Tutorial 65 - C++ Templates with Multiple Parameters

Introduction

- In the previous tutorial, we learned about **templates** and their use in **generic programming**.
- We **generalized a class** to calculate the **dot product** of two vectors **for any data type**.
- Now, we'll learn how to **handle multiple parameters** in a template, allowing **different data types** in a class.

Understanding Multiple Parameters in Templates

📌 **Definition:**

- Just like functions can have multiple parameters, **templates can also accept multiple data types**.
- We define **multiple template parameters** using a **comma-separated list**.

📌 **Syntax of a Template with Multiple Parameters:**

```
template <class T1, class T2>
class ClassName {
    // Class body
};

```

Example: Class with Two Data Members of Different Data Types

🚀 **Problem Statement:**

- Create a class `myClass` with:
  - `data1` (integer)
  - `data2` (character)
- Create a function `display()` to print both values.

📌 **Code: Without Templates (Fixed Data Types)**

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

class myClass {
    public:
        int data1;
        char data2;

        myClass(int a, char b) {
            data1 = a;
            data2 = b;
        }

        void display() {
            cout << this->data1 << " " << this->data2 << endl;
        }
};

int main() {
    myClass obj(1, 'c');
    obj.display(); // Output: 1 c
    return 0;
}

```

🚨 **Problem:**

- The class **only works for** `int` **and** `char`.
- We **cannot** pass other data types without creating a new class.

Solution: Using Templates with Multiple Parameters

📌 **Steps to Modify the Code:**
1️⃣ **Define a template with multiple parameters** (`T1, T2`).
2️⃣ Replace **fixed data types (**`int`**,** `char`**)** with `T1`**,** `T2`.
3️⃣ Accept different data types dynamically in `main()`.

📌 **Code: Template with Multiple Parameters**

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

template <class T1, class T2>
class myClass {
    public:
        T1 data1;
        T2 data2;

        myClass(T1 a, T2 b) {
            data1 = a;
            data2 = b;
        }

        void display() {
            cout << this->data1 << " " << this->data2 << endl;
        }
};

int main() {
    myClass<int, char> obj1(1, 'c'); // Integer & Character
    obj1.display(); // Output: 1 c

    myClass<int, float> obj2(1, 1.8); // Integer & Float
    obj2.display(); // Output: 1 1.8

    return 0;
}

```

✅ **Explanation:**
✔️ `T1` and `T2` **allow flexibility** in choosing data types.
✔️ Works for **multiple combinations** like `(int, char)`, `(float, double)`, etc.
✔️ Saves **time and effort** by avoiding multiple class definitions.

Key Takeaways

- 🔹 **Templates can have multiple parameters**, making them more flexible.
- 🔹 They **allow us to pass different data types dynamically** from `main()`.
- 🔹 **Saves effort**—no need to create separate classes for each data type combination.

🚀 **Next Tutorial: Templates with Default Parameters!** Stay tuned! 🎯

Short Notes

What are Templates with Multiple Parameters?

- A **template** can accept **multiple data types** using `<class T1, class T2>`.

Syntax

```
template <class T1, class T2>
class ClassName {
    T1 var1;
    T2 var2;
};

```

Example

```
template <class T1, class T2>
class myClass {
    T1 data1;
    T2 data2;
};

```

Advantages

✔️ **Supports multiple data types dynamically**
✔️ **Reduces code duplication**
✔️ **Increases flexibility and efficiency**

📌 **Next Lesson:** Templates with **default parameters**. 🚀 Keep coding! 🎯