Access Modifiers in C++

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. 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 in this article ahead) or access to one of these 10 members who is allowed direct access to the confidential information (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++:

1. **Public**
2. **Private**
3. **Protected**

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 in detail:

1. 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.

```
// C++ program to demonstrate public
#include<iostream>
using namespace std;
// class definition
class Circle
{
   public:
      double radius;

      double  compute_area()
      {
         return 3.14*radius*radius;
      }

};

// 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;
}
```

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().

2. 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.

```
// C++ program to demonstrate private

#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;
}
```

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.

```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;
      }

};

// 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;
}
```

3. 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.

```cpp
// 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;
}
```