Function Templates in C++

Function Templates We write a generic function that can be used for different data types. Examples of function templates are sort(), max(), min(), printArray().

A function template starts with the keyword template followed by template parameter(s) inside <> which is followed by the function definition.

```
template <typename T>

T functionName(T parameter1, T parameter2, ...) {

 // code

}
```

C++

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

// One function works for all data types. This would work
// even for user defined types if operator '>' is overloaded
template <typename T> T myMax(T x, T y)
{
    return (x > y) ? x : y;
}

int main()
{
    cout << myMax<int>(3, 7) << endl; // Call myMax for int
    cout << myMax<double>(3.0, 7.0)
        << endl; // call myMax for double
    cout << myMax<char>('g', 'e')
        << endl; // call myMax for char
```

```
    return 0;
}
```

Output

```
7
7
g
```

Below is the program to implement Bubble Sort using templates in C++:

```
// CPP code for bubble sort
// using template function
#include <iostream>
using namespace std;

// A template function to implement bubble sort.
// We can use this for any data type that supports
// comparison operator < and swap works for it.
template <class T> void bubbleSort(T a[], int n)
{
    for (int i = 0; i < n - 1; i++)
        for (int j = n - 1; i < j; j--)
            if (a[j] < a[j - 1])
                swap(a[j], a[j - 1]);
}

// Driver Code
int main()
{
    int a[5] = { 10, 50, 30, 40, 20 };
    int n = sizeof(a) / sizeof(a[0]);

    // calls template function
    bubbleSort<int>(a, n);

    cout << " Sorted array : ";
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
```

```
    cout << endl;

    return 0;
}
```

Output

```
Sorted array : 10 20 30 40 50
```