National University of Computer & Emerging Sciences, Karachi
Computer Science Department
Spring 2022, Lab Manual - 05

Course Code: CL-1004	Course : Object Oriented Programming Lab

Lab # 05

Outline

- Access Modifiers
- Static Keyword
- Constant Keyword
- Member initializer list

Introduction to Access Modifiers in C++:

Access modifiers is the techniques that is applied to members of class to restrict their access beyond the class.

In C++, access modifiers can be achieved by using three keywords:

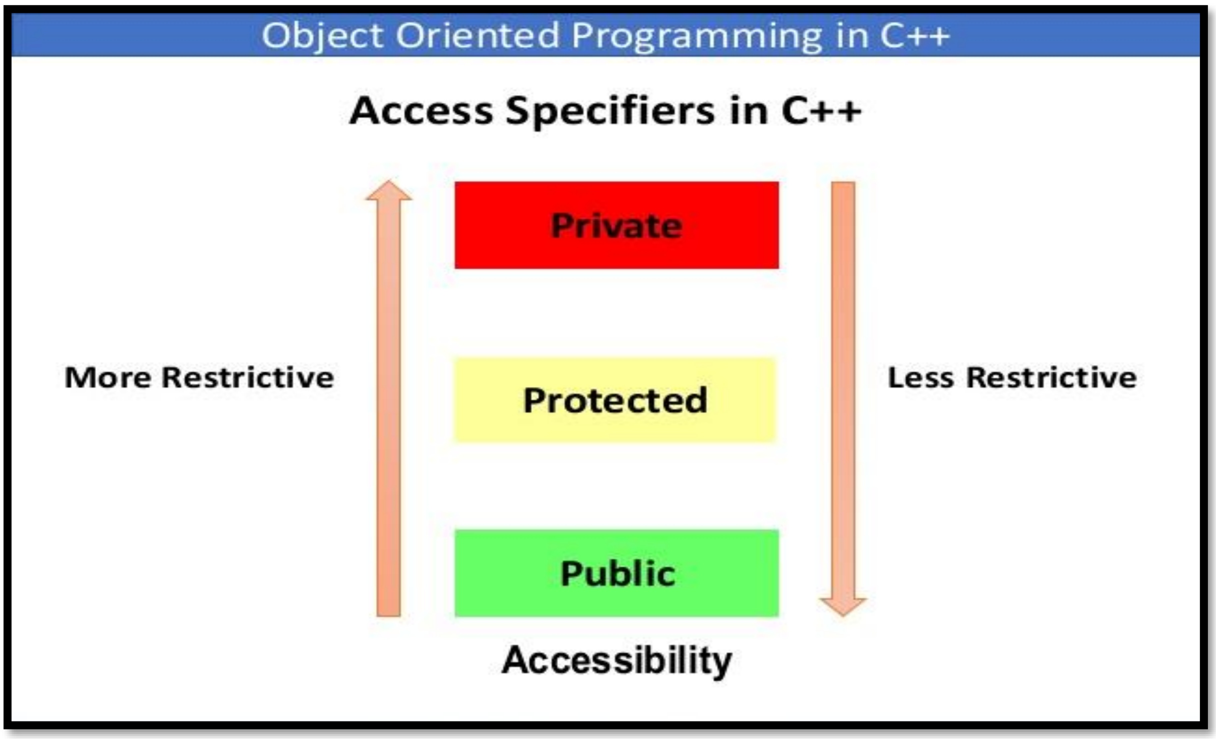
Public

Private

Protected

Types of Access Modifiers in C++

There are 3 types of Access Modifiers in C++

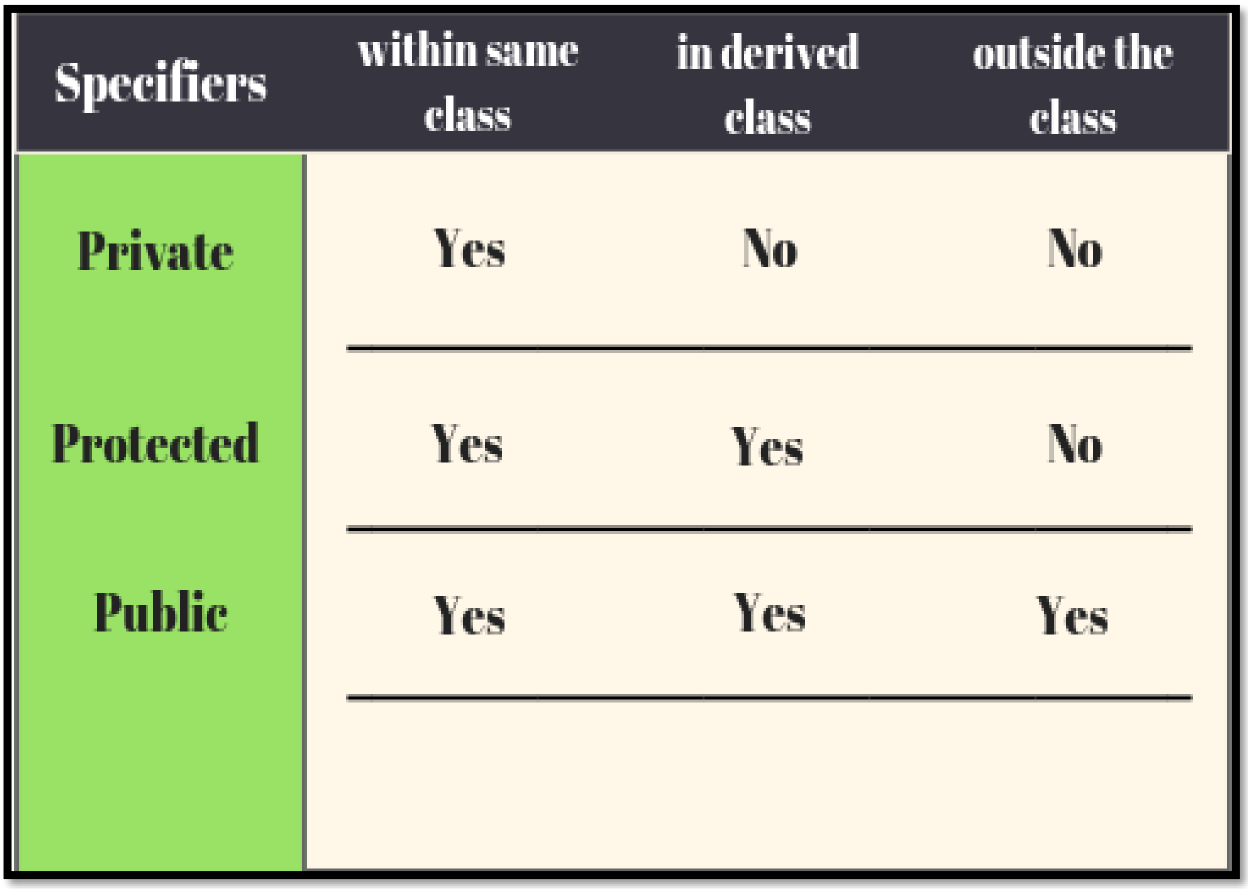


1. Public
2. Private
3. Protected

Public members can be accessed anywhere i.e. inside or outside the class but within the program only,

Private members can be accessed inside the class only,

Protected members are like the private they can be accessed in the child class/derived class.



Specifiers	within same class	in derived class	outside the class
Private	Yes	No	No
Protected	Yes	Yes	No
Public	Yes	Yes	Yes

Let’s look at these modifiers with examples:

Public:

As there are no restrictions in public modifier, we can use the (.)dot operator directly accesses member functions and data.

```
// C++ program to demonstrate public access modifier

#include<iostream>

using namespace std;

// class definition

class Circle

{    public:

        double radius;

        double  compute_area()

        {          return 3.14*radius*radius;      }

    };

int main ()

{   Circle obj;
```

```
    // accessing public data member outside class

     obj.radius = 5.5;

     cout << "Radius is: " << obj.radius << "\n";

     cout << "Area is: " << obj.compute_area();

     return 0;  }
```

Output:

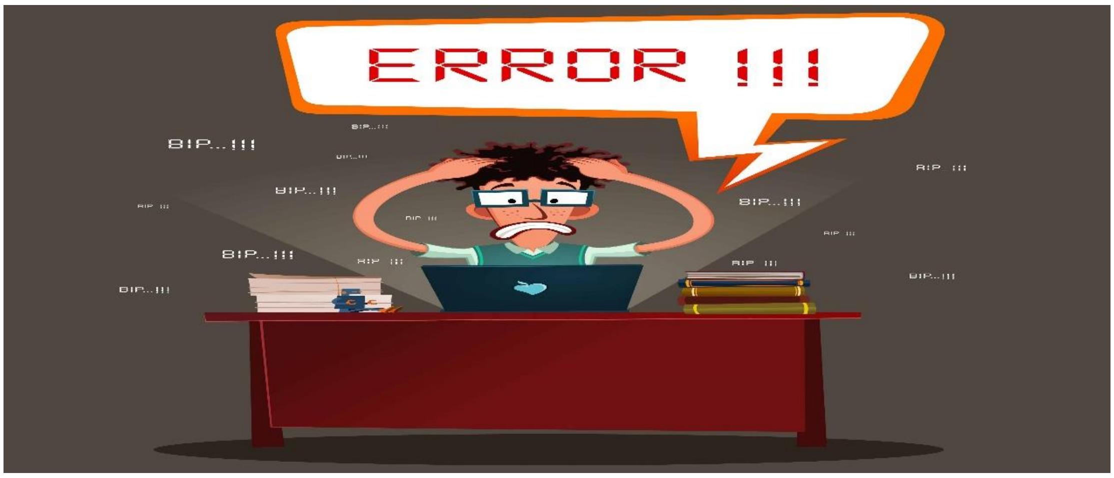
```
Radius is: 5.5

Area is: 94.985
```

Private:

Only the member functions or the friend functions are allowed to access the private data members of a class.

The example below has an error let's find out:



```
// C++ program to demonstrate private
// access modifier
#include<iostream>
using namespace std;

class Circle
{
    // private data member
    private:
            double radius;
            // public member function
    public:
            double compute_area()
            { // member function can access private
                    // data member radius
                    return 3.14*radius*radius;
            }
};
// main function
int main()
{
    // creating object of the class
    Circle obj;
    // trying to access private data member
    // directly outside the class
    obj.radius = 1.5;
    cout << "Area is:" << obj.compute_area();
    return 0;
}
```

```
// C++ program to demonstrate private
// access modifier
#include<iostream>
using namespace std;
class Circle
{
    // private data member
    private:
            double radius;
    // public member function
    public:
            void compute_area(double r)
            { // member function can access private
                    // data member radius
                    radius = r;
                    double area = 3.14*radius*radius;
                    cout << "Radius is: " << radius << endl;
                    cout << "Area is: " << area;
            }
    };
// main function
int main()
{   // creating object of the class
    Circle obj;
    // trying to access private data member
    // directly outside the class
    obj.compute_area(1.5);
    return 0;  }
```

Output:

```
Radius is: 1.5
```

Area is: 7.065

Protected:

```
#include <bits/stdc++.h>
using namespace std;
// base class
class Parent
{   // protected data members
    protected:
    int id_protected;
};
// sub class or derived class from public base class
class Child : public Parent
{   public:
    void setId(int id)
    {       // Child class is able to access the inherited protected data members of
    base class
            id_protected = id;
    }
    void displayId()
    {
            cout << "id_protected is: " << id_protected << endl;
    }
};
// main function
int main() {
    Child obj1;
     // member function of the derived class can access the protected data
    members of the base class
    obj1.setId(81);
    obj1.displayId();
    return 0;
}
```

Output:

```
id_protected is: 81
```

const **Keyword in C++**

Constant is something that doesn't change. In C language and C++ we use the keyword const to make program elements constant. **const** keyword can be used in many contexts in a C++ program. It can be used with:

1. Variables
2. Function arguments and return types
3. Class Data members
4. Class Member functions
5. Objects

Constant Variables in C++

```
int main
{
   const int i = 10;
   const int j = i + 10;     // works fine
   i++;    // this leads to Compile time error
}
```

If you make any variable as constant, using const keyword, you cannot change its value. Also, the constant variables must be initialized while they are declared.

In the above code we have made i as constant, hence if we try to change its value, we will get compile time error. Though we can use it for substitution for other variables.

Static Keyword in C++

Static is a keyword in C++ used to give special characteristics to an element. Static elements are allocated storage only once in a program lifetime in static storage area. And they have a scope till the program lifetime. Static Keyword can be used with following,

1. Static variable in functions
2. Static Class Objects
3. Static member Variable in class
4. Static Methods in class

A simple function call

```
void staticDemo() {

        int val = 0;
        ++val;
        cout << “val = "  << val << endl;

}

int main() {

        staticDemo();    // prints val = 1

        staticDemo();    // prints  val = 1

        staticDemo();    // prints val  = 1

}
```

Static Local Variables

```
void staticDemo() {

        static int val = 0;
        ++val;
        cout << “val = "  << val << endl;

}

int main() {

        staticDemo();    // prints val = 1
        staticDemo();    // prints  val = 2
        staticDemo();    // prints val  = 3

}
```

MEMBER INITIALIZATION LIST:

Initializer List is used in initializing the data members of a class. The list of members to be initialized is indicated with constructor as a comma-separated list followed by a colon. Following is an example that uses the initializer list to initialize x and y of Point class.

```
class Point {
private:
    int x;
    int y;
public:
    Point(int i = 0, int j = 0):x(i), y(j) {}
    int getX() const {return x;}
    int getY() const {return y;}
};

int main() {
  Point t1(10, 15);
  cout<<"x = "<<t1.getX()<<", ";
  cout<<"y = "<<t1.getY();
  return 0;
}
```

Uses:

- For initialization of non-static const data members.
- For initialization of reference members.
- For initialization of member objects which do not have default constructor.
- For initialization of base class members.
- When constructor's parameter name is same as data member.
- For Performance reasons.

Exercise Lab 05

Question # 01:

Create a class **Person**. It should contain the following **private** data members:

- CNIC
- Name
- Age
- DoB

Generate appropriate **accessors** & **mutators** for the class.
CNIC should be a **constant** member and should be initialized through a **member initializer list**.
Create **two objects** of the class and pass the values through a **parametrized constructor.**
Display all the values that you have set by using accessor functions.

Question # 02:

Create a class called **WarehouseItem**.
It should have the following the attributes:

- ItemID
- ItemName
- ItemAmount
- TotalUniqueItemsInWarehouse

TotalUniqueItemsInWarehouse should be a **static member**.
It should contain the number of total unique items that a warehouse has. (aka, should have the count of total objects created)
Create a **destructor** for the class that decrements the count of the unique items when an object is deleted.
Test it out by **allocating** and **deallocating** a WarehouseItem **dynamically**.
You can use the following lines of code to test it:

```
WarehouseItem *t = new WarehouseItem ();
delete t;
```

Question # 03:

Create a class named **Account** with the following **private** attributes:

- AccountId
- AccountName
- Branch
- Amount
- Tax

Create two objects called savingsAccount and currentAccount
Both Savings and Current account have different variables for Tax. For Savings Tax should be equal to **5%** and for Current, the Tax should be **3%**.
You should set the **Tax** values for each object through **initializer lists**.
You can set the values in the **Account** through **accessors** and **mutators**

Demonstrate the tax calculation for objects of these classes through an appropriate function.

Question # 04:
Design and implement a class dayType that implements the day of the week in a program. The class dayType should store the day, such as Sun for Sunday. The program should be able to perform the following operations on an object of type dayType:

- Set the day.
- Print the day.
- Return the day.
- Return the next day.
- Return the previous day.
- Calculate and return the day by adding certain days to the current day. For example, if the current day is Monday and we add 4 days, the day to be returned is Friday. Similarly, if today is Tuesday and we add 13 days, the day to be returned is Monday.
- Add the appropriate **constructors**. Use **initializer lists** to initialize the values.

Question # 05:

Ahmed is new to Cricket, and he is maintaining a list of his scores on each game that he plays. He wants to compare his scores to other colleagues who are also new to Cricket. Let's assume he only stores the scores for 10 recent games.
He has 5 other colleagues for whom he also stores the scores for 10 recent games. (You can store these via parametrized constructors. Store the values in an array and pass the array to the constructor as well)
Create a class that will store the scores for each object created.
Create a **static variable** called **highestScore** in this class, which is updated every time a new object is created. This variable should always have the highest score among all the players.
Create a function that should also check if the current object has the highest score or not.

Create variables of the class to demonstrate the working of your class.