Classes and Objects

Akash Hegde

Seventh Sense Talent Solutions

Vivekananda Institute of Technology

26 March 2021

Introduction

- Main purpose of C++ – to add object orientation to the C programming language.
- Classes are the central feature of C++ that supports object-oriented programming and are often called user-defined types.
- A class is used to specify the form of an object and it combines data representation and methods for manipulating that data into one neat package.
- The data and functions within a class are called members of the class.

Class Definition

- When you define a class, you define a blueprint for a data type.
- This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.
- 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.

Class Definition

- Example:

```
class Box {
      public:
            double length;        // Length of a box
            double breadth;      // Breadth of a box
            double height;        // Height of a box
};
```

- The keyword **public** determines the access attributes of the members of the class that follows it.
- A public member can be accessed from outside the class anywhere within the scope of the class object.

Object Definition

- A class provides the blueprints for objects, so basically an object is created from a class.
- We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types.
- Example:

```
Box Box1;    // Declare Box1 of type Box
Box Box2;    // Declare Box2 of type Box
```

- Both of the objects Box1 and Box2 will have their own copy of data members.

Accessing the Data Members

- The public data members of objects of a class can be accessed using the direct member access operator (.)
- Example:

```
#include <iostream>
using namespace std;
class Box {
        public:
                double length; // Length of a box
                double breadth; // Breadth of a box
                double height; // Height of a box
};

int main() {
        Box Box1; // Declare Box1 of type Box
        Box Box2; // Declare Box2 of type Box
        double volume = 0.0; // Store the volume of a box here
        // box 1 specification
        Box1.height = 5.0;
        Box1.length = 6.0;
        Box1.breadth = 7.0;
```

Accessing the Data Members

- Example(continued):

```
        // box 2 specification
        Box2.height = 10.0;
        Box2.length = 12.0;
        Box2.breadth = 13.0;
        // volume of box 1
        volume = Box1.height * Box1.length * Box1.breadth;
        cout << "Volume of Box1 : " << volume <<endl;
        // volume of box 2
        volume = Box2.height * Box2.length * Box2.breadth;
        cout << "Volume of Box2 : " << volume <<endl;
        return 0;
}
```

Class Member Functions

- A member function of a class is a function that has its definition or its prototype within the class definition like any other variable.
- It operates on any object of the class of which it is a member, and has access to all the members of a class for that object.
- Member functions can be defined within the class definition (inline, but without using **inline** specifier) or separately using **scope resolution operator.**

Class Member Functions

- Defining member function within class definition (inline):

```
class Box {
    public:
    double length; // Length of a box
    double breadth; // Breadth of a box
    double height; // Height of a box
    double getVolume(void) {
        return length * breadth * height;
    }
};
```

Class Member Functions

- Defining member function outside class definition:

```
class Box {
        public:
        double length; // Length of a box
        double breadth; // Breadth of a box
        double height; // Height of a box
};
double Box::getVolume(void) {
        return length * breadth * height;
 }
```

- A member function will be called using a dot operator (.) on an object where it will manipulate data related to that object only as follows –

```
Box myBox; // Create an object
myBox.getVolume(); // Call member function for the object
```

Class Member Functions

- Example:

```
#include <iostream>
using namespace std;
class Box {
    public:
        double length; // Length of a box
        double breadth; // Breadth of a box
        double height; // Height of a box
        // Member functions declaration
        double getVolume(void);
        void setLength( double len );
        void setBreadth( double bre );
        void setHeight( double hei );
};
```

Class Member Functions

- Example(continued):

```
// Member functions definitions
double Box::getVolume(void) {
      return length * breadth * height;
}
void Box::setLength( double len ) {
      length = len;
}
void Box::setBreadth( double bre ) {
      breadth = bre;
}
void Box::setHeight( double hei ) {
      height = hei;
}
```

Class Member Functions

- Example(continued):

```
// Main function for the program
int main() {
        Box Box1; // Declare Box1 of type Box
        Box Box2; // Declare Box2 of type Box
        double volume = 0.0; // Store the volume of box
        // box 1 specification
        Box1.setLength(6.0);
        Box1.setBreadth(7.0);
        Box1.setHeight(5.0);
        // box 2 specification
        Box2.setLength(12.0);
        Box2.setBreadth(13.0);
        Box2.setHeight(10.0);
```

Class Member Functions

- Example(continued):

```
      // volume of box 1
      volume = Box1.getVolume();
      cout << "Volume of Box1 : " << volume <<endl;
      // volume of box 2
      volume = Box2.getVolume();
      cout << "Volume of Box2 : " << volume <<endl;
      return 0;
}
```

Class Access Modifiers

- **Data hiding** is one of the important features of Object Oriented Programming which allows preventing the functions of a program to access directly the internal representation of a class type.
- The access restriction to the class members is specified by the labelled **public, private,** and **protected** sections within the class body.
- The keywords public, private, and protected are called access specifiers.

Class Access Modifiers

- A class can have multiple public, protected, or private labelled sections.
- Each section remains in effect until either another section label or the closing right brace of the class body is seen.
- The default access for members and classes is private.
- Example:

```
class Base {
    public:
        // public members go here
    protected:
        // protected members go here
    private:
        // private members go here
};
```

Class Access Modifiers - public

- A **public** member is accessible from anywhere outside the class but within a program.
- You can set and get the value of public variables without any member function.
- Example:

```
class Line {
      public:
            double length;
};
int main() {
      Line line;
      // set line length without member function
      line.length = 10.0; // OK: because length is public
}
```

Class Access Modifiers - private

- A **private** member variable or function cannot be accessed, or even viewed from outside the class.
- Only the class and friend functions can access private members.
- Example:

```
class Box {
      private:
          double width;
};
int main() {
      Box box;
      // set box width without member function
      box.width = 10.0; // Error: because width is private
}
```

Class Access Modifiers - protected

- A **protected** member variable or function is very similar to a private member but it provides one additional benefit that it can be accessed in child classes which are called derived classes.
- Example:

```
class Box {
    protected:
        double width;
};
class SmallBox:Box {        // SmallBox is the derived class.
    public:
        double getSmallWidth( void ) {
            return width ;
        }
};
```

Class Constructor

- A class **constructor** is a special member function of a class that is executed whenever we create new objects of that class.
- A constructor will have exact same name as the class and it does not have any return type at all, not even void.
- Constructors can be very useful for setting initial values for certain member variables.
- Example:

```
class Line {
      public:
            Line(); // This is the constructor
};
Line::Line(void) {
      cout << "Object is being created" << endl;
}
```

Class Parameterized Constructor

- A default constructor does not have any parameter, but if you need, a constructor can have parameters.
- This helps you to assign initial value to an object at the time of its creation.
- Example:

```
class Line {
      public:
          Line(double len); // This is the constructor
};
Line::Line(double len) {
      cout << "Object is being created, length = " << len;
}
```

Class Destructor

- A **destructor** is a special member function of a class that is executed whenever an object of its class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class.
- A destructor will have exact same name as the class prefixed with a tilde (~) and it can neither return a value nor can it take any parameters.
- Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories, etc.
- Example:

```
class Line {
      public:
            Line();     // This is the constructor
            ~Line();    // This is the destructor
};
Line::Line(void) {
      cout << "Object is being created" << endl;
}
Line::~Line(void) {
      cout << "Object is being deleted" << endl;
}
```

Friend Functions

- A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class.
- Even though the prototypes for friend functions appear in the class definition, friends are not member functions.
- A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all of its members are friends.

Friend Functions

- Example:

```
class Box {
	double width;
	public:
		friend void printWidth( Box box );
};
// printWidth() is not a member function of any class.
void printWidth( Box box ) {
	/* Because printWidth() is a friend of Box, it can
		directly access any member of this class */
	cout << "Width of box : " << box.width <<endl;
}
```

this Pointer

- Every object in C++ has access to its own address through an important pointer called **this** pointer.
- The **this** pointer is an implicit parameter to all member functions.
- Therefore, inside a member function, this may be used to refer to the invoking object.
- Friend functions do not have a **this** pointer, because friends are not members of a class.
- Only member functions have a **this** pointer.

this Pointer

- Example:

```
class Box {
    public:
    // Constructor definition
    Box(long l = 2.0, long b = 2.0, long h = 2.0) {
        cout <<"Constructor called." << endl;
        length = l; breadth = b; height = h;
    }
    long Volume() { return length * breadth * height; }
    int compare(Box box) {
        return this->Volume() > box.Volume();
    }
}
```

Thank you!