The Shirpur Education Society's
R. C. Patel College of Engineering & Polytechnic, Shirpur

Course Title - Object Oriented Programming Using C++

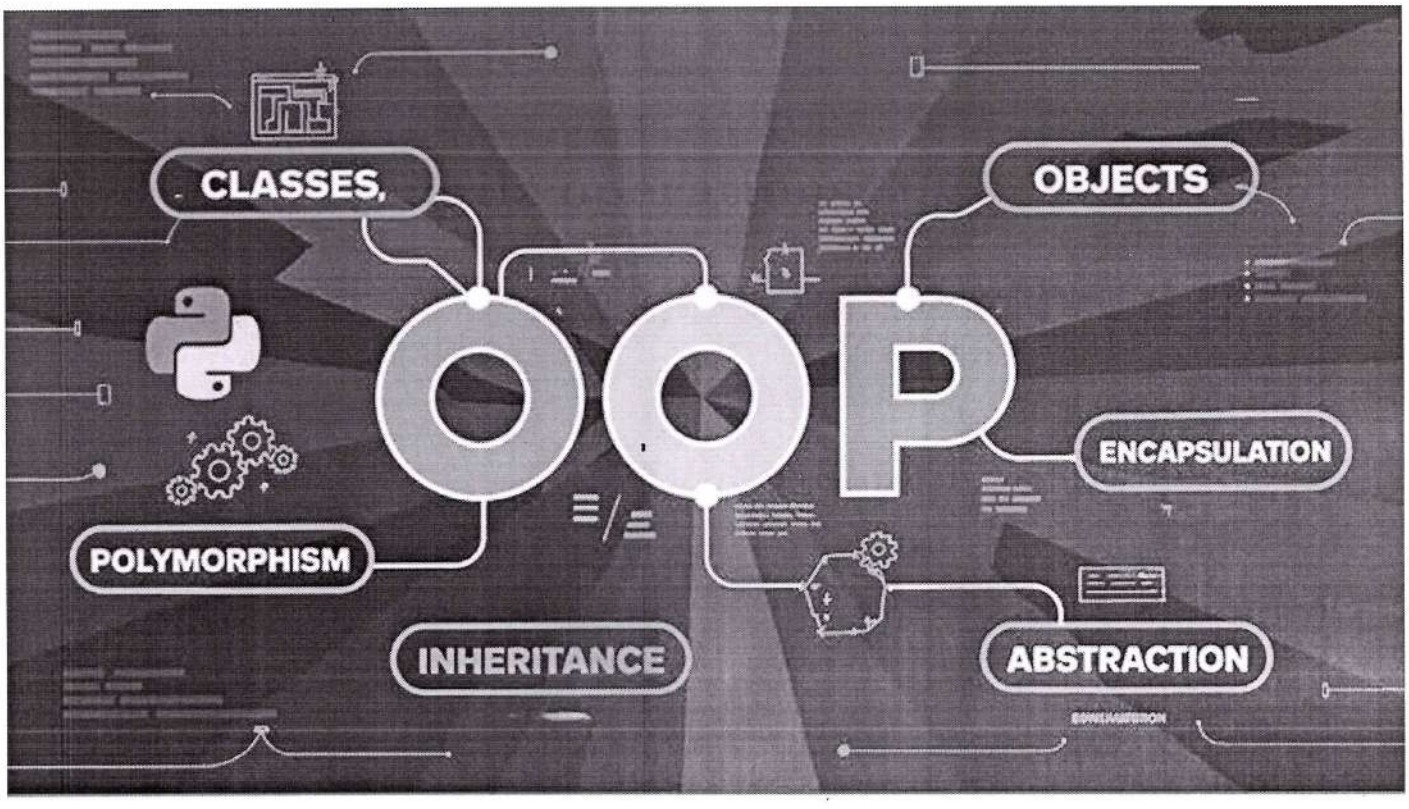
Course Code - 313304

Programme Name - Computer & Computer Science Engineering

Semester – Third

UNIT V: File operations (Marks 06)

Prepared By Mr.Shekar P. Chavan



5.1 C++ Stream classes, Classes for file stream operations

⊛ C++ Stream :- (stream is sequence of bytes),

→ Stream is a flow of data (either input or output)

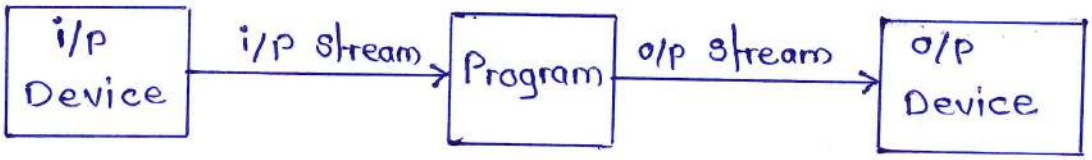
→ It acts as a source from which i/p data can be obtained or as a destination to which the input data can be sent.

→ input stream :- data to program.

→ output stream :- output from program.

→ Stream classes are defined in the iostream header.

→ These classes help perform keyboard input, console output and file operations.



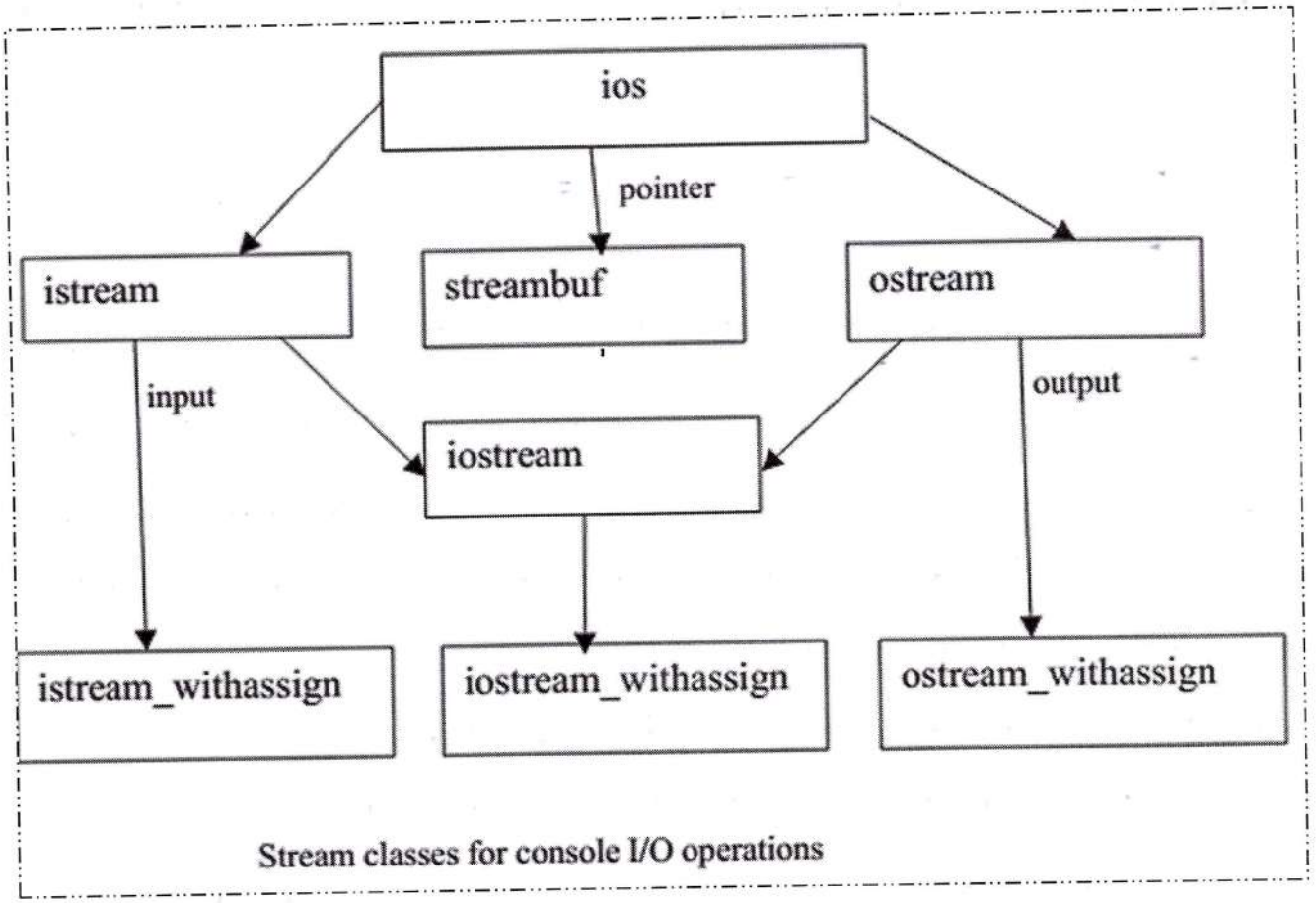
⊛ C++ Stream Classes

→ At the top level, the ios class is the base for all i/p / o/p classes.

→ istream and ostream are derived from ios.

→ iostream is derived from both. istream and ostream.

ios → base class for all stream classes
istream → for input operations (eg. cin)
ostream → for output operations (eg. cout, cess, clog).
iostream → for both input and output.



Stream classes for console I/O operations

Class name	Contents
ios(General input/output stream class)	Contains basic facilities that are ued by all other input and output classes Also contains a pointer to buffer object(streambuf object) Declares constants and functions that are necessary for handling formatted input and output operations
istream(input stream)	Inherits the properties of ios Declares input functions such as get(),getline() and read() Contains overloaded extraction operator>>
ostream(output stream)	Inherits the property of ios Declares output functions put() and write() Contains overloaded insertion operator <<
iostream (input/output stream)	Inherits the properties of ios stream and ostream through multiple inheritance and thus contains all the input and output functions
streanbuf	Provides an interface to physical devices through buffer

⑧ Unformatted I/O Operations :-

i) Put() and get() functions.

→ The classes istream & ostream define two members functions get() & put() respectively, to handle the single character i/o operations.

→ There are two prototypes of get() function.

ⓘ get (char *)

ⓘⓘ get (void).

→ These functions are member of i/p or o/p stream classes, must invoke them using correct object.

```
eg. char c;
    cin.get(c) // read a character
    while (c != '\n')
    { cout.get(c); // display character on screen.
    }
```

ii) getline & write() functions :-

→ getline() function reads a whole line of text that ends with a newline character.

→ This function can be invoked by using the object cin as follow,

```
eg. cin.getline (line, size);
```

→ Write() function display the entire line.

```
eg. cout.write (line, size);
```

Example :-
```
# include <iostream.h>
void main()
{ int xi = 20;
  char name[20];
```

```
cout<<"enter name of student";
cin.getline (name,size);
cout<<"enter name of student is";
cout.write (name,size);
getch();
}.
```

⑧ Formatted console I/O operations :-

→ The ios class contains a large no. of member function

i) width() – specify required field size.
ii) precision() – specify no. of digit.
iii) fill() – used to fill unused portion.
iv) Self() → specify format flags.
v) unself() → clear the flags.

Manipulators in c++ are special functions that modify the input / output stream without changing actual values of the variables.

i) setw() :- Sets the field width to n characters.
ii) setfill() :- Sets the fill character to c for padding.
iii) set precision :- Specify no. of digit to be display.
iv) setiosflags() :- To specify format flags that can control output.
v) mesetiosflags() :- To clear the flags.
vi) endl – insert newline character.
vii) ends – insert a null character into o/p stream.

→ Manipulators are accessed by including the <iomanip> header file in the program.

5.3 ⊛ opening file :-

i) using constructor

→ file is opened at the time of object creation.

Syntax :-

```
ifstream fin ("filename.txt");
ofstream fout ("filename.txt");
```

→ simple and direct method.

→ useful when file name known in advance.

ii) Using .open() member function

→ file is opened after creating the stream object.

→ syntax :-

```
fin.open ("filename.txt");
```

→ useful when file name is input at runtime

→ must check if the file opened successfully using if (!fin) or fin.fail().

eg.
```
ifstream fin ("data.txt"); // using constructor
ofstream fout;
fout.open ("output.txt"); // using open()
```

⊛ Close file :-

→ c++ program terminates, it automatically flushes the stream, release the alloted memory & close all the opened files.

→ fstream, ofstream & ifstream objects have the close() for closing files.

→ syntax :-

```
void close();
```

⊛ Classes for File stream operations :-

- File operations in C++ are performed using fstream classes
- These classes are defined in the fstream header.

File Stream classes

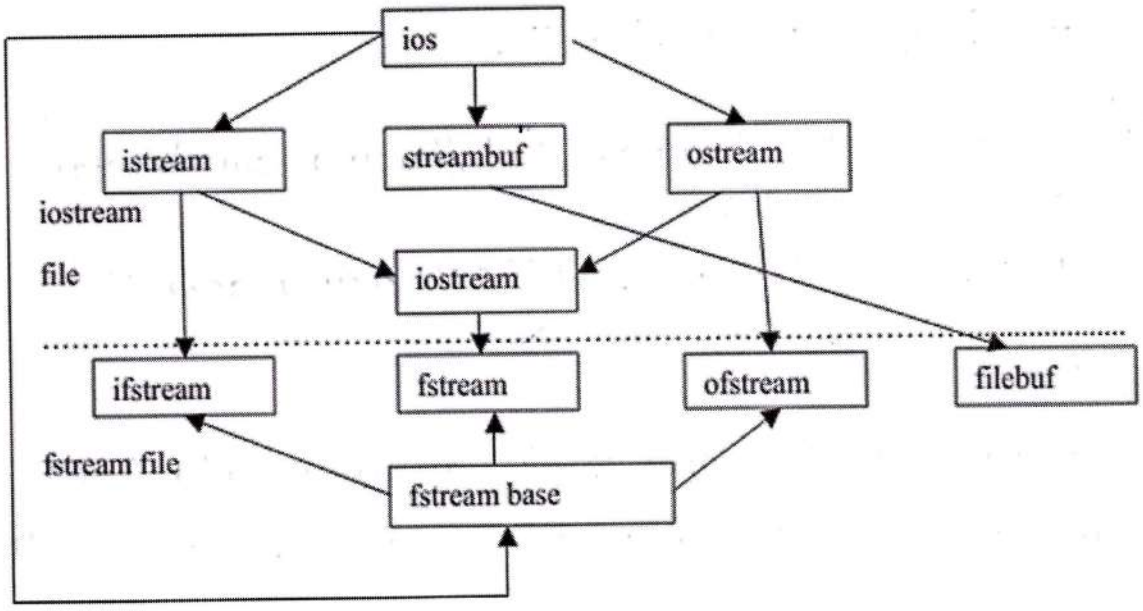
i) ifstream → input file stream
→ used for reading from files

ii) ofstream → output file stream
→ used for writing to files.

iii) fstream → Supports both reading and writing.

iv) filebuf → Set the file buffers to read & write, open() & close()

v) fstreambase → provides operations common to the file stream.



Stream classes for file operations

⊛ Reading a file :-

→ Read information from file into program using stream extraction operator (>>)

→ ">>" operator used to input information from keyboard.

→ eg.
```
ifstream inf;
inf >> data;
```
where inf is an object of ifstream.

⊛ Writing a file :-

→ To write a file (<<) operator is used.

eg.
```
ofstream ofs;
ofs << data << endl;
```

eg. Reading from file

```
#include <fstream>
#include <iostream>

int main() {
 string text;
 ifstream infile("ext.txt");
 while (getline (infile, text))
  { cout << text << endl;
  }
 infile.close();
 return 0;
}
```

eg. Writing to a file.

```
# include <fstream>

int main () {
ofstream outfile ("abc.txt");
outfile << "Hello file";
out file.close();
  return 0;
}
```

5.2 Detection of end of file, Files modes :-

① Detection of End of File :-

→ when a program read data from a file, it must know where the file ends.

→ EOF indicates that there is no more data available in the file.

→ EOF is a condition that occurs when all data from a file has been read.

→ C++ provides eof() function to check whether the end of a file has been reached.

Syntax :-

```
file_object.eof();
```

Value

0 (false) → end of file not reached.
1 (true) → end of file reached.

→ eof is a member function of ios class.

eg.

```
int main()
{ ifstream fin("abc.txt");
  char ch;
while (!fin.geteof())
 { fin.get(ch);
   cout<<ch;
 }
 fin.close();
 return 0;
}
```

if the file contains

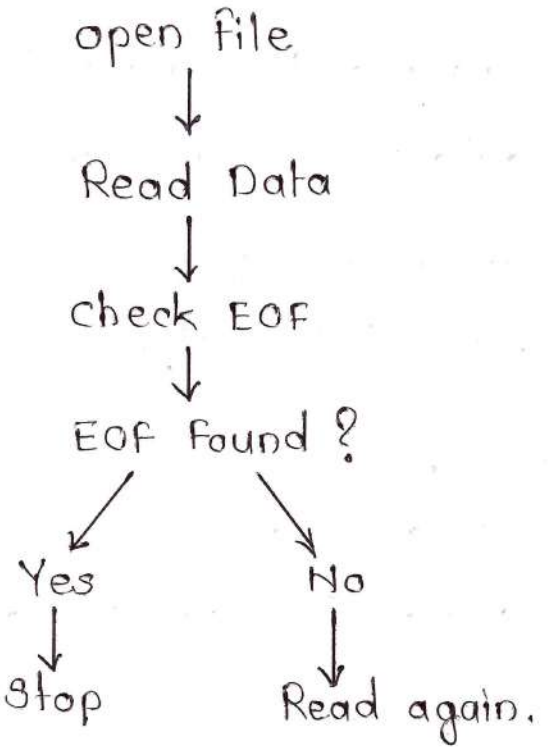
Welcome to C++

then

output of program is

Welcome to C++.

Flow of EOF Detection



⊛ File Modes :-

→ File modes in c++ define how file should be opened - for reading, writing, appending, binary access etc.

→ File modes are defined in the ios class.

→ File modes in c++.

i) ios::in → opens a file for reading.
→ used with ifstream.
→ eg. ifstream fin;
fin.open ("student.txt", ios::in);

ii) ios::out → opens a file for writing
→ used with ofstream
→ flow (program → file)

eg. ofstream fout;
fout.open("abc.txt", ios::out);

iii) ios::app → appends data at the end of file
→ existing data remains unchanged.
→ eg. ofstream fout;
fout.open("abc.txt", ios::app);

iv) ios::ate → moves file pointer to the end immediately after opening.
→ Reading and writing can still occur.
→ eg. fstream file;
file.open("abc.txt, ios::ate);

v) ios::trunc → Deletes existing contents of file.
→ used mainly with output mode.
→ eg. ofstream fout;
fout.open("abc.txt", ios::trunc);

vi) ios::binary → open file in binary mode.
→ used for images, audio, video & binary data
→ eg. fstream file
file.open("abc.txt", ios::binary);

(more than one mode can be used together using OR operator (|).

eg. file.open("abc.txt, ios::in | ios::out);

5.4. Types of File : Random access, Sequential access :-

→ File access refers to how data is read from or written to a file

→ There are two types
  i) Sequential access
  ii) Random access

① Sequential File access :-

→ Data is accessed in the same order as it is stored in the file.

→ Reading or writing starts from the beginning & goes line by line or record by record.

→ Most used for text files, reports, logs etc.

→ Cannot skip or jump directly to a specific part of the file.

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

int main() {
 ifstream file("data.txt");
 string line;

while (getline (file, line)) {
 cout << line << endl;
 }
 file.close();
 return 0;
}
```

(11) Random Access file :-

→ Data can be read or written at any position in the file.

→ useful for accessing large files quickly without reading the whole file.

→ Works mainly with binary files or fixed length records

→ supported using file pointers & functions like seekg(), seekp(), tellg() & tellp().

seekg() — moves the get (read) pointer
seekp() — moves the put (write) pointer
tellg() — returns curent position of the get pointer
tellp() — returns current position of the put pointer

eg.

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

int main() {
fstream file("abc.dat", ios::in|ios::out|ios::binary);
    file.seekp(5);
    file.put('x');

    file.seekg(5);
    char ch;
    file.get(ch);
cout<<"character at position 6:"<<ch<<endl;

    file.close();
    return 0;
}
```

Difference Between Sequential Access File and Random Access File

Sr. No.	Sequential Access File	Random Access File
1	Data is accessed in a sequence, one record after another.	Data can be accessed directly from any location in the file.
2	Previous records must be read before accessing a desired record.	Any record can be accessed without reading previous records.
3	Access speed is slower.	Access speed is faster.
4	File pointer moves sequentially from start to end.	File pointer can jump directly to any position.
5	Suitable for processing all records one by one.	Suitable for retrieving or updating specific records.
6	Easy to implement and manage.	More complex than sequential access.
7	Commonly used in text files, logs, and reports.	Commonly used in databases, banking systems, and student record systems.
8	Example: Reading a file line by line.	Example: Accessing the 10th record directly using `seekg()` or `seekp()`.

Difference Between Formatted and Unformatted I/O

Sr. No.	Formatted I/O	Unformatted I/O
1	Data is read or written in a specified format.	Data is read or written as it is without any format.
2	Uses formatting operators >> and <<.	Uses functions like `get()`, `put()`, `getline()`, `read()`, and `write()`.
3	Suitable for numbers, strings, and structured data.	Suitable for character-by-character or raw data processing.
4	Ignores white spaces, tabs, and new lines while reading input.	Can read white spaces, tabs, and new line characters.
5	Easier to display data in a user-friendly format.	Gives direct control over input and output operations.
6	Slower due to formatting overhead.	Faster because no formatting is performed.
7	Examples: `cin >> x;`, `cout << x;`	Examples: `cin.get(ch);`, `cout.put(ch);`

Write Program to append data from abc.txt to xyz.txt file

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

int main()
{
ifstream fin("abc.txt"); // Open source file
ofstream fout("xyz.txt", ios::app); // Open destination in
                                    append mode
char ch;

while(fin.get(ch))
{
fout.put(ch);
}

cout<<"Data appended successfully.";

fin.close();
fout.close();

return 0;
}
```

Write a program that copies contents of one file into another file. (6 Marks, W-24, S-25)

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

int main()
{
   ifstream fin("source.txt");
   ofstream fout("target.txt");

   char ch;

   while(fin.get(ch))
   {
      fout.put(ch);
   }

   fin.close();
   fout.close();

   cout<<"Contents copied successfully.";

   return 0;
}
```

Write " Welcome to Poly" in a file then read data from file & display it on screen.

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

int main()
{
ofstream fout("data.txt");

fout<<"Welcome to Poly";

fout.close();

ifstream fin("data.txt");

char str[50];

fin.getline(str,50);

cout<<"Data from file: "<<str;

fin.close();

return 0;
}
```

Write a program for closing a file. (4 Marks, S-23)

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

int main()
{
   ofstream fout("data.txt");

   fout << "Welcome to Polytechnic";

   fout.close();    // Closing the file

   cout << "File closed successfully.";

   return 0;
}
```

Program for Count number of lines in File

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

int main()
{
ifstream fin("data.txt");

string line;
int count = 0;

while(getline(fin, line))
{
count++;
}

cout << "Number of lines = " << count;

fin.close();

return 0;
}
```

Write a program for get() and put() functions. (4 Marks, S-23)

```
#include<iostream.h>
#include<fstream.h>
using namespace std;
int main()
{
   ofstream fout("data.txt");
   fout.put('H');
   fout.put('E');
   fout.put('L');
   fout.put('L');
   fout.put('O');
   fout.close();

   ifstream fin("data.txt");
   char ch;
   while(fin.get(ch))
   {
      cout << ch;
   }
   fin.close();
   return 0;
}
```

27/6/26.

27/6/26

OOP Question Bank

Unit – V: File Operations

2 Marks Questions

1. Define file with its operations. **(2 Marks, S-23)**
2. Give the syntax and use of fclose() function. **(2 Marks, W-23, S-24)**

---

4 Marks Questions

3. Write a program for get() and put() functions. **(4 Marks, S-23)**
4. Write a program for closing a file. **(4 Marks, S-23)**
5. Write a C++ program for writing into a file using file operations. **(4 Marks, W-23, S -24)**
6. Write a C++ program to copy data from one file to another. **(4 Marks, W-23, S -24)**
7. Explain different file opening modes with suitable example. **(4 Marks, W-24, W-25)**
8. Explain how file can be opened using constructor with suitable example. **(4 Marks, W-24)**
9. Write a program to read and write data in file. **(4 Marks, W-25)**
10. Develop a C++ program to check detection of End Of File (EOF). **(4 Marks, S-25)**
11. Explain different unformatted input/output functions in file. **(4 Marks, S-25)**

---

6 Marks Questions

12. Write a program that copies contents of one file into another file. **(6 Marks, W-24, S -25)**
13. Explain types of files in file handling (Sequential and Random Access). **(6 M, W-25)**
14. Write a program to count number of lines in a file. **(6 Marks, W-25)**
15. Explain the process of opening and closing a file using open() and close() functions with suitable C++ example. **(4 Marks, Expected Question)**