C++ Classes and Objects

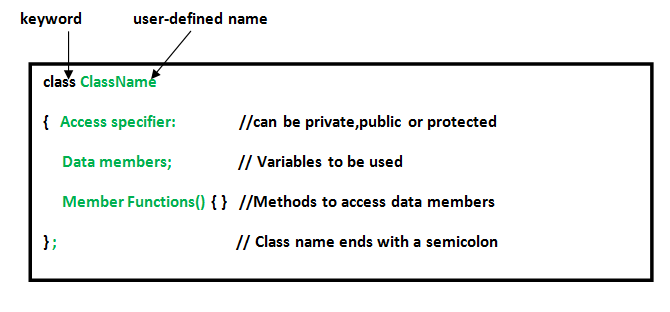
**Class:** The building block of C++ that leads to Object Oriented programming is a **Class**. It is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

For Example: Consider the Class of **Cars**. There may be many cars with different names and brand but all of them will share some common properties like all of them will have *4 wheels*, *Speed Limit*, *Mileage range* etc. So here, Car is the class and wheels, speed limits, mileage are their properties.

* A Class is a user defined data-type which has data members and member functions.
* Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.
* In the above example of class *Car*, the data member will be *speed limit*, *mileage* etc. and member functions can be *apply brakes*, *increase speed* etc.

An **Object** is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

**Defining Class and Declaring Objects**

A class is defined in C++ using keyword class followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.

**Declaring Objects:** When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.

**Syntax:**

ClassName ObjectName;

**Accessing data members and member functions**: The data members and member functions of class can be accessed using the dot(‘.’) operator with the object. For example if the name of object is *obj* and you want to access the member function with the name *printName()* then you will have to write *obj.printName()* .

**Accessing Data Members**

The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object. Accessing a data member depends solely on the access control of that data member.

This access control is given by cccess modifiers in C++. There are three access modifiers : public, private and protected.

|  |
| --- |
| // C++ program to demonstrate  // accessing of data members    #include <iostream>  using namespace std;  class Geeks  {      // Access specifier      public:        // Data Members      string geekName;        // Member Functions()      void printName()      {         cout << "Geek name is: " << geekName;      }  };    int main() {        // Declare an object of class geeks      Geeks obj1;        // accessing data member      obj1.geekName = "Henry";        // accessing member function      obj1.printname();      return 0;  } |

Output:

Geek name is: Henry

**Member Functions in Classes**

There are 2 ways to define a member function:

* Inside class definition
* Outside class definition

To define a member function outside the class definition we have to use the scope resolution :: operator along with class name and function name.

|  |
| --- |
| // C++ program to demonstrate function  // declaration outside class  **Contents of Class Definition File (geeks.h)**  #include <iostream>  #include <string>  using namespace std;  class Geeks  {      public:      string geekName;      int id;        //prototype      void printname();        //prototype  void printid()  };  **Contents of Class Implementation File (geeks.cpp)**  //Member function definitions using scope resolution operator ::  #include “geeks.h”  void Geeks::printname()  {      cout << "Geekname is: " << geekname;  }  void Geeks::printid()  {      cout << "Geek id is: " << id;  }  #include “geeks.h”  int main() {        Geeks obj1;      obj1.geekname = "xyz";      obj1.id=15;        // call printname()      obj1.printname();      cout << endl;        // call printid()      obj1.printid();      return 0;  } |
|  |

Output:

Geekname is: xyz

Geek id is: 15

[**Constructors**](https://www.geeksforgeeks.org/constructors-c/)

Constructors are special class members which are called by the compiler every time an object of that class is instantiated. Constructors have the same name as the class and may be defined inside or outside the class definition.  
There are 3 types of constructors:

* Default constructors
* Parametrized constructors
* Copy constructors

|  |
| --- |
| // C++ program to demonstrate constructors  //Class Definition File (geeks.h)  #include <iostream>  using namespace std;  class Geeks  {      public:      int id;    Geeks();  Geeks(int);  };  //Class Implementation File (geeks.cpp)  //Default Constructor  Geeks::Geeks()  {      cout << "Default Constructor called" << endl;      id=-1;  }    //Parametrized Constructor  Geeks::Geeks(int x)  {      cout << "Parametrized Constructor called" << endl;      id=x;  }  #include “geeks.h”  int main() {        // obj1 will call Default Constructor      Geeks obj1;      cout << "Geek id is: " <<obj1.id << endl;        // obj1 will call Parametrized Constructor      Geeks obj2(21);      cout << "Geek id is: " <<obj2.id << endl;      return 0;  } |
|  |

Output:

Default Constructor called

Geek id is: -1

Parametrized Constructor called

Geek id is: 21

A **Copy Constructor** creates a new object, which is exact copy of the existing copy. The compiler provides a default Copy Constructor to all the classes.  
Syntax:

class-name (class-name &){}

[**Destructors**](https://www.geeksforgeeks.org/destructors-c/)

Destructor is another special member function that is called by the compiler when the scope of the object ends.

|  |
| --- |
| // C++ program to explain destructors  //Class definition file (geeks.h)  #include <iostream>  using namespace std;  class Geeks  {      public:      int id;        //Definition for Destructor      ~Geeks()  };  //Class implementation file (geeks.cpp)  Geeks::~Geeks()  {      cout << "Destructor called for id: " << id <<endl;  }  int main()    {      Geeks obj1;      obj1.id=7;      int i = 0;      while ( i < 5 )      {          Geeks obj2;          obj2.id=i;          i++;      } // Scope for obj2 ends here        return 0;    } // Scope for obj1 ends here |

Output:

Destructor called for id: 0

Destructor called for id: 1

Destructor called for id: 2

Destructor called for id: 3

Destructor called for id: 4

Destructor called for id: 7