Lab-2 Constructors

What is constructor?

A *constructor* is a member function with the same name as its class. A constructor is a special type of member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object (instance of class) create. It is special member function of the class because it does not have any return type.

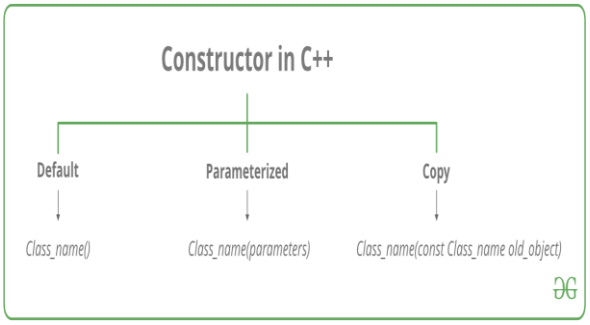
Why constructor is used in C++?

A constructor is a special "MEMBER FUNCTION" in C++ that has the same name as the class it belongs to and is used to initialize some useful values for an object's data members. The constructor is used to INITIALIZE VALUES and is automatically called by the compiler, which is why.

How constructors are different from a normal member function?

A constructor is different from normal functions in following ways:

- Constructor has same name as the class itself
- Constructors don't have return type
- A constructor is automatically called when an object is created.
- It must be placed in public section of class.
- If we do not specify a constructor, C++ compiler generates a default constructor for object (expects no parameters and has an empty body).



Let us understand the types of constructors in C++ by taking a real-world example. Suppose you went to a shop to buy a marker. When you want to buy a marker, what are the options? The first one you go to a shop and say give me a marker. So just saying give me a marker mean that you did not set which brand name and which color, you didn't mention anything just say you want a marker. So when we said just I want a marker so whatever the frequently sold marker is there in the market or in his shop he will simply hand over that. And this is whata default constructor is! The second method you go to a shop and say I want a marker a red in color and XYZ brand. So you are mentioning this and he will give you that marker. So in this case you have given the parameters. And this is what a parameterized constructor is! Then the third one you go to a shop and say I want a marker like this (a physical marker on your hand). So the shopkeeper will see that marker. Okay, and he will give a new marker for you. So copy of that marker. And that's what copy constructor is!

Types of Constructors

1. **Default Constructors:** Default constructor is the constructor which doesn'ttake any argument. It has no parameters.

```
// Cpp program to illustrate the concept of Constructors
#include <iostream>
using namespace std;
class construct {
 public:
    int a, b;
 // Default Constructor
    construct()
    {
                a = 10;
                b = 20;
    }
};
int main()
{
    // Default constructor called automatically
    // when the object is created
    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 as

Student s; Will flash an error

```
// CPP program to illustrate parameterized constructors

#include <iostream>

using namespace std;

class Point {

private:

int x, y;

public:

// Parameterized Constructor

Point(int x1, int y1)
```

```
{
x = x1;
y = y1;
}
int getX() { return x; }
int getY() { return y; }};
int main()
{
// Constructor called
Point p1(10, 15);
// Access values assigned by constructor
cout << "p1.x = " << p1.getX()
<< ", p1.y = " << p1.getY();
return 0; }
```

When an object is declared in a parameterized constructor, the initial values have to be passed as arguments to the constructor function. The normal way of object declaration may not work. The constructors can be called explicitly or implicitly.

```
Example e = Example(0, 50); // Explicit call
Example e(0, 50);           // Implicit call
```

Uses of Parameterized constructor:

- It is used to initialize the various data elements of different objects with different values when they are created.
- It is used to overload constructors.

3. Copy Constructor:

A copy constructor is a member function that initializes an object using another object of the same class. Whenever we define one or more non-default constructors (with parameters) for a class, a default constructor (without parameters) should also be explicitly defined as the compiler will not provide a default constructor in this case. However, it is not necessary but it's considered to be the best practice to always define a default constructor. Copy constructor takes a reference to an object of the same class as an argument.

// Implicit copy constructor

```
#include<iostream>
using namespace std;
class Sample
{
        int id;
        public:
        void init(int x)
        {
                id=x;
        }
        void display()
        {
                cout<<endl<<"ID="<<id;
        }
};

int main()
{
        Sample obj1;
        obj1.init(10);
        obj1.display();
        Sample obj2(obj1); //or obj2=obj1;
        obj2.display();
        return 0;
}
```

// Example: Explicit copy constructor

```
#include <iostream>
using namespace std;
class Sample
{
        int id;
        public:
        void init(int x)
        {
                id=x;
        }
        Sample(){} //default constructor with empty body
        Sample(Sample &t) //copy constructor
        {
                id=t.id;
        }
        void display()
        {
                cout<<endl<<"ID="<<id;
        }
};
int main()
{
        Sample obj1;
        obj1.init(10);
        obj1.display();
        Sample obj2(obj1); //or obj2=obj1; copy constructor called
        obj2.display();
        return 0;
}
```

- **Can we have more than one constructor in a class?**
  Yes, It is called Constructor Overloading.

Constructor Overloading

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 different 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;
}
```

// Defining the constructor within the class

```
#include <iostream>
using namespace std;
class student {
        int rno;
        char name[10];
        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;
}
```

// defining the constructor outside the class

```
#include <iostream>
using namespace std;
class student {
        int rno;
        char name[50];
        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;
}
```