File handling in C++

BCA SEM III

ASAD HUSSAIN

Using Input/Output Files

Files in C++ are interpreted as a sequence of bytes stored on some storage media.

The data of a file is stored in either readable form or in binary code called as text file or binary file.

The flow of data from any source to a sink is called as a stream

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.

Using Input/Output Files

A computer file

- is stored on a secondary storage device (e.g., disk);
- is permanent;
- can be used to provide input data to a program or receive output data from a program, or both;
- must be opened before it is used.

General File I/O Steps

Declare a file name variable

Associate the file name variable with the disk file name

Open the file

Use the file

Close the file

Using Input/Output Files

Streams act as an interface between files and programs. In C++ . A stream is used to refer to the flow of data from a particular device to the program's variablesThe device here refers to files, keyboard, console, memory arrays. In C++ these streams are treated as objects to support consistent access interface.

They represent as a sequence of bytes and deals with the flow of data.

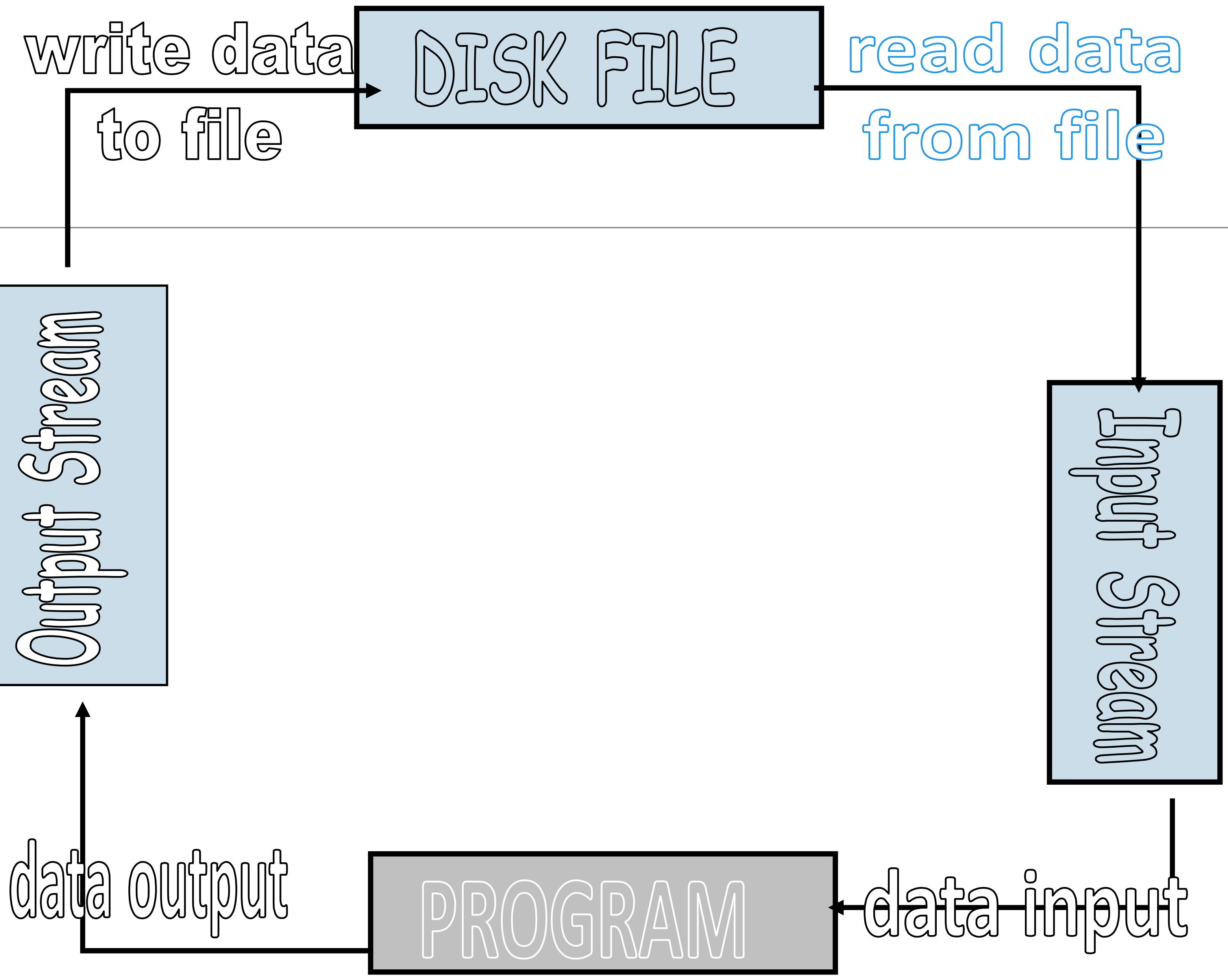
Every stream is associated with a class having member functions and operations for a particular kind of data flow.

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


write data to file
DISK FILE
read data from file
Output Stream
Input Stream
data output
PROGRAM
data input

Using Input/Output Files

stream - a sequence of characters

- interactive (iostream)
  - **cin** - input stream associated with **keyboard.**
  - **cout** - output stream associated with **display.**
- file (fstream)
  - **ifstream** - defines new input stream (normally associated with a file).
  - **ofstream** - defines new output stream (normally associated with a file).

• Stream of bytes to do input and output to different devices.
• 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

A stream is a series of bytes, which act either as a source from which data can be extracted or as a destination to which the output can be sent. Streams resemble the producer and consumer model

The producer produces the items to be consumed by the consumer. The producer and the consumers are connected by the C++ operators >> or <<. For instance , the keyboard exhibits the nature of only a producer,printer or monitor screen exhibit the nature of only a consumer. Whereas , a file stored on the disk , can behave as a producer or consumer, depending upon the operation initiated on it.

Predefined console streams

C++ contains several predefined streams that are opened automatically when the execution of a program starts.

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)

Why to use Files

Convenient way to deal large quantities of data.

Store data permanently (until file is deleted).

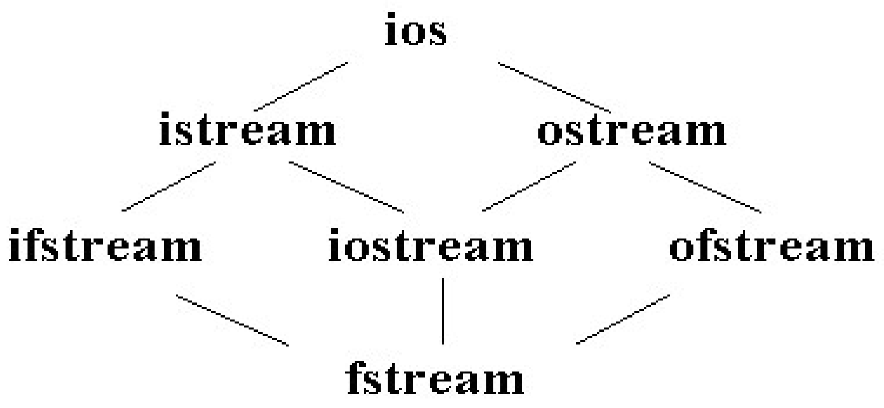
Avoid typing data into program multiple times.

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

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

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

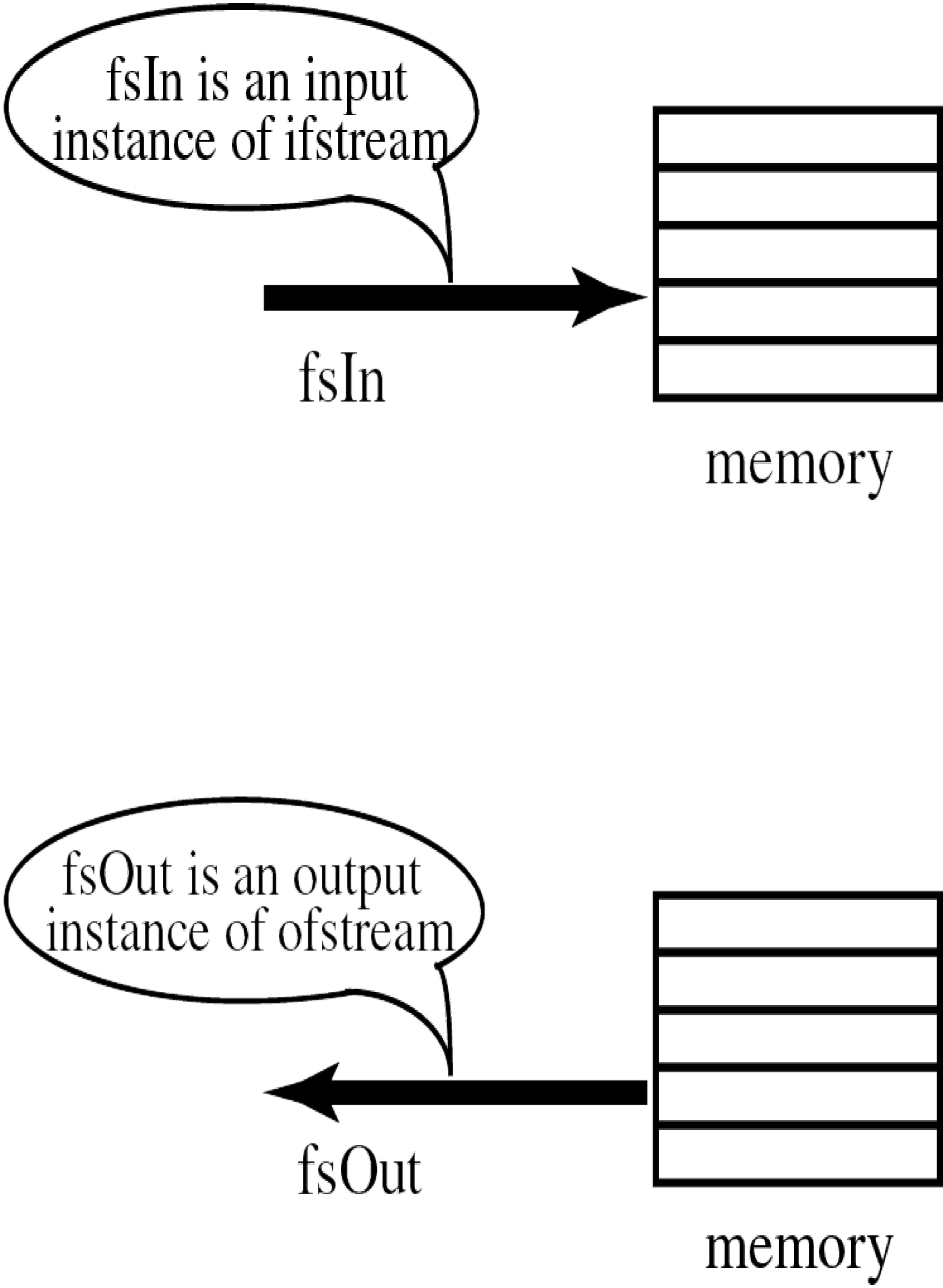
When ever we include <iostream.h>, an ostream object, is automatically defined – this object is cout.
ofstream inherits from the class ostream (standard output class).
ostream overloaded the operator >> for standard output….thus an ofstream object can use methods and operators defined in ostream.

```
#include <fstream.h>

int main (void)
{
//  Local Declarations
    ifstream      fsIn;

    ofstream      fsOut;
    •
    •
    •

}   //  main
```


fsIn is an input instance of ifstream
fsIn
memory
fsOut is an output instance of ofstream
fsOut
memory

File Modes

Name	Description
ios::in	Open file to read
ios::out	Open file to write
ios::app	All the date you write, is put at the end of the file. It calls ios::out
ios::ate	All the date you write, is put at the end of the file. It does not call ios::out
ios::trunc	Deletes all previous content in the file. (empties the file)
ios::nocreate	If the file does not exist, opening it with the open() function gets impossible.
ios::noreplace	If the file exists, trying to open it with the open() function, returns an error.
ios::binary	Opens the file in binary mode.

File Modes

Opening a file in ios::out mode also opens it in the ios::trunc mode by default. That is, if the file already exists, it is truncated

Both ios::app and ios::ate set the pointers to the end of file, but they differ in terms of the types of operations permitted on a file. The ios::app allows to add data from end of file, whereas ios::ate mode allows to add or modify the existing data anywhere in the file. In both the cases the file is created if it is non existent.

The mode ios::app can be used only with output files

The stream classes ifstream and ofstream open files in read and write modes by default.

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.

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.

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.

<u>Specifying the offset :</u>

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 Open Mode

```
#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
```

Dealing with Binary files

Functions for binary file handling

get(): read a byte and point to the next byte to read

put(): write a byte and point to the next location for write

read(): block reading

write(): block writing

flush():Save data from the buffer to the output file.

Binary File I/O Examples

```
//Example 1: 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 I/O Example: Writing

```
#include <fstream>
using namespace std;
int main(void)
{
 ofstream outFile(“fout.txt");
 outFile << "Hello World!";
 outFile.close();
 return 0;
}
```

File I/O Example: Reading

```
#include <iostream>

#include <fstream>


int main(void)

{

    ifstream openFile(“data.txt"); //open a text file data.txt

    char ch;

    while(!OpenFile.eof())

    {

                                    OpenFile.get(ch);

                                    cout << ch;

    }

    OpenFile.close();


    return 0;

}
```

File I/O Example: Reading

```
#include <iostream>

#include <fstream>

#include <string>


int main(void)

{

    ifstream openFile(“data.txt"); //open a text file data.txt

    string line;



      if(openFile.is_open()){ //

    while(!openFile.eof()){

    getline(openFile,line);//read a line from data.txt and put it in a string

                                      cout << line;

    }

      else{

       cout<<“File does not exist!”<<endl;

       exit(1);}

      }

    openFile.close();

     return 0;

}
```

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>}
```

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();

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();
}
```

C++ has some low-level facilities for character I/O.

char next1, next2, next3;

cin.get(next1);

Gets the next character from the keyboard. Does not skip over blanks or newline (\n). Can check for newline (next == '\n')

Example:

- cin.get(next1);
- cin.get(next2);
- cin.get(next3);

Predefined character functions must #include <ctype.h> and can be used to

- convert between upper and lower case
- test whether in upper or lower case
- test whether alphabetic character or digit
- test for space

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

```
#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 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.

Handling binary data

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

int main(){

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