19CSE201 :Advanced Programming

Lecture 7
Constructors and Destructors

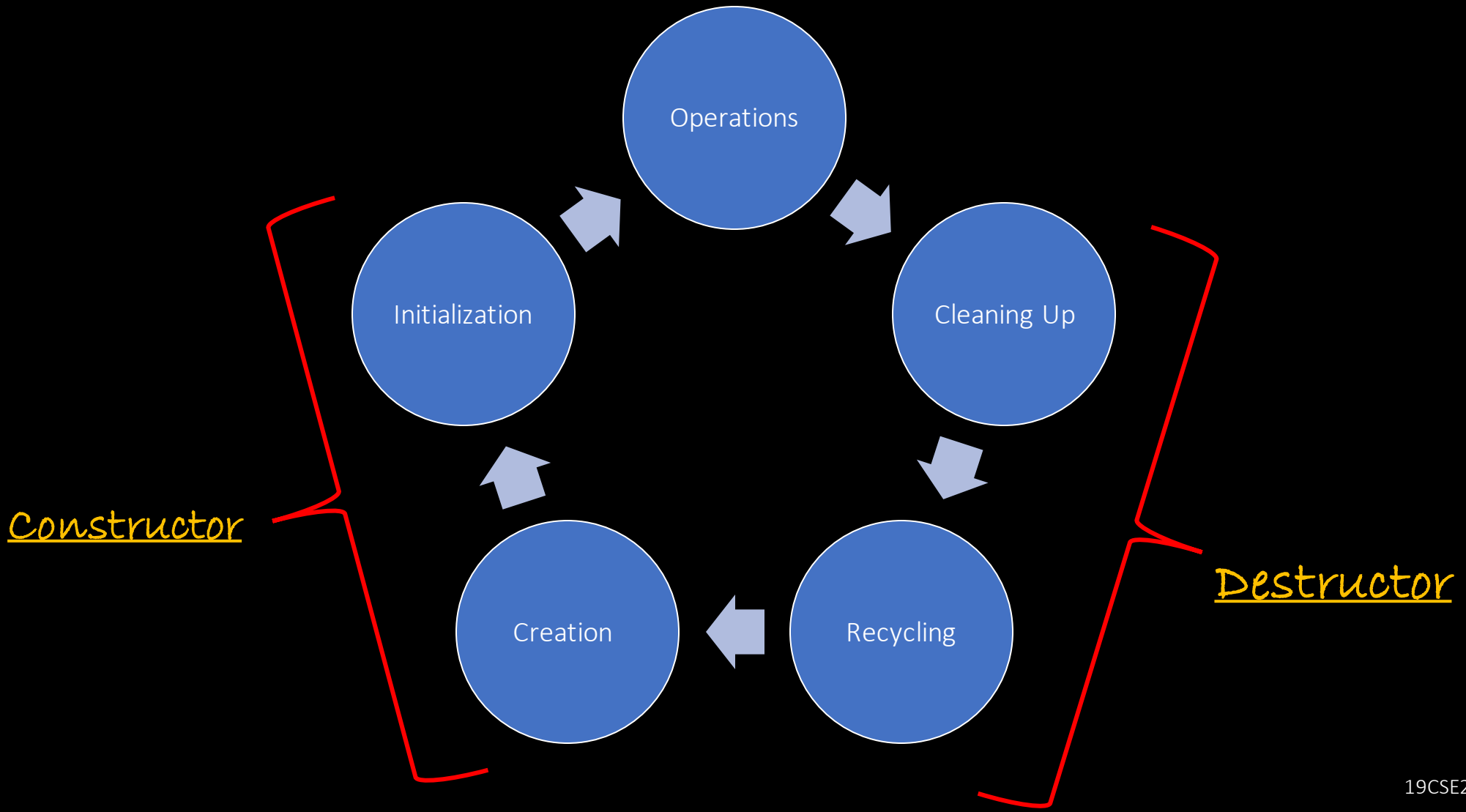
By
Ritwik M
Assistant Professor(SrGr)
Dept. Of Computer Science & Engg

Based on material from The Professor's C++Course by Linda W. Friedman and Stroustrup B. Programming: principles and practice using C++. Second edition,Addison Wesley; 2014.

A Quick Recap

- Objects
- Types of Objects
- Array of Objects
- Passing Objects as function parameters
- Returning objects from functions

Lifecycle of an Object



Constructors

- Constructor:
  - A constructor is a special member function that creates an object and initializes the data member.
  - Called automatically when the object is created – Need not be explicitly specified
  - Characteristics
    - It does not have any return value (not even void)
    - The name of the constructor is the same as the name of the class

Types of Constructors – Default Constructor

- Takes no arguments
- Implicitly (automatically) created and called by the compiler

Example: Default Constructor

```
using namespace std;
class construct {
public:
    int a, b;
    // Default Constructor
definition
    construct( )
    {
        a = 10;
        b = 20;
    }
};
```

```
int main()
{
  // Default constructor called
automatically
  //when the object is created
    construct c;
    cout << "a: " << c.a
    cout << "b: " << c.b;
    return 0;
}
```

Output:

```
a: 10
b: 20
```

Types of Constructors: Parameterized Constructor

- Parameterized Constructor:
  - It is possible to pass arguments to constructors.
  - These arguments help initialize an object when it is created.
  - To create a parameterized constructor, simply add parameters to it (the same way as to any other function).
  - When defining the constructor's body, use the parameters to initialize the object.
  - Can be called implicitly or explictly
    - `Point p1 = Point(10,20) // Explicit call`
    - `Point p1(10,20) // Implicit call`
  - Uses:
    - To initialize various data elements of different objects with diffferent values when they are created
    - Used to overload constructors – Discussed Later!!

Example: Parameterized Constructor

```
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;
    }
};   // end of class definition
```

```
int main(){
    // Constructor called
    Point p1(10, 15);


  // Access values assigned by constructor
cout << "p1.x = " << p1.getX() << ", p1.y = " <<
  p1.getY();
return 0;
}
```

Types of Constructors: Copy Constructor

- A Member function which initializes an object using another object of the same class
- Prototype:
  - `className (const className &oldObject);`
- After calling the copy constructor, the source and the destination objects have the same value for each data member, although they are different objects.
- The copy constructor has only one parameter that receives the source object by reference.
- The const modifier in front of the parameter type guarantees that the pass-by-reference cannot change the source object.
- Called when:
  - When an object of the class is passed (to a function) by value as an argument.
  - When an object is constructed based on another object of the same class

Example : Copy Constructor

```
class Point{
private:
    int x, y;
public:
    Point(int x1, int y1){
    x = x1;
    y = y1;
    }
    // Copy constructor
    Point(const Point &b){
        x = b.x;
        y = b.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
    cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
    cout << "\np2.x = " << p2.getX() << ", p2.y = " << p2.getY();
    return 0;
}
```

Destructors

- Destructor:
  - A destructor is a special member function that cleans and destroys an object.
  - Guaranteed to be called automatically and is executed by the system when an object instantiated from the class goes out of scope.
  - Characteristics:
    - The name of the destructor is the name of the class preceded by the tilde (~) symbol
    - It does not have any return value (not even void)

Destructors Cont.

- Rules
  - Name should begin with tilde sign(~) and must match class name.
  - There cannot be more than one destructor in a class.
  - Unlike constructors that can have parameters, destructors do not allow any parameter.
  - They do not have any return type, just like constructors.
  - When you do not specify any destructor in a class, compiler generates a default destructor and inserts it into your code.
- Called
  - When the program finished execution
  - When a scope (the { } parenthesis) containing local variable ends.
  - When the delete operator is called.

Example

```
class HelloWorld
{
public:
//default Constructor
HelloWorld()
{
cout<<"Constructor is called"<<endl;
}
//Destructor
~HelloWorld()
{
cout<<"Destructor is called"<<endl;
}
//Member function
void display()
{
cout<<"Hello World!"<<endl;
 }
};
```

```
int main()
{
//Object created
HelloWorld obj;

//Member function
called obj.display();
return 0;
 }
```

Quick Summary

- Constructor and Destructor
- Types of Constructor
- Destructors
- Examples

Up Next

OOP Concepts in C++