C++ Objects and Classes

An Overview about Objects and Classes

In object-oriented programming language C++, the data and functions (procedures to manipulate the data) are bundled together as a self-contained unit called an *object*. A *class* is an extended concept similar to that of *structure* in C programming language, this class describes the data properties alone. In C++ programming language, *class* describes both the properties (data) and behaviors (functions) of objects. *Classes* are not *objects*, but they are used to instantiate *objects*.

Features of Class:

Classes contain data known as members and member functions. As a unit, the collection of members and member functions is an object. Therefore, this unit of objects makes up a class.

How to write a Class:

In Structure in C programming language, a structure is specified with a name. The C++ programming language extends this concept. A class is specified with a name after the keyword class.

The starting flower brace symbol {is placed at the beginning of the code. Following the flower brace symbol, the body of the class is defined with the member functions data. Then the class is closed with a flower brace symbol} and concluded with a colon;

```
class demo
{
  data;
  member functions;
  ..............
};
```

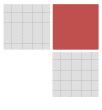
There are different access specifiers for defining the data and functions present inside a class.

Access specifiers:

Access specifiers are used to identify access rights for the data and member functions of the class. There are three main types of access specifiers in C++ programming language:

- ☑ private
- ☑ public
- ☑ protected

Note: A *private* member within a class denotes that only members of the same class have accessibility. The *private* member is inaccessible from outside the class.



Note: *Public* members are accessible from outside the class.

Note: A protected access specifier is a stage between *private* and *public* access. If member functions defined in a class are *protected*, they cannot be accessed from outside the class but can be accessed from the derived class.

Note: When defining access specifiers, the programmer must use the keywords: *private*, *public* or *protected* when needed, followed by a semicolon and then define the data and member functions under it.

```
class demo
{
  private:
  int x,y;
  public:
  void sum()
  {
    ……..
    ……..
  }
};
```

In the code above, the member *x* and *y* are defined as private access specifiers. The member function sum is defined as a public access specifier.

General Template of a class:

General structure for defining a class is:

```
class classname
{
  acess specifier:
  data member;
  member functions;

  acess specifier:
  data member;
  member functions;
};
```

Generally, in class, all members (data) would be declared as private and the member functions would be declared as public. Private is the default access level for specifiers. If no access specifiers are identified for members of a class, the members are defaulted to private access.

```
class demo
{
  int x,y;
  public:
  void sum()
  {
    .........
    .........
  }
};
```

In this example, for members x and y of the class *demo* there are no access specifiers identified. *demo* would have the default access specifier as private.

Creation of Objects:

Once the class is created, one or more objects can be created from the class as objects are instance of the class.

```
int x;
```

Objects are also declared as:

```
class name followed by object name;
```

```
demo e1;
```

This declares *e1* to be an object of class demo.

For example a complete class and object declaration is given below:

```
class demo
{
  private:
  int x,y;
  public:
  void sum()
  {
    .........
    .........
  }
};

main()
{
  demo e1;
  ..............
  ..............
}
```

The object can also be declared immediately after the class definition. In other words the object name can also be placed immediately before the closing flower brace symbol } of the class declaration.

For example -

```
class demo
{
  private:
  int x,y;
  public:
  void sum()
  {
    .........
    .........
  }
}e1 ;
```

The above code also declares an object *e1* of class demo.

Note: It is important to understand that in object-oriented programming language, when a class is created no memory is allocated. It is only when an object is created is memory then allocated.

How to Access C++ Class Members

In this C++ tutorial, you will learn how to access Class members, dot operator or class member access operator, difference between struct and class and scope resolution operator. It is possible to access the class members after a class is defined and objects are created. General syntax to access class member -

```
Object_name.function_name (arguments);
```

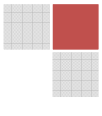
The dot ('.') used above is called the *dot operator* or *class member access operator*. The dot operator is used to connect the object and the member function. This concept is similar to that of accessing structure members in C programming language. The private data of a class can be accessed only through the member function of that class.

For example,

```
class demo
{
  int a, b;
  public:
  void sum(int,int);
} e1;
```

Then the member access is written as: **e1.sum(5,6);**

Where e1 is the object of class demo and sum() is the member function of the class.



The programmer now understands declaration of a *class*, creation of an object and accessibility of members of a *class*. It is also possible to declare more than one object within a *class*:

```
class demo
{
  private:
  int a;
  public:
  void sum(int)
  {
    .........
    .........
  }
};

main()
{
  demo e1,e2;
  ...........
  ...........
}
```

In these two objects e1 and e2 are declared of class demo.

Note: By default, the access specifier for members of a class is private. The default access specifier for a structure is public. This is an important difference to recognize and understand in object-oriented C++ programming language.

```
class demo
{
   int x;        //Here access specifier is private by default
};
```

whereas

```
struct demo
{
  int x;   //Here access specifier is public by default
};
```

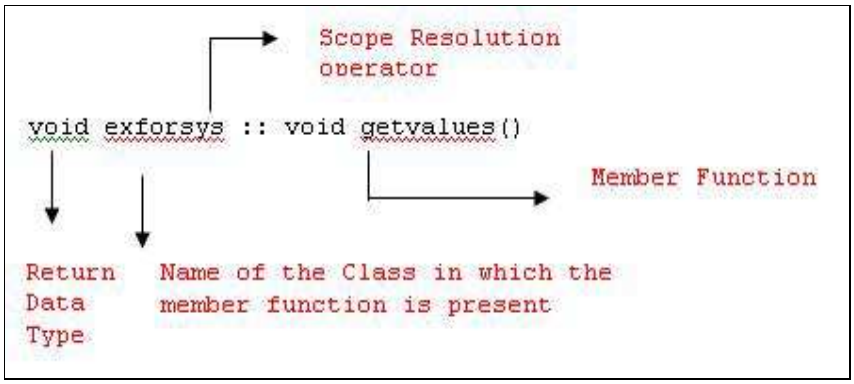
It is not always the case that the member function declaration and definition takes place within the class itself. Sometimes, declaration of member function alone can occur within the class. The programmer may choose to place the definition outside the class. In a situation such as this, it is important to understand the identifying member function of a particular class. This is performed by using the operator :: this is called ***scope resolution operator***.

```
class demo
{
  private:
  int a;
  public:
  void getvalues()            // Only Member Function declaration is
done
};

void demo :: getvalues()    // Here Member Function is defined
{
  ..................
  ..................
}

main()
{
  demo e1,e2;
  ............
}
```

So the usage of scope resolution operator is as follows:



C++ Standard Input Output Stream

In this C++ tutorial, you will learn about standard input stream and standard output stream explained along with syntax and examples. C++ programming language uses the concept of streams to perform input and output operations using the keyboard and to display information on the monitor of the computer.

What is a Stream?

A stream is an object where a program can either insert or extract characters to or from it. The standard input and output stream objects of C++ are declared in the header file ***iostream***.

Standard Input Stream

Generally, the device used for input is the keyboard. For inputting, the keyword *cin* is used, which is an object. The overloaded operator of extraction, >>, is used on the standard input stream, in this case: *cin* stream. Syntax for using the standard input

stream is cin *followed by the operator >> followed by the* variable *that stores the data extracted from the stream.*

For example:

```
int prog;
cin >> prog;
```

In the example above, the variable *prog* is declared as an integer type variable. The next statement is the *cin* statement. The *cin* statement waits for input from the user's keyboard that is then stored in the integer variable *prog*.

The input stream *cin* wait before proceeding for processing or storing the value. This duration is dependent on the user pressing the RETURN key on the keyboard. The input stream *cin* waits for the user to press the RETURN key then begins to process the command. It is also possible to request input for more than one variable in a single input stream statement. A single *cin* statement is as follows:

```
cin >> x >> y;
```

is the same as:

```
cin >> x;
cin >> y;
```

In both of the above cases, two values are input by the user, one value for the variable x and another value for the variable y.

```
// This is a sample program        This is a comment
Statement

#include <iostream.h>                //Header File Inclusion
void main()
{
        int sample, example;
        cin >> sample;
        cin >> example;
}
```

Note: If a programmer wants to write comments in C++ program, the comments should follow after a pair of slashes denoted by //. All the characters after the // are ignored by C++ compiler and the programmer can choose to comment after the //.

Standard Output Stream

By default, the device used for output is the screen of the computer. For outputting values the keyword *cout* is used, which is an object. The insertion operator << is used on the standard output *cout* stream. The syntax for using the standard output stream is *cout* followed by the operator << followed by the value to be inserted or output by the insertion operator.

For example:

```
int prog;
cin >> prog;
cout << prog;
```

In the above example, the variable *prog* is declared as an integer type variable. The next statement is the *cin* statement that waits for input from the user's keyboard. This information is then stored in the integer variable prog. The value of *prog* is displayed on the screen by the standard output stream *cout*. It is also possible to display a sentence as follows:

```
cout << " Training given by Demo";
```

If a programmer chooses to use constant strings of characters, they must be enclosed between double quotes " ".

Example to demonstrate the use of *input* and *output* streams

```
#include <iostream.h>
void main()
{
  int a,b;
  cout << "Enter the value of a:";
  cin >> a;
  b=a+10;
  cout << "Value of b is:"<<b;
}
```

C++ Operators - Part I

In this C++ tutorial, you will learn about operators, assignment operator, arithmetic operators, compound assignment operators, increment and decrement operator, the functionality of prefix and postfix operators, relational and equality operators. The operators available in C++ programming language are:

- **Assignment Operator**

This is denoted by symbol =. This operator is used for assigning a value to a variable. The left of the assignation operator is known as the lvalue (left value), which must be a variable. The right of the assignation operator is known as the rvalue (right value). The rvalue can be a constant, a variable, the result of an operation or any combination of these.

For example:

```
x = 5;
```

By following the right to left rule the value 5 is assigned to the variable x in the above assignment statement.

❖ **Arithmetic operators**

The operators used for arithmetic operation sin C++ are:

- ☑ + For addition
- ☑ - For subtraction
- ☑ * For multiplication
- ☑ / For division
- ☑ % For modulo

Compound assignment Operators

This operator is used when a programmer wants to update a current value by performing operation on the current value of the variable.

For example:

Old += new is equal to
Old = old + new

Compound assignment operators function in a similar way the other operators +=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |= function.

Increment and Decrement Operator

The increment operator is denoted by ++ and the decrement operator by --. The function of the increment operator is to increase the value and the decrement operator is to decrease the value.

These operators may be used as either **prefix** or **postfix**.

- ☑ A Prefix operator is written before the variable as ++a or –a.
- ☑ A Postfix operator is written after the variable as a++ or a--.

For Example:

```
y=3;
x=++y;          //Prefix : Here Value of x becomes 4
```

But for the postfix operator namely as below:

```
y=3                    //Postfix : Here Value of x is 3 and
Value of y is  4
x=y++;
```

Relational and Equality Operators

These operators are used for evaluating a comparison between two expressions. The value returned by the relational operation is a Boolean value (true or false value). The operators used for this purpose in C++ are:

- ☑ **== Equal to**
- ☑ **!= Not equal to**
- ☑ **> Greater than**
- ☑ **< Less than**
- ☑ **>= Greater than or equal to**
- ☑ **<= Less than or equal to**