Tutorial 66 - C++ Class Templates with Default Parameters

Introduction

- In the **previous tutorials**, we learned about **single** and **multiple parameter templates** in C++.
- Today, we'll explore an **advanced feature—default parameters** in class templates.
- **Default template parameters** allow us to **set default data types** when none are provided in `main()`.

Understanding Default Parameters in Class Templates

📌 **Definition:**

- Similar to **default function arguments**, **class templates** can have **default data types**.
- If no data types are specified in `main()`, **the default ones are used**.

📌 **Syntax of a Class Template with Default Parameters:**

```
template <class T1 = int, class T2 = float, class T3 = char>
class ClassName {
    // Class body
};

```

Example: Class with Default Data Types

🚀 **Problem Statement:**

- Create a class `Harry` with:
  - Three variables: `a`, `b`, `c`.
  - Constructor to initialize these variables.
  - `display()` function to print them.
- **Set default data types** (`int`, `float`, `char`) for template parameters (`T1`, `T2`, `T3`).

📌 **Code: Class Template with Default Parameters**

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

template <class T1 = int, class T2 = float, class T3 = char>
class Harry {
    public:
        T1 a;
        T2 b;
        T3 c;

        Harry(T1 x, T2 y, T3 z) {
            a = x;
            b = y;
            c = z;
        }

        void display() {
            cout << "The value of a is " << a << endl;
            cout << "The value of b is " << b << endl;
            cout << "The value of c is " << c << endl;
        }
};

```

✅ **Explanation:**
✔️ `T1 = int, T2 = float, T3 = char` are **default parameters**.
✔️ If a user **doesn't specify** data types in `main()`, **default types** are used.
✔️ **Constructor assigns values** to `a`, `b`, and `c`.

Using the Class Template in `main()`

📌 **Steps:**
1️⃣ **Create an object** `h` **without specifying types**—default (`int, float, char`) will be used.
2️⃣ **Create another object** `g`, but **specify custom types** (`float, char, char`).
3️⃣ **Call** `display()` to print the values.

📌 **Code: Testing the Default Parameters**

```
int main() {
    Harry<> h(4, 6.4, 'c');  // Uses default types (int, float, char)
    h.display();

    cout << endl;

    Harry<float, char, char> g(1.6, 'o', 'c');  // Custom types (float, char, char)
    g.display();

    return 0;
}

```

✅ **Explanation:**
✔️ **First Object** `h` **(Uses Defaults)**

- `int` → `4`
- `float` → `6.4`
- `char` → `'c'`
  ✔️ **Second Object** `g` **(Custom Types)**
- `float` → `1.6`
- `char` → `'o'`
- `char` → `'c'`

📌 **Output:**

```
The value of a is 4
The value of b is 6.4
The value of c is c

The value of a is 1.6
The value of b is o
The value of c is c

```

✅ **No Error!** If no types were given, default ones were used!

Key Takeaways

🔹 **Class templates can have default parameters**, making them more flexible.
🔹 **If no types are given in `main()`**, the compiler **automatically assigns default types**.
🔹 **Avoids errors** and **reduces code complexity**.

🚀 **Next Tutorial: Function Templates!** Stay tuned! 🎯

Short Notes

What are Class Templates with Default Parameters?

- A **template** can have **default data types**, just like default function arguments.

Syntax

```
template <class T1 = int, class T2 = float, class T3 = char>
class ClassName {
    T1 var1;
    T2 var2;
    T3 var3;
};

```

Example

```
template <class T1 = int, class T2 = float, class T3 = char>
class Harry {
    T1 a;
    T2 b;
    T3 c;
};

```

Advantages

✔️ **Allows flexibility** when specifying types.
✔️ **Prevents errors** if no data type is given.
✔️ **Reduces code duplication** and **improves readability**.

📌 **Next Lesson:** Function Templates! 🚀 Keep coding! 🎯