CAP444
OBJECT ORIENTED PROGRAMMING USING C++

Created By:
Kumar Vishal
(SCA), LPU

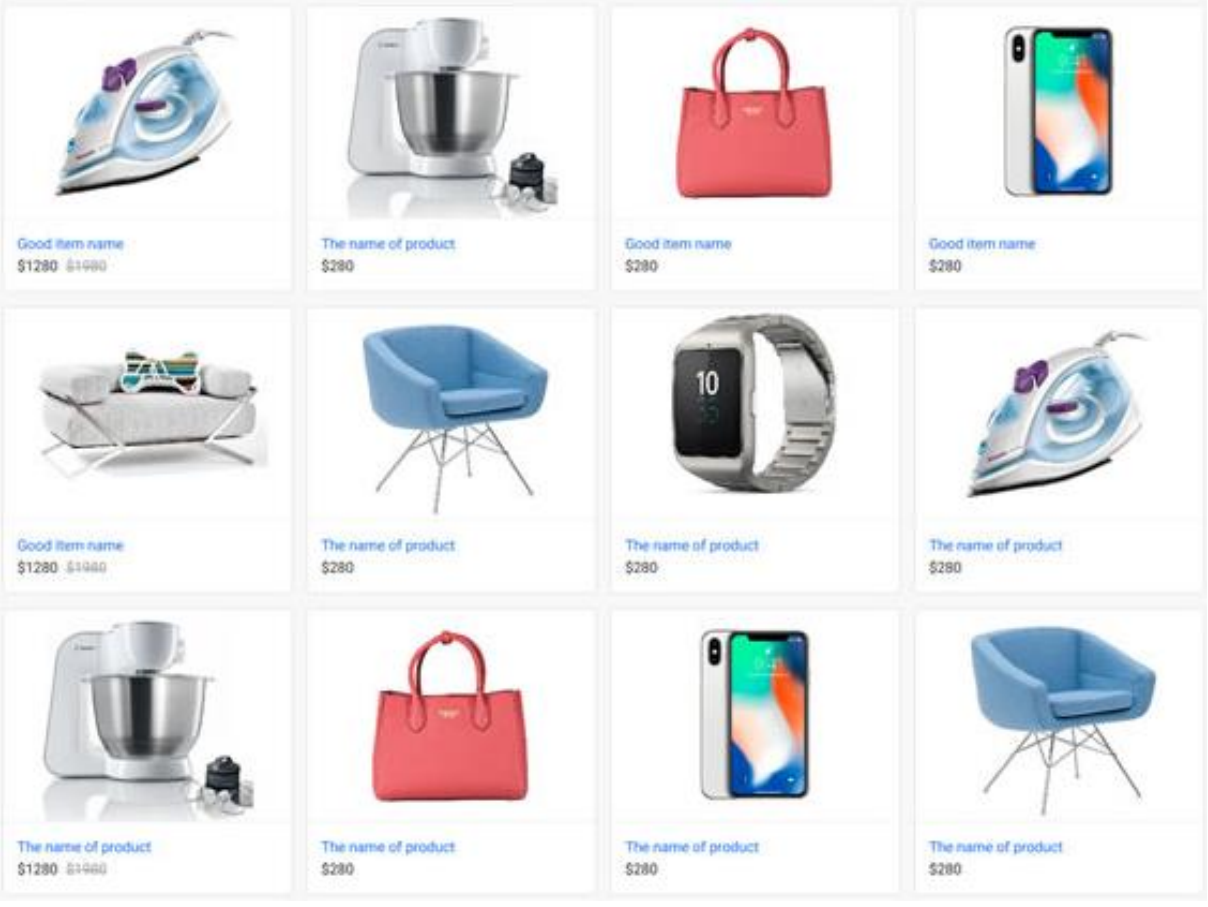
Unit-4

Working with files and streams :

- c++ streams, c++ stream classes,
- classes for file stream operations,
- opening & closing files,
- detection of end of file,
- more about open(): file modes,
- file pointer & manipulator,
- sequential input & output operation,
- updating a file: random access,
- command line arguments

Keeping record of products:

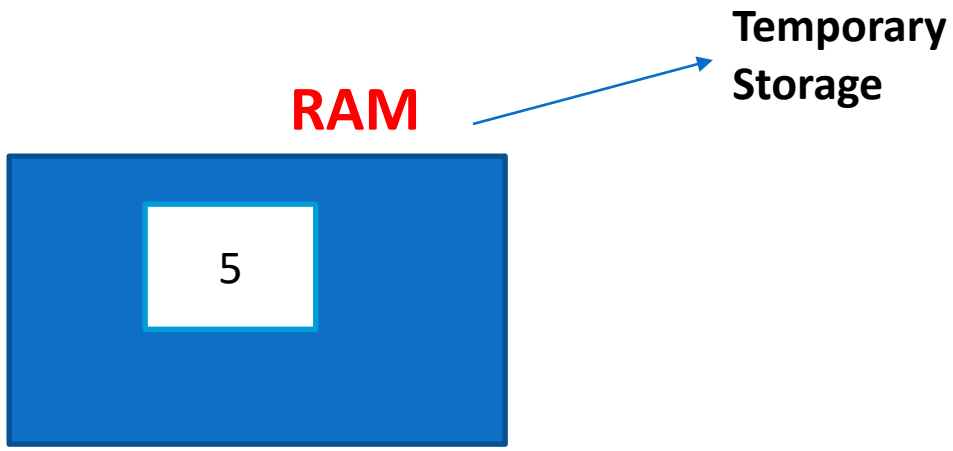
- using file handling mechanism




E-COMMERCE

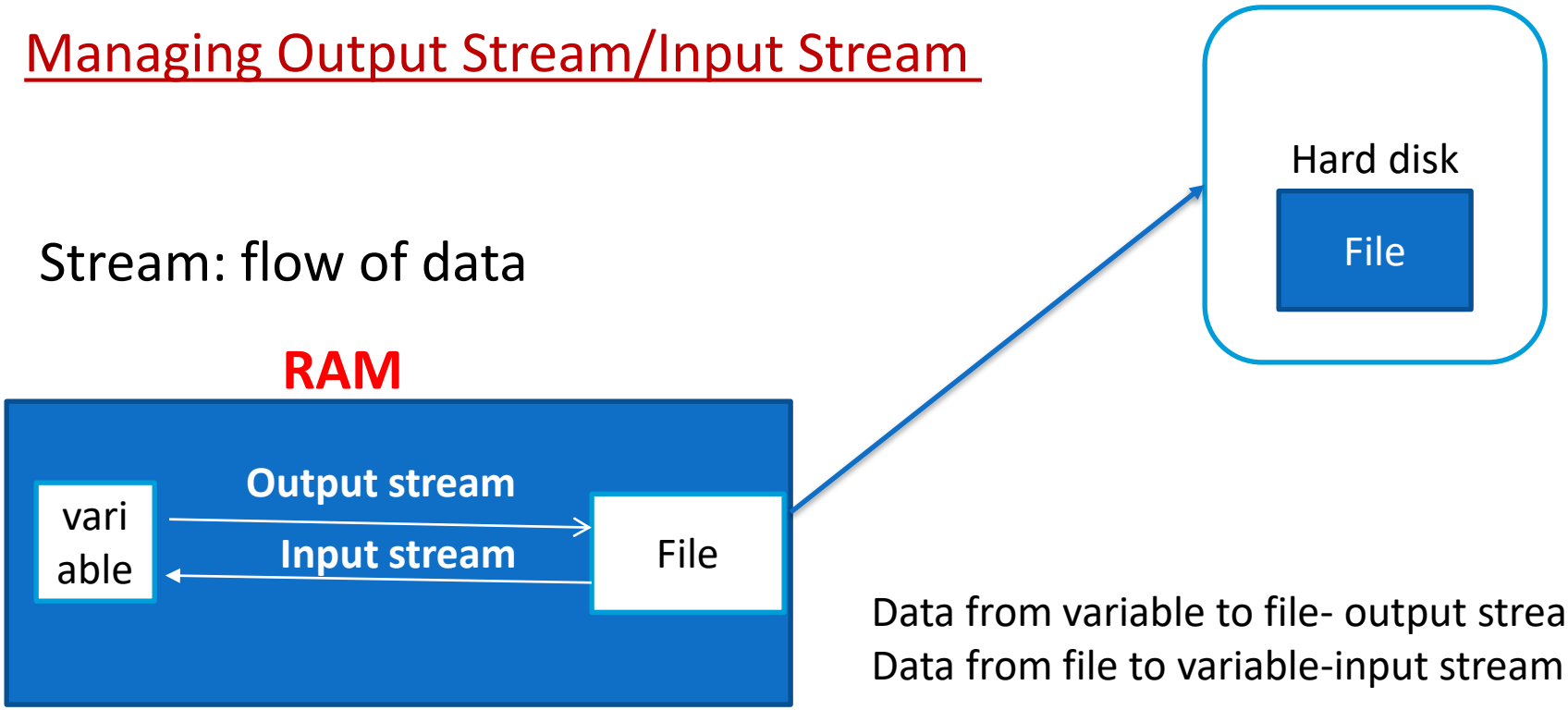
Real Life Example

```
#include <iostream>
using namespace std;
int main()
{
    int num;
 cout<<"Enter
number"<<endl;
    cin>>num;
    return 0;
}
```



Managing Output Stream/Input Stream

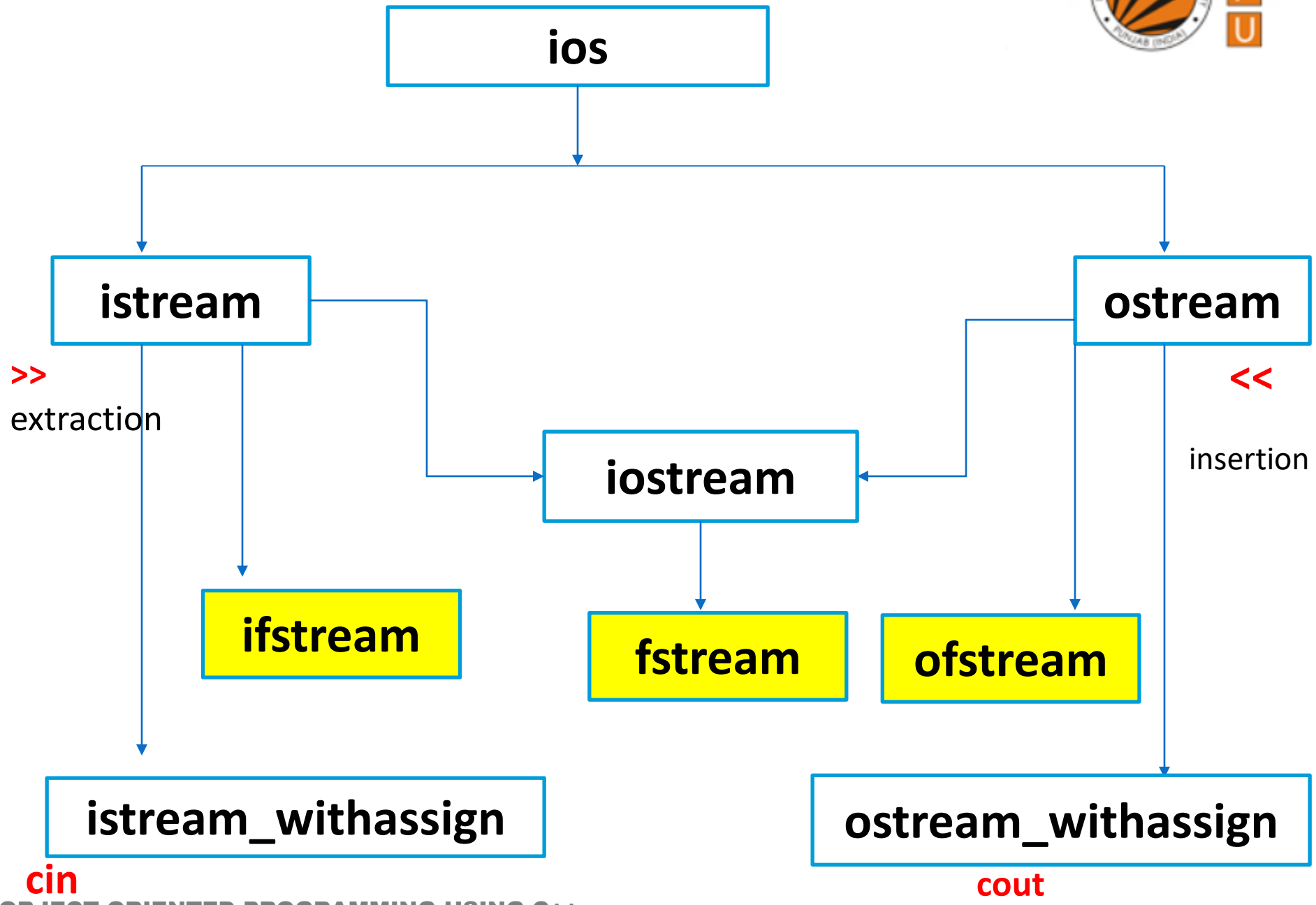
Stream: flow of data



Data from variable to file- output stream
Data from file to variable-input stream

We have predefine classes to manage all these things

Classes for file stream



In C++, files are mainly deal with three classes fstream, ifstream, ofstream.

ofstream: This Stream class indicates the output file stream and is applied to create files for writing information to files

ifstream: This Stream class indicates the input file stream and is applied for reading information from files

fstream: This Stream class can be used for both read and write from/to files.

C++ provides us with the following operations in File Handling:

- Creating a file: open()
- Reading data: read()
- Writing new data: write()
- Closing a file: close()

Which of the following is not a component of file system

A. Access method

B. Auxiliary storage management

C. Free integrity mechanism

D. None of the above

Opening Files

- open() In case of creating new file:
  - Using ofstream class

  Syntax:

  ofstream fout;

  Fout.open("filename")
- open() In case of reading file:
  - Using ifstream class

  Syntax:

  ifstream fin;

  fin.open("filename")

Closing Files

- close() In case of creating new file:
  - Using ofstream class

  Syntax:

  ofstream fout;

  Fout.close()
- close() In case of reading file:
  - Using ifstream class

  Syntax:

  ifstream fin;

  fin.close()

Reading and Writing into Files

- Writing File: used ofstream class:

  Syntax:

  ofstream fout;

  fout.open(“filename”);

  fout<<“data”;

- Reading File: used ifstream class:

  Syntax:

  ifstream fin;

  Ifstream.open(“filename”);

  using get() or getline()

Reading Files: using get() function

The get() function is member of ifstream class. It is used to read character form the file.

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

will read all the characters one by one up to EOF(end-of-file) reached.

Writing Files: using put() function

```
ofstream fout;        // create output stream
char ch;
int i;
fout.open("filename", ios::app) ;
/* write the characters to file using put() */
fout.put(ch);
fout.close();
```

Detecting End-of-File

- While reading data from a file, if the file contains multiple rows, it is necessary to detect the end of file.
- This can be done using the eof() function of ios class.
- It returns 0 when there is data to be read and a non-zero value if there is no data.

Syntax:

```
ifstream fin;
char ch;
Ifstream.open("filename");
while(!fin.eof())
{
   fin.get(ch);
   cout<<ch;
}
```

Reading Files: using getline()

How to process a file line by line in C++?

In C++, you may open a input stream on the file and use the getline() function from the <string> to read content line by line into a string and process them.

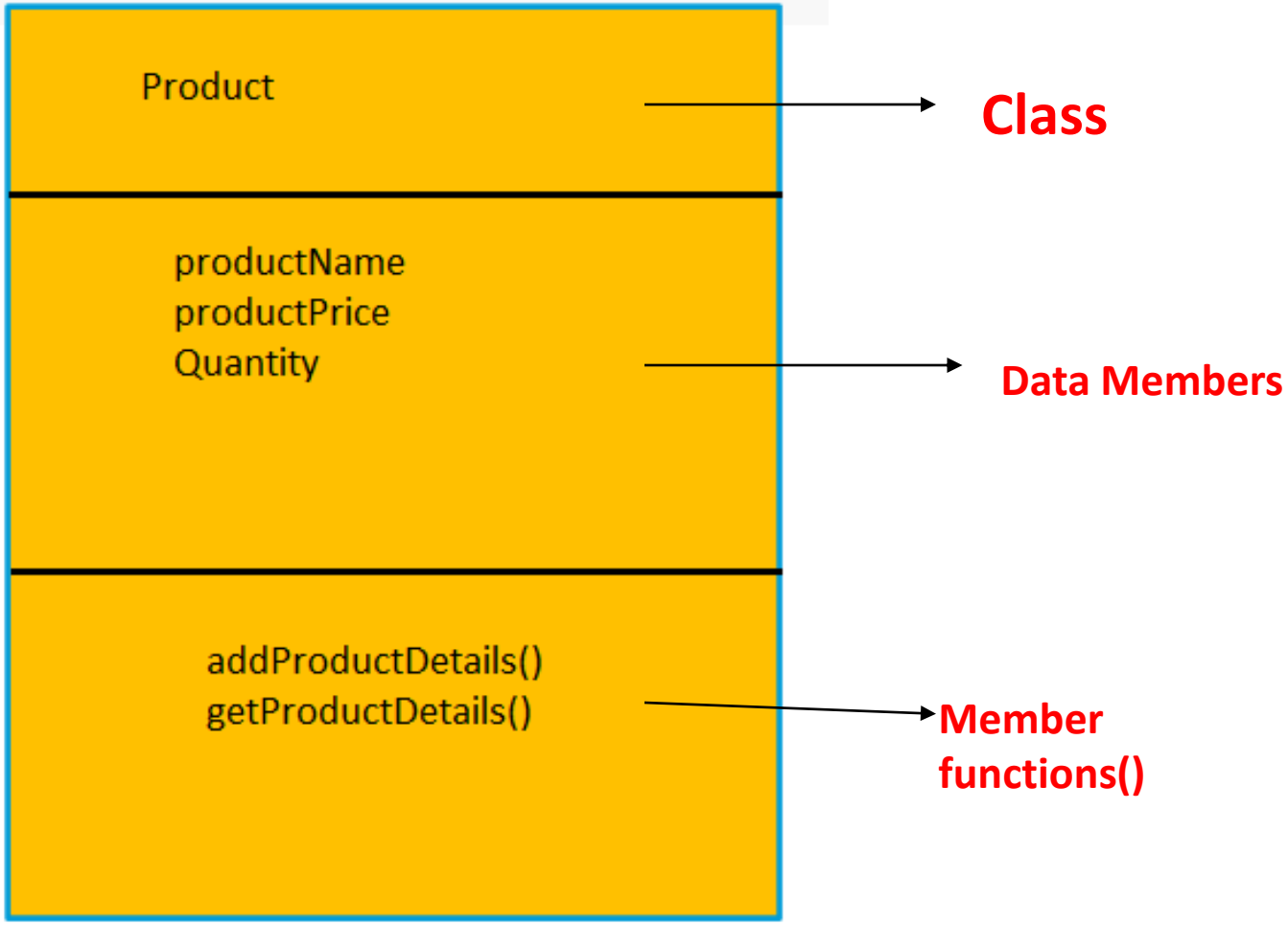
```
Ifstream fin;
fin.open("d://demo//file1.txt");
   string str;
   while(getline(in,str))
   {
     cout<<str<<endl;
   }
```

Which of the following methods can be used to open a file in file handling?

a) Using Open ( )
b) Constructor method
c) Destructor method
d) Both A and B

Steps:

1. Create a product class



Steps:

2. Create a file and fill all product records.

3. Update your file, fill more records into file

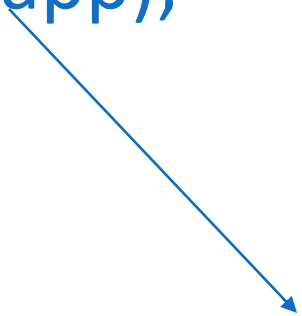
3. Display output to the user screen with product details.

To append file content

ios::app

ofstream fout;

fout.open(“filename”,ios::app);



File Mode

Check file is existing or not:

```
ifstream  fin;
fin.open(“abc.txt”);
If(fin)
{
cout<<“File is existing”<<endl;
}
else{
cout<<“File is not existing”<<endl;

}
```

```
#include <iostream>
#include<fstream>
using namespace std;
ofstream fout;
ifstream fin;

int main()
{
fout.open("hello.txt",  ios::in);
fout<<"hello"<<endl;
fout.close();
return 0;
}
```

A.

File will create and hello will be written on file

B.

Compilation Error

C.

Nothing will be happen

D.

None

File Modes

ios::in	Open for input operations.
ios::out	Open for output operations.
ios::binary	Open in binary mode.
ios::ate	Set the initial position at the end of the file. If this flag is not set, the initial position is the beginning of the file.
ios::app	All output operations are performed at the end of the file, appending the content to the current content of the file.

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

ios::in	Searches for the file and opens it in the **read** mode only(*if the file is found*).
ios::out	Searches for the file and opens it in the **write** mode. If the file is found, its content is overwritten. If the file is not found, a new file is created. *Allows you to **write** to the file.*
ios::app	Searches for the file and opens it in the **append** mode i.e. this mode allows you to **append** new data to the end of a file. If the file is not found, a new file is created.

ios::binary	Searches for the file and opens the file(if the file is found) in a binary mode to perform binary input/output file operations.
ios::ate	Searches for the file, opens it and positions the pointer at the end of the file. This mode when used with **ios::binary, ios::in** and **ios::out** modes, *allows you to **modify** the content of a file.*
ios::trunc	Searches for the file and opens it to truncate or deletes all of its content(*if the file is found*).

ios::nocreate	Searches for the file and if the file is not found, a new file will not be created.

Read/Write Class Objects from/to File in C++

The data transfer is usually done using '>>' and <<' operators. But if you have a class with 4 data members and want to write all 4 data members from its object directly to a file or vice-versa.

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));
```

Topics:

Working with files and streams :

- file pointer & manipulator,
- sequential input & output operation,
- updating a file: random access,
- command line arguments

file pointer

Topics will be there in coming slides

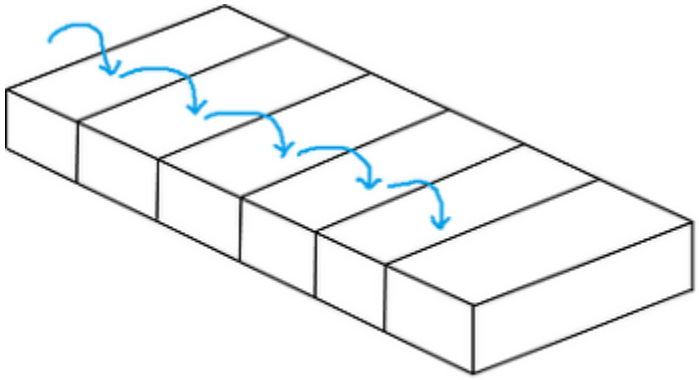
file manipulator

Manipulators are simply an instruction to the output stream that modify the output in various ways. In other words, we can say that manipulators are operators that are used to format the data display.

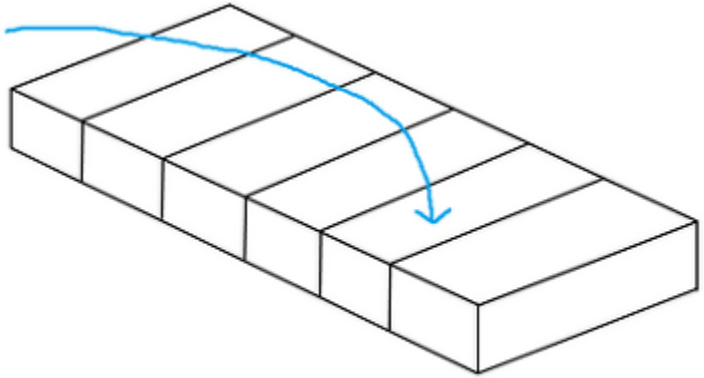
Manipulator	Purpose	Header File
endl	causes line feed to be inserted i.e. '\n'	iostream.h
dec,oct,hex	set the desired number system	iostream.h
setbase (b)	output integers in base b	iomanip.h
setw(w)	read or write values to w characters	iomanip.h
setfill (c)	fills the whitespace with character c	iomanip.h
setprecision(n)	set floating point precision to n	iomanip.h

Sequential and Random I/O

- C++ allows data to be read or written from a file in sequential or random fashion.
- Reading data character by character or record by record is called sequential access.
- Reading data in any order is known as random access.
- The fstream class provides functions like get(), read() for reading data and put(), write() for writing data to a file.



sequential access



random access


1
2
-1
0

- For example, if you have to modify a value in record no 21, then using random access techniques, you can place the file pointer at the beginning of record 21 and then straight-way process the record. If sequential access is used, then you'll have to unnecessarily go through first twenty records in order to reach at record 21.
- In C++, random access techniques is achieved by manipulating seekg(), seekp(), tellg() and tellp() functions.

File Pointers:

Every file will contain two pointers: a read pointer or also known as a get pointer and a write pointer also known as a put pointer. The read pointer or a get pointer is used to read data and the write pointer or put pointer is used to write data to a file. These pointers can be manipulated using the functions from stream classes. Those functions are:

seekg() and seekp()

seekp() function allow us to move the output pointer to specified location for writing purpose within the file. The basic syntax for seekp() function is :

fileObject.seekp(long_num, origin);

- fileObject: pointer to file
- long_num: no. of bytes in file we want to skip
- origin: where to begin

seekg() function allow us to move the Input pointer to specified location for reading purpose within the file. The basic syntax for seekg() function is :

fileObject.seekg(long_num, origin);

- fileObject: pointer to file
- long_num: no. of bytes in file we want to skip
- origin: where to begin

origin:

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

Ex:

```
fin.seekg(0, ios::beg);
```

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

Setting the EOF flag off, to allow the access of file again for reading:-

Ifstream fin;

fin.clear();

Which function is used to reposition the file pointer?

a) moveg()

b) seekg()

c) changep()

d) go_p()

Which of the following is used to move the file pointer to start of a file?

a) ios::beg

b) ios::start

c) ios::cur

d) ios::first

Command-line arguments

- Command-line arguments are given after the name of the program in command-line shell of Operating Systems.
- To pass command line arguments, we use main() with two arguments :
  - first argument is the total number of command line arguments and
  - second is list of command-line arguments.

Syntax:

```
int main(int argc, char *argv[ ])
{
return 0;
}
```

first parameter *argc* holds the count of command-line arguments and the second parameter, an array of character pointers holds the list of command-line arguments.

first element in the array, i.e., argv[0] holds the filename. First command-line parameter will be available in argv[1], second parameter in argv[2] and so on.

To run in command line:

D:\>g++ abc.cpp -o obj1.exe

D:\>obj1.exe

Any Query?

Unit-4 End