```
static_assert(!noexcept(f0()), "");  // doesn't say it doesn't throw
static_assert(!noexcept(f1()), "");  //    "     "   "    "      "

static_assert(!noexcept(f2()), "");  // f2 may throw an int.
static_assert(!noexcept(f3()), "");  // f3  "    "    "  "

static_assert(!noexcept(f4()), "");  // f4 may throw a double.
static_assert(!noexcept(f5()), "");  // f5  "    "   "    "

static_assert(!noexcept(f6()), "");  // f6 may throw int or double.
static_assert(!noexcept(f7()), "");  // f7  "    "    "   "    "

static_assert( noexcept(f8()), "");  // f8 may not throw.
static_assert( noexcept(f9()), "");  // f9  "   "    "
```

There are, however, practical drawbacks to dynamic exception specifications.

1. **Brittle** — These classic, fine-grained exception specifications attempt to provide excessively detailed information that is not programmatically useful and is subject to frequent changes due to otherwise inconsequential updates to the implementation.
2. **Expensive** — When an exception is thrown, a *dynamic*-exception list must be searched at run time to determine if that specific exception type is allowed.
3. **Disruptive** — When an exception reaches a *dynamic*-exception specification, the stack must be unwound, whether or not the exception is permitted by that specification, losing useful stack-trace information if the program is about to terminate.

These deficiencies proved, over time, to be insurmountable, and dynamic-exception specifications other than **throw()** were largely unused in practice.

As of C++11, dynamic-exception specifications are officially deprecated[3] in favor of the more streamlined **noexcept** specifier (see Section 3.1."**noexcept** Specifier" on page 1085), which we introduce briefly in the next section.

## Introducing noexcept exception specifications for functions

C++11 introduces an alternative exception-specification mechanism for arbitrary free functions, member functions, and lambda expressions (see Section 2.1."Lambdas" on page 573):

```
void f() noexcept(expr);  // expr is a Boolean constant expression.
void f() noexcept;        // same as void f() noexcept(true)
```

Instead of specifying a *list* of exceptions that may be thrown, whether *any* exception may be thrown is specified. As with C++03, the lack of any annotation is the equivalent of saying anything might be thrown (except for destructors, which are **noexcept** by default):

[3]C++17 removes std::unexpected and all dynamic exception specifications other than **throw()**, which becomes a synonym for **noexcept** before **throw()** too is removed by C++20.