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](https://www.geeksforgeeks.org/constructors-c/)
* Parameterized constructors
* [Copy constructors](https://www.geeksforgeeks.org/copy-constructor-in-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;

}

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 &){}

## **Characteristics of Copy Constructor**

**1.** The copy constructor is used to initialize the members of a newly created object by copying the members of an already existing object.

**2.** Copy constructor takes a reference to an object of the same class as an argument.

Sample(Sample &t)

{

id=t.id;

}

**3.** The process of initializing members of an object through a copy constructor is known as ***copy initialization.***

**4**. It is also called member-wise initialization because the copy constructor initializes one object with the existing object, both belonging to the same class on a member-by-member copy basis.

**5.** The copy constructor can be defined explicitly by the programmer. If the programmer does not define the copy constructor, the compiler does it for us.

**For Example:**

#include <iostream>

using namespace std;

class Point {

private:

int x, y;

public:

Point(int x1, int y1)

{

x = x1;

y = y1;

}

// Copy constructor

Point(const Point& p1)

{

x = p1.x;

y = p1.y;

}

int getX() { return x; }

int getY() { return y; }

};

int main()

{

Point p1(10, 15); // Normal constructor is called here

Point p2 = p1; // Copy constructor is called here

// Let us access values assigned by constructors

cout << "p1.x = " << p1.getX()

<< ", p1.y = " << p1.getY();

cout << "\np2.x = " << p2.getX()

<< ", p2.y = " << p2.getY();

return 0;

}

[**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    #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 |
| --- |

class MyClass { // The class

public: // Access specifier

MyClass() { // Constructor

cout << "Hello World!";

}

};

int main() {

MyClass myObj; // Create an object of MyClass (this will call the constructor)

return 0;

}

## Constructor Parameters

class Car { // The class

public: // Access specifier

string brand; // Attribute

string model; // Attribute

int year; // Attribute

Car(string x, string y, int z) { // Constructor with parameters

brand = x;

model = y;

year = z;

}

};

int main() {

// Create Car objects and call the constructor with different values

Car carObj1("BMW", "X5", 1999);

Car carObj2("Ford", "Mustang", 1969);

// Print values

cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year << "\n";

cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year << "\n";

return 0;

}