Member Function Templates & Overloading Template Functions in C++ | C++ Tutorials for Beginners #68

So, since we have finished learning about the two template categories, we can now swiftly dive deep into if it's possible for a template function to get overloaded, and if yes, then how.
Before starting to know what an overloaded template function is, we'll learn how to declare a template function outside a using the scope resolution operator, '::'.

First, we'll revise how to write a function inside the class by just following the snippet given below.
1. We'll declare a template, then a class named Harry.
2. We'll then define a variable *data* inside that class with variable data type T.

3. We then make a constructor feeding the value received from the main to data.
4. And then, we'll write the function, *display* and write its code.

This was an unchallenging task. But when we need the function to be declared outside the class, we follow the code snippet 2.

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

Code Snippet 1: Writing function inside the class

Here, we first write the function declaration in the class itself. Then move to the outside and use the scope resolution operator before the function and after the name of the class Harry along with the data type T. We must specify the function data type, which is void here. And it must be preceded by the template declaration for class T.

And write the display code inside the function and this will behave as expected. See the output below the

snippet.

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

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

Code Snippet 2: Writing function outside the class

So to check if it's working all fine, we'll call this function from the main.

```
int main()
{
    Harry<int> h(5.7);
    cout << h.data << endl;
```

```
    h.display();
    return 0;
}
```

Code Snippet 3: Calling the function from the main

And the output is:

```
5
5
PS D:\MyData\Business\code playground\C++ course>
```

Now, we'll move to the **overloading of a function template**. Overloading a function simply means assigning two or more functions with the same name, the same job, but with different parameters. For that, we'll declare a void function named func. And a template function with the same name. Follow the snippet below to do the same:

1. We made two void functions, one specified and one generic using a template.
2. The first one receives an integer and prints the integer with a different prefix.
3. The generic one receives the value as well as the data type and prints the value with a different prefix.
4. Now, we'll wish to see the output of the following functions, by calling them from the main. Refer to the main program below the snippet below.

```
#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 4: Overloading the template function

And now when we call the function func, we'll be interested to know which one among the two it calls. So here since we've entered a value with an integer parameter, it finds its exact match in the overloading and calls that itself. That is, it gives its exact match the highest priority. Refer to the output below the snippet:

```
int main()
{
    func(4);  //Exact match takes the highest priority
    return 0;
}
```

Code Snippet 5: Calling function func from the main

And the output is,

```
I am first func() 4
PS D:\MyData\Business\code playground\C++ course>
```

If we hadn't created the first function with int data type, the call would have gone to the templatised func only because a template function is an exact match for every kind of data type.

So this was enough preparation for the next topic, STL(Standard Template Library). It might have sounded boring to you for quite a few days, but the results would fascinate you once we enter STL, which is a must for all the competitive programmers out there.Thank you, for being with me throughout, hope you liked the tutorial. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. I hope you enjoy them all. See you all in the next tutorial where we'll start the STL. Till then keep coding.

Previous Next