   //set base to octal
        cout<<"  In Octal form       :";
        cout.width(6);
        cout<<a<<endl<<endl;
        cout.setf(ios::hex, ios::basefield);   // set base to hexadecimal
        cout<<"  In hexadecimal form  :";
        cout.width(6);
        cout<<a<<endl;
        cout.unsetf(ios::hex);                 // clear the  hex flag
        cout.unsetf(ios::uppercase);           // clear the uppercase flag
        cout.setf(ios::dec);                   // set base flag to decimal
        getchar();
        return 0;
      }
```

Output is :

Enter the any integer : 32

Value of 32 in three bases is given below

  In decimal form   :   32

  In Octal form   :   040

  In hexadecimal form : 0X20

Source: self

Using floatfield flag

In scientific notation, the significant and exponent are specified separately. Significand part is a floating point number that contains significant digits. The exponent specifies the magnitude of the number. It may be positive or negative. For example:

4.561200e+05

In fixed point notation, the number is displayed in normal decimal point format.

4561.20000

Value addition: Source Code

This program prints the given floating point number in scientific and fixed point notation.

```
#include<iostream.h>
int main()
{
   float a;
   cout<<"  Enter the floating point number :   ";
   cin>>a;
   cout.setf(ios::scientific, ios::floatfield);
   cout<<"\n  Scientific notation  :  "<<a<<endl;
   cout.setf(ios::fixed, ios::floatfield);
   cout<<"\n  Fixed point notation:  "<<a<<endl;
   getchar();
   return 0;
}
```

Output is:

```
  Enter the floating point number :   6733.89
  Scientific notation  :  6.733890e+003
  Fixed point notation:  6733.890137
```

Source: self

Display '+' sign in output

Till now sign is printed with negative number not with positive number. But it is also possible to print '+' sign with positive number. By using **setf()** function with **showpos** flag as an argument. If **showpos flag** is set, then '+' sign is printed with positive number.

Syntax is :

```
cout.setf(ios::showpos);
```

For example :

```
float x = 5.78, y = 134.8;
cout<<" values of x, y without showpos  :"<<endl;
cout<< x<<endl<<y<<endl;
cout.setf(ios::showpos);
```

```
cout<<" values of x, y showpos  flag :"<<endl;
cout<<x<<endl<<y;
```

output will be:

```
values of x, y without showpos  :
5.78
134.8
 values of x, y showpos  flag :
+5.78
+134.8
```

Display trailing zeros in output

If we want to print 19.00 or 45.50 then the output will be 19 or 45.5 . Here trailing zeros have been truncated. If we want to print these zeroes then use setf() function with showpoint flag.

Syntax is :

```
cout.setf(ios::showpoint);
```

Example :

```
float x = 5.00, y = 4.30;
cout<<" values of x, y without showpoint  :"<<endl;
cout<< x<<endl<<y<<endl;
cout.setf(ios::showpos);
cout<<" values of x, y with showpoint  flag :"<<endl;
cout<<x<<endl<<y;
```

output will be:

```
values of x, y without showpoint  :
5
4.3
values of x, y with showpoint flag :
5.00
4.30
```

Skip White Space using skipws:

By default this flag is set on. That's why C++ skips leading white space while reading. But if you want to read white space, then you have to turn off this flag using unsetf() function. The syntax is :

```
cin.unsetf(ios::skipws);
```

Unit Buffer

It ensures that output has been written before the program terminates. If program is going to abort and we want any pending output be written. Then set unitbuf flag on.

stream.setf(ios::unitbuf)

Here stream is output.

1.1.5.2 Formatting with Manipulators:

Manipulators are defined in iomanip.h and used to manipulate the output formats. They provide same features as the member functions of ios class, but more convenient to use.

They are defined in two categories:

1) Parameterized manipulators : Takes argument
2) Non parameterized manipulators : Takes no argument

Parameterized manipulators:

Manipulators	Desciption	Corresponding Member function in ios class
setfill(int)	Fill character	fill(char)
setw(int)	Sets the width of field	width(int)
setprecision(int)	Sets floating point precision	precision(int)
setiosflags(long)	Sets the format of flag	setf(long)

Table 1.3 List of Parameterized manipulators

Non parameterized manipulators

Manipulators	Deascription
endl	Insert newline character('\n') to the output stream and flush it.
ends	Outputs a NULL character('\0') to the output stream.
flush	Flushes the output stream.
dec	Set the base as decimal

oct	Set the base as octal
hex	Set the base as hexadecimal
fixed	Set floating value as fixed point
scientific	Set floating value as scientific notation

Table 1.4 List of Non Parameterized manipulators

Output Manipulators (no args)

Example:

```
int  x = 45;
cout << oct << x << endl;        // Output is 55
cout << hex << x << endl;        // Output is 0x2d
cout << dec << x << endl;        // Output is 45
```

Output Manipulators (1 arg)

Program to demonstrate the use of predefined manipulators

```
#include<iostream.h>
#include<iomanip.h>
    int main()
        {
            cout << setiosflags(ios::fixed | ios::showpoint);
            cout<< setw(7);
            cout<< setprecision(2);
            cout<< setfill('*');
            cout<< 1234.267 << endl;
            getchar();
            return 0;
         }
```

Output is :

```
        1234.27
```

1.2 File Handling - I

Programs we have done till now accepts the input from the keyboard at the time of execution. Any data or instructions that are to be processed are present in main memory. When processing completes, operating system reallocates the space occupied by processed data and instructions to some other program and its data and instructions. So, every thing stored in previous program is rewritten or not available. If we want that data again then we have to execute the program again. Main problem to this approach comes when we handle large amount of data. For example if we have to enter data for 100 students or more, what will happen……..

- It takes a lot of time.
- If we make a mistake then we may have start from beginning.
- If same data is to be processed again at some later stage, then we have to enter the whole data again.
- And…..if we need initial data or processed data after some days, then what will we do? Either repeat the process (as RAM is volatile) or store the data at the place from where we can access as and when required. And here comes the **Files**.

1.2.1 Learning objectives

In this chapter you will learn the basics of file handling in C++ and able to :

- Define different types of C++ Stream Classes
- Use the data file for I/O operations
- To open and close files
- Use different types of file modes like output, append, truncate and many more…

1.2.2 C++ stream classes

You have read about streams in previous section. Now we will study streams that you use in files. C++ I/O system contains the hierarchy of classes (known as stream classes) that define various streams to handle console and disk files.

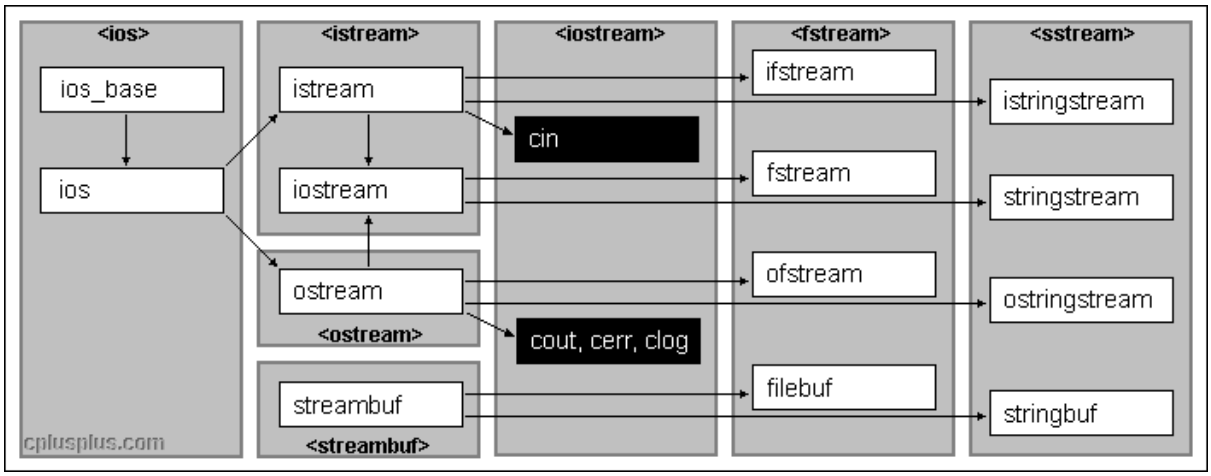

Figure 1.3 Hierarchy of console stream classes(Source cplusplus.com)

Here ios is the base class for istream (for input) and ostream (for output). istream and ostream are base classes for iostream(input/output stream). The class ios provides the support for formatted and unformatted I/O operations.

These classes are declared in the header file iostream.h. So this file should be included in all programs that communicate with the console unit.Data coming in or going out of the computer can be viewed as an infinite stream of bytes, and the stream library provides controlled access to this data.

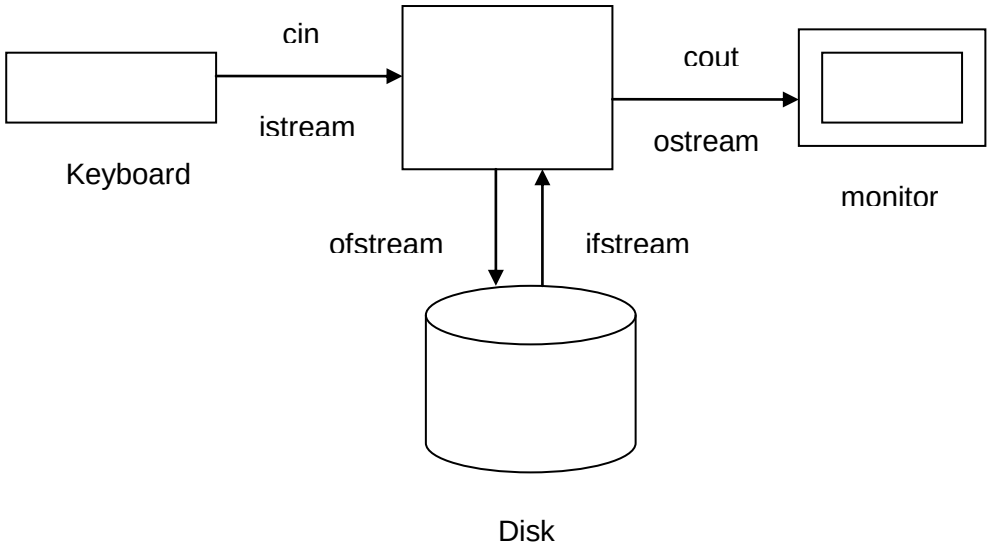

Figure 1.4 Flow of data using secondary storage device

The above diagram illustrates streams **cin** and **cout**, the console I/O streams. You can use the same stream mechanism, with minor adjustments, to do I/O to files .

C++ I/O system contains a set of classes for handling data files.

Description of the classes is given below:

Class Name	**Description**
filebuf	It sets the file buffers to read and write. It is one of the two classes defined in the stream library that provide a place for input to be taken from and a place for output to go. The I/O functions of class istream and ostream makes calls to the functions of filebuf to do the actual insertion or extraction on the streams.It occurs when the bp data member of class ios has been assigned a pointer to class filebuf.
Fstreambase	It is the base class for all file stream classes. hence supports operations common to file streams contains open() and close() function in addition

	to other member functions.
ifstream	Provides input operations. Contains open() with default input mode. Inherits get(), getline(), read(), seekg() and tellg() functios from istream class
ofstream	Provides output operations. Contains open() with default output mode. Inherits putt(), write(), seekp() and tellp() functios from ostream class
fstream	Provides support for both input and output operations. Contains open() with default input mode. Inherits all the functions from istream and ostream classes through iostream.

Table 1.4 Description of classes

1.2.3 The process of using a File in C++:

A file is a collection of data stored on secondary storage devices (hard disk, floppy disk, pen drive etc.). As secondary memory is non volatile, i.e.whatever we store in a file remains there and we can use it at any time.

Therefore, if data is input for a program, processed and finally stored in a disk file, then the program can read the file from the disk whenever required and that too at very fast speed (very fast than human typing).

Also, the output of one program may become input for another program.

Almost all real world programs save data in files. Word processors, spreadsheets, Database management system, C++ programs etc. are the examples that use files.

File Names

Every file has a name that is used for the identification by the operating system and the user. Each operating system has some rules for naming the file. For example MS-DOS allows file names of maximum of eight characters with an optional three character extension.

File name identifies file's purpose and the extension identifies the type of information in the file.

Following steps are taken to process a file:

1) Naming a file: First step is to select proper name. File name should indicate the type of its contents and purpose. Some examples of valid name are text.dat, input.dat, output.dat, file1.txt, datafile.txt .

2) Opening a file: File must be opened. If the file does not exist, then it must be created. File should be opened in appropriate mode. We will discuss different types of mode later on.

3) Reading from / writing onto a file: Once a file is opened data can be read from a file or written to a file in a number of ways.

4) Closing the file: Once the program finished using the file, the file must be closed.

Set the program for file I/O

A C++ program must be set properly before file I/O is performed.

As iostream.h is required for cin and cout, fstream.h header file is required to access file operations in C++. File processing in C++ is performed using the **fstream** class. **fstream** is a complete C++ class with constructors, a destructor and overloaded operators and it contains all the declarations required for file operations. It is included with the following statement:

```
#include<fstream.h>
```

You can declare an instance of an **fstream** object, while using file. If you do not know the name of the file you want to process, you can use the default constructor. The **fstream** class provides two classes for file processing. One for writing into a file and the other for reading from a file. The fstream.h header file declares the data types ofstream, ifstream and fstream. Before using a file in a program, it must declare an object of one of these data types.

1.2.4 Opening a File

Before writing and reading, the file must be opened first. For opening a file, First create a file stream then link it with the file. A file stream can be from class ifstream, ofstream or fstream that are defined in fstream.h header file. The requirement of class depends on the purpose i.e., whether you want to read from the file or write to the file.

A file can be opened in two ways:

Using constructor of the stream class..

Using the function open().

1.2.4.1 Opening a file using constructor

As we have studied earlier constructor is invoked to initialize an object while it is created. Similarly, the constructors of stream classes (ifstream, ofstream or fstream) are used to initialize file stream objects with the filenames passed to them.

If you want to input data from **keyboard** and save data in a **file** then use **ofstream** object. The syntax of constructor of ofstream class is:

```
ofstream (const char* FileName, int FileMode);
```

The **ofstream(const char*** FileName**, int** FileMode**)** constructor provides a method for creating a file. It does this with two arguments. The first argument, **FileName**, is a

string that specifies the name of the file that needs to be saved. The second argument, **FileMode,** specifies the kind of operation you want to perform on the file (table of file modes, is given in next topic).

Similarly, If you want to open an already existing file to have access to its contents, C++ provides the **ifstream** class. Like **ofstream**, the **ifstream** class provides various constructors you can use. If you have enough information about the file you want to open, you can use the following constructor:

ifstream(const char* *FileName*, int *FileMode*);

The first argument of the constructor, FileName, is a constant string that represents the file that you want to open. The second argument, **FileMode,** specifies the kind of operation you want to perform on the file(table of different file modes, is given in next topic).

You can use **fstream** for above options. The syntax is:

fstream(const char* *FileName*, int *FileMode*);

Here file mode will include both input and output mode

When you open a file, you should provide the path that specify its location as well as its name if the file is not present in current directory. For example, if you want to open a file on A: drive of a DOS or Window computer, you have to specify file's drive and path. If you want to open a file located on some secondary device say A: then you will write

```
ofstream outputfile("A:\\data.txt");
```

In the above statement, the file "A:\\data.txt" is opened and linked with outputfile.

Following examples demonstrate the opening of file using constructor.

i
```
ifstream infile("text1.dat");  // opens a file named as text1.dat for input mode.
char ch;
infile>>ch                     // read a character from the file "text1.dat"
```

ii
```
ofstream outf("text.dat");  //opens a file named as text.dat for output mode.
int sum=20;
out<<sum;                     // stores value of sum variable in file "text.dat"
```

iii
```
fstream file1("text.dat");  //opens a file named as text.dat for both input and    //
                              output mode.
```

iv
```
fstream dfile("sum.dat", ios::in | ios::out);  //open  the file "sum.dat" in both I/O
                                                //mode
```

Following program uses << operator to store data.

Here is a program that demonstrate the usage of << operator for saving data in a file. It uses ofstream constructor.

```
#include <fstream.h>
#include<iostream.h>
int main()
{
   char FirstName[15], LastName[20];
   int Age;
   char FileName[10];
   cout << "Enter First Name: ";
   cin >> FirstName;
   cout << "Enter Last Name:  ";
   cin >> LastName;
   cout << "Enter Age:       ";
   cin >> Age;
   cout << "\nEnter the name of the file you want to create: ";
   cin >> FileName;
   ofstream Students(FileName);              // here you create an output file
   Students<<FirstName<<endl<<LastName<<endl<<Age;  // stores data
   cout<<endl;
   return 0;
}
```

Output will be :

```
Enter First Name: Ishita
Enter Last Name:  Singhal
Enter Age:       7

Enter the name of the file you want to create: name.dat
```

Data of file name.dat is

```
Ishita
Singhal
7
```

Table 1.5

Similarly, you can take data from above created file and display on the screen. Following program demonstrate the use of ifstream constructor.

Value addition: Source code

Here is a program accepts data from file "name.dat" and display on screen. "name.dat" file we have created in earlier. It uses ifstream constructor.

```
#include <fstream>
#include <iostream>
using namespace std;
int main()
{
   char FirstName[15], LastName[10];
   int Age;
   char FileName[10];
   cout << "Enter the name of the file you want to open: ";
   cin >> FileName;
   ifstream Students(FileName);
   Students >> FirstName >> LastName >> Age;
   cout << "\nFirst Name: " << FirstName;
   cout << "\nLast Name:  " << LastName;
   cout << "\nEnter Age:  " << Age;
   getchar();
   return 0;
}
```

Output is:

```
Enter the name of the file you want to open: name.dat
First Name: Ishita
Last Name:  Singhal
Enter Age:  7
```

Source : Self

1.2.4.2 Opening a file using open() member function

You can also open a file using the **open()** method. The syntax of the open method is:

void open(const char* FileName, int FileMode);

This method behaves exactly in the same way, as the constructor described above. The first argument represents the name of the file you are using and the second argument,

FileMode, specifies the kind of operation you want to perform on the file (table of file modes, is given in next topic).

The open() function is invoked on a file stream object as

```
FileStreamObject.open(" Filename",file_mode);
```

For example

```
ofstream outfile; // the default constructor to declare an ofstream variable,

outfile.open(" data.txt", ios::out);
```

Here file named data.txt is opened for output with read and write permissions.

The **fstream** class in this case is declared as **ofstream**, the compiler is aware that you want to save a file (as the use of **ofstream** means that you want to write to a file, in other words you want the FileMode with a value of **ios::out**), you can call the **open()** method with only the name of the file.

For example

```
ofstream outfile;

outfile.open(" data.txt");
```

Similarly you can open file in input mode.

For example

```
ifstream infile; // the default constructor to declare an ifstream variable,

infile.open(" input.txt", ios::in);
```

Here file named input.txt is opened in input mode with read and write permissions.

As discussed earlier, if file is not in working directory then mention path that specify its location as well as its name. For example If you want to open a file located on secondary storage device drive then you will write

```
infile.open("A:\\data.txt");
```

In the above statement, the file "A:\\data.txt" is opened and linked with infile.

1.2.5 File modes :

When processing a file, you will specify the type of operation you want to perform. The operation is specified by the **file mode**. It can be one of the following:

Mode	**Description**
Ios::in	If FileName is a new file, then an empty file is created. If FileName already exists, then it is opened and its content is made

	available for processing.
ios::out	If FileName is a new file, then an empty file is created. Once you create an empty file, you can write data to it. If FileName already exists, then it is opened, its content get destroyed, and you can write new data to it. And when you save the file, the new contents are saved.
ios::ate	If FileName is a new file, data is written to it and subsequently added to the end of the file. If FileName already exists and contains data, then it is opened and data is written in the current position.
ios::app	If FileName is a new file, data is written to it. If FileName already exists and contains data, then it is opened, the compiler goes to the end of the file and adds the new data to it.
Ios::trunc	If FileName already exists, its content is destroyed and treated as new file.
Ios::nocreate	If FileName is a new file, the operation fails because it cannot create a new file. If FileName already exists, then it is opened and its content is made available for processing.
Ios::noreplace	If FileName is a new file, then an empty file is created. If FileName already exists and you try to open it, this operation would fail because it cannot create a file of the same name in the same location.

Table 1.5 File modes

Several modes may be used together by connecting them with bit-wise OR operator (|). Suppose you want to open a file in both input and output mode i.e information may be written to and read from the file then we write

```
fstream dfile("sum.dat", ios::in | ios::out);
```

If you want to open a file in such a way that information will only be written to its end then write the following statement:

```
fstream dfile("sum.dat", ios::out | ios::app);
```

1.2.6 Closing the File

After using a file, you should close it. This is taken care by using the **ofstream::close()** method. A file is closed by disconnecting it with the stream it is attached with. The syntax is:

```
void close();
```

For example

infile.close()';

It closes file associated with file stream object infile irrespective of file opened for reading, writing or others.

The close() function flushes the buffer before terminating the connection of the file with the stream.

1.3 File Handling - II

In section 1.2, you have learnt opening and closing of files in specific modes. What will happen if you try to open a file to see the contents of that file and file does not exist? Or want to write on a file that already contains some different data. How do you know the end of a file? In this section you will learn how to handle above situations. Here we study about text files, reading and writing data in it.

1.3.1 Learning objectives

After reading this section you should be able to:

- Handle different type of errors that occur in file operation by using eof(), bad(), good(), fail().
- Read and write characters in text file
- Write different programs that use text files i.e. opening, closing, reading, writing data in multiple files.

1.3.2 Error Handling during file operations:

You may face problem during following situations while handling the files:

- trying to open a file for reading that does not exist.
- giving the name for a new file and that already exists.
- trying to open a file with invalid filename.
- trying to manipulate an unopened file.
- trying to attempt an invalid operation such as reading after the end-of-file, writing into a file that is opened in read-only mode.

There are some errors handling functions that may help you to handle above situations.

i) eof()

ii) fail()

iii) bad()

iv) good()

v) clear()

eof()

This function returns true value(non zero value) if end of file (eof) is encountered while reading the file otherwise it returns false (zero).

For example :

```
while(!file.eof())
{
 ch = file.get();
}
```

fail()

It returns true value if any of open, write or read operation fails, else return zero i.e. false to indicate the success of operation.

For example :

```
ifstream infile;
infile.open("text.dat");
if(infile.fail())
 {  cout<<"unable to open the file text.dat  ";
     return;
 }
```

In this example we have opened a file in input mode and want to read the data from file. By using fail() function we can check whether file can be opened or not (i.e. file does not exist or there is some other problem in reading due to media error file permission does not match).

good()

It returns non-zero value if every thing is fine otherwise it returns zero value.

```
if(datafile.good())
    {
      datafile>>st.a>>st.b;
    }
```

bad()

It returns non-zero value if invalid operation is attempted or any want to read unrecoverable data from media otherwise it returns zero (false) value.

For example :

```
ifstream datafile;
datafile.open("text.dat");
while(!datafile.eof())
{   datafile>>ch;
    if(datafile.bad())
    {
      cout<<" \n error while reading file "<<endl;
      return 0;
    }
}
```

In this example we have opened a file in input mode and reading the data from file. By using bad() function we can check if there is any problem during reading the data from file, If bad() returns true, then there is problem else will return false means we can read data.

1.3.3 Text files

In C++ File system, depending on the way a file is opened for processing, it is classified in two categories: Text file and Binary Files

The mode of opening a specific file tells how files are handled. For example the numbers in the text format are stored as string of characters and in binary format they are stored in the same way as they are stored in computer's main memory. Now we will discuss reading and writing in both types of files.

Text file are those file where we do not include the ios::binary flag in their opening mode. These files are designed to store text and all values that we input or output from/to them. In these files, data is stored in its ASCII code irrespective of the data type.
For character data, ASCII codes of the individual characters are stored.
For numeric data, ASCII codes of the individual digits of a number are stored.
Text files can be manipulated by any text editor. Text files do not provide an efficient storage.

By default files are opened in text mode.

For example:

```
ofstream outfile("text1.dat");
int a = 271;
```

outfile<<a;

Value of a is stored in file like this:

2	7	1	<eof>

In ASCII code :

50	55	49	<eof>

1.3.3.1 Reading and Writing of character data in a text file:

In section 1.2, you have studied a simple way of reading and writing in file when you opened the file using constructors. Here we will learn functions for reading and writing data in the file.

For reading and writing character data we use character I/O member function of ifstream and ofstream classes. For writing in a file, we can use put() and write() function of ofstream class. For reading from a file using character input function get() and getline() of ifstream class can be used. As we have discussed this function with cin and cout, their role is same here. get() function read single character and getline() reads a line or string including whitespace characters.

For example:

outfile.put(ch);

infile.get(ch);

infile.getline(string,80, '|')

getline() read character array string with maximum number 80. Here '|' is a delimiter character of our choice.It mean if this delimiter is encountered, function will stop reading before completing maximum number of characters.

By default '\n' is the delimiter.

Reading and writing in the file using get() and put()

```
#include<fstream.h>
#include<iostream.h>
int main()
{
```

```
            char text[12],ch;
            int length;
            cout<<" enter the file name \n";
            cin>>text;
            ofstream file(text);
            if(file.fail())
            {
               cout<<"unable to open file ";
               return 0;
            }
            cout<<"enter text and terminate by ^z followed by <enter> key\n";
            while(!cin.eof())
            {
                 cin.get(ch);
                 file.put(ch);
             }
            file.close();
            ifstream infile(text);
            cout<<"contents of file : ";
            while(!infile.eof())
            {
                infile.get(ch);
                cout<<ch;
            }
            file.close();
            getchar();
            return 0;
          }
```

Output is :

```
enter the file name
file1.dat
enter text and terminate by ^z followed by <enter> key
Hard work is the key to sucess^Z
^Z
contents of file
Hard work is the key to sucesss
```

\

Value addition: Did you know ?
Role of Buffer
When we operate with file streams, these are always an internal buffer of type streambuf is attached. It is a memory block that acts as an intermediator between the stream and the physical file. For example, when we use an ofstream, whenever member function put() (that writes a single character) is called, the character is not written directly to the physical file with which the stream is associated. Instead of that, the character is inserted in that stream's intermediate buffer.And from the buffer the data goes to file.
Source:

1.3.4 Copy the contents of one text file into another text file

Here we will use above information to copy the contents of one file into the another.So, What steps you are going to follow?

Here you need two files, One for input and other for storing output. So open these files in corresponding modes.

```
ifstream file(file1);  // for input

ofstream outfile(file2);  // for output
```

(The file we open in input mode should contain data. So either put data in the file by opening it in output mode through programming or store data in the file using MS-Word, wordpad etc.)

After trying to open, You have to check whether file is opened in desire form or not.

For that you can use fail() function. If it retruns true then file is not opened, otherwise we are successful in opening the files.

After opening the files. start reading character by character from the file you have opened in read mode. You can take care of end of file using the following statement.

```
while(!file.eof())
```

While reading , if error occurs check it by using error handling functions bad() as given below:

```
if(file.bad())  // if true error in reading else start reading
```

After reading character using get() function, start putting the character in destination file using put() function. Again check whether data is written on file or not by using bad() or

good() functions.If it is written. Well you are successful in putting all data from source file to destination file (as you have used while loop).

In the end close both the files.

C++ code with the output is given below.

//Program that copies the contents of source file to destination file

```
#include<fstream.h>
#include<iostream.h>
#include<iomanip.h>

int main()
{
  char file1[12],file2[12],ch;
  int length;
  cout<<" Enter the file name of source file : ";
  cin>>file1;
  ifstream file(file1);
  if(file.fail())
  {
     cout<<"Unable to open "<<file1<<"file "<<endl;
     return 0;
  }
   cout<<" Enter the file name of destination file : ";
   cin>>file2;
   ofstream outfile(file2);
   if(outfile.fail())
   {
     cout<<"Unable to open "<<file2<<"file "<<endl;
     return 0;
    }
   while(!file.eof())
  {
    ch = file.get();
    if(file.bad())
{
```

```
          cout<<"\n Error in reading the file  "<<file1<<endl;
          return 0;
        }
           outfile.put(ch);
           if(outfile.bad())
        {

          cout<<"\n Error in writing the file  "<<file2<<endl;
          return 0;
          }
        }

        cout<<"\n file copied successfully.  \n";
        file.close();
        outfile.close();
}
```

Output is:

Enter the file name of source file : ddd

Enter the file name of destination file : bbb

Contents of file ddd (we have stored data in this file i.e. source file)

To every action there is equal and opposite reaction

We all are Indian first and last.

Contents of file bbb (destination file)

To every action there is equal and opposite reaction

We all are Indian first and last.

1.4 File Handling – III

Till now we have studied basics of files, reading and writing in text file and handling of error that occurs during file operation. Although text file store information in the form that is easily readable, but it takes more space as information is in ASCII form. But if we

want to store blocks of information by using single statement and in efficient manner then it is not a good solution. For these purpose we can use binary files.

The programs we have done till now do not take inputs from command line. In this section you are going to learn how can an argument be taken from command line and used in the program.

1.4.1 learning Objectives

After reading this section you are able to

- Open binary file for reading and writing.
- Store or retrieve arrays, structures and object of classes in or from a file.
- Write programs that uses binary files.
    - Pass arguments from command line.
    - Write programs using command line arguments.

1.4.2 Binary Files:

All the files we are working till now are text files. That means the data is stored as ASCII text in the file. Whatever data we stored with the <<operator, is converted to text.

By default files are opened in text mode.

For example:

```
ofstream outfile("text1.dat");
 int a = 271;
outfile<<a;
```

Value of a is stored in file like this:

2	7	1	<eof>

In ASCII code (as you have seen earlier):

50	55	49	<eof>

but if it is formatted as a binary number,it will occupy two bytes(size of short int type is 2 bytes).So binary mode use memory efficiently.

Information can also be stored in a file in binary format.

In binary files, data is stored in the same way as it is stored in memory irrespective of the data type. For character data, their ASCII codes of the individual characters are stored. For numeric data, numbers are stored in binary format.
Binary files provide more efficient way of storage in memory, but we can read the binary files only through programs. So, Binary files can not be manipulated by any text editor.
In binary files, input and output data with the extraction and insertion operators (<< and >>) and functions like getline is not efficient, as we do not need to format any data, and data may not use the separation codes used by text files to separate elements (like space, newline, etc...).
For opening a file in binary mode use ios::binary flag.

For example :

```
file.open("text.dat", ios::out | ios::binary);
```

file is opened in both output and binary modes.

Here, File streams include two member functions specifically designed to input and output binary data sequentially: write() and read(). The first one (write) is a member function of ostream inherited by ofstream. And read is a member function of istream that is inherited by ifstream. Objects of class fstream have both members.

1.4.2.1 write () member function:

To store binary data to a file write () member function is used.

Syntax of write() is:

```
file.write((char *)buffer, sizeof(buffer));
```

(char *)buffer This argument is the starting address of the memory that is to be written to the file. Here buffer is the name of an array. The address is always in pointer-to-char form, so cast operator is applied here.

sizeof(buffer) This argument is the size of the data being written(in byte).

1.4.2.2 read() member function:

Read() function is used to read unformatted data from a file into memory.

Syntax is :

```
file.read((char *)buffer, sizeof(buffer));
```

Here

(char *)buffer This argument is the starting address of the memory that is to be read from the file. Here buffer is the name of an array. Here too address is in pointer-to-char form, so cast operator is applied here.

sizeof(buffer) This argument is the size of the data being read(in byte).

1.4.2.3 Reading /write an array in a file

You can read and write an entire array in a file just by a single statement. This way storing information in binary mode makes life simple. If you want to write and read elements in an **array** a[] having 10 elements use following single write() and read() functions .

Statements are:

```
outfile.write((char )a, 10*sizeof( int ));      // for writing
infile.read((char )a, 10*sizeof( int ));        // for reading
```

Value addition: Source Code

Program demonstrating reading and writing an array in binary file.

```
/* This program firstly writes an array a[] in a file then read from file in
other array b[] and display all the values of array b[].                    */
 #include<iostream.h>
 #include<fstream.h>
 int main()
 {
   int a[6]= { 10,20,30,40,50,60};
   int b[6];
   ofstream ofile("text.dat",ios::binary| ios::out);     //open text.dat for writing
   if(ofile.fail())                                      // check if file can be opened
   {
      cout<<"\n Unable to open the  file "<<endl;
      return 0;
   }
   ofile.write((char *)a, 6*sizeof(int));                // writes array in the file
   ofile.close();                                                //  close the file
   ifstream ifile("text.dat",ios::binary| ios::in);          // open file for reading
   if(ifile.fail())                                     // check if file can be opened
   {
      cout<<"\n Unable to open the  file "<<endl;
      return 0;
   }
```

```
    ifile.read((char *)b, 6*sizeof(int));  //reads the array from file & store in b[]
    ifile.close();                                                    // close the file
    cout<<"\n Array read from binary file: \n"<<endl;
    for(int i=0;i<6;i++)
    cout<<"b["<<i<<"]   =  "<<b[i]<<endl;                  // displays on screen
    getchar();
    return 0;
}
```

Output is:

```
Array read from binary file:
b[0]   =  10
b[1]   =  20
b[2]   =  30
b[3]   =  40
b[4]   =  50
b[5]   =  60
```

Source: Self

1.4.2.4 Reading/Writing structure in a file

You have done structures in earlier chapters. Recall the concept of structures. In a structure we can have a number of data items of different data types. So if you want to store information of an employee, you just declare structure containing employee's code, name, dept., salary. To store the information of this structure we just need single statement that makes storing information in file simple.

We can read and write entire **structure** (For example **student**) in single read() and write() operation as we have done in arrays.

Statements are:

```
outfile.write((char ) &student, sizeof( student));

infile.read((char ) &student, sizeof( student));
```

This Program writes the **structure** student (Data of student) in a file using single write operation, and then reads the same data using read operation.

```
#include<iostream.h>
```

```
#include<fstream.h>

struct student                // this structure contains information of student

{

int rollnum;

char name[20];

int marksmath;

int markscomputer;

int marksenglish;

};

 int main()

{

student stud = { 15, "Ananya", 89, 90, 86 };     // Initializing the values

ofstream ofile( "out.dat", ios::binary | ios::out );      // open file in output mode

if (ofile.fail() )

{


 cout<<"\n Unable to open the file "<<endl;

return 0;

}

ofile.write( (char*) &stud, sizeof(stud));        // writing data of structure in the file

if (ofile.fail() )

{
 cout<<"\n Unable to write data in the file "<<endl;

return 0;

}
```

```
ofile.close();

ifstream infile("out.dat", ios::binary | ios::in );   // open file in input mode

if (infile.fail() )

{
 cout<<"\n Unable to open the file for reading this file  "<<endl;

return 0;

}

infile.read( (char*) &stud, sizeof(stud) );      // Reading the data from file

if (infile.fail() )

{
 cout<<"\n Unable to read from the file "<<endl;

return 0;

}

infile.close();

cout<<"\n Information of a student is given below "<<endl<<endl;

          cout<<" Roll number of the student  :     " << stud.rollnum <<endl;
          cout<<" Name of the student         :     " << stud.name <<endl;
          cout<<" Marks in Maths              :     " << stud.marksmath<<endl;
          cout<<" Marks in Computer           :     " << stud.markscomputer<<endl;
          cout<<" Marks in English            :     " << stud.marksenglish<<endl;
           getchar();

           return 0;

}
```

Output :

```
 Information of a student is given below

 Roll number of the student :      15
```

Name of the student	:	Ananya
Marks in Maths	:	89
Marks in Computer	:	90
Marks in English	:	86

1.4.2.5 Reading/Writing an Object of class in file

Similarly, we can read and write an object (For example **stud**) in single read() and write() operation as we have done in above statements.

Statements are:

```
outfile.write((char ) &stud, sizeof( stud));

infile.read((char ) &stud, sizeof( stud));
```

Value addition: Source Code

Program reads and writes the data of an object of a class student in a binary file;

```
/* This program firstly writes data of an object in a file then reads from file in
   other object and display all the values.                    */

  #include<iostream.h>


  #include<fstream.h>
   class student
   {
      private:
         int rollno;
         char name[15];
         float total;
      public:
         student(){ }
         void input()
        {
           cout<<" Enter roll number of the student   :      ";
```

```
        cin>>rollno;
        cout<<" Enter name of the student          :     ";
        cin>>name;
        cout<<" Enter total marks of the student   :     ";
        cin>>total;
      }

      void display()
      {
        cout<<" Roll number of the student         :     " << rollno <<endl;
        cout<<" Enter name of the student          :     " << name <<endl;
        cout<<" Enter total marks of the student   :     " << total<<endl;
      }
  };

int main()
{
   student st;
   char ch = 'y';
   int sno = 0;
   ofstream ofile("text.dat",ios::binary| ios::out);
   if(ofile.fail())
   {
      cout<<"\n Unable to open the  file "<<endl;
      return 0;
   }
   while(1)
   {
     st.input();
     ofile.write((char *)&st, sizeof(st));
     if(ofile.bad())
     {
       cout<<"\n Error while writing to file \n";
       return 0;
     }
     cout<<"\n Want to enter more data (y/n)   :   ";
     cin>>ch;
     if((ch == 'n') ||(ch == 'N'))
```

```
        {
           ofile.close();
           break;
        }
     }
    ifstream ifile("text.dat",ios::binary| ios::in);
    if(ifile.fail())
     {
       cout<<"\n Unable to open the  file "<<endl;
       return 0;
     }
    while(!ifile.eof())
    {
        ifile.read((char *)&st, sizeof(st));
        if(ifile.bad())
      {
           cout<<"\n Error while reading file \n";
           return 0;
        }
        sno++;
        cout<<"\nStudent No  :   "<<sno<<endl<<endl;
        st.display();
        cin.get();
     }
      ifile.close();
      getchar();
      return 0;
}
```

Output is :

```
         Enter roll number of the student    :      1
         Enter name of the student           :      Reena
         Enter total marks of the student    :      70

         Want to enter more data (y/n)       :      y
         Enter roll number of the student    :      2
         Enter name of the student           :      Ajay
         Enter total marks of the student    :      75

         Want to enter more data (y/n)   :   n

         Student's Data is given below :
```

```
Student No  :  1
 Roll number of the student          :    1
 Enter name of the student           :     Reena
 Enter total marks of the student    :    70

Student No  :  2
 Roll number of the student          :    2
 Enter name of the student           :     Ajay
 Enter total marks of the student    :    75
```

Source: Self

1.4.3 Command Line Arguments:

In the programs we have done till now, we never pass arguments or use parameters in main ().Here we are introducing this concept. Command line arguments are used when we invoke a program from command line. They are used to pass the name of a data file or some value you want to input through command prompt.

C> sum text.dat data

Here, sum is the name of the file having program to be executed. text.dat and data are two filenames that are passed to the program as command-line argument.

Command-line arguments are typed by the user and are delimited by a space. The first argument is always the filename that contains the program to be executed.

To read command-line arguments in program, main() function is used. The main() function that we are using till now without arguments, will take two arguments.

Statement is:

```
main(int argc, char * argv[])
```

The first argument argc known as argument counter, represents the total number of arguments in the command line. The second argument argv known as argument vector, is an array of char type pointers that point to the command line argument. The size of this array is equal to the value of argc.

For example, for the command line

c>add abc.dat xxx.dat

the value of argc would be 3 and the argv would be an array of three pointers to strings as given below.

```
argv[0]   =  add       ( the name of the program )

argv[1]   =  abc.dat

argv[2]   =   xxx.dat
```

argv[0] always contains the name of program invoked. Here argv[1], argv[2] are the filename you want to use in program.

Similarly for the statement

c>sum 5

argv[0] i.e. sum, is the file name to be executed and argv[1] is 5.

The following program demonstrate the usage of command-line arguments

This Program calculate the sum of first n natural numbers. The value of n is taken as command line argument.

```
#include<iostream.h>
#include<cstdlib>
main(int argc, char * argv[])
{
int sum=0, i, n;
n = atoi(argv[1]);
for(i = 1; i<=n; i++)
sum = sum + i;
cout<<" Sum of  first "<<n<<" natural numbers  :-  "<<sum;
return 0;
}
```

Output is :
Sum of first 8 natural numbers :- 36
Here we have passed 8 as command line argument

Value addition: Source Code

Program that stores the square of n numbers in a file and then displays data by taking input from file. Value of n and name of file is given by user as command line argument

```
#include<iostream.h>
#include<fstream.h>
#include<cstdlib>
main(int argc, char * argv[])
```

```
{
    int m, i;
    ofstream file1;
    file1.open(argv[1]);
    if(file1.fail())
     {
       cout<<"  Unable to open file \n";
       return 0;
     }
    for( i=1; i<atoi(argv[2]);i++)
     {
       m=i*i;
       file1<<m<<"     ";
      }
    file1.close();
    ifstream infile(argv[1]);
    char ch;
    cout<<"\n Square of First "<<argv[2]<<" numbers taken from file";
    cout<<argv[1]<<endl<<endl;
     do
        {
             infile.get(ch);
             cout<<ch;
         }
      while(infile);
      cout<<endl;
      infile.close();
      getchar();
      return 0;
 }
```

c> program1.cpp 6 ddd

Here, program1.cpp is the name of this program and file name "ddd" and "6" are passed as argv[2] and argv[1] i.e. arguments to this program. Firstly square is calculated and then data is stored in file "ddd".
Then output on screen is displayed by taking data from "ddd" file.

Output is :

```
 Square of First 6  numbers  taken from file  ddd
```

0 1 4 9 16 25
Source: Self

Summary

- A file is a collection of data on some storage device.
- Large volume of data can be handled easily by using files.
- The standard input device is keyboard.
- The standard output device and standard error device is monitor.
- By default, every C++ program has access to cin, cout, cerr and clog streams.
- Input can be performed either by using extraction operator (>>) or by using get() and getline() function of the input stream.
- Output can be performed either by using insertion operator (<<) or by using put() and write() function of the output stream.
- Output of C++ can be formatted or unformatted
- Output of C++ can be formatted by using member functions of ios class or manipulators.
- Use precision() to print fixed precision numbers (3.40 instead of 3.4)
- Set the width of a printing field by width().
- The default fill character is a space.
- Once a file is opened, it may be used exactly as cin is used.
- When reading an entire file, put the file input inside of the loop condition ckecking eof().
- A file can store data either as a text file or as a binary file.
- A file can be opened by constructor of the required stream class or by open() member function of the appropriate stream class.
- For handling file I/O fstream header file is included.
- The ifstream class ties a file to the input stream for input
- The ofstream class ties a file to the output stream for output
- The fstream class ties a file to the stream for both input and output.

- A file mode describes how a file is to be used to read it, write to it, append the data.
- Different file mode are ios::in, ios::out, ios::ate, ios::binary.
- To open an existing file in append mode use ios :: out | ios :: app.
- Data in binary files can be read from and written to file by using read() and write() functions.
- C++ provides some error handling functions. These functions are eof(), bad(), fail() and good().
- File can be opened in two mode: Text file and Binary file.
- You cannot print the contents of a binary file.

Exercises

1 Answer following Questions:

1.1 Define stream? Name the streams used for file I/O.

1.2 What is the role of iomanip.h file?

1.3 What is the difference between manipulators and ios member functions? Give example.

1.4 How are binary files different from text files in C++?

1.5 What are file modes? Explain different types of file mode constants?

1.6 What are different functions for error handling in C++ file I/O?

1.7 Name the library file that provides file I/O operations.

1.8 What are the two ways of opening files ?

1.9 What is the purpose of ios::binary filemode?

1.10 What happens when a file is opened for output and that file does not exist?

1.11 Write a program that reads a text file and copy the contents in other file.

1.12 Write a program that find sum of first n natural numbers and stores the series and sum in a text file.

1.13 Write a program that store records of n students in binary files.

1.14. Write a program that appends the contents of one file to another file. File name is provided by user.

1.15 How data is stored when a file is opened in binary mode?

1.16. Write a program that prompts the user to input the name of text file that already contains some data, read the file and change all lower case characters in uppercase.

1.17 Write a program that copies the odd numbers to one binary file and even numbers in second binary file from a given list of numbers.

1.18 Differentiate between text files and binary file.

1.19 Write a program in C++ to find sum of first n even numbers. The value of n should be taken as command line argument.

a) **1.20 State whether the following statements are True or False.**

b) Manipuators are used to format the output.

c) The ios::ate mode allows to write data anywhere in file

d) The data written with write() function in file can be read with get() function.

e) The ios::app mode is used to read data from the file.

f) The << operator may be used to read data from a file.

g) Several file access flags may be joined by using the | operator.

h) Binary files stores unformatted data.

1.21 Give the statement for following output specifications for printing float values:

(a) 14 columns width

(b) left – justified

(c) Filling of unused places with ^

(d) 3 digits precision

(e) two digits after the decimal point in a five character width.

(f) trailing zeros and '+' sign with positive number

Glossary

address: The physical location in memory where data or program instructions are stored.

append: the mode that allows to add data to the end of a file during processing of file.

ASCII: American Standard Code for Information Interchange. This encoding scheme defines characters for the 128 values in a byte.

binary file: a collection of data stored in the same format in a file, as in memory of computer.

buffer: memory used to store data temporarily that have been read before they are written or processed.

efficency: optimal use of computer resources.

eof: end of file. An ios member function that indicates the end of the file.

File: A collection of related data stored on secondary storage device.

File mode: A designation of file's I/O that define how a file is used in a program i.e. reading, writing, appending.

flag: usedin a program to indicate the presence or absence of a condition.

formatted I/O: By using standard library functions reformat data while read or written.

hexadecimal: a number system with base 16.Its digits are 0-9,A-F.

Insertion operator: C++ operator (<<) which receives data from the program and gives to output object.

manipulator: an I/O function which provides efficient formatting to data.

octal: number system with base 8. Its digits are 0-7.

standard error file: the file to which cerr is connected.

standard input file: the file to which cin is connected.

standard output file: the file to which cout is connected.

Stream: A sequence of bytes.

text file: A file in which data is stored as character.

write mode: file attribute indicates that file is opened for output only.

References

1. B. A. Forouzan and R. F. Gilberg, Computer Science, A structured Approach using C++, Cengage Learning, 2004.
2. R.G. Dromey, How to solve it by Computer, Pearson Education
3. E. Balaguruswamy, Object Oriented Programming with C++ , 4th ed., Tata McGraw Hill

4. S.K. Salaria, Object Oriented Programming using C++ , 2nd ed.,Khanna book publishing.
5. Tony Gaddis, Starting out with C++, 2nd ed., Scott/Jones publishers
6. J. R. Hubbard, Programming with C++ (2nd ed.), Schaum's Outlines, Tata McGraw Hill
7. D S Malik, C++ Programming Language, First Indian Reprint 2009, Cengage Learning
8. R. Albert and T. Breedlove, C++: An Active Learning Approach, Jones and Bartlett India Ltd.

Web Links

1. http://www.gurus4pcs.com
2. http://www.functionx.com
3. http://www.cprogramming.com/tutorial/
4. http://www.cplusplus.com
5. http://www.tenouk.com