```
#include <vector>     // std::vector
#include <utility>    // std::min

#ifdef _MSC_VER
#include <Windows.h>  // defines the min macro when NOMINMAX is defined
#endif

int test6(const std::vector<int>& data)
{
    assert(!data.empty());
    return std::accumulate(
        data.begin() + 1, data.end(), data[0], [](int acc, int val) {
            using namespace std;   // Enable min to be called as a nonmacro.
            return min(acc, val);  // Note that min may or may not be a macro.
        });
}
```

Wrapping the invocation of `min` in a stateless lambda in the code example above will work irrespective of whether the `min` macro is defined by `Windows.h`. What's more, such wrapping enables proper overload resolution and template argument deduction for an `std::min` function should the `min` macro not be defined.

Being stateless, the closure type of this special form of lambda is eligible for **empty-base optimization** (EBO). For types that need to store a function object, EBO can reduce object size compared to storing a function pointer. For example, the deleter stored by instances of the `std::unique_ptr` class template is eligible for such optimization:

```
#include <memory>  // std::unique_ptr

void del(int* ptr) { /* Do some extra work, then delete. */ }
auto delWrap = [](int* ptr) { del(ptr); };

static_assert(
    sizeof(std::unique_ptr<int>)                    ==     sizeof(void*) &&
    sizeof(std::unique_ptr<int, decltype(&del)>)    == 2 * sizeof(void*) &&
    sizeof(std::unique_ptr<int, decltype(delWrap)>) ==     sizeof(void*), "");
```

Using the `del` function's type as the deleter for `std::unique_ptr` doubles the object size in contrast to using the default deleter. When the function is wrapped into a stateless lambda, however, the compiler is able to use EBO to avoid increasing the object size. Note that the compiler is, again, likely to inline calls to `delWrap` but not `del`.

Potential Pitfalls

Dangling references

Closure objects can capture references to local variables and copies of the **this** pointer. If a copy of the closure object outlives the stack frame in which it was created, these references