OBJECT PROGRAMMING

- LECTURE 2-

(1ST YEAR OF STUDY)

Skopje, 2023/24

Contents

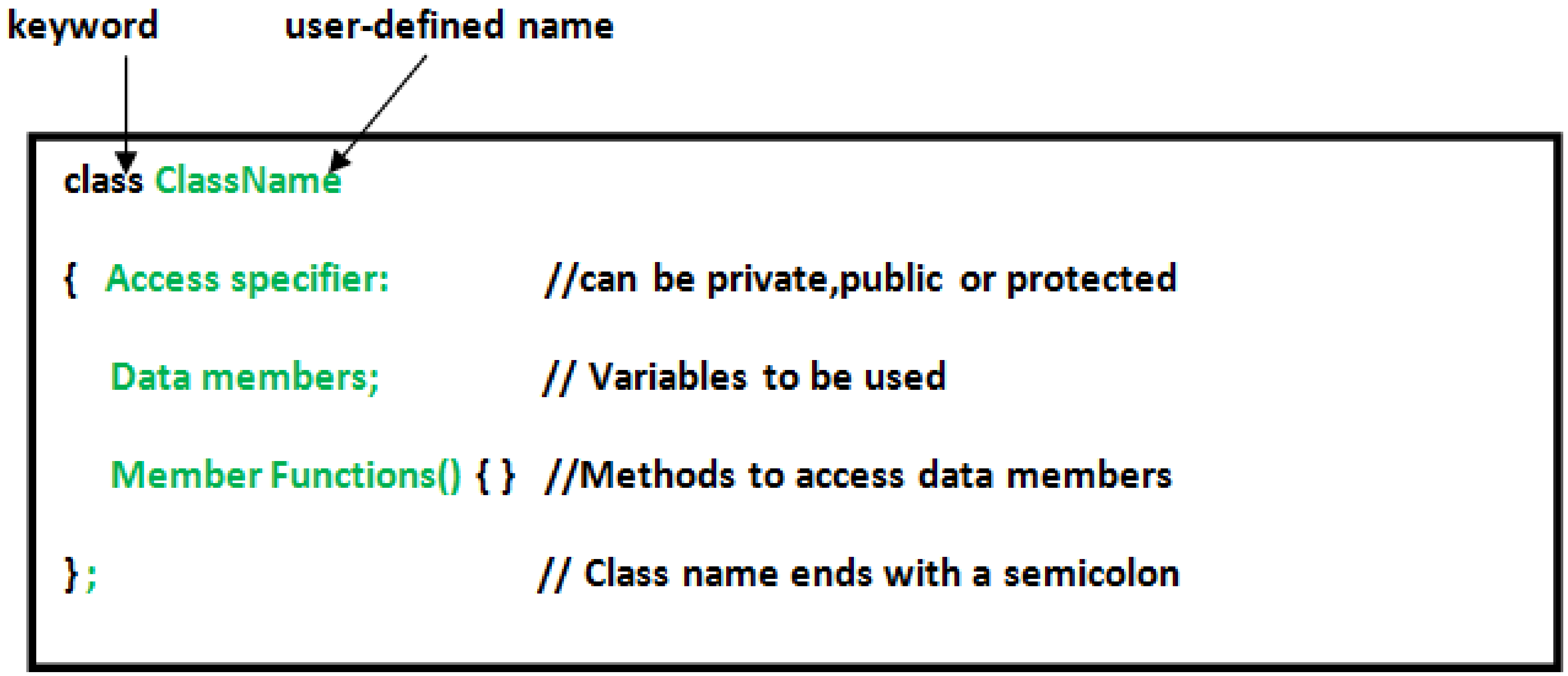
2.1. Introduction

- A **Class** is a user-defined data type that has: ***data members*** (*variables*) and ***member functions*** (*methods*).
- *Data members* are the *data variables*, and *member functions* are the *functions* (*methods*) used to manipulate these variables. Together, these data members and member functions define the *properties* and *behavior* of the objects in a Class.
- In the class *Car*, the data members will be *speed limit*, *mileage,* etc, and function members can be *applying brakes*, *increasing speed*, etc.

2.1. Introduction

- An **Object** is an *instance* of a Class. When a class is defined, no memory is allocated, but when it is instantiated (i.e., an object is created), memory is allocated.
- **Defining Class**
    - A class is defined in C++ using the keyword **class** followed by the name of the class.
    - The body of the class is defined inside the curly brackets and terminated by a semicolon at the end.

2.1. Introduction



```
class ClassName

{   Access specifier:              //can be private,public or protected

    Data members;                  // Variables to be used

    Member Functions() { }         //Methods to access data members

};                                 // Class name ends with a semicolon
```

2.1. Introduction

- **Declaring Objects**
  - When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.
- **Syntax**

```
ClassName ObjectName;
```

2.1. Introduction

- **Accessing data members and member functions:**
  - The data members and member functions of the class can be accessed using the dot ('.') operator with the object.
  - For example, if the name of the object is ***obj*** and you want to access the member function with the name ***PrintName()***, then you will have to write ***obj.PrintName()***.

2.1. Introduction

- **Accessing Data Members**
  - The public data members are also accessed in the same way given, however the private data members are not allowed to be accessed directly by the object.
  - Accessing a data member depends solely on the access control of that data member.
  - This access control is given by ***Access modifiers in C++***.
  - There are three access modifiers: **public, private,** and **protected.**

2.1. Introduction

```
// C++ program to demonstrate accessing of data members
#include <bits/stdc++.h>
using namespace std;
class Geeks {
    // Access specifier
public:
    // Data  Members
    string geekname;
    // Member Functions()
    void printname() { cout << "Geekname is:" << geekname; }
};
int main()
{
    // Declare an object of class geeks
    Geeks obj1;
    // accessing data member
    obj1.geekname = "Abhi";
    // accessing member function
    obj1.printname();
    return 0;
}
```

Output

```
Geekname is:Abhi
```

2.1. Introduction

- **Member Functions in Classes**
- There are 2 ways to define a member function:
  - Inside class definition
  - Outside class definition
- To define a member function outside the class definition we have to use the **scope resolution operator ::** along with the *class name* and *function name.*

2.1. Introduction

```cpp
// C++ program to demonstrate function
// declaration outside class

#include <bits/stdc++.h>
using namespace std;
class Geeks
{
    public:
    string geekname;
    int id;

    // printname is not defined inside class definition
    void printname();

    // printid is defined inside class definition
    void printid()
    {
        cout <<"Geek id is: "<<id;
    }
};

// Definition of printname using scope resolution operator ::
void Geeks::printname()
{
    cout <<"Geekname is: "<<geekname;
}
```

```cpp
int main() {

    Geeks obj1;
    obj1.geekname = "xyz";
    obj1.id=15;

    // call printname()
    obj1.printname();
    cout << endl;

    // call printid()
    obj1.printid();
    return 0;
}
```

Output

```
Geekname is: xyz
Geek id is: 15
```

2.1. Introduction

- Note that all the member functions defined inside the class definition are by default **inline**, but you can also make any non-class function inline by using the keyword inline with them.
- Inline functions are actual functions, which are copied everywhere during compilation, like pre-processor macro, so the overhead of function calls is reduced.
- **Note:** Declaring a ***friend function*** is a way to give private access to a non-member function.

2.2. Constructors

- ***Constructors*** are special class members which are called by the compiler every time an object of that class is instantiated.
- Constructors have the same name as the class and may be defined inside or outside the class definition. There are 3 types of constructors:
  - Default Constructors
  - Parameterized Constructors
  - Copy Constructors

2.2. Constructors

```cpp
// C++ program to demonstrate constructors
#include <bits/stdc++.h>
using namespace std;
class Geeks
{
    public:
    int id;

    //Default Constructor
    Geeks()
    {
        cout << "Default Constructor called" << endl;
        id=-1;
    }

    //Parameterized Constructor
    Geeks(int x)
    {
        cout <<"Parameterized Constructor called "<< endl;
        id=x;
    }
};
```

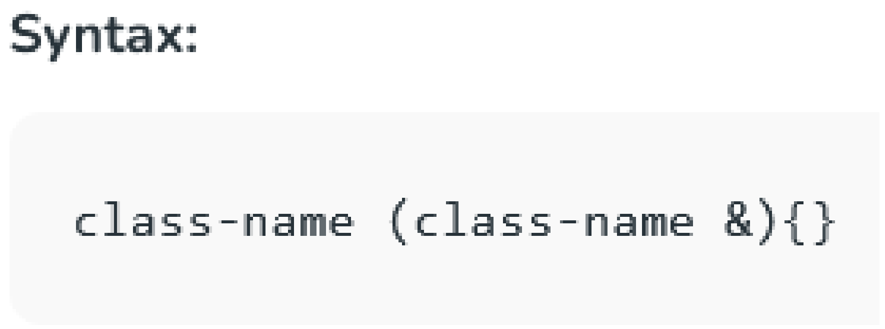
```cpp
int main() {

    // obj1 will call Default Constructor
    Geeks obj1;
    cout <<"Geek id is: "<<obj1.id << endl;

    // obj2 will call Parameterized Constructor
    Geeks obj2(21);
    cout <<"Geek id is: " <<obj2.id << endl;
    return 0;
}
```

Output

```
Default Constructor called
Geek id is: -1
Parameterized Constructor called
Geek id is: 21
```

2.2. Constructors

- A **Copy Constructor** creates a new object, which is an exact copy of the existing object.
- The compiler provides a default Copy Constructor to all the classes.

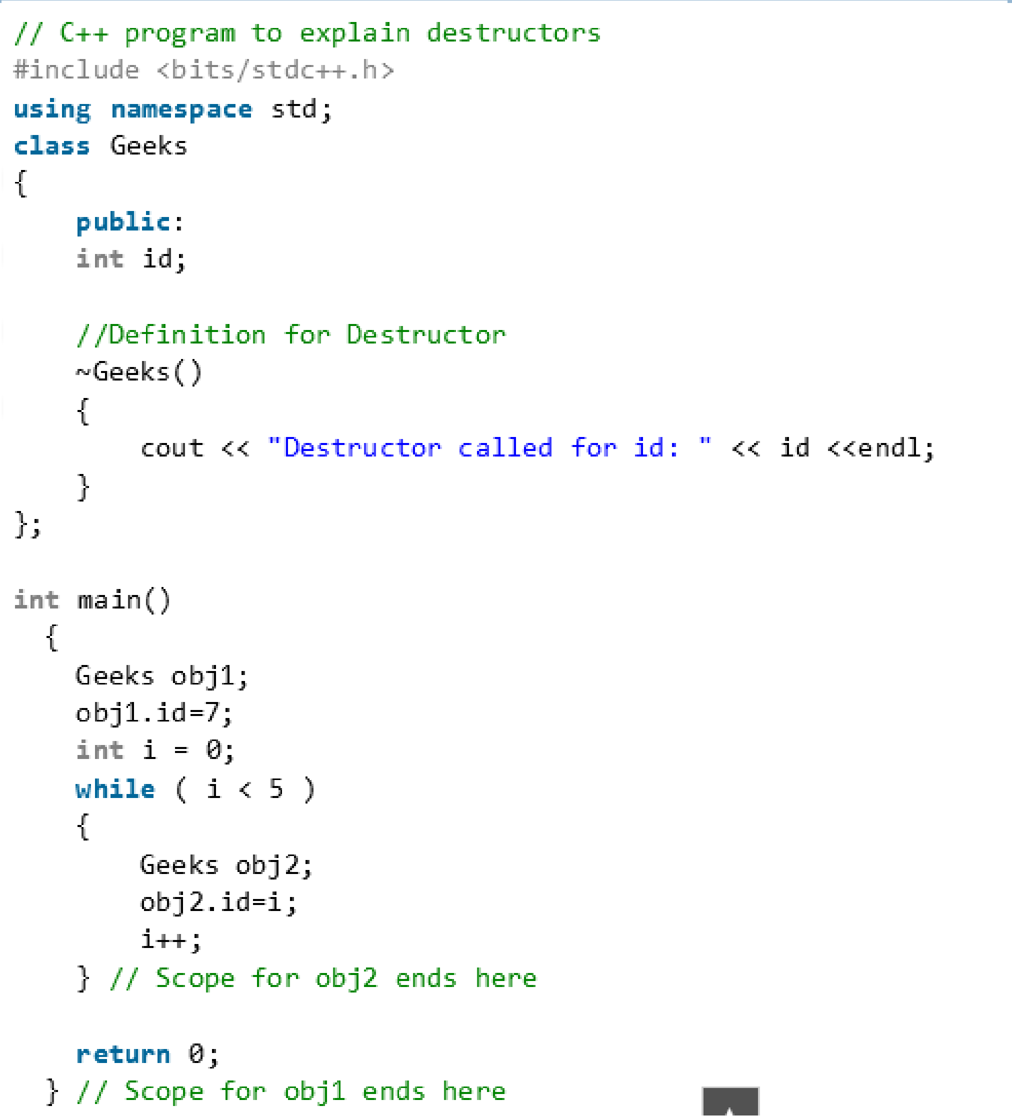


Syntax:

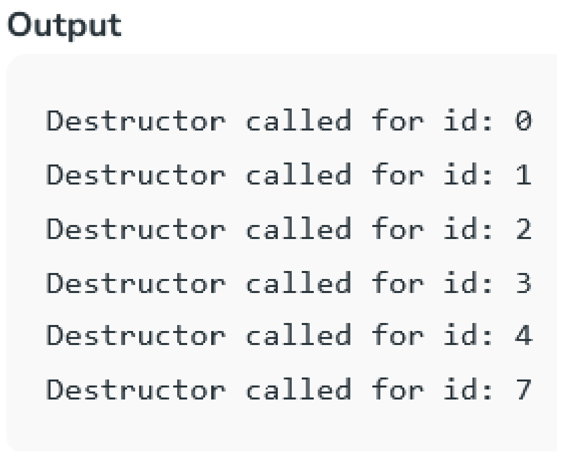
```
class-name (class-name &){}
```

- ***Destructor*** is another special member function that is called by the compiler when the scope of the object ends.

2.2. Constructors



```
// C++ program to explain destructors
#include <bits/stdc++.h>
using namespace std;
class Geeks
{
    public:
    int id;

    //Definition for Destructor
    ~Geeks()
    {
        cout << "Destructor called for id: " << id <<endl;
    }
};

int main()
  {
    Geeks obj1;
    obj1.id=7;
    int i = 0;
    while ( i < 5 )
    {
        Geeks obj2;
        obj2.id=i;
        i++;
    } // Scope for obj2 ends here

    return 0;
  } // Scope for obj1 ends here
```



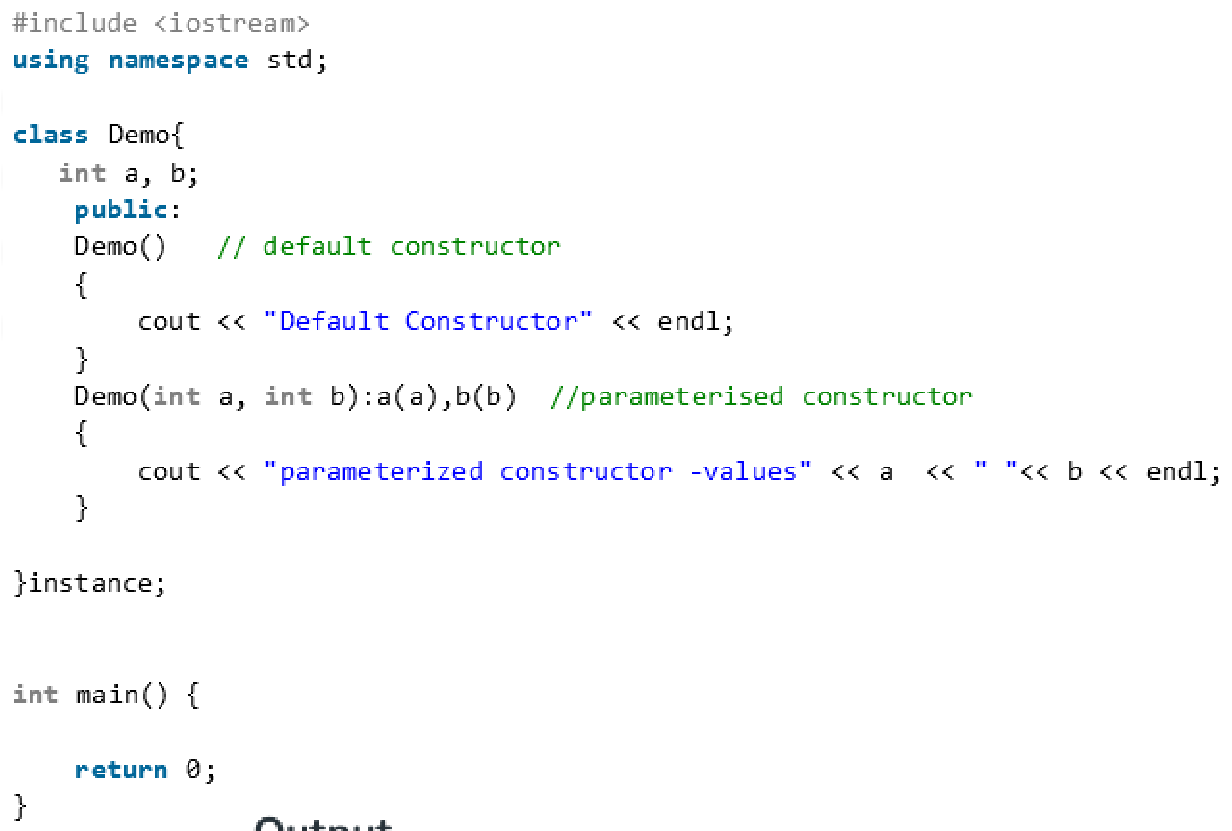
Output

```
Destructor called for id: 0
Destructor called for id: 1
Destructor called for id: 2
Destructor called for id: 3
Destructor called for id: 4
Destructor called for id: 7
```

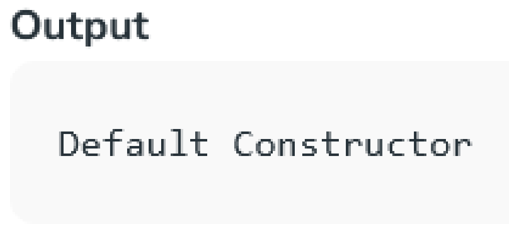
2.2. Constructors

- **Interesting Fact – Why do we give semicolons at the end of class?**
- Many people might say that it's a basic syntax and we should give a semicolon at the end of the class as its rule defines in cpp. But the main reason why semi-colons are there at the end of the class is: compiler checks if the user is trying to create an instance of the class at the end of it.
- Just like structure and union, we can also create the instance of a class at the end, just before the semicolon. As a result, once execution reaches at that line, it creates a class and allocates memory to your instance.

2.2. Constructors



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

class Demo{
   int a, b;
    public:
    Demo()   // default constructor
    {
        cout << "Default Constructor" << endl;
    }
    Demo(int a, int b):a(a),b(b)  //parameterised constructor
    {
        cout << "parameterized constructor -values" << a  << " "<< b << endl;
    }

}instance;


int main() {

    return 0;
}
```



Output

```
Default Constructor
```

- We can see that we have created a ***class instance*** of Demo with the name “instance”.
- As a result, the output is Default Constructor is called.

2.2. Constructors

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

class Demo{
    public:
    int a, b;
    Demo()
    {
        cout << "Default Constructor" << endl;
    }
    Demo(int a, int b):a(a),b(b)
    {
        cout << "parameterized Constructor values-" << a << " "<< b << endl;
    }


}instance(100,200);


int main() {

    return 0;
}
```

Output

```
parameterized Constructor values-100 200
```

- We can also call the parameterized constructor just by passing values here.
- So, by creating an instance just before the semicolon, we can create the Instance of class.

2.3. Access Modifiers

- ***Access Modifiers*** are used to implement an important aspect of Object-Oriented Programming known as **Data Hiding**.
- Consider a real-life example: The Research and Analysis Wing (R&AW), having 10 core members, has come into possession of sensitive confidential information regarding national security. Now we can correlate these core members to data members or member functions of a class, which in turn can be correlated to the R&A Wing.

2.3. Access Modifiers

- These 10 members can directly access the confidential information from their wing (the class), but anyone apart from these 10 members can't access this information directly, i.e., outside functions other than those prevalent in the class itself can't access the information (that is not entitled to them) without having either assigned privileges (such as those possessed by a *friend* class or an *inherited* class, as will be seen ahead) or access to one of these 10 members who are allowed direct access to the confidential information.

2.3. Access Modifiers

- This is similar to how private members of a class can be accessed in the outside world through public member functions of the class that have direct access to private members. This is what *data hiding* is in practice.
- ***Access Modifiers*** or ***Access Specifiers*** in a **class** are used to assign the accessibility to the class members, i.e., they set some restrictions on the class members so that they can't be directly accessed by the outside functions.
- There are 3 types of access modifiers available in C++:
  - **Public, Private, Protected.**

2.3. Access Modifiers

- **Note:** If we do not specify any access modifiers for the members inside the class, then, by default, the access modifier for the members will be **Private.**
- Let us now look at each one of these access modifiers:
- **Public:** All the class members declared under the ***public specifier*** will be available to everyone. The *data members* and *member functions* declared as public can be accessed by other classes and functions, too. The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.

2.3. Access Modifiers

```cpp
// 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;
        }

};
```

```cpp
// main function
int main()
{
    Circle obj;

    // accessing public datamember 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
```

2.3. Access Modifiers

- In the above program, the data member *radius* is declared as ***public***, so it could be accessed outside the class and thus was allowed access from inside main().
- **Private:** The class members declared as *private* can be accessed only by the member functions *inside* the class. They are not allowed to be accessed directly by any object or function *outside* the class. Only the member functions or the ***friend functions*** are allowed to access the private data members of the class.

2.3. Access Modifiers

```
// 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;
}
```

Output:

```
 In function 'int main()':
11:16: error: 'double Circle::radius' is private
         double radius;
                ^
31:9: error: within this context
     obj.radius = 1.5;
         ^
```

2.3. Access Modifiers

- The output of the above program is a *compile time error* because we are not allowed to access the private data members of a class directly from outside the class. Yet an access to obj.radius is attempted, but radius being a private data member, we obtained the above compilation error.
- However, we can access the *private data members* of a class indirectly using the *public member functions* of the class.

2.3. Access Modifiers

```cpp
// 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;
        }

};
```

```cpp
// 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
```

2.3. Access Modifiers

- **Protected:** The protected access modifier is similar to the private access modifier in the sense that it can't be accessed outside of its class unless with the help of a friend class. The difference is that the class members declared as Protected can be accessed by any ***subclass*** (***derived class***) of that class as well.
- **Note:** This access through inheritance can alter the access modifier of the elements of base class in derived class depending on the ***mode of Inheritance***.

2.3. Access Modifiers

```
// C++ program to demonstrate
// protected access modifier
#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
```