BewareStdStringLiterals

The short story: in C++ code using `std::string` never use string literals except to initialize static-scoped `std::string` constants.
(And by the way: NeverUseStaticLocalVariables

The long story: `std::string` is all about avoiding copies. Reference counting and copy-on-write serve to maximise the sharing of a single heap-allocated char array while maintaining memory safety. When used consistently in a program it works rather nicely.

However, when mixed with classic C-style string literals `std::string` can actually *cause* needless heap-allocated copies. Consider these innocent looking constructs:

```
void f(const std::string& s);
void g(const std::string& s = "hello");
std::string h() { return "foo"; }

void copy_surprise {
  std::string x = "x"; // 1
  f("y"); // 2
  g(); // 3
  x = h(); //4
  while (x != "end") { ... } // 4
}
```

Lines 1-4 all cause creation and destruction of an implicit temporary `std::string` to hold the literal value. Line 5 does this for every execution of the while loop. That's a new/memcpy/delete each time. The heap is a heavily used resource, in tight inner loops in multi-threaded code this can be a *severe* contention bottleneck that cripples scalability.

Use static class `std::string` constants or file-private constants instead. You can make global declarations file-private by using a nameless namespace (this is preferred over the use of the `static` keyword.)

```
namespace {
   const std::string end("end");
}
void f() { std::string x; while (x != end) {...} }
```

And once again NeverUseStaticLocalVariables