Tutorial 64 - Writing Our First C++ Template in VS Code

Introduction

- **Templates** in C++ allow us to write **generic classes and functions** that work with multiple data types.
- They **reduce code duplication** and follow the **DRY (Don't Repeat Yourself) principle**.
- In this tutorial, we will **implement templates** by modifying a class that calculates the **dot product of two vectors**.

Dot Product of Integer Vectors (Without Templates)

📌 **Steps to Calculate the Dot Product (Without Templates):**
1️⃣ Create a **class** `vector` with an integer pointer `arr`.
2️⃣ Define an integer variable `size` to store the size of the vector.
3️⃣ Implement a **constructor** to initialize the vector.
4️⃣ Write a function `dotProduct()` that:

- Takes another vector as a parameter.
- **Traverses both vectors**, multiplies corresponding elements, and adds them to `d`.
- Returns the **dot product** as an integer.
  5️⃣ Print the result in `main()`.

📌 **Code: Dot Product Without Templates**

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

class vector {
    public:
        int *arr;
        int size;

        vector(int m) {
            size = m;
            arr = new int[size];
        }

        int dotProduct(vector &v) {
            int d = 0;
            for (int i = 0; i < size; i++) {
                d += this->arr[i] * v.arr[i];
            }
            return d;
        }
};

int main() {
    vector v1(3); // Vector 1
    v1.arr[0] = 4;
    v1.arr[1] = 3;
    v1.arr[2] = 1;

    vector v2(3); // Vector 2
    v2.arr[0] = 1;
    v2.arr[1] = 0;
    v2.arr[2] = 1;

    int a = v1.dotProduct(v2);
    cout << a << endl; // Output: 5
    return 0;
}

```

🚨 **Problem:** This code **only works for integers**.

- If we try to use `float`, `double`, or `char`, we need **separate classes**, increasing complexity.

Dot Product Using Templates (Generalized for Any Data Type)

📌 **Steps to Convert the Code into a Template:**
1️⃣ **Define a template** using `template <class T>`.
2️⃣ Replace `int` with `T` (a placeholder for any data type).
3️⃣ Modify the constructor, function return type, and vector size accordingly.
4️⃣ Pass the **data type** when declaring vectors in `main()`.

📌 **Code: Dot Product Using Templates**

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

template <class T>
class vector {
    public:
        T *arr;
        int size;

        vector(int m) {
            size = m;
            arr = new T[size];
        }

        T dotProduct(vector &v) {
            T d = 0;
            for (int i = 0; i < size; i++) {
                d += this->arr[i] * v.arr[i];
            }
            return d;
        }
};

int main() {
    vector<float> v1(3); // Vector 1 with float data type
    v1.arr[0] = 1.4;
    v1.arr[1] = 3.3;
    v1.arr[2] = 0.1;

    vector<float> v2(3); // Vector 2 with float data type
    v2.arr[0] = 0.1;
    v2.arr[1] = 1.90;
    v2.arr[2] = 4.1;

    float a = v1.dotProduct(v2);
    cout << a << endl; // Output: 6.82
    return 0;
}
```

✅ **Advantages of Using Templates:**
✔️ Works for **all data types** (`int`, `float`, `double`, etc.).
✔️ No need to write **separate classes** for different data types.
✔️ Reduces **effort, time, and chances of errors**.

Key Takeaways

🔹 **Templates make code reusable** and follow the **DRY principle**.
🔹 **Generic classes** work for multiple data types without rewriting code.
🔹 **Competitive programmers** use templates to improve efficiency.

🚀 **Next Tutorial:** Learning about **multiple parameters in templates**. Stay tuned!

Short Notes

What are Templates?

- A **template** is a blueprint for creating classes and functions that work with any data type.

Why Use Templates?

✅ **Avoids code duplication**
✅ **Supports multiple data types**
✅ **Improves efficiency**

Syntax of a Template Class

```
template <class T>
class ClassName {
    T var;
};
```

Example Usage

```
vector<int> v1(3);    // Integer vector
vector<float> v2(3);  // Float vector
```

Dot Product Using Templates (Generalized Version)

```
template <class T>
class vector {
    T *arr;
    int size;
};
```

Advantages of Templates

✔️ **Saves time**
✔️ **Reduces redundancy**
✔️ **Increases code efficiency**
✔️ **Essential for competitive programming**

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