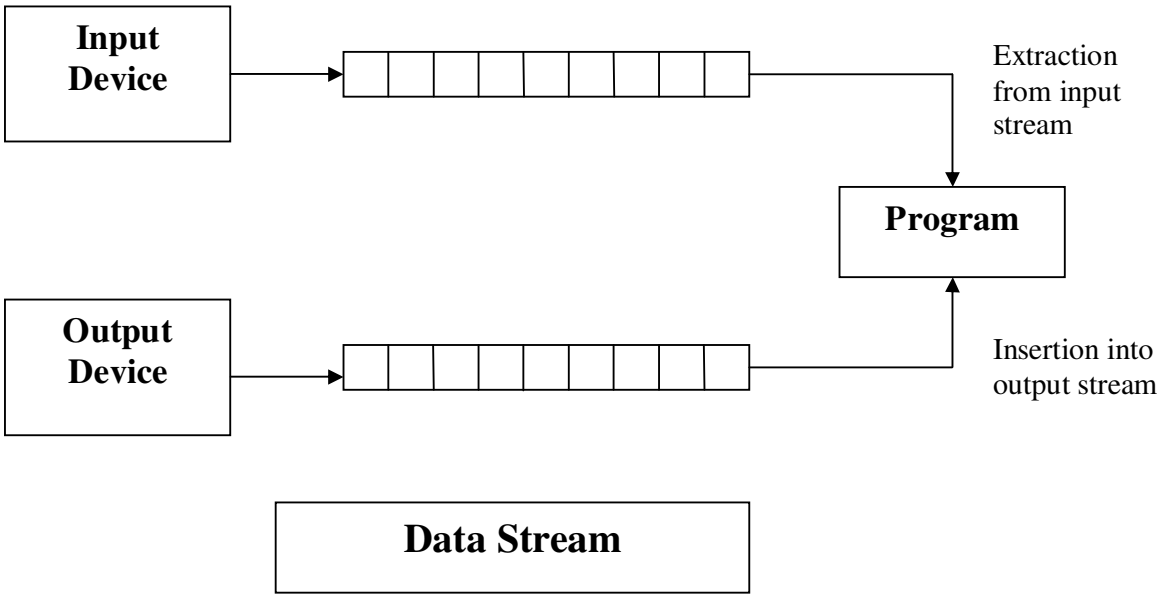
Console I/O Operations

Every program takes some data as input and generates processed data as output following the familiar input output cycle.

C++ Streams

The I/O system in C++ is designed to work with a wide variety of devices including terminals, disks, and tape drives. Although each device is very different, the I/O system supplies an interface to the programmer that is independent of the actual device being accessed. This interface is known as stream.

A stream is a sequence of bytes. It acts either as source from which the input data can be obtained or as a destination to which the output data can be sent. The source stream that 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 other words a program extracts the bytes from an input stream and inserts bytes into an output stream as illustrated in following figure.

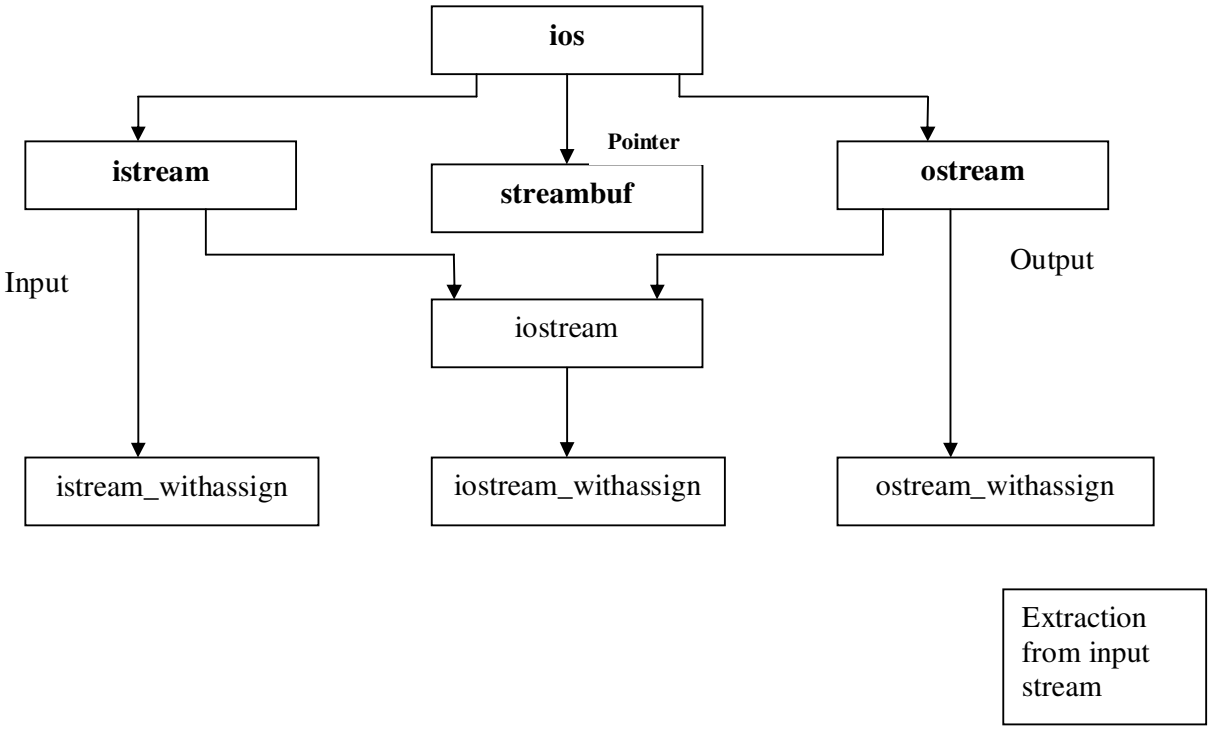


The data in the input stream can come from the keyboard or any other storage devices. Similarly, the data in the output stream can go to the screen or any other storage device. A stream acts as an interface between the program and the input/output device. Therefore a C++ program handles data input or output independent of the devices used.

C++ Stream classes

The C++ I/O system contains a hierarchy of classes that are used to define various streams to deal with both the console and disk files. Theses are called stream classes. Following figure shows hierarchy of the stream classes used for input and output operations with the console units. These classes are declared in the header file iostream.h. This file should be included in all the programs that communicate with the console unit.

The **ios** is the base class for istream (input stream) and ostream (output stream) The class ios is declared as virtual base class so that only one copy of its members are inherited by the iostream.



Class name	Contents
ios	Used by all other input output classes
istream	Inherits the properties of ios and functions get( ) and getline ( )
ostream	Inherits the properties of ios and functions put( ) and write ( )
iostream	Inherits the properties of ios istream and ostream classes.
Streambuf	Provides an interface to physical devices through buffers
Stream classes for console operations	

Unformatted and formatted I/O operations

Overloaded operators >> and <<

We have used the objects cin and cout predefined in the iostream file for the input and output of data of various type. This has been made by overloading the operators >> and << to recognize the basic C++ data types. The >> operator is overloaded in the istream class and << is overloaded in the ostream class. The following format for reading data from the keyboard is used:

cin >> variable1 >> variable2 >> variable3 … … >> variable N

This statement will cause the computer to stop the execution and look for input data from the keyboard. The input data should be separated by white spaces and should match the type of variable in the cin list. Spaces and new line and tabs will be skipped.

The operator >> reads the data character by character and assigns it to the indicated location. The reading for a variable will be terminated at the encounter of a white space or a character that does not match the destination type.

The general form for displaying data on the screen is:

Cout << item1 << iem2 << item3 << … … << itemN

The item1 through itemN may be variables or constants of any basic type.

Example :

```
int code;
cin >> code;
cout << code;
```

put( ) and get( ) functions

The classes istream and ostream define two functions get( ) and put respectively to handle the single character input / output operations. There are two type of get( ) functions. We can use both get(char *) and get(void) prototypes to fetch a character including the blank space and tab and the new line character. The getchar (char *) version assign the input character to its argument and the get(void) version returns the input character.

getline() and write( ) functions

We can read and display a line of text more efficiently by using the lien-oriented input/output functions getline( ) and writ( ) function. The getline ( ) function reads a whole line of text that ends with a new line character transmitted by the RETURN key. This function can be invoked by using the object cin as follows.

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

example :
```
char name[20];
cin.getline(name, 20);
```

The write function display an entire line and has the following form :

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

example : `cout.write(name, 20);`

Formatted Console I/O Operations and Manipulators

C++ supports a number of features that could be used for formatting the output. These features include :

1. **ios** class function and flags.
2. Manipulators.
3. User-Defined output functions.

The ios class contains a large number of member function that would help us to format the output in a number of ways. The most important once among them are listed below.

Function	Task
Width( )	To Specify required field size for displaying an output value.
Precision( )	To specify the number of digits to be displayed after decimal point of a float value.
fill( )	To specify a character that is used to fill the unused portion of a field.
setf( )	To specify format flags that can control the form of output display uch as left or right justification.
unsetf( )	To clear the flags specified.

Manipulators

Manipulators are special functions that can be included in the I/O statements to alter the format parameters of a stream. Following table shows some important manipulators functions that are frequently used. To access manipulators, the file iomanip.h should be included in the program.

Manipulator	Equivalent ios function
setw( )	width ( )
setprecision ( )	precision ( )
setfill ( )	fill ( )
setiosflags ( )	setf ( )
resetiosflags	unsetf ( )

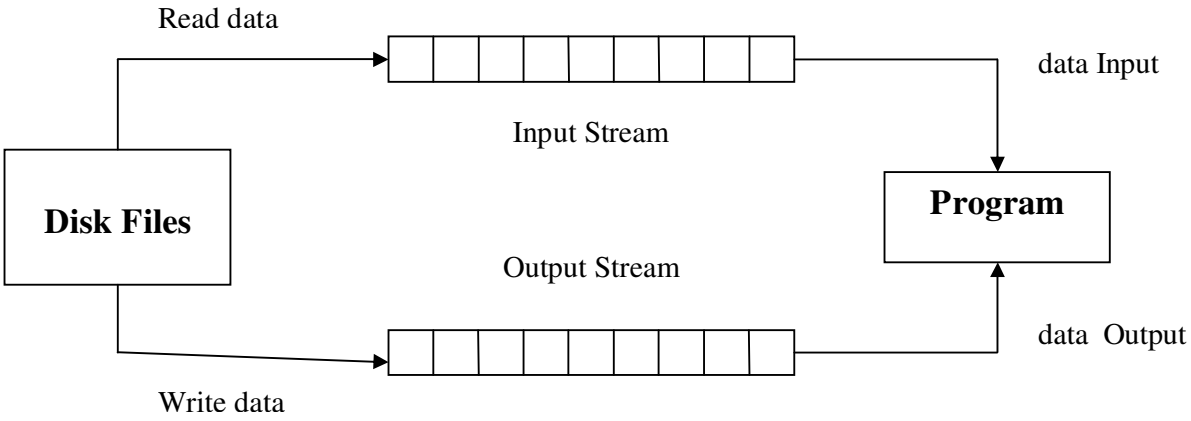
Working with files

Many real-life problems handle large volumes of data and in such situations; we need to use some devices such as floppy disks or hard disk to store the data. The data is stored in these devices using the concept of files. A file is a collection of related data stored in a particular area on the disk. Programs can be designed to perform the read and write operations on these files.

A program typically involves either or both of the following kinds of data communication:
1. Data transfer between the console unit and the program.
2. Data transfer between the program and a disk file.

The I/O system of C++ handles file operations which are very much similar to the console input and output operations. It uses file streams as an interface between the programs and files. The stream that supplies data to the program is known as input stream and the one receives data from the program is known as output stream. In other words, the input stream extracts or reads data from the file and the output stream inserts or writes data to the file. This is illustrated in following figure.

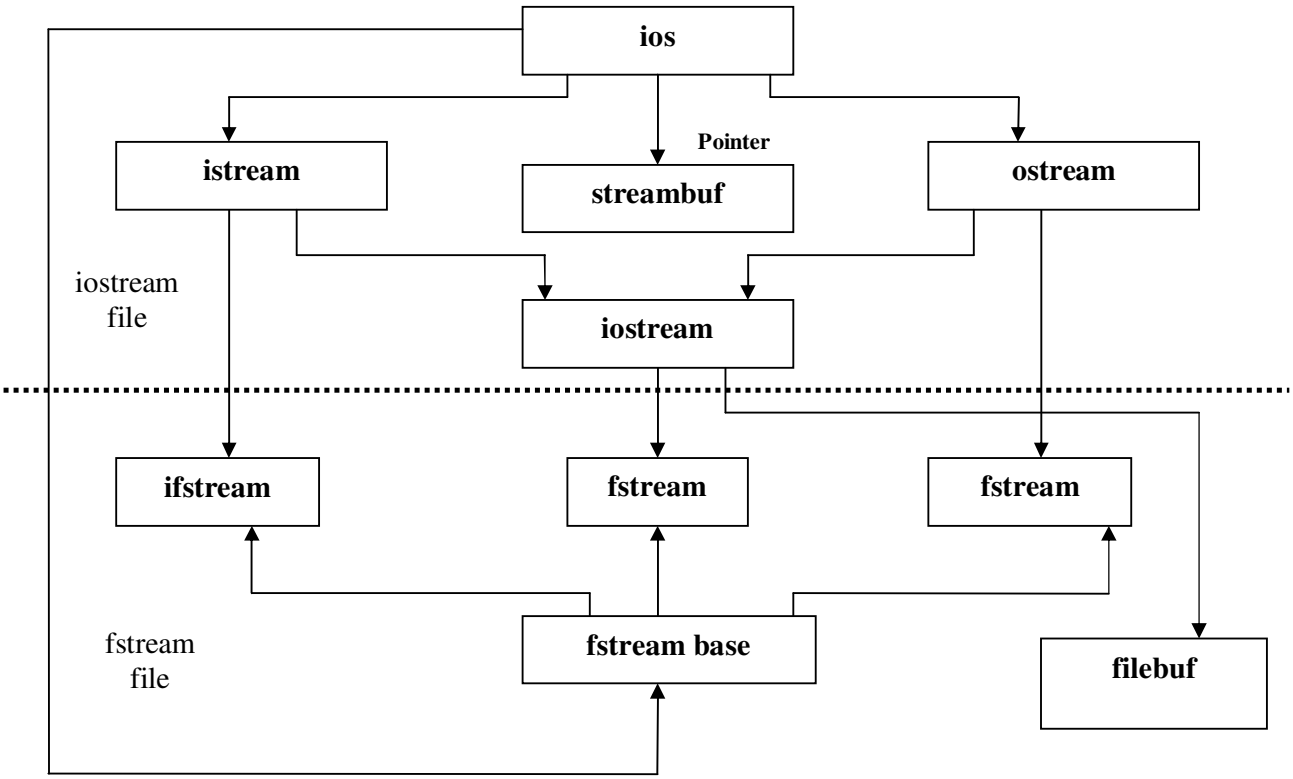
The input operation involves the creation of an input stream and linking it with the program and the input file. Similarly the output operation involves establishing an output stream with the necessary links with the program and the output files.



File Input and Output Stream

File stream classes

The I/O system of C++ contains a set of classes that define the file handling methods. These include ifstream, ofstream and fstream. These classes are derived from fstreambase and from the corresponding iostream class as shown in following hierarchy. These classes are designed to manage the disk files, are declared in fstream and therefore we must include this file in any program that uses files.



Stream classes for file operations contained in fstream file

Following table shows the details of file stream classes. Note that these classes contains many more features.

Class	Contents
Filebuf	Set the file buffer to read and write. Contains Openprot constant used in open( ) of file stream classes. Also contains close( ) and open( ) as members.
Fstreambase	Provides common file stream operations. Serves a base for fstream, ifstream and ofstream class. Contains open( ) and close( ) functions.
Ifstream	Provides input operations. Contains open with default input mode. Inherits the function get( ), getline( ), read( ), seekg( ), and tellg( ) functions from istream.
Ostream	Provides output operations. Contains open( ) with default mode. Inherits put( ) , seekp( ). tellp( ), and write( ) functions from ostream.
Fstream	Provides supports for simultaneous input and output operations. Contauns open( ) with default input mode. Inherits all the function from istream and ostream classes through iostream.

Opening and Closing a file

If we want to use a disk file we, need to decide the following things about the file and its intended use.

1. Suitable name for the file
2. Data type and structure.
3. Purpose.
4. Opening method.

The filename is a string of characters that make up a valid filename for the operating system. It may contain two parts, a primary name and an optional period with extension.

We must first create a file stream and then link it to the filename. A file stream can be defined using the classes ifstream , ofstream and fstream that are contained in the header file fstream. The class to be used depends upon the purpose that is, whether we want to read data from the file or write data to it. A file can be opened in two ways.

1. Using the constructor function of the class.
2. Using the member function open ( ) of the class.

The first method is useful when we use only one file in the stream. The second method is used when we want to manage multiple files using one stream.

Opening Files Using Constructor

We know that a constructor is used to initialize an object while it is being created. Here, a filename is used to initialize the file stream object. This involves the following steps.

1. Create a file stream object to manage the stream using the appropriate class. That is to say the class ofstream is used to create the output stream and the class ifstream to create the input stream.
2. Initialize the file object with the desired filename.

For example following statement opens a file named "results" for output.

```
ostream outfile("results");
```

Similarly the following statement declares infile as an ifstream object and attaches it to the file data for reading or input.

```
ifstream infile("data");
```

Example reading writing using infile and outfile.

```
outfile << "Total";
outfile << sum;

infile >> number;
infile >> string;
```

Every file that has been opened must be closed when no further necessary. To close a file following statement is used.

```
outfile.close( );
infile.close ( );
```

Above statements disconnects the file from the output stream and input stream. Remember, the objects infile and outfile are still exist and files may again be connected to outfile or infile objects.

Opening a file using open ( )

As, the function open( ) can be used to open multiple files that use the same stream object. Foe example, we may want to process a set of file sequentially. In such cases we may create a single stream object and use it to open each file in tern. This is done as follows.

```
File-stream-class stream-object;
Stream-object.open("filename");

ofstream outfile;
outfile.open("DATA1");

… …
… …
outfile.close( );
outfile.open("DATA2");

… …
… …
outfile.close( );
```

Error Handling during file Operations

One of the following thing may happen when dealing with the files:

1. A file which we are attempting to open for reading does not exist.
2. The file name used for a new file may already exist.
3. We may attempt an invalid operation such as reading past the end-of-file.
4. There may not be any space in the disk for storing more data
5. We may use an invalid file name.
6. We may attempt to perform an operation when the file is not opened for that purpose.

The C++ file stream inherits a "stream-state" member form the class ios. This member records information on the status of a file in being used. The stream state member uses bit field to store the status of the error conditions stated above.

The class ios supports several member functions that can be used to read the status recorded in a file stream. These functions along with their meanings are listed in following table.

Function	Return value and meaning
eof( )	Returns true if end-of-file is encountered while reading. Otherwise returns false.
fail( )	Returns true when an input or output operation is failed.
Bad( )	Returns true if an invalid operation is attempted or any unrecoverable error has occurred. However if it is false, it may be possible to recover from any other error reported and continue operation.
Good( )	Returns true if no error has occurred. This means all the above functions are false. For example if file.good( ) is true, all is well with the stream file and we can proceed to perform I/O operations. When it returns false, no further operations can be carried out.

File Modes

The open function can take two arguments one for a file name and second foe a file mode. The general form of the function open( ) with two argument is :

Stream-object.open ("filename", mode);

The second argument mode called file mode parameter specifies the purpose for which the file is opened. The second parameter uses the default values in the absence of the actual values. The default values are as follows.

ios :: in // for ifstream functions meaning open for reading only.
ios :: out // for ofstream functions meaning open for writing only.
The file mode parameter can take one or more of such constants defined in the class ios. Following table lists the file mode parameters and their meanings.

Parameter	Meaning
ios : app	Append to end-of-file
ios :: ate	Go to the end-of-file.
ios :: binary	Binary file
ios :: in	Open file for reading only
ios:: nocreate	Open fails if the file does not exist
ios :: noreplace	Open fails if the file already exists.
ios :: out	Open file for writing only
ios :: trunk	Delete the contents of the file if it exists.

File Pointers

Each file has two associated pointers known as the file pointers. One of them is called the input pointer or get pointer and the other is called the output pointer or put pointer. We can use these pointers to move through the file while reading or writing. The input pointer is used for reading the contents of a given file location and the output pointer is used for writing to a given file location. Each time an input or output operation takes place, the appropriate pointer is automatically advanced.

Function for manipulating of File Pointers

seekg( ) Moves get pointer (input) to a specified location.
seekp( ) Moves put pointer (output) to a specified location.
tellg( ) Gives the current position of the get pointer.
tellp( ) Gives the current position of the put pointer.

seekg (offset , refposition);
seekp(offset, refposition);

The parameter offset represents the number of bytes the file pointer is to be moved from the location specified by the parameter refposition. The refposition takes one of the following three constants defined in the ios class:

ios :: beg start of the file
ios :: cur current position of the pointer
ios :: end of file End of file

Sequential Input and Output Operations

Reading and writing sequentially in files, C++ offers several member functions in a file stream classes.
The functions put( ) and get( ) are designed for handling a single character at a time. Similarly write ( ) and read( ) handle the data in binary form. This means that the values are stored in the disk file in the same format in which they are stored in memory. It is possible to write and read entire object of class using these functions. Only one important point is to remember is that only data are written to the disk file and the meber functions are not.

Updating a file : Random Access

Updating is a routine task in the maintenance of any data file. The updating would include one or more of the following tasks.

1. Displaying the contents of a file.
2. Modifying an existing item.
3. Adding a new item.
4. Deleting an existing item.

These actions require the file pointers to move to a particular location that corresponds to the item/object under consideration. This can be easily implemented if the file contains a collection of items/objects of equal lengths. In such cases, the size of each object can be obtained using the sizeof operator. The file size can be obtained using the function tellg( ) and tellp( ) when the file pointer is located at the end of the file. By using seekg( ) seekp ( ) file pointer can be moved randomly to read or write at specific location.