Lab No.7 Dynamic Memory Allocation and miscellaneous topics

7.1 Objectives of the lab:

This lab covers some miscellaneous topics such as

27 Dynamic memory allocation
28 Copy Constructor (Deep & Shallow copy)
29 Const members and objects
30 Cascaded function calls
31 Static data members, member functions, and objects

7.2 In-Lab

7.2.1 Activity

Study the following programs, execute them, and determine what possibly the cause of error. Write the additional code to make these programs run.

PROGRAM 1

```
#include <iostream.h>
#include <string.h>

class student
{
private:
        char    *name;
        int     roll;
        int     semester;
public:
        student(char *n, int r, int s): roll(r), semester(s)
        {
                name=new char[strlen(n) + 1];
                strcpy(name, n);
        }
        void set()
        {
                cout<<"Enter name: "<<endl;
                cin>>name;
                cout<<"Enter roll no: "<<endl;
                cin>>roll;
                cout<<"Enter semester: "<<endl;
                cin>>semester;
        }

        void show()
        {
                cout<<"Name: "<<name<<endl;
                cout<<"Roll NO: "<<roll<<endl;
                cout<<"Semester: "<<semester<<endl;
        }

        ~student()
        {
                delete[] name;
        }

};

void main()
{
        student s1("Bjarne Stroustrup", 3, 3);
        s1.show();

        {
                student s2=s1;
                s2.show();
        }
        s1.show();
}
```

PROGRAM 2

```
#include <iostream.h>
#include <string.h>
class student
{
private:
        char    *name;
        int     roll;
        int     semester;
public:

        student(): roll(0),    semester(0)
        {
                name=new char[20];
                strcpy(name, "");
        }

        student(char *n, int r, int s): roll(r), semester(s)
        {
                name=new char[strlen(n) + 1];
                strcpy(name, n);
        }


        void set()
        {
                cout<<"Enter name: "<<endl;
                cin>>name;
                cout<<"Enter roll no: "<<endl;
                cin>>roll;
                cout<<"Enter semester: "<<endl;
                cin>>semester;
        }

        void show()
        {
                cout<<"Name: "<<name<<endl;
                cout<<"Roll NO: "<<roll<<endl;
                cout<<"Semester: "<<semester<<endl;
        }

        ~student()
        {
                delete[] name;
        }

};

void main()
{
        student s1("Bjarne Stroustrup", 3, 3);
        s1.show();

        student s2(s1);
        s2.show();
        s2.set();
        cout<<"After setting s2:"<<endl;
        cout<<"Data of student S1"<<endl;
        s1.show();

        cout<<"Data of student s2"<<endl;
        s2.show();
}
```

7.2.2 Activity

Create a class called **student**. This class contains data members for name, roll no, and CGPA of a student.

1. Provide a **no-argument constructor** for initializing the data members to some fixed value.

2. Provide a **2-argument constructor** to initialize the data members to the values sent from the calling function.

3. Provide separate **setter** functions for setting each data member. These functions should take the values from user at run-time.

4. Provide separate **getter** functions for each data member. The getter functions should return the value of the corresponding fields.

5. Create a function **display** that displays all the information to user.

Let us suppose that we want to keep information about average CGPA of students in a particular department. Make appropriate changes in the class to handle this extra information (**Hint**: provide **static** data members for average CGPA and no of students and set the values for these members in constructor). Provide a **static** function to display this additional information.

7.3 Home-Lab

7.2.3 Activity

Create a class called **employee**. This class maintains information about name (**char***), department (**char***), salary (**double**), and period of service in years (**double**).

1. Provide a **no-argument constructor** to initialize the data members to some fixed values.

2. Provide a **4-argument constructor** to initialize the members to values sent from calling function.

   (You have to make dynamic allocation for both name and department data members in constructor.)

3. Provide a **copy-constructor** that performs the deep copy of the data members.

4. Provide an **input** function that takes all the values from user during run-time.
5. Provide a **display** function that displays all the information about a specific student to user.
6. Provide a **destructor** to free the memory allocated to name and department in constructor.

   Write a driver program to test the functionality of the above-mentioned class.

7.2.4 Activity

Create a class called **Complex** for performing arithmetic with complex numbers. Complex numbers have the form realPart + imaginaryPart * i

where i is $\sqrt{-1}$

Write a program to test your class. Use floating-point variables to represent the private data of the class.

1. Provide a **constructor** that enables an object of this class to be initialized when it is declared.
2. Provide a **no-argument cons**tructor with default values in case no initializers are provided.
3. **Print** Complex numbers in the form (a, b), where **a** is the real part and **b** is the imaginary part.
4. Provide separate **setter** functions for setting the real and imaginary portions of this class.
5. Provide separate **getter** functions for both the fields that must return the values of real and imaginary parts of a complex number.

Note: Write the setter functions as well as the print function in a manner that allows for the cascaded calls of these functions.

Write a driver program that tests the functionality of this class.

7.4 References:

18 **Class notes**

19 **Object-Oriented Programming in C++ by** ***Robert Lafore*** **(Chapter 6)**

20 **How to Program C++ by** ***Deitel & Deitel*** **(Chapter 7)**