C++ Classes and Objects

Difficulty Level : Easy • Last Updated : 16 Feb, 2023

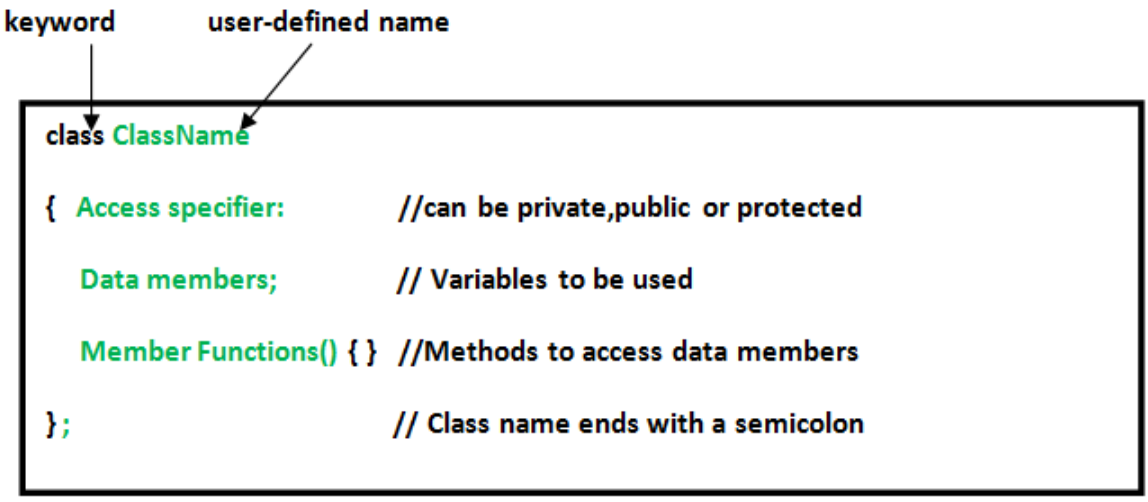
Class: A class in C++ is the building block that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object. For Example: 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 Access modifiers in C++. There are three access modifiers : **public, private and protected**.

CPP

```
// C++ program to demonstrate accessing of data members
#include <bits/stdc++.h>
using namespace std;
```

```
class Geeks {
    // Access specifier
public:
    // Data  Members
    string geekname;
    // Member Functions()
    void printname() { cout << "Geekname is:" << geekname; }
};
int main()
{
    // Declare an object of class geeks
    Geeks obj1;
    // accessing data member
    obj1.geekname = "Abhi";
    // accessing member function
    obj1.printname();
    return 0;
}
```

Output

```
Geekname is:Abhi
```

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.

CPP

```
// C++ program to demonstrate function
// declaration outside class

#include <bits/stdc++.h>
using namespace std;
class Geeks
{
    public:
    string geekname;
    int id;

    // printname is not defined inside class definition
    void printname();

    // printid is defined inside class definition
    void printid()
    {
```

```
        cout <<"Geek id is: "<<id;
    }
};

// Definition of printname using scope resolution operator ::
void Geeks::printname()
{
    cout <<"Geekname is: "<<geekname;
}
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
```

Note that all the member functions defined inside the class definition are by default **inline**, but you can also make any non-class function inline by using keyword inline with them. Inline functions are actual functions, which are copied everywhere during compilation, like pre-processor macro, so the overhead of function calling is reduced. Note: Declaring a friend function is a way to give private access to a non-member function.

Constructors

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
- Parameterized constructors
- Copy constructors

CPP

```cpp
// C++ program to demonstrate constructors

#include <bits/stdc++.h>
using namespace std;
class Geeks
{
    public:
    int id;

    //Default Constructor
    Geeks()
    {
        cout << "Default Constructor called" << endl;
        id=-1;
    }

    //Parameterized Constructor
    Geeks(int x)
    {
        cout <<"Parameterized Constructor called "<< endl;
        id=x;
    }
};
int main() {

    // obj1 will call Default Constructor
    Geeks obj1;
    cout <<"Geek id is: "<<obj1.id << endl;

    // obj2 will call Parameterized Constructor
    Geeks obj2(21);
    cout <<"Geek id is: " <<obj2.id << endl;
    return 0;
}
```

Output:

```
Default Constructor called
Geek id is: -1
Parameterized Constructor called
Geek id is: 21
```

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

```
class-name (class-name &){}
```

Destructors

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

CPP

```
// C++ program to explain destructors

#include <bits/stdc++.h>
using namespace std;
class Geeks
{
    public:
    int id;

    //Definition for Destructor
    ~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
```

Interesting Fact (Rare Known concept)

Why do we give semicolon at the end of class ?

Many people might say that its a basic syntax and we should give semicolon at end of class as its rule define in cpp . But the main reason why semi-colons is there at end of class is compiler checks if user is trying to create an instance of class at the end of it .

Yes just like structure , Union we can also create the instance of class at the end just before the semicolon. As a result once execution reaches at that line it create class and allocates memory to your instance

---

C++

```
#include <iostream>
using namespace std;

class Demo{
   int a, b;
    public:
    Demo()   // default constructor
    {
        cout << "Default Constructor" << endl;
    }
    Demo(int a, int b):a(a),b(b)  //parameterised constructor
    {
        cout << "parameterized constructor -values" << a  << " "<< b << endl;
    }

}instance;


int main() {

    return 0;
}
```

Output

```
 Default Constructor
```

We can see that we have created class instance of Demo with name "instance" , as a result output we can see is Default Constructor is called.

Similarly we can also call the parameterized constructor just by passing values here

---

C++

```
#include <iostream>
using namespace std;

class Demo{
    public:
```

```
    int a, b;
    Demo()
    {
        cout << "Default Constructor" << endl;
    }
    Demo(int a, int b):a(a),b(b)
    {
        cout << "parameterized Constructor values-" << a << " "<< b << endl;
    }


}instance(100,200);


int main() {

    return 0;
}
```

Output

```
parameterized Constructor values-100 200
```

So by creating instance just before the semicolon , we can create the Instance of class

Pure Virtual Destructor **Related Articles:**

- Multiple Inheritance in C++
- C++ Quiz

This article is contributed by **Abhirav Kariya**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

771

Related Articles

1. Classes and Objects in Java
2. Catching Base and Derived Classes as Exceptions in C++ and Java
3. Pure Virtual Functions and Abstract Classes in C++