Tutorial 63 - C++ Templates – A Must for Competitive Programming

Introduction

- Templates are one of the **most powerful** features in C++.
- They allow us to **write generic code** that works with **any data type**, reducing redundancy.
- Templates are especially useful in **competitive programming** as they help write efficient, reusable code.

What is a Template in C++?

📌 **Definition:**

- A **template** in C++ allows us to **define generic classes and functions** without specifying data types explicitly.
- Instead of writing multiple versions of a class or function for different data types, we write **one generic template**, and the compiler generates the required code automatically.

📌 **Analogy:**

- Just like **classes are templates for objects**,
- **Templates are templates for classes and functions.**

✅ **Templates follow the DRY (Don't Repeat Yourself) principle**, eliminating redundant code.

Why Use Templates?

1️⃣ DRY (Don't Repeat Yourself) Rule

- Without templates, we would need to **write the same code multiple times** for different data types.
- Example: If we need a **vector** for `int`, `float`, and `char`, we would have to write **three separate classes**.
- With templates, we **write one class** that works for all data types.

2️⃣ Generic Programming

- Templates enable **generic programming**, meaning we can **write code once and use it for multiple data types**.
- This makes code **more efficient, reusable, and maintainable**.

📌 **Example Without Templates (Redundant Code)**

```
class VectorInt {
    int *arr;
    int size;
};

class VectorFloat {
    float *arr;
    int size;
};

class VectorChar {
    char *arr;
    int size;
};

```

❌ **Repetitive & Inefficient**

📌 **Solution: Using Templates (Single Generic Class)**

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

```

✅ **One class for all data types!**

Syntax of Templates in C++

📌 **Steps to Create a Template Class:**

1. Use `template <class T>` to define a template.
2. Use `T` as a **placeholder** for the data type inside the class.
3. The actual data type will be **specified** when an object of the template is created.

📌 **Example: Template for a Vector Class**

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

// Declaring a template
template <class T>
class Vector {
    T *arr;
    int size;

public:
    Vector(T *arr) {
        // Constructor code
    }
    // Other methods
};

int main() {
    Vector<int> myVec1();   // Vector of integers
    Vector<float> myVec2(); // Vector of floats

    return 0;
}

```

✅ This **single class** can handle **multiple data types** like `int`, `float`, `char`, etc.

Key Advantages of Templates in Competitive Programming

1️⃣ **Saves Time** – Write once, use for multiple data types.
2️⃣ **Increases Code Efficiency** – Reduces redundant code and improves readability.
3️⃣ **Gives an Edge Over Others** – Many top programmers use templates for quick, efficient coding.
4️⃣ **Enhances Reusability** – Works for functions and classes alike.

Summary Table

Feature	Without Templates	With Templates
Code Redundancy	High (Repeating same code for different types)	Low (One generic code for all types)
Efficiency	Low (Multiple versions of the same class)	High (Single codebase)
Readability	Decreases as code grows	Increases, making maintenance easier
Usage in Competitive Coding	Less effective	Highly effective

Short Notes

What are Templates?

- A **template** allows defining a **generic class or function** that works with multiple data types.

Why Use Templates?

- Follows **DRY principle**, avoids repetition.
- Supports **generic programming**, improving efficiency.

Template Syntax

```
template <class T>
class ClassName {
    T var;  // T can be any data type
};

```

Example Usage

```
Vector<int> v1;   // Integer vector
Vector<float> v2; // Float vector

```

Advantages of Templates

✅ Saves time
✅ Reduces redundancy
✅ Improves code efficiency
✅ Essential for competitive programming

Final Thoughts

- Templates are **a must for competitive programming** as they make coding **faster and more efficient**.
- **Next Tutorial:** Writing a **program using templates** for better understanding. 🚀

📌 **Don't miss out on mastering templates!** Keep coding and stay ahead of your competitors. 🎯