

# Exception Specifications in the Type System

This lesson touches upon specifying exception and the reasons for adding this feature.

## WE'LL COVER THE FOLLOWING ^

- Specifying Exception
- Why use this Feature?

## Specifying Exception #

Exception Specification for a function didn't use to belong to the type of the function, but now it will be part of it.

You can now have two function overloads: **one with** `noexcept` **and one without it.**

See below:

```
using TNoexceptVoidFunc = void (*)() noexcept;
void SimpleNoexceptCall(TNoexceptVoidFunc f) {
    f();
}

using TVoidFunc = void (*);
void SimpleCall(TVoidFunc f) {
    f();
}

void fNoexcept() noexcept { }
void fRegular() { }

int main() {
    SimpleNoexceptCall(fNoexcept);
    //SimpleNoexceptCall(fRegular); // cannot convert

    SimpleCall(fNoexcept); // converts to regular function
    SimpleCall(fRegular);
}
```



A pointer to `noexcept` function can be converted to a pointer to a regular function (this also works for a pointer to a member function). But it's not possible the other way around (from a regular function pointer into a function pointer that is marked with `noexcept`).

## Why use this Feature? #

One of the reasons for adding the feature is a chance to optimise the code better. If the compiler has a guarantee that a function won't throw, then it can generate faster code.

Also, as described in the previous chapter [about Language Fixes](#), in C++17, the Exception Specification is cleaned up. Effectively, you can only use [the `noexcept` specifier](#) for declaring that a function won't throw.

*Extra Info:* The change was proposed in: [P0012R1](#).

---

Now that you've learned all the concepts, it's time for a short quiz. Click next in order to attempt the quiz!