Lecture 04

Constructors in C++:

- Constructor in C++ is a special method that is invoked automatically at the time an object of a class is created.
- 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.
- It constructs the values i.e. provides data for the object which is why it is known as a constructor.

Constructors in C++:

Syntax of Constructors in C++:

<class-name> (){

...

}

Constructors in C++:

Characteristics of Constructors in C++

- The name of the constructor is the same as its class name.
- Constructors are mostly declared in the public section of the class though they 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 in C++:

Types of Constructor Definitions in C++

In C++, there are 2 methods by which a constructor can be declared:

1. Defining the Constructor Within the Class:

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

Constructors in C++:

```
#include <iostream>
using namespace std;
class student { int rno;char name[50];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;
  s.display();
    return 0;
}
```

Constructors in C++:

Types of Constructor Definitions in C++

In C++, there are 2 methods by which a constructor can be declared:

2. Defining the Constructor Outside the Class

```
<class-name> {
      // Declaring the constructor
    <class-name>();
// Defining remaining class
}
<class-name>: :<class-name>(list-of-parameters) {
 // constructor definition
}
```

Constructors in C++:

```
#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;
}
```

Constructors in C++:

Types of Constructors in C++:

Constructors can be classified based on in which situations they are being used. There are 4 types of constructors in C++:

Default Constructor: No parameters. They are used to create an object with default values.

Parameterized Constructor: Takes parameters. Used to create an object with specific initial values.

Copy Constructor: Takes a reference to another object of the same class. Used to create a copy of an object.

Move Constructor: Takes an rvalue reference to another object. Transfers resources from a temporary object.

Constructors in C++:

1.Default Constructor:

A default constructor is a constructor that doesn't take any argument. It has no parameters. It is also called a zero-argument constructor.

Syntax of Default Constructor:

```
className() {
    // body_of_constructor
}
```

The compiler automatically creates an implicit default constructor if the programmer does not define one.

Constructors in C++:

2. Parameterized Constructor

Parameterized constructors make it 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.

Syntax of Parameterized Constructor

```
className (parameters...) {
    // body
}
```

Constructors in C++:

3. Copy Constructor

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

Syntax of Copy Constructor

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

```
ClassName (ClassName &obj)
{
  // body_containing_logic
}
```

Constructors in C++:

4. Move Constructor

The move constructor is a recent addition to the family of constructors in C++. It is like a copy constructor that constructs the object from the already existing objects., but instead of copying the object in the new memory, it makes use of move semantics to transfer the ownership of the already created object to the new object without creating extra copies.

It can be seen as stealing the resources from other objects.

Syntax of Move Constructor

```
className (className&& obj) {
    // body of the constructor
}
```

Destructors in C++:

- Destructor is an instance member function that is invoked automatically whenever an object is going to be destroyed.
- Meaning, a destructor is the last function that is going to be called before an object is destroyed.

Syntax to Define Destructor:

- The syntax for defining the destructor within the class:

```
~ <class-name>() {
    // some instructions
}
```

Destructors in C++:

- Just like any other member function of the class, we can define the destructor outside the class too:

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

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

Destructors in C++:

- Characteristics of a Destructor
- A destructor is also a special member function like 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 cannot be overloaded.

Destructors in C++:

- Characteristics of a Destructor
- It cannot be declared static or const.
- Destructor neither requires any argument nor returns any value.
- It is automatically called when an object goes out of scope.
- Destructor release memory space occupied by the objects created by the constructor.
- In destructor, objects are destroyed in the reverse of an object creation.

Destructors in C++:

```
#include <iostream>
using namespace std;
class Test {
public:
    Test() { cout << "\n Constructor executed"; }
    ~Test() { cout << "\nDestructor executed"; }
};
main()
{
    Test t;
    return 0;
}
```