Access Modifiers & Abstraction

Data abstraction is one of the most essential and important features of object-oriented programming in C++. Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop the car but he does not know about how on pressing accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes, etc in the car. This is what abstraction is.

Abstraction using Classes: We can implement Abstraction in C++ using classes. Classes help us to group data members and member functions using available access specifiers. A Class can decide which data member will be visible to the outside world and which is not.

Abstraction in Header files: One more type of abstraction in C++ can be header files. For example, consider the pow() method present in math.h header file. Whenever we need to calculate the power of a number, we simply call the function pow() present in the math.h header file and pass the numbers as arguments without knowing the underlying algorithm according to which the function is actually calculating the power of numbers.

Abstraction using Access Modifiers

There are 3 types of access modifiers in C++, which we discuss in detail below:

- **Public**: All the class members declared under public will be available to everyone. The data members and member functions declared public can be accessed by other classes 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.

  ```
  #include <bits/stdc++.h>
  using namespace std;

  ```

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

    return 0;
}
```

Output:

```
Radius is: 5.5
Area is: 94.985
```

In the above program the data member *radius* is public so we are allowed to access it outside the class.

- **Private**: The class members declared as **private** can be accessed only by the 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 (discussed later) are allowed to access the private data members of a class.

```
#include <bits/stdc++.h>
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;
        }
};

int main()
{
    Circle obj;
    //trying to access private data member
    //directly outside the class
    obj.radius = 1.5;
    cout << "Area is: " << obj.compute_area();
    return 0;
}
```

Error generated by the above code:

```
prog.cpp: In function 'int main()':
prog.cpp:9:16: error: 'double Circle::radius' is private
         double radius;
                ^
prog.cpp:27:9: error: within this context
     obj.radius = 1.5;
         ^
```

The output of the above program will be a compile-time error because we are not allowed to access the private data members of a class directly outside the class. If we comment out the *obj.radius = 1.5;* statement, then the program compiles fine. We can access the private data members of a class indirectly using the public member functions of the class. Below program explains how to do this:

```
#include <bits/stdc++.h>
using namespace std;

class Circle
{
    //private data member
    private:
        double radius;

    //public member functions
    public:
        int getRadius() { return radius; }
        void setRadius(double r) { radius = r; }

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

int main()
{
    Circle obj;

    obj.setRadius(5);
    cout << "Radius: " << obj.getRadius() << endl;
    cout << "Area: " << obj.compute_area() << endl;

    return 0;
}
```

Output:

```
Radius: 5
Area: 78.5
```

We can use **getters** and **setter** public functions to indirectly access and manipulate the values of private data-members.

- **Protected**: Protected access modifier is similar to that of private access modifiers, the difference is that the class member declared as Protected are inaccessible outside the class but they can be accessed by any subclass(derived class) of that class.

```
#include <bits/stdc++.h>
using namespace std;

//Base Class
class Parent
{
    //protected data members
    protected:
        int id_protected;
};

//Derived Class
class Child : public Parent
{
    public:
        void setId(int id) {
            id_protected = id;
        }

        void displayId() {
            cout << "id_protected is: " << id_protected <<
        }
};

int main() {

    Child obj;

    //member function of the derived class can
    //access the protected data members of the base class
    obj.setId(81);
    obj.displayId();
    return 0;
}
```

Output:

```
id_protected is: 81
```

Advantages of Data Abstraction:

- Helps the user to avoid writing the low level code
- Avoids code duplication and increases reusability.
- Can change internal implementation of class independently without affecting the user.
- Helps to increase security of an application or program as only important details are provided to the user.