FAST NATIONAL UNIVERSTIY OF COMPUTER AND EMERGING SCIENCES, PESHAWAR

DEPARTMENT OF COMPUTER SCIENCE

CL217 – OBJECT ORIENTED PROGRAMMING LAB

Composition, Friend Function and Friend class in C++

LAB MANUAL # 10

Instructor: Iqra Rehman

SEMESTER SPRING 2024

C++ Composition

In real-life complex objects are often built from smaller and simpler objects. For example, a car is built using a metal frame, an engine some tires, a transmission system, a steering wheel, and a large number of other parts. A personal computer is built from a CPU, a motherboard, memory unit, input and output units, etc. even human beings are building from smaller parts such as a head, a body, legs, arms, and so on. This process of building complex objects from simpler ones is called c++ composition. It is also known as object composition.

So far, all of the classes that we have used had member variables that are built-in data type (e.g. int, float, double, char). While this is generally sufficient for designing and implementing small, simple classes, but it gradually becomes difficult to carry out from more complex classes, especially for those built from many sub-parts. In order to facilitate the building of complex classes from simpler ones, C++ allows us to do object C++ Composition in a very simple way by using classes as member variables in other classes.

A class can have one or more objects of other classes as members. A class is written in such a way that the object of another existing class becomes a member of the new class. this relationship between classes is known as C++ Composition. It is also known as containment, part-whole, or has-a relationship. A common form of software reusability is C++ Composition.

In C++ Composition, an object is a part of another object. The object that is a part of another object is known as a sub-object. When a C++ Composition is destroyed, then all of its subobjects are destroyed as well. Such as when a car is destroyed, then its motor, frame, and other parts are also destroyed with it. It has a do and die relationship.

Example # 01

How to use C++ Composition in programming

```
#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;
    void print_result()
    {
        a.show_sum(5);

    }

};

int main()
{
  Y b;
  b.a.set_value(20);
  b.a.show_sum(100);
  b.print_result();

}
/*
Output
sum of 20 and 100 = 120
sum of 20 and 5 = 25
*/
```

Programming Explanation:

- 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'. the show_sum() function uses an integer type parameter. It adds the value of parameter with the value of 'd' and displays the result on 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 two 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.

Example # 02

```
#include <iostream>
#include <string>

using namespace std;

class Birthday{

public:
    Birthday(int cmonth, int cday, int cyear){
        cmonth = month;
        cday = day;
        cyear = year;

    }
    void printDate(){
        cout<<month <<"/" <<day <<"/" <<year <<endl;

    }
private:
    int month;
    int day;
    int year;

};

class People{

public:
    People(string cname, Birthday cdateOfBirth)
    :name(cname),
    dateOfBirth(cdateOfBirth)
    {

    }
    void printInfo(){
        cout<<name <<" was born on: ";
        dateOfBirth.printDate();
    }

private:
    string name;
    Birthday dateOfBirth;

};


int main() {

    Birthday birthObject(7,9,97);
    People infoObject("Lenny the Cowboy", birthObject);
    infoObject.printInfo();

}
```

Friend Functions in C++

Friend functions of the class are granted permission to access private and protected members of the class in C++. They are defined globally outside the class scope. Friend functions are not member functions of the class. So, what exactly is the friend function?

A friend function in C++ is a function that is declared outside a class but is capable of accessing the private and protected members of the class. There could be situations in programming wherein we want two classes to share their members. These members may be data members, class functions. In such cases, we make the desired function, a friend to both these classes which will allow accessing private and protected data of members of the class.

Generally, non-member functions cannot access the private members of a particular class. Once declared as a friend function, the function is able to access the private and the protected members of these classes.

```
class class_name

{

  friend data_type function_name(arguments/s); //syntax of friend function.

};
```

Characteristics of Friend Function in C++

- The function is not in the 'scope' of the class to which it has been declared a friend.
- Friend functionality is not restricted to only one class
- Friend functions can be a member of a class or a function that is declared outside the scope of class.
- It cannot be invoked using the object as it is not in the scope of that class.
- We can invoke it like any normal function of the class.
- Friend functions have objects as arguments.
- It cannot access the member names directly and has to use dot membership operator and use an object name with the member name.
- We can declare it either in the 'public' or the 'private' section.

C++ program to demonstrate the working of friend function

```
#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
```

Accessing private data members of a class from the member function of another class using friend function

```
#include <iostream>
using namespace std;
//forward declaration
class Distance;
class Add {
        public:
                int addFive(Distance d);
};
class Distance {
        private:
                int meter;
                // friend function
        public:
                int getDistance()
                {
                        return meter;
                }
                Distance(int meter) {
                        this->meter=meter;
                }
                friend int Add::addFive(Distance d);
};

int Add::addFive(Distance d) {
        //accessing private members from the friend function
        d.meter += 5;
        return d.meter;
}

int main() {
        Distance d(5);
        cout<<"Distance before: "<<d.getDistance()<<endl;
        Add a;
        cout<<"Distance now: "<<a.addFive(d);
        return 0;
}
```

Access members of two different classes using friend functions

```
#include <iostream>
using namespace std;
// forward declaration
class ClassB;
class ClassA {
        private:
                int numA;
        public:
                ClassA(int numA)
                {
                        this->numA=numA;
                }
                // friend function declaration
                friend int add(ClassA, ClassB);
};
class ClassB {
        private:
                int numB;
        public:
                ClassB(int numB)
                {
                        this->numB=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(4);
        ClassB object(6);
        cout << "Sum: " << add(objectA, objectB);
        return 0;
}
```

Output:

Sum: 10

Friend class

We can also use a friend Class in C++ using the friend keyword.

When a class is declared a friend class, all the member functions of the friend class become friend functions. If ClassB is a friend class of ClassA. So, ClassB has access to the members of classA, but not the other way around. Now in the following code, if we want the class Operations to access all the member functions of the Distance class, instead explicitly declaring all the functions of class Distance as friends, we just make the whole class as a friend.

Syntax:

```
friend class class_name;
```

It includes two classes, "Distance" and "Operations". The "Distance" class has a private member "meter" representing distance in meters and a public function "getDistance()" to return the distance. The "Operations" class has two friend functions, "addFive()" and "subFive()", which can access the private member of the "Distance" class. The main function creates an object of the "Distance" class and initializes it with a value of 5. It then creates an object of the "Operations" class and calls the "addFive()" and "subFive()" functions to increment and decrement the distance by 5, respectively.

Forward declaration is a technique in C++ that allows declaring the name of a class, function, or object before defining it. It informs the compiler that the name is a valid entity, and the definition will be provided later in the code.

```
#include <iostream>
using namespace std;
//forward declaration
class Distance;
class Operations {
        public:
                int addFive(Distance &d);
                int subFive(Distance &d);
};
class Distance {
        private:
                int meter;
                // friend function
```

```
	public:
		int getDistance()
		{
			return meter;
		}
		Distance(int meter) {
			this->meter=meter;
		}
		friend class Operations;
};

int Operations::addFive(Distance &d) {
	//accessing private members from the friend function
	d.meter += 5;
	return d.meter;
}

int Operations::subFive(Distance &d) {
	//accessing private members from the friend function
	d.meter -= 5;
	return d.meter;
}
int main() {
	Distance d(5);
	cout<<"Distance before: "<<d.getDistance()<<endl;
	Operations a;
	cout<<"Distance now by adding 5: "<<a.addFive(d)<<endl;
	cout<<"Distance now by subtracting 5: "<<a.subFive(d);
}
```

Output:

```
Distance before: 5

Distance now by adding 5: 10

Distance now by subtracting 5: 5
```