Overuse

The ability to write functions, especially functions with state, at the point where they are needed and without much of the syntactic overhead that accompanies normal functions and class methods, can potentially lead to a style of code that uses "lambdas everywhere," losing the abstraction and well-documented interfaces of separate functions. Lambda expressions are not intended for large-scale reuse. Sprinkling lambda expressions throughout the code can result in less well-factored, less maintainable code.

Mixing captured and noncaptured variables

A lambda body can access both automatic-duration local variables that were captured from the enclosing block and static-duration variables that need not and cannot be captured. Variables captured by copy are "frozen" at the point of capture and cannot be changed except by the lambda body (if **`mutable`**), whereas **`static`** variables can be changed independent of the lambda expression. This difference is often useful but can cause confusion when reasoning about a lambda expression:

```
void f1()
{
    static int a;
    int        b;

    a = 5;
    b = 6;

    auto c1 = [b]{ return a + b; };  // OK, b is captured by copy.
    assert(11 == c1());              // OK, a == 5 and b == 6.
    ++b;                             // Increment *primary* b.
    assert(11 == c1());              // OK, captured b did not change.
    ++a;                             // Increment static-duration a.
    assert(11 == c1());              // Fires, a == 6 and captured b == 6
}
```

When the closure object for `c1` is created, the captured `b` value is frozen within the lambda body. Changing the primary `b` has no effect. However, `a` is not captured, nor is it allowed to be. As a result, there is only *one* `a` variable, and modifying that variable outside of the lambda body changes the result of invoking the call operator. In C++14, such lambda-capture expressions can be used to effectively capture a copy of such nonlocal variables if it is desired; see Section 2.2."Lambda Captures" on page 986.