Function Templates

In this lesson, we'll explore function templates in detail.

WE'LL COVER THE FOLLOWING

Function Templates

A function template is defined by the keyword `template` followed by type or non-type parameters in front of a concrete function. After that, replace the concrete types or non-types with the type or non-type parameters in the function.

- The keywords `class` or `typename` declare the parameters.
- The name `T` is usually used for the first parameter.
- The parameters can be used in the body of the function.

Passing Arguments in Function Templates

In the given code snippet, we'll look at how we can call the initialized variables with our template. Looking at line 2, the function arguments `x` and `y` in the function `xchg` must have the same type. By providing two type parameters like in line 5, the types of arguments can be different. In line 9, we see a non-type template parameter `N`.

```
template <typename T>
void xchg(T& x , T& y){
...

template <typename T, typename T1>
```

```
void  add(T& x, T1& y){
...


template <int N>
int nTimes(int n){
...
```

Instantiation

The process of substituting the template parameters for the template arguments is called **template instantiation**.

The compiler:

- Automatically creates an instance of the function template.
- Will automatically create a function template if the template parameters can be derived from the function arguments.

If the compiler cannot deduce the template arguments from the function arguments, then they must be specified explicitly.

```
template <typename T>
void xchg(T& x, T& y){ ...

int a, b;
xchg(a, b);

template <int N>
int nTimes(int n){ ...

int n = 5;
nTimes<10>(n);
```

Overloading

Function templates can be overloaded.

The following rules hold:

1. Templates do not support automatic type conversion.
2. If a free function is better or equally as good as a function template that already exists, the free function can be used.
3. The type of the function template can be specified explicitly.

```
    func<type>(...)
```

4. We can specify that we are only interested in a specific instantiation of a function template.

```
func<>(...)
```

To learn more about function templates, click here.

In the next lesson, we'll look at some examples of function templates.