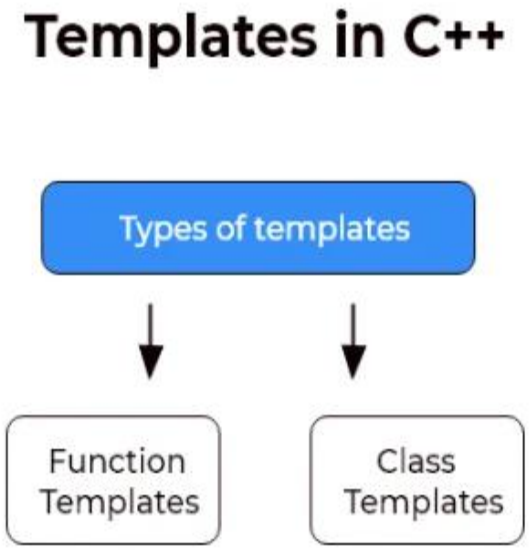
Function Templates

Dr. Mohsin H. Hussein

- The template is one of C++'s most sophisticated and high-powered features.

- Although not part of the original specification for C++, it was added several years ago and is supported by all modern C++ compilers.

- Using templates, it is possible to create generic functions and classes.



The generic function

- A generic function defines a general set of operations that will be applied to various types of data.
- For example, the Quicksort sorting algorithm is the same whether it is applied to an array of integers or an array of floats.
- In generic function, the compiler will automatically generate the correct code for the type of data that is actually used when you execute the function.

- To understand what is template , let take the following a small function that you might write to find the maximum of two integers.

```
int maximum(int a, int b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

- And here's a small function that you might write to find the maximum of two double numbers.

```
double maximum(double a, double b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

- A generic function is created using the keyword **template**. It is used to create a template (or framework) that describes what a function will do, leaving it to the compiler to fill in the details as needed. The general form of a template function definition is shown here:

```
template <class dtype>
ret-type  func_name  ( parameters)
{
// body of function
}
```

The solution (template function)

- This template function can be used with many data types.

```
template <class dtype>
dtype maximum(dtype a, dtype b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

```
// calling Functions
cout << maximum(1 , 2);
cout << maximum(1.3 , 0.9);
...
```

```
#include <iostream>

template <class X>
void swapargs(X &a, X &b)
{
X temp;
temp = a;
a = b;
b = temp;
}
```

```
int main() {
int i=10, j=20;
double x=10.1, y=23.3;
char a='x', b='z';
swapargs(i, j); // swap integers
swapargs(x, y); // swap floats
swapargs(a, b); // swap chars
cout << << i << ' ' << j << '\n';
cout << << x << ' ' << y << '\n';
cout << << a << ' ' << b << '\n';
return 0; }
```

Using Standard Parameters with Template Functions

- You can mix standard parameters with generic type parameters in a template function. These nongeneric parameters work just like they do with any other function. For example:

```
template <class Type>
Type  array_max(Type data[  ], int size)
{
Type max;
max= data[0];
    for (i = 1; i < size; i++)
        if (data[i] > max) max= data[i];
    return max;  }
```

A Function with Two Generic Types

```
#include <iostream>
template <class type1 , class  type2>
void myfunc(type1 x, type2 y)
{
cout << x << ' ' << y << '\n';
}
int main()
{
myfunc(10, "I like C++");
myfunc(98.6 , 19 );
return 0;   }
```

Thank you