The `builder` object is an *lvalue* and is intended to be modified several times before yielding a built-up `ImmutableString` value. After it is modified using `append` and `erase` — selecting the *lvalue* overloads in both cases — attempting to convert it directly to `ImmutableString` fails because there is no such conversion from an *lvalue* `builder`. The initialization of `s2`, conversely, succeeds, *moving* the value from the `StringBuilder` into the result.

The expression `StringBuilder()` constructs an *rvalue*, which is then modified by a chain of calls to `append` and `erase`. The *rvalue* overload of `append` is selected, which returns an rvalue reference that, in turn, drives the selection of the *rvalue* overload of `erase`. Because the result of the chain of modifiers is an rvalue reference, **`operator`** `ImmutableString` can be invoked without calling `std::move`. This usage is safe because the temporary `StringBuilder` object is destroyed immediately afterward, so there is no opportunity for improperly reusing the builder object.

Potential Pitfalls

Forbidding modifications to *rvalues* breaks legitimate use cases

An earlier use case, *Use Cases — Forbidding modifying operations on rvalues* on page 1163, is also the subject of a potential pitfall. Consider a string class with a `toLower` modifier member function:

```
class String
{
public:
    // ...
    String& toLower();
        // Convert all uppercase letters to lowercase, then return modified
        // *this object.
};

String x;    // variable of type String
String f();  // function returning String

void test()
{
    String& a = x.toLower();    // OK, a refers to x.
    f().toLower();              // Defect (1), modifies temporary ~~variable~~; no-op
    String& b = f().toLower();  // Defect (2), b is a dangling reference.
}
```

Defect (1) arises from the statement modifying a temp variable and hence having no effect. Defect (2) results from `toLower` unintentionally acting as an *rvalue*-to-*lvalue* reference cast because it returns an lvalue reference to a possibly *rvalue* object. The lvalue reference, `b`, is bound to the modified temporary `String` returned by `f()`, after it is modi-