C++ Constructor

In C++, constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new object generally. The constructor in C++ has the same name as class or structure.

There can be two types of constructors in C++.

* Default constructor
* Parameterized constructor

C++ Default Constructor

A constructor which has no argument is known as default constructor. It is invoked at the time of creating object.

Let's see the simple example of C++ default Constructor.

1. #include <iostream>
2. **using** **namespace** std;
3. **class** Employee
4. {
5. **public**:
6. Employee()
7. {
8. cout<<"Default Constructor Invoked"<<endl;
9. }
10. };
11. **int** main(**void**)
12. {
13. Employee e1; //creating an object of Employee
14. Employee e2;
15. **return** 0;
16. }

Output:

Default Constructor Invoked

Default Constructor Invoked

C++ Parameterized Constructor

A constructor which has parameters is called parameterized constructor. It is used to provide different values to distinct objects.

Let's see the simple example of C++ Parameterized Constructor.

#include <iostream>

using namespace std;

class Employee {

public:

int id;//data member (also instance variable)

string name;//data member(also instance variable)

float salary;

Employee(int i, string n, float s)

{

id = i;

name = n;

salary = s;

}

void display()

{

cout<<id<<" "<<name<<" "<<salary<<endl;

}

};

int main(void) {

Employee e1 =Employee(101, "Sonoo", 890000); //creating an object of Employee

Employee e2=Employee(102, "Nakul", 59000);

e1.display();

e2.display();

return 0;

}

Output:

101 Sonoo 890000

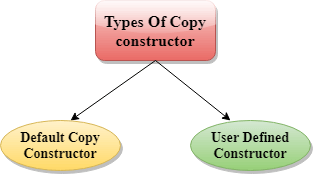
102 Nakul 59000

C++ Copy Constructor

A Copy constructor is an **overloaded** constructor used to declare and initialize an object from another object.

Copy Constructor is of two types:

* **Default Copy constructor:** The compiler defines the default copy constructor. If the user defines no copy constructor, compiler supplies its constructor.
* **User Defined constructor:** The programmer defines the user-defined constructor.



Syntax Of User-defined Copy Constructor:

1. Class\_name(**const** class\_name &old\_object);

Consider the following situation:

1. **class** A
2. {
3. A(A &x) //  copy constructor.
4. {
5. // copyconstructor.
6. }
7. }

In the above case, **copy constructor can be called in the following ways:**

C++ Copy Constructor

Let's see a simple example of the copy constructor.

Exception Handling in Java - Javatpoint

**// program of the copy constructor.**

1. #include <iostream>
2. **using** **namespace** std;
3. **class** A
4. {
5. **public**:
6. **int** x;
7. A(**int** a)                // parameterized constructor.
8. {
9. x=a;
10. }
11. A(A &i)               // copy constructor
12. {
13. x = i.x;
14. }
15. };
16. **int** main()
17. {
18. A a1(20);               // Calling the parameterized constructor.
19. A a2(a1);                //  Calling the copy constructor.
20. cout<<a2.x;
21. **return** 0;
22. }

**Output:**

20

When Copy Constructor is called

Copy Constructor is called in the following scenarios:

* When we initialize the object with another existing object of the same class type. For example, Student s1 = s2, where Student is the class.
* When the object of the same class type is passed by value as an argument.
* When the function returns the object of the same class type by value.

# C++ Destructor

A destructor works opposite to constructor; it destructs the objects of classes. It can be defined only once in a class. Like constructors, it is invoked automatically.

A destructor is defined like constructor. It must have same name as class. But it is prefixed with a tilde sign (~).

#### Note: C++ destructor cannot have parameters. Moreover, modifiers can't be applied on destructors.

## C++ Constructor and Destructor Example

Let's see an example of constructor and destructor in C++ which is called automatically.

#include <iostream>

**using** **namespace** std;

**class** Employee

 {

**public**:

        Employee()

        {

            cout<<"Constructor Invoked"<<endl;

        }

        ~Employee()

        {

            cout<<"Destructor Invoked"<<endl;

        }

};

**int** main(**void**)

{

    Employee e1; //creating an object of Employee

    Employee e2; //creating an object of Employee

**return** 0;

}

Output:

HTML Tutorial

Constructor Invoked

Constructor Invoked

Destructor Invoked

Destructor Invoked

**Dynamic Constructor in C++ with Examples**

When allocation of memory is done dynamically using dynamic memory allocator new in a constructor, it is known as **dynamic constructor**. By using this, we can dynamically initialize the objects.

|  |
| --- |
| #include <iostream>  using namespace std;    class geeks {      const char\* p;    public:      // default constructor      geeks()      {            // allocating memory at run time          p = new char[6];          p = "geeks";      }        void display()      {          cout << p << endl;      }  };    int main()  {      geeks obj = new geeks();      obj.display();  } |

**Output:**

geeks

**Explanation**: In this we point data member of type **char** which is allocated memory dynamically by **new** operator and when we create dynamic memory within the constructor of class this is known as dynamic constructor.

**Example 2:**

|  |
| --- |
| #include <iostream>  using namespace std;    class geeks {      int\* p;    public:      // default constructor      geeks()      {            // allocating memory at run time          // and initializing          p = new int[3]{ 1, 2, 3 };            for (int i = 0; i < 3; i++) {              cout << p[i] << " ";          }            cout << endl;      }  };    int main()  {        // five objects will be created      // for each object      // default constructor would be called      // and memory will be allocated      // to array dynamically      geeks\* ptr = new geeks[5];  } |

**Output:**

1 2 3

1 2 3

1 2 3

1 2 3

1 2 3

dynamically .

**Explanation**: In this program we have created array of object dynamically. The first object is ptr[0], second is ptr[1] and so on . For each object creation default constructor is called and for each object memory is allocated to pointer type variable by new operator.

**Example 3:**

|  |
| --- |
| #include <iostream>  using namespace std;    class geeks {      int\* p;    public:      // default constructor      geeks()      {            // allocating memory at run time          p = new int;          \*p = 0;      }        // parameterized constructor      geeks(int x)      {          p = new int;          \*p = x;      }      void display()      {          cout << \*p << endl;      }  };    int main()  {        // default constructor would be called      geeks obj1 = new geeks();      obj1.display();        // parameterized constructor would be called      geeks obj2 = new geeks(7);      obj2.display();  } |

**Output:**

0

7

**Explanation:** In this integer type pointer variable is declared in class which is assigned memory dynamically when the constructor is called. When we create object **obj1**, the default constructor is called and memory is assigned dynamically to pointer type variable and initialized with value 0. And similarly when **obj2** is created parameterized constructor is called and memory is assigned dynamically.

**Constructor Overloading in C++**

**Prerequisites:**Constructors in C++   
In C++, We can have more than one constructor in a class with same name, as long as each has a different list of arguments.This concept is known as Constructor Overloading and is quite similar to function overloading. 

* Overloaded constructors essentially have the same name (exact name of the class) and differ by number and type of arguments.
* A constructor is called depending upon the number and type of arguments passed.
* While creating the object, arguments must be passed to let compiler know, which constructor needs to be called.

|  |
| --- |
| // C++ program to illustrate  // Constructor overloading  #include <iostream>  using namespace std;    class construct  {    public:      float area;        // Constructor with no parameters      construct()      {          area = 0;      }        // Constructor with two parameters      construct(int a, int b)      {          area = a \* b;      }        void disp()      {          cout<< area<< endl;      }  };    int main()  {      // Constructor Overloading      // with two different constructors      // of class name      construct o;      construct o2( 10, 20);        o.disp();      o2.disp();      return 1;  } |

Output:

0

200