Tutorial 49 - Initialization List in Constructors (C++)

What is an Initialization List?

- An **initialization list** in a constructor initializes class data members directly before the body of the constructor is executed.
- **Syntax**:

```
Constructor(argument-list) : initialization-section {
    assignment + other code;
}

```

Code Example 1: Simple Initialization List

```
class Test {
    int a;
    int b;
public:
    Test(int i, int j) : a(i), b(j) { // Initialization list
        cout << "Constructor executed" << endl;
        cout << "Value of a is " << a << endl;
        cout << "Value of b is " << b << endl;
    }
};
int main() {
    Test t(4, 6); // Create an object and pass values
    return 0;
}

```

Explanation:

1. **Private Data Members**: `a` and `b`.
2. **Constructor**: Uses an initialization list `: a(i), b(j)` to set the values of `a` and `b`.
3. **Output**:

```
Constructor executed
Value of a is 4
Value of b is 6

```

Important Notes on Order of Initialization:

- Data members are **initialized in the order of their declaration** in the class, not the order in the initialization list.
- **Example 1 (Error)**:

```
Test(int i, int j) : b(j), a(i + b) {} // ERROR: `b` is initialized after `a`

```

- **Example 2 (Correct)**:

```
Test(int i, int j) : a(i), b(a + j) {} // CORRECT: `a` initialized before `b`

```

Why Use Initialization Lists?

1. **Performance**: Directly initializes members, avoiding default initialization followed by reassignment.
2. **Necessary for**:
    - **Const data members**: Must be initialized during object creation.
    - **Reference members**: Cannot be reassigned later.
    - **Base class constructors**: In derived classes, base class constructors can only be invoked in the initialization list.

Short Notes for Notebook:

1. **Syntax**:

    ```
    Constructor(args) : member1(value1), member2(value2) {
        // Constructor body
    }

    ```

2. **Execution**:
    - Members are initialized before the constructor body executes.
    - Order of initialization follows the **declaration order** in the class.
3. **Advantages**:
    - Avoids redundant initialization.
    - Required for const, reference, and base class constructors.
4. **Examples**:
    - **Simple Initialization**: `: a(i), b(j)`
    - **Correct Order**: `: a(i), b(a + j)`
    - **Error**: `: b(j), a(i + b)`
5. **Usage**: Essential for **const**, **references**, and **complex base class constructors**.