Classes and Objects in C++

Class:

- The class are the most important feature of C++ that leads to Object Oriented programming.
- Class is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating instance or object of that class.
- The variables inside class definition are called as data members and the functions are called member functions.
- Class name must start with an uppercase letter. If class name is made of more than one word, then first letter of each word must be in uppercase.

  Syntax to Defining a class

  ```
  class ClassName
  {
      Access Specifier: Data Member
      Access Specifier: Member Function
  };
  ```

Objects

- Objects are instances of class or runtime entity, which holds the data variables and functions declared in class together.
- Each object has different data variables.
- Objects are initialized using special class functions called **Constructors or by member function**.
- And whenever the object is out of its scope, another special class member function called **Destructor** is called, to release the memory reserved by the object.
- C++ doesn't have Automatic Garbage Collector like in JAVA, in C++ Destructor performs this task.

Syntax to Declaring Object

```
ClassName objectname;
```

Example 1: If Data member is Public.

```
#include<iostream>
using namespace std;
class Box
{
        public: int length, width, height;
};
int main()
{
        Box b1;
```

```
        b1.length=13;
        b1.width=15;
        b1.height=16;
        cout<<endl<<"Box Dimension:"<<b1.length<<"x"<<b1.width<<"x"<<b1.height;
        cout<<endl<<"Box Volume:"<<b1.length*b1.width*b1.height;
}
```

Example 2: If Data Member is Private.

```
#include<iostream>
using namespace std;
class Box
{
        private: int length, width, height;
};
int main()
{
        Box b1;
        b1.length=13;
        b1.width=15;
        b1.height=16;
        cout<<endl<<"Box Dimension:"<<b1.length<<"x"<<b1.width<<"x"<<b1.height;
        cout<<endl<<"Box Volume:"<<b1.length*b1.width*b1.height;
}
```

Above program gives error because we cannot access private member outside the class. It can be access only by its own member function.

Defining Member Function:

There are two way defining a function.

1. Inside Class Definition
2. Outside Class Definition

1. Inside Class Definition

```
#include<iostream>
using namespace std;
class Box
{
        private: int length, width, height;
        public:         void setLength(int l)
                        {
                                length=l;
                        }
                        void setWidth(int w)
                        {
                                width=w;
                        }
                        void setHeight(int h)
```

```
				{
					height=h;
				}
				void showDimension()
				{
					cout<<length<<"x"<<width<<"x"<<height;
				}
				int getVolume()
				{
					return length*width*height;
				}
};
int main()
{
	Box b1;
	b1.setLength(13);
	b1.setWidth(14);
	b1.setHeight(15);
	cout<<endl<<"Box Dimension:";
	b1.showDimension();
	cout<<endl<<"Box Volume:"<<b1.getVolume();
}
```

2. Outside Class Definition:

As the name suggests, here the functions are defined outside the class however they are declared inside the class. Functions should be declared inside the class to bound it to the class and indicate it as it's member but they can be defined outside of the class. To define a function outside of a class, scope resolution operator:: is used.

```
#include<iostream>
using namespace std;
class Box
{
	public:
			double length, breadth, height;
			double getVolume();

};
double Box::getVolume()
{
	return length*breadth*height;
}
int main()
{
	Box b1;
	b1.length=5.5;
```

```
        b1.breadth=6.5;
        b1.height=7.5;
        cout<<"Box volume:"<<b1.getVolume();
}
```

Access Specifiers

The access modifier of C++ allows us to determine which class members are accessible to other classes and functions, and which are not. There are 3 types of Access Specifiers as

1. **Public**
2. **Private**
3. **Protected**

Public: All the class members declared under the public specifier will be available to everyone. The data members and member functions declared as public can be accessed by other classes and functions too. The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.

Example: The above 2 programs are the example of public access specifier.

Private: The class members declared as *private* can be accessed only by the member functions inside the class. They are not allowed to be accessed directly by any object or function outside the class. Only the member functions or the friend functions are allowed to access the private data members of a class.

Example Program:

```
#include<iostream>
using namespace std;
class Box
{
        private:
                        double length;
                        double breadth;
                        double height;
        public:
                        double findvolume()
                        {
                                return length*breadth*height;
                        }
};
int main()
{
        Box b1;
        b1.length=5.5;
        b1.breadth=7.5;
        b1.height=4.5;
        cout<<"\n Box b1 Volume:"<<b1.findvolume();

}
```

The above program gives error because data member length, breadth and height declare by private access specifier which is access by only its member function. So the correct program as to access private member.

Program

```
#include<iostream>
using namespace std;
class Box
{
	private:
			double len;
			double breadth;
			double height;

	public:
			void setLength(double l)
			{
				len=l;
			}
			void setBreadth(double b)
			{
				breadth=b;
			}
			void setHeight(double h)
			{
				height=h;
			}
			double findvolume()
			{
				return len*breadth*height;
			}
};
int main()
{
	Box b1;
	b1.setLength(5.5);
	b1.setBreadth(7.5);
	b1.setHeight(4.5);
	cout<<"\n Box b1 Volume:"<<b1.findvolume();

}
```

Protected: Protected access modifier is similar to private access modifier in the sense that it can’t be accessed outside of its class unless with the help of friend class, the difference is that the class members declared as Protected can be accessed by any subclass (derived class) of that class as well.

Constructor in C++

C++ constructors are *special* member functions which are created when the object is created or defined and its task is to initialize the object of its class. It is called **constructor** because it constructs the values of data members of the class.
A constructor has the same name as the class and it doesn't have any return type. It is invoked whenever an object of its associated class is created.

Syntax

Types of Constructor

- Default Constructor
- Parameterized Constructor
- Copy Constructor

Default Constructor:

If no constructor is defined in the class then the compiler automatically creates one for the program. This constructor which is created by the compiler when there is no user defined constructor and which doesn't take any parameters is called **default constructor**.

Parameterized constructor

To put it up simply, the constructor that can take arguments are called parameterized constructor.
In practical programs, we often need to initialize the various data elements of the different object with different values when they are created. This can be achieved by passing the arguments to the constructor functions when the object is created.
Note: Remember that constructor is always defined and declared in public section of the class and we can't refer to their addresses.

Copy Constructor

Generally in a constructor the object of its own class can't be passed as a value parameter. But the classes own object can be passed as a reference parameter. Such constructor having reference to the object of its own class is known as copy constructor.
Moreover, it creates a new object as a copy of an existing object. For the classes which do not have a copy constructor defined by the user, compiler itself creates a copy constructor for each class known as default copy constructor.
Note: In copy constructor passing an argument by value is not possible.

Program

```
#include<iostream>
using namespace std;
class Box
{
	private: int length, width, height;
	public:
			//Default Constructor
			Box()
			{
				length=width=height=5;
```

```
                    }
                    //parameterised constructor
                    Box(int l, int w, int h)
                    {
                            length=l;
                            width=w;
                            height=h;
                    }
                    //copy constructor
                    Box(Box &b)
                    {
                            length=b.length;
                            width=b.width;
                            height=b.height;
                    }
                    void showDimension()
                    {
                            cout<<length<<"x"<<width<<"x"<<height;
                    }
                    int getVolume()
                    {
                            return length*width*height;
                    }

};
int main()
{
        Box b1; //default constructor
        Box b2(5,6,7); // parameterised constructor
        Box b3(b2); // copy constructor

        cout<<endl<<"Box b1:";
        cout<<endl<<"Box b1 Dimesion:";
        b1.showDimension();
        cout<<endl<<"Box Volume"<<b1.getVolume();

        cout<<endl<<"Box b2:";
        cout<<endl<<"Box b2 Dimesion:";
        b2.showDimension();
        cout<<endl<<"Box Volume"<<b2.getVolume();

        cout<<endl<<"Box b3:";
        cout<<endl<<"Box b3 Dimesion:";
        b3.showDimension();
        cout<<endl<<"Box Volume"<<b3.getVolume();
}
```

Destructor in C++:

A destructor is a special member function that works just opposite to constructor, unlike constructors that are used for initializing an object, destructors destroy (or delete) the object.

Syntax of Destructor

```
~class_name()
{
  //Some code
}
```

When does the destructor get called?

A destructor is automatically called when:

1) The program finished execution.
2) When a scope (the { } parenthesis) containing local variable ends.
3) When you call the delete operator.

```
#include<iostream>
using namespace std;
class Box
{
	private: int length, width, height;
	public:
			//Default Constructor
			Box()
			{
				length=width=height=5;
			}
			//destructor
			~Box()
			{
				cout<<endl<<"Object Destroyed";
			}
			void showDimension()
			{
				cout<<length<<"x"<<width<<"x"<<height;
			}
			int getVolume()
			{
				return length*width*height;
			}

};
int main()
{
	Box b1; //default constructor

	cout<<endl<<"Box b1:";
```

```
        cout<<endl<<"Box b1 Dimesion:";
        b1.showDimension();
        cout<<endl<<"Box Volume"<<b1.getVolume();
}
```

this pointer

The **this** pointer holds the address of current object, in simple words you can say that this pointer points to the current object of the class. Let's take an example to understand this concept.

Example

Here you can see that we have two data members num and ch. In member function setMyValues() we have two local variables having same name as data members name. In such case if you want to assign the local variable value to the data members then you won't be able to do until unless you use this pointer, because the compiler won't know that you are referring to object's data members unless you use this pointer. This is one of the example where you must use **this** pointer.

Example Program

```
#include <iostream>
using namespace std;
class Demo
{
        private:
                 int num;
                 char ch;
        public:
                void setMyValues(int num, char ch)
                {
                        this->num = num;
                        this->ch = ch;
                }
                void displayMyValues()
                {
                         cout<<num<<endl;
                        cout<<ch;
                }
};
int main()
{
        Demo d1;
        d1.setMyValues(100, 'A');
        d1.displayMyValues();
}
```