Classes and Objects

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 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 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.

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:

```
 #include <iostream>

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;

 }
```

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.

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.

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 same function outside the class using 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:

```
#include <iostream>

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( )
```

```
{

B  ox 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;

}
```

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 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:

```
 #include <iostream>

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;
}
```

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.

```
#include <iostream>
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;

}
```

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 can check following example where I have derived one child class Small Box 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 Small Box.

```
#include <iostream>

 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;
}
```