Disk I/O Operations (File Handling)

File handling

- When a large amount of data is to be handled, external devices such as hard disks are required 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 a disk.
- A program typically may involve following kinds of data communication:

1. Data transfer b/w the console unit and the program.

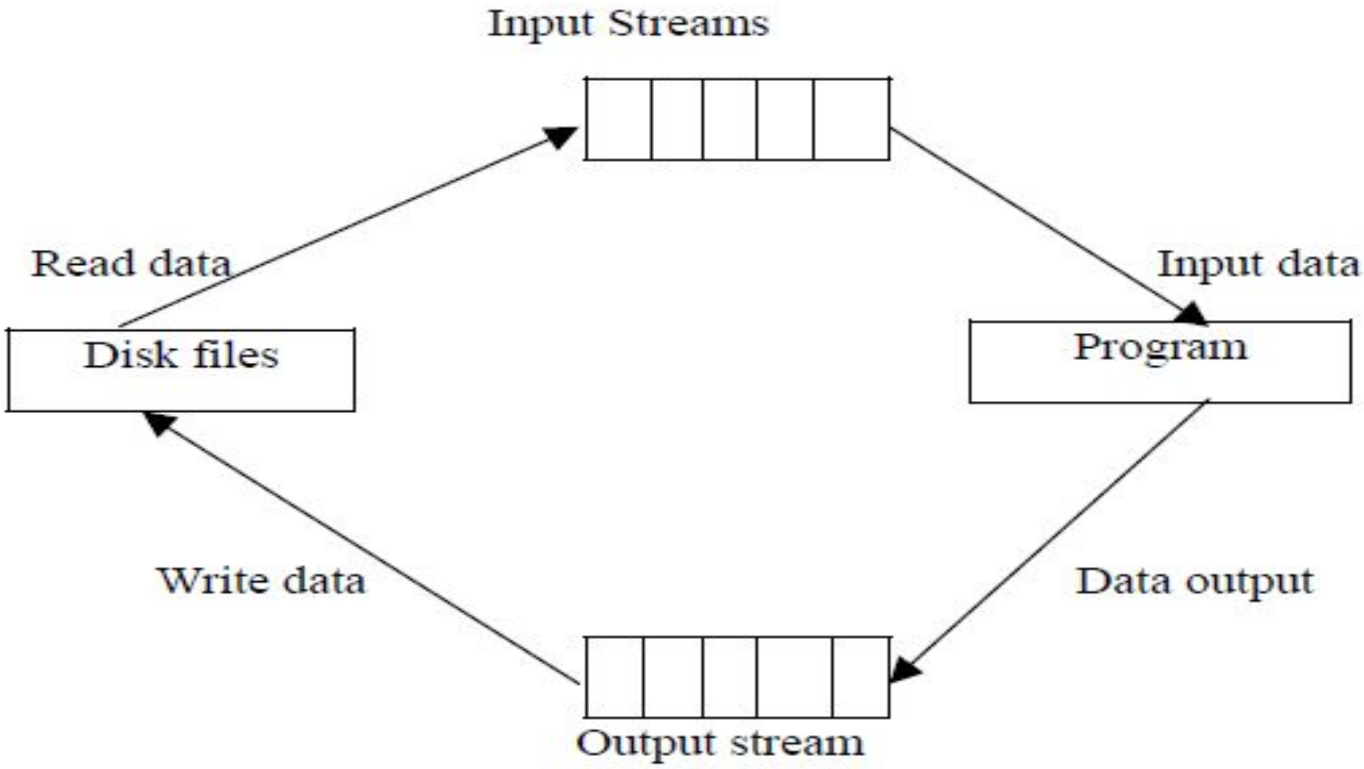
2. Data transfer b/w the program and disk files.

File handling

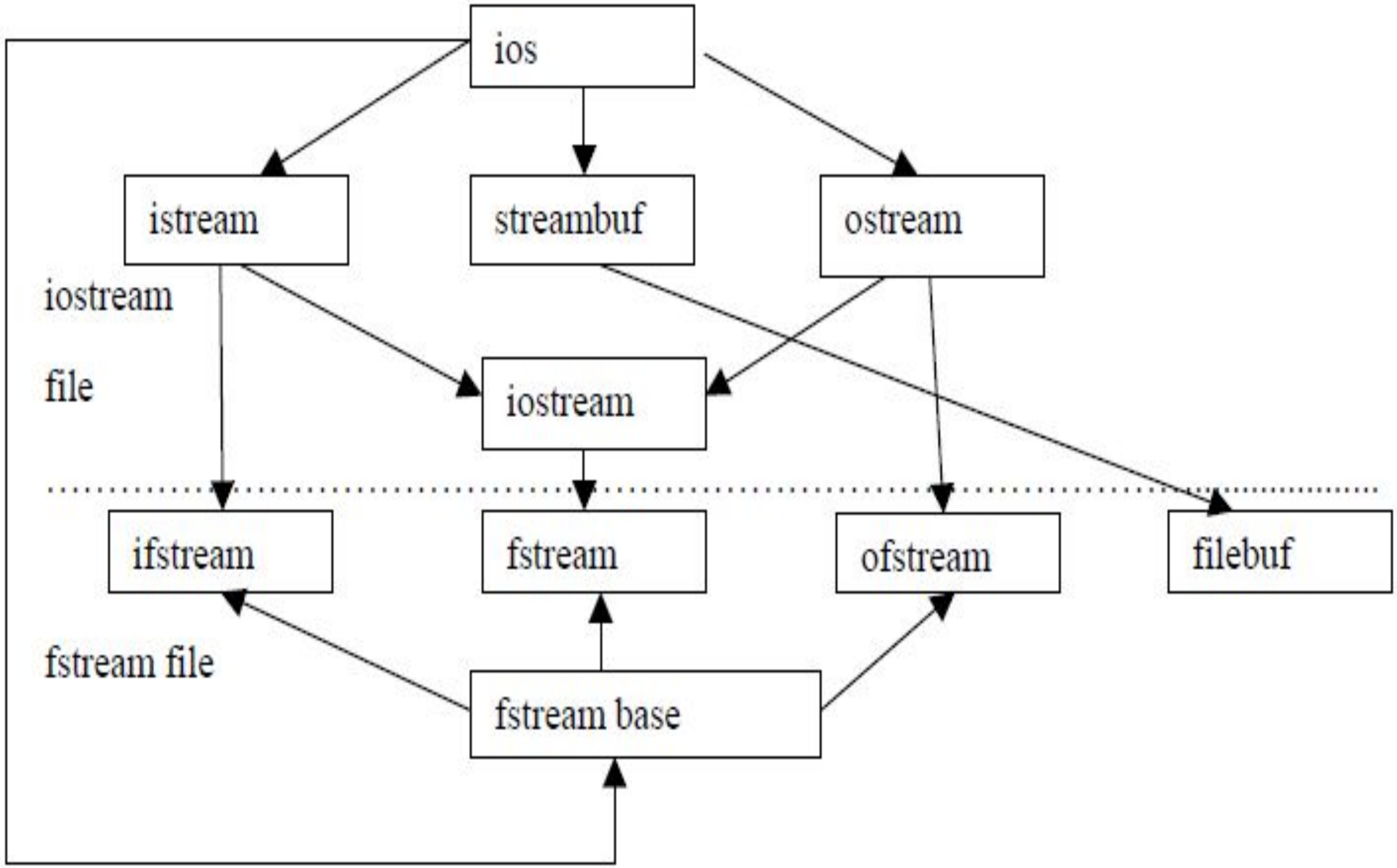
- Programs can be designed to perform the read and write operations on the disk files.
- The I/O system of C++ handles file operations which are very much similar to the console input and output operations.
- It uses **file streams** as an interface between the programs and files.
- The stream that supplies data to the program is called input stream and the one that receives data from the program is called output stream.
- In other words input stream extracts(reads) data from the file and output stream inserts(writes) data to the file.

File input and output streams

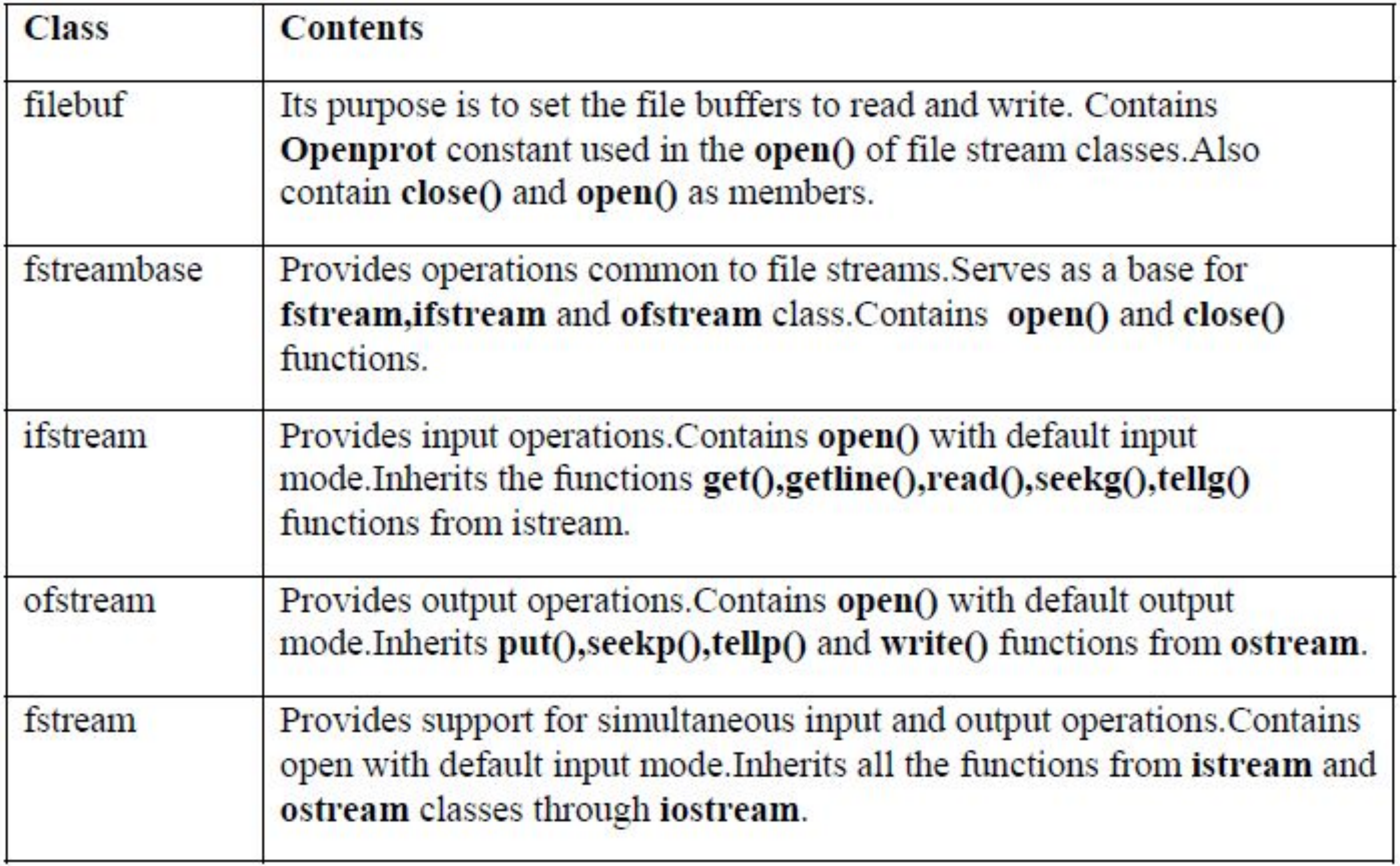
- The input operation involves the creation of an input stream and linking it with the program and input file.
- Similarly, the output operation involves establishing an output stream with the necessary links with the program and output file.



Stream classes for file operations



Stream classes for file operations



Class	**Contents**
filebuf	Its purpose is to set the file buffers to read and write. Contains **Openprot** constant used in the **open()** of file stream classes.Also contain **close()** and **open()** as members.
fstreambase	Provides operations common to file streams.Serves as 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 functions **get(),getline(),read(),seekg(),tellg()** functions from istream.
ofstream	Provides output operations.Contains **open()** with default output mode.Inherits **put(),seekp(),tellp()** and **write()** functions from **ostream**.
fstream	Provides support for simultaneous input and output operations.Contains open with default input mode.Inherits all the functions from **istream** and **ostream** classes through **iostream**.

Steps of File Operations

- For using a disk file the following things are necessary

1. Suitable name of file
2. Purpose ie. whether the write data or read data operation is to be performed on the file
3. File opening Method (i.e. app, ate, etc.)
4. Reading or Writing the file (file processing)
5. Closing the file

Steps of File Operations

- The filename is a string of characters that makeup a valid filename for the operating system.
- It may contain two parts ,primary name and optional period with extension.
- Examples are My_file.txt Input.data, Test.doc etc.
- For opening a file firstly a file stream is created and then it is linked to the filename.
- A file stream can be defined using the classes ifstream, ofstream and fstream that contained in the header file fstream.
- **The class to be used depends upon the purpose whether the write data or read data operation is to be performed on the file**

Opening a file

- A file can be opened in two ways:
  - Using constructor function of the class.
  - Using the member function of the class.
- First method is used when we use only one file in the stream.
- Second method is used when we want to manage multiple files using one stream.

Opening file using constructors

- Constructors as discussed is used to initialize an object while it is being created.
- Here file name is used to initialize the file stream object.
- This involves the following steps:
  - *Create file stream object to manage the stream using the appropriate class.*
    - ofstream is used to create output stream and
    - ifstream to create the input stream.
  - *Initialize the file object with the desired filename.*
    - Eg: ofstream outfile(“results.txt”);   //output only

      ifstream infile(“data.txt”);   //input only

Opening file using constructors(cont…)

- ofstream outfile(“results.txt”); create outfile as an ofstream object that manages the output stream.
- Similarly , ifstream infile (“data.txt”);statement declares infile as an ifstream object and attaches it to the file data for reading (input).
- **The same file name can be used for both reading and writing data**
- **Example:**

```
ofstream outfile (“salary”); //creates outfile and connects salary to it
outfile<<“72000 Rs ”;
outfile.close(); //disconnect salary from outfile and connect to infile
ifstream infile (“salary”);
infile>>str;
infile.close();
```

WORKING WITH SINGLE FILE

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
        char line[100],line1[100];
        ofstream outf("myfile.txt"); // if file is not exist it will be created
        cout<<"good bit="<<outf.good()<<endl;
        if(outf.good()) // good() function returns 1 if everything is ok
          cout<<"Everything is OK\n File is opened :\n";
        else{
          cout<<"Error: File does not exist!\n";
          return 0;
        }
        cout <<"enter some text: ";
        cin.getline(line,100);
        outf <<line<<"\n";
        outf.close(); // close the file
        }
```

WORKING WITH SINGLE FILE

```
    ifstream inf("myfile.txt"); // open file to read data
    cout<<"good bit="<<inf.good()<<endl;
    if(inf.good()) // good() function returns 1 if everything is ok
      cout<<"Everything is OK\n File is opened :\n";
    else{
      cout<<"Error: File does not exist!\n";
      return 0;
    }
    inf.getline(line1,1000); // read text from file
    cout <<"text is : " << line1 <<"\n";
    inf.close(); // close the file
    return 0;
}
```

Closing the file

- outfile.close();

This will disconnect the file “results” from the output stream outfile.

- infile.close();

This will disconnect the file “data” from the input stream infile

Note:

Outfile object of class ofstream will open an existing file If the file doesn’t exist, it is created. If it does exist, it is truncated and the new data replaces the old

Opening file Using the member function

- The function **open()** can be used to open multiple files that uses the same stream object sequentially.
- Create single stream object and use to open each file.
- Example:
- **File-stream-class stream_object_name;**

  Stream_object_name.**open (“filename”);**
  - ofstream fout; //object created
  - fout.open(“country”); //country file is created if not exist

Finding End of File

- While reading a data from a file,it is necessary to find where the file ends i.e end of file.
- The EOF is a signal sent to the program from the operating system when there is no more data to read.
- The programmer cannot predict the end of file,if the program does not detect end of file,the program drops in an infinite
- An **ifstream** object such as fin returns a value of 0 if any error occurs in the file operation including the end-of –file condition.
- Thus the while loop terminates when 'fin' returns a value of zero on reaching the end-of –file condition.
- while(fin){statement;}
- There is an another approach to detect the end of file condition.The statement

```
if(fin1.eof() !=0 ) //eof() returns non-zero if end-of-file encounter
{
exit(1);
}
```

WORKING WITH MULTIPLE FILES

```
#include <iostream >
#include<fstream>
using namespace std;
int main(){
        ofstream fout; //object created
        fout.open("country.txt"); //1st file
        fout<<"United states of America \n";
        fout<<"United Kingdom\n";
        fout<<"South korea\n";
        fout.close();
        fout.open("capital.txt"); //2nd file
        fout<<"Washington\n";
        fout<<"London\n";
        fout<<"Seoul \n";
        fout.close();
        const int N =80;
        char line[N];
        ifstream fin;
        fin.open("country.txt");
        cout<<"contents of country file \n";
```

```
        while (fin)
        {
        fin.getline(line,N);
        cout<<line<<endl;
        }
        fin.close();
        fin.open("capital.txt");
        cout<<"contents of capital file\n";
        while(fin)
        {
        fin.getline(line,N);
        cout<<line<<endl;
        }
        fin.close();
        return 0;
}
```

File Opening Modes

- The ifstream and ofstream constructors and the function open() are used to open the files.
- Up to now a single arguments is used that is filename However,these functions can take two arguments, the second one for specifying the file mode.
- The general form of function open() with two arguments is:

  stream-object.open(“filename”,mode);
- first argument specifies the name and location of the file to be opened and the second argument defines the mode in which the file should be opened.

mode	Stand for	Description
ios::in	input	File open for reading: the internal stream buffer supports input operations.
ios::out	output	File open for writing: the internal stream buffer supports output operations.
ios::ate	at end	The output position starts at the end of the file. Edit is possible.
ios::app	append	All output operations happen at the end of the file, appending to its existing contents.
ios::trunc	truncate	Any contents that existed in the file before it is open are discarded.
ios::binary	binary	Operations are performed in binary mode rather than text.

File Opening Modes

- We can combine two or more file opening modes by **OR**ing them together as:
- ofstream outfile; outfile.open("file.txt", ios::out | ios::trunc);
- This can be used to open a file in write mode and to delete the text if already exists.
- Similar way, you can open a file for reading and writing purpose as follows:

  fstream afile;

  afile.open("file.txt", ios::out | ios::in);

- **Default Open Modes :**

ifstream	ios::in
ofstream	ios::out
fstream	ios::in | ios::out

File Opening Modes

- Opening a file in ios::out mode also opens in the ios::trunc mode by default.
- Both ios :: app and ios :: ate take us to the end of the file when it is opened.
- The difference between the two parameters is that the ios :: app allows us to add data to the end of file only, while ios :: ate mode permits us to add data or to modify the existing data any where in the file.

File Pointers and Manipulators

- Each file has two pointers known as file pointers: **input pointer** and **output pointer**.
- **Input pointer** is used for reading the contents of a given file location
- **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.
- When a file is opened in read-only mode, the input pointer is automatically set at the beginning so that file can be read from the start.
- Similarly when a file is opened in write-only mode the existing contents are deleted and the output pointer is set at the beginning. This enables us to write the file from start.
- In case an existing file is to be opened in order to add more data,the file is opened in 'append' mode.This moves the pointer to the end of file.

Functions for Manipulating the File pointers

- For controlling the movement of file pointers, file stream classes provide the following functions :

1. **seekg() Moves get pointer (input)to a specified location.**
2. **seekp() Moves put pointer (output) to a specified location.**
3. **tellg() Give the current position of the get pointer.**
4. **tellp() Give the current position of the put pointer.**

Functions for Manipulating the File pointer

- For example, the statement

  infile.seekg(10); will move the pointer to the 10th index.

  Note: That the bytes in a file are numbered beginning from zero. Therefore ,the get pointer will point to the 11th byte in the file.

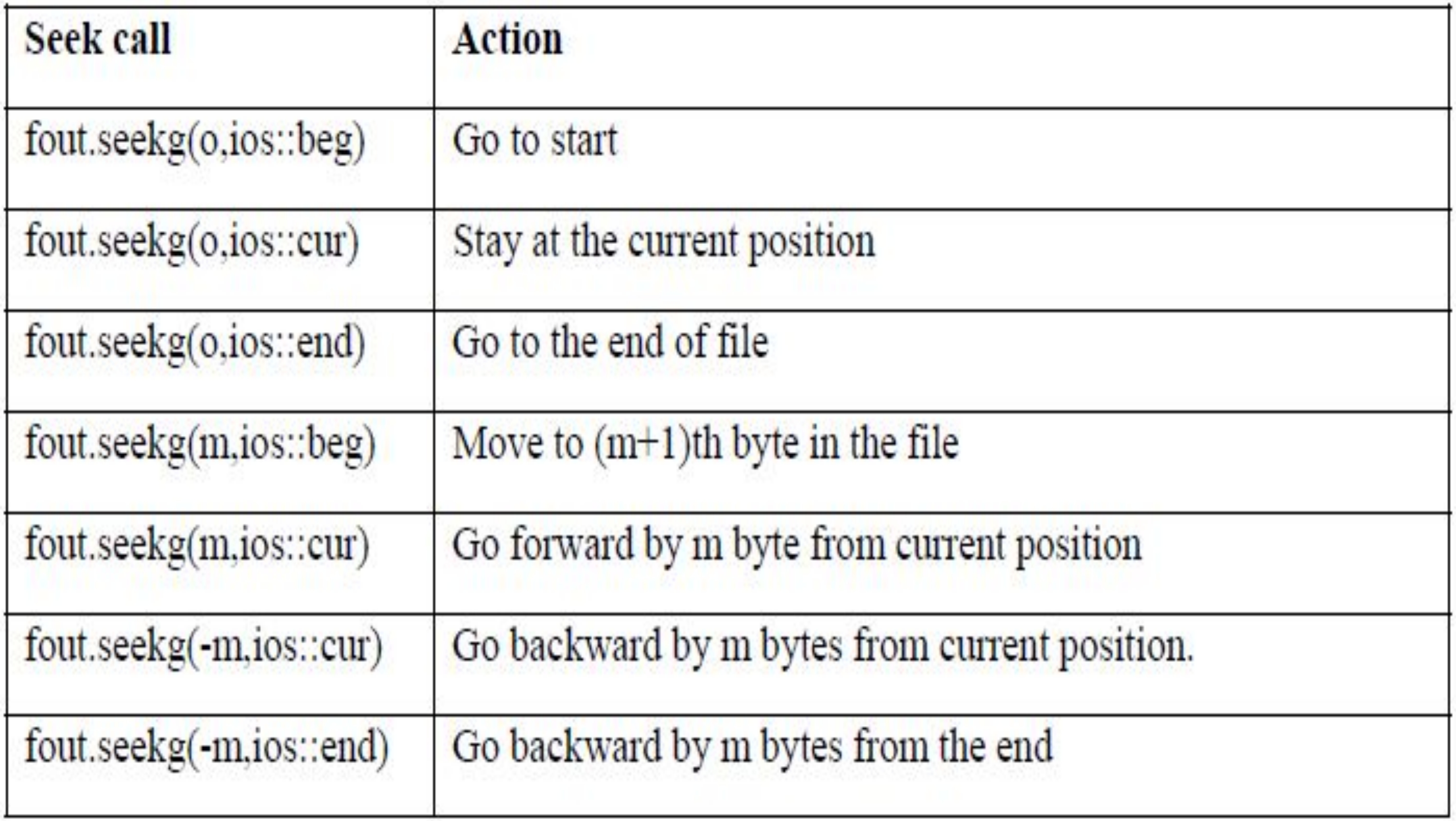
Consider the following statements:

- **ofstream fileout;**
- **fileout.open("hello.txt",ios::app);**
- **int p=fileout.tellp();**
- On execution of these statements, the output pointer is moved to the end of file "hello.txt"
- And the value of p will represent the number of bytes in the file.

Specifying the Offset

- **seekg() and seekp()** functions can also be used with two arguments as follows:
- **seekg (offset,refposition); //move get pointer**
- **seekp (offset,refposition); //move put pointer**
- 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 file
- **ios::cur** //Current position of the pointer
- **ios::end** //End of file

Specifying the Offset



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 from current position
fout.seekg(-m,ios::cur)	Go backward by m bytes from current position.
fout.seekg(-m,ios::end)	Go backward by m bytes from the end

Exercises

- **Q1 Write statements using seekg() to achieve the following:**
  (a) To move the pointer by 15 positions backward from current position.
  (b) To go to the beginning after an operation is over.
  (c) To go backward by 20 bytes from the end.
  (d) To go to 50th byte in the file.

Predict the Output

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

#include<conio.h>

using namespace std;

int main() {

    ofstream outf("myfile66.txt"); // if file is not exist it will be created

    outf <<"abc def !";

    outf.seekp(6); // set output file pointer to 7th byte

    outf <<"ghi jkl";

outf.close();

    return 0;
```

Input and Output Operations on files

- put() and get() are designed for handling a single character at a time.
- write(),read() are designed to write and read blocks of binary data.
- **put()-** writes a single character to the associated stream.
- **get() -** reads a single character from the associated stream.

Using put() and get() Functions

```
#include<iostream.h.>
#include<fstream.h>
#include<string.h>
 void main()
{
char string[50];
        cout<<“Enter a String:”;
        cin>> string;
        int len=strlen(string);
        fstream file;
        file.open(“ Myfile.txt”, ios::in| ios::out);
        for(int i=0;i<len;i++)
                file.put(string[i]);      // writing character wise into a file
file.seekg(0);   // go to the start of the file
char ch;
while(file)    // detecting end of file
        {
        file.get(ch);   // reading character wise from the file
                cout<<ch;
        }
}
```

O/p

```
Enter a string :programming  // input
programming                   // output
```

write() and read() functions

- Handle the data in binary form.
- The values are stored in the disk file in the same format in which they are stored in the internal memory.
- The binary format is more accurate for storing the numbers as they are stored in the exact internal representation.
- There is no conversions while saving the data and therefore saving is faster.
- **Binary vs character formats: Example:**
- Integer value 2594 is represented as follows:

  binary format (2 bytes): 00001010 00100010

  character format(4 bytes): 2 5 9 4

write() and read() functions

Syntax:

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

- First argument is the address of the variable V
- Second argument is the length of that variable in bytes.

Reading and writing a class object

- Since the class objects are the central elements of C++ programming, it is quite natural that the language supports features for reading and writing from the disk files objects directly.
- Binary input and output functions read() and write() are designed to do this job.
- These functions handle the entire structure of an object as a single unit.
- **Only data members are written on the disk file and not the member functions**

Error Handling during File Operations

- There are many problems encounterd while dealing with files like
  - a file which we are attempting to open for reading does not exist.
  - The file name used for a new file may already exist.
  - We are attempting an invalid operation such as reading past the end of file.
  - There may not be any space in the disk for storing more data.
  - We may use invalid file name.
  - We may attempt to perform an operation when the file is not opened for that purpose.
- The class ios support several member functions that can be used to read the status recorded in a file stream.

Error Handling Functions

	NAME	PURPOSE
1	eof()	Returns true if eof file is reached
2	fail()	Returns true if fail bit, bad bit or hard flag is set
3	bad()	Returns true if bad bit or hard flag is set
4	good()	Returns true, if everything is ok