📘 Understanding Constructors in C++

🧠 What is a Constructor?

A **constructor** is a special member function of a class in C++ that is **automatically called** when an object of the class is created. Its primary purpose is to **initialize the fields** (data members) of the object.

🔑 Key Characteristics:

- Has the **same name** as the class.
- **No return type**, not even `void`.
- Can be **overloaded** (multiple constructors with different parameters).
- Can be **default**, **parameterized**, or **copy constructors**.

🔄 Types of Constructors

Type	Description
Default	No parameters; initializes with default values.
Parameterized	Takes arguments to initialize fields.
Copy Constructor	Initializes a new object as a copy of an existing one.
Delegating	One constructor calls another constructor in the same class.
Explicit	Prevents implicit conversions for single-argument constructors.

🧱 Constructor Syntax

```
class ClassName {
  public:
    ClassName();              // Default constructor
    ClassName(int x);         // Parameterized constructor
    ClassName(const ClassName&); // Copy constructor
};
```

🍼 Example: Default and Parameterized Constructors

```
class Student {
  private:
    string name;
    int age;

  public:
    Student() {
        name = "Unknown";
        age = 0;
    }

    Student(string n, int a) {
        name = n;
        age = a;
    }
};
```

🚀 Member Initialization Lists

🔍 What is a Member Initialization List?

A **member initialization list** is a more efficient and idiomatic way to initialize class fields, especially for:

- **Const** members
- **Reference** members
- **Base class constructors**
- **Fields with non-trivial constructors**

✅ Syntax

```
ClassName(type1 arg1, type2 arg2) : field1(arg1), field2(arg2) {
    // constructor body (optional)
}
```

🍼 Example with Initialization List

```
class Student {
  private:
    string name;
    int age;

  public:
    Student(string n, int a) : name(n), age(a) {
        // Initialization done before body executes
    }
};
```

⚙️ Why Use Initialization Lists?

Benefit	Explanation
Performance	Avoids default construction followed by assignment.
Required for const/reference	These must be initialized at declaration.
Clarity	Makes initialization intent explicit.

🍼 Example with `const` and `reference`

```
class Book {
  private:
    const int id;
    string& title;

  public:
    Book(int i, string& t) : id(i), title(t) {}
};
```

✅ Summary

Concept	Description
Constructor	Initializes object fields
No return type	Constructors do not return values
Overloading	Multiple constructors with different parameters
Copy constructor	Creates a new object as a copy of another
Initialization List	Preferred way to initialize fields
	Avoids unnecessary default construction
	Required for `const`, references, and base classes

🎯 Quiz!

Here's a short quiz on the topic: quiz

🔭 Markdown Viewer

How to view the markdown files in a browser...

- Markdown Viewer

🧠 Lecture Practices

Here are the lecture Practices...

- Day 7
- Day 8
- Day 9

🔍 Lecture Quizzes

Here are the lecture quizzes...

- Day 7
- Day 8
- Day 9

Weekly Topics

Here are the topics for the week...

- Classes
- Structs
- Fields
- Getters and Setters
- Constructors
- Instances
- Inheritance
- Polymorphism
- Pointers
- Upcasting
- Misc. Concepts
- 4 Pillars of OOP