Chapter 4. Concept of object

- ❑From C to C++, Classes and Objects
- ❑Protection, Access
- ❑Instantiation, Constructor, Destructor
- ❑Overloading
- ❑Operator “This”
- ❑Object and modeling UML/SysML
- ❑Automatic code generation

Classes and Objects

- A class is a blueprint or a template for creating objects. It defines a data structure along with the methods (functions) that can operate on that data.

- In essence, a class is a user-defined data type that encapsulates data (attributes or member variables) and the operations (member functions or methods) that can be performed on that data.

Example:

```
class Rectangle
{public:
         double length;
         double width;

        double area()
         { return length * width; }
};
```

In this example, a class named Rectangle has two member variables length and width, and one member function area that calculates the area of a rectangle.

An object, on the other hand, is an instance of a class. It is a concrete realization or instance created from the class blueprint.
You can create multiple objects from the same class, and each object will have its own set of member variables.
Example: creating objects of the Rectangle class:

```
Rectangle rectangle1;
Rectangle rectangle2;

rectangle1.length = 5.0;
rectangle1.width = 3.0;

rectangle2.length = 4.0;
rectangle2.width = 2.5;
```

You can access the member variables and call member functions on objects:

double area1 = rectangle1.area(); // Calculates the area of rectangle1double
area2 = rectangle2.area(); // Calculates the area of rectangle2

Protection, Access

- Access and protection in classes and objects are managed through access specifiers and member access control.
- C++ provides three access specifiers for controlling the visibility of class members: public, private, and protected.
- These access specifiers determine which parts of a class are accessible from outside the class and from derived classes.

public: Members declared as public are accessible from any part of the program, including code outside the class. This means that you can access public members both from within the class and from objects of the class.

Example:

```
class MyClass
{public:
        int publicVar;

      void publicFunction()
         {  // This function is accessible from outside the class.    }
};
```

You can access publicVar and call publicFunction from outside the class and from objects of the class.

private: Members declared as private are only accessible from within the class itself. They are not accessible from outside the class or from objects of the class.

Example:

```
class MyClass
     {private:
          int privateVar;

          void privateFunction()
            { // This function is only accessible from within the class.    }
};
```

You cannot directly access privateVar or call privateFunction from outside the class or from objects of the class.

protected: Members declared as protected are similar to private members but have additional visibility within derived classes (classes that inherit from the base class). They are not accessible from outside the class, but they can be accessed by derived classes.

Example:

```
class BaseClass
 {protected:
        int protectedVar;
        void protectedFunction()
          { // Accessible from derived classes but not from outside the class.    }
};

class DerivedClass : public BaseClass
 {    // Inherited protectedVar and protectedFunction can be accessed here.};
```

In addition to these access specifiers, you can control access to a class by making it a friend of another class, which allows the friend class to access the private and protected members of the class.

Constructor, Destructor

- A constructor is a special member function that gets called automatically when an object of a class is created.

- Constructors are used to initialize the state of objects, allocate resources, and perform other setup tasks necessary for the proper functioning of the object.

Key points about constructors:
Constructors have the same name as the class,
Constructors do not have a return type, not even void.
They are automatically called when an object is created.

Initialization: Constructors are primarily used to initialize the data members (class variables) of the object to appropriate initial values. This helps ensure that the object is in a consistent state when it is created.

Default Constructor: if there is no constructors provided for a class, C++ will automatically generate a default constructor with no parameters. This constructor initializes the data members with default values (zero for numeric types, null for pointers, etc.).

EX:

```
class MyClass
{public:
        MyClass()
       {  // Constructor code here    }};
```