Overloading Function Templates

Template:

A template is a tool that reduces the efforts in writing the same code as templates can be used at those places.

A template function can be overloaded either by a non-template function or using an ordinary function template.

Function Overloading: In function overloading, the function may have the same definition, but with different arguments. Below is the C++ program to illustrate function overloading:

```
// C++ program to demonstrate the
// function overloading
#include <bits/stdc++.h>
using namespace std;
// Function to calculate square
void square(int a)
{
    cout << "Square of " << a
         << " is " << a * a
         << endl}
// Function to calculate square
void square(double a)
{
    cout << "Square of " << a
         << " is " << a * a
         << endl;}
// Driver Code
int main()
{
    // Function Call for side as
    // 9 i.e., integer
    square(9);
    // Function Call for side as
    // 2.25 i.e., double
    square(2.25);
    return 0;}
```

Output:

Square of 9 is 81
Square of 2.25 is 5.0625

Explanation:

In the above code, the **square** is overloaded with different parameters.

The function **square** can be overloaded with other arguments too, which requires the same name and different arguments every time.

To reduce these efforts, C++ has introduced a generic type called **function template**.

Function Template: The function template has the same syntax as a regular function, but it starts with a keyword **template** followed by template parameters enclosed inside **angular**

brackets <>.

```
template <class T>
T functionName(T arguments)
{
    // Function definition
    ……. …… ….. …….
}
```

where, **T** is **template** argument accepting different arguments and **class** is a keyword.

Template Function Overloading:

The name of the function templates is the same but called with different arguments is known as **function template overloading**.

If the function template is with the ordinary template, the name of the function remains the same but the number of parameters differs.

When a function template is overloaded with a non-template function, the function name remains the same but the function's arguments are unlike.

```
#include <bits/stdc++.h>
using namespace std;
// Template declaration
template <class T>
// Template overloading of function
void display(T t1)
{
   cout << "Displaying Template: "
        << t1 << "\n";
}
// Template overloading of function
void display(int t1)
{
   cout << "Explicitly display: "
        << t1 << "\n";
}
// Driver Code
int main()
{
   // Function Call with a
   // different arguments
   display(200);
   display(12.40);
   display('G');
   return 0;}
```

Output:

```
Explicitly display: 200

Displaying Template: 12.4

Displaying Template: G
```