Wk6 /S4/ Lecture # : DSOOPS-28

Topics Covered

- What is a Constructor?
- Purpose of Constructors
- Constructor Initialization List
- Types of Constructors (Default, Parameterized, Copy)
- What is a Destructor?
- Purpose of Destructors
- Constructor and Destructor Syntax
- Key Points and Best Practices
- Practice Problems (2 Easy, 1 Medium, 1 Hard)

What is a Constructor?

A constructor is a special member function of a class that is automatically called when an object is created. Its main role is to initialize the object's data members.

- Its name is exactly the same as the class name.
- It has no return type, not even `void`.
- Called automatically when an object is instantiated.

Purpose of Constructors

- To automatically set up objects with meaningful initial values.
- Avoid repetitive code for manually initializing each object.

Constructor Initialization List

The constructor initialization list is a powerful C++ feature that lets you initialize class members *directly* before the constructor's body executes. It appears after the constructor's parameter list and starts with a colon (`:`), followed by each member and its initial value in parentheses or curly braces.
Syntax:
`cpp`

```
ClassName(parameter_list) : member1(value1), member2(value2),
... {
    // Constructor body (optional)
}
```

Why Use Initialization Lists?

- More efficient, as members are initialized directly, not assigned.
- Required for initializing const members, reference members, and members of classes without default constructors.
- Preferred style for initializing members in most cases.

Example of Constructor Initialization List

```
#include <iostream>

class Point {
private:
    int x;
    int y;

public:
    // Constructor using initialization list
    Point(int a, int b) : x(a), y(b) {
        std::cout << "Point initialized with x = " << x << " and
y = " << y << "\n";
    }
};

int main() {
    Point p1(3, 4);  // Constructor called with initialization
list
    return 0;
}
```

Output:

```
Point initialized with x = 3 and y = 4
```

Example: Initialization of const and Reference Members

These must be initialized via an initialization list because they cannot be assigned after declaration.

```
#include <iostream>

class Example {
private:
    const int value;
    int& ref;

public:
    Example(int v, int& r) : value(v), ref(r) {
        std::cout << "Const value: " << value << " | Reference 
value: " << ref << "\n";
    }
};

int main() {
    int num = 10;
    Example ex(5, num);
    return 0;
}
```

Output:

```
Const value: 5 | Reference value: 10
```

Types of Constructors

1. Default Constructor:
   No parameters or all parameters have default values.
2. Parameterized Constructor:
   Takes parameters to initialize data members.
3. Copy Constructor:
   Copies data from another object of the same class.

What is a Destructor?

A destructor is a special member function called automatically when an object goes out of scope or is destroyed. It cleans up resources, like dynamically allocated memory.

- Name is the class name prefixed by tilde `~`.
- No parameters and no return type.
- Only one destructor per class.

Destructor Example

```
class Point {
public:
    Point() { std::cout << "Constructor called\n"; }
    ~Point() { std::cout << "Destructor called\n"; }
};

int main() {
    Point p;  // Constructor called here
    // Destructor called automatically when 'p' goes out of scope
    return 0;
}
```

Key Points & Best Practices

- If no constructor is defined, the compiler provides a *default constructor*.
- Use constructor initialization lists for efficiency and to initialize const/reference members.
- Define copy constructors when handling dynamic memory.
- Destructors clean up dynamic resources; match every `new` with a `delete`.
- You generally do not call destructors yourself—they are called automatically.

Practice Problems and Activities

Easy 1

Write a class `Rectangle` with two integer members: `width` and `height`.

- Define a default constructor that initializes them to `1`.
- Define a destructor that prints `"Rectangle destroyed"`.
- In `main()`, create a `Rectangle` object and observe output.

Easy 2

Create a class `Circle` with:

- A parameterized constructor taking radius as input using an initialization list.
- A method `area()` that returns area.
- In `main()`, create a `Circle` with radius `5` and print its area.

Medium

Implement a copy constructor for a class `Person` containing:

- A dynamically allocated `char* name` member.
- An `int age` member.
- Show deep copying by initializing one object from another.

Hard

Write a class `DynamicArray` with:

- An integer pointer `arr` and an integer `size`.

- A parameterized constructor using an initialization list that allocates memory dynamically.
- A destructor that releases allocated memory.
- Methods to set and get values at indices.
- Demonstrate creating and deleting objects, ensuring no memory leaks.

Wrap-Up & Key Takeaways

- Constructor initialization lists initialize members directly, improving efficiency and necessary for certain member types (const, references).
- Constructors are for object initialization; destructors for cleanup.
- Understand different constructor types: default, parameterized, copy.
- Use initialization lists as best practice when defining constructors.