

# Leveraging D to Mitigate Code Smell

Mark Isaacson

# A Smelly Task

- ODBC
- Documentation
- C interface
- Windows 64-bit
- DLL's

# Madness

```
SQLRETURN SQLNativeSqlW(  
    void* connectionHandle,  
    SQLWCHAR* inSql,  
    SQLINTEGER inSqlLength,  
    SQLWCHAR* outSql,  
    SQLINTEGER outSqlMaxLength,  
    SQLINTEGER* outSqlLength  
);
```

# First Principles

- C is evil
- Abstractions > Conventions
- Explicit + safe > unremarkable + subtle
- All failures must be sane

# Safety First

```
SQLRETURN SQLNativeSqlW(  
    void* connectionHandle,  
    SQLWCHAR* inSql,  
    SQLINTEGER inSqlLength,  
    SQLWCHAR* outSql,  
    SQLINTEGER outSqlMaxLength,  
    SQLINTEGER* outSqlLength  
);
```

```
export SQLRETURN SQLNativeSqlW(  
    OdbcConnection connectionHandle,  
    in SQLWCHAR* _inSql,  
    SQLINTEGER _inSqlLengthChars,  
    SQLWCHAR* _outSql,  
    SQLINTEGER _outSqlMaxLengthBytes,  
    SQLINTEGER* _outSqlLengthBytes) {  
    return exceptionBoundary!(() => {  
        auto inSql = toDString(_inSql, _inSqlLengthChars);  
        auto outSql =  
            outputWChar(_outSql, _outSqlMaxLengthBytes, _outSqlLengthBytes);  
        // ...  
        return SQL_SUCCESS;  
    }());  
}
```

# Mind blown

```
inout(C)[] toDString(C)(inout(C)* cString, size_t lengthChars) if (isSomeChar!C) {  
    return cString[0 .. lengthChars];  
}
```

# With some ODBC smell

```
inout(C)[] toDString(C)(inout(C)* cString, size_t lengthChars) if (isSomeChar!C) {  
    if (cString == null) {  
        return null;  
    }  
    if (lengthChars == SQL_NTS) {  
        lengthChars = strlen(cString);  
    }  
    return cString[0 .. lengthChars];  
}
```


# Abstracting output strings

```
struct OutputWChar {  
    // ...  
    wchar[] buffer;  
    size_t storedLengthChars;  
    alias buffer this;  
}
```

# Allocations/Storage in a DLL

- Must pass data to/from C
- Must keep track of references from C
- Can't rely on GC for cleanup
- Can't rely on OS for cleanup

# Allocations/Storage in a DLL

```
export SQLRETURN SQLNativeSqlW(  
    OdbcConnection connectionHandle,   
    in SQLWCHAR* _inSql,  
    SQLINTEGER _inSqlLengthChars,  
    SQLWCHAR* _outSql,  
    SQLINTEGER _outSqlMaxLengthBytes,  
    SQLINTEGER* _outSqlLengthBytes) {  
    // ...  
}
```

# **Allocations/Storage in a DLL**

A case for with statements

# Allocations/Storage in a DLL

```
auto makeWithoutGC(T, TList...)(auto ref TList args) {  
    import std.c.stdlib : malloc;  
    auto ptr = malloc(getInstanceSize!T);  
    return emplaceWrapper!T(ptr, args);  
}
```

```
auto emplaceWrapper(T, TList...)  
    (void* memory, auto ref TList args) {  
    import std.conv : emplace;  
  
    static if (is(T == class)) {  
        return emplace!T(cast(void[]) memory[0 .. getInstanceSize!T], args);  
    } else {  
        return emplace!T(cast(T*) memory, args);  
    }  
}
```

```
size_t getInstanceSize(T)() if (is(T == class)) {  
    return __traits(classInstanceSize, T);  
}
```

```
size_t getInstanceSize(T)() if (!is(T == class)) {  
    return T.sizeof;  
}
```

```
auto cleanUp(T)(T value) {  
    import std.c.stdlib : free;  
    value.destroy();  
    memset(cast(void*) handle, 0, getInstanceSize!T);  
    free(cast(void*) handle);  
}
```

# Fun Things Along the Way

# DLL Printstrumentation

```
void showPopupMessage(string message) {  
    version (Windows) {  
        import std.c.windows.windows;  
        MessageBoxW(GetForegroundWindow(), message.ptr, "Presto ODBC Driver", MB_OK);  
    }  
}  
  
void debugMessage(string message = "derp", string file = __FILE__, int line = __LINE__) {  
    import core.exception : Exception;  
    auto ex = new Exception(message, file, line);  
    logCriticalMessage(ex);  
    showPopupMessage(ex);  
}
```

# Leveraging Your Optimizer

```
auto dispatchOnVariantType(alias fun, TList...)(Variant value, auto ref TList vs) {  
    auto type = value.type();  
    foreach (T; TypeTuple!(string, short, ushort, int, uint, long, ulong, bool, typeof(null))) {  
        if(type == typeid(T)) {  
            return fun!T(value, vs);  
        }  
    }  
    assert(false, "Unexpected type in variant: " ~ text(value.type()));  
}
```

# Trouble in Paradise

# Stack Traces

# The Unloved: `std.json`

- Didn't compile on 64-bit Windows
- Had slicing but no operator in
- Was [is?] not const-correct

# Too Good For its Own Good

make

# DUB for a GUI library

# Best. GUI. Ever.

```
string getInput(string fileTemplate = "") {  
    import std.process : execute;  
  
    auto tempFile = makeTempFile(fileTemplate);  
    scope (exit) { tempFile.remove; }  
    enforce(!execute(["notepad.exe", tempFile]).status);  
    return tempFile.readText!string();  
}
```

# An Anti-Pattern

```
version(unittest) {  
    char[][] mockCurlResults;  
    void enqueueCurlResult(char[] result) {  
        mockCurlResults ~= result;  
    }  
    char[] get(const(char)[] url, HTTP conn = HTTP()) {  
        assert(!mockCurlResults.empty);  
        auto result = mockCurlResults.front;  
        mockCurlResults.popFront;  
        return result;  
    }  
    char[] post(PostUnit)(const(char)[] url, const(PostUnit)[] postData, HTTP conn = HTTP()) {  
        return get(url);  
    }  
    void del(const(char)[] url, HTTP conn = HTTP()) { /* no-op */ }  
} else {  
    public import std.net.curl : post, get, del;  
}
```

# Proper Mocking



# Proper Mocking

```
struct Curl {  
    char[] get(const(char)[] url, HTTP conn = HTTP()) {  
        return std.net.curl.get(url, conn);  
    }  
    // ...  
}
```

```
struct MockCurl {  
    char[][] mockCurlResults;  
    char[] get(const(char)[] url, HTTP conn = HTTP()) {  
        assert(!mockCurlResults.empty);  
        auto result = mockCurlResults.front;  
        mockCurlResults.popFront;  
        return result;  
    }  
    // ...  
}
```

# Proper Mocking

```
struct Curl {  
    char[] get(const(char)[] url, HTTP conn = HTTP()) {  
        return std.net.curl.get(url, conn);  
    }  
    // ...  
}
```

```
struct MockCurl {  
    char[][] mockCurlResults;  
    char[] get(const(char)[] url, HTTP conn = HTTP()) {  
        assert(!mockCurlResults.empty);  
        auto result = mockCurlResults.front;  
        mockCurlResults.popFront;  
        return result;  
    }  
    // ...  
}
```

```
void printGoole(C)(C curl) {  
    curl.get("http://google.com").writeln;  
}
```

```
unittest {  
    MockCurl curl;  
    curl.mockCurlResults ~= "foobar";  
    printGoogle(curl);  
}
```

```
void main() {  
    Curl curl;  
    printGoogle(curl);  
}
```

# More D in the Wild: Cleaning up C++

# D for Scripting

- Fast dev cycle
- *Tested* reusable components
- ❤️ std.process

# Caching std.process

```
Tuple!("status", int, "output", File) cachedExecute(  
    string[] command,  
    string loadingDescription = null,  
    Duration evictAfter = 5.days  
);
```

# A Simple Pattern

```
string foobar(string input) {  
    import std.functional;  
    static string impl(string input) {  
        return /* ... */;  
    }  
    alias memoized = memoize!impl;  
    return memoized(input);  
}
```

# A Simple Pattern

```
string foobar(string input) {  
    import std.functional;  
    static string impl(string input) {  
        return /* ... */;  
    }  
    alias memoized = memoize!impl;  
    return memoized(input);  
}
```

```
string hgRoot(string filePathInRepo) {  
    import std.functional;  
    static string impl(string filePathInRepo) {  
        alias memoizedExists = memoize!exists;  
        auto query =  
            filePathInRepo  
                .fullyQualify  
                .pathSubsets  
                .find!(a => memoizedExists(subset ~ "/.hg"));  
        return query.empty ? null : query.front;  
    }  
    alias memoized = memoize!impl;  
    return memoized(filePathInRepo);  
}
```

# Programs That Flow

```
#!/usr/bin/rdmd

import std.stdio, std.range, std.algorithm;

void main(string[] args) {
    bool[string] seen;

    bool keepLine(S)(S line) {
        if (line in seen) {
            return false;
        }
        seen[line.idup] = true;
        return true;
    }

    stdin
        .byLine
        .filter!(a => keepLine(a))
        .map!(a => a.writeln)
        .walk;
}
```

# Programs That Flow

```
#!/usr/bin/rdmd

import std.stdio, std.range, std.algorithm;

void main(string[] args) {
    bool[string] seen;

    bool keepLine(S)(S line) {
        if (line in seen) {
            return false;
        }
        seen[line.idup] = true;
        return true;
    }

    stdin
        .byLine
        .filter!(a => keepLine(a))
        .map!(a => a.writeln)
        .walk;
}
```

```
void walk(R)(R range) {
    while (!range.empty) {
        range.front;
        range.popFront;
    }
}
```

# Debugging in This Style

```
void foo() {  
    stdin  
        .filter!(a => !a.find("magic string").empty) //Bug here?  
        .map!(a => a.transmogrify) //Or here?  
        .map!(a => a.writeln)  
        .walk;  
}
```

# Debugging in This Style

```
void foo() {  
    stdin  
        .filter!(a => !a.find("magic string").empty)  
        .debugRange  
        .map!(a => a.transmogrify)  
        .map!(a => a.writeln)  
        .walk;  
}
```

# Debugging in This Style

```
void foo() {  
    stdin  
    .filter!(a => !a.find("magic string").empty)  
    .debugRange  
    .map!(a => a.transmogrify)  
    .map!(a => a.writeln)  
    .walk;  
}
```

```
auto debugRange(R)(R inRange) {  
    static if (isForwardRange!R) {  
        auto range = inRange.save;  
    } else {  
        auto range = inRange.array;  
    }  
    auto copy = range;  
    stderr.writeln("Debugging a range");  
    range  
        .map!(a => stderr.writeln(a))  
        .walk;  
    return copy;  
}
```

# Debugging in This Style

```
void foo() {  
    stdin  
    .filter!(a => !a.find("magic string").empty)  
    .debugRange  
    .map!(a => a.transmogrify)  
    .map!(a => a.writeln)  
    .walk;  
}
```

```
auto debugRange(R)(R inRange) {  
    static if (isForwardRange!R) {  
        auto range = inRange.save;  
    } else {  
        auto range = inRange.array;  
    }  
    auto copy = range;  
    stderr.writeln("Debugging a range");  
    range  
        .map!(a => stderr.writeln(a))  
        .walk;  
    return copy;  
}
```

```
auto identityDebug(T, R)(T value, R message) {  
    stderr.writeln(message);  
    return value;  
}
```

# One Last Complaint

```
import std.stdio;

void main() {
    foreach (i; 0 .. 10) {
        i.writeln;
    }
    [0 .. 10].writeln; //Where are my turtles?
}
```

# Summary

- D makes it easy to build safe abstractions on top of smelly code
- There are some rough edges
- D facilitates some beautiful scripting opportunities

# Fin