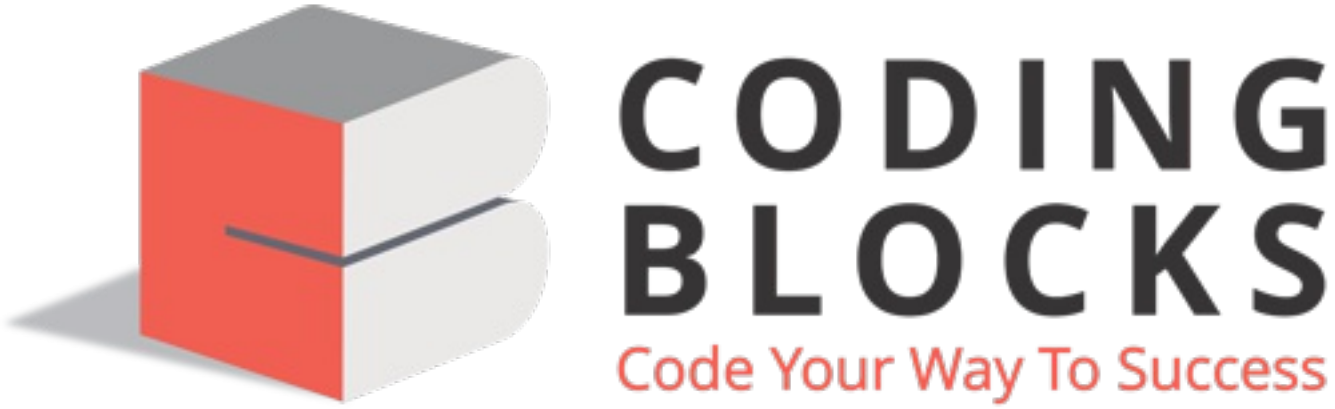

Object Oriented Programming

- uses objects in programming
- mapping real world entity to an object
- The main aim of OOP is to bind together the data and the functions that operates on them so that no other part of code can access this data except that function.

Class and Objects

- **Class** is a blueprint of data and functions or methods. Class does not take any space. A class definition starts with the keyword class followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations.
- **Objects** objects are instances of a class these are defined as user defined data types. Object take up space in memory and have an associated address. We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types.

```
Car Car1;           // Declare Car1 of type Car
Car Car2;           // Declare Car2 of type Car
```

Example

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

class Car{
public:
    int price;
    int model_no;
    char name[20];

    void start(){
        cout<<"Grrrr...starting the car "<<name<<endl;
    }

};

int main() {

    Car C;
    //Initialisation
    C.price =500;
    C.model_no = 1001;
    C.name[0] = 'B';
```

```
    C.name[1] = 'M';
    C.name[2] = 'W';
    C.name[3] = '\0';
    C.start();


    //cout<<sizeof(C)<<endl; // C is an actual object 28 byte
s
    //cout<<sizeof(Car)<<endl; // It will take 28 bytes
    //Car C[100]; //Array an objects


    return 0;

}
```

Access Modifiers

- used to implement important feature of Object Oriented Programming known as Data Hiding.
- **What is Data Hiding** Consider a real life example: What happens when a driver applies brakes? The car stops. The driver only knows that to stop the car, he needs to apply the brakes. He is unaware of how actually the car stops. That is how the engine stops working or the internal implementation on the engine side. This is what data hiding is.
- Access modifiers or Access Specifiers in a class are used to set the

accessibility of the class members. That is, it sets some restrictions on the class members not to get directly accessed by the outside functions.

3 types of access modifiers available:

1. **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.

```
// 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;
        }
```

```
};

// 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. **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 are allowed to access the private data members of a class.

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

class Circle
{
    // private data member
    private:
        double radius = 1.0;

    // 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; // this will give compile error

    cout << "Area is:" << obj.compute_area();
    return 0;
}
```

OUTPUT

```
Area is:3.14
```

Access the private data members of a class indirectly using the public member functions of the class

```
// 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(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
```

3. **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.

```
// 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
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 derived class can
    // access the protected data members of base class
```

```
    obj1.setId(81);
    obj1.displayId();
    return 0;
}
```

OUTPUT

```
id_protected is:81
```

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.

Friend class and function in C++

Friend Class A friend class can access private and protected members of other class in which it is declared as friend. It is sometimes useful to allow a particular class to access private members of other class.

For example a LinkedList class may be allowed to access private members of Node.

```
class Node
{
private:
  int key;
  Node *next;
```

```
  /* Other members of Node Class */

  friend class LinkedList; // Now class  LinkedList can
                           // access private members of Node
};
```

Friend Function Like friend class, a friend function can be given special grant to access private and protected members. A friend function can be:
a) A method of another class
b) A global function

```
class Node
{
private:
  int key;
  Node *next;

  /* Other members of Node Class */
  friend int LinkedList::search(); // Only search() of linked
List
                                   // can access internal memb
ers
};
```

Getters and Setters

Getters and Setters allow you to effectively protect your data. This is a

technique used greatly when creating classes. For each variable, a `get()` method will return its value and a `set()` method will set the value.

If you decide that some action should be taken every time you change the value of a particular variable, you only need to edit the `set()` method instead of looking for every place you change the variable.

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

class Car{
private:
    int price;

public:
    int model_no;
    char name[20];

    void start(){
        cout<<"Grrrr...starting the car "<<name<<endl;
    }

    void setPrice(int p){
        if(p>1000){
            price = p;
        }else{
            price = 1000;
        }
```

```
    }

    int getPrice(){
        return price;
    }
};

int main() {

    Car C;
    //Initialisation
    //C.price =500;
    C.model_no = 1001;
    C.name[0] = 'B';
    C.name[1] = 'M';
    C.name[2] = 'W';
    C.name[3] = '\0';
    C.start();
    C.setPrice(1200);
    cout<<C.getPrice()<<endl;

    //cout<<sizeof(C)<<endl; // C is an actual object 28 byte
s
    //cout<<sizeof(Car)<<endl; // It will take 28 bytes
    //Car C[100]; //Array an objects


    return 0;
```

```
}
```

OUTPUT

```
Grrrr...starting the car BMW
1200
```