Object Oriented programming using C++

File Handling

INTRODUCTION

A program typically involves either or both of the following kinds of data communication:

- Data transfer between the console unit and the program.
- Data transfer between the program and a disk file.

INTRODUCTION

- A file is a collection of related data stored in a particular area on the disk. Program can be designed to perform the read and write operation on these data

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:

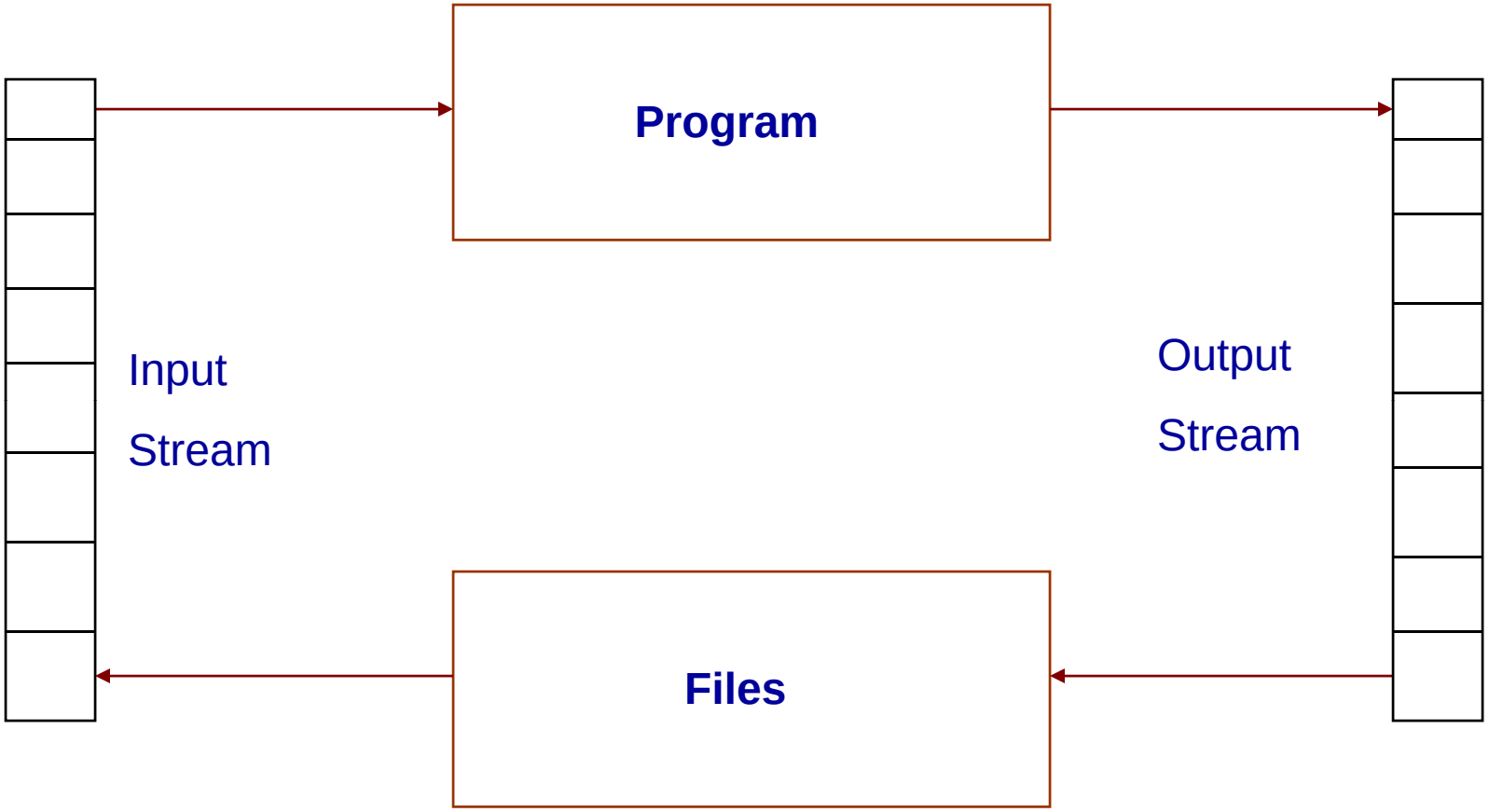
- ✓how to "connect" file to program
- ✓how to tell the program to read data
- ✓how to tell the program to write data
- ✓error checking and handling EOF

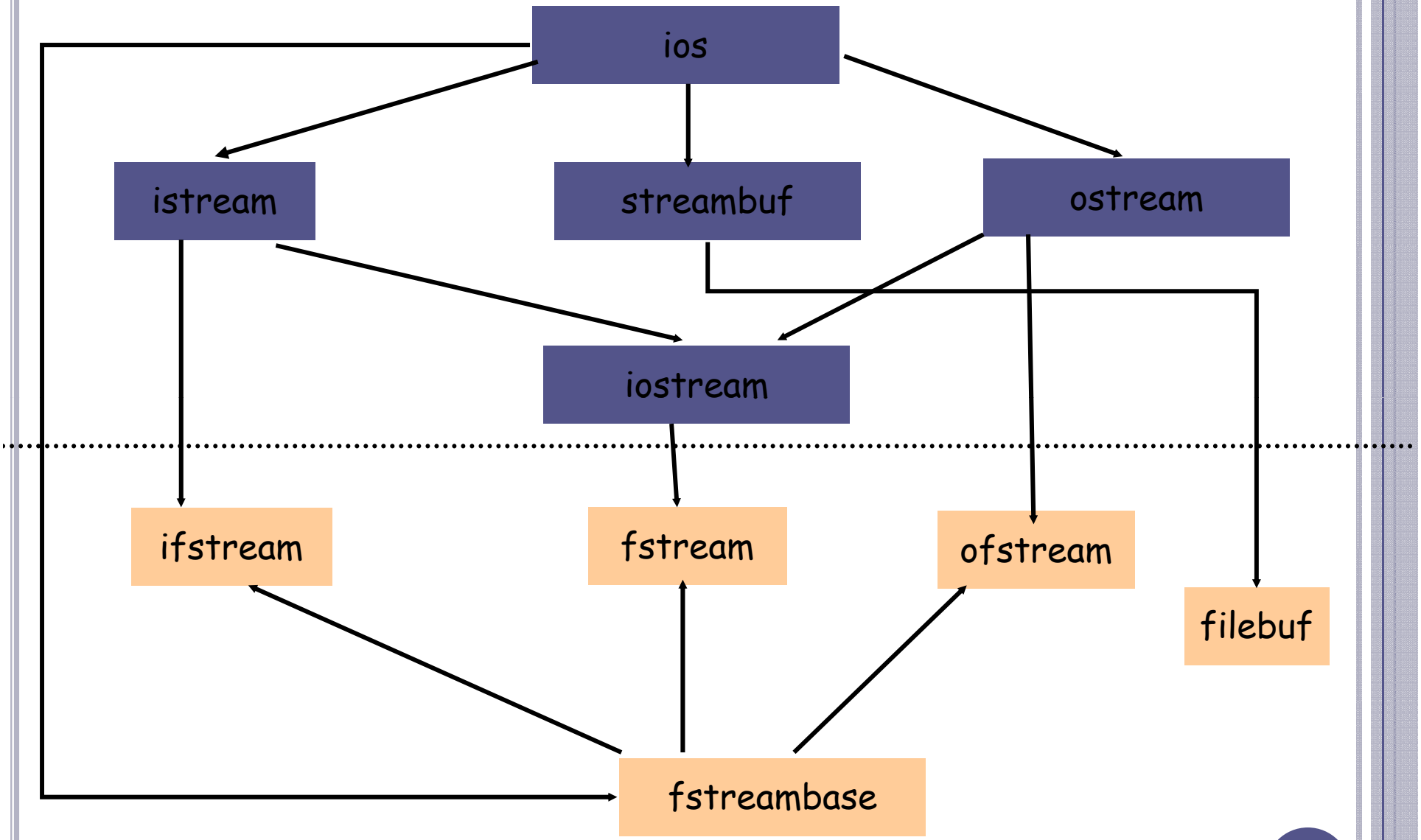
THE *FSTREAM.H* HEADER FILE

- **Streams act as an interface between files and programs.**
- **They represent as a sequence of bytes and deals with the flow of data.**

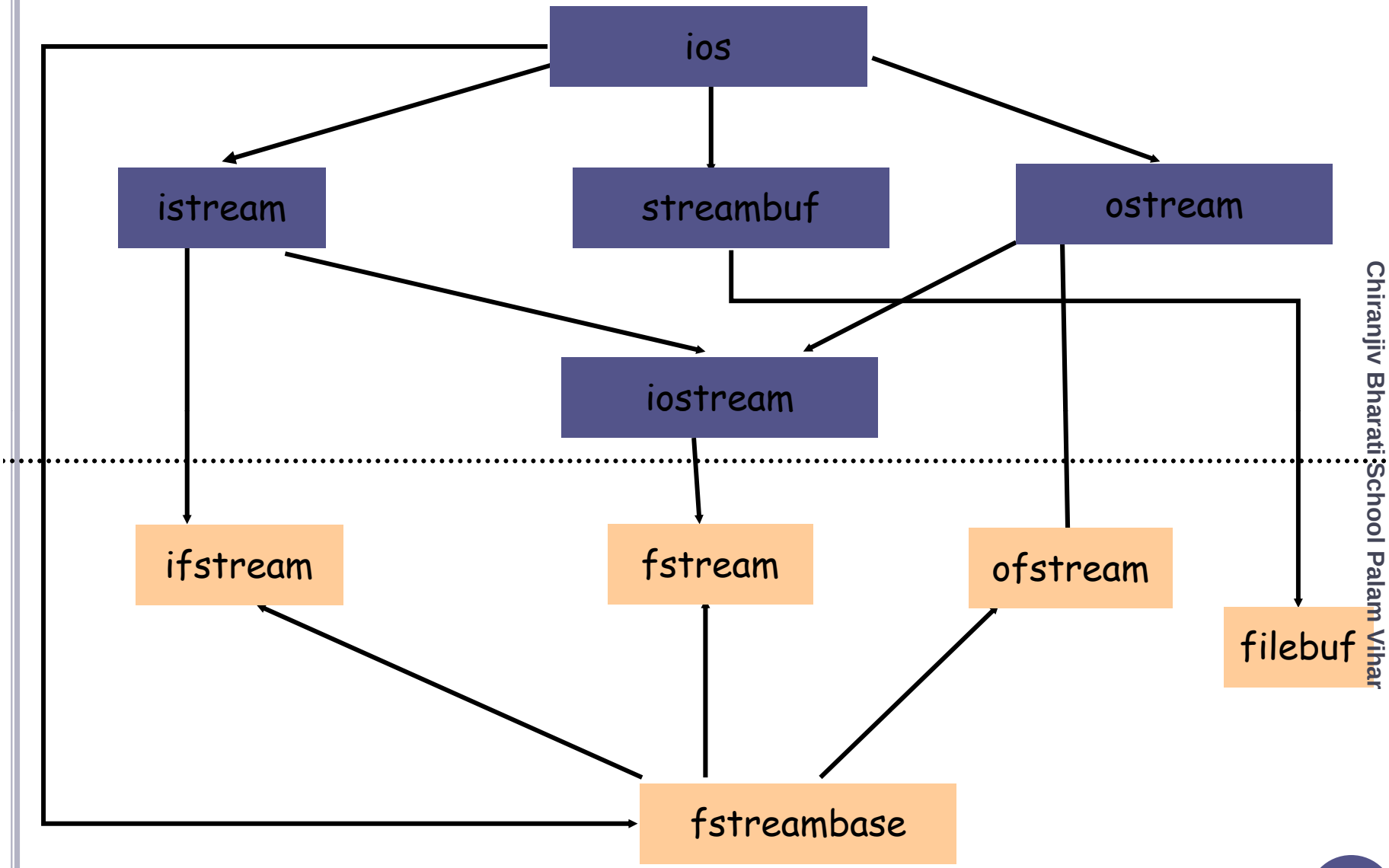
- **File → Program (Input stream) - reads**
- **Program → File (Output stream) – write**

- **Diagrammatically as shown in next slide**


Program
Input
Stream
Output
Stream
Files



" Classes for file stream operations "



" Classes for file stream operations "

I/O STREAMS

Stream	Description
cin	**Standard input stream**
cout	**Standard output stream**

File I/O Streams

Stream Classes required for File i/o :

- ifstream
- ofstream
- fstream

FSTREAM

- It supports files for simultaneous input and output operation
- **open()** is a member function of the class fstream
- fstream is derived from
  - iostream

FSTREAM

- Member functions of the class fstream
  - open
  - close
  - close all
  - seekg
  - seekp
  - tellg
  - tellp

File Handling Classes

- **When working with files in C++, the following classes can be used:**
  - **ofstream** – writing to a file
  - **ifstream** – reading from a file
  - **fstream** – reading / writing

OPENING & CLOSING A FILE

If we want to use a disk file, we need to decide the following thing about the file and its intended use:

- Suitable name for the file
- Structure for the file
- Purpose
- Opening method

OPENING A FILE

A file can be opened in two ways:-

- Using the constructor of the class.
- Using the member function **open()** of the class.

OPENING FILE USING CONSTRUCTOR

- Create a file stream object to manage the stream using the appropriate class.
- the class **ostream** is used to create the output stream and the class **istream** to create the input stream.
- Initialize the file object with the desired filename.

OPENING FILE USING CONSTRUCTOR - SYNTAX

```
ofstream outfile ( " result.doc" );
```

The above statement open a file named "result " for output.

```
ifstream infile ( " data.doc" );
```

The above statement open a file named "result " for input.

OPENING FILE USING CONSTRUCTOR

Data output to a file can also be performed in the same way that we did with cout

```
// writing on a text file
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream myfile ("example.txt");
  if (myfile.is_open())
  {
    myfile << "This is a line.\n";
    myfile << "This is another line.\n";
    myfile.close();
  }
  else cout << "Unable to open file";
  return 0;
}
```

```
[file example.txt]
This is a line.
This is another line.
```

Opening File using constructor

Data input from a file can also be performed in the same way that we did with cin

```
// reading a text file
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main () {
  string line;
  ifstream myfile ("example.txt");
  if (myfile.is_open())
  {
    while (! myfile.eof() )
    {
      getline (myfile,line);
      cout << line << endl;
    }
    myfile.close();
  }

  else cout << "Unable to open file";

  return 0;
}
```

```
This is a line.
This is another line.
```

OPENING FILE USING CONSTRUCTOR

```
#include <fstream.h>
Void main()
{
    char name[30];
    float cost;
    ofstream outfile("ITEM");
    cout<< " Enter item name";
    cin>> name;
    outfile << name << "\n";
    cout<< " Enter item cost";
    cin>> cost;
    outfile << cost<< "\n";
    outfile.close();
    ifstream infile( "ITEM");
    infile >> name;
    infile >> cost;
    cout << " item name" <<
    name <<"\n";
    cout << " item cost" << cost
    <<"\n";
    infile.close();
}
```

OPENING FILE USING CONSTRUCTOR

- When a file is opened for writing only, a new file is created if there is no file of that name.
- If a file by that name exist already, then its contents are deleted and the file is presented as a clean file.
- We shall discuss later how to open an existing file for updating it without losing its original contents.

OPENING FILE USING OPEN()

```
File-stream-class stream-object;
Stream-object .Open( " filename ");
```

Example:

```
ofstream outfile;
outfile .open() ( " result.doc" );
……..
……
outfile.close();
outfile.open("Data.txt");
……..
……
outfile.close();
```

THIS CODE CREATES A FILE CALLED EXAMPLE.TXT AND INSERTS A SENTENCE INTO IT IN THE SAME WAY WE ARE USED TO DO WITH COUT, BUT USING THE FILE STREAM MYFILE INSTEAD.

```
// basic file operations
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream myfile;
  myfile.open ("example.txt");
  myfile << "Writing this to a file.\n";
  myfile.close();
  return 0;
}
```

```
[file example.txt]
Writing this to a file
```

OPENING FILE USING OPEN()

```
Void main()
{
    ofstream fout;
    fout.open(" country");
    fout << " United states of America \n";
    fout << " United kingdom \n";
    fout << " south Korea \n";
    fout.close();

    fout.open(" Capital ");
    fout<< " Washington \n";
    fout<< " London \n";
    fout<< " Seoul \n";
    fout.close();

    char line [80];

    ifstream fin;
    fin.open(" country ");
    cout << " contents of country file" ;
    while (fin)
    {
      fin.getline ( line, 80);
      cout<< line;
    }
    fin.close();
    fin.open(" Capital " );
    cout << " contents of capita file" ;
    while (fin)
    {
      fin.getline ( line, 80);
      cout<< line;
    }
    fin.close();
}
```

OPENING FILE USING OPEN()

Contents of country file

united states of America

united kingdom

south Korea

Contents of capital file

Washington

London

seoul

OPENING MORE THEN ONE FILE SIMULTANEOUSLY

```
Void main()
{
    const int size=80;
    char line [ size ];
    ifstream fin1, fin2;
    fin1.open( " country ");
    fin2.open( " capital ");
    for ( int i=1; i<=10; i++)
    {
    if( fin1.eof ( ) !=0)
    {
    cout << " Exit from country";
    exit (1);
    }
    fin1.getline( line, size );
    cout << " capital of " << line ;

    if( fin2.eof ( ) !=0)
    {
    cout << " Exit from capital";
    exit (1);
    }
    fin2.getline( line, size );
    cout << line ;
    }
}
```

OPENING MORE THEN FILE SIMULTANEOUSLY

Output-

Capital of united states of America

Washington

Capital of united kingdom

London

Capital of south Korea

seoul

OPENING A FILE WITH DIFFERENT MODES

- **A file can be open by the method "open()" or immediately in the constructor (the natural and preferred way).**

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

- **filename – file to open (full path or local)**
- **mode – how to open (specifies the purpose for which the file is opened)**

Opening a File with different Modes

- ios::app – append
- ios::ate – open with marker at the end of the file
- ios::in / ios::out – (the defaults of ifstream and ofstream)
- ios:nocreate / ios::noreplace – open only if the file exists / doesn't exist
- ios::trunc – open an empty file
- ios::binary – open a binary file (default is textual)

Don't forget to close the file using the method "close()"

class	default mode parameter
ofstream	ios::out
ifstream	ios::in
fstream	ios::in | ios::out

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

FILE POINTERS AND THEIR MANIPULATIONS:

Each file has two associated pointers known as the file pointers.

One of them is called the input pointer or get pointer.

Other is called the output pointer or put pointer.

We can use these pointers to move through the files while reading or writing.

The input pointer is used for reading the contents of a given file location and the output pointer is used for writing to a given file location.

FUNCTIONS FOR MANIPULATION OF FILE POINTERS

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.

FUNCTIONS FOR MANIPULATION OF FILE POINTERS

infile.seekg(10);

- Moves the file pointer to the byte number 10.
- The bytes in a file are numbered beginning from zero.
- Thus, the pointer will be pointing to the 11th byte in the file.

Ofstream outfile;

Outfile.open(“ hello”, ios::app);

Int p= outfile.tellp();

SPECIFYING THE OFFSET :

The seek functions seekg() and seekp() can also be used with two arguments as follows:

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

The parameter offset represents the number of bytes the file pointer to be moved from the location specified by the parameter refposition. The refposition takes one of the following these constant defined in the ios class.

ios::beg	start of the file
ios::cur	current position of the pointer
ios::end	end of the file.

WRITE () AND READ () FUNCTION

infile. Read ((char *) & V, sizeof (v));

outfile. write ((char *) & V, sizeof (v));

WRITE () AND READ () FUNCTION

```
#include<iostream.h>
#include<fstream.h>
#include<iomanip.h>
int main()
{
float
    ht[4]={12.3,15.3,34.7,12.8};
ofstream outfile;
outfile.open("myfile",ios::binary
    );
outfile.write((char
    *)&ht,sizeof(ht));
outfile.close();
for(int i=0;i<4;i++)
{
ht[i]=0;
ifstream infile;
infile.open("myfile");
infile.read((char
    *)&ht,sizeof(ht));
for(i=0;i<4;i++)
{
cout.setf(ios::showpoint);
cout<<setw(10)<<setprecision
    (2)     <<ht[i];
}
Infile.close();
Return 0;
}
```

WRITING AND READING OBJECTS OF A CLASS :

So far we have done I/O of basic data types. Since the class objects are the central elements of C++ programming, it is quite natural that the language supports features for writing and reading from the disk files objects directly.

The binary input and output functions read() and write() are designed to do exactly this job.

The write() function is used to write the object of a class into the specified file and read() function is used to read the object of the class from the file.

Both these functions take two arguments:
1. address of object to be written.
2. size of the object.

The address of the object must be cast to the type pointer to char.

One important point to remember is that only data members are written to the disk file and the member functions are not.

WRITING AN OBJECT INTO THE FILE

```
#include
class Person
{
   private:
    char name[40];
    int age;
   public:
    void getData()
    {
    cout << “\n Enter name:”;
    cin >> name;
    cout << “\n Enter age:”;
    cin >> age;
    }
} ; // End of the class definition
```

```
void main()
{
    Person per ; // Define an object
     per.getData();

    ofstream outfile(“Person.txt”); // Open
    the file in output mode

    outfile.write((char*)&per, sizeof(per));
     // Write the object into the file
    }
```

READING AN OBJECT INTO THE FILE

```
class person
{
  private:
    char name[40];
    int age;
  public:
    void showData()
    {
    cout << "\n Name = " << name;
    cout << "\n Age = " << age;
    }
};
void main()
{
  person pers;
  ifstream infile;
  infile.open("Person.txt");
  infile.seekg(0, ios::end);
  int endposition = infile.tellg();
  int n = endposition / sizeof(person);

  cout << "\n There are " << n << "
  persons in file: ";

  cout << "\n Enter person number: ";
  cin >> n;

  int position = (n-1) *sizeof(person);

  infile.seekg (position);

 infile.read((char*)&pers,
  sizeof(pers));

  pers.showData();
}
```