Let's now consider what happens if we flip the order of `g()` and `s`:

```
struct S { ~S(); };
void g() G_EXCEPTION_SPEC;
void f2() F_EXCEPTION_SPEC
{
    S s;  // non-trivially destructible automatic variable
    g();  // throwing expression comes after variable
}
```

Lastly, we have a scenario in which declaring either `f2`, `g`, or both to be **`noexcept`** yields a potential net reduction in size for the resulting translation unit. Let's look at each of the four possible combinations of **`noexcept`** assignment in turn. If neither `f2` nor `g` is declared **`noexcept`**, then the compiler is obliged to generate unwind code to clean up `s` and propagate a potential exception thrown from `g` back to the caller of `f2`, which in turn can significantly increase the size of the body of `f2`. If `g` is declared **`noexcept`**, then, regardless of whether or not `f2` is declared **`noexcept`**, no additional code needs to be laid down to guard against the impossible case of an exception being thrown from `g`. Finally, if `f` is declared **`noexcept`** but `g` is not, most, if not all, of the size benefit is likely to be realized, but now there is the possibility of a small increase in the size of `f` resulting from the obligatory call to `std::terminate`.

A measurement of relative compiled-code sizes in both build modes (unoptimized/optimized) for each of the two function definitions, `f1` and `f2`, and for each of the four combinations of exception specifications on `g` and `f`, respectively, confirms our hypothesis, as shown in Table 1.[18]

Table 1: Comparing code sizes without/with `noexcept` on `f` and/or `g`

Candidate Function		G_S = noexcept		
		F_S = noexcept		
`void f1() F_S { g(); S s; }`	88/84[a]	88/84	88/84	96/92
`void f2() F_S { S s; g(); }`	152/132	88/84	88/84	96/92

[a] Where, e.g., 88/84 means 88 bytes unoptimized / 84 bytes optimized

Looking at the data above, we observe that there was no opportunity to reduce the size of `f1` from a throwing `g` because the invocation of `g` preceded construction of any non-trivially destructible non**`static`** local variables. But, unless `g` is itself declared **`noexcept`** (or

[18] For information on how to approach benchmarking the impact of **`noexcept`**, see **dekker19b**.