Finally, we will often encounter *general* functions, e.g., `getGoodNewsPlease`, which must rely on an *optimistic* function in their implementation but report other erroneous situations through reliably methods:

```
const char* getGoodNewsImp(Mutex* p);  // reliable function
    // Return good news; otherwise return nullptr.

const char* getGoodNewsPlease()        // general function
    // Return good news; otherwise return nullptr.
{
    Mutex* mtx = allocateMutex();  // fallible
    return getGoodNewsImp(mtx);    // reliable
}
```

In the admittedly contrived example above, a *reliable* function, `getGoodNewsImp`, having a **reporting contract** and an **infallible implementation**, is called from a function, `getGoodNewsPlease`, that calls an *optimistic* function having a **fallible** implementation. Consequently, the higher-level wrapper function has a **reporting contract** and a **fallible implementation**.

Determining whether a function overall is **nofail** can be challenging, especially with more involved reporting mechanisms than a simple return status. Recall that the `fopen` function returned status in two ways: (1) via the return and (2) via global state. To report success/failure, only a single bit need be transmitted. Two other possible reporting channels would be to **signal** or to throw an exception.

As our next specimen, let's look at two seemingly similar member functions of `std::vector`:

```
#include <stdexcept>  // std::out_of_range

template <typename T>
class vector {
// ...
    T& operator[](std::size_t index);
        // Return a reference to the modifiable element at the specified index.
        // The behavior is undefined unless index < size().

    T& at(std::size_t index);
        // Return a reference to the modifiable element at the specified index
        // unless !(index < size()) in which case throw std::out_of_range.
// ...
};
```

Can we say that either of these contracts is **nofail**? The answer is yes, exactly one, but which one? Recall that answering this question involves answering two subquestions: (1) ~~is the contract~~ ~~**nonreporting**~~, and (2) is the implementation **infallible**. When it is not immediately obvious, it can be helpful to translate a function that reports errors by some other mechanism to a canonical form that returns zero on success and a nonzero value otherwise, possibly storing additional information in global state (e.g., `errno`):