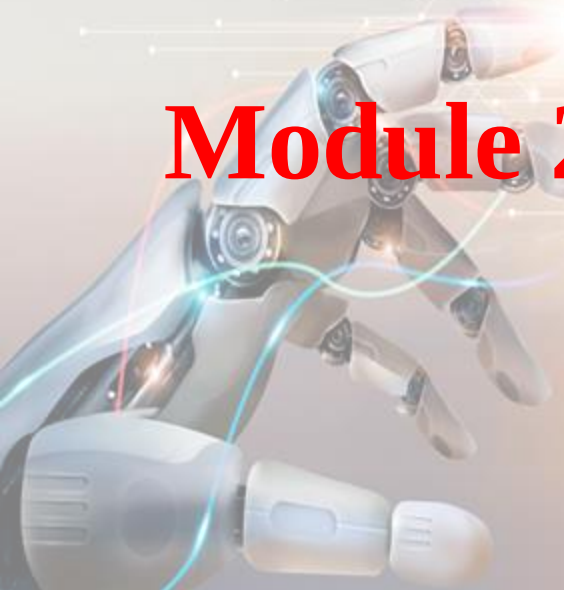
BENNETT UNIVERSITY

TIMES OF INDIA GROUP

CSET334 - Programming using C++

Module 2: FILE HANDLING



File Handling In C++

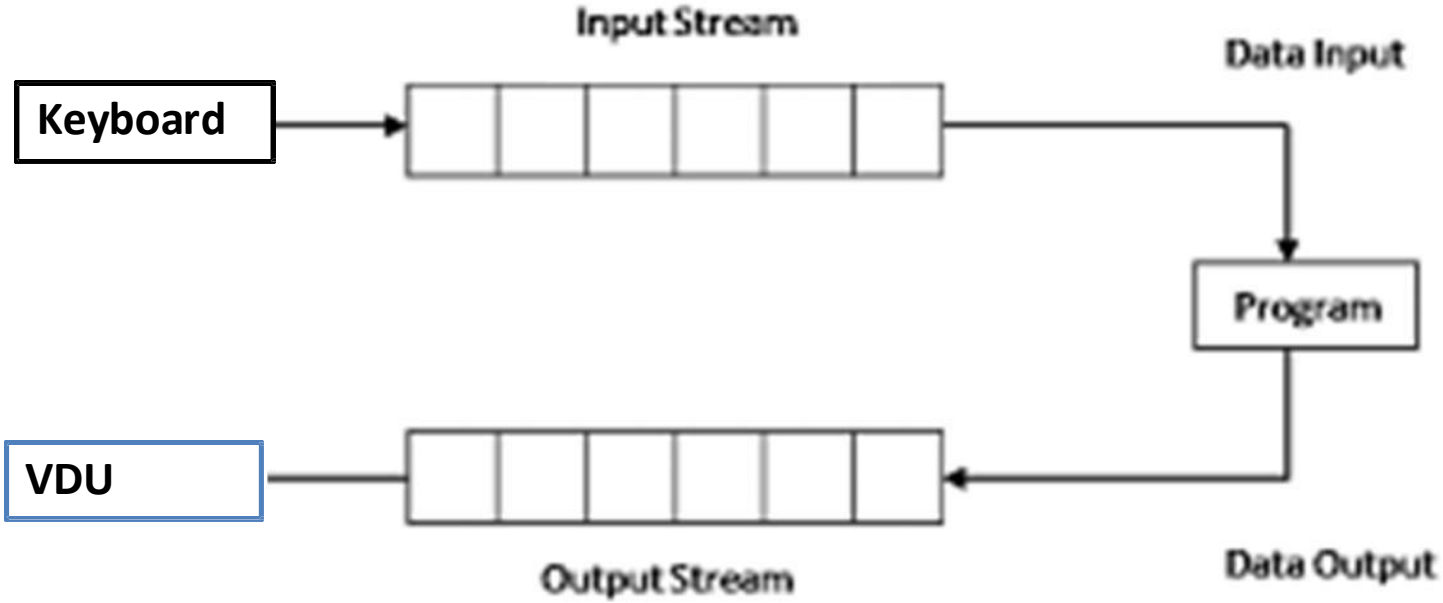
In Standard computer programming , we take input through the keyboard and output is displayed on screen.

For that we make use of iostream.h header file and uses two standard objects.

cin of istream

cout of ostream

In programming we use cin and cout objects of istream.h and ostream.h (sub classes of iostream.h) for input and output.



Here the input goes to temporary memory, as program is closed data is no more available on disk.

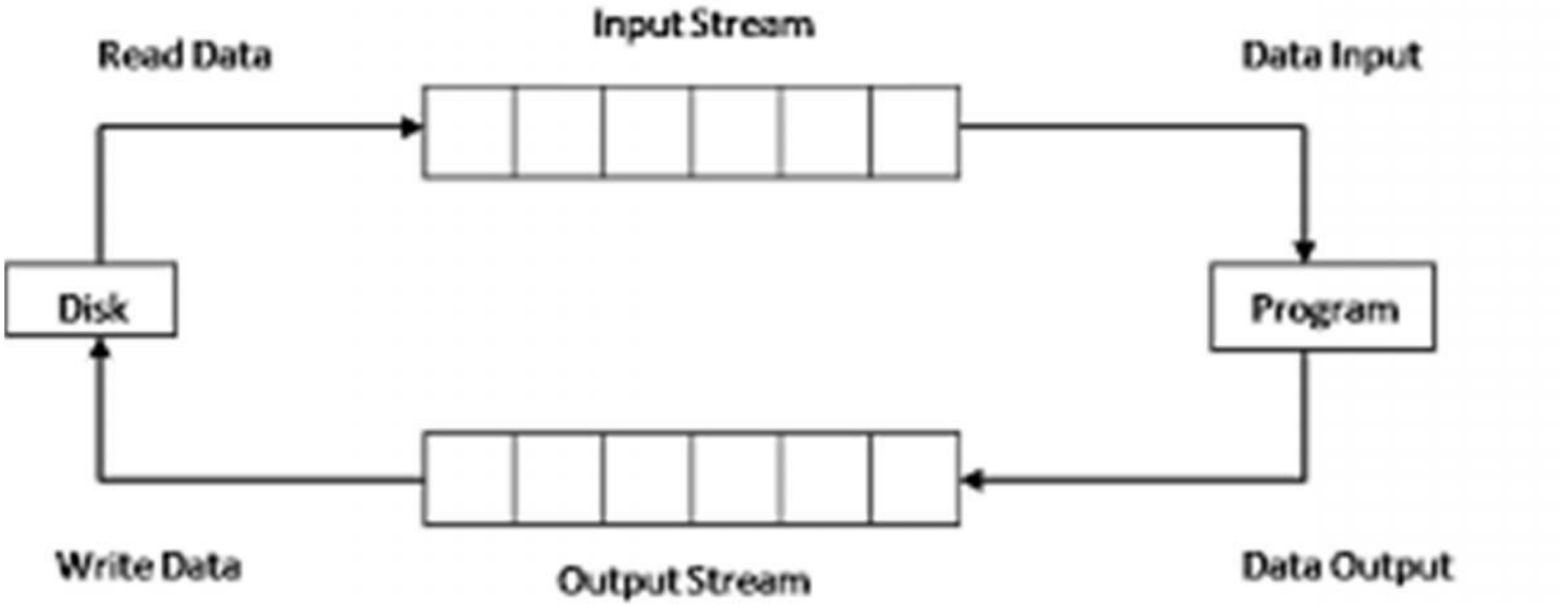
File Handling In C++

Computer programs work with files
-as it helps in storing data & information permanently.

File: is a bunch of bytes stored under a specific name on a storage device.
Stream: It refers to a sequence of bytes.

Every file is linked to a **stream.**
& each stream is associated with particular **class**.

File Handling In C++



Kinds of files: a file can be two types.

Text file:-

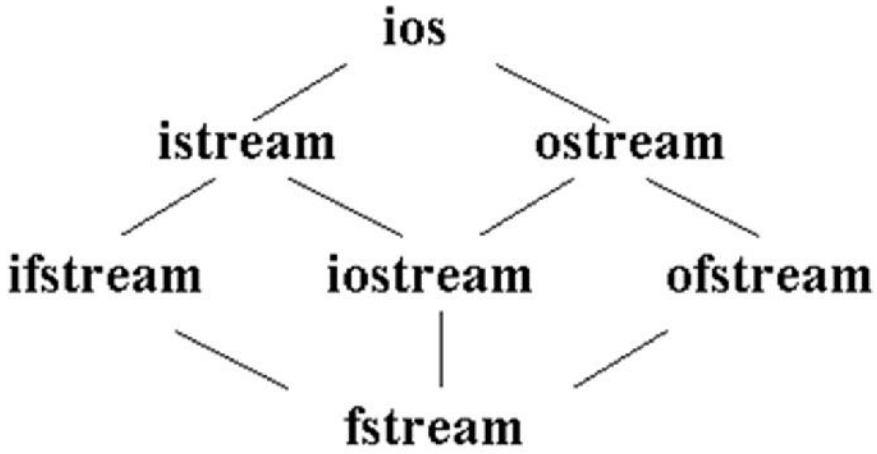
1. It is a file that stores information in ASCII characters.
2. In text files, each line of text is terminated with a special character known as EOL (End of Line) character or delimiter character.
3. When this EOL character is read or written, certain internal translations take place.

Binary file.

1. It is a file that contains information in the same format as it is held in memory.
2. In binary files, no delimiters are used for a line and no translations occur here.
3. Binary files are faster and easier for programs to read & write.

By default files in c++ are considered as text files.

Classes for file stream operation



ofstream : Stream class to write on files.

ifstream : Stream class to read from files

fstream: Stream class to both read and write from/to files.

Class	Functions
ifstream	Being an input file stream class, it provides the input operations for file. It inherits the functions get(), getline(), read() and functions supporting random access (seekg() and tellg()) from istream class defined inside the header file iostream.h
ofstream	Being an output file stream class, it provides the output operations. It inherits the functions put() and write() along with functions supporting random access (seekp() and tellp()) from ostream class defined inside the header file iostream.h
fstream	It is an input-output file stream class. It provides support for the simultaneous input and output operations. It inherits all the functions from istream and ostream classes through the iostream class defined inside the header file iostream.h

In C++, you open a file, you must first obtain a stream. There are t… following three types of streams:

1. input
2. output
3. input/output

Create an Input Stream

To create an input stream, you must declare the stream to be of class **ifstream.**
Here is the syntax:

```
ifstream fin;
```

Create an Output Stream

To create an output stream, you must declare it as class **ofstream.**
Here is an example:

```
ofstream fout;
```

Create both Input/Output Streams

Streams that will be performing both input and output operations must be declared as **class fstream.** Here is an example:

```
fstream fio;
```

Opening File Using Constructors

We know that a constructor of class initializes an object of its class when it (the object) is being created.

Same way, **the constructors of stream classes** (ifstream, ofstream, or fstream) are used to initialize file stream objects with the filenames passed to them.

First create a file stream object of input type i.e., ifstream type. For example:

ifstream fin("myfile", ios::in) ;

The above given statement creates an object, fin, of input file stream.

After creating the ifstream object fin, the file myfile is opened and attached to the input stream, fin.

Now, the data being read from myfile has been channelised through the input stream object.

To read from this file, this stream object will be used using the getfrom operator (">>").

```
char ch;
fin >> ch ; // read a character from the file
float amt ;
fin >> amt ; // read a floating-point number form the file
```

Similarly, when you want a program to write a file
i.e., to open an output file (on which no operation can take place except writing only).

This will be accomplish by
1. creating ofstream object to manage the output stream
2. associating that object with a particular file

example,
ofstream fout("secret" ios::out) ; // create ofstream object named as fout

To write something to it, you can use << (put to operator) in familiar way.
Here is an example,

```
int code = 2193 ;
fout << code << "xyz" ;
 /* will write value of code and "xyz" to fout's associated file namely "secret" here. */
```

Opening a file

1. Opening file using constructor

```
ofstream outFile("sample.txt");
ifstream inFile(“sample.txt”);
```

2. Opening File Using open ()

StreamObject.open(“filename”, [mode]);

```
ofstream oFile;
ofile.open(“temp”,ios::out);
  //is same as
oFile.open(“temp");// by default
ifstream inFile;

inFile.open("sample.txt“,ios::in);
//is same as
inFile.open("sample.txt”);// by default
```

The Concept of File Modes

The filemode describes how a file is to be used : to read from it, to write to it, to append it, and so on.

When you associate a stream with a file, either by **initializing a file stream object** with a file name or by **using the open()** method,
you can provide a **second argument specifying the file mode**, as mentioned below :

```
stream_object.open("filename", (filemode) ) ;
```

The second method argument of open(), the filemode, is of type int, and you can choose one from several constants defined in the ios class.

Constant	Meaning	Stream Type
ios :: in	It opens file for reading, i.e., in input mode.	ifstream
ios :: out	It opens file for writing, i.e., in output mode. This also opens the file in ios :: trunc mode, by default. This means an existing file is truncated when opened, i.e., its previous contents are discarded.	ofstream
ios :: ate	This seeks to end-of-file upon opening of the file. I/O operations can still occur anywhere within the file.	ofstream ifstream
ios :: app	This causes all output to that file to be appended to the end. This value can be used only with files capable of output.	ofstream
ios :: trunc	This value causes the contents of a pre-existing file by the same name to be destroyed and truncates the file to zero length.	ofstream
ios :: nocreate	This cause the open() function to fail if the file does not already exist. It will not create a new file with that name.	ofstream
ios :: noreplace	This causes the open() function to fail if the file already exists. This is used when you want to create a new file and at the same time.	ofstream
ios :: binary	This causes a file to be opened in binary mode. By default, files are opened in text mode. When a file is opened in text mode, various character translations may take place, such as the conversion of carriage-return into newlines. However, no such character translations occur in file opened in binary mode.	ofstream ifstream

We can combine two or more filemode constants using the C++ bitwise OR operator (symbol |).
For example, the following statement :

```
ofstream fout;
fout.open("Master", ios :: app | ios :: nocreate);
```

will open a file in the append mode if the file exists and will abandon the file opening operation if the file does not exist.

Closing File

```
outFile.close();
inFile.close();
```

```
/* C++ File Streams */

#include<conio.h>
#include<fstream.h>
#include<stdlib.h>
void main()
{
char inform[80];
char  fname[20];
char ch;
clrscr();
cout<<"Enter file name: ";
cin.get(fname, 20);
ofstream fout(fname, ios::out);

            ;
}
```

```
            if(!fout)
            {
                        cout<<"Error in creating the
file..!!\n";
                        cout<<"Press any key to exit...\n";
                        getch();
                        exit(1);
            }
            cin.get(ch);

            cout<<"Enter a line to store in the file:\n";
            cin.get(inform, 80);
            fout<<inform<<"\n";

cout<<"\nEntered informations successfully stored.!\n";
            fout.close();
            cout<<"Press any key to exit...\n";
            getch();
}
```

Get rollnumbers and marks of the students of a class (get from the user) and store these details into a file called marks.dat :

```
#include<conio.h>
#include<fstream.h>
void main()
{           ofstream fout;        // stream decided and declared
            fout.open("marks.dat", ios::out);       // file linked
            char ans = 'y' ;        // process as required –

            int rollno;
            float marks ;
            while(ans == 'y' || ans == 'Y')
            {           cout << "\nEnter Rollno. : " ;
                        cin >> rollno ;
                        cout << "\nEnter Marks : " ;
                        cin >> marks ;
                        fout << rollno << '\n' << marks << '\n' ;
                        cout << "\nWant to enter more ? (y/n) " ;
                        cin >> ans ;
            }
            fout.close() ;       // de-link the file
            getch();
}
```

The default behaviour of an ofstream type stream (ios::out file mode), upon opening a file, is that it create the file if the file doesn't yet exist and it truncates the file (i.e., deletes its previous contents) if the file already exists

```
fout.open(fname, ios::app);
```

Text File functions:

Char I/O :

1. **get() –** read a single character from text file and store in a buffer.

e.g **file.get(ch);**

2. **put()** - writing a single character in textfile e.g. **file.put(ch);**
3. **getline() -** read a line of text from text file store in a buffer.

e.g **file.getline(s,80);**

We can also use **file>>ch** for reading and **file<<ch** writing in text file. But >> operator
does not accept white spaces.

Binary file functions:

- **read()-** read a block of binary data or reads a fixed number of bytes from the specified

stream and store in a buffer.

Syntax : Stream_object.read((char *)& Object, sizeof(Object));

e.g file.read((char *)&s, sizeof(s));

- **write()** – write a block of binary data or writes fixed number of bytes from a specific

memory location to the specified stream.

Syntax : **Stream_object.write((char *)& Object, sizeof(Object));**

e.g file.write((char *)&s, sizeof(s));

Both functions take two arguments.

• The first is the **address of variable**, and the second is the **length of that variable in bytes.**

The address of variable must be type cast to type char*(pointer to character type)

File Pointer: The file pointer indicates the position in the file at which the next input/output is to occur.

Moving the file pointer in a file for various operations viz modification, deletion , searching etc. Following functions are used:

1. seekg(): It places the file pointer to the specified position in input mode of file.

e.g file.seekg(p,ios::beg); or file.seekg(-p,ios::end), or file.seekg(p,ios::cur)

i.e to move to **p** byte position from beginning, end or current position.

2. seekp(): It places the file pointer to the specified position in output mode of file.

e.g file.seekp(p,ios::beg); or file.seekp(-p,ios::end), or file.seekp(p,ios::cur)

i.e to move to **p** byte position from beginning, end or current position.

3. tellg(): This function returns the current working position of the file pointer in the input mode.

e.g int p=file.tellg();

4. tellp(): This function returns the current working position of the file pointer in the output mode.

e.f int p=file.tellp();

eof()

We can detect when the end of the file is reached by using the member function eof()

It returns non-zero when the end of file has been reached, otherwise it returns zero.

For example, consider the following code fragment :

```
ifstream fin ;
fin.open("master", ios::in | ios::binary);
while(!fin.eof())                    //as long as eof() is zero
{                                      //that is, the file's end is not reached
        :                              //process the file
}
if(fin.eof())                        //if non-zero
cout << "End of file reached ! \n" ;
```

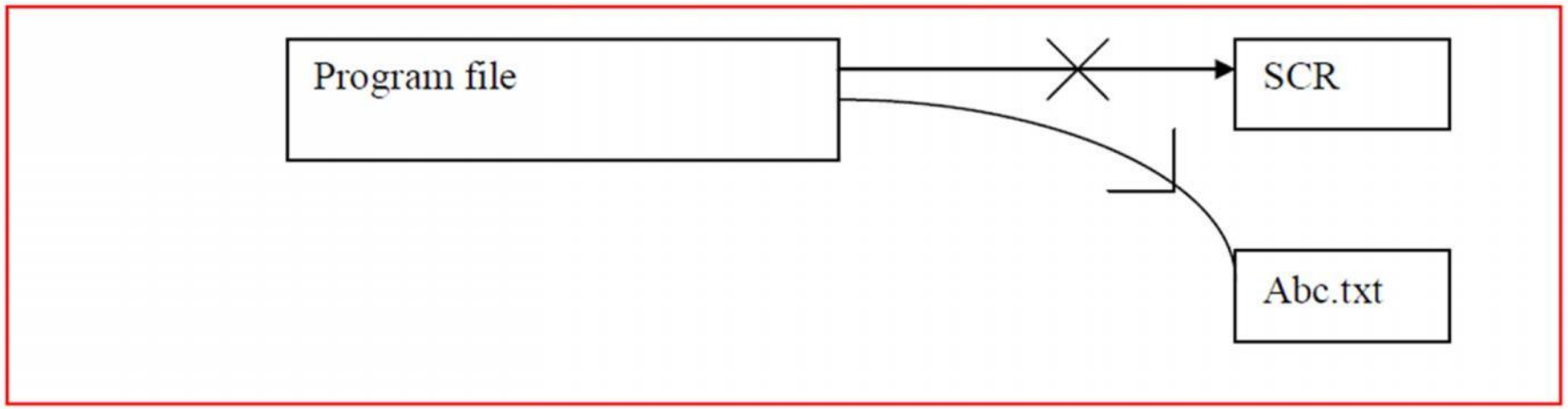
To detect end of file, without using EOF(), you may check whether the stream object has become NULL or not.

For example,

```
ifstream fin;
fin.open("master", ios::in | ios::binary);
while(fin)              //as long as  the file's end is not reached
        :               //process the file
}
```

The std namespace is special, The built in C++ library routines are kept in the standard namespace. That includes stuff like cout, cin, string, vector, map, etc. Because these tools are used so commonly, it's popular to add "using namespace std" at the top of your source code so that you won't have to type the std:: prefix constantly. It only make our task easy, It is not not necessary.

Q1 Get three integer values from user and store them into a file Integer.txt and also display the contents of file Interger.txt



```
#include<fstream.h>
void main()
{
ofstream ofile; // decide and declare stream object
ofile.open("Integer.txt",ios::out); // link file
int x,y,z;
// processing
cout<<"Enter any three integer\n";
cin>>x>>y>>z;
ofile<<x<<" "<<y<<" "<<z;
ofile.close();

// read and display
ifstream ifile;
ifile.open("Integer.txt",ios::in);
cout<<"the three integer are:\n";
ifile>>x>>y>>z;//read from file
cout<<"the values are\n";
cout<<x<<endl<<y<<endl<<z;
ifile.close();
}
```

```
#include<fstream.h>
int main()
{
ofstream fout;
fout.open("abc.txt");
fout<<"This is my first program in file handling";
fout.close();
return 0;
}
```

To create a text file using strings I/O

```
#include<fstream.h> //header file for file operations
#include<stdio.h>
void main()
{
char s[80], ch;
ofstream file("myfile.txt"); //open myfile.txt in default output mode
do
{ cout<<"\n enter line of text";
gets(s); //standard input
file<<s; // write in a file myfile.txt
cout<<"\n more input y/n  ";
cin>>ch;
}while(ch=='y'||ch=='Y');
file.close();
} //end of main
```

To create a text file using characters I/O

```
#include<fstream.h> //header file for file operations
#include<stdio.h>
void main()
{
char ch;
ofstream file("myfile2.txt"); //open myfile.txt in default output mode
ch=getchar();
file<<ch; // write a character in text file ‘myfile.txt ‘
file.close();
} //end of main
```

Text files in input mode:

To read content of ‘myfile.txt’ and display it on monitor.

```
#include<fstream.h> //header file for file operations
void main()
{
char ch;
ifstream file(“myfile2.txt”); //open myfile.txt in default input mode
while(file)
{
  file.get(ch) // read a character from text file ‘myfile.txt’
  cout<<ch; // display a character from text file to screen
}
file.close();
} //end of main
```

Q2 Write a function in a C++ to read the content of a text file “DELHI.TXT” and display all those lines on screen, which are either starting with ‘D’ or starting with ‘M’.

```
void DispD_M()
{
ifstream File(“DELHI.TXT”) ;    // by default in input mode
char str[80];
while(File.getline(str,80))
{
if(str[0] = =’D’ || str[0] = =’M’)
cout<<str<<endl;
}
File.close();
}
```

Q3 Write a function in a C++ to count the number of lowercase alphabets present in a text file “BOOK.txt”.

Program to copy contents of file to another file.

```
#include<fstream>
int main()
{
   ifstream fin;
   fin.open("abc.txt");

   ofstream fout;
   fout.open("xyz.txt");

 char ch;
            while(!fin.eof())
        {  fin.get(ch);
           fout << ch;
        }
        fin.close();
        fout.close();
        return 0;
     }
```

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

The other prototype for these functions is:

```
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	end of the file

example:

```
file.seekg(-10, ios::cur);
```

Lets create a text file to learn working of tellg(), seekg() etc.


```
#include<fstream.h>
int main()
{
ofstream fout;
fout.open("aps.txt");
fout<<"APS BEAS";
fout.close();
return 0;
}
```

// working of seekg() and tellg()

```
#include<iostream.h>
#include<fstream.h>
#include<stdio.h>
void main()
{ // char ch;
 ifstream file("APS.txt",ios::in);
 if(!file)
 {cout<<"error";}
 else
 {   cout<<file.tellg()<<endl;
 // change get pointer value with the help of seekg
  file.seekg(4);
  //file.seekg(-2,ios::cur);

 // lets read a line from APS.TXT
 char str[80];
 file.getline(str,80);
                cout<<str<<endl;
  cout<< file.tellg()<<endl;

 }
 }
```

// seekp() and tellp()

```
#include<iostream.h>
#include<fstream.h>
void main()
{   ofstream file("APS1.txt",ios::out);
 if(!file)
 {cout<<"error";}
 else
 { cout<<file.tellp()<<endl;    // 0 empty file , mode is out
// lets write text into APS1
                file<<"APS Beas Rocks";

                cout<<file.tellp()<<endl;
/* 14 gives the current position of the put pointer
ie. from where next out op. is to be performed  */
                file.seekp(-10, ios::end);     // another version file.seekp(-5,ios::end)
/*moves put pointer (output) to a specified location i.e. where the next char
to be returned in next write op */

                cout<<file.tellp()<<endl;
                file<<"is awesome";   // see file content after this step
  }
 }
```

/* C++ Sequential Input and Output Operations with Files */

```
#include<iostream.h>      // optional
#include<fstream.h>
#include<process.h>
//<stdlib.h>  for exit()
#include<conio.h>

struct customer
{
char name[20];
float balance;
};
void main()
{  clrscr();
 customer savac;
cout<<"Enter your name: ";
cin.getline(savac.name, 20);
cout<<"Enter balance: ";
cin>>savac.balance;
ofstream fout;
```

```
fout.open("Saving", ios::out | ios::binary);
if(!fout)
{ cout<<"File can\'t be opened..!!\n";
cout<<"Press any key to exit...\n";
            getch();
            exit(1);
  }
fout.write((char *) & savac, sizeof(customer));
fout.close();              // close connection
// read it back now
                ifstream fin;
fin.open("Saving", ios::in | ios::binary);
fin.read((char *) & savac, sizeof(customer);
cout<<savac.name;
cout<<" has the balance amount of Rs.
"<<savac.balance<<"\n";

fin.close();
cout<<"\nPress a key to exit...\n";
                getch();
}
```