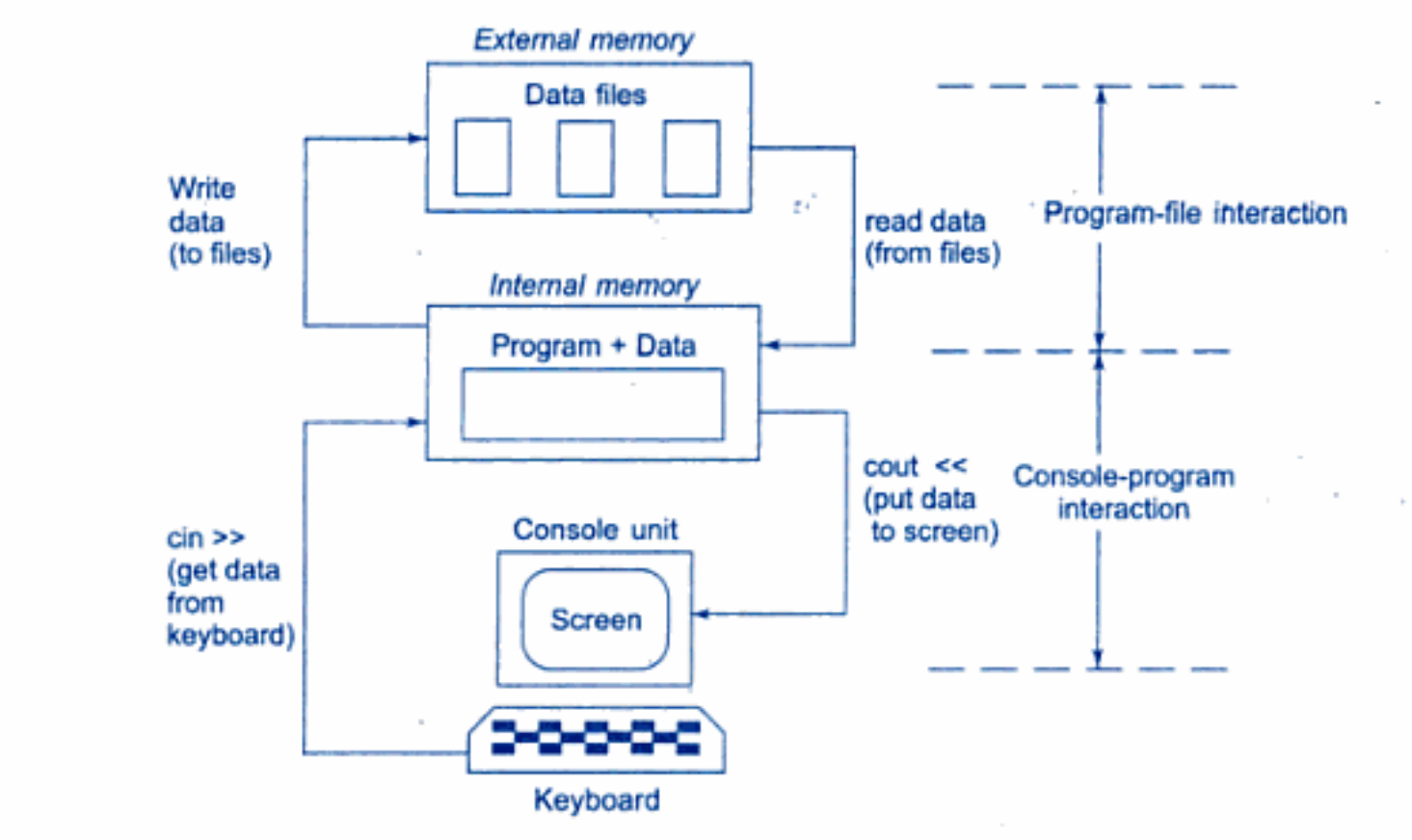
Data File Handling

Files

- A file is a collection of related data stored in a particular area on the disk.
- In C++, Programs can be designed to perform the read and write operations on these files
- A program involves either or both of following kinds of data communication:
  - Data transfer between the console unit and the program
  - Data transfer between the program and a disk file



Console-program-file interaction

Introduction

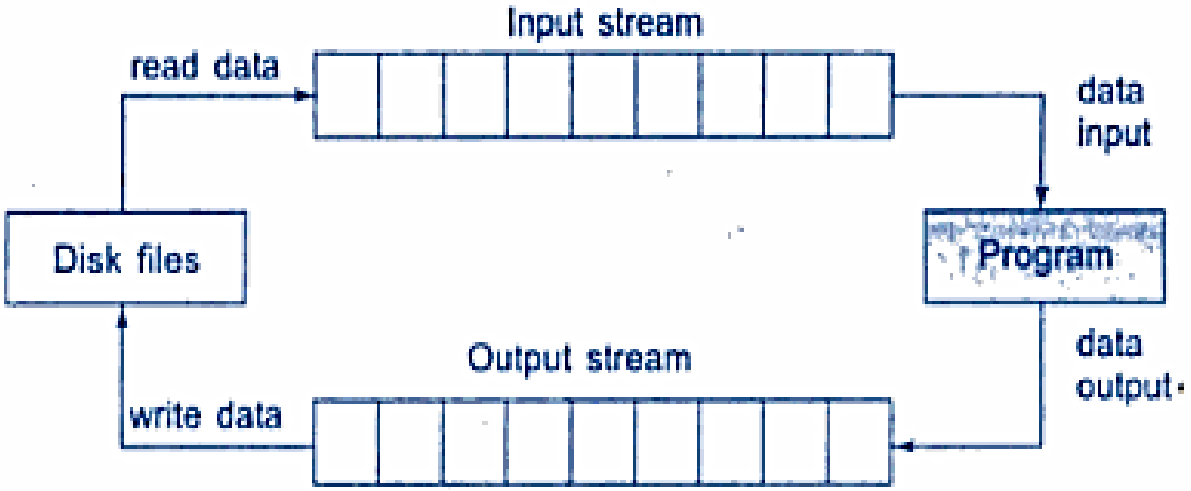
- Computer programs are associated to work with files as it helps in storing data & information permanently.
- File - itself a bunch of bytes stored on some storage devices.
- In C++ this is achieved through a component header file called *fstream.h*
- The I/O library manages two aspects- as interface and for transfer of data.
- The library predefine a set of operations for all file related handling through certain classes.

- C++ provides a new technique for handling I/O operations through mechanism known as streams.
- A stream refers to a **flow of data**.
- Classified in 2 categories:

1. Output stream
2. Input stream

In output stream flow of data is from program to the output device.

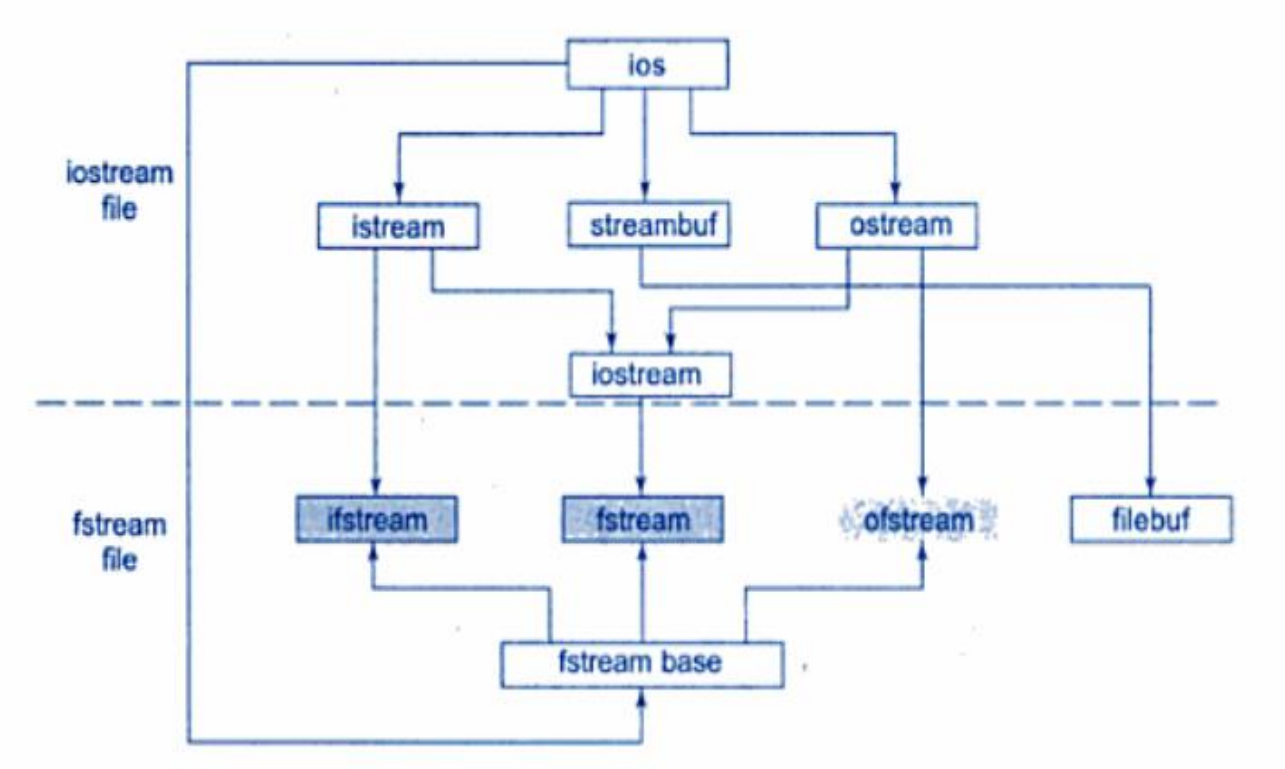
In input stream the flow of data is from input device to a program in main memory.



File Input and output streams

Classes for file stream operations

- The I/O system of C++ contains a set of classes that define the file handling methods.
- These include:
  - ifstream
  - ofstream
  - fstream
  - These classes are derived from **fstreambase** and from corresponding iostream class



System defined streams

User defined streams

Stream classes for file operations

Why to use Files:

- Convenient way to deal large quantities of data.
- Store data permanently (until file is deleted).
- Avoid typing data into program multiple times.
- Share data between programs.
- We need to know:

    how to "connect" file to program

    how to tell the program to read data

    how to tell the program to write data

    error checking and handling EOF

Opening and closing a file

- If we want to use a disk file, we need to decide the following things:
  - Suitable name for the file
  - Data type and structure
  - Purpose
  - Opening Method

Filename

- The file name is a string of characters that make up a valid filename for the operating system.
- It has two parts, a primary name and an optional period with extension
- Example:
  - Input.data
  - Test.doc

Opening a file

- For opening a file, 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***
- A file can be opened in two ways:
  - Using a constructor function of the class
  - Using the member function **open()** of the class

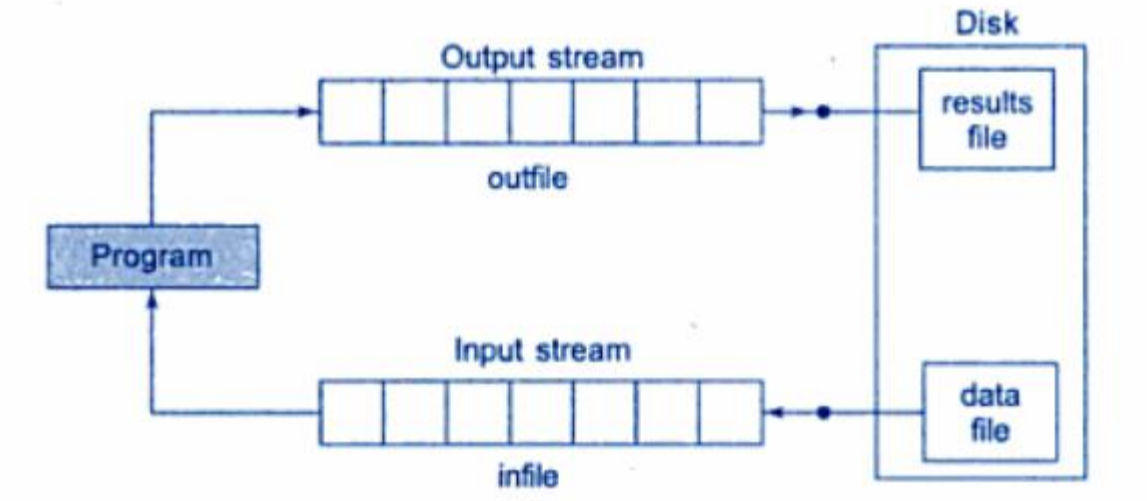
Opening files using constructor

- 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 following steps:
  - Create a file stream object to manage the stream using appropriate class
  - Initialize the file object with the desired filename.

- For example:

  ofstream outfile("results"); // output only

- This creates **outfile** as an **ofstream** object that manages the output stream.
- This statement opens the file results and attaches it to the output stream **outfile**



- The following statement declares **infile** as an **ifstream** object and attaches it to the file data for reading

```
ifstream infile (“data”);          //input only
```

The program may contain statements like:

```
outfile<<total;
outfile<<sum;
infile>>number;
infile>>string
```

- We can also use same file for both reading and writing data

```
Program1
……..
ofstream outfile(“salary”);    //creates outfile and connects “salary” to it
……..
Program2
…….
ifstream infile(“salary”);      //creates infile and connects “salary” to it
……
```

- Instead of using two program, one for writing data and another for reading data, we can use single program to do both operations on file.

```
…..
outfile.close();   //disconnect salary file from outfile
ifstream infile(“salary”);        // and connect to infile
…..
…..
infile.close();
```

Write file

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
   ofstream of("result.txt");
   of<<"hello";
   of.close();
   cout<<"data saved\n";

}
```

Read file

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
   char s[10];
   ifstream inf("result.txt");
   inf>>s;
   inf.close();
   cout<<s;
}
```

Two file streams working on same file

- Program1

…….

…….

ofstream outfile(“salary”);

……

……

Program2

……

……

ifstream infile(“salary”);

……

……

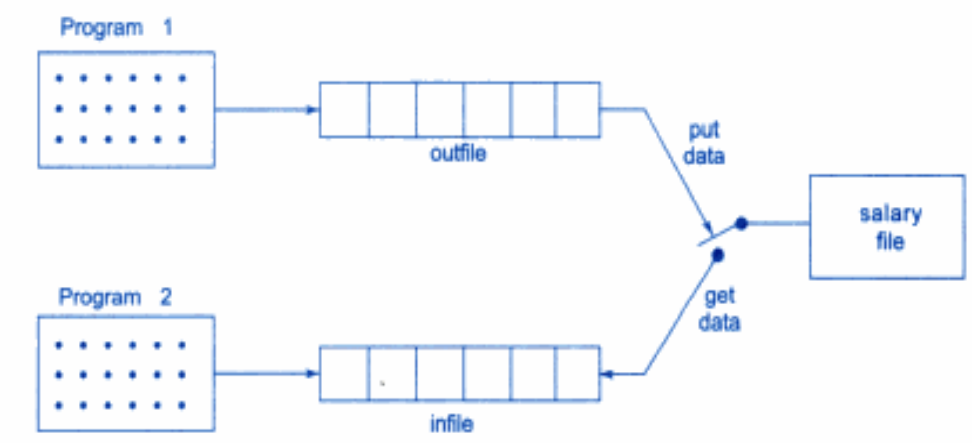


Fig. 11.5 ⇔ *Two file streams working on one file*

One program read write

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
   char s[30];
   ofstream of("result");
   of<<"hello";
   of.close();
   cout<<"data saved\n";
   ifstream inf(“result”);
   inf>>s;
   cout<<s;
   inf.close();
   return 0;
}
```

Opening files using open()

- This function is used to open multiple files that uses same stream object.
- Syntax:-

file_stream class stream_object;

stream_object.open(“filename”);

Example

```
ofstream outfile;
outfile.open(“Country.txt”);
……..
……..
outfile.close();
outfile.open(“Capital.txt”);
……..
……..
outfile.close();
```

Program to read write

```
int main()
{
    ofstream fout;
    fout.open("country.txt");
    fout<<"india\n";
    fout<<"USA\n";
    fout<<"UK\n";
    fout.close();

    fout.open("capital.txt");
    fout<<"Delhi\n";
    fout<<"Washington\n";
    fout<<"Londan\n";
    fout.close();
}
```

```
int main()
{
    const int n=80;
    char s[n];
     ifstream fin;
     fin.open("country");
     while(fin)
    {
      fin.getline(s,n);
      cout<<s;
    }
     fin.close();
     fin.open("capital");
     while(fin)
    {
      fin.getline(s,n);
      cout<<s;
    }
     fin.close();
```

Detecting end-of-file

- This condition is necessary for preventing any further attempt to read data from the file.

  while(fin)

  here fin is ifstream object which returns a value 0 if any error occur in file including end-of-file condition.

  if(fin1.eof()!=0)

  { exit(1);}

eof() is a member function of class ios. It returns non zero value if the end-of-file(eof) condition is encountered and zero otherwise.

Which header file is required to use file I/O operations?

a) <ifstream>

b) <ostream>

c) <fstream>

d) <iostream>

Which of the following is used to create an output stream?

a) ofstream

b) ifstream

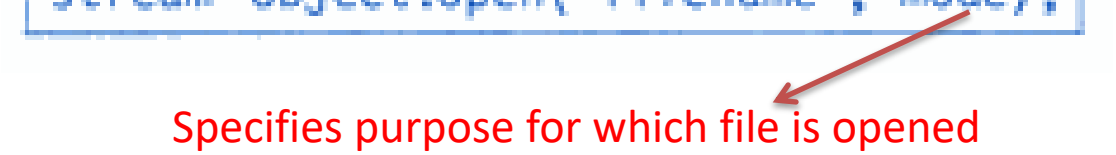
c) iostream

d) fsstream

File modes

- General form of **open()** with 2 arguments is:

```
stream-object.open("filename", mode);
```



- Prototype of member function contains default values as:

```
ios::in  for ifstream functions meaning open for reading only.
ios::out for ofstream functions meaning open for writing only.
```

Parameter	Meaning
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	Open fail if file does not exist
ios::noreplace	Open fail if file already exist
ios::out	Open file for writing only
ios::trunc	Delete contents of file

Examples

1. ofstream fileout;
 fileout.open(“hello”,ios::app);

2) int main()
{
fstream infile;
Infile.open(“data_file”, ios::in|ios::out)

}

append mod

```
int main()
{
    char s[30];
    ofstream fileout;
    ifstream filein;
    fileout.open("data",ios::app);
    fileout<<"hello world";
    fileout.close();

    filein.open("data",ios::in);
    filein.getline(s,30);
    cout<<s;
```

What is the use of ios::trunc mode?

a) To open a file in input mode

b) To open a file in output mode

c) To truncate an existing file to half

d) To truncate an existing file to zero

Which of the following is the default mode of the opening using the ofstream class?

a) ios::in

b) ios::out

c) ios::app

d) ios::trunc

What is the return type open() method?

a) int

b) char

c) bool

d) float

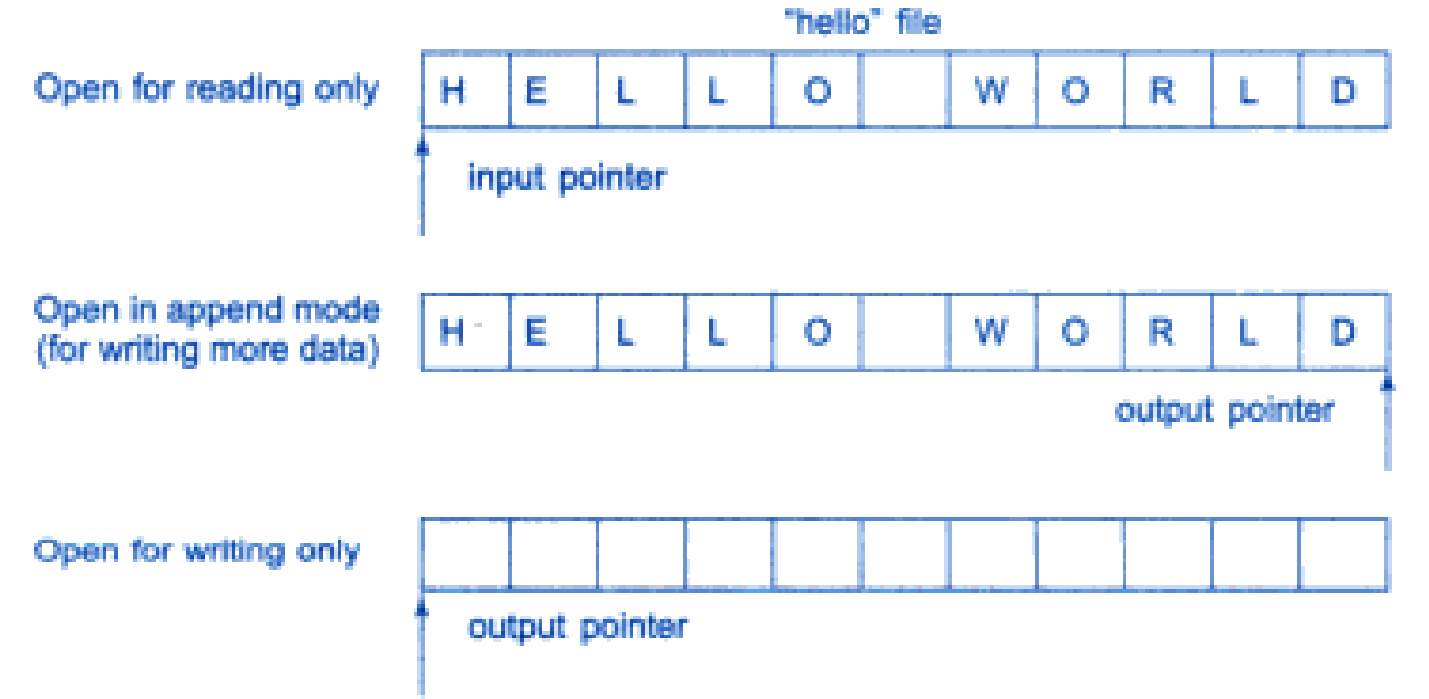
File pointers and their Manipulations

- Each file has two associated pointers known as the file pointers.
- Input pointer/get pointer
- Output pointer/put pointer

Default actions

- In read only mode the Input pointer is automatically set at the beginning.
- In write only mode, the existing contents are deleted and output pointer is set at the beginning.
- In append mode, the output pointer moves to the end of file.

- **File opened in read mode:**
  - I/p pointer automatically set at beginning.
- **File opened in write mode:**
  - Existing contents deleted & o/p pointers sets at beginning.
- To add more data, file is opened in append mode.



File pointer manipulators

- These are used to move file pointer to any desired position inside file.

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.

For example, the statement

```
infile.seekg(10);
```

moves the file pointer to the byte number 10.

```
int main()
{
    char s[30];
    ofstream fileout;
    fstream file;
    fileout.open("hello",ios::out);
    fileout<<"hello world";
    fileout.close();
    file.open("hello",ios::in|ios::out);
    cout<<file.tellg();
    cout<<file.tellp();
    file.seekp(6);
    file.seekg(6);
    cout<<file.tellg();
    cout<<file.tellp();
    file>>s;
    cout<<s;
}
```

Specifying the Offset

- The arguments to seek function represents the actual position in the file.
- seekp(offset, refposition);
- seekp(offset, refposition);
- Refposition takes one of the following three contants.
- ios::beg
- ios::cur
- ios::end

'Seek' functions **seekg()** and **seekp()** can also be used with two arguments as follows:

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

No. of bytes file pointer is to be moved

Specifies position from where pointer to be moved

Takes one of 3 constants:

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

Examples

Seek call	*Action*
fout.seekg(o, ios::beg);	Go to start
fout.seekg(o, ios::cur);	Stay at the current position
fout.seekg(o, ios::end);	Go to the end of file
Fout.seekg(m,ios::beg);	Move to (m + 1)th byte in the file
fout.seekg(m,ios::cur);	Go forward by m byte form the current position
fout.seekg(-m,ios::cur);	Go backward by m bytes from the current position
fout.seekg(-m,ios::end);	Go backward by m bytes form the end

Specifying the Offset

- The arguments to seek function represents the actual position in the file.
- seekp(offset, refposition);
- seekp(offset, refposition);
- Refposition takes one of the following three contants.
- ios::beg
- ios::cur
- ios::end

Sequential input and output operations

- put(),get() used to handle single character at a time.
- write(), read() functions are used to handle a stream of data.

```
file.put(ch);
file.get(ch);
```

```
 #include<string.h>
using namespace std;
int main()
{
    char s[80];
    cout<<"enter a string";
    cin>>s;
    int len=strlen(s);
     fstream file;
     file.open("test",ios::in|ios::out);
     for(int i=0;i<len;i++)
        file.put(s[i]);
    file.seekg(0);
    char ch;
    while(file)
    {
            file.get(ch);
            cout<<ch;
    }
    return 0;
}
```

write() and read()

- infile.read((char *)&V, sizeof(V));
- outfile.write((char *)&V, sizeof(V));

These functions take two arguments, first is address of variable V, second is length of that variable in bytes.

```
 int main()
{
    float height[4]={17.5,15.0,3.8,5.0};
    ofstream outfile;
    outfile.open("abc");
    outfile.write(( char *)height,sizeof(height));
    outfile.close();

    float p[4];
    ifstream infile;
    infile.open("abc");
    infile.read((char*) p,32);
    cout<<p[0]<<p[1];

    return 0;
}
```

Reading and Writing Class object

- The binary input output read() and write() are designed to write and read from disk file objects.
- These functions handles entire structure of an object as a single unit .
- function write() copies a class object from memory byte by byte with no conversion.

Example

```
class student
{
    char name[30];
    int roll_no;
    float marks;
  public:
    void getdata()
    {
            cout<<"enter name";
            cin>>name;
            cout<<"enter rollno";
            cin>>roll_no;
            cout<<"enter marks";
            cin>>marks;

    }
    void putdata()
    {
            cout<<name;
            cout<<roll_no;
            cout<<marks;
    }
};
```

```
int main()
{
    student ob1,ob2;
    ob1.getdata();
    fstream file;
    file.open("data",ios::in|ios::out);
    file.write((char*)&ob1,sizeof(ob1));

    file.seekg(0);
    file.read((char*) &ob2,sizeof(ob2));

    ob2.putdata();
}
```

Updating a file

- Updating means to perform following tasks:

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

- These actions require the file pointers to move to a particular location.
- This can be easily implemented if file contains objects of equal length.
- Size of each object can be obtained using:

```
int ob_len= sizeof(object);
```

Then location of desired object from beginning is:-

```
int location=(m-1)*ob_len;
```

- total number of objects in a file:-

```
int n=file_size/ob_len;
```

ERROR HANDLING FUNCTION

FUNCTION	RETURN VALUE AND MEANING
eof()	returns true (non zero) if end of file is encountered while reading; otherwise return false(zero)
fail()	return true when an input or output operation has failed
bad()	returns true if an invalid operation is attempted or any unrecoverable error has occurred.
good()	returns true if no error has occurred.

Which function is used in C++ to get the current position of file pointer in a file?

a) tellp()

b) get_pos()

c) get_p()

d) tell_pos()

Which function is used to reposition the file pointer?

a) moveg()

b) seekg()

c) changep()

d) go_p()