Tutorial 67 - C++ Function Templates & Function Templates with Parameters

Introduction

- **Previously**, we studied **class templates** and how they generalize classes for different data types.
- Now, we will explore **function templates**, which allow us to create **generic functions**.
- **Why Function Templates?**
  - Writing multiple functions for different data types **violates the DRY (Don't Repeat Yourself) principle**.
  - Function templates **eliminate redundancy** and **increase code efficiency**.

Basic Function Without Templates

📌 **Problem Statement:**

- Write a function that **calculates the average of two integers**.

📌 **Code: Function for Integer Averages**

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

float funcAverage(int a, int b) {
    float avg = (a + b) / 2.0;
    return avg;
}

int main() {
    float a;
    a = funcAverage(5, 2);
    printf("The average of these numbers is %f", a);
    return 0;
}

```

✅ **Output:**

```
The average of these numbers is 3.500000

```

🚨 **Issue:**

- This function only works for **integers**.
- If we need the same function for **floats, doubles, or mixed types**, we must **write separate functions**.
- **Solution?** Use **function templates**!

Function Template for Averages

📌 **Steps to Generalize the Function:**
1️⃣ **Use the `template` keyword** to define **generic data types**.
2️⃣ **Replace fixed types (like `int`)** with **template parameters** (`T1`, `T2`).

📌 **Code: Function Template for Averages**

```
template<class T1, class T2>
float funcAverage(T1 a, T2 b) {
    float avg = (a + b) / 2.0;
    return avg;
}

```

✅ **Now this function works for all data type combinations!**

Using the Function Template

📌 **Example 1: Passing Two Integers**

```
int main() {
    float a;
    a = funcAverage(5, 2);
    printf("The average of these numbers is %f", a);
    return 0;
}

```

✅ **Output:**

```
The average of these numbers is 3.500000

```

📌 **Example 2: Passing One Integer & One Float**

```
int main() {
    float a;
    a = funcAverage(5, 2.8);
    printf("The average of these numbers is %f", a);
    return 0;
}

```

✅ **Output:**

```
The average of these numbers is 3.900000

```

🚀 **Advantage:** We didn't have to write separate functions for different types!

Function Template for Swapping Values

📌 **Problem Statement:**

- Create a **generic swap function** that works for **any data type**.

📌 **Code: Generic Swap Function**

```
template <class T>
void swapp(T &a, T &b) {
    T temp = a;
    a = b;
    b = temp;
}
```

```
7
```

✅ **Now, we can swap integers, floats, or even characters!**

📌 **Example: Swapping Two Integers**

```
int main() {
    int x = 10, y = 20;
    swapp(x, y);
    cout << "x = " << x << ", y = " << y;
    return 0;
}

```

✅ **Output:**

```
x = 20, y = 10

```

📌 **Example: Swapping Two Floats**

```
int main() {
    float x = 5.5, y = 9.9;
    swapp(x, y);
    cout << "x = " << x << ", y = " << y;
    return 0;
}

```

✅ **Output:**

```
x = 9.9, y = 5.5

```

🚀 **Advantage:** No need to write multiple `swap()` functions for different types!

Key Takeaways

✅ **Function templates** allow writing **generic functions**.
✅ **Reduces redundancy**—we don't need to rewrite functions for different data types.
✅ **Supports multiple data types** using template parameters.
✅ **Can be used for mathematical operations, swapping, searching, and more!**

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

Short Notes

What are Function Templates?

- **A function template allows creating generic functions** that work with **any data type**.
- **Similar to class templates**, function templates avoid code duplication.

Syntax

```
template <class T>
ReturnType FunctionName(T param1, T param2) {
```

```
    // Function body
}

```

Example: Function Template for Averages

```
template<class T1, class T2>
float funcAverage(T1 a, T2 b) {
    return (a + b) / 2.0;
}

```

Example: Swap Function Template

```
template <class T>
void swapp(T &a, T &b) {
    T temp = a;
    a = b;
    b = temp;
}

```

Advantages of Function Templates

✔️ **Reduces redundant code** (follows DRY principle).
✔️ **Works for multiple data types** (int, float, char, etc.).
✔️ **Saves time and improves efficiency**.

📌 **Next Topic: Overloading Function Templates!** 🚀 Keep coding! 🎯