جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

قسم الامن السيبراني

Department of Cyber Security

Subject:

Object Oriented Programming (OOP)

Class:

Second

Lecturer:

Dr. Abdulkadhem A. Abdulkadhem

Lecture: (5)

Constructors and Destructors

1. Introduction to Constructors

A **constructor** is a special member function that is automatically called when an object is created. Its primary purpose is to initialize objects of a class. The constructor has the same name as the class and does not have a return type.

Types of Constructors:

1. **Default Constructor:** A constructor that takes no arguments.
2. **Parameterized Constructor:** A constructor that takes one or more arguments to initialize an object.
3. **Copy Constructor:** A constructor that creates a new object as a copy of an existing object.

Example of a Default Constructor:

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

class Student {
public:
    string name;
    int age;

    // Default Constructor
    Student() {
        name = "Unknown";
        age = 0;
    }

    void displayInfo() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    Student student1;  // Default constructor is called
    student1.displayInfo();
    return 0;
}
```

Explanation:

- The default constructor initializes the object with default values (`name = "Unknown"` and `age = 0`).

- When `Student student1;` is executed, the default constructor is automatically invoked, and it assigns the default values to `student1`.

Example of a Parameterized Constructor:

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

class Student {
public:
    string name;
    int age;

    // Parameterized Constructor
    Student(string n, int a) {
        name = n;
        age = a;
    }

    void displayInfo() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    Student student1 ("Ali :", 20);  // Parameterized constructor
is called
    student1.displayInfo();
    return 0;
}
```

Explanation:

- The parameterized constructor takes two arguments (`n` and `a`) and uses them to initialize the `name` and `age` attributes.
- When `Student student1("Alice", 20);` is executed, the constructor is invoked, and the object is initialized with the given values.

2. Copy Constructor

A **copy constructor** creates a new object as a copy of an existing object. This is particularly useful when you need to duplicate an object.

Syntax:

```
ClassName(const ClassName &old_obj);
```

Example of a Copy Constructor:

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

class Book {
public:
    string title;
    int pages;

    // Parameterized Constructor
    Book(string t, int p) {
        title = t;
        pages = p;
    }

    // Copy Constructor
    Book(const Book &b) {
        title = b.title;
        pages = b.pages;
    }

    void displayInfo() {
      cout << "Title: " << title << ", Pages: " << pages << endl;
    }
};

int main() {
    Book book1("C++ Programming", 500);  // Parameterized
constructor is called
    Book book2 = book1;            // Copy constructor is called
    book2.displayInfo();
    return 0;
}
```

Explanation:

- The copy constructor copies the values of `book1`'s attributes to `book2`.
- When `Book book2 = book1;` is executed, the copy constructor is invoked, creating an identical copy of `book1`.

3. Destructor

A **destructor** is a special member function that is automatically called when an object is destroyed. Its primary purpose is to release any resources that the object may have acquired during its lifetime. The destructor has the same name as the class but is preceded by a tilde (~).

Example of a Destructor:

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

class Student {
public:
    string name;
    int age;

    // Constructor
    Student(string n, int a) {
        name = n;
        age = a;
        cout << "Constructor is called for " << name << endl;
    }

    // Destructor
    ~Student() {
        cout << "Destructor is called for " << name << endl;
    }

    void displayInfo() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    Student student1("Alice", 20);
    student1.displayInfo();
    return 0;
}
```

Explanation:

- The constructor is called when the object `student1` is created, and it initializes the object's attributes.
- The destructor is automatically called when the program ends, or when the object goes out of scope. It is used to clean up any resources that the object might have used.
- In the output, you will see the constructor message when the object is created and the destructor message when the object is destroyed.

4. Key Differences Between Constructor and Destructor:

Constructor	Destructor
Initializes an object when it is created.	Cleans up when an object is destroyed.
Can be overloaded (multiple constructors).	Cannot be overloaded (only one destructor).
Can take parameters (parameterized constructor).	Does not take any parameters.
Called automatically when an object is instantiated.	Called automatically when an object goes out of scope or is deleted.

5. Constructor Overloading

In C++, constructors can be **overloaded**, meaning a class can have multiple constructors with different parameter lists.

Example of Constructor Overloading:

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

class Rectangle {
public:
    int length, width;

    // Default Constructor
    Rectangle() {
        length = 0;
        width = 0;
    }

    // Parameterized Constructor
    Rectangle(int l, int w) {
        length = l;
        width = w;
    }

    void displayArea() {
        cout << "Area: " << length * width << endl;
    }
};

int main() {
  Rectangle rect1;           // Default constructor is called
  Rectangle rect2(5, 10); // Parameterized constructor is called

  rect1.displayArea();  //Displays area of default rectangle (0)
```

```
    rect2.displayArea();       // Displays area of rectangle (50)
    return 0;
}
```

Explanation:

- The class `Rectangle` has two constructors: a default constructor and a parameterized constructor.
- Depending on how the object is created, the appropriate constructor is called.
- `rect1` calls the default constructor, and `rect2` calls the parameterized constructor.

6. Conclusion:

- **Constructors** initialize objects and can be overloaded to provide different ways of creating an object.
- **Destructors** clean up resources when an object is destroyed.
- Proper management of resources using constructors and destructors ensures efficient and safe use of memory and other resources.

QUESTIONS OF THE LECTURE

1. What is the primary purpose of a constructor in a class?
2. When is a constructor automatically called in C++?
3. What is the main characteristic that distinguishes a constructor from other member functions?
4. What is the syntax used to define a copy constructor in C++?
5. Which constructor is invoked when an object is created using another existing object of the same class?
6. What happens when the destructor of a class is called?
7. How does the compiler identify a destructor in a class definition?
8. Why can constructors be overloaded, but destructors cannot?
9. What is the difference between a default constructor and a parameterized constructor?