Classes and Object-Oriented Programming [C++]

- The main purpose of C++ programming is to add object orientation to the C programming language and 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.

C++ Class Definitions

- 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. Typically a class *definition* always starts with a capitol letter. A class definition must be followed either by a semicolon or a list of declarations, e.g.

```
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 follow it. A public member can be accessed from outside the class anywhere within the scope of the class object. You can also specify the members of a class as **private** or **protected** which we will discuss later in these notes.

Define C++ Objects

- 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. Following statements declare two objects of class Box,

```
     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 Data Members

The public data members of objects of a class can be accessed using the direct member access operator (.) or “dot” syntax, e.g.,

```
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;

   // 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;
}
```

- At this early stage a `class` looks very similar to a `struct`.
- It is important to note that private and protected members can not be accessed directly using direct member access operator (.).

C++ Class member functions (AKA methods)

A member function of a class is a function that has its definition 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.

Let us take previously defined class to access the members of the class using a member function instead of directly accessing them:

```
class Box
```

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

          double getVolume()
          {
               return length*breadth*height;
          }
};

...

// 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;
```

- Member functions of a class are usually referred to as **methods** rather than **functions**.
- A method is similar to a function—but is internal to part of a class. The term method is used almost exclusively in object-oriented programming.
- Methods can be defined within the class definition or separately using the **scope resolution operator** (**::**), in which case the above class would look like so,

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

double Box::getVolume()
{
     return length*breadth*height;
}
```

- The main thing here is you have to use the class name just before **::** operator.
- Note the method must still be declared in the class, much like a C prototype.

Accessors and Mutators

- Data hiding is one of the important features of Object Oriented Programming which allows preventing 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 labeled **public**, **private**, and **protected** sections within the class body. Those keywords are called access specifiers.

```
class Base {

   public:

   // public members go here

   protected:

   // protected members go here

   private:

   // private members go here

};
```

- 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.
- A private member variable or function cannot be accessed, or even viewed from outside the class. By default all the members of a class are private.
- A protected member variable or function is very similar to a private member but it provided one additional benefit that they can be accessed in child classes which are called derived classes. Derived classes and inheritance will be covered later.
- By convention typically a private member was indicated with the prefix `m_` to show that it's a member variable. Though this is an older convention, later people omitted the m altogether and just use an underscore _. The use of _ as a prefix for private member became very popular in the Java language. Some specifics of the C++ language involving templates the leading _ becomes confusing or even hazardous, so some C++ programmers use a trailing _ instead.
- An accessor is a member function that allows someone to retrieve the contents of a private (or protected) data member. Accessor functions typically beginning with the "get" prefix.
- A mutator is a member function that allows for editing of the contents of a private (or protected) data member. Mutator functions typically beginning with the "set" prefix.
- Mutators and accessors are used to ensure encapsulation into and out of private class variables. By looking at what code uses the mutator or accessor you can see exactly who, what, and were changes or reads are being made to your class and you have much more control, control which is often needed when collaborating in a group.
- Accessors and mutators into a private variable typically look like this,

```
    class Base
    {

          public:
            // public members go here
            void setBreadth(double value)
            {
                 mBreadth = value;
            }

            double getBreadth()
            {
                 return mBreadth;
            }

       private:
            // private members go here
            double mBreadth;

  };
```

The Class Constructor and Destructor

- 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 as a constructor can have parameters. This helps you to assign initial value to an object at the time of its creation.
- A destructor is a special member function of a class that is executed whenever an object of it's 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.

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.

Here is the Box `class` example where constructors with parameters are used to create Box objects. It also includes an example `compare()` function to illustrate the use of the `this` pointer.

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

```
class Box
{
    public:
    Box(double length,      // constructor
        double breadth,
        double height)
    {
        cout << "Box object being created." << endl;
        mLength = length;
        mBreadth = breadth;
        mHeight = height;
    }

    ~Box()                  // destructor
    {
        cout << "Box object being deleted." << endl;
    }

    double getVolume()
    {
        return mLength*mBreadth*mHeight;
    }

    bool compare(Box *box)
    {
        if (this->getVolume() > box->getVolume())
        {
            return true;
        }
        return false;
    }

    private:
    double mLength;   // Length of a box
    double mBreadth;  // Breadth of a box
    double mHeight;   // Height of a box
};



int main( )
{
    Box box1(5.0, 6.0, 7.0);        // Declare box1 of type Box
    Box box2(10.0, 12.0, 13.0);     // Declare box2 of type Box
    double volume;

    // 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;

    if (box1.compare(&box2))
    {
        cout << "box2 is smaller than box1" << endl;
    }
    else
    {
        cout << "box2 is equal to or larger than box1" << endl;
    }

    return 0;
}
```

new and ***delete*** constructs

- In the C++ programming language, `new` and `delete` are a pair of language constructs that perform dynamic memory allocation, object construction and object destruction.
- `new` and `delete` are the C++ equivalents of (or at least very similar to), the C commands `malloc` and `free`.
- the `new` operator denotes a request for memory allocation on a process's heap. If sufficient memory is available, new initializes the memory, calling object constructors if necessary, and returns a pointer address to the newly allocated and initialized memory.
- The deallocation counterpart of new is `delete`, which first calls the destructor (if any) on its argument and then returns the memory allocated by new back to the free store. Every call to `new` must be matched by a call to `delete`; failure to do so causes memory leaks.
- A function call-like syntax is used to call a class constructor and pass it arguments.

Here is the `main()` function of the Box class example above but using `new` and `delete` constructs and pointers. The `new` and `delete` constructs give you complete control of your memory.

```
int main( )
{
    // memory allocated with new
    Box *ptrBox1 = new Box(5.0, 6.0, 7.0);
    Box *ptrBox2 = new Box(10.0, 12.0, 13.0);
    double volume;

    // volume of box 1
    volume = ptrBox1->getVolume();
    cout << "Volume of box1 : " << volume << endl;
    // volume of box 2
    volume = ptrBox2->getVolume();
    cout << "Volume of Box2 : " << volume << endl;

    if (ptrBox1->compare(*ptrBox2))
    {
        cout << "box2 is smaller than box1" << endl;
    }
    else
    {
        cout << "box2 is equal to or larger than box1" << endl;
    }

    // Note: if delete not called a memory leak occurs!
    delete ptrBox1;
    delete ptrBox2;

    return 0;
}
```