Files and Streams

Files and Streams

- Introduction to file handling: text file Vs. binary file
- Hierarchy of file stream classes: Functions of File Stream classes
- Steps to process a File in a program:

-Create an stream object (input or output or i/o) by declaring the stream to be of appropriate class

-Associate a file with this stream object that is to open the file by using constructor or by using Open() function.

-Process the file

-Closing the file by using close() function

- File modes
- Sequential access:

-The get(), getline() and put() functions

-The read() and write() functions

-Reading and writing class objects

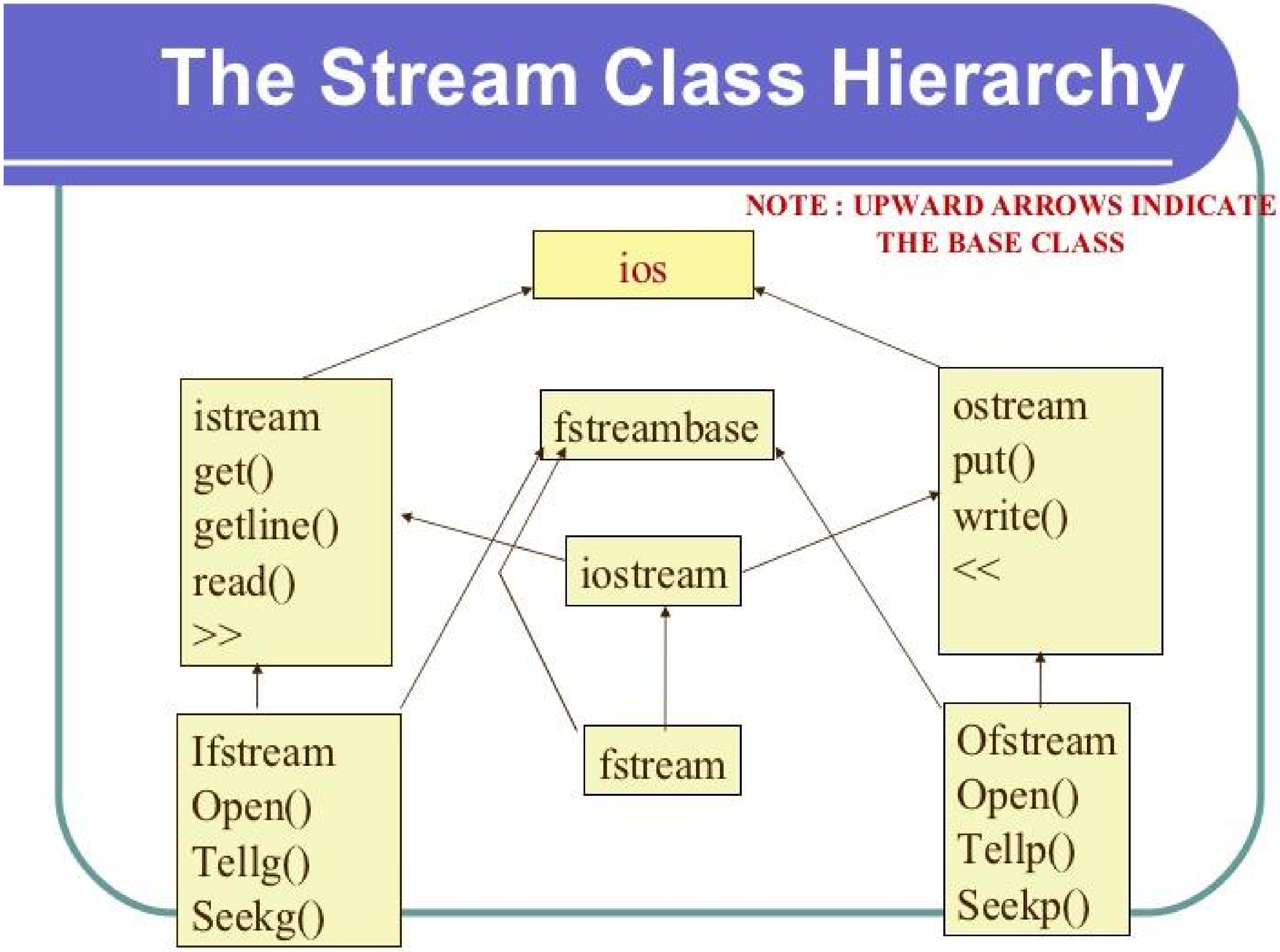
- File pointers and their Manipulations: two file pointers (get pointer, put_pointer), Functions for manipulation of file pointers (seekg(), seekp(), tellg(), tellp())
- Updating a File: Random Access:
- Error handling during file operation: Error handling functions (eof(), fail(), bad(), good())

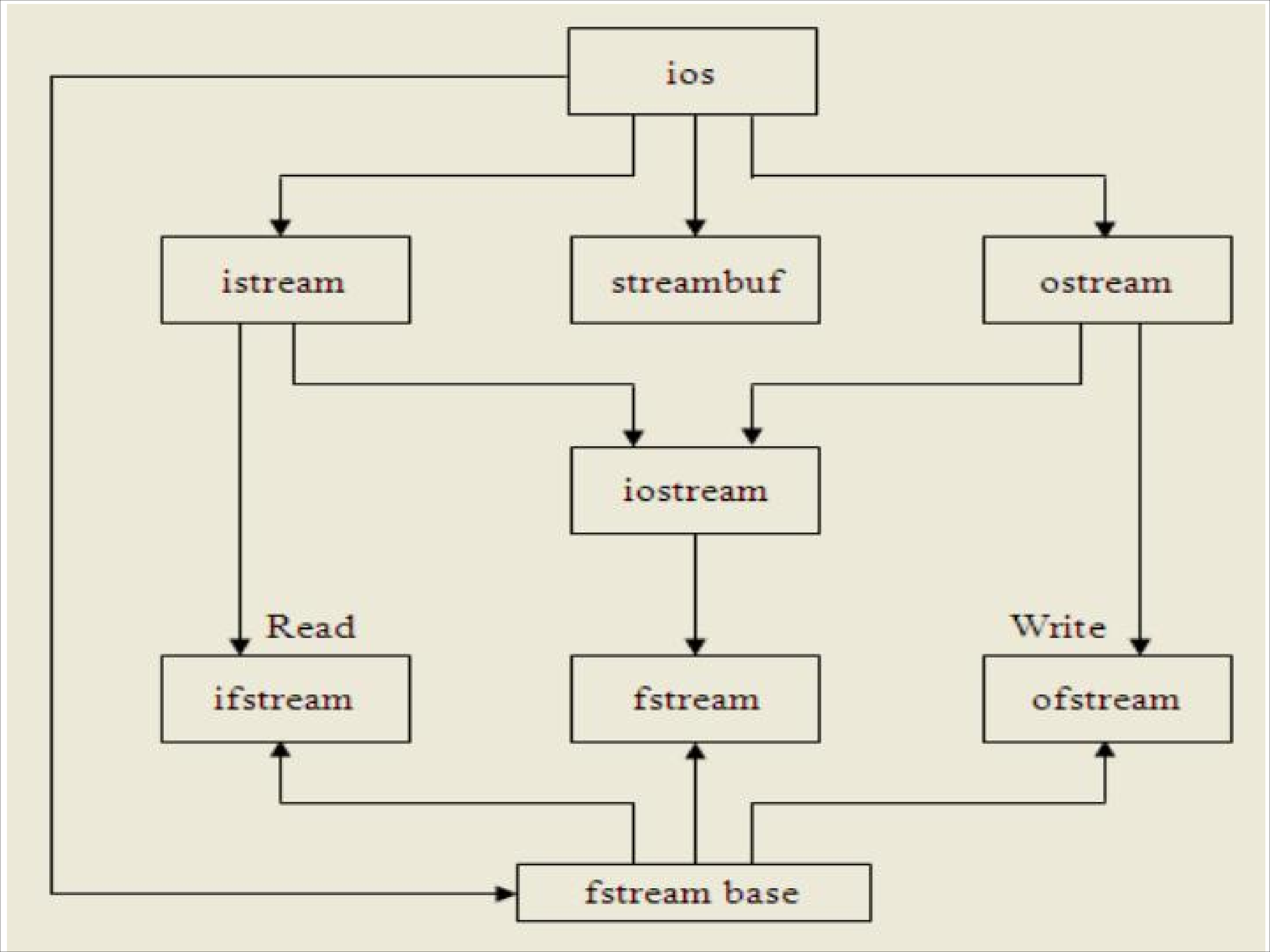
File Handling

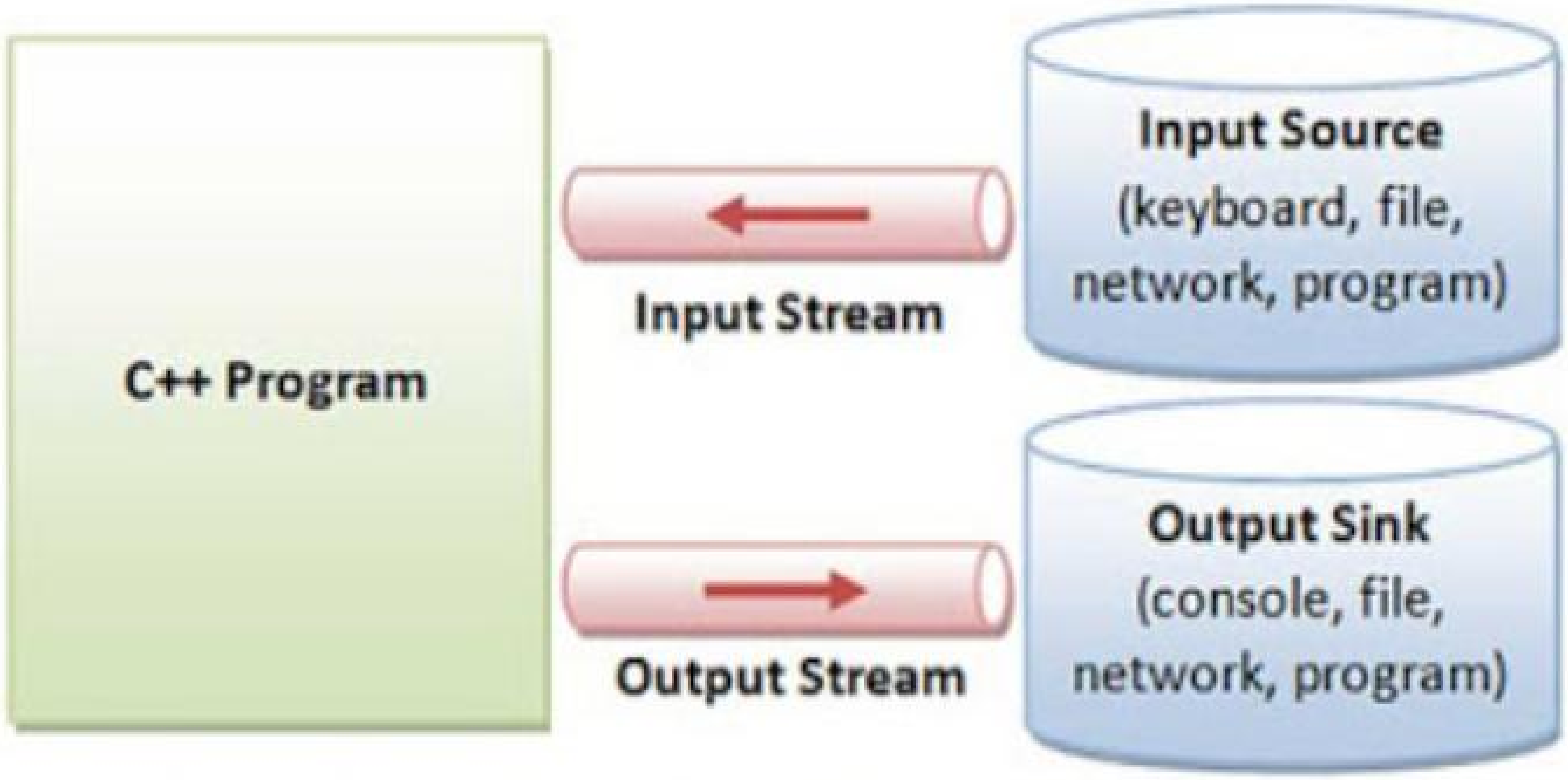
Introduction to file handling: text file Vs. binary file

- File. The information / data stored under a specific name on a storage device, is called a file.
- Stream. It refers to a sequence of bytes.
- Text file. It is a file that stores information in ASCII characters. In text files, each line of text is terminated with a special character known as EOL (End of Line) character or delimiter character. When this EOL character is read or written, certain internal translations take place.
- Binary file. It is a file that contains information in the same format as it is held in memory. In binary files, no delimiters are used for a line and no translations occur here.

Hierarchy of file stream classes: Functions of File Stream classes




ios
istream
streambuf
ostream
iostream
Read
Write
ifstream
fstream
ofstream
fstream base


C++ Program
Input Stream
Output Stream
Input Source
(keyboard, file, network, program)
Output Sink
(console, file, network, program)

Internal Data Formats:

- Text: `char`, `wchar_t`
- `int`, `float`, `double`, etc.

External Data Formats:

- Text in various encodings (US-ASCII, ISO-8859-1, UCS-2, UTF-8, UTF-16, UTF-16BE, UTF16-LE, etc.)
- Binary (raw bytes)

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.

Opening a file: Two Methods

1)Opening a file using Constructor method

Syntax:

```
ofstream outFile("sample.txt");    //output only
ifstream inFile(“sample.txt”);  //input only
```

2)Opening a file open() mehtod

Syntax:

```
Stream-object.open(“filename”, mode)
    ofstream outFile;//write data into file
    outFile.open("sample.txt");
    ifstream inFile;//read data from file
    inFile.open("sample.txt");
```

- All these flags can be combined using the bitwise operator OR (|). For example, if we want to open the file example.bin in binary mode to add data we could do it by the following call to member function open():

```
fstream file;
file.open ("example.bin", ios::out | ios::app | ios::binary);
```

Closing File

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

File mode parameter **[FILE MODES]**	Meaning
ios::app	Append to end of file
ios::ate	go to end of file on opening
ios::binary	file open in binary mode
ios::in	open file for reading only
ios::out	open file for writing only
ios::nocreate	open fails if the file does not exist
ios::trunc	delete the contents of the file if it exist

Sequential access:

INPUT AND OUTPUT OPERATION

put() and get() function

the function put() writes a single character to the associated stream. Similarly, the function get() reads a single character form the associated stream.

Example :

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

write() and read() function

write() and read() functions write and read blocks of binary data.

Example:

```
file.read((char *)&obj, sizeof(obj));
file.write((char *)&obj, sizeof(obj));
```

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.

Basic Operation On Text File

File I/O is a five-step process:

1. Include the header file fstream in the program.
2. Declare file stream object.
3. Open the file with the file stream object.
4. Use the file stream object with >>, <<, or other input/output functions.
5. Close the files.

Program to write in a text file

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

int main()
{
    ofstream fout;
    fout.open("out.txt");

    char str[300] = "Time is a great teacher but
            unfortunately it kills all its pupils. Berlioz";

    //Write string to the file.
    fout << str;

    fout.close();
    return 0;
}
```

//Program to read from text file and display it

```
#include<fstream>
#include<iostream>
using namespace std;
int main()
{
    ifstream fin;
    fin.open("out.txt");

    char ch;

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

```
//Program to count number of lines
#include<fstream>
#include<iostream>
using namespace std;
int main()
{
    ifstream fin;
    fin.open("out.txt");

    int count = 0;
    char str[80];

    while(!fin.eof())
    {
        fin.getline(str,80);
        count++;
    }
   cout << "Number of lines in file are " <<
count;
    fin.close();
    return 0; }
```

```
//Program to count number of characters.
#include<fstream>
#include<iostream>
using namespace std;
int main()
{
    ifstream fin;
    fin.open("out.txt");

    int count = 0;
    char ch;
    while(!fin.eof())
    {
        fin.get(ch);
        count++;
    }
   cout << "Number of characters in file are "
<< count;

    fin.close();
    return 0;
```

```
//Program to count number of words
#include<fstream>
#include<iostream>
using namespace std;
int main()
{
    ifstream fin;
    fin.open("out.txt");

    int count = 0;
    char word[30];

    while(!fin.eof())
    {
        fin >> word;
        count++;
    }
    cout << "Number of words in file are "
<< count;
    fin.close();
    return 0;}
```

```
//Program to copy contents of file to another
file.
#include<fstream>
using namespace std;
int main()
{
    ifstream fin;
    fin.open("out.txt");

    ofstream fout;
    fout.open("sample.txt");
    char ch;
    while(!fin.eof())
    {
        fin.get(ch);
        fout << ch;
    }
    fin.close();
    fout.close();
    return 0;
}
```

Read/Write Class Objects from/to File:

Given a file “Input.txt” in which every line has values same as instance variables of a class.

Read the values into the class’s object and do necessary operations.

To write object's data members in a file :

```
// Here file_obj is an object of ofstream
file_obj.write((char *) & class_obj, sizeof(class_obj));
```

To read file's data members into an object :

```
// Here file_obj is an object of ifstream
file_obj.read((char *) & class_obj, sizeof(class_obj));
```

Example:

Input.txt :

```
Micheal 19 1806
Kemp 24 2114
Terry 21 2400
```

Operation : Print the name of the highest rated programmer.

Output :

```
Terry
```

```
// C++ program to demonstrate read/write of class
// objects in C++.
#include <iostream>
#include <fstream>
using namespace std;
// Class to define the properties
class Contestant {
public:
    // Instance variables
    string Name;
    int Age, Ratings;

    int input();    // Function declaration of input() to input info

    int output_highest_rated();    // Function declaration of output_highest_rated() to extract info from file Data Base
};
```

```
// Function definition of input() to input info
int Contestant::input()
{
    // Object to write in file
    ofstream file_obj;
    // Opening file in append mode
    file_obj.open("Input.txt", ios::app);
     // Object of class contestant to input data in file
    Contestant obj;
    // Feeding appropriate data in variables
    string str = "Micheal";
    int age = 18, ratings = 2500;
    // Assigning data into object
    obj.Name = str;
    obj.Age = age;
    obj.Ratings = ratings;
    // Writing the object's data in file
    file_obj.write((char*)&obj, sizeof(obj));
// Feeding appropriate data in variables
    str = "Terry";
    age = 21;
    ratings = 3200;
    // Assigning data into object
    obj.Name = str;
    obj.Age = age;
    obj.Ratings = ratings;

    // Writing the object's data in file
    file_obj.write((char*)&obj, sizeof(obj));

    return 0;
}
```

```
// Function definition of
output_highest_rated() to extract info from
file Data Base
int Contestant::output_highest_rated()
{
    // Object to read from file
    ifstream file_obj;
    // Opening file in input mode
    file_obj.open("Input.txt", ios::in);
    // Object of class contestant to input data
in file
    Contestant obj;

    // Reading from file into object "obj"
    file_obj.read((char*)&obj, sizeof(obj));

    // max to store maximum ratings
    int max = 0;
     // Highest_rated stores the name of
highest rated contestant
    string Highest_rated;
    // Checking till we have the feed
    while (!file_obj.eof())      {
        // Assigning max ratings
        if (obj.Ratings > max) {
            max = obj.Ratings;
            Highest_rated = obj.Name;
              }

        // Checking further
        file_obj.read((char*)&obj,
sizeof(obj));
    }

    // Output is the highest rated contestant
    cout << Highest_rated;
    return 0;
}
```

```
int main()
{
    // Creating object of the class
    Contestant object;

    // Inputting the data
    object.input();

    // Extracting the max rated contestant
    object.output_highest_rated();

    return 0;
}
```

Output:

```
Terry
```

Basic Operation On Binary File

When data is stored in a file in the binary format, reading and writing

data is faster because no time is lost in converting the data from one format to another format. Such files are called binary files.

```
//This following program explains how to create binary files and also how to read, write, search, delete and modify data from binary files.
#include<iostream>
#include<fstream>
#include<cstdio>
using namespace std;
class Student
{
    int admno;
    char name[50];
public:
    void setData()
    {
        cout << "\nEnter admission no. ";
        cin >> admno;
        cout << "Enter name of student ";
        cin.getline(name,50);
  }
    void showData()
    {
        cout << "\nAdmission no. : " << admno;
        cout << "\nStudent Name : " << name;
    }

    int retAdmno()
    {
        return admno;
    }
};
```

```
/* function to write in a binary file.*/

void write_record()
{
    ofstream outFile;
    outFile.open("student.dat", ios::binary |
ios::app);

    Student obj;
    obj.setData();

    outFile.write((char*)&obj, sizeof(obj));

    outFile.close();
}
```

```
/*  function to display records of file */

 void display()
{
    ifstream inFile;
    inFile.open("student.dat", ios::binary);

    Student obj;

    while(inFile.read((char*)&obj, sizeof(obj)))
    {
        obj.showData();
    }

    inFile.close();
}
```

```
/* function to search and display from
binary file */
void search(int n)
{
    ifstream inFile;
    inFile.open("student.dat", ios::binary);

    Student obj;

    while(inFile.read((char*)&obj,
sizeof(obj)))
    {
        if(obj.retAdmno() == n)
        {
            obj.showData();
        }
    }
    inFile.close();
}
```

```
/* function to delete a record*/
void delete_record(int n)
{
    Student obj;
    ifstream inFile;
    inFile.open("student.dat", ios::binary);
    ofstream outFile;
    outFile.open("temp.dat", ios::out |
ios::binary);
   while(inFile.read((char*)&obj, sizeof(obj)))
    {
        if(obj.retAdmno() != n)
        {
            outFile.write((char*)&obj,
sizeof(obj));
        }    }
    inFile.close();
    outFile.close();
    remove("student.dat");
    rename("temp.dat", "student.dat");
}
```

```
/* function to modify a record*/
void modify_record(int n)
{
    fstream file;
    file.open("student.dat",ios::in | ios::out);
    Student obj;
    while(file.read((char*)&obj, sizeof(obj)))
    {
        if(obj.retAdmno() == n)
        {
        cout << "\nEnter the new details of
student";
            obj.setData();
            int pos = -1 * sizeof(obj);
            file.seekp(pos, ios::cur);
            file.write((char*)&obj, sizeof(obj));
        }
    }
     file.close();
}
```

```
 main()
{
    //Store 4 records in file
    for(int i = 1; i <= 4; i++)
       write_record();

    //Display all records
    cout << "\nList of records";
    display();

    //Search record
    cout << "\nSearch result";
    search(100);
    //Delete record
    delete_record(100);
    cout << "\nRecord Deleted";
    //Modify record
    cout << "\nModify Record 101 ";
    modify_record(101);
}
```

File pointers and their Manipulations: two file pointers (get pointer, put_pointer), Functions for manipulation of file pointers (seekg(), seekp(), tellg(), tellp())

All file objects hold two file pointers that are associated with the file. These two file pointers provide two integer values. These integer values indicate the exact position of the file pointers in the number of bytes in the file. The read or write operations are carried out at the location pointed by these file pointers .One of them is called get pointer (input pointer), and the second one is called put pointer (output pointer). During reading and writing operations with files, these file pointers are shifted from one location to another in the file. The (input) get pointer helps in reading the file from the given location, and the output pointer helps in writing data in the file at the specified location. When read and write operations are carried out, the respective pointer is moved.

While a file is opened for the reading or writing operation, the respective file pointer input or output is by default set at the beginning of the file. This makes it possible to perform the reading or writing operation from the beginning of the file. The programmer need not explicitly set the file pointers at the beginning of files. To explicitly set the file pointer at the specified position, the file stream classes provides the following functions:

Read mode: When a file is opened in read mode, the get pointer is set at the beginning of the file, as shown in Figure. Hence, it is possible to read the file from the first character of the file.

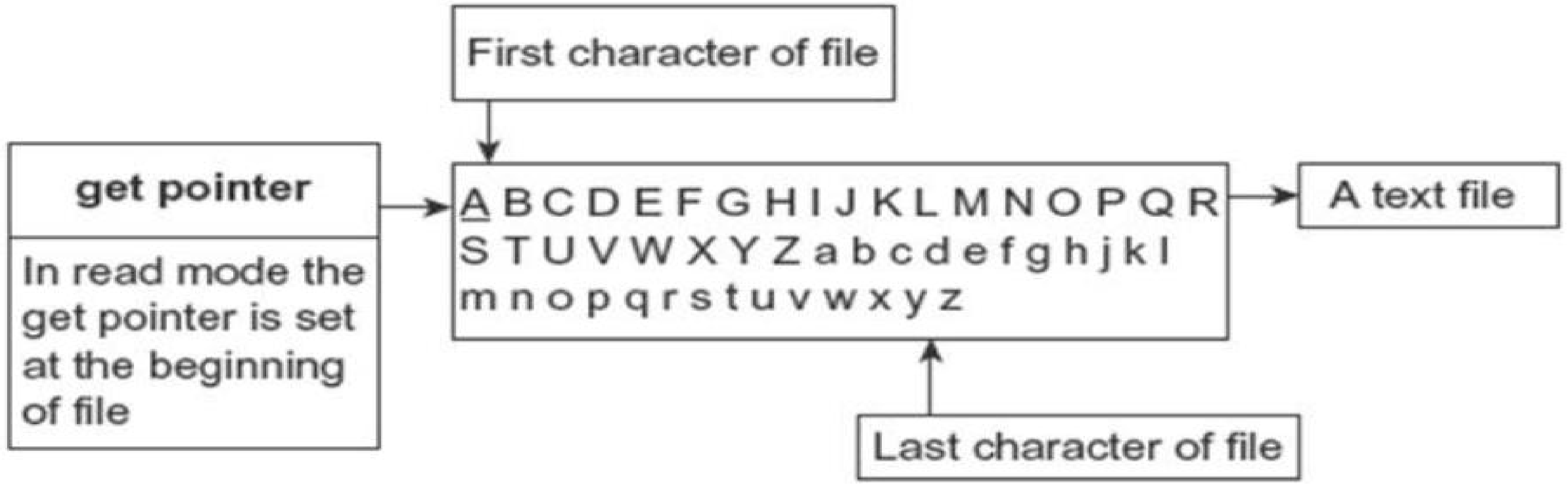

First character of file
get pointer
In read mode the get pointer is set at the beginning of file
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h j k l m n o p q r s t u v w x y z
A text file
Last character of file

Fig: Status of get pointer in read mode

Write Mode: When a file is opened in write mode, the put pointer is set at the beginning of the file, as shown in Figure. Thus, it allows the write operation from the beginning of the file. In case the specified file already exists, its contents will be deleted.

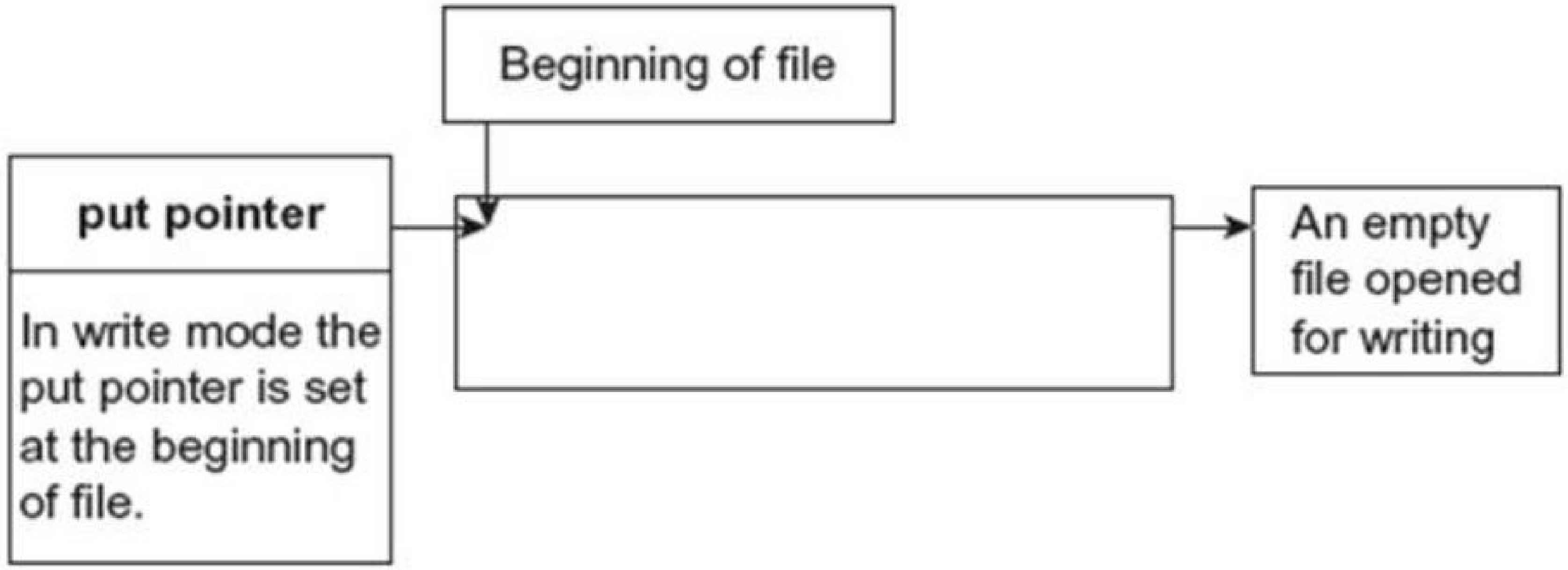

Beginning of file
put pointer
In write mode the put pointer is set at the beginning of file.
An empty file opened for writing

Fig: Status of put pointer in write mode

Append Mode: This mode allows the addition of data at the end of the file. When the file is opened in append mode, the output pointer is set at the end of the file, as shown in Figure. Hence, it is possible to write data at the end of the file. In case the specified file already exists, a new file is created, and the output is set at the beginning of the file. When a pre-existing file is successfully opened in append mode, its contents remain safe and new data are appended at the end of the file.

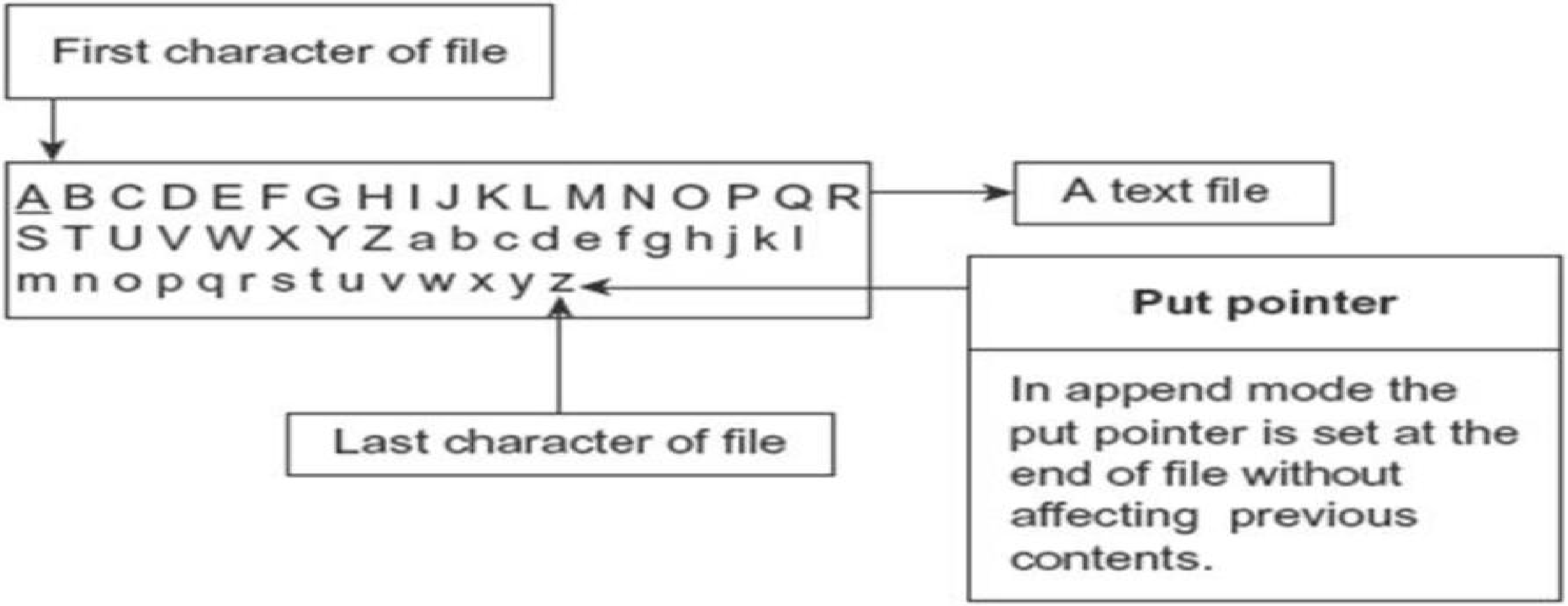
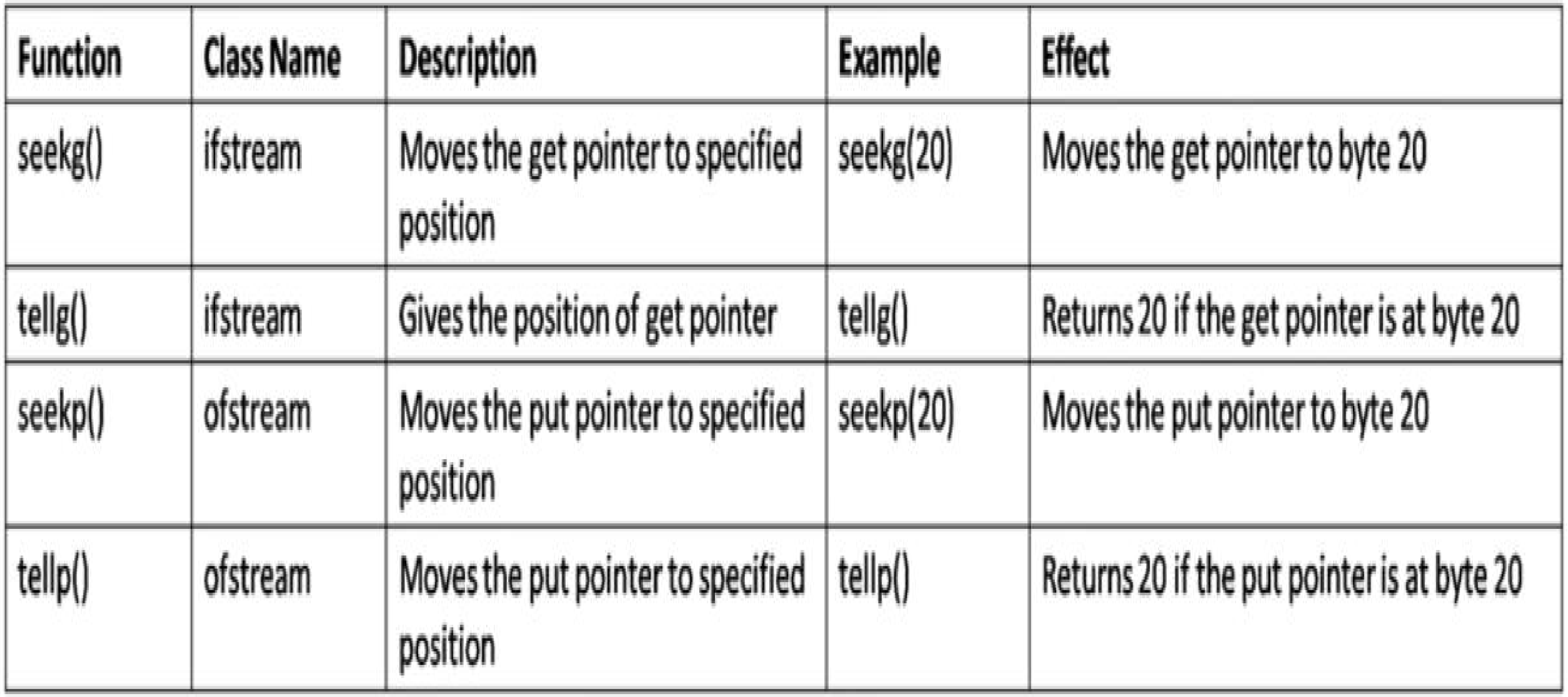


Fig: Status of put pointer in append mode

C++ has four functions for the setting of points during file operation. The position of the curser in the file can be changed using these functions. These functions are described in Table.

Table: File pointer handling functions



Function	Class Name	Description	Example	Effect
seekg()	ifstream	Moves the get pointer to specified position	seekg(20)	Moves the get pointer to byte 20
tellg()	ifstream	Gives the position of get pointer	tellg()	Returns 20 if the get pointer is at byte 20
seekp()	ofstream	Moves the put pointer to specified position	seekp(20)	Moves the put pointer to byte 20
tellp()	ofstream	Moves the put pointer to specified position	tellp()	Returns 20 if the put pointer is at byte 20

As given in Table, the seekg() and tellg() are member functions of the ifstream class. All the above four functions are present in the class fstream. The class fstream is derived from ifstream and ofstream classes. Hence, this class supports both input and output modes, as shown in Figure. The seekp() and tellp() work with the put pointer, and tellg() and seekg() work with the get pointer.

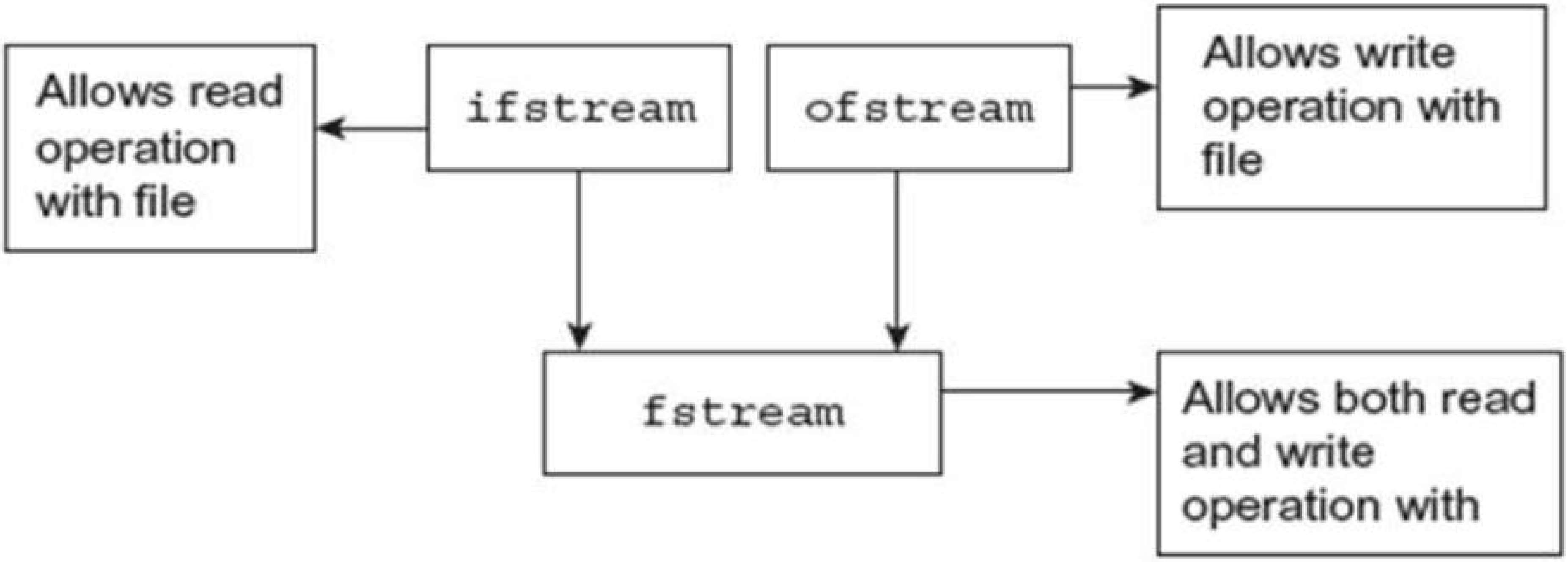

Allows read operation with file
ifstream
ofstream
Allows write operation with file
fstream
Allows both read and write operation with

Fig: Derivation of fstream class

In C++ we have a get pointer and a put pointer for getting (i.e. reading) data from a file and putting(i.e. writing) data on the file respectively.

seekg() is used to move the get pointer to a desired location with respect to a reference point.

Syntax: file_pointer.seekg (number of bytes ,Reference point);

Example: fin.seekg(10,ios::beg);

tellg() is used to know where the get pointer is in a file.

Syntax: file_pointer.tellg();

Example: int posn = fin.tellg();

seekp() is used to move the put pointer to a desired location with respect to a reference point.

Syntax: file_pointer.seekp(number of bytes ,Reference point);

Example: fout.seekp(10,ios::beg);

tellp() is used to know where the put pointer is in a file.

Syntax: file_pointer.tellp();

Example: int posn=fout.tellp();

The reference points are:

ios::beg – from beginning of file

ios::end – from end of file

ios::cur – from current position in the file.

In seekg() and seekp() if we put – (minus) sign in front of number of bytes then we can move backwards.

Function	Operation
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.
fout . seekg(0, ios :: beg)	go to start
fout . seekg(0, ios :: cur)	stay at current position
fout . seekg(0, ios :: end)	go to the end of file
fout . seekg(m, ios :: beg)	move to m+1 byte in the file
fout . seekg(m, ios :: cur)	go forward by m bytes from 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 from the end

```
//C++ program to demonstrate example of
tellg() and tellp() function.
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    fstream file;
    //open file sample.txt in and Write mode
    file.open("sample.txt",ios::out);
    if(!file) {
        cout<<"Error in creating file!!!";
        return 0;
    }
    //write A to Z
file<<"ABCDEFGHIJKLMNOPQRSTUVW
XYZ";
    //print the position
cout<<"Current position is:
"<<file.tellp()<<endl;
    file.close();
```

```
     //again open file in read mode
    file.open("sample.txt",ios::in);
    if(!file)
    {
        cout<<"Error in opening file!!!";
        return 0;
    }
    cout<<"After opening file position is:
"<<file.tellg()<<endl;
    //read characters untill end of file is not
found
    char ch;
    while(!file.eof())
    {
        cout<<"At position : "<<file.tellg();
//current position
        file>>ch;   //read character from file
        cout<<" Character \""<<ch<<"\""<<endl;
    }
    file.close(); //close the file
    return 0; }
```

Error Handling:

Sometimes during file operations, errors may also creep in.

Example,

A file being opened for reading might not exist.

Or a file name used for a new file may already exist.

Or an attempt could be made to read past the end-of-file.

Or such as invalid operation may be performed.

There might not be enough space in the disk for storing data.

To check for such errors and to ensure smooth processing, C++ file streams inherit 'stream-state' members from the ios class that store the information on the status of a file that is being currently used. The current state of the I/O system is held in an integer, in which the following flags are encoded :

Name	Meaning
eofbit	1 when end-of-file is encountered, 0 otherwise.
failbit	1 when a non-fatal I/O error has occurred, 0 otherwise
badbit	1 when a fatal I/O error has occurred, 0 otherwise
goodbit	0 value

Error Handling Functions

There are several error handling functions supported by class ios that help you read and process the status recorded in a file stream.

Following table lists these error handling functions and their meaning :

Function	Meaning
int bad()	Returns a non-zero value if an invalid operation is attempted or any unrecoverable error has occurred. However, if it is zero (false value), it may be possible to recover from any other error reported and continue operations.
int eof()	Returns non-zero (true value) if end-of-file is encountered while reading; otherwise returns zero (false value).
int fail()	Returns non-zero (true) when an input or output operation has failed.
int good()	Returns non-zero (true) if no error has occurred. This means, all the above functions are false. For example, if fin.good() is true, everything is okay with the stream named as fin and we can proceed to perform I/O operations. When it returns zero, no further operations can be carried out.
clear()	Resets the error state so that further operations can be attempted.

The above functions can be summarized as

eof() returns true if eofbit is set;

bad() returns true if badbit is set.

fail() function returns true if failbit is set;

good() returns true there are no errors. Otherwise, they return false.

These functions may be used in the appropriate places in a program to locate the status of a file stream and thereby take the necessary corrective measures.

Example:

```
int main(){
ifstream fin;
fin.open("master", ios::in);
while(!fin.fail())
{
            :       // process the file
}
if(fin.eof())
{
            :        // terminate the program
}
else if(fin.bad())
{
            :        // report fatal error
}
else
{           fin.clear();     // clear error-state
flags
}
}
```

Program to check file has been successfully opened or not.[Error Checking]

```
#include<fstream.h>
#include<constream.h>
void main()
{
   clrscr();
   ifstream in ("text");
   if (!in) cerr <<"\n File is not opened"<<in;
   else cerr <<"\n file is opened"<<in;
}
```

OUTPUT

```
File is opened 0x8f15ffd2
```

Explanation: In the above program, an existing file text is opened for reading. The file is opened successfully. Hence, no error is generated. The if() statement checks the contents in objects using the following statement: if (!in). The value of the object is **0x8f15ffd2.** In case the operation fails, the value of the object in would be 0x8f160000.

```
#include<fstream.h>
#include<constream.h>
#include<iomanip.h>
main()
{
    clrscr();
    char c, f_name[10];
    cout<<“\n Enter file name:”;
    cin>>f_name;
    ifstream in(f_name);
    if (!in)
    {
    cerr<<“ Error in opening file”<<f_name<<endl;
    return 1;
    }
    in>>resetiosflags(ios::skipws);
    while (in)
    {
    in>>c;
    cout<<c;
    }
    return 0;
}
```

OUTPUT

```
Enter file name : TEXT
One Two Three Four
1 2 3 4
** The End **
```

Explanation: In the above program-using constructor of the class ifstream, a file is opened. The file that is to be opened is entered by the user during program execution. If the file does not exist, the ifstream object (in) contains 0. The if() statement checks the value of an object in, and if the operation fails, a message will be displayed and the program is terminated. In case the file exists, using the while() loop, the contents of the file are read one character at a time and displayed on the screen. You can observe the use of the NOT operator with object in if() and the while() statement. If the statement in>>resetiosflags (ios::skipws); is removed, the contents of the file would be displayed without space in one line. The operator >> ignores the white space character. Consider the following statement:

```
With NOT operator
if (!in)
{
   statement1;
   else
   statement2
}
```

The NOT operator is used in association with the object. It is also possible to perform the operation without the use of the NOT operator. The following statement works opposite as compared with the above statement:

```
Without NOT operator
if (in)
{
   statement1;
   else
   statement2
}
```

```cpp
/*  C++ Program to Read Write Student
Details using File Handling  */
#include<iostream>
#include<fstream>
using namespace std;

// define a class to store student data
class student
{
   int roll;
   char name[30];
   float marks;
public:
   student() { }
   void getData(); // get student data from
user
   void displayData(); // display data
};
```

```cpp
void student :: getData()
{
   cout << "\nEnter Roll No. :: ";
   cin >> roll;
   cin.ignore(); // ignore the newline char
inserted when you press enter
   cout << "\nEnter Name :: ";
   cin.getline(name, 30);
   cout << "\nEnter Marks :: ";
   cin >> marks;
}

void student :: displayData()
 {
   cout << "\nRoll No. :: " << roll << endl;
   cout << "\nName :: " << name << endl;
   cout << "\nMarks :: " << marks << endl;
}
```

```
int main()
{
   student s[3]; // array of 3 student objects
   fstream file;
   int i;

file.open("C:\\Users\\acer\\Documents\\file4.txt", ios :: out); // open file for writing
    cout << "\nWriting Student information to the file :- " << endl;
    cout << "\nEnter 3 students Details to the File :- " << endl;

   for (i = 0; i < 3; i++)
    {
      s[i].getData();
      // write the object to a file
      file.write((char *)&s[i], sizeof(s[i]));
    }

   file.close(); // close the file


file.open("C:\\Users\\acer\\Documents\\file4.txt", ios :: in); // open file for reading
   cout << "\nReading Student information to the file :- " << endl;

   for (i = 0; i < 3; i++)
    {
      // read an object from a file
      file.read((char *)&s[i], sizeof(s[i]));
      s[i].displayData();
    }

   file.close(); // close the file

   return 0;
}
```

16.27 Write a program to copy content of one file in another file in reverse order. Display the contents of the screen.

```
#include<fstream.h>
#include<conio.h>
#include<process.h>
main()
{
   clrscr();
   char s[12],t[12],c;
   fstream out;
   ifstream in;
   cout<<"\n Enter a Source file name:";
   cin>>s;
   cout<<"\n Enter a target file name:";
   cin >>t;
   in.open(s,ios::in ); //| ios::nocreate);
```

```
if (in.fail())
{
cout<<“\nFile”<<s <<“ Not Found”;
exit(1);
}
else
in.seekg(0,ios::end);
out.open(t,ios::out | ios::nocreate);
int b;
if (out.fail())
{
out.open(t,ios::out);
in.seekg(0,ios::end);
b=in.tellg();
```

```
    for (int i=1;i<=b;i++)
    {
    in.seekg(-i,ios::end);
    in.get(c);
    out.put(c);
    cout<<c;
    }
    }
    else
    cout<<“\nTarget file alredy exist.”;
    in.close();
    out.close();
    return 0;
}
```

```
OUTPUT
Enter a Source file name : cpp
Enter a target file name : cp2
GnimmargorP detneirO tcejbO
```

Explanation: In the above program, source and target file names are entered. The content of the source file is copied to the target file in reverse order. The source file is opened for reading, and the target file is opened for writing. Using the tellg() function, the size of the source file is obtained and stored in the variable b. The for loop executes from 1 to b (size of source file). The seekg() function moves the get file pointer in the reverse order, that is, from end to top. The argument –i specifies the number of bytes to be read from the end of the file. The character read by the get() function is written to the target file by the put() function. The cout() statement displays the contents of the variable c on the screen.