Section 4-Lecture 10

Introduction

Files and Streams, Creating a Sequential Access File, Reading Data From A Sequential Access File, Updating Sequential Access Files, Random Access Files, Creating A Random Access File, Writing Data Randomly To a Random Access File, Reading Data Sequentially from a Random Access File. Stream Input/Output Classes and Objects, Stream Output, Stream Input, Unformatted I/O (with read and write), Stream Manipulators, Stream Format States, Stream Error States.

Introduction

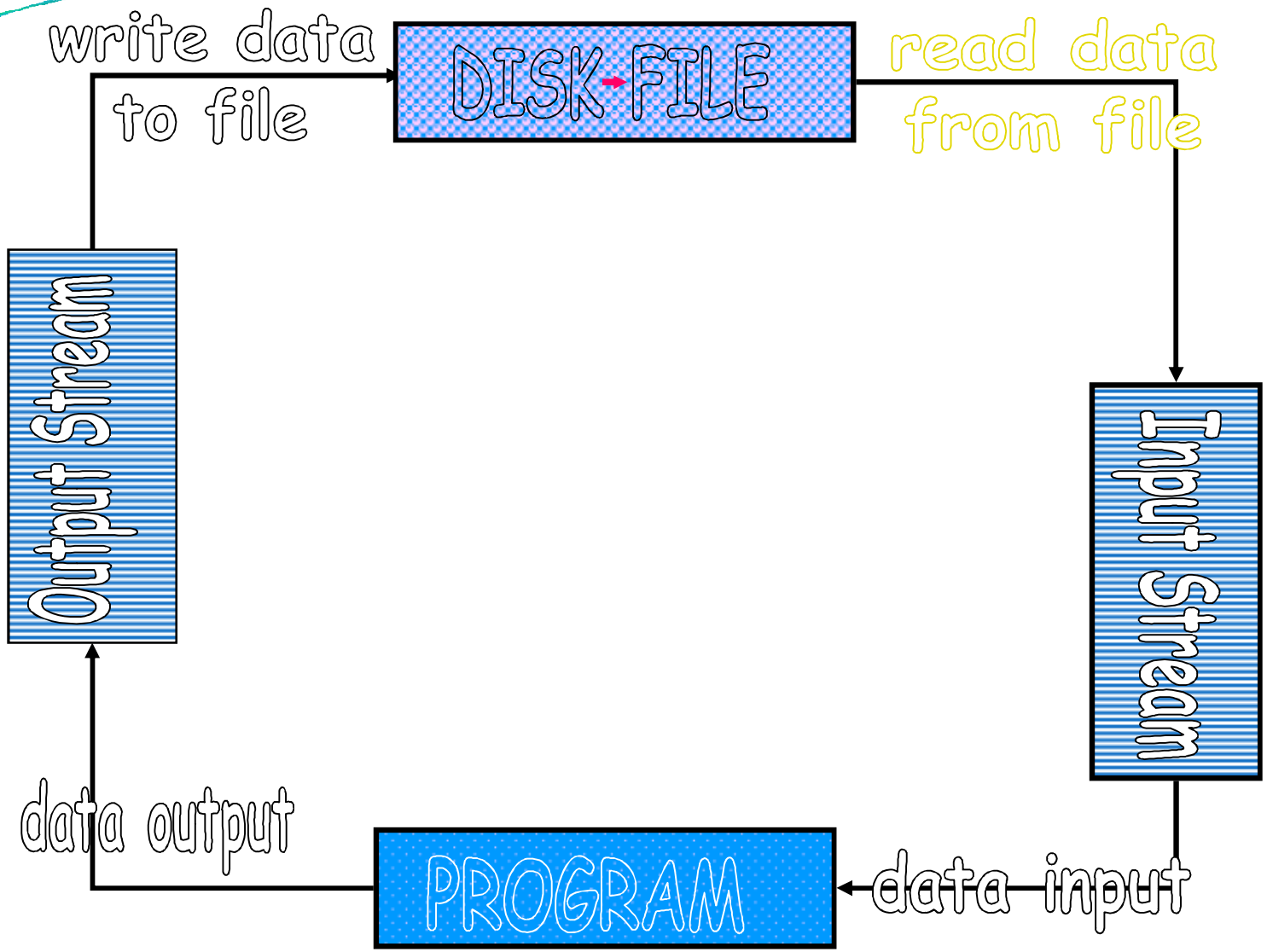
- Computer programs are associated to work with files as it helps in storing data & information permanently.
- File - itself a bunch of bytes stored on some storage devices.
- In C++ this is achieved through a component header file called *fstream.h*
- The I/O library manages two aspects- as interface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

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.
- Every stream is associated with a class having member functions and operations for a particular kind of data flow.

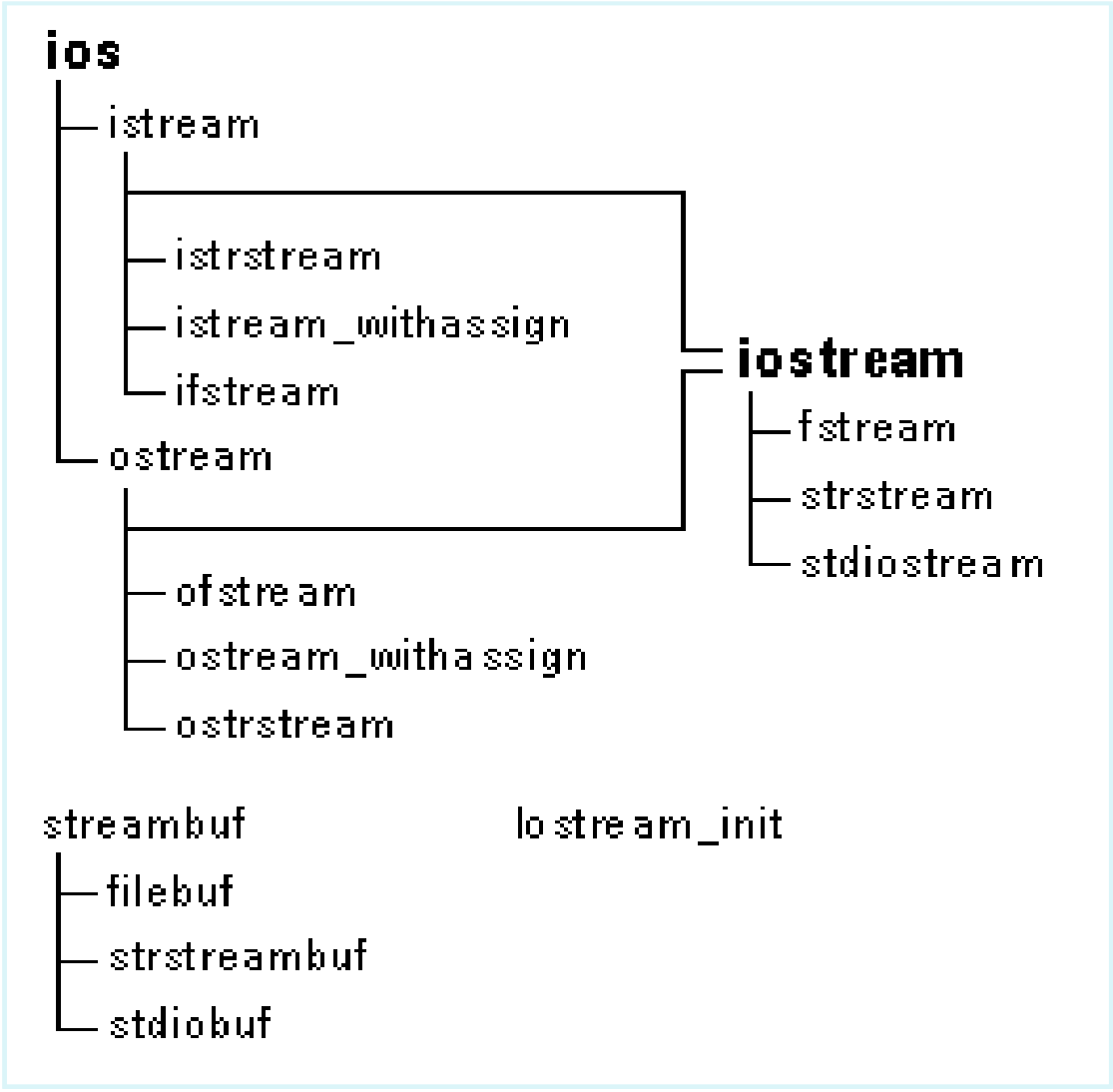
- File → Program (Input stream) - reads
- Program → File (Output stream) - write

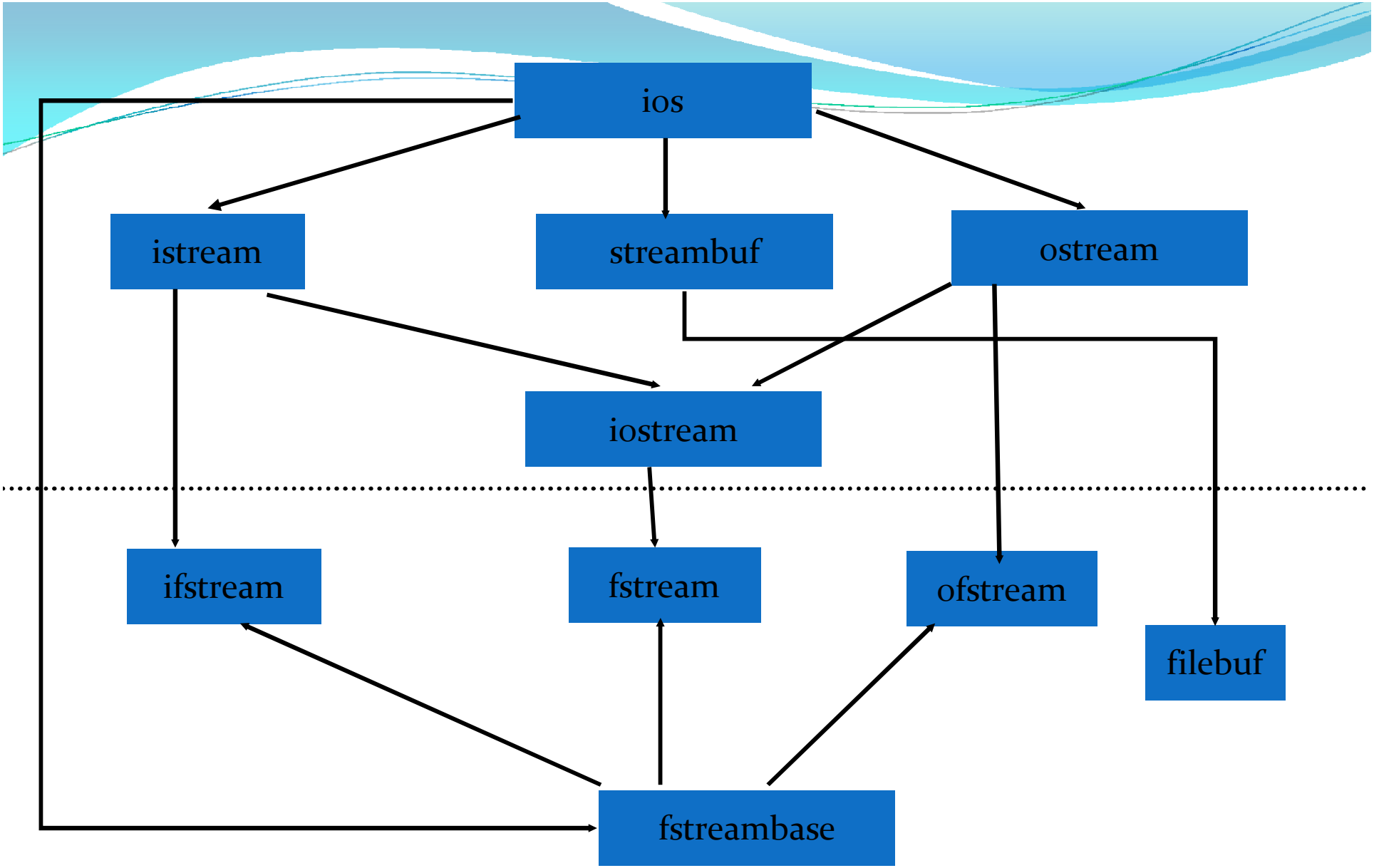
- All designed into fstream.h and hence needs to be included in all file handling programs.
- Diagrammatically as shown in next slide



File Handling Classes

Hierarchy Diagram





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

```
// Initial experience reading and writing files
#include <fstream.h>
#include <iostream.h>
#include <stdlib.h>
int main()
{  ifstream in_stream;
  ofstream out_stream;
  int num;
  in_stream.open("numbers.dat");
  if (in_stream.fail()) { cout << "Input file could not be opened.\n";
        exit(1);  }
  out_stream.open("squares.dat");
  if (out_stream.fail()) {  cout <<"Output file could not opened.\n";
        exit(1);  }
  in_stream >> num;
  out_stream << "The square of " << num << " is " <<num * num;
  in_stream.close();
  out_stream.close();
}
```

File Handling Classes

- **When working with files in C++, the following classes can be used:**
  - **ofstream** – writing to a file
  - **ifstream** – reading for a file
  - **fstream** – reading / writing
- **What does it all have to do with cout?**
  - When ever we include **<iostream.h>**, an **ostream** object, pointing to **stdout** is automatically defined – this object is **cout**.
- **ofstream** inherits from the class **ostream** (standard output class).
- **ostream** overloaded the operator >> for standard output….thus an **ofstream** object can use methods and operators defined in **ostream**.

Opening & Closing a File

- 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 – purpose for which file is opened.
  - 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

- Each one of the open() member functions of the classes ofstream, ifstream and fstream has a default mode that is used if the file is opened without a second argument:

  class default mode parameter
- Ofstream ios::out
- Ifstream ios::in
- Fstream ios::in | ios::out

Stream state member functions

- In C++, file stream classes inherit a stream state member from the ios class, which gives out the information regarding the status of the stream.

For e.g.:bad()Returns true if a reading or writing operation fails.

For example in the case that we try to write to a file that is not open for writing or if the device where we try to write has no space left.

fail()Returns true in the same cases as bad(), but also in the case that a format error happens, like when an alphabetical character is extracted when we are trying to read an integer number.

eof()Returns true if a file open for reading has reached the end.

good()It is the most generic state flag: it returns false in the same cases in which calling any of the previous functions would return true.

Detecting end of file

```
int main()
{
Int SIZE=80;
Char line[SIZE];
Ifstream fin;
Fin.open("country");
For(i=0;i<=10;i++)
{
if(fin.eof()!=0)     //fin return 0 on reaching end of file
{ exit(1);
fin.getline(line,SIZE);
}  //end
```

Output:-
India
UK
London

Note:eof() is a function of ios class.

File operations

The following member functions are used for reading and writing a character from a specified file.

get()- is used to read an alphanumeric character from a text file.

-Member of istream.h

-Read()-used to handle a single character for input from binary file.

--Member of istream.h

-write()-used to handle a single character for output to binary file.

-Member of ostream.h

put()- is used to write a character to a text file

- Member of ostream.h

Text file

- Text file streams are those where we do not include the ios::binary flag in their opening mode. These files are designed to store text and thus all values that we input or output from/to them can suffer some formatting transformations, which do not necessarily correspond to their literal binary value.
- Data output operations on text files are performed in the same way we operated 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.
```

reading a text file

```
#include <iostream.h>
#include <fstream.h>
 #include <string.h>
 using namespace std;
 int main ()
{
 string line;
 ifstream myfile ("example.txt");
 if (myfile.is_open())
{
 while ( myfile.good() )
{
getline (myfile,line);
cout << line << endl;
}
 myfile.close();
 }
 else cout << "Unable to open file";
 return 0;
}
```

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

get and put stream pointers

- All i/o streams objects have, at least, one internal stream pointer:

  ifstream, like istream, has a pointer known as the *get pointer* that points to the element to be read in the next input operation.

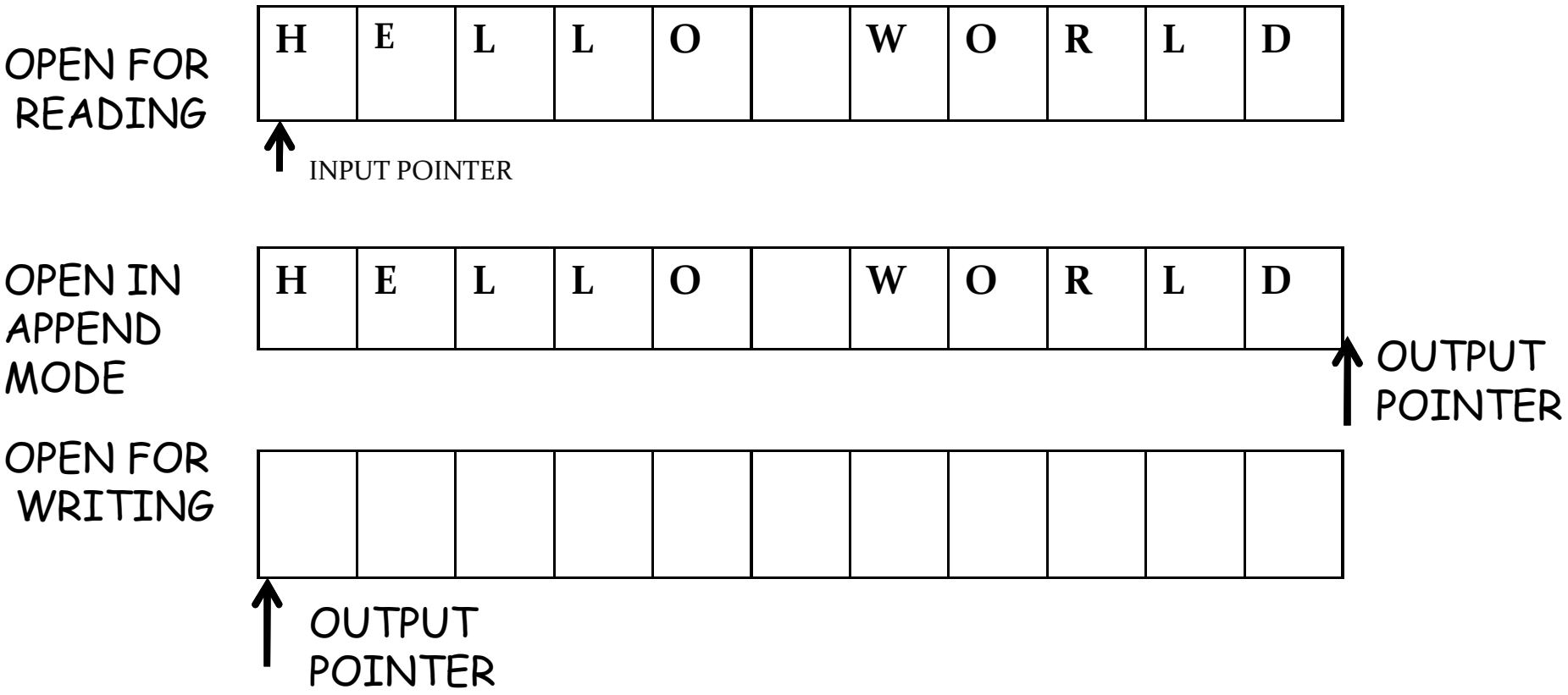
  ofstream, like ostream, has a pointer known as the *put pointer* that points to the location where the next element has to be written.

  Finally, fstream, inherits both, the get and the put pointers,
  from iostream (which is itself derived from bothistream and ostream).

Stream File pointers and their manipulator(Default)

1 Input pointer(get pointer)

2 Output pointer(put pointer)



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.

For example- infile.seekg(10);

Moves the file pointer to byte no. 10.and the pointer will be now pointed to the 11th byte in a file,becuse Bytes in a file are numbered beginning from zero.

What is the output:

```
Ofstream fileout;
Fileout.open("hello",ios::app);
Int p=fileout.tellp();
```

Specify the offset

- Seekg(offset,reposition);
- Seekp(offset,reposition);

  offset represents no.of bytes the file pointer is to be moved from the location specified by the parameter "reposition"

Reposition takes 3 constants defined in ios class:

- Ios::beg start of file
- Ios::cur current position of the pointer
- Ios::end end of the file

Obtaining file size

```
#include <iostream>
 #include <fstream>
using namespace std;
 int main ()
 {
 long begin,end;
 ifstream myfile ("example.txt");
 begin = myfile.tellg();
 myfile.seekg (0, ios::end);          //go to end of file
end = myfile.tellg();
 myfile.close();
 cout << "size is: " << (end-begin) << " bytes.\n";
return 0;
}
```

size is: 40 bytes.

Bytes in a file are numbered begning from zero

Binary files

- In binary files, to input and output data with the extraction and insertion operators (<< and >>) and functions like getline is not efficient, since we do not need to format any data, and data may not use the separation codes used by text files to separate elements (like space, newline, etc...).

File streams include two member functions specifically designed to input and output binary data sequentially: write and read. The first one (write) is a member function of ostream inherited by ofstream. And read is a member function of istream that is inherited by ifstream. Objects of class fstream have both members. Their prototypes are:

```
write ( memory_block, size );
read ( memory_block, size );
```

Where memory_block is of type "pointer to char" (char*), and represents the address of an array of bytes where the read data elements are stored or from where the data elements to be written are taken. The size parameter is an integer value that specifies the number of characters to be read or written from/to the memory block.

// reading a complete binary file

```
#include <iostream>
 #include <fstream>
using namespace std;
 ifstream::pos_type size;
 char * memblock;
int main ()
 {
 ifstream file ("example.bin", ios::in|ios::binary|ios::ate);
 if (file.is_open())
 { size = file.tellg();
 memblock = new char [size];
 file.seekg (0, ios::beg);
 file.read (memblock, size);
file.close();
cout << "the complete file content is in memory";
delete[] memblock;
}
 else
cout << "Unable to open file";
return 0;
}
```

the complete file content is in memory

Reading /Writing from/to Binary Files

- **To write n bytes:**
  - **write (const unsigned char* buffer, int n);**
- **To read n bytes** (to a pre-allocated buffer)**:**
  - **read (unsighed char* buffer, int num)**

```
#include <fstream.h>
main()
{
     int array[] = {10,23,3,7,9,11,253};
     ofstream OutBinaryFile("my_b_file.txt“, ios::out |
ios::binary);
     OutBinaryFile.write((char*) array, sizeof(array));

    OutBinaryFile.close();
}
```