

Section 2.1 C++11

Lambdas

```

public:
    void mf()
    {
        auto c1 = [this]{ ++d_value; }; // Increment this->d_value.
        d_value = 1;
        c1();
        assert(2 == d_value);          // Change to d_value is visible.
    }
};

```

Here, we captured **this** in **c1** but then proceeded to modify the object pointed to by **this** within the lambda body.⁵

A lambda expression can occur wherever other expressions can occur, including within other lambda expressions. The set of entities that can be captured in a valid lambda expression depends on the surrounding scope. A lambda expression that does not occur immediately within block scope cannot have a lambda capture:

```

namespace ns1
{
    int v = 10;
    int w = [v]{ /*...*/ }();
    // Error, capture in global/namespace scope.

    void f5(int a = [v]{ return v; }()); // Error, capture in default argument.
}

```

When a lambda expression occurs in block scope, it can capture any local variables with *automatic* (i.e., nonstatic) storage duration in its **reaching scope**. The Standard defines the **reaching scope** of the lambda expression as the set of enclosing scopes up to and including the innermost enclosing function and its **parameters**. Static variables can be used without capturing them; see *Lambda body* on page 595:

```

void f6(int a)
{
    int b = 2 * a;
    if (a)
    {
        int c;
        // ...
    }
    else
    {
        int d = 4 * a;
        static int e = 10;
    }
}

```

⁵In C++17, it is possible to capture ***this**, which results in the entire class object being copied, not just the **this** pointer; for an example of why capturing ***this** might be useful, see *Annoyances — Can't capture *this by copy* on page 611.

Lambdas

Chapter 2 Conditionally Safe Features

```

    auto c1 = [a]{ /*...*/ }; // OK, capture argument a from f6.
    auto c2 = [=]{ return b; }; // OK, implicitly capture local b.
    auto c3 = [&c]{ /*...*/ }; // Error, c is not in reaching scope.
    auto c4 = [&]{ d += 2; }; // OK, implicitly capture local d.
    auto c5 = [e]{ /*...*/ }; // Error, e has static duration.
}

struct LocalClass
{
    void mf()
    {
        auto c6 = [b]{ /*...*/ }; // Error, b is not in reaching scope.
    }
};
}

```

The reaching scope of the lambda expressions for `c1` through `c5`, above, includes the local variable `d` in the **else** block, `b` in the surrounding function block, and `a` from `f6`'s arguments. The local variable, `c`, is not in their reaching scope and cannot be captured. Although `e` is in their reaching scope, it cannot be captured because it does not have automatic storage duration. Finally, the lambda expression for `c6` is within a member function of a **local class**. Its reaching scope ends with the innermost function, `LocalClass::mf`, and does not encompass the surrounding block that includes `a` and `b`.

Only when the innermost enclosing function is a nonstatic class member function can **this** be captured:

```

void f7()
{
    auto c1 = [this]{ /*...*/ }; // Error, f7 is not a member function.
}

class Class6
{
    static void sf1()
    {
        auto c2 = [this]{ /*...*/ }; // Error, sf1 is static.
    }

    void mf()
    {
        auto c3 = [this]{ /*...*/ }; // OK, mf is nonstatic member function.

        struct LocalClass
        {
            static void sf2()
            {

```