```
class Class1
{
    int d_value;

    void mf1()
    {
        Class1& self = *this;
        auto c1 = [self]{ return self.d_value; };
    }
};
```

In C++14, it is possible to achieve the same effect in a terser manner using a lambda capture expression `[self = *this]` (see Section 2.2."Lambda Captures" on page 986).

Confusing mix of immediate and deferred-execution code

The main selling point of lambda expressions — i.e., the ability to define a function object at the point of use — can sometimes be a liability. The code within a lambda body is typically not executed immediately but is deferred until some other piece of code, e.g., an algorithm, invokes it as a callback. The code that is immediately executed and the code whose invocation is deferred are visually intermixed in a way that could confuse a future maintainer. For example, let's look at a simplified excerpt from an earlier use case, *Use Cases — Use with `std::function`* on page 601.

```
#include <cstdlib>     // std::strtol
#include <functional>  // std::function
#include <string>      // std::string
#include <vector>      // std::vector

using Instruction = std::function<long*(long* sp)>;

std::vector<Instruction> instructionStream;

std::string nextToken();                // Read the next token.
char tokenOp(const std::string& token);  // operator for token

void readInstructions()
{
    std::string token;
    Instruction nextInstr;
    while (!(token = nextToken()).empty())
    {
        switch (tokenOp(token))
        {
            // ... more cases
            case '+':
            {
                // + operation
```