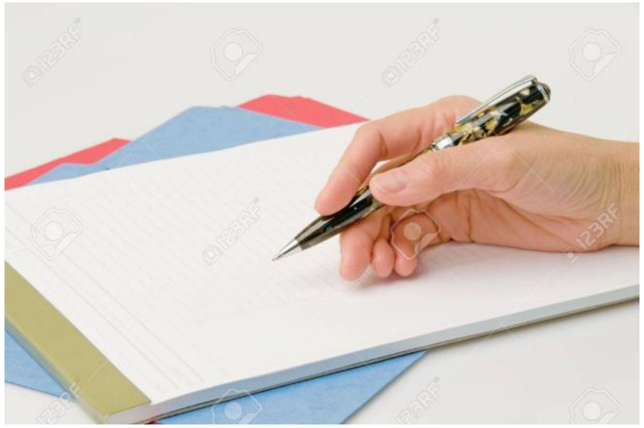
CAP444
OBJECT ORIENTED PROGRAMMING USING C++

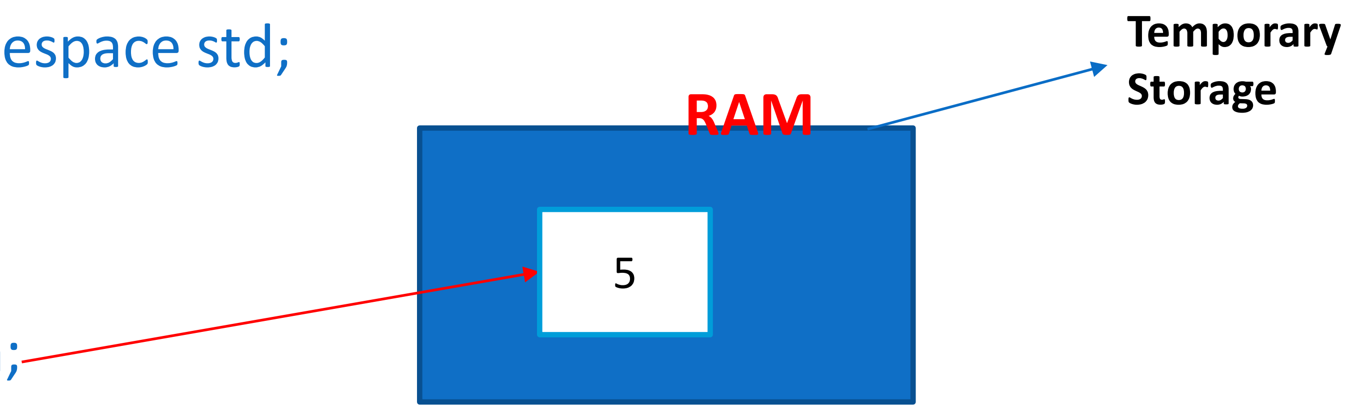
Created By:
Kumar Vishal
(SCA), LPU

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*



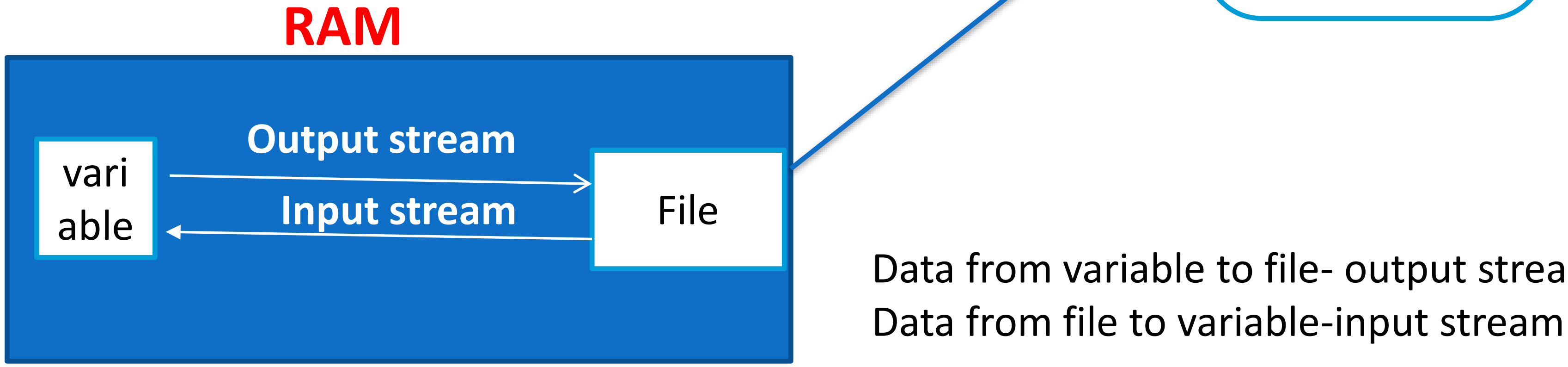
```
#include <iostream>
using namespace std;
int main()
{
    int num;
    cout<<"Enter number"<<endl;
    cin>>num;
   return 0;
}
```



Managing Output Stream/Input Stream

The sequence of bytes given as input to the executing program and the sequence of bytes that comes as output from the executing program are called stream. In other words, streams are nothing but the flow of data in a sequence.

Stream: flow of data



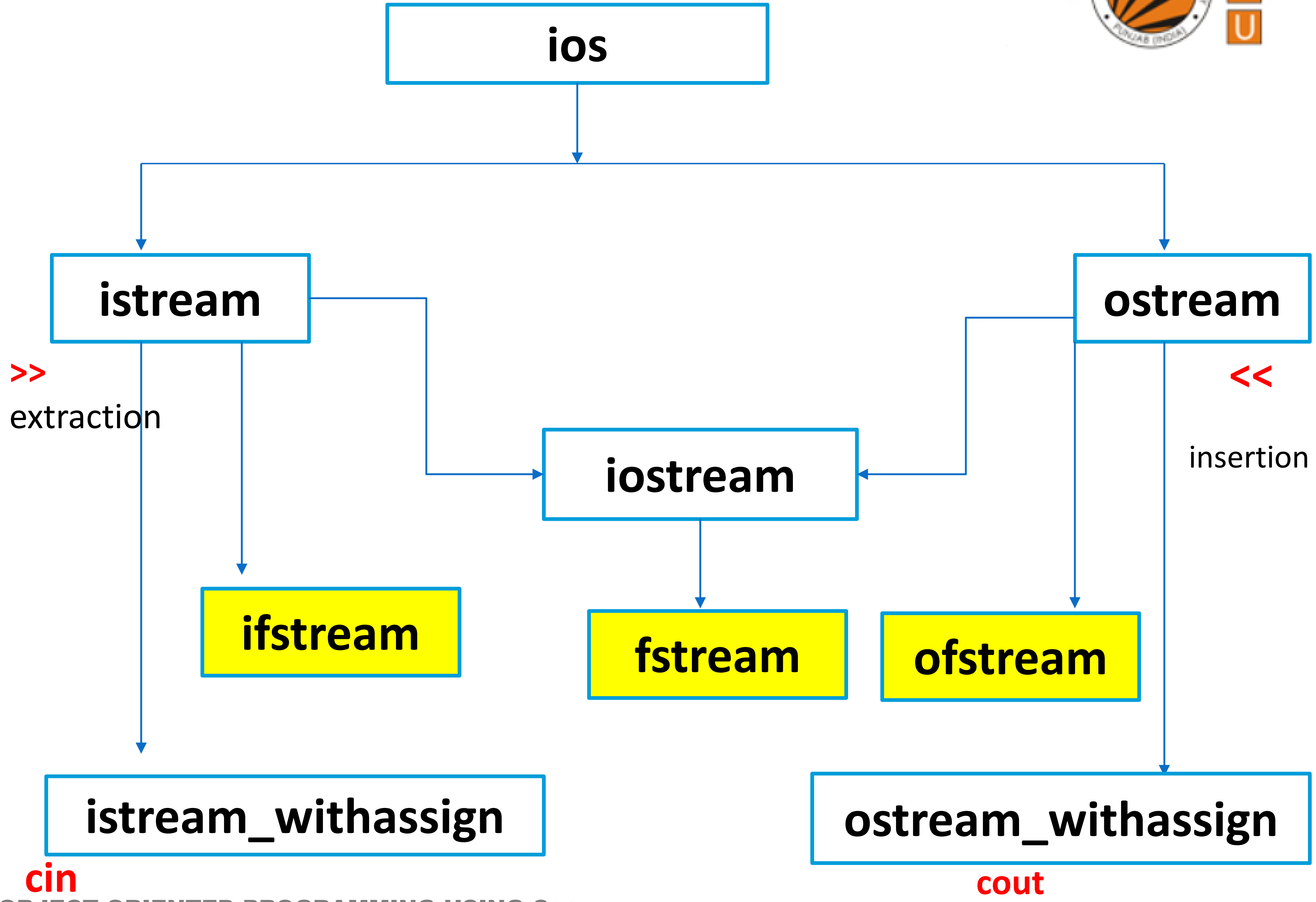
Data from variable to file- output stream
Data from file to variable-input stream

We have predefine classes to manage all these things

C++ Stream Classes

The I/O system of C++ contains a set of classes which define the file handling methods.

Classes for file stream



1. ios/iostream

- ios stands for input output stream.
- This class is the base class for other classes in this class hierarchy.
- This class contains the necessary facilities that are used by all the other derived classes for input and output operations.

2. istream

- istream stands for input stream.
- This class is derived from the class ‘ios’.
- This class handle input stream.
- The extraction operator(>>) is overloaded in this class to handle input streams from files to the program execution.
- This class declares input functions such as get(), getline() and read().

3. ostream

- ostream stands for output stream.
- This class is derived from the class ‘ios’.
- This class handle output stream.
- The insertion operator(<<) is overloaded in this class to handle output streams to files from the program execution.
- This class declares output functions such as put() and write().

4. streambuf:-

- This class contains a pointer which points to the buffer which is used to manage the input and output streams.

5. fstreambase:-

- This class provides operations common to the file streams. Serves as a base for fstream, ifstream and ofstream class.
- This class contains open() and close() function.

6. ifstream:-

- This class provides input operations.
- It contains open() function with default input mode.
- Inherits the functions get(), getline(), read(), seekg() and tellg() functions from the istream.

7. ofstream:-

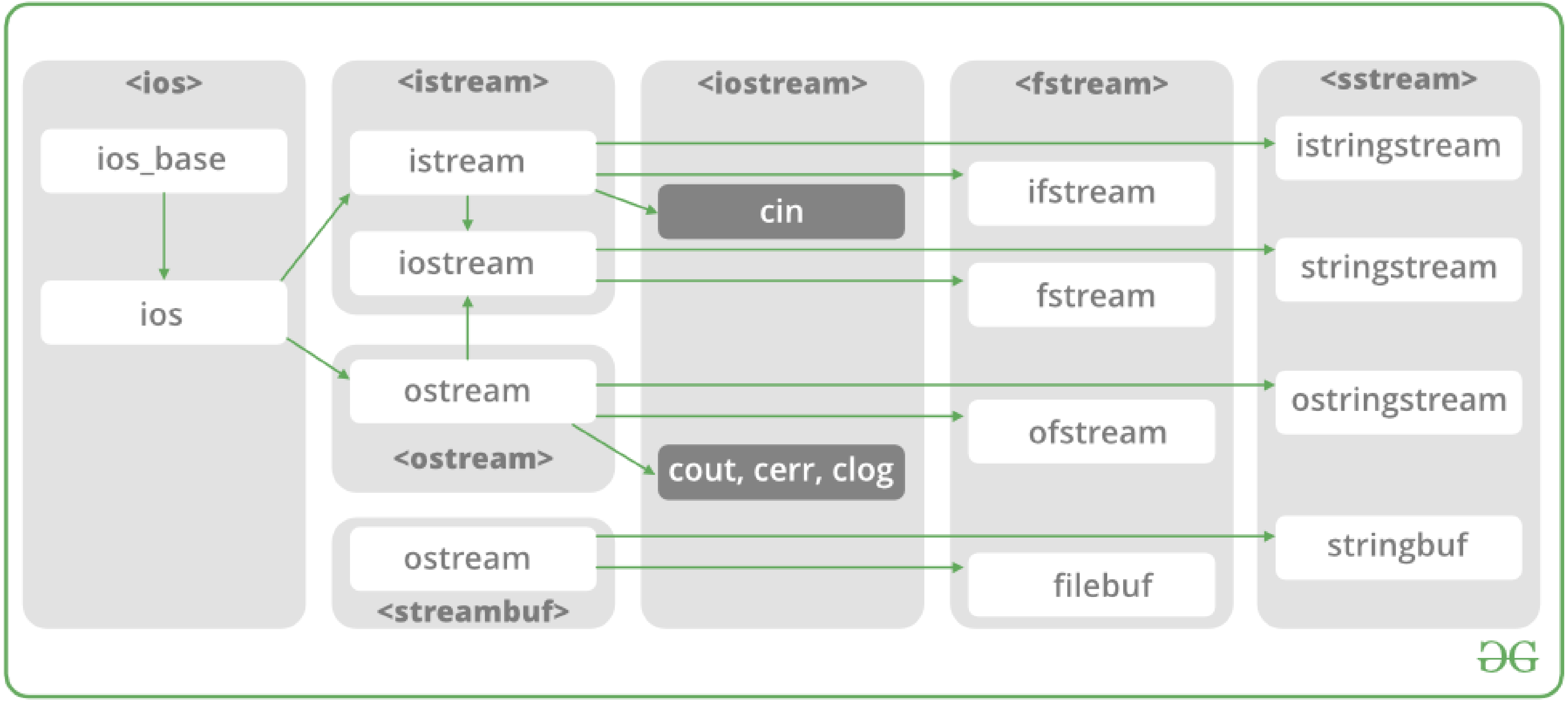
- This class provides output operations.
- It contains open() function with default output mode.
- Inherits the functions put(), write(), seekp() and tellp() functions from the ostream.

8. fstream:-

- This class provides support for simultaneous input and output operations.
- Inherits all the functions from istream and ostream classes through iostream.

9. filebuf:-

- Its purpose is to set the file buffers to read and write.
- We can also use file buffer member function to determine the length of the file.



<ios>
ios_base
ios
<istream>
istream
iostream
<ostream>
ostream
<streambuf>
ostream
<iostream>
cin
cout, cerr, clog
<fstream>
ifstream
fstream
ofstream
filebuf
<sstream>
istringstream
stringstream
ostringstream
stringbuf

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

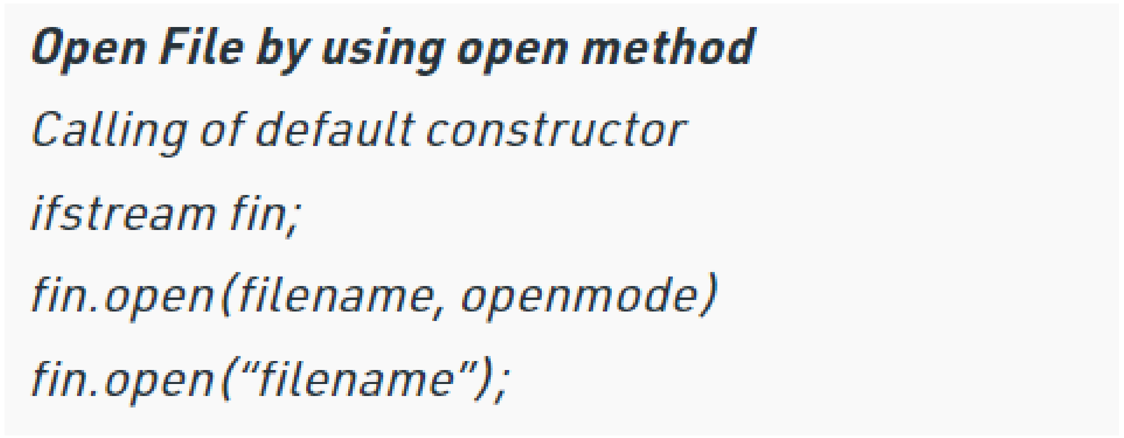
ofstream: This Stream class indicates the output file stream and is applied to create files for writing information to files

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()



Open File by using open method

Calling of default constructor

ifstream fin;

fin.open(filename, openmode)

fin.open("filename");

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.

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()

Opening Files using open()

- 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”)

Opening Files using constructor

```
ofstream fout("filename");
```

```
ifstream infile("salary");
```

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=‘a’;
  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;
}
```

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;

}
```

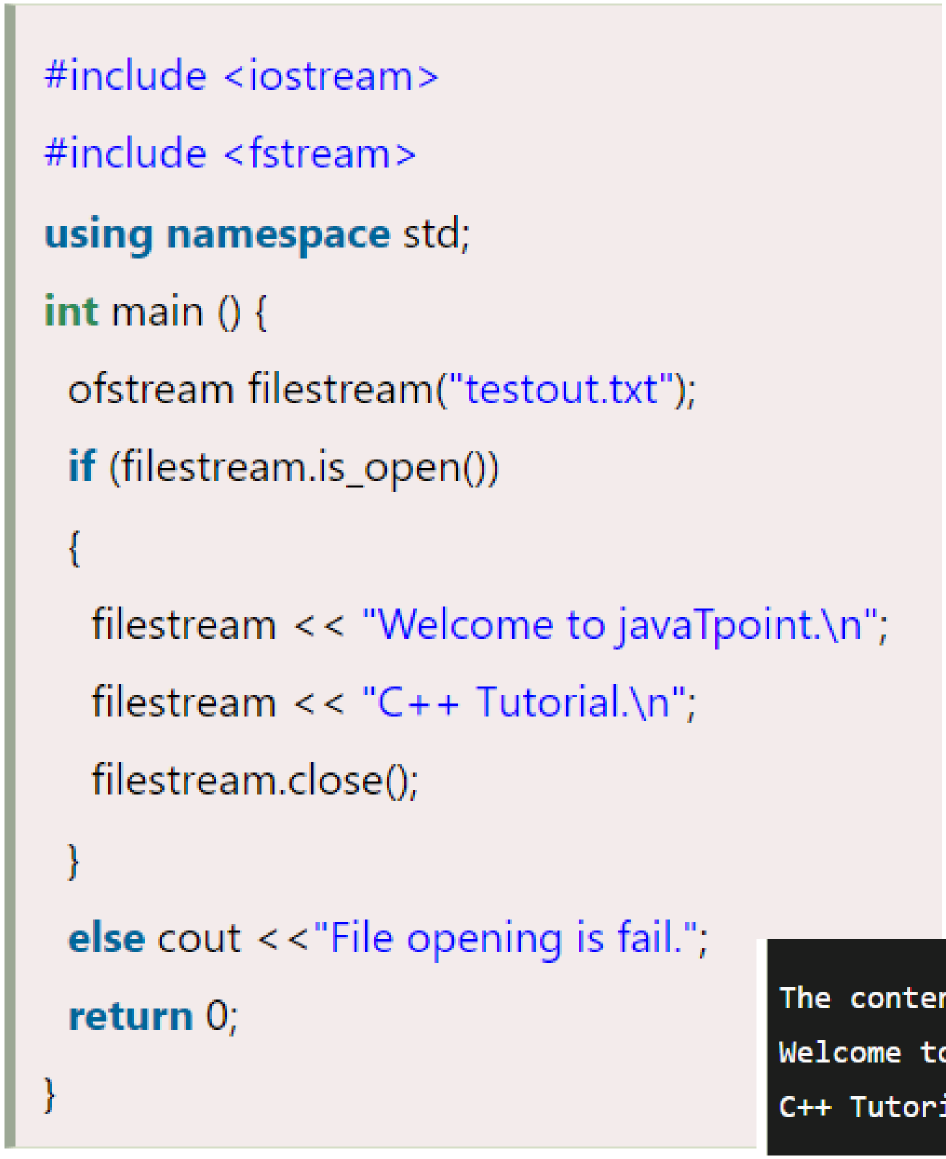
Reading Files: using getline()

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

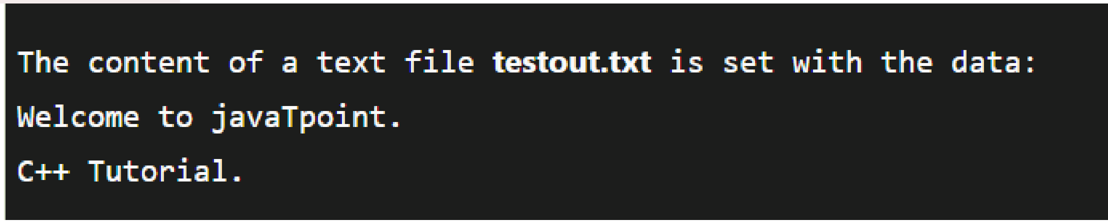
In C++, you may open a input stream on the file and use the getline() function from the <string> to read content line by line into a string and process them.

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

writing to a file



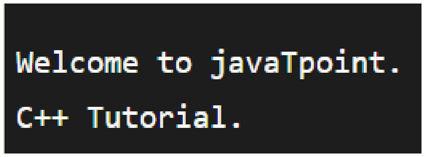
```cpp
#include <iostream>
#include <fstream>
using namespace std;
int main () {
  ofstream filestream("testout.txt");
  if (filestream.is_open())
  {
    filestream << "Welcome to javaTpoint.\n";
    filestream << "C++ Tutorial.\n";
    filestream.close();
  }
  else cout <<"File opening is fail.";
  return 0;
}
```



```
The content of a text file testout.txt is set with the data:
Welcome to javaTpoint.
C++ Tutorial.
```

reading from a file

```cpp
#include <iostream>
#include <fstream>
using namespace std;
int main () {
  string srg;
  ifstream filestream("testout.txt");
  if (filestream.is_open())
  {
    while ( getline (filestream,srg) )
    {
      cout << srg <<endl;
    }
    filestream.close();
  }
  else {
      cout << "File opening is fail."<<endl;
    }
  return 0;
}
```



```
Welcome to javaTpoint.
C++ Tutorial.
```

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

int main () {
   char data[100];

   // open a file in write mode.
   ofstream outfile;
   outfile.open("afile.dat");

   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;
   cin.ignore();

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

   // close the opened file.
   outfile.close();
```

```cpp
   // open a file in read mode.
   ifstream infile;
   infile.open("afile.dat");

   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();

   return 0;
}
```

```
Writing to the file
Enter your name: Zara
Enter your age: 9
Reading from the file
Zara
```

```cpp
#include <iostream>

#include <fstream>

int main()

{

std::ofstream fout; //create the output stream

fout.open("country"); // connect "country" to it

fout<<"United states \n";

fout<<"united kingdom \n";

fout<<"south korea \n";

fout.close(); // disconnect the country

fout.open("capital"); // and connect the capital

fout<<"washington \n";


fout<<"London \n";

fout<<"seoul \n";

fout.close(); // disconnect the "capital"
```

```cpp
// reading the files

const int n=80; // size of line

char line[n];

std::ifstream fin; // create input stream

fin.open("country"); // connect country to it

std::cout << "the contents of country file \n";

while (fin) // check end of file

fin.getline(line,n); // ready a line

std::cout << line; // display it


fin.close(); // disconnect country

fin.open("capital"); // and connect to capital

std::cout << "\n contents of capital file \n";

while(fin)

{

fin.getline(line,n);

std::cout << line;

}

fin.close();

}
```

```
output:
Contents of country file
United states
united kingdom
south korea
contents of capital file
washington
london
seoul
```

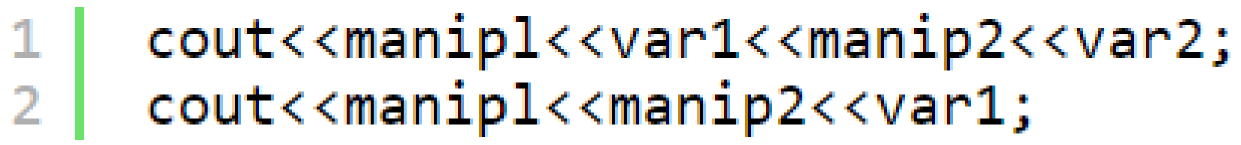
```
// Reads the files created in program
#include < iostream>
#include <fstream>
#include <stdlib.h>
int main()
{
const int SIZE=80;
char line[SIZE];
std::ifstream fin1,fin2; // create two input streams
fin1.open("country");
fin2.open("capital");
for (int i=1;i<=10;i++)
{
if (fin1.eof()!=0)
{
std::cout << "Exit from country \n";
exit (1);
}
fin1.getline(line,SIZE);
std::cout << "capital of " << line;
if(fin2.eof()!=0)
{
std::cout << "Exit from capital \n";
exit (1);
}
fin2.getline(line,SIZE);
std::cout << line << "\n";
}
}
```

```
output:
capital of united states of America
Washington
capital of united kingdom
London
capital of south korea
seoul
```

C++ File pointers and Manipulators

- The header file iomanip provides a set of functions called manipulators which can be used to the manipulate the output formats.
- The manipulators in C++ are special functions that can be used with insertion (<<) and extraction (>>) operators to manipulate or format the data in the desired way.
- Certain manipulators are used with << operator to display the output in a particular format, whereas certain manipulators are used with >> operator to input the data in the desired form.
- The manipulators are used to set field widths, set precision, inserting a new line, skipping white space etc.

- In a single I/O statement, we can have more than one manipulator, which can be chained as shown



```
cout<<manipl<<var1<<manip2<<var2;
cout<<manipl<<manip2<<var1;
```

To use most of the manipulators, we need to include the header file iomanip.h in the program.

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

endl Manipulator

The endl manipulator stands for endline and is used with an insertion operator (<<) that moves the cursor to the next line. If we do not use endl, the next output will be displayed in the same line. The endl has the same function as that of ‘\n.’

```
#include<iostraam.h>
#include<conio.h>
int main() {
 cout<<"Entar name"<<endl;
 cout<<"Myname is Thakur";
 getch();
 return 0;
}
Output
Enter name
Myname is Thakur
```

Dec, Oct,Hex Manipulator

All the numbers are displayed and read in decimal notation by default. However, you may change the base of an integer value to octal or hexadecimal or back to a decimal using the manipulator's oct, hex or dec, respectively. These manipulators are preceded by the appropriate variables to be used with.

```
#include<iostream.h>
#include<conio.h>
int: main() {
 int i;
 cout<<"Enter hexadecimal number =";
 cin>>hex>>i;
 cout:<<"Hexadecimal value = "<<hex<<i<<endl;
 cout<<"Octal Value = "<<oct<<i<<endl;
 cout<<"Dcimal Value = "<<dec<<i<<endl;
 getch();
 return 0;
}
Output :
Enter hexadecimal = f
Hexadecimal = f
Octal value = 17
Decimal value = 15
```

setbase(b) Manipulator

The setbase () manipulator is used to change the base of a numeric value during inputting and outputting. It is an alternative to Dec, Oct and hex manipulators. It is a function that takes a single integer argument(b) having values 8, 10 or 16 to set the base of the numeric value to octal, decimal and hexadecimal, respectively. The default base is 10.

```
#include<iostream.h>
#include<iomanip.h>
int main() {
 int num;
 cout<<"Enter number in Octal form = ";
 cin>>setbase(8)>>num;
 cout<<"Value of number in decimal form = "<<setbase(10)<<num<<endl;
 cout<<"Value of number in octal form = "<<setbase(8)<<num<<endl;
 cout<<"Value of number in hexadecimal form = "<<setbase(16)<<num;
 return 0;
}
Output
Enter numberin Octal form = 21
Value of number in decimal fonn = 17
Value of number in octal fonn = 21
Value of number in hexadecimal fonn = 11
```

setw(w) Manipulator

The setw() stands for set width. It is a function that takes a single integer argument which specifies the amount of space used to display the required value. We typically use the setw() manipulator for displaying output so that it becomes more understandable. It can be used to format only one value at a time.

```cpp
#include<iostream.h>
#include<iomanip.h>
#include<conio.h>
int main() {
 int age = 22,rollno = 9101;
 cout<<setw(12)<<"My Rollno is"<<setw(8)<<rollno<<endl;
 cout<<setw(12)<<"My Age is"<<setw(8)<<age;
 getch();
 return 0;
}
```

M	y		R	o	l	l	n	o		i	s					9	1	0	1
			M	y		A	g	e		i	s							2	2

setfill(c) Manipulator

The setfill() manipulator is used in conjunction with the setw() manipulator. The compiler leaves the empty spaces on the left side of the required value if the set width is greater than the needed space. If you wish to fill the blank space with an alternative character instead of a blank space, you can use the setfill () manipulator.

The setfill () manipulator takes a single character as an argument and fills the empty spaces with the specified character c on the left of the value displayed if the width specified using setw() manipulator is greater than the value to be displayed.

```
#inclucle<iostream.h>
#include<iomanip.h>
#include<conio.h>
int main() {
 int age = 22,rollno = 9101; cout<<setfill('#');
 cout<<setw(4)<<age<<setw(6)<<rollno<<endl;
 cout<<setw(6)<<age<<setw(8)<<rollno;
 getch();
 return 0;
}
Output :
##22##9101
####22####9101
```

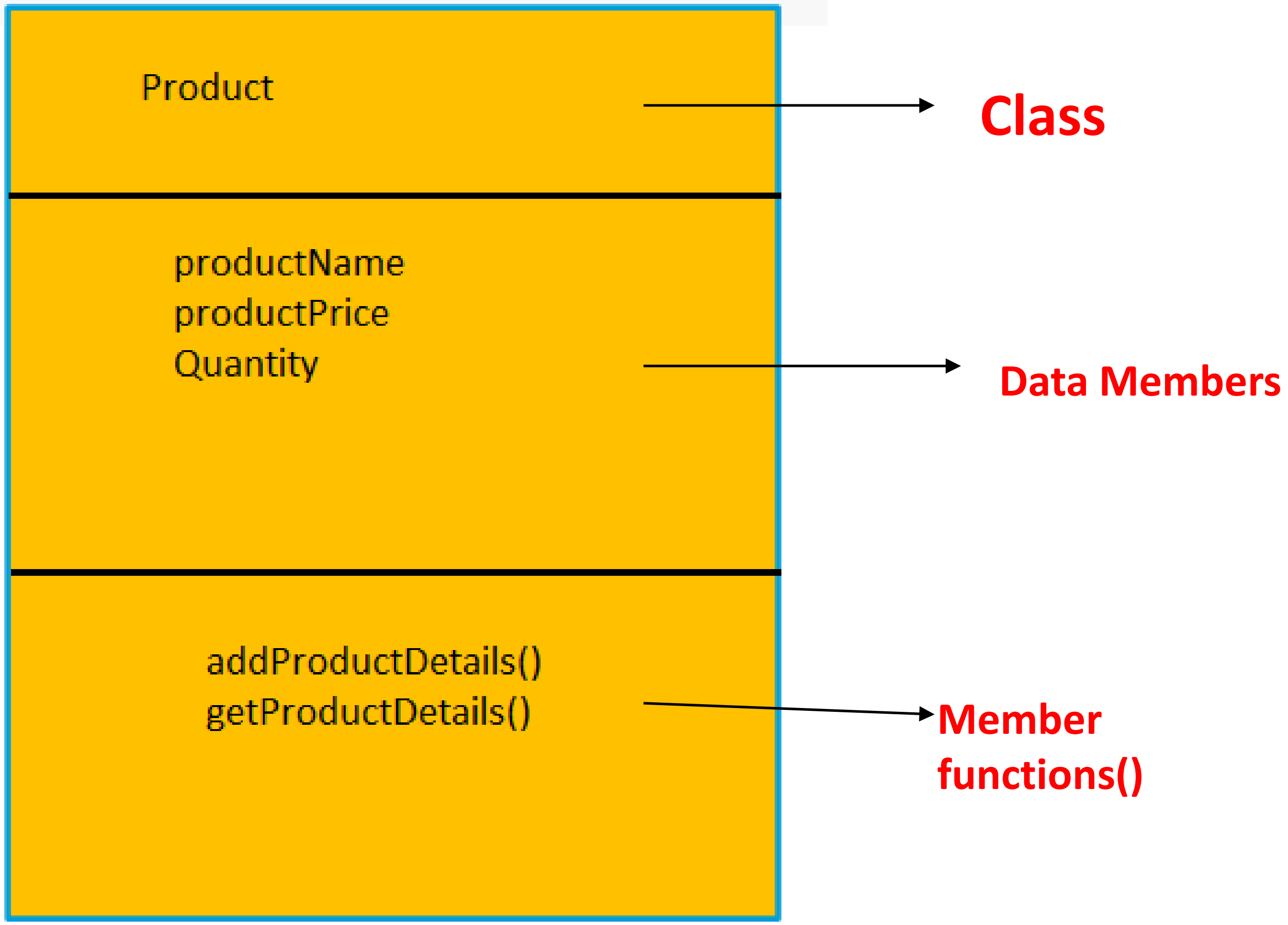
setprecision(n) Manipulator

The setprecision() manipulator is used to control the precision of floating-point numbers, i.e. the number of digits to the right of the decimal point. By default, the precision of the floating-point number displayed is 6. This precision can be modified by using a setprecision () manipulator. This function takes an integer argument n that specifies the number of digits displayed after the decimal point. The floating-point number will be rounded to the specified precision.

```
#include<iostream.h>
#includi<iomanip.h>
#inelude<eonio.h>
int main() {
 float a = 129.455396;
 cout<<setprecision(2)<<a<<endl;
 cout<<setprecision(3)<<a;
 getch();
 return 0;
}
Output :
129.46
129.455
```

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.

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));
```

```
#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.

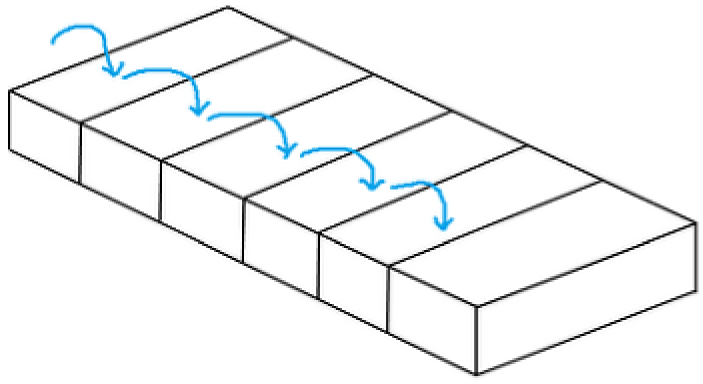
Nothing will be happen

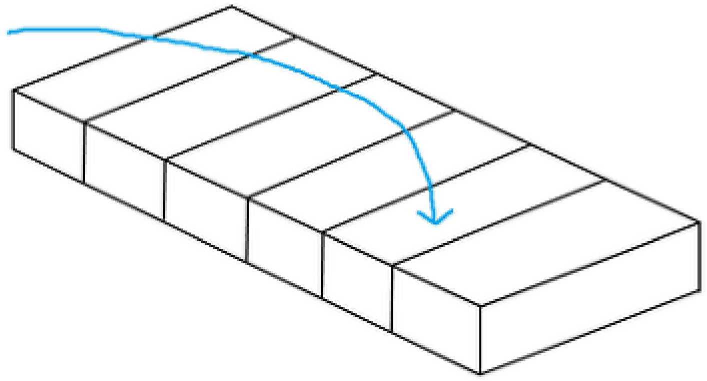
D.

None

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

ELEVATOR

- 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