Object Oriented Programming with C++

Before starting to learn C++ it is essential that one must have a basic knowledge of the concepts of Object oriented programming. Some of the important object oriented features are namely:

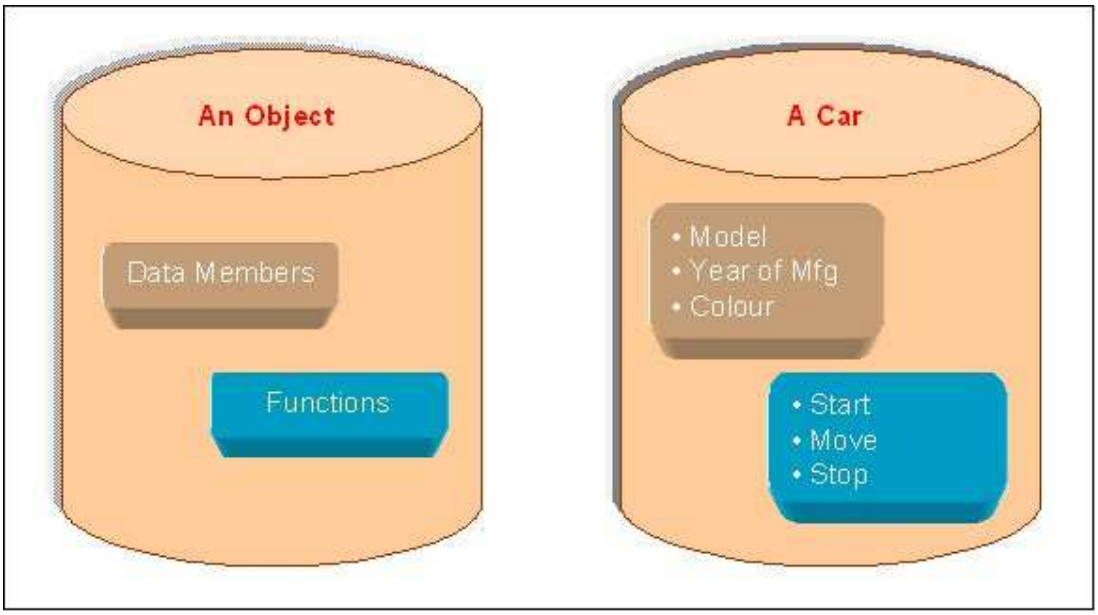
- Classes
- Objects
- Inheritance
- Data Abstraction
- Data Encapsulation
- Polymorphism
- Overloading
- Reusability

In order to understand the basic concepts in C++, the programmer must have a command of the basic terminology in object-oriented programming.

Below there is a brief outline of the concepts of Object-oriented programming languages:

Objects:

Object is the basic unit of object-oriented programming. Objects are identified by its unique name. An object represents a particular instance of a class. There can be more than one instance of an object. Each instance of an object can hold its own relevant data.



An Object is a collection of data members and associated member functions also known as methods.

Classes:

Classes are data types based on which objects are created. Objects with similar properties and methods are grouped together to form a Class. Thus a Class represent a set of individual objects. Characteristics of an object are represented in a class as **Properties**. The actions that can be performed by objects becomes functions of the class and is referred to as **Methods**.

For example consider we have a Class of *Cars* under which *Santro Xing*, *Alto* and *WaganR* represents individual Objects. In this context each *Car* Object will have its own, Model, Year of Manufacture, Colour, Top Speed, Engine Power etc., which form **Properties** of the *Car* class and the associated actions i.e., object functions like Start, Move, Stop form the **Methods** of *Car* Class.

No memory is allocated when a class is created. Memory is allocated only when an object is created, i.e., when an instance of a class is created.

Inheritance:

Inheritance is the process of forming a new class from an existing class or *base class*. The base class is also known as *parent class* or *super class*, The new class that is formed is called *derived class*. Derived class is also known as a *child class* or *sub class.* Inheritance helps in reducing the overall code size of the program, which is an important concept in object-oriented programming.

Data Abstraction:

Data Abstraction increases the power of programming language by creating user defined data types. Data Abstraction also represents the needed information in the program without presenting the details.

Data Encapsulation:

Data Encapsulation combines data and functions into a single unit called Class. When using Data Encapsulation, data is not accessed directly; it is only accessible through the functions present inside the class. Data Encapsulation enables the important concept of data hiding possible.

Polymorphism:

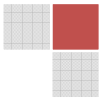
Polymorphism allows routines to use variables of different types at different times. An operator or function can be given different meanings or functions. Polymorphism refers to a single function or multi-functioning operator performing in different ways.

Overloading:

Overloading is one type of Polymorphism. It allows an object to have different meanings, depending on its context. When an exiting operator or function begins to operate on new data type, or class, it is understood to be overloaded.

Reusability:

This term refers to the ability for multiple programmers to use the same written and debugged existing class of data. This is a time saving device and adds code efficiency to the language. Additionally, the programmer can incorporate new features to the existing class, further developing the application and allowing users to achieve increased performance. This time saving feature optimizes code, helps in gaining



secured applications and facilitates easier maintenance on the application.

The implementation of each of the above object-oriented programming features for C++ will be highlighted in later sections.

A sample program to understand the basic structure of C++

```
#include<iostream.h>              //Preprocessor directive
#include<conio.h>

class employee                    //Class Declaration
{
  private:
            char empname[50];
            int empno;

  public:
        void getvalue()
        {
          cout<<"INPUT Employee Name:";
          cin>>empname;
          cout<<"INPUT Employee Number:";
          cin>>empno;
        }

        void displayvalue()
        {
          cout<<"Employee Name:"<<empname<<endl;
          cout<<"Employee Number:"<<empno<<endl;
        }
};

void main()
{
      employee e1;                //Creation of Object
      e1.getvalue();
      e1.displayvalue();
}
```

Overview of the Basic Structure of C++ Programming

- The // in first line is used for representing comment in the program.
- The second line of the program has a # symbol which represents the preprocessor directive followed by header file to be included placed between < >.
- The next structure present in the program is the class definition. This starts with the keyword class followed by class name *employee*. Within the class are data and functions. The data defined in the class are generally private and functions are public. These explanations we will be detailed in later sections. The class declaration ends with a semicolon. **main()** function is present in all C++

programs.

- An object e1 is created in employee class. Using this e1 the functions present in the employee class are accessed and there by data are accessed.

- The input namely ename and eno is got using the input statement namely cin and the values are outputted using the output statement namely cout.

Variable, Constants and Data types in C++

In this C++ tutorial, you will learn about variable, constants and data types in C++, rules for defining variable name, short int, int, long int, float, double, long double, char, bool, declaring variables and constants.

Variables

A variable is the storage location in memory that is stored by its value. A variable is identified or denoted by a variable name. The variable name is a sequence of one or more letters, digits or underscore, for example: character _

Rules for defining variable name:

- A variable name can have one or more letters or digits or underscore for example character _.

- White space, punctuation symbols or other characters are not permitted to denote variable name.

- A variable name must begin with a letter.

- Variable names cannot be keywords or any reserved words of the C++ programming language.

- C++ is a case-sensitive language. Variable names written in capital letters differ from variable names with the same name but written in small letters.

- A variable is the storage location in memory that is stored by variable value. The amount of memory allocated or occupied by each variable differs as per the data stored. The amount of memory used to store a single character is different from that of storing a single integer. A variable must be declared for the specific data type.

Data Types

Below is a list of the most commonly used *Data Types* in C++ programming language:

- short int
- int
- long int
- float
- double
- long double
- char
- bool

☑ **short int :** This data type is used to represent short integer.

☑ **int:** This data type is used to represent integer.

☑ **long int:** This data type is used to represent long integer.

☑ **float:** This data type is used to represent floating point number.

☑ **double:** This data type is used to represent double precision floating point number.

☑ **long double:** This data type is used to represent double precision floating point number.

☑ **char:** This data type is used to represent a single character.

☑ **bool:** This data type is used to represent boolean value. It can take one of two values: True or False.

Declaring Variables:

In order for a variable to be used in C++ programming language, the variable must first be declared. The syntax for declaring variable names is

data type variable name;

The date type can be int or float or any of the data types listed above. A variable name is given based on the rules for defining variable name (refer above rules).

Example:

```
int a;
```

Note: If there exists more than one variable of the same type, such variables can be represented by separating variable names using comma.

```
int x,y,z
```

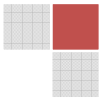
Note: The data type using integers (int, short int, long int) are further assigned a value of signed or unsigned. Signed integers signify positive and negative number value. Unsigned integers signify only positive numbers or zero.

```
unsigned short int a;
signed int z;
```

Note: By default, unspecified integers signify a signed integer.

Constants:

Constants have fixed value. Constants, like variables, contain data type. Integer constants are represented as decimal notation, octal notation, and hexadecimal notation. Decimal notation is represented with a number. Octal notation is represented with the number preceded by a zero character.



A hexadecimal number is preceded with the characters 0x.

Example

80	**represent decimal**
0115	**represent octal**
0x167	**represent hexadecimal**

Note: By default, the integer constant is represented with a number.