Unit-V
Files and Streams

Syllabus:

Files and Streams – Opening and Closing a file – file modes – file pointers and their manipulation, sequential access to a file – random access to a file- Reading and Writing files.

Introduction

- All programs we looked earlier:
  - input data from the keyboard.
  - output data to the screen.
- Output would be lost as soon as we exit from the program.
- How do we store data permanently?
  - We can use secondary storage device.
  - **Data is packaged up on the storage device as data structures called files.**


Streams

Streams

- **Stream:[Standard]**
  - a channel or medium where data are passed to receivers from senders.
  - Standard Streams use #include <iostream.h>
- **Output Stream:**
  - **a channel where data are sent out to a receiver.**
  - **cout;** the standard output stream (to monitor).
  - the monitor is a *destination* device.
  - the extractor operator >> extracts, gets, or receives the next element from the stream.
- **Input Stream:**
  - **a channel where data are received from a sender.**
  - **cin;** the standard input stream (from the keyboard).
  - the keyboard is a *source* device.
  - the inserter operator << inserts, puts, or sends the next element to the stream.
  - Opening & closing of the standard streams occur automatically when the program begins & ends.

Streams

- ***stream*** - a sequence of characters
  - **Standard Streams: [interactive (iostream)]**
    - cin - input stream associated with keyboard.
    - cout - output stream associated with display.
  - **File Streams: (fstream)**
  - **data structures that are stored separately from the program (using auxiliary memory).**
  - **streams must be connected to these files.**
  - **#include <fstream.h> creates two new classes**
    - **ofstream (output file stream): defines new input stream (normally associated with a file).**
      extracts, receives, or gets data from the file
    - **ifstream (input file stream): defines new output stream (normally associated with a file).**
      inserts, sends, or puts data to the file
- Stream is the basic concepts which can be attached to files, strings, console and other devices.
- User can also create their own stream to cater specific device or user defined class.

Streams Usage

- We've used streams already
- cin
  - Input from stream object connected to keyboard
- cout
  - Output to stream object connected to screen
- Can define other streams
  - To or from files
  - Used similarly as cin, cout

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.

Streams

- The stream is an object flowing from one place to another.
- In c++,a stream is used to refer to the flow of data from a particular device to the program's variable.
- Stream is a series of bytes, which act either as a source from which input data can be extracted or as a destination to which the output can be sent.
- The source stream provides data to the program called the input stream and the destination stream that receives data from the program is called the output stream.
- The device refer to files,keyborad,memory arrays, console and so on.
- **Predefined Console Stream are**
- 1) cin :standard input (usually keyboard) corresponding to stdio in C
- 2) cout :standard output (usually screen) corresponding to stdout in C
- 3) cerr :standard error output (usually screen) corresponding to stderr in C
- 4) clog : A fully buffered version of cerr (No C equivalent)

Streams

- Different streams are used to represent different kinds of data flow.
- Each stream is associated with particular class which contains member functions and definitions for dealing with that particular kind of data flow.
- The sequence of input bytes are called as Input Stream.
- The sequence of output bytes are called as Output Stream.
- The cin object with >>operator is used for getting input.
- The cout object with <<operator is used for displaying output.
- In C++, the number of base and derived classes are available in Stream I/O classes.

Files

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

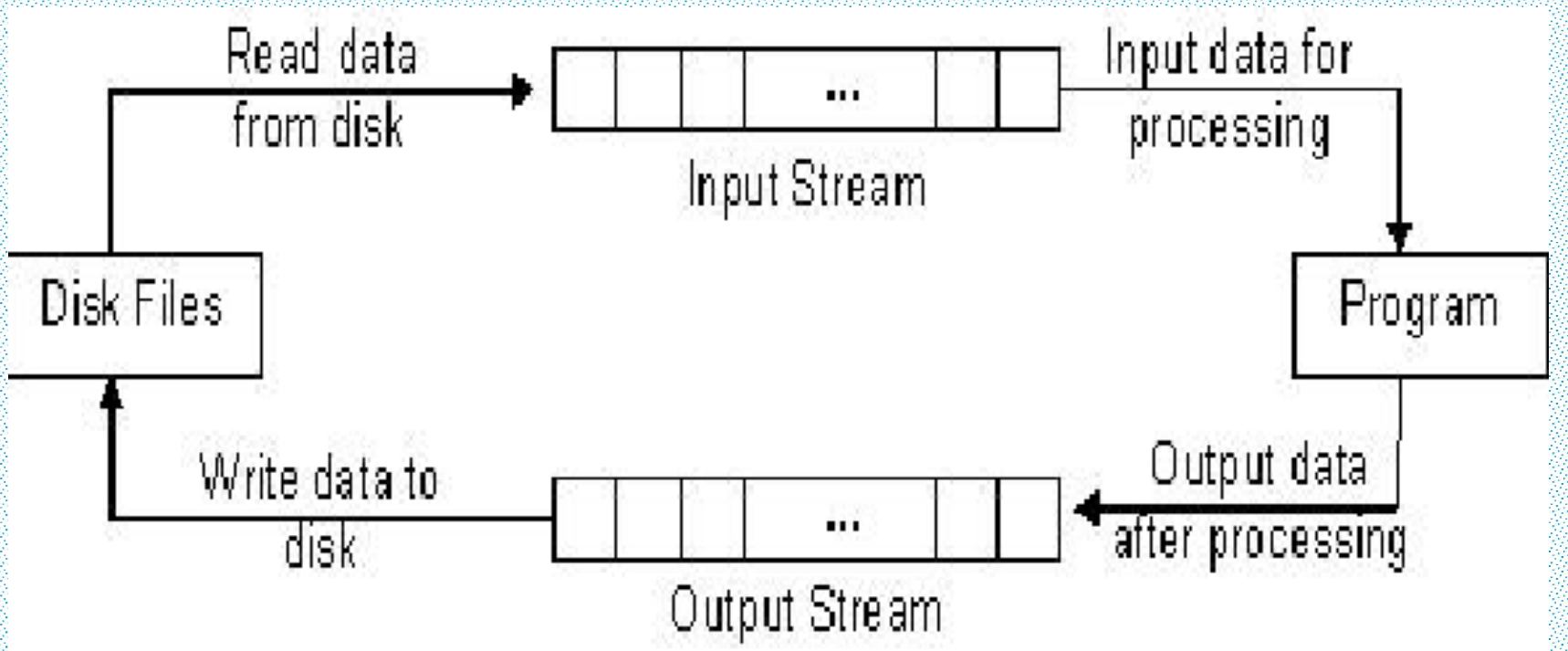
File

- A file is a **collection of related information** defined by its creator.
- There are three classes for handling files
  - ifstream ----for handling input files
  - ofstream ---for handling output files.
  - fstream ---for handling files on which both input and output can be performed
- It is necessary to include the header file fstream.h
- Manipulation of a file involves the following steps:
  - **Name the file** on the disk
  - **Open the file** to get the file pointer
  - **Process the file(read/write)**
  - **Check for error** while processing
  - **Close the file** after its complete usage

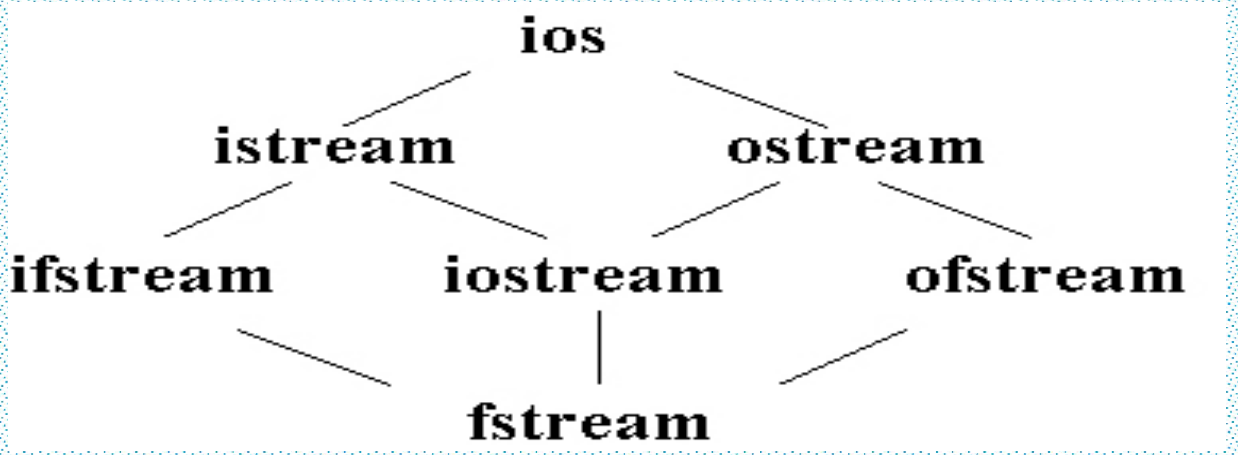
C++ Files and Streams

- C++ views each files as a sequence of bytes.
- Each file ends with an ***end-of-file*** marker.
- When a file is ***opened***, an object is created and a stream is associated with the object.
- To perform file processing in C++, the header files **<iostream.h>** and **<fstream.h>** must be included.
- <fstream.> includes <ifstream> and <ofstream>

File Input and Output Streams



Classes for Stream I/O in C++



- ios is the base class.
- istream and ostream inherit from ios
- ifstream inherits from istream (and ios)
- ofstream inherits from ostream (and ios)
- iostream inherits from istream and ostream (& ios)
- fstream inherits from ifstream, iostream, and ofstream

Classes for Stream I/O in C++

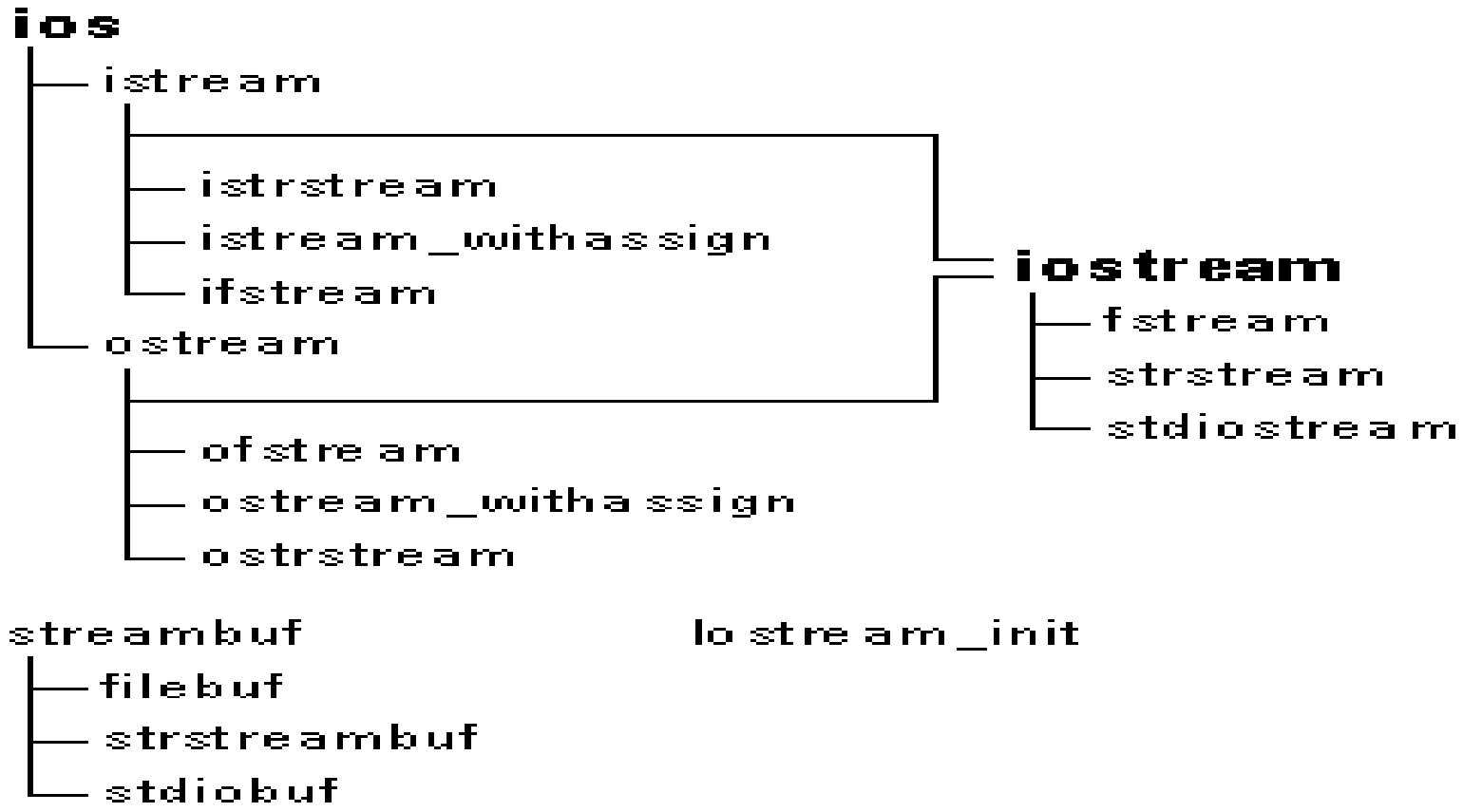
When working with files in C++, the following classes can be used:
ofstream – writing to a file
ifstream – reading for a file
fstream – reading / writing

Whenever we include <iostream.h>, an ostream object, is automatically defined – this object is cout.

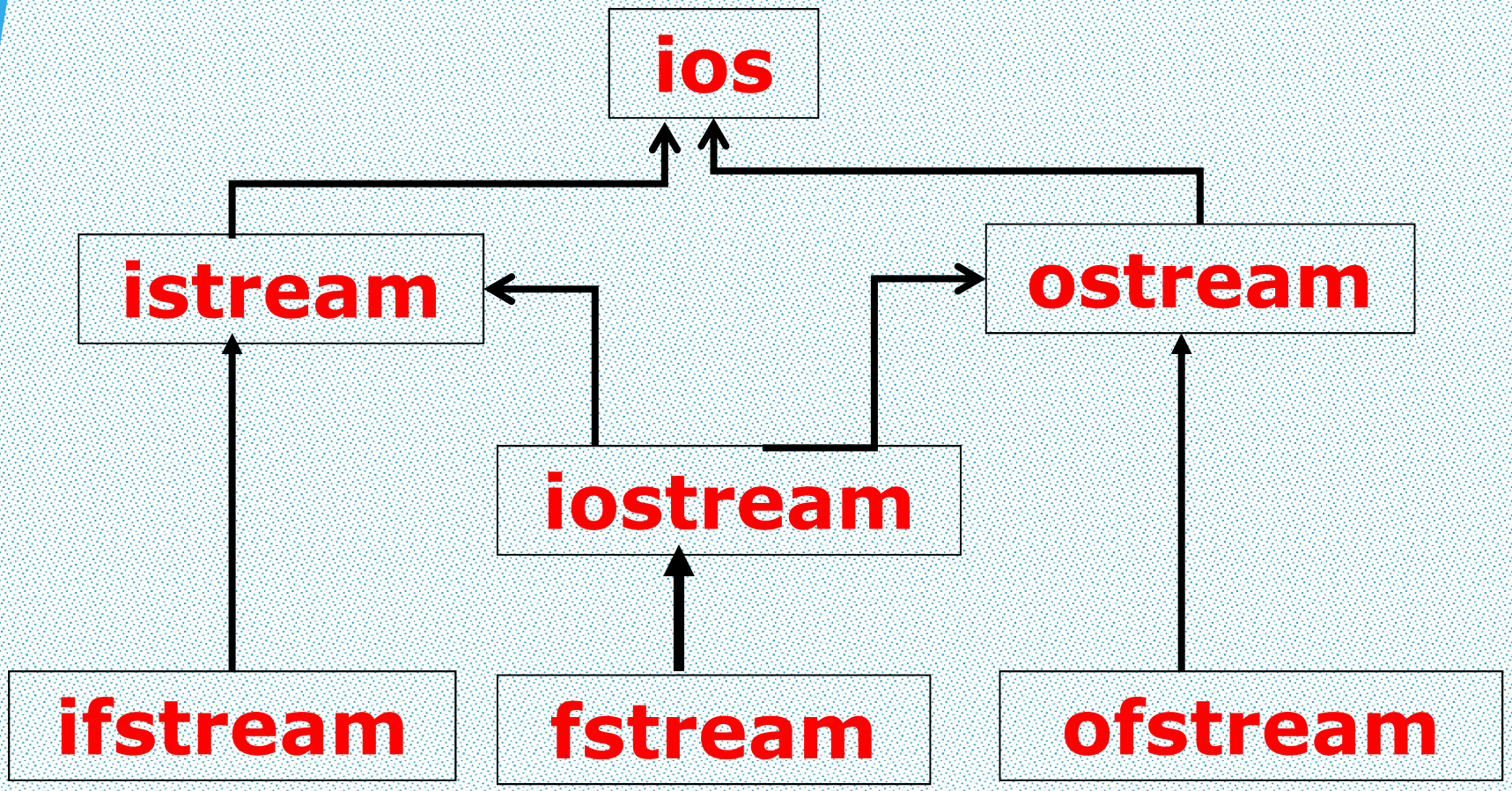
- ofstream inherits from the class ostream (standard output class).
- ostream overloaded the operator >> for standard output.
- Thus an ofstream object can use methods and operators defined in ostream.

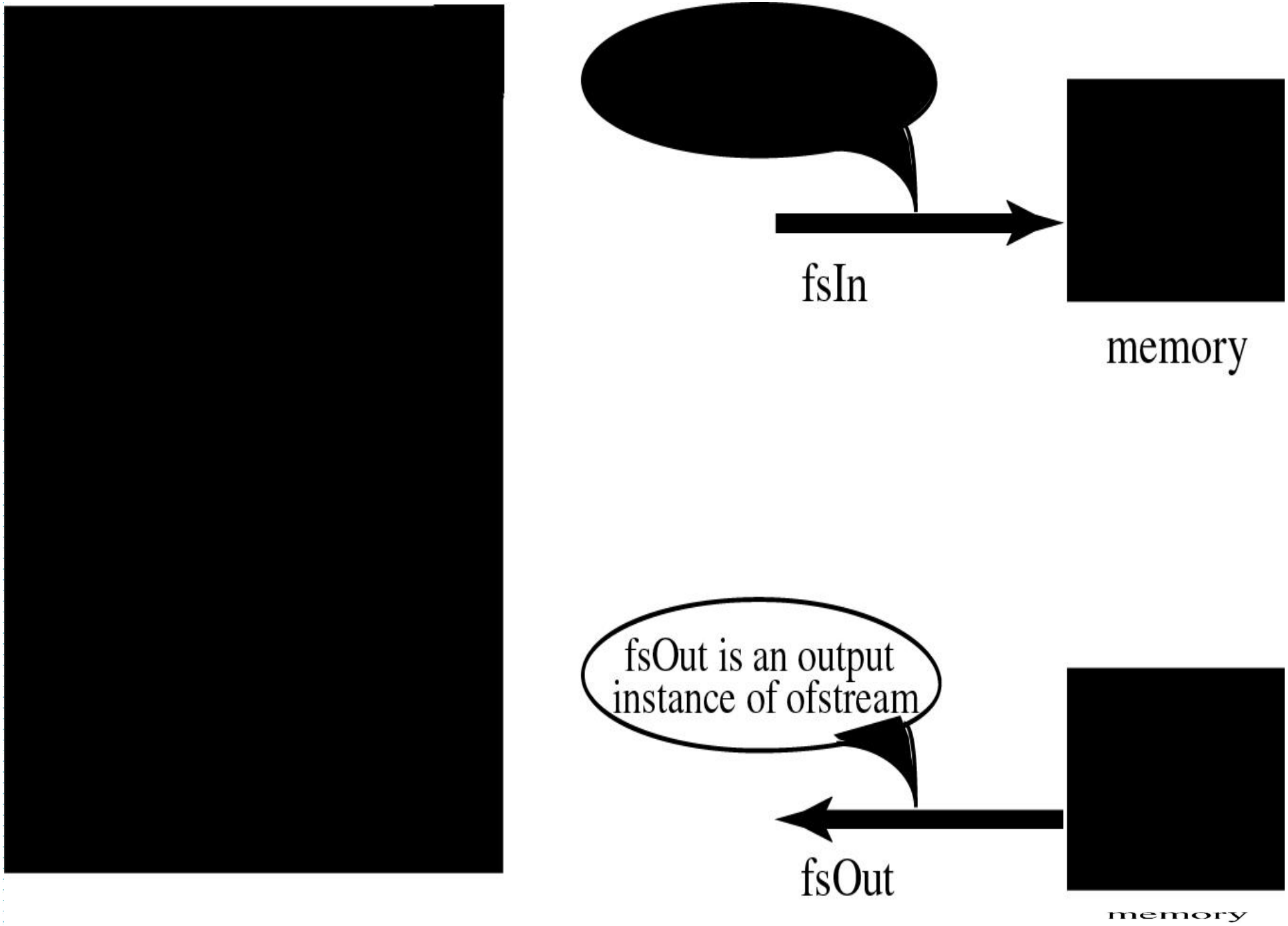
File Handling Classes

Hierarchy Diagram



Classes for Stream I/O in C++




fsIn
memory
fsOut is an output
instance of ofstream
fsOut
memory

Opening and Closing a File

Opening a File

- Use open function or include file name when declaring variable:
  - ifstream inobj1;
  - inobj1.open(“in1.dat”)
  - ifstream inobj2(“in2.dat”);
- To check if file successfully opened check object in condition:
  - if (!inobj1)
  - cout << “Unable to open file in1.dat” << endl;

Opening output Files

ofstream outClientFile(“clients.dat”, ios:out)

OR

ofstream outClientFile;

outClientFile.open(“clients.dat”, ios:out)

Various Syntaxes in File Handling

1: To access file handling routines:

```
#include <fstream.h>
```

2: To declare variables that can be used to access file:

```
ifstream in_stream;
ofstream out_stream;
```

3: To connect your program's variable (its internal name) to an external file (i.e., on the Unix file system):

```
in_stream.open("infile.dat");
out_stream.open("outfile.dat");
```

4: To see if the file opened successfully:

```
if (in_stream.fail())
{   cout << "Input file open failed\n";
    exit(1); // requires <stdlib.h>}
```

Various Syntaxes in File Handling

5: To get data from a file (one option), must declare a variable to hold the data and then read it using the extraction operator:

```
int num;
in_stream >> num;
```

[Compare: cin >> num;]

6: To put data into a file, use insertion operator:

```
out_stream << num;
```

[Compare: cout << num;]

NOTE: Streams are sequential – data is read and written in order – generally can't back up.

7: When done with the file:

```
in_stream.close();
out_stream.close();
```

Opening a File

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

void ofstream / ifstream::open(const char* filename, int mode);

- filename – file to open (full path or local)
- mode – how to open (1 or more of following – using |)
- The Open() function is used to open different files that uses the same stream object.
- Syntax : **open(“filename”, file mode);**
- Don’t forget to close the file using the method “close()”

File Opening Modes

File Mode	Description
ios :: in	Open file for reading
ios :: out	Open file for writing
ios :: app	Open file for reading and/or writing the data at the end of file (append)
ios :: ate	Erase the file before reading or writing(Truncated)
ios :: nocreate	Error when opening, if file does not already exists
ios :: no replace	Error when opening, if file does not already exists, unless ate or app is set
ios :: binary	Open file in binary mode (not text mode)

File Opening Modes

```
// File open for writing
#include <fstream>
int main(void)
{
      ofstream outFile("file1.txt", ios::out);
      outFile << "That's new!\n";
      outFile.close();
        return 0;
}
```

If you want to set more than one open mode, just use the OR operator- |. This way:

ios::ate | ios::binary

Closing a File

Closing a File

- **The file is closed implicitly when a destructor for the corresponding object is called**

  OR

- by using member function *close:*

  outClientFile.close();

- To close a file, the member function close() must be used. It takes no parameters and returns no value.

  Syntax : close();

- Use close() on object to close connection to file:

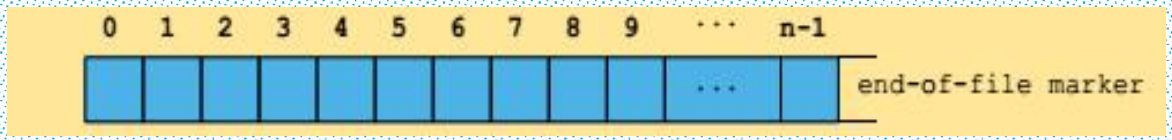
  ifstream in("in.dat");

  …

  in.close();

Detecting End-Of-File:

- It is necessary for preventing any attempt to read data from the file.
  - Syntax : eof()



File Modes

File Modes

- C++ provides a mechanism of opening a file in different modes in which case the second parameter must be explicitly passed.the syntax is as follows

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

- List of modes:

Mode value	Effect on the mode
ios::in	Open for reading
ios::out	Open for writing
ios::ate	Go to the end of file at opening time
ios::app	Append mode
ios::trun	If the file already exists, its contents will be deleted(Truncate). This is default by using ios::out.
ios::nocreate	If the file does not exist, this flag will cause the open function to fail. (The file will not be created)
ios::noreplace	If the file already exists, this flag will cause the open function to fail. (The existing file will not be opened)
ios::binary	Open as a binary file.

File Pointers and their manipulation

File Pointers

- **Each file has two associated pointers** known as the file pointers.
- One of them is called the input pointer or get pointer.
- The get pointer specifies a location from which the current reading operation is initiated.
- Other is called the output pointer or put pointer.
- The put pointer specifies a location from where the current writing operation is initiated.
- **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.**

File Pointers

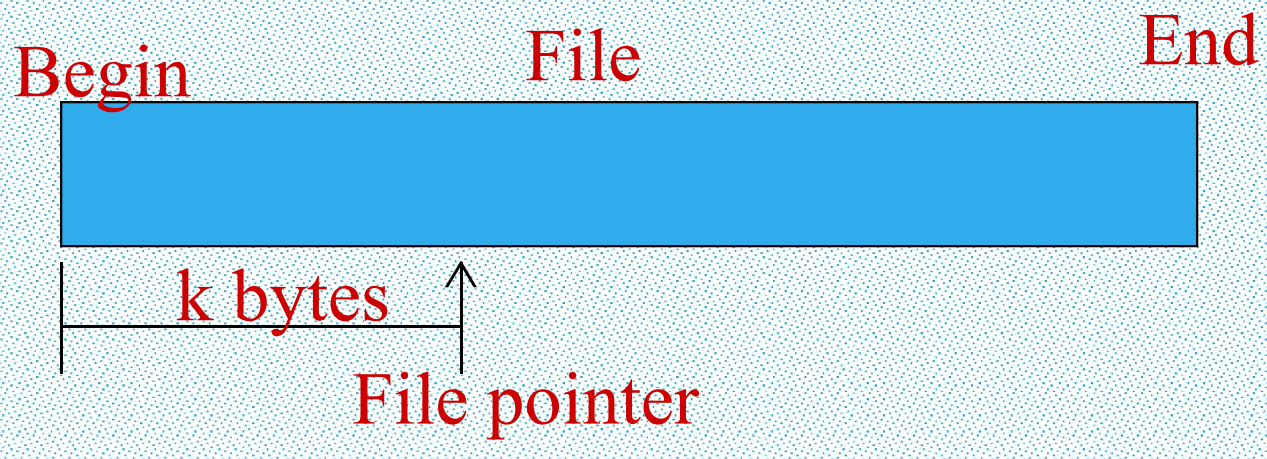
- seekg ( ) – (seek get) Moves get(input) file pointer to a specific location.
- seekp ( ) - (seek put) Moves put(output) file pointer to a specific location.
- tellg ( ) – (tell get) **Returns**(gives) the current position of the get(input) pointer.
- tellp ( ) – (tell put) **Returns**(gives) the current position of the put(output) pointer .

The seekg() function with one argument

seekg() function :

- With one argument :

  seekg(k) where k is absolute position from the beginning. The start of the file is byte 0



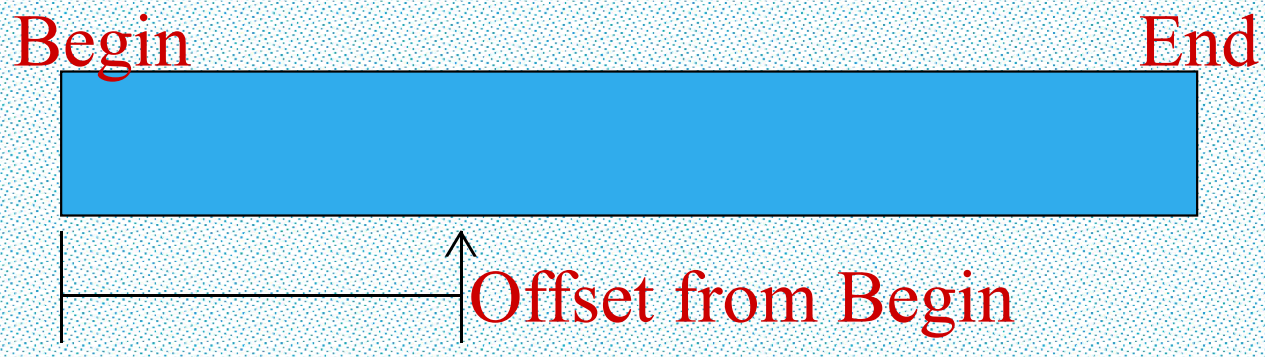
The seekg() function with two argument

seekg() function :

With two arguments :

the first argument represents an offset from a particular location in the file.

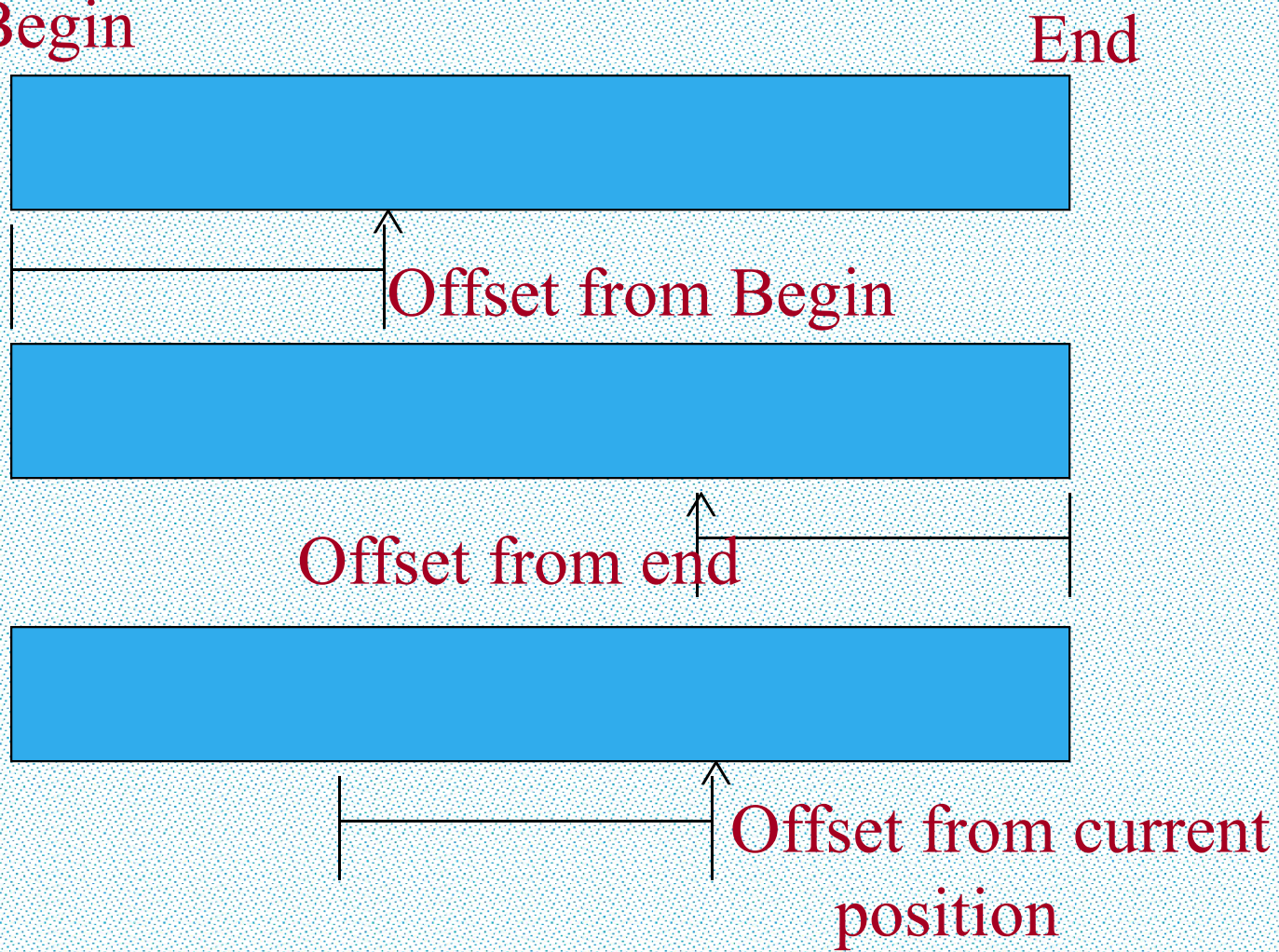
the second specifies the location from which the offset is measured.



The seekg() function with two argument

seekg() function :

With two arguments :



```
#include <fstream.h>
#include <conio.h>
#include <stdio.h>
void main()
{
  //clrscr();
  char c,d,ans;
  char str[80];
  ofstream outfl("try.txt"),out("cod.dat");
  ifstream infl;
  do
  {   cout<<"please give the string : ";
      gets(str);
      outfl<<str;
      cout <<"do you want to write  more...<y/n> : ";
      ans=getch();
  }
  while(ans=='y');
  outfl<<'\0';
  outfl.close();
```

```
//clrscr();
  getch();
cout <<"reading from created file \n";
  infl.open("try.txt");
  out.open("cod.dat");
  //*********************************

    c=infl.get();
  do
  {  d=c+1;
     cout<<c<<d<<'\n';
     out.put(d);
     c= infl.get();
  }
  while (c!='\0');
  out<<'\0';
  infl.close();
  outfl.close();
  getch();
  //*********************************
  }
```

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.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.

File Pointers

- **inClientFile.seekg(0)** - repositions the file get pointer to the beginning of the file.
- **inClientFile.seekg(n, ios:beg)** - repositions the file get pointer to the n-th byte of the file.
- **inClientFile.seekg(m, ios:end)** -repositions the file get pointer to the m-th byte **from the end of file.**
- **nClientFile.seekg(0, ios:end)** - repositions the file get pointer **to the end of the file.**
- **The same operations can be performed with <ostream> function member *seekp*.**

Member functions tellg() and tellp()

- Member functions **tellg** and **tellp** are provided to return the current locations of the get and put pointers, respectively.

long location = inClientFile.tellg();

- To move the pointer relative to the current location use ios:cur
- inClientFile.seekg(n, ios:cur) - moves the file get pointer n bytes forward.

File Types

Types of files

- **Sequential Access File:**

  A sequential file has to be accessed sequentially; to access the particular data in the file all the preceding data items have to be read and discarded. For example a file on a tape must be accessed sequentially.

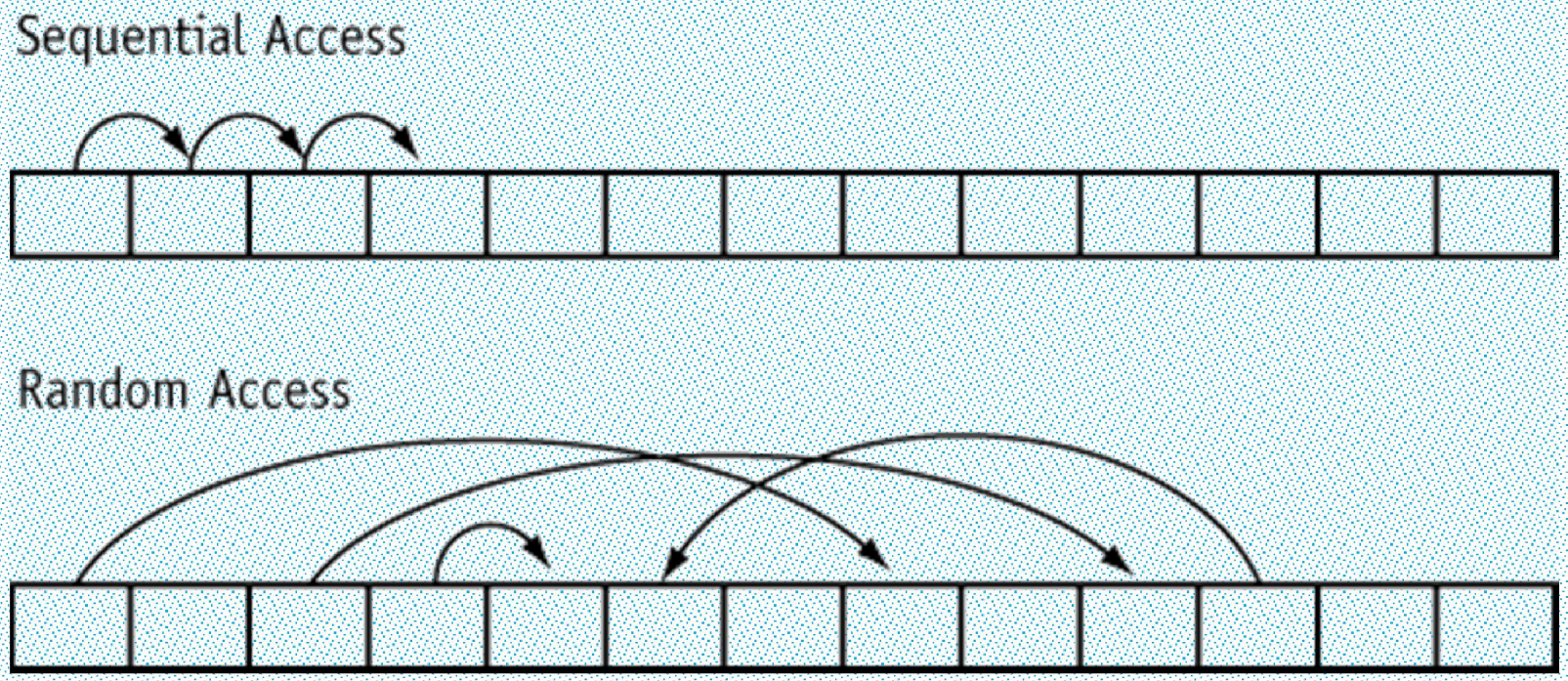
- **Random(Direct) Access File:**

  A random file allows access to the specific data without the need for accessing its preceding data items. However, it can be accessed sequentially . For example, a file on a hard disk or floppy disk can be accessed either sequentially or randomly.

- **Binary Files.**

  Files containing integers in binary and other data in ASCII format.

Types of files



Sequential Access to a File

Sequential Access to a File

- A **sequential access file** is often called a **text file**
- **Bit** - smallest data item
  - value of **0** or **1**
- **Byte** – 8 bits
  - used to store a character
    - Decimal digits, letters, and special symbols
- **Field** - group of characters conveying meaning
  - Example: your name
- **Record** – group of related fields
  - Represented a **struct** or a **class**
  - Example: In a payroll system, a record for a particular employee that contained his/her identification number, name, address, etc.
- **File** – group of related records
  - Example: payroll file
- **Database** – group of related files

Sequential Access to a File

- To open file, create objects
- Creates "line of communication" from object to file–Classes
  - ifstream (input only)
  - ofstream (output only)
  - fstream (I/O)
- Constructors take file name and file-open mode
  - ofstreamoutClientFile("filename",fileOpenMode);
- To attach a file later
  - OfstreamoutClientFile;
  - outClientFile.open("filename",fileOpenMode);

Sequential Access to a File

```
// 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();
}
```

Sequential Access to a File

```
#include <fstream>
#include <iostream>
#include <conio.h>
using namespace std;

int main ()
{
   char data[100];
   // open a file in write mode.
   ofstream outfile;
   outfile.open("afile.txt");

   cout << "Writing to the file" << endl;
   cout << "Enter your name: ";
   cin.getline(data, 100);

   // write inputted data into the file.
   outfile << data << endl;

   cout << "Enter your age: ";
   cin >> data;
   // again write inputted data into the file.
   outfile << data << endl;
   // close the opened file.
   outfile.close();
   // open a file in read mode.
   ifstream infile;
   infile.open("afile.txt");
   cout << "Reading from the file" << endl;
   infile >> data;
   // write the data at the screen.
   cout << data << endl;
   // again read the data from the file and display it.
   infile >> data;
   cout << data << endl;
   // close the opened file.
   infile.close();
   getch();
   return 0;
}
```

Random(Direct) Access to a File

Random(Direct) Access to a File

- Instant access is possible with random access files.
- Individual records of a **random access file** can be accessed directly (and quickly) without searching many other records.

Random(Direct) Access to a File

- Every file maintains two internal pointers:
  - **get_pointer and put_pointer**
- **They enable to attain the random access in file** otherwise which is sequential in nature.
- In C++ randomness is achieved by manipulating certain functions

Creating a Random Access File

- **write** - outputs a fixed number of bytes beginning at a specific location in memory to the specified stream
  - When writing an integer **number** to a file,
    - **outFile.write(reinterpret_cast<const char *>(&number), sizeof(number));**
  - First argument: pointer of type **const char *** (location to write from)
    - address of **number** cast into a pointer
  - Second argument: number of bytes to write(**sizeof(number)**)
    - recall that data is represented internally in binary (thus, integers can be stored in 4 bytes)
  - Do not use
    - **outFile << number;**
    - could print 1 to 11 digits, each digit taking up a byte of storage

Random(Direct) Access to a File

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include "clntdata.h"
int main()
{
  ofstream outCredit( "credit.dat", ios::ate );

  if ( !outCredit ) {
    cerr << "File could not be opened." <<
    endl;
    exit( 1 );
  }
cout << "Enter account number "
        << "(1 to 100, 0 to end input)\n? ";

    clientData client;
    cin >> client.accountNumber;
while ( client.accountNumber > 0 &&
        client.accountNumber <= 100 )
{cout << "Enter lastname, firstname,
   balance\n? ";
     cin >> client.lastName >>
   client.firstName
        >> client.balance;

outCredit.seekp( ( client.accountNumber
   - 1 ) *
                sizeof( clientData ) );
    outCredit.write(
      reinterpret_cast<const char *>(
   &client ),
      sizeof( clientData ) );

    cout << "Enter account number\n?
   ";
    cin >> client.accountNumber;
  }

  return 0;
}
```

Writing Data Randomly to a Random Access File

- **seekp** can be used in combination with **write** to store data at exact locations in an output file

Reading data from a random file

```
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <stdlib.h>
#include "clntdata.h"
void outputLine( ostream&, const clientData &
   );
int main()
{
  ifstream inCredit( "credit.dat", ios::in );
  if ( !inCredit ) {
     cerr << "File could not be opened." <<
     endl;
     exit( 1 );
  }
cout << setiosflags( ios::left ) << setw( 10 ) <<
   "Account"
        << setw( 16 ) << "Last Name" << setw( 11 )
        << "First Name" << resetiosflags( ios::left )
        << setw( 10 ) << "Balance" << endl;
clientData client;

  inCredit.read( reinterpret_cast<char *>( &client
),
          sizeof( clientData ) );
while ( inCredit && !inCredit.eof() ) {

    if ( client.accountNumber != 0 )
      outputLine( cout, client );
   inCredit.read( reinterpret_cast<char *>( &client
),
          sizeof( clientData ) );
  }
  return 0;
}
void outputLine( ostream &output, const
clientData &c )
{
  output << setiosflags( ios::left ) << setw( 10 )
       << c.accountNumber << setw( 16 ) <<
c.lastName
       << setw( 11 ) << c.firstName << setw( 10 )
       << setprecision( 2 ) << resetiosflags( ios::left )
       << setiosflags( ios::fixed | ios::showpoint )
       << c.balance << '\n';
}
```

Reading /Writing from/to Textual Files

To write:

- **put()** – writing single character
- **<< operator** – writing an object

To read:

- **get()** – reading a single character of a buffer
- **getline()** – reading a single line
- **>> operator** – reading a object

rajesh.m@vit.ac.in

```
#include <fstream.h>
    main()
    {
      // Writing to file
      ofstream OutFile("my_file.txt");
      OutFile<<"Hello "<<5<<endl;
      OutFile.close();

      int number;
      char dummy[15];


    // Reading from file
      ifstream InFile("my_file.txt");
      InFile>>dummy>>number;


    InFile.seekg(0);

    InFile.getline(dummy,sizeof(dummy));
      InFile.close();
    }
```

Binary Access to a File

Binary file operations

- In connection with a binary file, the file mode must contain the **ios::binary** mode along with other mode(s).

- To read & write a or on to a binary file, as the case may be blocks of data are accessed through the use of C++ **read()** and **write()** respectively.

Reading & Writing Files

put() and get() functions :

- **put()** function displays a single character on the screen.
  - Syntax : put(char);

- **get()** function gets a single character from the file.
  - Syntax : get(void)

Reading & Writing Files

The getline() and write() functions

- These two functions are line oriented input and output functions respectively.
- getline() function **reads a complete line until it encounters a newline** character.
- *Syntax :* **cin.getline(char*,size);**
- Char* → holds the inputline
- Size → The number of characters to be read
- *write Function*
- It is used for **displaying a text in a line until it encounters a '\n'** characters.
- *Syntax* : **cout.write(char*,size);**

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();
}
```

Handling binary data

```
//Example 1: Using read() and write()
#include <fstream>
using namespace std;

int main()
{
ifstream in("binfile.dat",ios::binary);
ofstream out("out.dat", ios::binary);
if(!in || !out) { // return}
unsigned int buf[1024];
while(!in)
{    in.read(buf, sizeof(unsigned int)*1024);
   out.write(buf, sizeof(unsigned int)*1024);
}
}
```

Reading /Writing from/to Binary Files

```
//Example 2: Using read() and write()
// This program uses the write and read functions.
#include <iostream>
#include <fstream>
using namespace std;
int main()
{     fstream file("NUMS.DAT", ios::out | ios::binary);
      int buffer[10] = {1,  2,  3,  4,  5,  6,  7,  8,  9,  10};
      cout << "Now writing the data to the file.\n";
      file.write((char*)buffer, sizeof(buffer));
      file.close();
      file.open("NUMS.DAT", ios::in);  // Reopen the file.
      cout << "Now reading the data back into memory.\n";
      file.read((char*)buffer, sizeof(buffer));
      for (int count = 0; count < 10; count++)
        cout << buffer[count] << " ";
      file.close();
      return(0);}
```

Reading /Writing from/to Binary Files

```
//Example 3: Using get() and put()
#include <iostream>
#include <fstream>
void main()
{
    fstream File("test_file",ios::out | ios::in | ios::binary);
    char ch;
    ch='o';
    File.put(ch); //put the content of ch to the file
    File.seekg(ios::beg); //go to the beginning of the file
    File.get(ch); //read one character
    cout << ch << endl; //display it
    File.close();
}
```

File example

```
#include <iostream.h>
#include <fstream.h>
    /* Input using cin*/
       int main ()
 {
          char myline[256];
          int lc = 0;
          ofstream outfile("demo.txt",ios::app);
          ifstream infile("stdcodes.xyz");
          if (! infile)
       { cerr << "Failed to open input file\n";
                exit(1);  }
          while (1)
       {   infile.getline(myline,256);
                if (infile.eof()) break;
                lc++;
                outfile << lc << ": " << myline << "\n";
             }
          infile.close();
          outfile.close();
          cout << "Output " << lc << " records" << endl;
}
```

Various Operations on Files

Reading & Writing Files

```
// getline() and write() function Example
#include<iostream.h>
#include<conio.h>
void main()
{
char name[50];
clrscr();
Cout<<“Enetr any string”<<endl;
cin.getline(name,40);
cout.write(name,40);
getch();
}
```

Result:

Enter any String : India is My Country

India is My Country

Writing Files

```
//File creation(Writing the contents or data to a file )
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
void main()
{   ofstream outfile;
char fname[20];
cout<<"enter the filename"<<endl;
cin>>fname;
outfile.open(fname);
outfile<<"Sachin"<<endl;
outfile<<"Jackie"<<endl;
outfile<<"Vijay"<<endl;
cout<<"\n "<<"Sucessfully created ";
outfile.close();
}
```

Reading Files

```
//Read the contents(data) from the file
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<fstream.h>
void main()
{    ifstream infile;
   char fname[20];
   char ch;
   cout<<"enter the filename ";
   cin>>fname;
   infile.open(fname);

if(infile.fail())
   {cout<<"file does not exist“; exit(1);}
```

```
while(!infile.eof())
{
ch=infile.get();
cout<<ch;
}
infile.close();
}
```

Copying Files

```
//copy the contents from one file into
   another file
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<fstream.h>
void main()
{
ofstream outfile;
ifstream infile;
char source[10],target[10],ch ;
clrscr();
cout<<"enter the source file name";
cin>>source;
cout<<"enter the target file name";
cin>>target;
infile.open(source);
if(infile.fail())
{
cout<<"source file does not exist";
exit(1);
}
outfile.open(target);
if(outfile.fail())
{cout<<"unable to create a file";
exit(1);}
while(!infile.eof())
{ch=infile.get();
outfile.put(ch);
cout<<ch;}
infile.close();
outfile.close();
getch();}
```

Summary

- A **stream** is generally referred to a **flow of data**.
- Each stream is associated with a particular class which contains member functions and definitions for dealing with that particular kind of data flow.
- The **sequence of input bytes** are called **Input stream.**
- The **sequence of output bytes** are called as **output stream.**
- The **Ofstream** is used for **writing information to a file.**
- The **ifstream** class is used for **reading information from a file .**
- The stream class is used for both writing and reading from a file.

Thank You!...