Table 3: Interface and implementation properties pertaining to a *nofail* function

Reporting Contract	Fallible Implementation	Nickname	Example Function(s) Having These Properties
No	No	nofail	**int** half(**int**); or **double** sqrt(**double**);
Yes	No	reliable	FILE* fopen(**const char***, **const char***);
No	Yes	optimistic	**int** factorial(**int**); // recursive impl.
Yes	Yes	general	getGoodNewsPlease

As the table above summarizes, for a function overall to be nofail, it must be both **non-reporting** and have an infallible implementation, as was the case for half, std::abs, and sqrt. A function such as fopen that is infallible but reporting is, nonetheless, *reliable* in that we can call it from a nofail function and, if it fails, fall back on an infallible (albeit perhaps less efficient or otherwise less desirable) way to satisfy the contract. As an example, calculating the area of a polygon is always possible, but if we can quickly and reliably determine that it is a rectangle and access that representation, we can bypass the slower, more general algorithm.

Sometimes we might write a contract that is accidentally or, perhaps, even deliberately over constrained with no provision *in the contract* for reporting failure: "Whenever you call me, I promise to do Y (or die trying)." As an example, imagine a function, allocateMutex, that claims always to return a pointer to a newly allocated Mute~~a~~x object:

```
class Mutex { /*...*/ };

Mutex* allocateMutex();
    // Allocate a distinct Mutex and return its address. Period. :)
```

If we were to place the allocateMutex function above in a loop that ran enough times, we could exhaust available memory on any physical machine. On the other hand, the intent of the function is for gainfully employed engineers to do their work without forcing them to test for running out of heap memory, a situation they might not intend to spend the time supporting in practice. For that reason, the part of the contract that says "...or, if I cannot, I will return **nullptr**" is deliberately omitted so as not to invite useless checks on the part of the caller. On the other hand, should this function ever fail to allocate a Mutex, returning normally would be an explicit violation of the contract. Ironically, failing to provide an escape clause in an English contract for human developers is analogous to what the compiler does at the object-code level when it sees a function marked **noexcept**; see *Use Cases — Reducing object-code size* on page 1101.