OBJECT ORIENTED PROGRAMMING LAB

Lab Manual # 12

Inheritance in C++

Instructor: Muhammad Abdullah Orakzai

Semester Fall, 2021

Course Code: CL1004

Fast National University of Computer and Emerging Sciences Peshawar

Department of Computer Science

OBJECT ORIENTED PROGRAMMING LANGUAGE

Table of Contents

Derived Class Constructors

In inheritance, a constructor of the derived class as well as the constructor functions of the base class are automatically executed when an object of the derived class is created.

Constructor in Single Inheritance without Arguments

The following example explains the concept of execution of constructor functions in single inheritance.

Example:

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

class Parent
{
    public:
    Parent(void)
    {
        cout<<"Constructor of the Parent Class Called"<<endl;
    }
}; // end of Parent class

class Child : public Parent
{
    public:
    Child(void)
    {
        cout<<"Constructor of the Child Class Called"<<endl;
    }
};  // end of child class

int main() {

    Child obj;   // object of the child class


    return 0;
}

/*Sample Output

Constructor of the Parent Class Called
Constructor of the Child Class Called   */
```

In the above program, when an object of the derived class Child is created, the constructor of both the derived class as well the base class are executed.

Constructor in Single Inheritance with Arguments

In case of constructor functions with arguments, the syntax to define constructor of the derived class is different. To execute a constructor of the base class that has arguments through the derived class constructor, the derived class constructor is defined as:

- Write the name of the constructor function of the derived class with parameters.
- Place a colon (:) immediately after this and then write the name of the constructor function of the base class with parameters.

The following example explains the concept of the constructor functions with arguments in single inheritance.

```
class Student
{
    private:
    char name[15];
    char address[25];

    public:
    Student(char nm[], char add[]);
    void show();

}; // end of Student class

class Marks : public Student
{
    private:
    int sub1, sub2, sub3, total;

    public:
    Marks(char nm[], char add[], int a, int b, int c):Student(nm, add);
    void show_detail();

};  // end of Marks class
```

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

class Student
{
    private:
    char name[15], address[25];

    public:
    Student(char nm[], char add[])
    {
        strcpy(name,nm);
        strcpy(address, add);

    }
    void show()
    {
        cout<<"Name is: "<<name<<endl;
        cout<<"Address is: "<<address<<endl;
    }
}; // end of Student class

class Marks : public Student
{
    private:
    int sub1, sub2, sub3, total;

    public:
    Marks(char nm[], char add[], int a, int b, int c):Student(nm, add)
    {
        sub1=a;
        sub2=b;
        sub3=c;
        total=a+b+c;
    }
    void show_detail();
};  // end of Marks class

int main() {

    Marks marks("Aiman", "Hangu", 66,77,88);
    marks.show_detail();
    return 0;
}  // end of main() function
```

```
//defining
void Marks:: show_detail()
{
    show();
    cout<<"Marks of 1st subject: "<<sub1<<endl;
    cout<<"Marks of 1st subject: "<<sub2<<endl;
    cout<<"Marks of 1st subject: "<<sub2<<endl;
    cout<<"Total Marks         : "<<total<<endl;
}

/*Sample Output
Name is: Aiman
Address is: Hangu
Marks of 1st subject: 66
Marks of 1st subject: 77
Marks of 1st subject: 88
Total Marks         : 231
*/
```

The compiler automatically calls a base class constructor before executing the derived class constructor. The compiler's default action is to call the default constructor in the base class. You can specify which of several base class constructors should be called during the creation of a derived class object.

```
#include<iostream>
using namespace std;
class Rectangle
{
private:
    float length;
    float width;
public:
    Rectangle ()
    {
        length = 0;
        width = 0;
    }

    Rectangle (float len, float wid)
    {
        length = len;
        width = wid;
    }

    float area()
    {
        return length * width;
    }
};

class Box: public Rectangle
{
private:
    float height;
public:
    Box ()
    {
        height = 0;
    }

    Box (float len, float wid, float ht) : Rectangle(len, wid)
    {
        height = ht;
    }

    float volume()
    {
        return area() * height;
    }
};

int main ()
{
    Box bx;
    Box cx(4,8,5);
    cout << bx.volume() << endl;
    cout << cx.volume() << endl;
    return 0;
}
/*
output
0
160
*/
```

Constructors in Multiple Inheritance Without Arguments

When a class is derived from more than one base classes, the constructor of the derived class as well as of all its base classes are executed when an object of the derived class is created.

If the constructor functions have no parameters then first the constructor functions of the base classes are executed and then the constructor functions of the derived class are executed.

A program is given below to explain the execution of the constructor without parameters.

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

class A
{
    public:
    A(void)
    {
        cout<<"Constructor of class A"<<endl;
    }
}; // end of A class

class B
{
    public:
    B(void)
    {
        cout<<"Constructor of class B"<<endl;
    }
}; // end of B class

class C : public A, public B
{
    public:
    C(void)
    {
        cout<<"Constructor of class C"<<endl;
    }
}; // end of C class

int main() {

    C objC;

    return 0;
}
/*
Constructor of class A
Constructor of class B
Constructor of class C
*/
```

In the above program, three classes are defined. The **C** class is publicly derived from two classes **A** and **B**. All the classes have the constructor functions. When an object of the derived class **C** is created, the constructor functions are executed in the following sequence:

- Constructor of the class **A** is executed first.
- Constructor of the class **B** is executed second.
- Constructor of the derived class **C** is executed in the end.

Constructors in Multiple Inheritance with Arguments

To execute constructors of base classes that have arguments through the derived constructor functions, the derived class constructor is defined as:

- Write the constructor function of the derived class with parameters.
- Place a colon (**:**) immediately after this and then write the constructor functions names of the base classes with parameters separated by commas.

An example is given below to explain the constructor functions arguments in multiple inheritance.

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

class Student
{
    private:
    char name[15], address[25];

    public:
    Student(char nm[], char add[])
    {
        strcpy(name,nm);
        strcpy(address, add);
    }
    void show()
    {
        cout<<"Name is: "<<name<<endl;
        cout<<"Address is: "<<address<<endl;
    }
}; // end of Student class

class Marks
{
    private:
    int sub1, sub2, sub3, total;

    public:
    Marks(int a, int b, int c)
    {
        sub1=a;
        sub2=b;
        sub3=c;
        total=a+b+c;
    }

     void show_marks()
     {
     cout<<"Marks of 1st subject: "<<sub1<<endl;
     cout<<"Marks of 1st subject: "<<sub2<<endl;
     cout<<"Marks of 1st subject: "<<sub3<<endl;
     cout<<"Total Marks         : "<<total<<endl;
     }
};  // end of Marks class

class Show : public Student, public Marks
{
     public:
     Show(char nm[], char add[], int s1, int s2, int s3) : Student(nm,
add), Marks(s1, s2, s3)
           {
            show();          // calling show() method of student class
            show_marks();   // calling show_detail() of marks class
           }


};  // end of Show class

int main() {

    Show marks("Amir", "Kohat", 66,99,88);
    return 0;
}  // end of main() function

/*
output
Name is: Amir
Address is: Kohat
Marks of 1st subject: 66
Marks of 1st subject: 99
Marks of 1st subject: 88
Total Marks         : 253 */
```

Friend Functions

Private members are accessed only within the class they are declared. Friend function is used to access the private and protected members of different classes. **It works as bridge between classes.**

- ❖ Friend function must be declared with **friend** keyword.
- ❖ Friend function must be declared in all the classes from which we need to access private or protected members.
- ❖ Friend function will be defined outside the class without specifying the class name.
- ❖ Friend function will be invoked like normal function, without any object.

As we have seen in the previous sections, private and protected data or function members are normally only accessible by the code which is part of same class. However, situations may arise in which it is desirable to allow the explicit access to private members of class to other functions.

If we want to declare an external function as friend of a class, thus allowing this function to have access to the private and protected members of this class, we do it by declaring a prototype of this external function within the class, and preceding it with the keyword friend.

Declaration of friend function in C++

```
class class_name
{
    friend data_type function_name(argument/s); // syntax of friend function.

};
```

In the above declaration, the friend function is preceded by the keyword friend. The function can be defined anywhere in the program like a normal C++ function. The function definition does not use either the keyword **friend or scope resolution operator**.

Example - 01 of friend function

```
#include <iostream>
using namespace std;
class Rectangle
{
    private:
        int length;
        int width;
    public:
        void setData(int len, int wid)
        {
            length = len;
            width = wid;
        }
      int getArea()
        {
            return length * width ;
        }

      friend double getCost(Rectangle);//friend of class Rectangle
};  // end of class body
//friend function getCost can access private member of class
double getCost (Rectangle rect)
{
    double cost;
    cost = rect.length * rect.width * 5;
    return cost;
}
int main ()
{
    Rectangle floor;
    floor.setData(20,3);
    cout<<"Area is:"<<floor.getArea()<<endl;
    cout << "Expense " << getCost(floor) << endl;
    return 0;
}
/*
Output
Area is:60
Expense 300
*/
```

The getCost function is a friend of Rectangle. From within that function we have been able to access the members length and width, which are private members.

Example - 02 of friend function

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

class RectangleTwo;   // class declaration

class RectangleOne    // class definition
{
    int L,B; // private data members
    public:
```

```
    RectangleOne(int l,int b)
    {
        L = l;
        B = b;
    }
    //friend function
    friend void Sum(RectangleOne, RectangleTwo);
};  // end of RectangleOne class

class RectangleTwo
{
    int L,B; // private data members
    public:
    RectangleTwo(int l,int b)
    {
        L = l;
        B = b;
    }

    //friend function
    friend void Sum(RectangleOne, RectangleTwo);
};  // end of RectangleTwo

//friend function definiton
void Sum(RectangleOne R1,RectangleTwo R2)
{
    cout<<"\n\t\tLength\tBreadth";
    cout<<"\n Rectangle 1  :   "<<R1.L<<"\t  "<<R1.B;
    cout<<"\n Rectangle 2  :   "<<R2.L<<"\t  "<<R2.B;
    cout<<"\n -----------------------------";
    cout<<"\n\tSum   :   "<<R1.L+R2.L<<"\t  "<<R1.B+R2.B;
    cout<<"\n -----------------------------";
}
int main()
{
    RectangleOne Rec1(5,3);
    RectangleTwo Rec2(2,6);

    Sum(Rec1,Rec2);
    return 0;

}  // end of main() function
/*
Output
Length  Breadth
 Rectangle 1  :   5       3
 Rectangle 2  :   2       6
 -------------------------------
           Sum   :   7       9
 -------------------------------
*/
```

In the above example, we have created two classes RectangleOne and RectangleTwo with a common friend function Sum().

Sum() method is receiving two objects, one of RectangleOne class and second of RectangleTwo class. This method is displaying the values of private data members of both the classes and also calculating the sum of both the rectangle classes.

C++ Friend Classes

- ❖ Like friend function, a class can also be a friend of another class. A friend class can access all the private and protected members of other class.
- ❖ In order to access the private and protected members of a class into friend class we must pass on object of a class to the member functions of friend class.
- ❖ One class member function can access the private and protected members of other class. We do it by declaring a class as friend of other class.

Example - 01 of C++ friend class

```
#include <iostream>
using namespace std;
class CostCalculator;
class Rectangle
{
    private:
        int length;
        int width;
    public:
        void setData(int len, int wid)
        {
            length = len;
            width = wid;
        }
        int getArea()
        {
            return length * width ;
        }
        friend class CostCalculator; //friend of class Rectangle
};
```

```
//friend class costCalculator can access private member of class Rectangle

class CostCalculator
{
    public :
        double getCost (Rectangle rect)
        {
            double cost;
            cost = rect.length * rect.width * 5;
            return cost;
        }
};
int main ()
{
    Rectangle floor;
    floor.setData(20,3);
    CostCalculator calc;
    cout<<"Area is:"<<floor.getArea()<<endl;
    cout << "Expense " << calc.getCost(floor) << endl;
    return 0;
}
//Note:An empty declaration of class costCalculator at top is necessary.

/*
Output
Area is:60
Expense 300
*/
```

Example - 02 of C++ friend class

```
#include<iostream>
using namespace std;
class Rectangle
{
    int L,B;

    public:
    Rectangle()
    {
        L=10;
        B=20;
    }

    friend class Square;        //Statement 1
};  // end of Rectangle class
class Square
{
    int S;
    public:
    Square()
    {
        S=5;
    }
    void Display(Rectangle Rect)
    {
        cout<<"\n\n\tLength : "<<Rect.L;
        cout<<"\n\n\tBreadth : "<<Rect.B;
        cout<<"\n\n\tSide : "<<S;
    }
}; // end of Square class
int main()
{
    Rectangle R;
    Square S;

    S.Display(R);       //Statement 2
    return 0;
}
/*
Output
Length : 10
Breadth : 20
Side : 5
*/
```

- In the above example, we have created two classes Rectangle and Square. Using statement 1 we have made Square class, a friend class of Rectangle class. In order to access the private and protected members of Rectangle class into Square class we must explicitly pass an object of Rectangle class to the member functions of Square class as shown in statement 2.
- This is similar to passing an object as function argument but the difference is, an object (R) we are passing as argument is of different class (Rectangle) and the calling object is of different class (Square).

References

https://beginnersbook.com/2017/08/cpp-data-types/

http://www.cplusplus.com/doc/tutorial/basic_io/

https://www.w3schools.com/cpp/default.asp

https://www.javatpoint.com/cpp-tutorial

https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/?ref=lbp

https://www.programiz.com/

https://ecomputernotes.com/cpp/