```
#include <cassert>   // standard C assert macro
#include <iostream>  // std::cout

void f(int n)
{
    assert(factorial(5) == 120);
        // OK, factorial(5) might be evaluated at compile time since 5 is a
        // constant expression but the argument of assert does not have to be
        // a constant expression.

    static_assert(factorial(5) == 120, "");
        // OK, factorial(5) is evaluated at compile time since arguments of
        // static_assert must be constant expressions.

    std::cout << factorial(n);
        // OK, likely evaluated at run time since n is not a constant
        // expression

    static_assert(factorial(n) > 0, "");
        // Error, n is not a constant expression.
}
```

As illustrated above, simply invoking a **constexpr** function with arguments that are constant expressions does *not* guarantee that the function will be evaluated at compile time. The only way to *guarantee* compile-time evaluation of a **constexpr** function is to invoke it in places where a constant expression is mandatory.

If the value of a constant expression is needed at compile time (e.g., for the bounds of an array) and computing that value involves the execution of an operation that is not available at compile time (e.g., **throw**), the compiler will have no choice but to report an error:

```
constexpr int h(int x) { return x < 5 ? x : throw x; }  // OK, constexpr func

int a4[h(4)];  // OK, creates an array of four integers
int a6[h(6)];  // Error, unable to evaluate h on 6 at compile time
```

In the code snippet above, although we are able to size the file-scope[2] `a4` array because the path of execution within the valid **constexpr** function `h` does not involve a **throw**, such is

[2] A common extension of popular compilers allows, by default, variable-length arrays within function bodies but, as illustrated above, *never* at file or namespace scope:

```
void g()
{
    int a4[h(4)];  // OK, creates an array of four integers
    int a6[h(6)];  // Warning: ISO C++ forbids variable-length array a6.
                   // But with some compilers, h(6) might be invoked at
                   // run time and throw.
}
```

It is only by compiling with `-Wpedantic` that GCC issues a warning.