Object-oriented Programming
Lab 07
Order of constructor call, Console I/O operations and File management
Version 1.0

Order of constructor call and virtual base class

The order of initializing class is following:

1. Constructors of Virtual base classes are executed, in the order that they appear in the base list.
2. Constructors of nonvirtual base classes are executed, in the declaration order.
3. Constructors of class members are executed in the declaration order (regardless of their order in the initialization list).
4. The body of the constructor is executed.

Program:

```
#include <iostream>
class A
{
public:
    A() { std::cout << "A" ; }
};

class B: public A
{
public:
    B() { std::cout << "B" ; }
};

class C: virtual public A
{
public:
    C() { std::cout << "C" ; }
};

class D: public B, public C
{
public:
    D()  { std::cout << "D" ; }
};

int main()
{
    D d;
    return 0;
}
```

Output:

AABCD

This makes sense, as virtual base class is constructed before non-virtual base classes. So in your case it is: virtual A, non-virtual A, BCD. If you change the inheritance order it is virtual A, C, non-virtual A, BD.

Formatted Console I/O Operations:

C++ supports a number of features that could be used for formatting the output. These features include:

1. ios class functions and flags
2. Manipulators
3. User-defined output functions

The following table shows the ios format functions and their equivalent manipulators.

Functions	Equivalent Manipulators	Task
width()	setw()	To specify the required field size for displaying an output value
precision()	setprecision()	To specify the number of digits to be displayed after the decimal point of a float value
fill()	setfill()	To specify a character that is used to fill the unused portion of a field
setf()	setiosflags()	To specify format flags that can control the form of output display (such as left or right justification)
unsetf()	resetiosflags()	To clear the flags specified

To access these manipulators, the file iomanip should be included in the program. To see the definitions of these above functions and manipulators you have to read the text book(s). Now analyze the following program yourself:

```
#include<iostream>
#include<cmath>
using namespace std;
int main()
{
      cout.fill('*');
      cout.width(10);
      cout.precision(4);
      cout.setf(ios::left,ios::adjustfield);
      cout<<sqrt(2);
      cout.width(10);
      cout<<1234<<"\n";
      return 0;
}
```

The above program gives output like **1.414*****1234******.** If we do not use the statement cout.setf(ios::left,ios::adjustfield); The output will be *******1.414******1234**

File management:

A file is a collection of related data stored in a particular area on the disk. Many real life problems handle large volumes of data and, in such situations, we need to use the concept of files.

The I/O system of C++ contains a set of classes that define the file handling methods. These include **ifstream**, **ofstream**, and **fstream**. These classes are contained in the header file *fstream*.

Opening and Closing a file:

A file can be opened in two ways:

1. Using constructor function of the class.
2. Using member function open() of the class.

The first method is useful when we use only one file in the stream whereas the second method is useful to manage multiple files.

In closing a file, we can simply use member function close() of the class.

Opening files using constructor:

We know that a constructor is used to initialize an object while it is being created. Here a file name is used to initialize the file stream objects. Let us see a program of creating files with constructor functions:

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

class X{
    int cost;
    char name[20];
    public:
    void file_write()
    {
    ofstream outf("item.txt");
      cout<<"item name:";
      cin>>name;
      outf<<name<<"\n";

      cout<<"item cost:";
      int cost;
      cin>>cost;
      outf<<cost<<"\n";
      outf.close();
    }
    void file_read(){
      ifstream inf("item.txt");
```

```
      inf>>name;
      inf>>cost;
      cout<<"\n";
      cout<<"item:"<<name<<"\n";
      cout<<"expenditure:"<<cost<<"\n";
      inf.close();
    }

};
int main()
{
      X ob1;
      ob1.file_write();
      ob1.file_read();
      return 0;
}
```

In the above program, the file stream objects outf and inf are created for writing to and reading from file item.txt respectively. The item.txt file from outf is disconnected by using the following statement:

outf.close();

And the item.txt file from inf is disconnected by using the following statement:

inf.close();

Opening files using open():

As stated earlier, the function open() can be used to open multiples files that use the same stream object. This is done as follows:

file-stream-class stream-object;

stream-object .open("filename");

Example:

ofstream outfile;

outfile.open("Data1.txt");

……..

……..

outfile.close();

outfile.open("Data2.txt");

……..

……..

outfile.close();

……..

……..

Let us see a program of creating files with open():

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

class X{
    public:
    void file_write()
    {
    ofstream fout;
    fout.open("country.txt");
    fout<<"USA\t";
    fout<<"UK";
    fout.close();
    fout.open("capital.txt");
    fout<<"Washington\t";
    fout<<"London\n";
    fout.close();
    }
    void file_read(){
    char line[50];
    ifstream fin;
    fin.open("country.txt");
    cout<<"The contents of the country file:\n";
    while(fin)
    {
        fin.getline(line,50);
        cout<<line;

    }
    fin.close();
    fin.open("capital.txt");
    cout<<"\nThe contents of the capital file:\n";
    while(fin)
    {
        fin.getline(line,50);
        cout<<line;

    }
    fin.close();
    }
};
int main()
{
```

```
        X ob1;
        ob1.file_write();
        ob1.file_read();
        return 0;
}
```

Detecting end-of-file:

Detection of the end-of-file condition is necessary for preventing any further attempt to read data from the file.

Suppose, the following statement

```
while(fin)
{
   }
```

An **ifstream** object, such as **fin**, returns a value of 0 if any error occurs in the file otherwise non-zero value.

There is another approach using **eof()** member function of **ios** class. It returns a non-zero value when end of file is encountered, and zero, otherwise. This can be done using the following statement:

```
if(fin.eof()!=0)
{
……..
…….
exit(1);
}
```

File modes using open():

The general form of the function open() with two arguments is:

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

The second argument specifies for which the file is opened. By default ios :: in for reading only and ios :: out for writing only.

Home Task

A file contains a list of roll numbers of your classmates in the following form:

Ahmed 1907001

John 1907002

and so on. The name contains only one word and the name and roll numbers are separated by white spaces. Write a program to read the file and output the list in two columns.