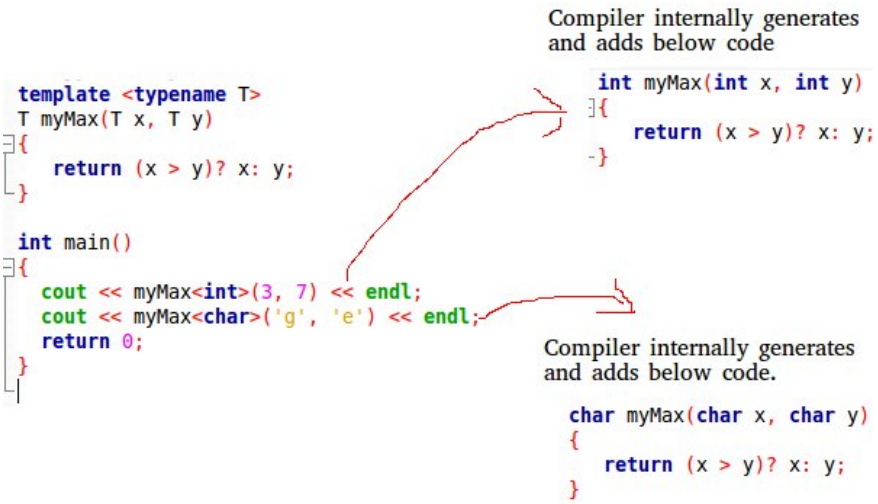
Templates in C++

A template is a simple and 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 sort() for different data types. Rather than writing and maintaining the 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 keyword 'class'.

How templates work?

Templates are expanded at compiler time. This is like macros. The difference is, 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 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 on Generics in 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(a, 5);

    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 template Array class.

```
#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];
}

template <typename T>
void Array<T>::print() {
    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 arguments to templates?

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

```
#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;
}
```

Output:

```
Constructor Called
Constructor Called
```

Can we specify default value for template arguments?

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

```
#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 feature of OOP. Function overloading is used when multiple functions do similar operations, templates are used when multiple functions do identical operations.

What happens when there is static member in a template class/function?
Each instance of a template contains its own static variable. See Templates and Static variables for more details.

What is template specialization?
Template specialization allows us to have different code for a particular data type. See Template Specialization for more details.

Can we pass nontype 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, they must be const. The compiler must know the value of non-type parameters at compile time. Because compiler needs to create functions/classes for a specified non-type value at compile time. In below program, if we replace 10000 or 25 with a variable, we get a compiler error. Please see this.

Below is a C++ program.

```
// A 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
   cout << arrMin<int, 10000>(arr1, n1) << endl;
   cout << arrMin<char, 256>(arr2, n2);
   return 0;
}
```

Output:

```
10
1
```

What is template metaprogramming?
See Template Metaprogramming

You may also like to take a quiz on templates.

Java also support these features. Java calls it generics .

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Posts:

Templates and Default Arguments
Templates in C++ vs Generics in Java
Templates and Static variables in C++
Variadic function templates in C++
not1 and not2 function templates in C++ STL with Examples
Implementing upper_bound() and lower_bound() for Ordered Set in C++
How to flatten a Vector of Vectors or 2D Vector in C++
Remove odd frequency characters from the string
Different ways to use Const with Reference to a Pointer in C++
std::to_address in C++ with Examples
Program to create Custom Vector Class in C++
std::is_trivially_copy_constructible in C/C++
Difference between Python and C++
tgamma() method in C/C++ with Examples

Improved By : anu7699, stathpaul

Article Tags : C++ cpp-template

Practice Tags : CPP

44

2.5

To-do Done

Based on **40** vote(s)

Feedback/ Suggest Improvement | Add Notes | Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Comments **Community** **Privacy Policy** **Login**

Recommend 7 Tweet Share **Sort by Newest**

Join the discussion…

LOG IN WITH **OR SIGN UP WITH DISQUS**

Name

tamry hamza • 11 days ago • edited

Thanks that was very helpful articl demonstrating how powerful are template in C++ and also well explained :) .

• Reply • Share ›

Zhiyuan Gu • 5 months ago • edited

I think the example used to demonstrate template with nontype parameters should be like this:

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

return m;
}

• Reply • Share ›

Zhiyuan Gu ➚ Zhiyuan Gu • 5 months ago
()after template should be <>

• Reply • Share ›

pawan choure • 5 months ago • edited

1 • Reply • Share ›

mrbuddhu • 5 months ago
Helpful :)

• Reply • Share ›

Data_Beast • 9 months ago
"'Templates are expanded at compiler time. This is like macros."

This is not like macros. Macros are expanded during pre-processing and templates are expanded during compiler proper.

Use
g++ -E main.cpp

This will print the pre-processed code and one can see the macros expanded, while the templates are not.

• Reply • Share ›

Avatar This comment was deleted.

Data_Beast ➚ Guest • 8 months ago
Just because you think it, doesn't make it right.

Check facts.

While thinking of templates as Macros can help you understand the concept of templates, it is important to distinguish the difference

-Macros are expanded during pre-processing

-Templates are expanded during compiler proper.

Why?

When you need to debug you can find out which stage of the compilation process is failing and home in on where the issue in your code takes place.

A mediocre programmer will stay at the idea that macros are like templates, which they are not, because if they were, they would be named the

because if they were, they would be named the same thing.

1 • Reply • Share ›

jinendra family ➔ Data_Beast
• 6 months ago

Dude,may i know ?
how can i use g++ -E main.cpp

• Reply • Share ›

Swastik Sahoo • 10 months ago • edited

In the main function should it not be bubblesort<int>(a,5) instead of bubblesort(a,5).

1 • Reply • Share ›

Robert Hamm • 10 months ago

I have a function to load a Deck52 cards. It is only used on the Deck52 obj. I am thinking, a Template would not be good to use in this type of scenario. Am I right? To use other Data Types would be a good use for Template, but not where only one DataType can be use. For this, I am thinking, to use other decks I would use abstract Interface Class ..

• Reply • Share ›

Nfrankenstein • 2 years ago

template<class x="">
void f()
{
X * arr = (X*)malloc(sizeof(X));
}

How will you call f() from main?

• Reply • Share ›

Kartik Gupta • 2 years ago

Can we specify default value for template arguments?
Here we have
template < Class T , Class U = char >
Now Suppose I had
template < Class U = char , Class T >
How Can i call <char,int> ? using just one argument ?

• Reply • Share ›

Aman Sheikh • 2 years ago

nice

• Reply • Share ›

Bindeshwar Kush • 2 years ago

A computer science portal for geeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org