CL-217 OBJECT ORIENTED PROGRAMMING LAB

LAB Manual 6

INSTRUCTOR: MUHAMMAD HAMZA

SEMESTER SPRING 2020

CLASSES

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. A class definition must be followed either by a semicolon or a list of declarations. For example, we defined the Box data type using the keyword class as follows −

```
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. You can also specify the members of a class as private or protected which we will discuss in a sub-section.

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 the Data Members

The public data members of objects of a class can be accessed using the direct member access operator (.). Let us try the following example to make the things clear −

Live Demo

```
#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;
```

```
   // 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;
}
```

When the above code is compiled and executed, it produces the following result −

```
Volume of Box1 : 210
Volume of Box2 : 1560
```

It is important to note that private and protected members can not be accessed directly using direct member access operator (.). We will learn how private and protected members can be accessed.

Member Function:

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.

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(void);// Returns box volume

};
```

Member functions can be defined within the class definition or separately using scope resolution operator, : −. Defining a member function within the class definition declares the function inline, even if you do not use the inline specifier. So either you can define Volume() function as below −

```
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;

      }

};
```

If you like, you can define the same function outside the class using the scope resolution operator (::) as follows −

```
double Box::getVolume(void) {

   return length * breadth * height;
```

```
}
```

Here, only important point is that you would have to use class name just before :: operator. A member function will be called using a dot operator (.) on a 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
```

Let us put above concepts to set and get the value of different class members in a class −

Live Demo

```
#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 );
```

```
};

// 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;
}

// 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 a box here

   // 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);

   // 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;
}
```

When the above code is compiled and executed, it produces the following result −

```
Volume of Box1 : 210
Volume of Box2 : 1560
```

Access Modifier:

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 labeled public, private, and protected sections within the class body. The keywords public, private, and protected are called access specifiers.

A class can have multiple public, protected, or private labeled 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.

```
class Base {

   public:

      // public members go here

      protected:


   // protected members go here

   private:

   // private members go here


};
```

The public Members

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 as shown in the following example −

Live Demo

```
#include <iostream>


using namespace std;


class Line {

   public:

      double length;

      void setLength( double len );
```

```
      double getLength( void );
};

// Member functions definitions
double Line::getLength(void) {
   return length ;
}

void Line::setLength( double len) {
   length = len;
}

// Main function for the program
int main() {
   Line line;

   // set line length
   line.setLength(6.0);
   cout << "Length of line : " << line.getLength() <<endl;

   // set line length without member function
   line.length = 10.0; // OK: because length is public
   cout << "Length of line : " << line.length <<endl;

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Length of line : 6

Length of line : 10
```

The private Members

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.

By default all the members of a class would be private, for example in the following class width is a private member, which means until you label a member, it will be assumed a private member −

```
class Box {

   double width;


   public:

      double length;

      void setWidth( double wid );

      double getWidth( void );

};
```

Practically, we define data in private section and related functions in public section so that they can be called from outside of the class as shown in the following program.

Live Demo

```
#include <iostream>


using namespace std;


class Box {
```

```
   public:
      double length;
      void setWidth( double wid );
      double getWidth( void );

   private:
      double width;
};

// Member functions definitions
double Box::getWidth(void) {
   return width ;
}

void Box::setWidth( double wid ) {
   width = wid;
}

// Main function for the program
int main() {
   Box box;

   // set box length without member function
   box.length = 10.0; // OK: because length is public
   cout << "Length of box : " << box.length <<endl;
```

```
   // set box width without member function
   // box.width = 10.0; // Error: because width is private
   box.setWidth(10.0);  // Use member function to set it.
   cout << "Width of box : " << box.getWidth() <<endl;

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Length of box : 10
Width of box : 10
```

The protected Members

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.

You will learn derived classes and inheritance in next chapter. For now you can check following example where I have derived one child class SmallBox from a parent class Box.

Following example is similar to above example and here width member will be accessible by any member function of its derived class SmallBox.

Live Demo

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

class Box {
   protected:
      double width;
};
```

```
class SmallBox:Box { // SmallBox is the derived class.
   public:
      void setSmallWidth( double wid );
      double getSmallWidth( void );
};

// Member functions of child class
double SmallBox::getSmallWidth(void) {
   return width ;
}

void SmallBox::setSmallWidth( double wid ) {
   width = wid;
}

// Main function for the program
int main() {
   SmallBox box;

   // set box width using member function
   box.setSmallWidth(5.0);
   cout << "Width of box : "<< box.getSmallWidth() << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Width of box : 5
```

Class Constructor and Destructor:

The 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.

Following example explains the concept of constructor −

Live Demo

```
#include <iostream>

using namespace std;

class Line {
   public:
      void setLength( double len );
      double getLength( void );
      Line();  // This is the constructor
   private:
      double length;
};

// Member functions definitions including constructor
Line::Line(void) {
```

```
   cout << "Object is being created" << endl;
}
void Line::setLength( double len ) {
   length = len;
}
double Line::getLength( void ) {
   return length;
}

// Main function for the program
int main() {
   Line line;

   // set line length
   line.setLength(6.0);
   cout << "Length of line : " << line.getLength() <<endl;

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Object is being created
Length of line : 6
```

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 as shown in the following example −

Live Demo

```
#include <iostream>

using namespace std;
class Line {
   public:
      void setLength( double len );
      double getLength( void );
      Line(double len);  // This is the constructor

   private:
      double length;
};

// Member functions definitions including constructor
Line::Line( double len) {
   cout << "Object is being created, length = " << len << endl;
   length = len;
}
void Line::setLength( double len ) {
   length = len;
}
double Line::getLength( void ) {
   return length;
```

```
}

// Main function for the program
int main() {
   Line line(10.0);

   // get initially set length.
   cout << "Length of line : " << line.getLength() <<endl;

   // set line length again
   line.setLength(6.0);
   cout << "Length of line : " << line.getLength() <<endl;

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Object is being created, length = 10
Length of line : 10
Length of line : 6
```

Using Initialization Lists to Initialize Fields

In case of parameterized constructor, you can use following syntax to initialize the fields −

```
Line::Line( double len): length(len) {
   cout << "Object is being created, length = " << len << endl;
}
```

Above syntax is equal to the following syntax −

```
Line::Line( double len) {
   cout << "Object is being created, length = " << len << endl;
   length = len;
}
```

If for a class C, you have multiple fields X, Y, Z, etc., to be initialized, then use can use same syntax and separate the fields by comma as follows −

```
C::C( double a, double b, double c): X(a), Y(b), Z(c) {
   ....
}
```

The Class Destructor

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.

Following example explains the concept of destructor −

Live Demo

```
#include <iostream>

using namespace std;
class Line {
   public:
```

```
      void setLength( double len );
      double getLength( void );
      Line();   // This is the constructor declaration
      ~Line();  // This is the destructor: declaration

   private:
      double length;
};

// Member functions definitions including constructor
Line::Line(void) {
   cout << "Object is being created" << endl;
}
Line::~Line(void) {
   cout << "Object is being deleted" << endl;
}
void Line::setLength( double len ) {
   length = len;
}
double Line::getLength( void ) {
   return length;
}

// Main function for the program
int main() {
   Line line;
```

```
   // set line length
   line.setLength(6.0);
   cout << "Length of line : " << line.getLength() <<endl;

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Object is being created
Length of line : 6
Object is being deleted
```

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.

Let us try the following example to understand the concept of this pointer −

Live Demo

```
#include <iostream>

using namespace std;

class Box {
   public:
      // Constructor definition
```

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

   private:
      double length;     // Length of a box
      double breadth;    // Breadth of a box
      double height;     // Height of a box
};

int main(void) {
   Box Box1(3.3, 1.2, 1.5);    // Declare box1
   Box Box2(8.5, 6.0, 2.0);    // Declare box2

   if(Box1.compare(Box2)) {
      cout << "Box2 is smaller than Box1" <<endl;
   } else {
```

```
      cout << "Box2 is equal to or larger than Box1" <<endl;
   }

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Constructor called.
Constructor called.
Box2 is equal to or larger than Box1
```

Pointer to C++ Class:

A pointer to a C++ class is done exactly the same way as a pointer to a structure and to access members of a pointer to a class you use the member access operator -> operator, just as you do with pointers to structures. Also as with all pointers, you must initialize the pointer before using it.

Let us try the following example to understand the concept of pointer to a class −

Live Demo

```
#include <iostream>

using namespace std;

class Box {
   public:
      // Constructor definition
      Box(double l = 2.0, double b = 2.0, double h = 2.0) {
         cout <<"Constructor called." << endl;
         length = l;
```

```
         breadth = b;
         height = h;
      }
      double Volume() {
         return length * breadth * height;
      }

   private:
      double length;      // Length of a box
      double breadth;     // Breadth of a box
      double height;      // Height of a box
};

int main(void) {
   Box Box1(3.3, 1.2, 1.5);     // Declare box1
   Box Box2(8.5, 6.0, 2.0);     // Declare box2
   Box *ptrBox;                 // Declare pointer to a class.

   // Save the address of first object
   ptrBox = &Box1;

   // Now try to access a member using member access operator
   cout << "Volume of Box1: " << ptrBox->Volume() << endl;

   // Save the address of second object
   ptrBox = &Box2;
```

```
   // Now try to access a member using member access operator
   cout << "Volume of Box2: " << ptrBox->Volume() << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Constructor called.
Constructor called.
Volume of Box1: 5.94
Volume of Box2: 102
```

Static Members of a Class:

We can define class members static using static keyword. When we declare a member of a class as static it means no matter how many objects of the class are created, there is only one copy of the static member.

A static member is shared by all objects of the class. All static data is initialized to zero when the first object is created, if no other initialization is present. We can't put it in the class definition but it can be initialized outside the class as done in the following example by redeclaring the static variable, using the scope resolution operator :: to identify which class it belongs to.

Let us try the following example to understand the concept of static data members −

Live Demo

```
#include <iostream>

using namespace std;

class Box {
```

```
   public:
      static int objectCount;

      // Constructor definition
      Box(double l = 2.0, double b = 2.0, double h = 2.0) {
         cout <<"Constructor called." << endl;
         length = l;
         breadth = b;
         height = h;

         // Increase every time object is created
         objectCount++;
      }
      double Volume() {
         return length * breadth * height;
      }

   private:
      double length;     // Length of a box
      double breadth;    // Breadth of a box
      double height;     // Height of a box
};

// Initialize static member of class Box
int Box::objectCount = 0;
```

```
int main(void) {

   Box Box1(3.3, 1.2, 1.5);    // Declare box1

   Box Box2(8.5, 6.0, 2.0);    // Declare box2


   // Print total number of objects.

   cout << "Total objects: " << Box::objectCount << endl;


   return 0;

}
```

When the above code is compiled and executed, it produces the following result −

```
Constructor called.

Constructor called.

Total objects: 2
```

Static Function Members

By declaring a function member as static, you make it independent of any particular object of the class. A static member function can be called even if no objects of the class exist and the static functions are accessed using only the class name and the scope resolution operator ::.

A static member function can only access static data member, other static member functions and any other functions from outside the class.

Static member functions have a class scope and they do not have access to the this pointer of the class. You could use a static member function to determine whether some objects of the class have been created or not.

Let us try the following example to understand the concept of static function members −

Live Demo

```
#include <iostream>
```

```
using namespace std;

class Box {
   public:
      static int objectCount;

      // Constructor definition
      Box(double l = 2.0, double b = 2.0, double h = 2.0) {
         cout <<"Constructor called." << endl;
         length = l;
         breadth = b;
         height = h;

         // Increase every time object is created
         objectCount++;
      }
      double Volume() {
         return length * breadth * height;
      }
      static int getCount() {
         return objectCount;
      }

   private:
      double length;     // Length of a box
      double breadth;    // Breadth of a box
```

```
      double height;     // Height of a box
};

// Initialize static member of class Box
int Box::objectCount = 0;

int main(void) {
   // Print total number of objects before creating object.
   cout << "Inital Stage Count: " << Box::getCount() << endl;

   Box Box1(3.3, 1.2, 1.5);    // Declare box1
   Box Box2(8.5, 6.0, 2.0);    // Declare box2

   // Print total number of objects after creating object.
   cout << "Final Stage Count: " << Box::getCount() << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Inital Stage Count: 0
Constructor called.
Constructor called.
Final Stage Count: 2
```