Tutorial 68 - Member Function Templates & Overloading Template Functions in C++

Introduction

- We have already learned **class templates** and **function templates**.
- Now, we will explore:
  a. **Declaring a template function outside a class using `::` (scope resolution operator).**
  b. **Overloading template functions.**

Defining Member Functions Outside a Class

📌 **Steps:**
1️⃣ **Declare a class template** and a **class named** `Harry`.
2️⃣ **Define a variable** `data` of type `T` inside the class.
3️⃣ **Create a constructor** to initialize `data`.
4️⃣ **Define the function** `display()` inside the class (Code Snippet 1).
5️⃣ **Define** `display()` **outside the class** using `::` (Code Snippet 2).

📌 **Code Snippet 1: Function Inside the Class**

```
template <class T>
class Harry {
public:
    T data;
    Harry(T a) {
        data = a;
    }
    void display() {
        cout << data;
    }
};

```

📌 **Code Snippet 2: Function Outside the Class**

```
template <class T>
class Harry {
public:
    T data;
    Harry(T a) {
        data = a;
    }
    void display();  // Function declaration
};

// Function definition outside the class
template <class T>
void Harry<T>::display() {
    cout << data;
}

```

🚀 **Why Define Functions Outside the Class?**
✔ **Improves readability** when class definitions are long.
✔ **Separates declarations and definitions**, making debugging easier.

📌 **Code Snippet 3: Calling the Function from `main()`**

```
int main() {
    Harry<int> h(5.7);  // 5.7 is cast to int (5)
    cout << h.data << endl;
    h.display();
    return 0;
}

```

✅ **Output:**

```
5
5

```

Overloading Function Templates

📌 **Function Overloading:**

- **Multiple functions** with the **same name** but **different parameters**.
- In **template function overloading**, we define:
  a. **A normal function** (for a specific data type).
  b. **A template function** (for all data types).

📌 **Code Snippet 4: Overloading a Template Function**

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

void func(int a) {
    cout << "I am first func() " << a << endl;
}

template<class T>
void func(T a) {
    cout << "I am templatised func() " << a << endl;
}

```

📌 **Code Snippet 5: Calling `func()` from `main()`**

```
int main() {
    func(4);  // Calls the exact match
    return 0;
}

```

✅ **Output:**

```
I am first func() 4

```

🚀 **Explanation:**

- The **normal function (`void func(int a)`) has higher priority** if an exact match exists.
- If there was no `func(int a)`, the **templated function would be called** instead.

📌 **Example: Calling `func()` with a `char`**

```
int main() {
    func('A');  // No exact match, calls templated function
    return 0;
}

```

✅ **Output:**

```
I am templatised func() A

```

🚀 **Key Takeaway:**

- **Exact match (non-template function) is always preferred**.
- **Template functions are used when no exact match is found**.

Key Takeaways

✔ **Member function templates** can be defined **inside or outside** the class.
✔ **Scope resolution operator `::` is used** when defining functions outside the class.
✔ **Function templates can be overloaded** using:

- A **specific function** (exact match).
- A **templated function** (generic case).
  ✔ **Exact matches are always prioritized** over template functions.

🚀 **Next Topic: Standard Template Library (STL)!** 🔥 Get ready for competitive programming!

Short Notes

Member Function Templates

- Used to define **class member functions** as **templates**.
- Can be **defined inside** or **outside** the class using `::`.

📌 **Example: Function Inside the Class**

```
template <class T>
class Harry {
public:
    T data;
    Harry(T a) { data = a; }
    void display() { cout << data; }
};

```

📌 **Example: Function Outside the Class**

```
template <class T>
class Harry {
public:
```

```
    T data;
    Harry(T a) { data = a; }
    void display();
};

template <class T>
void Harry<T>::display() { cout << data; }

```

Overloading Template Functions

- **Function overloading** means **same function name but different parameters**.
- **Template function overloading** allows:
  a. **Normal function (exact match priority).**
  b. **Generic template function.**

📌 **Example: Overloading a Function Template**

```
void func(int a) {
    cout << "I am first func() " << a << endl;
}

template<class T>
void func(T a) {
    cout << "I am templatised func() " << a << endl;
}

```

📌 **Calling `func()` with an Integer (Exact Match Exists)**

```
int main() {
    func(4);  // Calls the normal function (higher priority)
}

```

✅ **Output:**

```
I am first func() 4

```

📌 **Calling `func()` with a Character (No Exact Match)**

```
int main() {
    func('A');  // Calls the template function
}

```

✅ **Output:**

```
I am templatised func() A

```

🚀 **Rule: Exact match (non-template function) is always prioritized!**

📌 **Next Topic: STL - The Powerhouse for Competitive Programming!** 🚀 Keep coding! 🔥