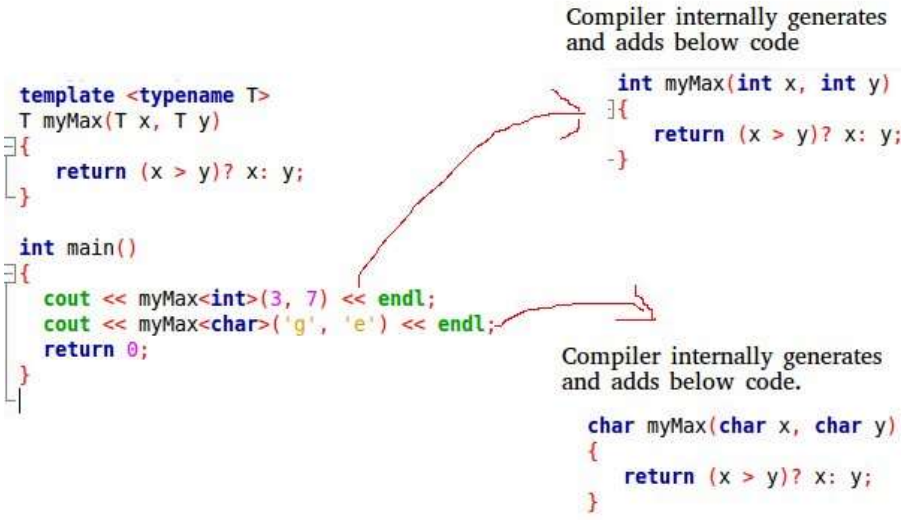
Templates in C++ with Examples

Difficulty Level : Medium • Last Updated : 06 Aug, 2022

A **template** is a simple yet very powerful tool in C++. The simple idea is to pass data type as a parameter so that we don't need to write the same code for different data types. For example, a software company may need to sort() for different data types. Rather than writing and maintaining multiple codes, we can write one sort() and pass data type as a parameter.
C++ adds two new keywords to support templates: *'template'* and *'typename'*. The second keyword can always be replaced by the keyword 'class'.

How Do Templates Work?

Templates are expanded at compiler time. This is like macros. The difference is, that the compiler does type checking before template expansion. The idea is simple, source code contains only function/class, but compiled code may contain multiple copies of the same function/class.



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

Know more about ***Generics in C++***.

CPP

```cpp
#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
```

Output

```
7
7
g
```

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

CPP

```cpp
// 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()
```

```
    // 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
```

Class Templates like function templates, class templates are useful when a class defines something that is independent of the data type. Can be useful for classes like LinkedList, BinaryTree, Stack, Queue, Array, etc.

Following is a simple example of a template Array class.

CPP

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

template <typename T> class Array {
private:
    T* ptr;
    int size;

public:
    Array(T arr[], int s);
    void print();
};

template <typename T> Array<T>::Array(T arr[], int s)
{
    ptr = new T[s];
    size = s;
    for (int i = 0; i < size; i++)
        ptr[i] = arr[i];
```

```
{
    for (int i = 0; i < size; i++)
        cout << " " << *(ptr + i);
    cout << endl;
}

int main()
{
    int arr[5] = { 1, 2, 3, 4, 5 };
    Array<int> a(arr, 5);
    a.print();
    return 0;
}
```

Output

```
 1 2 3 4 5
```

Can there be more than one argument to templates?

Yes, like normal parameters, we can pass more than one data type as arguments to templates. The following example demonstrates the same.

CPP

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

template <class T, class U> class A {
    T x;
    U y;

public:
    A() { cout << "Constructor Called" << endl; }
};

int main()
{
    A<char, char> a;
    A<int, double> b;
    return 0;
}
```

```
Constructor Called
Constructor Called
```

Can we specify a default value for template arguments?

Yes, like normal parameters, we can specify default arguments to templates. The following example demonstrates the same.

CPP

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

template <class T, class U = char> class A {
public:
    T x;
    U y;
    A() { cout << "Constructor Called" << endl; }
};

int main()
{
    A<char> a; // This will call A<char, char>
    return 0;
}
```

Output

```
Constructor Called
```

What is the difference between function overloading and templates?

Both function overloading and templates are examples of polymorphism features of OOP. Function overloading is used when multiple functions do quite similar (not identical) operations, templates are used when multiple functions do identical operations.

What happens when there is a static member in a template class/function?

What is template specialization?

Template specialization allows us to have different codes for a particular data type. See Template Specialization for more details.

Can we pass non type parameters to templates?

We can pass non-type arguments to templates. Non-type parameters are mainly used for specifying max or min values or any other constant value for a particular instance of a template. The important thing to note about non-type parameters is, that they must be const. The compiler must know the value of non-type parameters at compile time. Because the compiler needs to create functions/classes for a specified non-type value at compile time. In the below program, if we replace 10000 or 25 with a variable, we get a compiler error. Refer to the following link.

CPP

```
// C++ program to demonstrate working of non-type
// parameters to templates in C++
#include <iostream>
using namespace std;

template <class T, int max> int arrMin(T arr[], int n)
{
    int m = max;
    for (int i = 0; i < n; i++)
        if (arr[i] < m)
            m = arr[i];

    return m;
}

int main()
{
    int arr1[] = { 10, 20, 15, 12 };
    int n1 = sizeof(arr1) / sizeof(arr1[0]);

    char arr2[] = { 1, 2, 3 };
    int n2 = sizeof(arr2) / sizeof(arr2[0]);

    // Second template parameter to arrMin must be a
    // constant
```

```
}
```

Output

```
10
1
```

Here is an example of a C++ program to show different data types using a constructor and template. We will perform a few actions

- passing character value by creating an object in the main() function.
- passing integer value by creating an object in the main() function.
- passing float value by creating an object in the main() function.

C++

```
// #include <conio.h>
#include <iostream>
using namespace std;

// defining a class template
template <class T> class info {
public:
    info(T A) // constructor of type template
    {
        cout << "\n"
             << "A = " << A
             << " size of data in bytes:" << sizeof(A);
    } // end of info()
}; // end of class

// Main Function
int main()
{
    // clrscr();

    // passing character value by creating an objects
    info<char> p('x');

    // passing integer value by creating an object
    info<int> q(22);
```

```
    return 0;
} // end of Main Function
```

Output

```
A = x size of data in bytes:1
A = 22 size of data in bytes:4
A = 2.25 size of data in bytes:4
```

What is template metaprogramming?

Refer to the following article – Template Metaprogramming

Take a Quiz on Templates. Java also supports these features. Java calls it generics.

Previous Next

Related Articles

1. not1 and not2 function templates in C++ STL with Examples
2. Templates and Static variables in C++
3. Templates and Default Arguments

5. Overloading function templates in C++
6. Templates in C++ vs Generics in Java
7. Difference between Preprocessor Directives and Function Templates in C++
8. Implementing Stack Using Class Templates in C++
9. isalpha() and isdigit() functions in C with cstring examples.
10. strtok() and strtok_r() functions in C with examples

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : anu7699, stathpaul, biplab_prasad, hakukiro, dey0btpch57lmvgz5mqhpaiqn337p09fd8yq1lw4, RishabhPrabhu, aryaman2050

Article Tags : cpp-template, C++

Practice Tags : CPP

Improve Article | Report Issue

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

ReactJS

NodeJS

Video Internship