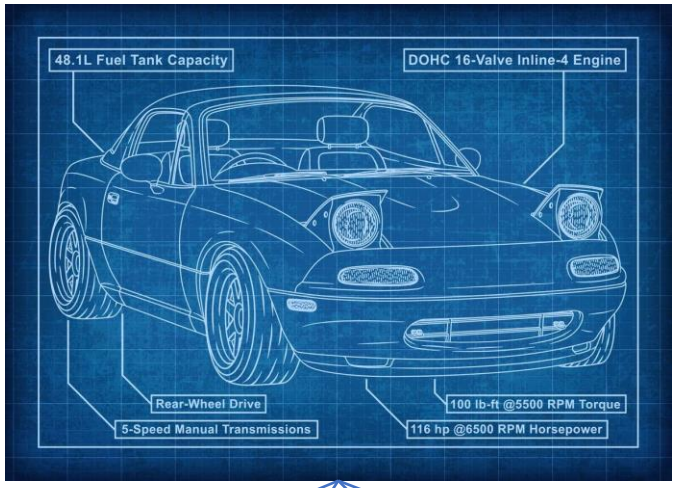
Unit-2

Concept	Definition/use
Class	• Abstract data type or user-defined data type • Logical grouping of data and methods that may modify the data • Collection of objects • Blueprint or specification
Data members	• Used to specify the state of the object
Member functions	• Used to define the behaviour of the object(i.e., how state is changed)
Object	• Instance of a class • Real-word entity
Access Specifiers	• Data hiding or protection of data
Constructors & Destructors	• Creating an object and Initialization of data members at the time of creation. • Deallocating the memory for the object
this pointer	• To refer current object • Implicitly provided to each member function
static	• To define class members(data and methods)
friend functions	• Providing access to class members to non-class member functions
Operator overloading	• To support the operations on user-defined data types(i.e.,class)

Class



Attributes

- name
- model
- seatingCapacity
- color
- fuelType
- Speed etc.,

Methods

- Drive()
- Brake()

Objects

- Mercedes-Benz
- AMT
- 5
- Safron
- petrol

- Ferrari
- Manual
- 2
- Blue
- petrol

- BMW
- Manual
- 4
- Blue
- Diesel

- Rolls-Royce
- Manual
- 2
- Red
- Petrol

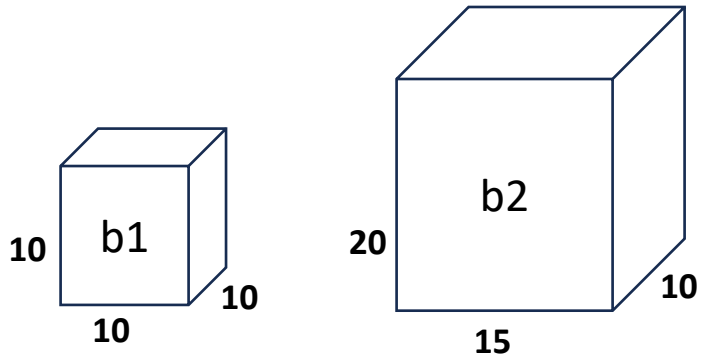
data members
(Attributes)

member functions
(Behaviour)

Object

```
class Box {
   double length,breadth,height;
 public:
   void initialize(double l,double b,double h){
      length=l;
      breadth=b;
      height=h;
   }
   double volume(){
      return length*breadth*height;
   }
   double perimeter(){
      return 4*(length+breadth+height);
   }
};
int main(){
   Box b1,b2;
   b1.initialize(10.0,10.0,10.0);
   b2.initialize(15.0,10.0,20.0);
   cout<<"Area of box1="<<b1.volume()<<endl;
   cout<<"Perimeter of box1="<<b1.perimeter()<<endl;
   cout<<“Area of box 2=“<<b2.volume()<<endl;
   cout<<“Perimeter of box2=“<<b2.perimeter<<endl;
   return 0;
}
```

```
class classname {
    access specifier:
    data members (data)
    access specifier:
    member functions (behaviour)
};
```



Access Specifiers

Why?	• By default C++ class members are defined *as private which are not accessible outside the class* . If it is not accessible what is the use of creating a class? So, *C++ provides access specifiers* to grant access *what members of the class are accessible outside the class*. • Provides data hiding or protection of data.

```
class A {
    int a;
    float b;
    void initialize(int x,float y){
        a=x;
        b=y;
    }
    void display(){
       cout<<a<< “  “<<b<<endl;
    }
};
```

```
int main(){
  //to use the class create an object of the class
  A obj;
  // initialize data member of the class
  obj.a=10; // error
  obj.b=23.4; //error
              (or)
  obj.initialize(10,23.4); //error
}
```

Access Specifiers

C++ provides three types of access specifiers for providing access to members of the class.

- private - No access outside of the class
- public - Access outside of the class using object
- protected - Access to restricted group of classes(not everyone) used in inheritance

Example

Account no	protected (known to you, bank employees , family members ,and some group)
Name	public (known to every one)
IFSC code	public (known to every one)
Bank balance	protected (known to you, bank employees , family members ,and some group)
ATM pin	private (known to you)
Userid	protected (known to you, bank employees , family members ,and some group)
Password	private (known to you)
Mobile Number	
Aadhar Number	
E-mail id	

Contd..

- Member functions can be declared as private , public and protected.
- Defining member functions as private is not recommended because , member functions are used to access or modify data members(which are declared as private).
- Defining Member functions as public can be accessible outside of the class through object.
- Defining Member functions as protected can be accessible in the successor classes only (inheritance)

```
class A {
        public:
        int x,y;
        void display(){
            cout<<x<<"  "<<y<<endl;
          }
};
int main(){
        A obj;
        obj.x=10;
        obj.y=20;
        obj.display();
        return 0;
}
```

```
class A {
        private:
        int x,y;
        void display(){
            cout<<x<<"  "<<y<<endl;
          }
};
int main(){
        A obj;
        obj.x=10; //error
        obj.y=20; //error
        obj.display(); //error
        return 0;
}
```

```
class A {
        protected:
        int x,y;
        void display(){
            cout<<x<<"  "<<y<<endl;
          }
};
int main(){
        A obj;
        obj.x=10; //error
        obj.y=20; // error
        obj.display(); //error
        return 0;
}
```

The recommended way for defining a class is

- *declare **data members(data)** as private and*
- *use **setter and getter methods** to access the data members of the class by declaring member functions as public*

```
class A {
        private:  int a,b;
        public:
                void seta(int x){
                      a=x;
                }
                int geta(){
                       return a;
                }
                 void setb(int x){
                      b=x;
                }
                int getb(){
                      return b;
                }
                 void display(){
                    cout<<a<<"   "<<b<<endl;
                 }
 };
```

```
int main(){
    A obj1;
    obj1.seta(10);
    obj1.setb(20);
    cout<<obj1.geta()<<endl;
    cout<<obj1.getb()<<endl;
    A obj2;
    obj2.seta(30);
    obj2.setb(50);
    cout<<obj2.geta()<<endl;
    cout<<obj2.getb()<<endl;
    obj2.display();
    return 0;
}
```

Changes to Object state

	a= b=	obj1
	a=10 b=	obj1
	a=10 b=20	obj1
Display a of obj1		
Display b of obj1		
	a= b=	obj2
	a=30 b=	obj2
	a=30 b=50	obj2
Display a of obj2		
Display b of obj2		
using display method		

Compiler creates default constructor and destructor for creating and destroying the objects

```
class A {
    int a,b;
    public:
        void init(int x,int y){
            a=x;b=y;
        }
        void display(){
            cout<<"a= "<<a<<"b= "<<b<<endl;
        }
};
int main(){
    A obj1;
    obj1.init(10,20);
    obj1.display();
    A obj2;
    obj2.init(30,40);
    obj2.display();
    return 0;
}
```

```
class A{
    int a,b;
    public:
        A(){ }
        ~A(){ }
        void init(int x,int y){
            a=x;b=y;
        }
        void display(){
            cout<<"a= "<<a<<"b= "<<b<<endl;
        }
};
int main(){
    A obj1;      // calls A::A()
    obj1.init(10,20);
    obj1.display();
    A obj2; // calls A::A()
    obj2.init(30,40);
    obj2.display();
    //calls A::~A() for obj2 and obj1
    return 0;
}
```

compiler creates default constructor and default destructor

this pointer

this pointer refers the current class object, which is implicitly provided by compiler.

```
using namespace std;
class A{
   int x,y;
   public:
     void init(int a,int b){
           x=a;y=b;
     }
     void display(){
           cout<<x<<"  "<<y<<"  "<<endl;
      }
};
int main(){
    A obj1,obj2;
    obj1.init(10,20);
    obj1.display();
    obj2.init(30,40);
    obj2.display();
    return 0;
}
```

Compiler inserts the ***this*** in binary code

```
using namespace std;
class A{
   int x,y;
   public:
     void init(A * const this, int a,int b){
           this->x=a;this->y=b;
     }
     void display(A * const this){
           cout<<this->x<<"  "<<this->y<<"  "<<endl;
      }
};
int main(){
    A obj1,obj2;
    obj1.init(&obj1,10,20);
    obj1.display(&obj1);
    obj2.init(&obj2,30,40);
    obj2.display(&obj2);
    return 0;
}
```

Illustration of ***this pointer*** is pointing to current object

```
using namespace std;
class A{
   int x,y;
   public:
     void init(int a,int b){
            x=a;
            y=b;
            cout<<"Address of object in init ="<<this<<endl;
     }
     void display(){
            cout<<x<<"  "<<y<<"  "<<endl;
      }
};
int main(){
    A obj1,obj2;
    cout<<"address of obj1 in main()="<<&obj1<<endl;
    obj1.init(10,20);
    obj1.display();
    cout<<"address of obj2 in main()="<<&obj2<<endl;
    obj2.init(30,40);
    obj2.display();
    return 0;
}
```

```
address of obj1 in main()=0x136e7ffc98
Address of object in init =0x136e7ffc98
10  20
address of obj2 in main()=0x136e7ffc90
Address of object in init =0x136e7ffc90
30  40
```

Defining member functions

Member functions can be defined

With in the class , implicitly considered as **inline**

Example:

```
class A {
        int a,b; // private data members
        public:
          void init(int x,int y){
                a=x;
                b=y;
          }
          void display(){
             cout<<a<<"   "<<b<<endl;
          }
};
```

Outside the class

- method prototype should be declared with in the class and
- definition is specified outside the class in the following way

syntax

```
return-type classname::function-name(argument list){
        /*.....*/
}
```

Example:

```
class A{
        int a,b; //private data members
        public:
           void init(int,int);
           void display();
};
void A::init(int x,int y) {
           a=x;b=y;
}
void A::display(){
        cout<<a<<"  "<<b<<endl;
}
```