Template Arguments

In this lesson, we will look at template arguments.

WE'LL COVER THE FOLLOWING

Template arguments can automatically be deduced for function templates. The compiler deduces the template arguments for the function arguments. Function Templates act like functions.

Conversion

- The compiler uses simple conversions for deducing the template arguments from the function arguments.
- The compiler removes `const` or `volatile` from the function arguments. It converts C-arrays and functions to pointers.

The types of function arguments must be exact since no conversion takes place.

```
template <typename T>
  bool isSmaller(T fir, T sec){
    return fir < sec;
   }
isSmaller(1, 5LL); // ERROR int != long long int
```

Providing a second template parameter completes the example.

```
template <typename T, typename U>
```

```
bool isSmaller(T fir, U sec){
  return fir < sec;


}
isSmaller(1, 5LL);   // OK
```

Explicit Template Arguments

- They are necessary if the template argument cannot be deduced from the function argument.
- They are required if a specific instance of a function template is needed.

```
template <typename R, typename T, typename U>
R add(T fir, U sec){
  return fir * sec; }
add<long long int>(1000000, 1000000LL);
```

> Missing template arguments are automatically deduced from the function arguments.

Automatic Return Type

By using `auto` and `decltype`, you can use function templates in the alternative function syntax that will automatically deduce their return type.

```
template< typename T1, typename T2>
auto add(T1 fir, T2 sec) -> decltype(fir + sec){
  return fir + sec;
}
auto res = add(1.2, 5);
```

- `auto`: indicates the delayed return type
- `decltype`: defines the return type

> With C++14, `decltype(fir + sec)` is no longer necessary.

Default Arguments

- The default for template parameters can be specified for class templates and function templates.

- If a template parameter has a default parameter, all subsequent template parameters also need a default argument.

```
template <typename T, typename Pred = std::less<T>>
bool isSmaller(T fir, T sec, Pred pred = Pred(){
  return pred(fir, sec);
}
```

Let’s take a look at a few examples of template arguments in the next lesson.