Automatic Return Type

Now, we'll learn a technique to automatically deduce the return type of a function.

WE'LL COVER THE FOLLOWING

- Rules
- C++14
- Further information

Using `auto` and `decltype` together, a function template is able to automatically deduce its return type. Here's a function with a trailing return type:

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

Rules

The syntax above follows a few rules:

- `auto`: introduces the syntax for the delayed return type.
- `decltype`: declares the return type.
- The alternative function syntax is obligatory.

Using this strategy, the return type of the function can be deduced by the types of its arguments.

C++14

C++14 makes things even simpler. We don't need to use `decltype` to deduce

the function's return type anymore. `auto` handles everything:

```
template <typename T1, typename T2>
auto add(T1 fir, T2 sec){
    return fir + sec;
}
```

With the expression `decltype(auto)`, `auto` uses the same rules to determine the type as `decltype`. This means in particular, no decay takes place.

Both declarations are identical:

```
decltype(expr) v= expr;
decltype(auto) v= expr;
```

This syntax also applies to the automatic return type of a function template:

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

> **Note**: When a function template multiple return statements, all of them must have the same type.

Further information

- decay

In the next lesson, we'll look at a coding example of automatic type deduction.