The ~~noexcept~~ Function Specification

A function having a **noexcept** exception specification denotes a programmatically observable, runtime-enforced guarantee that no thrown C++ exception will ever escape that function's body.

Description

C++, when first standardized in 1998, provided a mechanism to declare which specific exception types a function may throw including none at all (**throw()**), and to detect any violations of that specification at runtime by terminating the program; see Section 2.1."**noexcept** Operator" on page 615. In practice, using this feature, known as **dynamic exception specifications**, led to more fragile and less efficient programs and, hence, was never used widely. This original foray into exception specifications was deprecated by C++11[1] in favor of a simpler scheme — employing a newly minted keyword, **noexcept** — that communicates only the critically important information of whether a thrown exception (of any kind) may ever escape from the body of a function so annotated.

Unconditional exception specifications

We may choose to decorate a function to indicate that it *cannot* exit via a thrown exception; any exception that would have escaped will instead be caught automatically at runtime, and `std::terminate` will be invoked (see *Potential Pitfalls — Overly strong contracts guarantees* on page 1112). We provide this **programmatically accessible** annotation by inserting the **noexcept** exception specification after the parameter list and (for a member function) any cv-ref qualifiers, but before any pure-virtual marker (`= 0`), any specifiers such as **override** (see Section 1.1."**override**" on page 104), or **final** (see Section 3.1."**final**" on page 1007), and, if present, any trailing return type (see Section 1.1."Trailing Return" on page 124):

```
struct B  // noexcept goes after any cv-ref qualifiers but before = 0.
{
    virtual int foo() const& noexcept = 0;  // The noexcept keyword goes thusly.
    virtual int bar() const& = 0;  // Derived classes may have an exception spec.
};

struct D1 : B  // noexcept goes before override or final.
{
    int foo() const& noexcept override;  // OK
    int bar() const& noexcept override;  // OK
};
```

[1] C++17 removed all support for dynamic exception specifications; only the **throw()** spelling remained (until C++20 when that too was removed) and only as an alias for **noexcept**.