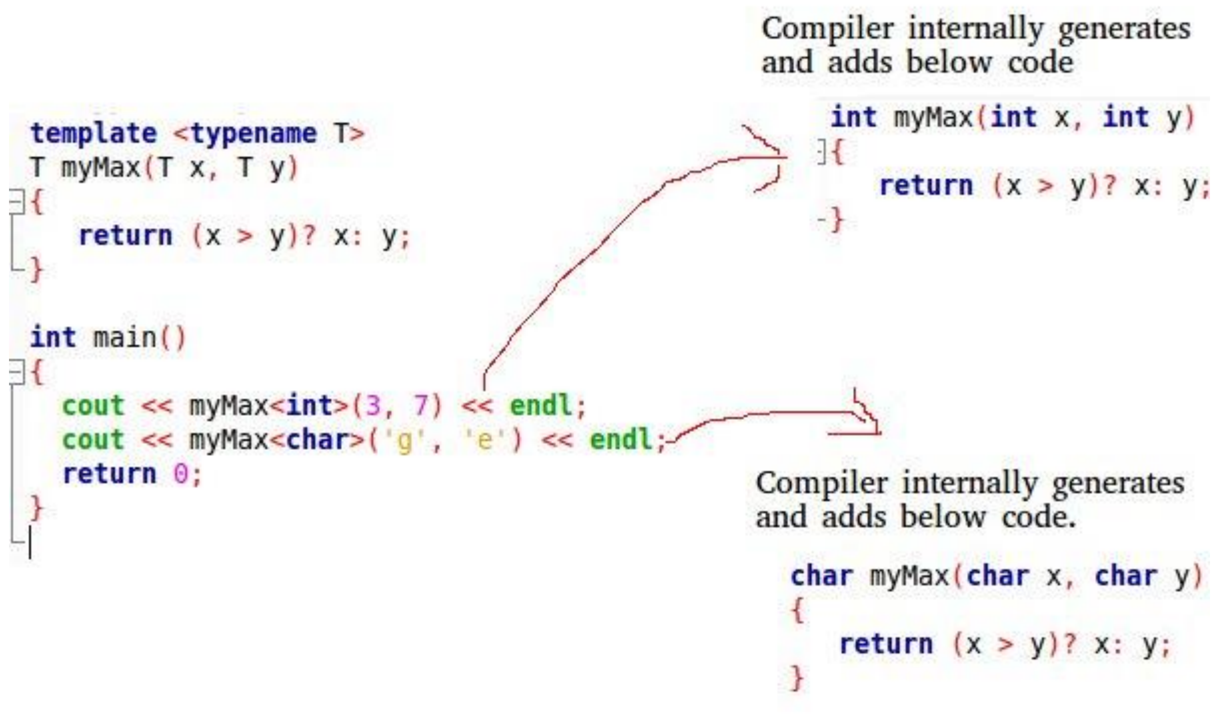
Template in C++

Jaydeep shah <radhey04ec@gmail.com>

A **template** is a simple yet very powerful tool in C++. The simple idea is to **pass the data type as a parameter** so that we don't need to write the same code for different data types.

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.



```
template <typename T>
T myMax(T x, T y)
{
    return (x > y)? x: y;
}

int main()
{
   cout << myMax<int>(3, 7) << endl;
   cout << myMax<char>('g', 'e') << endl;
   return 0;
}
```

Compiler internally generates and adds below code

```
int myMax(int x, int y)
{
    return (x > y)? x: y;
}
```

Compiler internally generates and adds below code.

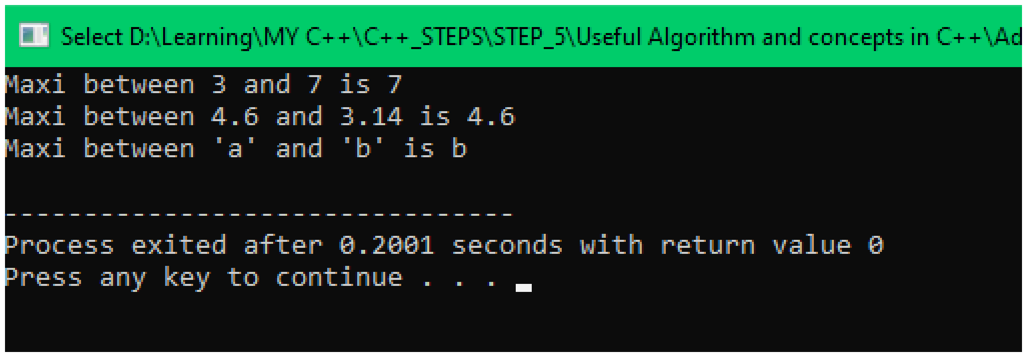
```
char myMax(char x, char y)
{
    return (x > y)? x: y;
}
```

Function Templates

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

Example:

```
//Templates in C++
//A template is a simple yet very powerful tool in C++.
//The simple idea is to pass the data type as a parameter so that we don't need to write the same code for different data types


//Example : Find maximum value using function maxi - which can support multiple datatype.
#include<iostream>
using namespace std;


//Write genric function - return type is "J" , and argument type "J" (which is created using template)
template <typename J> J maxi(J x, J y)
{
	return ((x > y) ? x : y);
}

int main()
{
	//Call function including passing datatype
	cout << "Maxi between 3 and 7 is " << maxi<int>(3, 7) << endl;

	//Call generic function with datatype
	cout << "Maxi between 4.6 and 3.14 is " << maxi<double>(4.6, 3.14) << endl;

	//Call generic function with datatype
	cout << "Maxi between 'a' and 'b' is " << maxi<char>('a', 'b') << endl;


	return 0;
}
```



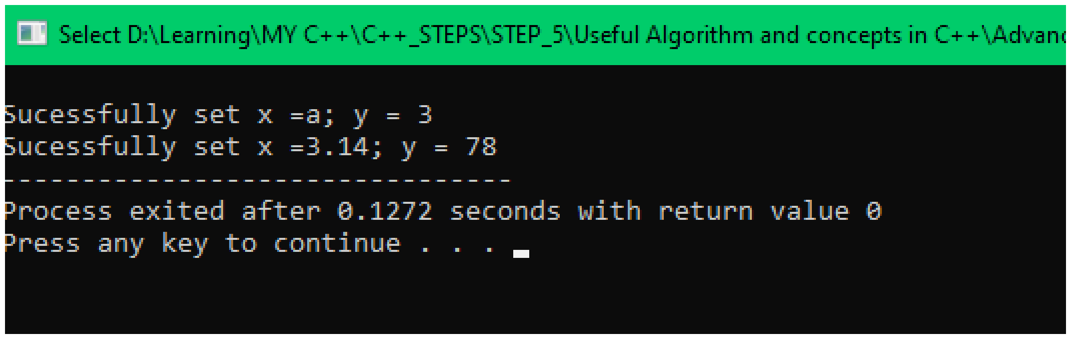
Select D:\Learning\MY C++\C++_STEPS\STEP_5\Useful Algorithm and concepts in C++\Ad

```
Maxi between 3 and 7 is 7
Maxi between 4.6 and 3.14 is 4.6
Maxi between 'a' and 'b' is b

--------------------------------
Process exited after 0.2001 seconds with return value 0
Press any key to continue . . .
```

We can use Template concept with class. Here when inner member of class not fix datatype at that time this concept is useful.

Example:

```
//T2: Template in C++
//Tenmplate with two different function

#include<iostream>
using namespace std;

//Use two different datatype with class
template<class my_datatype, class my_datatype_2>class test
{
private:
	my_datatype x;
	my_datatype_2 y;
public:

	//Constructor function
	test(my_datatype a, my_datatype_2 b)
	{
		x = a;
		y = b;

		cout << endl << "Sucessfully set x =" << x << "; y = " << y;
	}
};


int main()
{
	//Create object, pass dattype information
	test <char, int>t1('a', 3);
	test <double, int>t2(3.14, 78);
	return 0;


}
```



Select D:\Learning\MY C++\C++_STEPS\STEP_5\Useful Algorithm and concepts in C++\Advanc

```
Sucessfully set x =a; y = 3
Sucessfully set x =3.14; y = 78
--------------------------------
Process exited after 0.1272 seconds with return value 0
Press any key to continue . . .
```