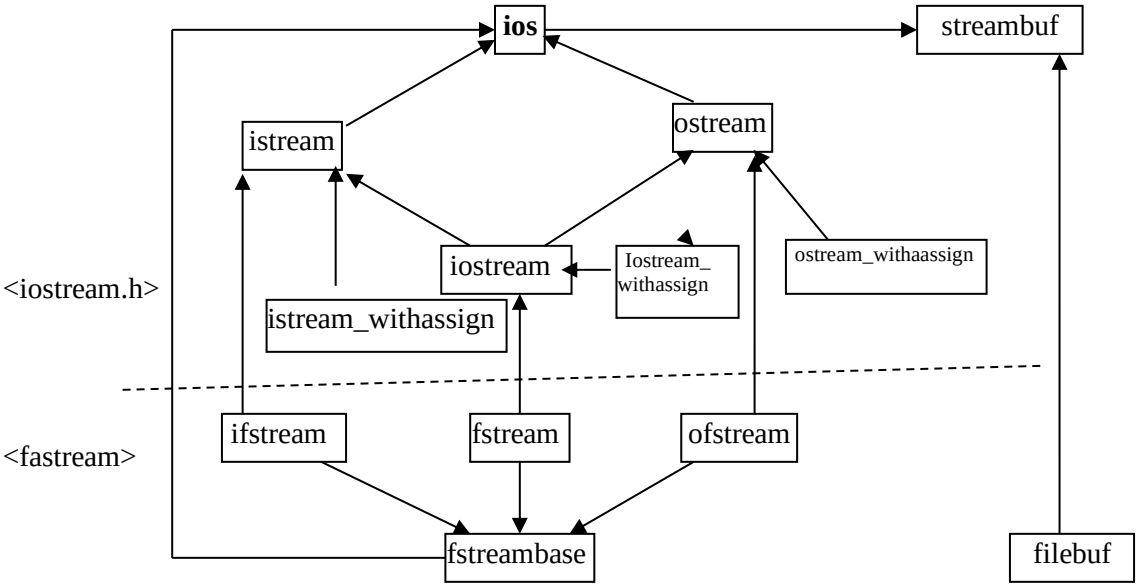
Stream Classes(Class hierarchy)

A stream is a name given to flow of data. In C++ stream is represented by an object of a particular class e.g. *cin* and *cout* are input and output stream objects.

There are no any formatting characters in stream like %d, %c etc in C which removes major source of errors.Due to overloading operators and functions, we can make them work with our own classes.

The Stream class hierarchy:



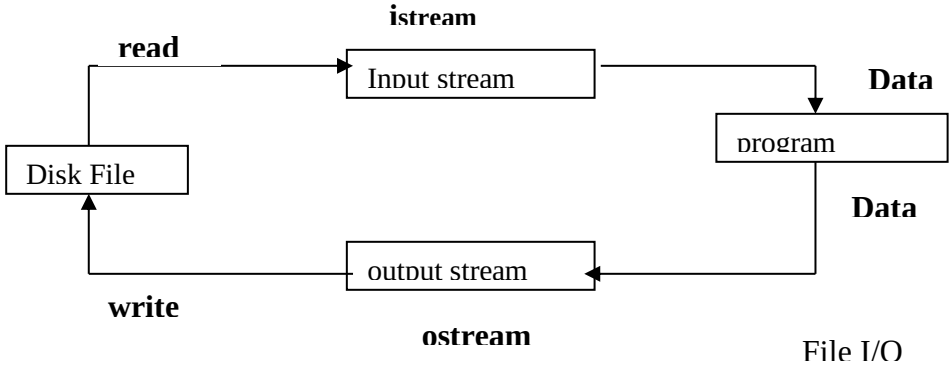
C++ Class Hierarchy

filebuf: The class filebuf sets the file buffer to read and write.

ios: ios class is parent of all stream classes and contains the majority of C++ stream features.

istream class: Derived from ios and perform input specific activities.

ostream class: derived from ios class and perform output specific activities.



File I/O

iostream class: Derived from both istream and ostream classes, it can perform both input and output activities and used to derive iostream_withassign class.

_withassign classes: There are three _withassign classes.

- istream_withassign
- ostream_withassign
- iostream_withasssign

These classes are much like those of their parent but include the overloaded assignment operators.

streambuf: sets stream buffer i.e. an area in memory to hold the objects actual data. Each object of a class associated with the streambuf object so if we copy the stream object it cause the confusion that we are also copying streambuf object. So _withassign classes can be used if we have to copy otherwise not.

fstreambase: Provides operations common to file streams. Serves as a base for fstream, ifstream and ofstream and contains open() and close() functions.

ifstream: Contains input operations in file. Contains open() with default input mode, inherits get(), getline() read(), seekg(), tellg() from istream.

ofstream: Provides output operation in file. Contains open() with default output mode, inherits put(). Seekp(), tellp() and write() from ostream.

fstream: Provides support for simultaneous input and output operations. Contains open() with default input mode: Inherits all the functions of istream and ostream through iostream.

File I/O with stream classes:

In C++,file handling is done by using C++ streams. The classes in c++ for file I/O are ***ifstream*** for input files,***ofstream*** for output files, and ***fstream*** for file used for both input and output operation. These classes are derived classes from ***istream,ostream***,and ***iostream*** respectively and also from fstreambase.

-The header file for *ifstream,ofstream and fstream* classes is <fstream>
-To create and write disk file we use ofstream class and create object of it.
e.g. ofstream outf;

The creation and opening file for write operation is done either using its constructor or using ***open()*** member function which had already been defined in ofstream class.

Creating and opening file for write operation is as:

```
ofstream outf("myfile.txt"); //using constructor of ofstream class.
```

Or

```
ofstream outf;
outf.open("myfile.txt");  // using open() member function.
```

Writing text into file:

We use the object of *ofstream* to write text to file created as:

```
outf<<"This is the demonstration of file operation\n";
outf<<"You can write your text\n";
outf<<"The text are written to the disk files\n";
```

An example for writing to disk file.

```
#include<fstream>
int main()
     {          //constructor creates file and ready to write
          ofstream outf("myfile.txt");

        /* Alternate for above line is
         ofstream outf;     //using open() member function
         outf.open("myfile.txt");
       */

          outf<<"File demonstration program\n";
          //writes strings to file myfile.txt
          outf<<"These strings are written to disk\n";
    }
```

Writing data to file:

```
int x = 20; float f = 2.5;
char ch = 'c'; char* str = " string";
```

Writing to file is done as

```
Outf<<x<<" " <<f<<' '<<ch<<' '<<str;
```

Example

```
#include<iostream>
#include<fstream>

int main()
{
     char ch='c';
     int i= 70;
     float f = 6.5;
     char *str= "Patan";
     ofstream fout("Test.data");
     fout<<ch<<' '<<' '<<f<<' '<<str;
     cout<<"Data written to file\n";
}
```

Reading data from file:

-To read data from file , we use an object of ifstream class File is opened for reading using constructor of ifstream class or open() member function as;

```
ifstream fin("test.txt");           //constructor
or
ifstream fin;
fin.open("test.txt");         // member function open();
Reading data is done as:
Fin>>ch>>i>>f>>str; which is similar as reading data from keyboard  by
cin object.
```

String with embedded blanks:

-Require delemeter line \n for each string with embedded blank and read/write operation is easy.

Reading text from file:

To read text from file we use ***ifstream*** class and file is opened for read operation using constructor or open() member function.

```
e.g.  :  ifstream  infile("myfile.txt");  //using constructor
    or
          ifstream infile;
          infile.open("myfile.txt");

  //  Reading from file myfile.txt:
 While(infile) // or while(!infile.eof())   until end of file
 {
   infile.getline(buffer,maxlength); //buffer to be defined as char
                                     //string of length maxlength
   cout<<buffer;            // for display to screen
 }
```

A sample program to read from myfile.txt

```
#include<fstream>
#include<iostream>

int main()
{
    const int LEN = 100;
    char  text[LEN];      //for buffer
    ifstream infile("myfile.txt");
    while(infile)  //until end of file Alternate is
                   //while(!infile.eof())
    {
       infile.getline(text,LEN);    // read a line of text
       cout<<endl<<text;            //display line of text
```

```
    }
}
```

Character I/O in file[get() and put() function]

put() and get() functions are members of ostream and istream classes so they are inherited to ofstream and ifstream objects. put() is used to write a single character in file and get is used for reading a character from file.
Example:

```
#include<iostream>
#include<fstream>
#include<string>

int main()
{
     char*str="This is a string written to file one char at a time";
     ofstream fout;
     fout.open("myfile.txt");
     for(int i=0;i<strlen(str);i++)]
     {
          fout.put(str[i]);
     }
     cout<<"File write completed";
}
```

// Reading character wise from above file

```
char ch;
ifstream infile;
infile.open("myfile.txt");
while(infile)
{
     infile.get(ch);
     cout<<ch;
}
```

Working with multiple file:

When more than one file is used in a single program for read write operation one file is closed or disconnected from program using close() member function and other file is opened using open().

A sample program:

```
#include<iostream>
#include<fstream>


int main()
{
```

```
ofstream outfile;
//create file district and open for write
outfile.open("district");
outfile<<"Kathmandu\n";
outfile<<"Lalitpur\n";
outfile<<"Kavreplanchowk\n";
outfile<<"Dhading\n";

outfile.close();   // close the file district after writing

outfile.open("headqtr");
outfile<<"Kathmandu\n";
outfile<<"Patan\n";
outfile<<"Dhulikhel\n";
outfile<<"Trisuli\n";
outfile.close();    //closes the file headqtr

//Reading the above files
const int LEN = 80;
char text[LEN];
ifstream infile("district");  //opens file district for read
while(infile)
{
  infile.getline(text,LEN);
  cout<<endl<<text;
}
inflie.close();   //closes file district after display

infile.open("headqtr");
while(infile)
{
   infile.getline(text,LEN);
   cout<<text;
}
infile.close(); //closes file headqtr
}
```

Writing and reading of user input to the file:

We can also write user-input (values of variables in a program input from keyboard) and read those values by using objects of ofstream and ifstream respectively same as done above . Look at these simple program example.

```
#include<fstream>
int main()
{
```

```
    ofstream  fout("test"); //creates and open for writing
    cout<<"Enter the Name:";
    char name[20];
    cin>>name;          //reading from keyboard
    fout<<name<<endl;  //writing to the file "test"
    cout<<"Enter telephohe:";
    int tel;
    cin>>tel;    //reading from keyboard
    fout<<tel;   //writing to file "test"

    fout.close();  //Closes the file "test"
    ifstream fin("test");  //opens the file test for read
    fin>>name;     //reading from file
    fin>>tel;      //reading from file
    cout<<endl<<"The name is: "<<name;
    cout<<endl<<"Telephone no: " <<tel;
    fin.close();
}
```

Opening file in different mode:

In above example we have used the **ofstream** and **ifstream** constructors or **open()** member function using only one argument i.e. filename e.g. "test" etc. However this can be done by using two argument. One is filename and other is filemode.

Syntax:

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

The second argument filemode is the parameter which is used for what purpose the file is opened. If we haven't used any filemode argument and only filename with open() function, the default mode is as:

ios::in for ifstream functions means open for reading only.
i.e. **fin.open("test");** is equivalent to **fin.open("test",ios::in);** as default
ios::out for ofstream functions means open for writing only.
fout.open("test"); is same as **fout.open("test",ios::out);** as default.

Class fstream inherits all features of ifstream and ofstream so we can use fstream object for both input/output operation in file. When fstream class is used , we should mention the second parameter <filemode> with open().
The file mode parameter can take one or more such predefined
constants in **ios** class. The following are such file mode parameters.

Parameters	**Meanings**
ios::app	Append to end of file
ios::ate	Go to end-of-file on opening
ios::binary	Binary file
ios::in	open file for reading only
ios::nocreate	Opens fails if the file does not exists

ios::noreplace	Open files if the file already exists
ios::out	Open file for writing only
ios::trunc	Delete the contents of files if it exits

- Opening file in **ios::out** mode also opens in the **ios::trunc** mode default
- **ios::app** and **ios::ate** takes to the end-of-file when opening but only difference is that **ios::app** allows to add data only end-of-file but **ios::ate** allows us to add or modify data at anywhere in the file. In both case file is created if it does not exists.
- Creating a stream **ofstream** default implies output(write) mode and **ifstream** implies input(read), but **fstream** stream does not provide default parameter so we must provide the mode parameter with **fstream.**
- **The mode can combine two or more parameters using bitwise OR operator (|)**
  e.g. fout.open("test",ios::app|ios::out);

File Pointers:

The file management system associates two types of pointers with each file.

1. get pointer (input pointer)
2. put pointer (output pointer)

These pointers facilitate the movement across the file while reading and writing.

- The get pointer specifies a location from where current read operation initiated.
- The put pointer specifies a location from where current write operation initiated.

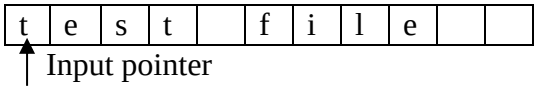
The file pointer is set to a suitable location initially depending upon the mode which it is opened.

• **Read-only Mode**: When a file is opened in read-only mode, the input (get) pointer is initialized to the beginning of the file.

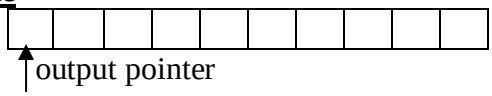
• **Write-only: mode**: In this mode, existing contents are deleted if file exists and put pointer is set to beginning of the file.

• **Append mode**: In this mode, existing contents are unchanged and put pointer is set to the end of file so writing can be done from end of file.

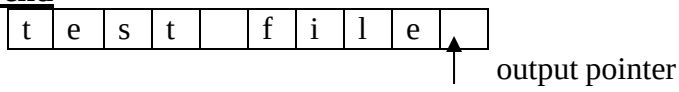
Read



write



Append



Functions manipulating file pointers:

C++ I/O system supports 4 functions for setting a file to any desired position inside the file.

The functions are

Function	**member of class**	**Action**
seekg()	ifstream	moves get file pointer to a specific location
seekp()	ofstream	moves put file pointer to a specific location
tellg()	ifstream	Return the current position of the get ptr
tellp()	ofstream	Return the current position of the put ptr

These all four functions are available in fstream class by inheritance. The two seek() functions have following prototypes.

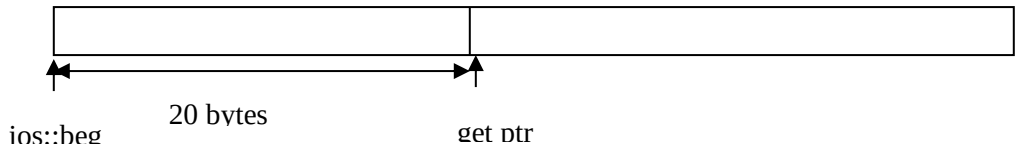
```
istream & seekg (long offset, seek_dir origin =ios::beg);
ostream & seekp (long offset, seek_dir origin=ios::beg);
```

- Both functions set file ptr to a certain offset relative to specified origin. The origin is relative point for offset measurement. The default value for origin is ios::beg.
- (seek_dir) an enumeration declaration given in ios class as

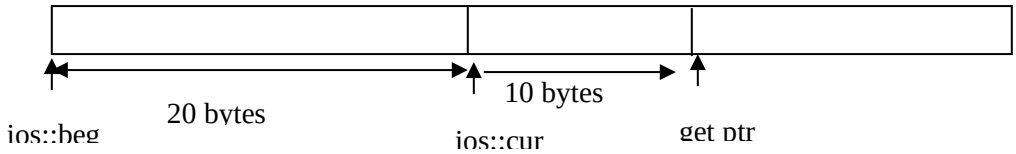
orgin value	**seek from**
ios::beg	seek from beginning of file
ios::cur	seek from current location
ios::end	seek from end of file

e.g. `ifstream infile;`

`infile.seekg(20,ios::beg);` or `infile.seekg(20);` // default ios::beg move file ptr to 20th byte in the file. The reading start from 21st item [byte start from 0] with file.



Then after, `infile.seekg(10,ios::cur);` moves get pointer 10 bytes further from current position.



Similarly:

```
ofstream  outfile;
outfile.seekp(20,ios::beg);    // out file. seek p (20);
```

moves file put pointer to 20th byte and if write operation is initiated, start writing from 21st item.

Consider following example

```
ofsteam outfile("student",ios::app);
int size=outfile.tellp();
```

Return the size of file in byte to variable size since ios::app takes file put ptr at end of file. The function tellp() returns the takes file put ptr at end of file. The function tellp() returns the current position of put ptr.

Equivalently:

```
ifstream infile("student");
infile.seekg(0,ios::end);
int size=infile.tellg() ;
     This returns the current file pointer position which is at end of
file so we gget he size of fife "student".
```

Some of pointer offset calls and their actions:

Assume: ofstream fout;

Seek	**Action**
fout.seekg(0,ios::beg)	Go to beginning of the file
fout.seekg(0,ios::cur)	Stay at current location
fout.seekg(0,ios::end)	Go to the end of file
fout.seekg(n,ios::beg)	move to (n+1) byte from beginning of file.
fout.seekg (n,ios::cur)	move forword by n bytes from currrent position
fout.seekg(-n,ios:: cur)	move backward by n bytes from currnt position
fout.seekp(n,ios:: beg)	move write pointer (n+1) byte location
fout.seekp(-n,ios:: cur)	move write ptr n bytes backwards.

File I/O with fstrem class

Fstream class supports simultaneous input/output operations. It inherits function from istream and ostream class through iostream.

Following program illustrates this

```
#include<iostream>
#include<fstream>          //Assume file student.in
#include<conio>                 //is created with
#include<process>               //1. no of student (count)
int main()                      //2. for n students
{
     fstream infile;     // input file name
     fstream outfile;    // output file percentage sane
     int i, count, percentage;
     char name[20];
               //open for read mode
     infile.open("student.in",ios::in);
     if(infile.fail())   // if operation failed.
     {
          cout<<"Error: student.in open fail";
          exit(1);
     }
          //open next file for write
```

```
     outfile.open("student.out",ios::out);
     if(outfile.fail())
     {    cout<<"Error:......"; exit(1);
     }
     infile>>count; // no of student
     outfile<<"     student Information processing" <<endl;
     outfile<<"     ------------------------------" <<endl;
     for(i=0; i<count; i++)
     {         // Read data percentage from input file
          infile>>name;
          infile>>percentage;
               // write in output file.
          outfile<<"Name:"<<name<<endl;
          outfile<<"precentage:"plercentage<<endl;
          outfile<<"passed in:";
          if(percentage>=75)
               outfile<<"first Division/distinction";
          else if(percentage>=45)
               outfile<<" Second Div";
          else if(percentage>=35)
               outfile<<"Passed";
          else
          outfile<<"Failed";
          outfile<<endl;
          outfile<<"....................."<<endl;
     }
                    // close files;
          infile.close();
          outfile.close();
}
```

The put () and get () function:

- The function get() is a member function of the file stream class fstream, and used to read a single character from file.
- The function put() is member function of fstream class and used to write a single character into file.

Example:

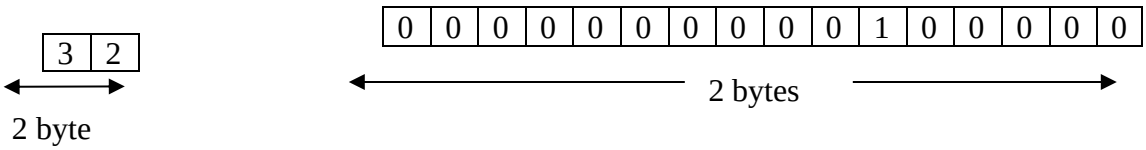
```
#include<fstream>
int main()
{
     char c, string[100];
     fstream file("student.txt",ios::in|ios::out);
     cout<<"Enter string:";
     for (int i=0; string[i]!='\0'; i++)
     file.put(string[i]);
     file.seekg(0); // seek to the beging
```

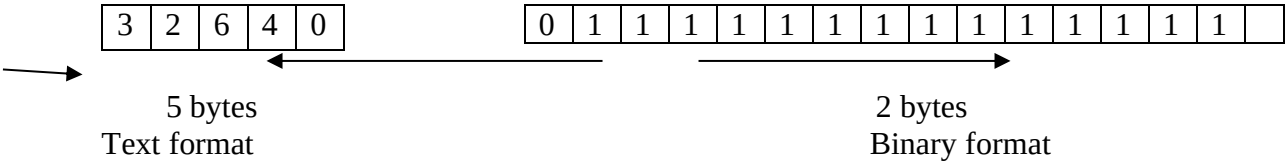
```
      cout<<"output string:";
      while(file)
      {
            file.get(c);
            cout<<c;
      }
}
```

The write () and read () function:

- The write () function is a member of stream class fstream and used to write data in file as binary format.
- The read () function is used to read data (binary form) from a file.
- The data representation in binary format in file is same as in system. The no of byte required to store data in text form is proportional to its magnitude but in binary form, the size is fixed.

e.g.





The prototype for read () & write () functions are as:.

```
infile.read((char*)&variable, sizeof(variable));
outfile.write((char*)&variable, sizeof(variable));
```

- The first parameter is a pointer to a memory location at which the data is to be retrieved [read()] or to be written [write()] function.
- The second parameter indicates the number of bytes to be transferred.

Example :writing variable in to files

```
#include<fstream>
int main()
{
      int number1=530;
      float number2=100·50;
//  open file in read binary mode, read integer and class
      ofstream ofile("number.bin",ios::binary);
```

```
     ofile.write((char*)&number1, sizeof(number1) );
     ofile.write((char*)&number2, sizeof(float) );
     ofile:close();
//  open file in read binary mode, read integer & close
     ifstream ifile ("number.bin",ios::binary);
     ifile.read((char*)&number1,sizeof(number1));
     ifile.read( (char*) &number2,sizeof(number2));
     cout<<number1<<" "<<number2<<endl;
     ifile.close();

}
```

Overloading stream operators (<< and >>)

In C++, stream insertion operator "<<" is used for output and extraction operator ">>" is used for input.

We must know the following things before we start overloading these operators.
1) cout is an object of ostream class and cin is an object of istream class
2) These operators must be overloaded as a global function. And if we want to allow them to access private data members of the class, we must make them friend.

Why these operators must be overloaded as global?
In operator overloading, if an operator is overloaded as a member, then it must be a member of the object on the left side of the operator. For example, consider the statement "ob1 + ob2" (let ob1 and ob2 be objects of two different classes). To make this statement compile, we must overload '+' in a class of 'ob1' or make '+' a global function.

The operators '<<' and '>>' are called like 'cout << ob1' and 'cin >> ob1'. So if we want to make them a member method, then they must be made members of ostream and istream classes, which is not a good option most of the time. Therefore, these operators are overloaded as global functions with two parameters, cout and object of user-defined class.

Following is a complete C++ program to demonstrate overloading of << and >> operators.

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

class Complex
{
private:
    int real, imag;
```

```
public:
    Complex ()
{
real=0;
imag=0;
}
    Complex (int r, int i)
    {  real = r;   imag = i; }
    friend ostream & operator << (ostream &out, const Complex &c);
    friend istream & operator >> (istream &in, Complex &c);
};

ostream & operator << (ostream &out, const Complex &c)
{
    out << c.real;
    out << "+i" << c.imag << endl;
    return out;
}

istream & operator >> (istream &in, Complex &c)
{
    cout << "Enter Real Part ";
    in >> c.real;
    cout << "Enter Imaginary Part ";
    in >> c.imag;
    return in;
}

int main()
{
   Complex c1;
   cin >> c1;
   cout << "The complex object is ";
   cout << c1;
   return 0;
}
```

Output:

```
Enter Real Part 10
Enter Imaginary Part 20
The complex object is 10+i20
```

Object I/O in file

C++ is Object-oriented language so we need objects to be written in file and read from file . Following examples show the I/O operations.

Writing object to disk file:

Generally binary mode is used which writes object in disk in bit configurations.

```
//example
#include<fstream>
#include<iostream>
class emp
{
     char empname[20];
     int eno;
     float sal;
     public:
     int getdata()
     {
          cout<<"Enter Name:"; cin>>empname;
          cout<<"Enter Emp No:"; cin>>eno;
          cout<<"Enter salary:"; cin>>sal;
     }

};

int main()
{
emp em;
cout<<"Enter the detail of employee"<<endl;
em.getdata();
ofstream fout("emp.dat");
fout.write((char*)&em,sizeof(em));
cout<<"Object written to file";
}
```

READING FROM FILE:

```
#include<fstream>
#include<iostream>
class emp
{
     char empname[20];
     int eno;
     float sal;
     public:

     int showdata()
```

```
     {
          cout<<"\nName:"<<empname<<endl;
          cout<<"Emp NO:"<<eno<<endl;
          cout<<"Salary:"<<sal<<endl;
     }
};

int main()
{
emp em;
ifstream fin("emp.dat");
fin.read((char*)&em,sizeof(em));
cout<<"Object detail from file";
em.showdata();
}
```

Writing and reading objects:

```
//student.cpp
#include<iostream>
#include<fstream>
#include<iomanip>
class student
{
     char name[20];
     int roll;
     char add[20];
     public:
     int readdata()
     {
          cout<<"Enter name:";cin>>name;
          cout<<"Enter Roll. no.:";cin>>roll;
          cout<<"Enter address:";cin>>add;
     }
     int showdata()
     {
          cout<<setw(10)<<roll<<setiosflags(ios::left)<<setw(10)
          <<name<<setiosflags(ios::left)<<setw(10)<<add<<endl;
     }
};

int main()
{

     student s[5];
     fstream file;
     file.open("record.dat", ios::in|ios::out);
```

```
     cout<<"enter detail for 5 students:";
     for(int i=0;i<5;i++)
     {
          s[i].readdata();
          file.write((char*)&s[i],sizeof(s[i]));
     }
     file.seekg(0);//move pointer begining.
     cout<<"Output from file"<<endl;
     cout<<setiosflags(ios::left)<<setw(10)<<"RollNo"
     <<setiosflags(ios::left)<<setw(10)<<"Name"
     <<setiosflags(ios::left)<<setw(10)<<"Address"<<endl;
     for(i=0;i<5;i++)
     {
          file.read((char*)&s[i],sizeof(s[i]));
          s[i].showdata();
     }
     file.close();
}
```

Command Line Arguments

C++ supports the features that facilitates the supply of arguments to the main() function. The arguments are supplied to the main at the time of program execution from command line. The main function takes two arguments. First of which is argument count argc and second is an array of arguments name argv[] as

main(int argc, char*argv[])

such program is invoked in command prompt as
C> programname arg1 arg2….

Following example shows the use of command line arguments which reads two different files and display the contents one containing even numbers and another containing odd numbers.

//Program commandline arguments

//evenodd.cpp

```
#include<iostream>
#include<fstream>
#include<stdlib>
#include<conio>
int main(int argc,char*argv[])
{
     int number[9]={11,22,33,44,55,66,77,88,99};
     if(argc!=3)
     {
          cout<<"argc="<<argc<<endl;
```

```
        cout<<"Error in arguments"<<endl;
        getch();
        exit(1);
    }
    ofstream fout1, fout2;
    fout1.open(argv[1]);
    if(fout1.fail())
    {
        cout<<"couldnot open the file"<<argv[1]<<endl;
        getch();
        exit(1);
    }
    fout2.open(argv[2]);
    if(fout1.fail())
    {
        cout<<"couldnot open the file"<<argv[2]<<endl;
        getch();
        exit(1);
    }

    for(int i=0;i<9;i++)
    {
        if(number[i]%2==0)
        fout2<<number[i]<<" ";
        else
        fout1<<number[i]<<" ";
    }
    fout1.close();
    fout2.close();
    ifstream fin;
    char ch;
    for(i=1;i<argc;i++)
    {
        fin.open(argv[i]);
        cout<<"Contents of "<<argv[i]<<endl;
        while(fin)
        {
            fin.get(ch);
            cout<<ch;
        }
        cout<<endl<<endl;
        fin.close();
    }
    return 0;
}
```

This program can be invoked in command line as:

C:\evenodd EVEN ODD

The output of program will be;

Contents of EVEN
11 33 55 77 99

Contents of ODD
22 44 66 88

Example 2:

A program that copies contents of a text file to another file

```
#include<iostream>
#include<fstream>
#include<stdlib>
#include<conio>
int main(int arg,char*argv[])
{
     char *str="This is a test file written and saved into disk for
     copy";
     if(arg!=3)
     {
          cout<<"argc="<<arg<<endl;
          cout<<"Error in arguments"<<endl;
          getch();
          exit(1);
     }
     ofstream fout;
     fout.open("test");
     fout<<str<<endl;
     fout<<"The contents of string is written into the file"<<endl;
     fout.close();
     fout.open(argv[2]);
     if(fout.fail())
     {
          cout<<"couldnot open the6 file"<<argv[2]<<endl;
          getch();
          exit(1);
     }
     ifstream fin;
     fin.open(argv[1]);
     if(fin.fail())
     {
```

```
          cout<<"couldnot open the file"<<argv[1]<<endl;
          getch();
          exit(1);
     }
     while(fin)
     {  char ch;
          fin.get(ch);
          fout<<ch;
     }
     fin.close();
     fout.close();
     cout<<"File "<<argv[1]<<" is "<<"Copied to "<<argv[2]<<endl;
     return 0;
}
```

Execute it as:

C:\filecpy test TEST1

Output will be

File test is Copied to TEST1;