```
struct D2 : B  // The noexcept on an overriding function must be compatible.
{
    int foo() const& override;  // Error, incompatible exception specification
    int bar() const& override;  // OK
};

template <typename T>
auto sum(T a, T b) noexcept -> decltype(a + b);  // goes before trailing return
```

Note that exception specifications on **overriding** virtual functions must be compatible with (i.e., the same or stricter than) the corresponding virtual function **declaration**(s) in the corresponding base class(es) — e.g., see **struct**s `D1` and `D2` above. For additional details and a full example involving dynamic exception specifications, see Section 2.1."**noexcept** Operator" on page 615.

Decorating a function using just the keyword **noexcept** is equivalent to using the longer, conditional **noexcept** syntax, **noexcept(true)**. The absence of **noexcept**, other than for the special cases of **defaulted special member functions** (see Section 1.1."Defaulted Functions" on page 33), as well as *any* **destructors** and deallocation functions (see below), is equivalent to using the conditional **noexcept** syntax, **noexcept(false)**.

An **implicitly declared** special member function for a **class type**, `T`, will be **noexcept(true)** unless the implicitly generated function must invoke a function that is not **noexcept(true)**.

A **user-declared special member function** having no explicitly stated exception specification that is **defaulted** *in class scope* will have the same exception specification as if it had been declared implicitly. If a defaulted user-declared special member function *is* also decorated with an explicitly stated exception specification, the stated specification will be honored irrespective of what might have otherwise been generated implicitly.[2]

For example, consider a family of classes, `S0` … `S3`, each having an explicitly stated exception specification for, say, its user-declared default constructor (but the same applies to the other five special member functions too)[3]:

[2] As originally designed for C++11, providing an exception specification on a defaulted user-declared special member function that did not match the implicit exception specification was **ill formed**. In 2014, a solution to CWG issue 1778 (**usa13**) was resolved — as a **defect report** — so that any such previously ill-formed special member functions would become **deleted**. Implementing this change proved problematic because exception specifications — being a **complete-class context** — could not generally be determined implicitly before they were needed. Moreover, C++ developers might legitimately want to explicitly supersede the implicitly generated exception specification in either direction; see *Use Cases — Declaring nonthrowing move operations* on page 1094. In 2019, changes introduced by **smith19** — also as a defect report — enabled an explicit exception specification on a defaulted user-declared special member function to simply take precedence over the implicit specification.

[3] Note that on older compilers that predate the implementation of the aforementioned changes the constructor of `S3` will be deleted, as will the corresponding implicit constructors of `C3` and `D3` below.