Lab Manual

CS112L – Object Oriented Programming Lab
Lab No: **08**
Topic: **Friend Functions and Classes, Composition vs Aggregation**

Class: **BSGM**
Semester: **II**
Session: **Spring, 2022**
Instructor: Ms. Saira Qamar

Lab Date: **April 12th, 2022**
Lab Time: **10:40hrs – 12:45hrs**

Air University Islamabad
FACULTY OF COMPUTING & ARTIFICIAL INTELLIGENCE

Faculty of Computing and AI

Instructions

Submission: Use proper naming convention for your submission file. Name the submission file as **LabNO_ROLLNUM (e.g. Lab01_00000).** Submit the file on Google Classroom within the deadline. Failure to submit according to the above format would result in deduction of 10% marks. Submissions on the email will not be accepted.

Plagiarism: Plagiarism cases will be dealt with strictly. If found plagiarized, both the involved parties will be awarded zero marks in the assignment, all of the remaining assignments, or even an F grade in the course. Copying from the internet is the easiest way to get caught!

Deadline: The deadlines to submit the assignment are hard. Late submission with marks deduction will be accepted according to the course policy shared by the instructor. Correct and timely submission of the assignment is the responsibility of every student; hence no relaxation will be given to anyone.

Comments: Comment your code properly. Bonus marks (maximum 10%) will be awarded to well comment code. Write your name and roll number (as a block comment) at the beginning of the solution to each problem.

Tip: For timely completion of the assignment, start as early as possible. Furthermore, work smartly - as some of the problems can be solved using smarter logic.

1. Note: Follow the given instructions to the letter, failing to do so will result in a zero.

Objectives

In this lab, you will learn:

- Friend Functions and
- Friend Classes
- Composition vs Aggregation

Concepts

1. Friend Functions:

Friend functions provide a relaxing mechanism through which we can **access the private data members of a class directly** without any question being asked. This does not mean that we have relaxed the access specifiers. A **friend function** provides **access to the private and public data members of a class** from only within its body. Friend functions are needed because sometimes if we have multiple classes and we want to manipulate the class members through a single function. This can be achieved through the following **syntax**:

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

class Distance {
    private:
        int meter;

        // friend function
        friend int addFive(Distance);

    public:
        Distance() : meter(0) {}

};

// friend function definition
int addFive(Distance d)
{

    //accessing private members from the friend function
    d.meter += 5;
    return d.meter;
}

int main()
{
    Distance D;
    cout << "Distance: " << addFive(D);
    return 0;
}
```

Output:

```
Distance: 5
--------------------------------
```

Here, **addFive()** is a friend function that can access both **private** and **public** data members. Though this example gives us an idea about the concept of a friend function, it **doesn't** show any meaningful use. A **more meaningful use would be operating on objects of two different classes**. That's when the friend function can be very helpful.

2. Friend Classes:

A friend function has access to those classes with which it is friend. Sometimes **a situation arises when we want to make two classes friends with each other**. This **means** that a **class can access the public and private members of another class directly**. To achieve this you must right the friend class statement in the befriending class.

```
// Add members of two different classes using friend functions
#include <iostream>
using namespace std;
// forward declaration
class ClassB;
class ClassA {
    public:
        // constructor to initialize numA to 12
        ClassA() : numA(10) {}
    private:
        int numA;
        // friend function declaration
        friend int add(ClassA, ClassB);
};
class ClassB {
    public:
        // constructor to initialize numB to 1
        ClassB() : numB(1) {}
    private:
        int numB;
        // friend function declaration
        friend int add(ClassA, ClassB);
};
// access members of both classes
int add(ClassA objectA, ClassB objectB) {
    return (objectA.numA + objectB.numB);
}

int main() {
    ClassA objectA;
    ClassB objectB;
    cout << "Addition: " << add(objectA, objectB);
    return 0;
}
```

Output:

```
Addition: 11
--------------------------------
```

In this program, **ClassA** and **ClassB** have declared **add()** as a friend function. Thus, this function can access **private** data of both classes.

3. Composition Vs. Aggregation:

Aggregation is type of association representing **weak relationship**. In aggregation, contained object exists even after the release of an owning object. Hence, in aggregation, there exists ownership without life-cycle dependency. **For example**, if a company no longer exists even then employee of that will continue to exist.
Composition is a special type of aggregation representing **strong relationship** in which the life- cycle of the part is dependent on the whole. Existence of the part is directly dependent on the existence of the whole. **For example**, the relationship between vehicle and its engine. Engine is build and destroyed whenever a vehicle is build or destroyed, respectively.

1) Example of Composition

In this program, class X has one data member 'd' and two member functions 'set_value()' and 'show_sum()'. The set_value() function is used to assign value to 'd'. Theshow_sum() function uses an integer type parameter. It adds the value of parameter with the value of 'd'and displays the result o the screen.Another class Y is defined after the class x. The class Y has an object of class x that is the C++ Composition relationship between classes x and y. This class has its own member function print_result().

In the main() function, an object 'b' of class y is created. The member function set_value() of object 'a' that is the sub-object of object 'b' is called by using tw dot operators. One dot operator is used to access the member of the object 'b' that is object 'a', and second is used to access the member function set_value() of sub-object 'a' and 'd' is assigned a value 20.

In the same way, the show_sum() member function is called by using two dot operators. The value 100 is also passed as a parameter. The member function print_result of object 'b' of class Y is also called for execution. In the body of this function, the show_sum() function of object 'a' of class X is called for execution by passing value 5.

```cpp
#include <iostream>
using namespace std;
class X
{
    private:
        int d;
    public:
    void set_value(int k)
    {
        d=k;
    }
    void show_sum(int n)
    {
        cout<<"sum of "<<d<<" and "<<n<<" = "<<d+n<<endl;
    }
};
class Y
{
    public:
    X a;  // Object of class X
    void print_result()
    {
        a.show_sum(5);
    }
};
int main()
{
  Y b;
  b.a.set_value(10);
  b.a.show_sum(50);
  b.print_result();
  return 0;
}
```

Output:

```
sum of 10 and 50 = 60
sum of 10 and 5 = 15

--------------------------------
```

2) Example of Aggregation

Here Person has instance variable name which tells the name of the person and a pointer variable to address class object. Address class object has variables such as House, street, city, and state. Here we have 2 persons Sam and Seemaliving on the same address thus share the same address object add1.

```cpp
#include <iostream>
#include<string.h>
using namespace std;
class Address {
    public:
        int houseNo;
        string city;
    Address(int hno, string city)
    {
        this->houseNo = hno;
        this->city = city;
    }
};
class Person
{
    private:
        Address* address;
        string name;
    public:
    Person(string name, Address* address)
    {
        this->name = name;
        this->address = address;
    }
    void display()
    {
    cout<< name<< " is living in "<<address->houseNo<<" " <<address->city<< " "<<endl;
    }
};
int main(void) {
    Address add1= Address(145 ,"ABC Town");
    Person p1 = Person("Sam",&add1);
    Person p2 = Person("Seema",&add1);
    p1.display();
    p2.display();
    return 0;
}
```

Output:

```
Sam is living in 145 ABC Town
Seema is living in 145 ABC Town

--------------------------------
```

Lab Tasks

1. Pak-Wheel requested for a software and asked for the following functionalities.
- Client name, contact, email and transection (only one) are important to keep.
- Client can buy or sell car through pinwheels. The transection details which are important to keep are date (day), transection type (Buy or Sell) and amount.

Implement the above demand in such a way what if the client is deleted, none of its transection record will remain. (Use Composition)

2. Later they requested another functionality to be incorporated within the program:
- The car should have its owner detail (client) but if the car is gone or sold, the client should remain. (Use Aggregation)

3. Create 2 classes Employee (Name, Father Name, Contact, Email) and Job_Detail (Title, Salay, Id, JobDescription). Each employee will have job detail. Link these 2 classes via using Composition. Employee owns Job_Description and Job_Descriptioncannot exist without Employee.

4. Create a Subject and Student class. A student is enrolled in subject. Implement a scenario if the subject is cancelled students will remain intact. (Use Aggregation)

5. In a client class (Name, ClientId, Contact Address, Personal Email, Personal detail), Personal Email and Personal detail are private variable and a private function that prints the complete detail of the client.
