Constructors in C++

Constructor in C++ is a special method that is invoked automatically at the time of object creation. It is used to initialize the data members of new objects generally. The constructor in C++ has the same name as the class or structure. Constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object which is why it is known as constructors.

- Constructor is a member function of a class, whose name is same as the class name.
- Constructor is a special type of member function that is used to initialize the data members for an object of a class automatically, when an object of the same class is created.
- Constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.
- Constructor do not return value, hence they do not have a return type

The prototype of the constructor looks like

```
<class-name> (list-of-parameters);
```

Constructor can be defined inside the class declaration or outside the class declaration

a. Syntax for defining the constructor within the class

```
<class-name>(list-of-parameters)
{
        //constructor definition
}
```

b. Syntax for defining the constructor outside the class

```
<class-name>: :<class-name>(list-of-parameters)
```

```
{
        //constructor definition
}
```

.

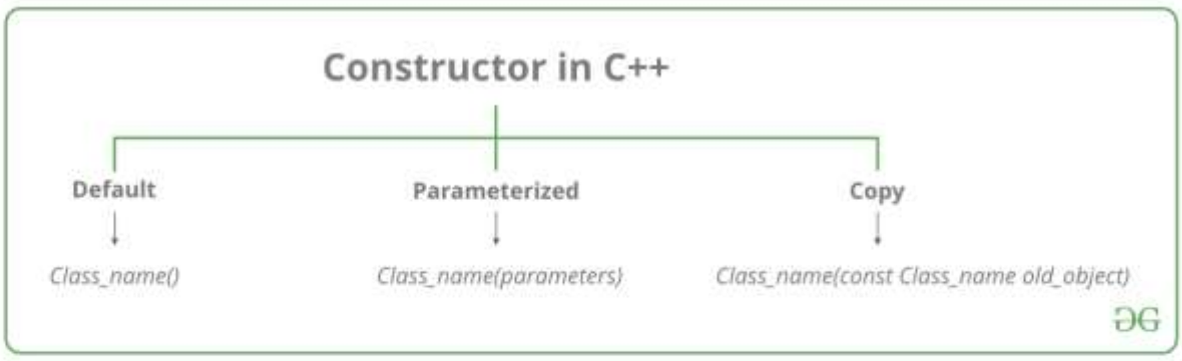
```
// Example: defining the constructor within the class
#include<iostream>
using namespace std;
class student
{
   int rno;
   string name;
   double fee;
   public:
   student()
   {
      cout<<"Enter the RollNo:";
      cin>>rno;
      cout<<"Enter the Name:";
      cin>>name;
      cout<<"Enter the Fee:";
      cin>>fee;
   }
   void display()
   {
      cout<<endl<<rno<<"\t"<<name<<"\t"<<fee;
   }
```

```
          };
          int main()
          {
             student s;  //constructor gets called automatically when we create the
object of the class
             s.display();
             return 0;
          }
```

```
               // Example: defining the constructor outside the class


#include<iostream>
using namespace std;
class student
{
int rno;
string name;
double fee;
public:
student();
void display();
};
student::student()
{
                  cout<<"Enter the RollNo:";
               cin>>rno;
```

```
		cout<<"Enter the Name:";
		cin>>name;
		cout<<"Enter the Fee:";
		cin>>fee;
}
void student::display()
{
		cout<<endl<<rno<<"\t"<<name<<"\t"<<fee;
}
int main()
{
student s;
s.display();
return 0;}
```



Characteristics of the constructor:

- The name of the constructor is the same as its class name.
- Constructors are mostly declared in the public section of the class though it can be declared in the private section of the class.
- Constructors do not return values; hence they do not have a return type.
- A constructor gets called automatically when we create the object of the class.
- Constructors can be overloaded.
- Constructor can not be declared virtual.
- Constructor cannot be inherited.
- Addresses of Constructor cannot be referred.
- Constructor make implicit calls to new and delete operators during memory allocation.

Types of Constructors

1. Default Constructors: Default constructor is the constructor which doesn't take any argument. It has no parameters. It is also called a zero-argument constructor.

```
#include <iostream>

using namespace std;

class construct

 {

public:

int a, b;
construct()
{
a = 10;
b = 20;
                }
};
int main()
{
                construct c;
                cout << "a: " << c.a << endl << "b: " << c.b;
                return 1;
}
```

2. Parameterized Constructors: It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor's body, use the parameters to initialize the object.

Note: when the parameterized constructor is defined and no default constructor is defined explicitly, the compiler will not implicitly call the default constructor and hence creating a simple object a

```
#include <iostream>
using namespace std;
class Employee {
  public:
      int id;
      string name;
      float salary;
      Employee(int i, string n, float s)
      {
          id = i;
          name = n;
          salary = s;
      }
     void display()
      {
          cout<<id<<"  "<<name<<"  "<<salary<<endl;
      }
};
int main()
{
   Employee e1 =Employee(101, "Sonoo", 890000);
   Employee e2=Employee(102, "Nakul", 59000);
   e1.display();
   e2.display();
   return 0;
}
```

3. Copy Constructor:

A copy constructor is a member function that initializes an object using another object of the same class.

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

```
        Sample(Sample &t)
                {
                        id=t.id;
                }
```

```
#include<iostream>
using namespace std
class wall
{
        private:
                double length;
                double height;
        public:
        wall(double len,double hgt)
                {
                        length=len;
                        height=hgt;
                }
        wall(wall & obj)
        {
                length=obj.length;
                height=obj.height;
        }
        double calculateArea()
        {
                return length*height;
        }
};
int main()
{
        wall wall1(10.5,8.6);

        wall wall2(wall1);

        cout<<"Area of wall 1:"<<wall1.calculateArea()<<endl;

        cout<<"area of wall 2:"<<wall2.calculateArea()<<endl;

        return 0;

}
```

Destructor:

A destructor is also a special member function as a constructor. Destructor destroys the class objects created by the constructor. Destructor has the same name as their class name preceded by a tilde (~) symbol. It is not possible to define more than one destructor. The destructor is only one way to destroy the object created by the constructor. Hence destructor can-not be overloaded. Destructor neither requires any argument nor returns any value. It is automatically called when the object goes out of scope. Destructors release memory space occupied by the objects created by the constructor. In destructor, objects are destroyed in the reverse of object creation.

The syntax for defining the destructor within the class

```
        ~ <class-name>()
        {
         }
```

The syntax for defining the destructor outside the class

```
<class-name>: : ~ <class-name>(){}
```

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

class Test {
public:
         Test()
          {
          cout << "\n Constructor executed";
          }
         ~Test()
          {
```

```
          cout << "\n Destructor executed";

           }

};

main()

{

         Test t;

         return 0;

}
```

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;
    Employee e2;
    return 0;
}
```

```
        #include <iostream>
using namespace std;
class Test {
public:
        Test() { cout << "\n Constructor executed"; }

        ~Test() { cout << "\n Destructor executed"; }
};

main()
{
        Test t, t1, t2, t3;
        return 0;
}
```