cppcon

Modernizing Legacy C++ Code

JAMES MCNELLIS
MICROSOFT VISUAL C++
@JAMESMCNELLIS

KATE GREGORY
GREGORY CONSULTING LIMITED
@GREGCONS

What is legacy code?

```
int _output (
    FILE* stream,
    char const* format,
    va_list arguments
    )
{
    // ...
}
```

```
#ifdef _UNICODE
int _woutput (
#else /* _UNICODE */
int _output (
#endif /* _UNICODE */
    FILE* stream,
    _TCHAR const* format,
    va_list arguments
    )
{
    // ...
}
```

```
#ifdef _UNICODE
#ifdef POSITIONAL_PARAMETERS
int _woutput_p (
#else /* POSITIONAL_PARAMETERS */
int _woutput (
#endif /* POSITIONAL_PARAMETERS */
#else /* _UNICODE */
#ifdef POSITIONAL_PARAMETERS
int _output_p (
#else /* POSITIONAL_PARAMETERS */
int _output (
#endif /* POSITIONAL_PARAMETERS */
#endif /* _UNICODE */
    FILE* stream,
    _TCHAR const* format,
    va_list arguments
    )
{
    // ...
}
```

```
#ifdef _UNICODE
#ifndef FORMAT_VALIDATIONS
#ifdef _SAFECRT_IMPL
int _woutput (
#else  /* _SAFECRT_IMPL */
int _woutput_l (
#endif  /* _SAFECRT_IMPL */
    FILE* stream,
#else  /* FORMAT_VALIDATIONS */
#ifdef POSITIONAL_PARAMETERS
#ifdef _SAFECRT_IMPL
int _woutput_p (
#else  /* _SAFECRT_IMPL */
int _woutput_p_l (
#endif  /* _SAFECRT_IMPL */
    FILE* stream,
#else  /* POSITIONAL_PARAMETERS */
#ifdef _SAFECRT_IMPL
int _woutput_s (
#else  /* _SAFECRT_IMPL */
int _woutput_s_l (
#endif  /* _SAFECRT_IMPL */
    FILE* stream,
#endif  /* POSITIONAL_PARAMETERS */
#endif  /* FORMAT_VALIDATIONS */
#else  /* _UNICODE */
#ifndef FORMAT_VALIDATIONS
#ifdef _SAFECRT_IMPL
int _output (
#else  /* _SAFECRT_IMPL */
int _output_l (
#endif  /* _SAFECRT_IMPL */
    FILE* stream,
#else  /* FORMAT_VALIDATIONS */
#ifdef POSITIONAL_PARAMETERS
#ifdef _SAFECRT_IMPL
int _output_p (
#else  /* _SAFECRT_IMPL */
int _output_p_l (
#endif  /* _SAFECRT_IMPL */
    FILE* stream,
#else  /* POSITIONAL_PARAMETERS */
#ifdef _SAFECRT_IMPL
int _output_s (
#else  /* _SAFECRT_IMPL */
int _output_s_l (
#endif  /* _SAFECRT_IMPL */
    FILE* stream,
#endif  /* POSITIONAL_PARAMETERS */
#endif  /* FORMAT_VALIDATIONS */
#endif  /* _UNICODE */
    _TCHAR const* format,
#ifndef _SAFECRT_IMPL
    _locale_t locale,
#endif  /* _SAFECRT_IMPL */
    va_list arguments
    )
{
    // ...
}
```

```
error4:         /* make sure locidpair is reusable */
                locidpair->stream = NULL;
error3:         /* close pstream (also, clear ph_open[i2] since the stream
                 * close will also close the pipe handle) */
                (void)fclose( pstream );
                ph_open[ i2 ] = 0;
                pstream = NULL;
error2:         /* close handles on pipe (if they are still open) */
                if ( ph_open[i1] )
                        _close( phdls[i1] );
                if ( ph_open[i2] )
                        _close( phdls[i2] );
done:   ;}
        __finally {
            _munlock(_POPEN_LOCK);
        }
error1:
        return pstream;
```

```
IFileDialog *pfd = NULL;
HRESULT hr = CoCreateInstance(CLSID_FileOpenDialog, NULL, CLSCTX_INPROC_SERVER, IID_PPV_ARGS(&pfd));
if (SUCCEEDED(hr)) {
    IFileDialogEvents *pfde = NULL;
    hr = CDialogEventHandler_CreateInstance(IID_PPV_ARGS(&pfde));
    if (SUCCEEDED(hr)) {
        DWORD dwCookie;
        hr = pfd->Advise(pfde, &dwCookie);
        if (SUCCEEDED(hr)) {
            DWORD dwFlags;
            hr = pfd->GetOptions(&dwFlags);
            if (SUCCEEDED(hr)) {
                hr = pfd->SetOptions(dwFlags | FOS_FORCEFILESYSTEM);
                if (SUCCEEDED(hr)) {
                    hr = pfd->SetFileTypes(ARRAYSIZE(c_rgSaveTypes), c_rgSaveTypes);
                    if (SUCCEEDED(hr)) {
                        hr = pfd->SetFileTypeIndex(INDEX_WORDDOC);
                        if (SUCCEEDED(hr)) {
                            hr = pfd->SetDefaultExtension(L"doc;docx");
                            if (SUCCEEDED(hr)) {
```

http://upload.wikimedia.org/wikipedia/en/f/f4/The_Scream.jpg

Legacy C++ Code

…Code that doesn't follow what we'd consider today to be best C++ development practices.

…It's not necessarily "old" code, but it often is.

So what can we do about it?

Increase the warning level; compile as C++

Rid yourself of the preprocessor

Learn to love RAII

Introduce exceptions, but carefuly

Embrace const

Cast properly (and rarely!)

Use the right loop—or avoid loops where possible

```
template<class... _Types1,
    class _Kx_arg,
    size_t... _Ix,
    size_t _Ix_next,
    class... _Types2,
    class... _Rest>
    struct _Tuple_cat2<tuple<_Types1...>, _Kx_arg, _Arg_idx<_Ix...>, _Ix_next,
        tuple<_Types2...>, _Rest...>
        : _Tuple_cat2<
            tuple<_Types1..., _Types2...>,
            typename _Cat_arg_idx<_Kx_arg,
                typename _Make_arg_idx<_Types2...>::type>::type,
            _Arg_idx<_Ix..., _Repeat_for<_Ix_next, _Types2>::value...>,
            _Ix_next + 1,
            _Rest...>
    {// determine tuple_cat's return type and _Kx/_Ix indices
    };
```

cppcon

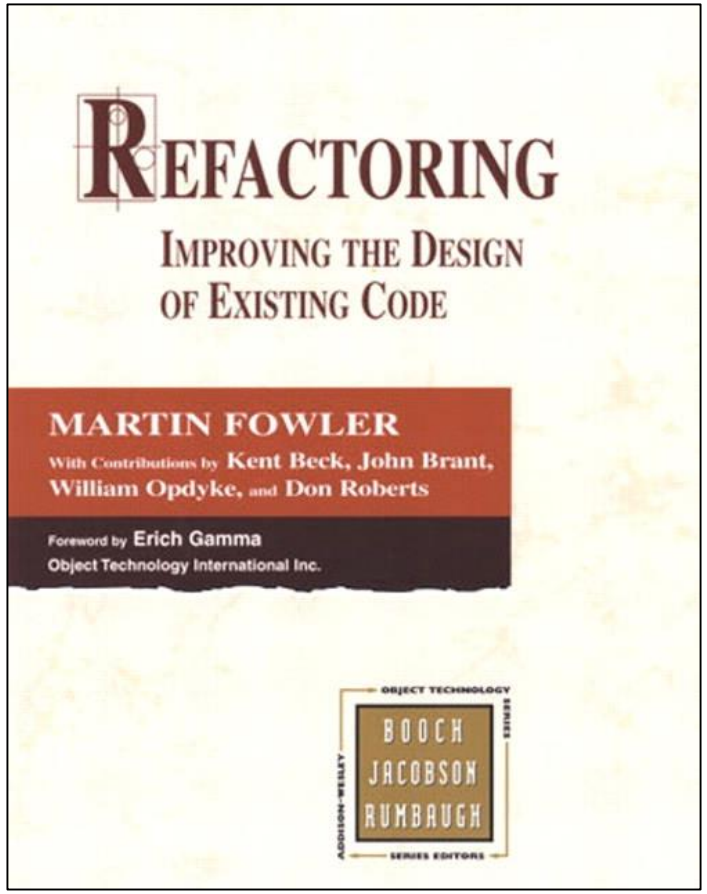
You may also want to attend…

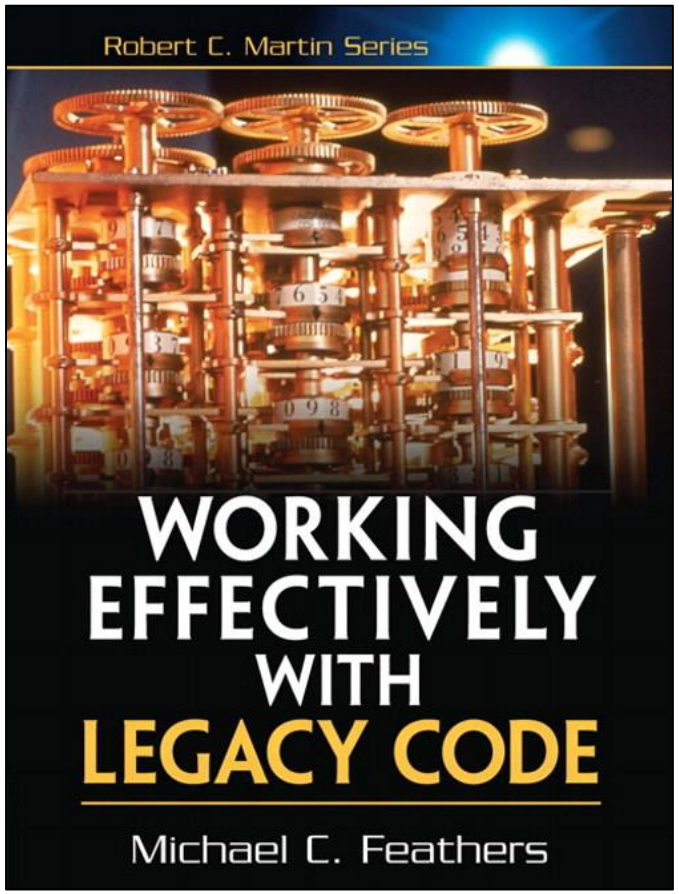
Making C++ Code Beautiful

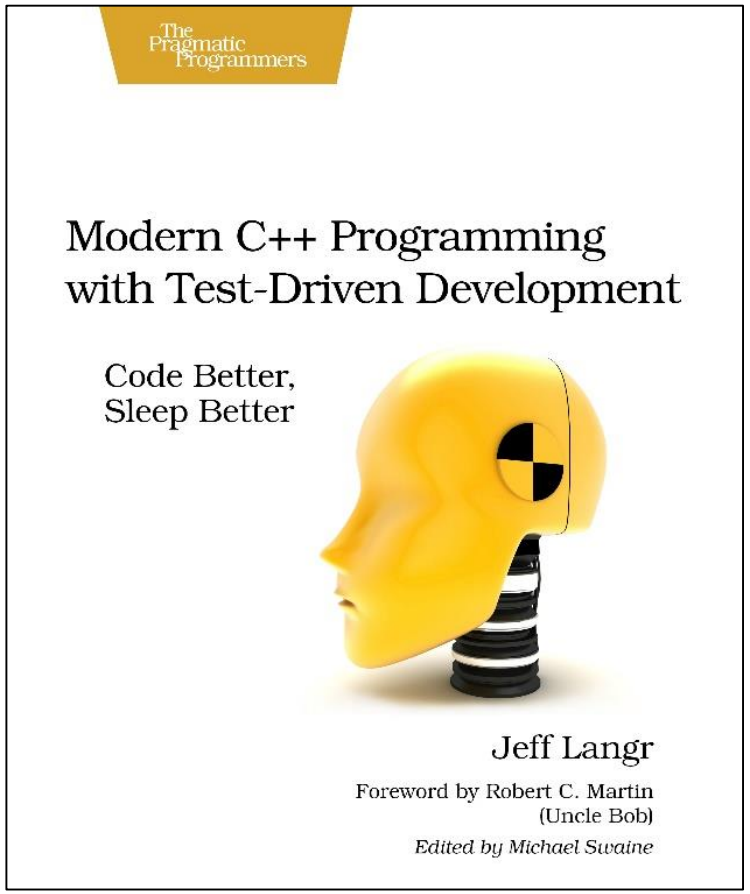
C++ Test-driven Development: Unit Testing, Code Assistance, and Refactoring

Pragmatic Unit Testing in C++

Resources


REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With Contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International Inc.
OBJECT TECHNOLOGY SERIES
BOOCH
JACOBSON
RUMBAUGH
ADDISON-WESLEY
SERIES EDITORS


Robert C. Martin Series
WORKING EFFECTIVELY WITH LEGACY CODE
Michael C. Feathers


The Pragmatic Programmers
Modern C++ Programming with Test-Driven Development
Code Better, Sleep Better
Jeff Langr
Foreword by Robert C. Martin (Uncle Bob)
Edited by Michael Swaine

If you do nothing else…

Turn up the Warning Level

For Visual C++, use /W4 (not /Wall)

Lots of usually-bad things are valid C++

The compiler really wants to help you

Enabling warnings can be done incrementally

```
int f();

int g()
{
    int status = 0;

    if ((status = f()) != 0)
        goto fail;
        goto fail;

    if ((status = g()) != 0)
        goto fail;

    return 0;

fail:
    return status;
}
```

```
int f();

int g()
{
    int status = 0;

    if ((status = f()) != 0)
        goto fail;
        goto fail;

    if ((status = f()) != 0)  // warning C4702: unreachable code
        goto fail;

    return 0;                 // warning C4702: unreachable code

fail:
    return status;
}
```

```
int f();

int g()
{
    int status = 0;

    if ((status = f()) != 0)
        goto fail;
        goto fail;

    if ((status = f()) != 0)  // warning: will never be executed [-Wunreachable-code]
        goto fail;

    return 0;                 // warning: will never be executed [-Wunreachable-code]

fail:
    return status;
}
```

```
int x = 3;
if (x = 2)
{
    std::cout << "x is 2" << std::endl;
}
```

```
int x = 3;
if (2 = x)
{
    std::cout << "x is 2" << std::endl;
}
```

```
int x = 3;
if (x = 2) // warning C4706: assignment within conditional expression
{
    std::cout << "x is 2" << std::endl;
}
```

Compile as C++

If you have C code, convert it to C++

…you will get better type checking

…you can take advantage of more “advanced” C++ features

Conversion process is usually quite straightforward

```
void fill_array(unsigned int* array, unsigned int count)
{
    for (unsigned int i = 0; i != count; ++i)
        array[i] = i;
}

int main()
{
    double data[10];
    fill_array(data, 10); // Valid C!
}
```

cppcon

The Terrible, Horrible, No Good, Very Bad Preprocessor

Conditional Compilation

Every #if, #ifdef, #ifndef, #elif, and #else…

…increases complexity

…makes code harder to understand and maintain

```
#ifdef _UNICODE
int _woutput(
#else /* _UNICODE */
int _output(
#endif /* _UNICODE */
    FILE*          stream,
    _TCHAR const* format,
    va_list        arguments
    )
{
    // ...
}
```

```
template <typename Character>
static int common_output(
    FILE*               stream,
    Character const*    format,
    va_list             arguments
    )
{
    // ...
}

int _output(FILE* stream, char const* format, va_list const arguments)
{
    return common_output(stream, format, arguments);
}

int _woutput(FILE* stream, wchar_t const* format, va_list const arguments)
{
    return common_output(stream, format, arguments);
}
```

Sometimes #ifdefs are okay…

Sometimes conditional compilation makes sense…

- 32-bit vs. 64-bit code
- _DEBUG vs non-_DEBUG (or NDEBUG) code
- Code for different compilers or target platforms
- Code for different languages (e.g. C vs. C++ using __cplusplus)

…but try to keep code within regions simple

- Try to avoid nesting #ifdefs (and refactor to reduce nesting)
- #ifdef entire functions, if possible, rather than just parts of functions

Macros

Macros pose numerous problems for maintainability...

…they don't obey the usual name lookup rules

…they are evaluated before the compiler has type information

Macros are used far more frequently than necessary

RAII and Scope Reduction

```
IFileDialog *pfd = NULL;
HRESULT hr = CoCreateInstance(CLSID_FileOpenDialog, NULL, IID_PPV_ARGS(&pfd));
if (SUCCEEDED(hr)) {
    IFileDialogEvents *pfde = NULL;
    hr = CDialogEventHandler_CreateInstance(IID_PPV_ARGS(&pfde));
    if (SUCCEEDED(hr)) {
        DWORD dwCookie;
        hr = pfd->Advise(pfde, &dwCookie);
        if (SUCCEEDED(hr)) {
            DWORD dwFlags;
            hr = pfd->GetOptions(&dwFlags);
            if (SUCCEEDED(hr)) {
                hr = pfd->SetOptions(dwFlags | FOS_FORCEFILESYSTEM);
                if (SUCCEEDED(hr)) {
                    hr = pfd->SetFileTypes(ARRAYSIZE(c_rgSaveTypes), c_rgSaveTypes);
                    if (SUCCEEDED(hr)) {
                        hr = pfd->SetFileTypeIndex(INDEX_WORDDOC);
                        if (SUCCEEDED(hr)) {
                            hr = pfd->SetDefaultExtension(L"doc;docx");
                            if (SUCCEEDED(hr)) {
```

```
IFileDialog *pfd = NULL;
HRESULT hr = CoCreateInstance(CLSID_FileOpenDialog, NULL, IID_PPV_ARGS(&pfd));
if (FAILED(hr))
    return hr;

IFileDialogEvents *pfde = NULL;
hr = CDialogEventHandler_CreateInstance(IID_PPV_ARGS(&pfde));
if (FAILED(hr))
    return hr;

DWORD dwCookie;
hr = pfd->Advise(pfde, &dwCookie);
if (FAILED(hr))
    return hr;

DWORD dwFlags;
hr = pfd->GetOptions(&dwFlags);
if (FAILED(hr))
    return hr;
```

```
                                    }
                                    psiResult->Release();
                                }
                            }
                        }
                    }
                }
            }
            pfd->Unadvise(dwCookie);
        }
        pfde->Release();
    }
    pfd->Release();
}

return hr;
```

```
void f(size_t const buffer_size)
{
    void* buffer1 = malloc(buffer_size);
    if (buffer1 != NULL)
    {
        void* buffer2 = malloc(buffer_size);
        if (buffer2 != NULL)
        {
            // ...code that uses the buffers...

            free(buffer2);
        }

        free(buffer1);
    }
}
```

```
void f(size_t const buffer_size)
{
    raii_container buffer1(malloc(buffer_size));
    if (buffer1 = nullptr)
        return;

    raii_container buffer2(malloc(buffer_size));
    if (buffer2 == nullptr)
        return;

    // ...code that uses the buffers...
}
```

```
void f(size_t const buffer_size)
{
    std::vector<char> buffer1(buffer_size);
    std::vector<char> buffer2(buffer_size);

    // ...code that uses the buffers...
}
```

```
void f(size_t const buffer_size)
{
    std::unique_ptr<char[]> buffer1(new (std::nothrow) char[buffer_size]);
    if (buffer1 == nullptr)
        return;

    std::unique_ptr<char[]> buffer2(new (std::nothrow) char[buffer_size]);
    if (buffer2 == nullptr)
        return;

    // ...code that uses the buffers...
}
```

```
struct free_delete
{
    void operator()(void* p) const { free(p); }
};

void f(size_t const buffer_size)
{
    std::unique_ptr<void, free_delete> buffer1(malloc(buffer_size));
    if (buffer1 == nullptr)
        return;

    std::unique_ptr<void, free_delete> buffer2(malloc(buffer_size));
    if (buffer2 == nullptr)
        return;

    // ...code that uses the buffers...
}
```

```
HRESULT BasicFileOpen()
{
    IFileDialog *pfd = NULL;
    HRESULT hr = CoCreateInstance(
        CLSID_FileOpenDialog,
        NULL,
        CLSCTX_INPROC_SERVER,
        IID_PPV_ARGS(&pfd));

    if (SUCCEEDED(hr))
    {
        pfd->Release();
    }

    return hr;
}
```

```
struct iunknown_delete
{
    void operator()(IUnknown* p)
    {
        if (p) { p->Release(); }
    }
};

HRESULT BasicFileOpen()
{
    std::unique_ptr<IFileDialog, iunknown_delete> pfd;
    HRESULT hr = CoCreateInstance(
        CLSID_FileOpenDialog,
        NULL,
        CLSCTX_INPROC_SERVER,
        IID_PPV_ARGS(&pfd));

    return hr;
}
```

```
HRESULT BasicFileOpen()
{
    ComPtr<IFileDialog> pfd;
    HRESULT hr = CoCreateInstance(
        CLSID_FileOpenDialog,
        NULL,
        CLSCTX_INPROC_SERVER,
        IID_PPV_ARGS(&pfd));

    return hr;
}
```

Keep Functions Linear

Functions that have mostly linear flow are…

…easier to understand

…easier to modify during maintenance and bug fixing

Use RAII Everywhere

Code that uses RAII is easier to read, write, and maintain

- RAII frees you from having to worry about resource management
- Functions that use RAII can freely return at any time
- Single-exit and goto-based cleanup should never be used anymore

RAII is an essential prerequisite for introducing exceptions

- But even if you never plan to use exceptions, still use RAII

This is the single easiest way to improve C++ code quality

Introducing Exceptions

```
void f(size_t const buffer_size)
{
    std::vector<char> buffer1(buffer_size);
    std::vector<char> buffer2(buffer_size);

    // ...code that uses the buffers...
}
```

Introducing Exceptions

Exceptions are the preferred method of runtime error handling

- Used by most modern C++ libraries (including the STL)
- Often we'd like to start using those modern libraries in legacy codebases

Use of well-defined *exception boundaries* enables introduction of throwing code into legacy codebases that don't use exceptions.

```
extern "C" HRESULT boundary_function()
{
    // ... code that may throw ...
    return S_OK;
}
```

```
extern "C" HRESULT boundary_function()
{
    try
    {
        // ... code that may throw ...
        return S_OK;
    }
    catch (...)
    {
        return E_FAIL;
    }
}
```

```
extern "C" HRESULT boundary_function()
{
    try
    {
        // ... code that may throw ...
        return S_OK;
    }
    catch (my_hresult_error const& ex) { return ex.hresult();  }
    catch (std::bad_alloc const&)      { return E_OUTOFMEMORY; }
    catch (...)                        { std::terminate();     }
}
```

```
#define TRANSLATE_EXCEPTIONS_AT_BOUNDARY                                  \
    catch (my_hresult_error const& ex) { return ex.hresult();  } \
    catch (std::bad_alloc const&)      { return E_OUTOFMEMORY; } \
    catch (...)                        { std::terminate();     }

extern "C" HRESULT boundary_function()
{
    try
    {
        // ... code that may throw ...
        return S_OK;
    }
    TRANSLATE_EXCEPTIONS_AT_BOUNDARY
}
```

```
inline HRESULT translate_thrown_exception_to_hresult()
{
    try { throw; }
    catch (my_hresult_error const& ex) { return ex.hresult();  }
    catch (std::bad_alloc const&)      { return E_OUTOFMEMORY; }
    catch (...)                        { std::terminate();     }
}

extern "C" HRESULT boundary_function()
{
    try
    {
        // ... code that may throw ...
        return S_OK;
    }
    catch (...) { return translate_thrown_exception_to_hresult(); }
}
```

```
template <typename Callable>
HRESULT call_and_translate_for_boundary(Callable&& f)
{
    try
    {
        f(); return S_OK;
    }
    catch (my_hresult_error const& ex) { return ex.hresult();  }
    catch (std::bad_alloc const&)      { return E_OUTOFMEMORY; }
    catch (...)                        { std::terminate();     }
}

extern "C" HRESULT boundary_function()
{
    return call_and_translate_for_boundary([&]
    {
        // ... code that may throw ...
    });
}
```

cppcon

The Glorious **const** Keyword

Const Correctness

The bare minimum; all APIs should be const correct

If you’re not modifying an object via a pointer or reference, make that pointer or reference const

Member functions that don’t mutate an object should be const

Const-Qualify Everything

If you write const-correct APIs, that is a good start…

…but you're missing out on a lot of benefits of 'const'

```
bool read_byte(unsigned char* result);

bool read_elements(
    void*  buffer,
    size_t element_size,
    size_t element_count)
{
    size_t buffer_size = element_size * element_count;

    unsigned char* first = static_cast<unsigned char*>(buffer);
    unsigned char* last  = first + buffer_size;
    for (unsigned char* it = first; it != last; ++it)
    {
        if (!read_byte(it))
            return false;
    }

    return true;
}
```

```
bool read_byte(unsigned char* result);

bool read_elements(
    void*  const buffer,
    size_t const element_size,
    size_t const element_count)
{
    size_t buffer_size = element_size * element_count;

    unsigned char* first = static_cast<unsigned char*>(buffer);
    unsigned char* last  = first + buffer_size;
    for (unsigned char* it = first; it != last; ++it)
    {
        if (!read_byte(it))
            return false;
    }

    return true;
}
```

```
bool read_byte(unsigned char* result);

bool read_elements(
    void*  const buffer,
    size_t const element_size,
    size_t const element_count)
{
    size_t const buffer_size = element_size * element_count;

    unsigned char* first = static_cast<unsigned char*>(buffer);
    unsigned char* last  = first + buffer_size;
    for (unsigned char* it = first; it != last; ++it)
    {
        if (!read_byte(it))
            return false;
    }

    return true;
}
```

```
bool read_byte(unsigned char* result);

bool read_elements(
    void*  const buffer,
    size_t const element_size,
    size_t const element_count)
{
    size_t const buffer_size = element_size * element_count;

    unsigned char* const first = static_cast<unsigned char*>(buffer);
    unsigned char* const last  = first + buffer_size;
    for (unsigned char* it = first; it != last; ++it)
    {
        if (!read_byte(it))
            return false;
    }

    return true;
}
```

Two Recommendations

Const-qualify (almost) everything that can be const-qualified.

Where possible, refactor code to enable more things to be const-qualified

What shouldn’t be const?

Data members (member variables)

By-value return types

Class-type local variables that may be moved from

Class-type local variables that may be returned

C-Style Casts

```
ClassType* ctp = (ClassType*)p;
```

What does a C cast do?

const_cast

static_cast

static_cast + const_cast

reinterpret_cast

reinterpret_cast + const_cast

```
ClassType* ctp = (ClassType*)p;
```

What does this cast do?

If p is a ClassType const*, it does a const_cast

If p is of a type related to ClassType, it does a static_cast

- Possibly combined with a const_cast, if required

If p is of a type unrelated to ClassType, it does a reinterpret_cast

- Possibly combined with a const_cast, if required

```
struct A;
struct B;

void f(B* p)
{
    A* ctp = (A*)p;
}
```

Eliminate usage of C casts

Absolutely, positively do not use C casts for pointer conversions

Avoid usage of C casts everywhere else too…

…Including for numeric conversions (e.g., double => int)

Transforming Loops

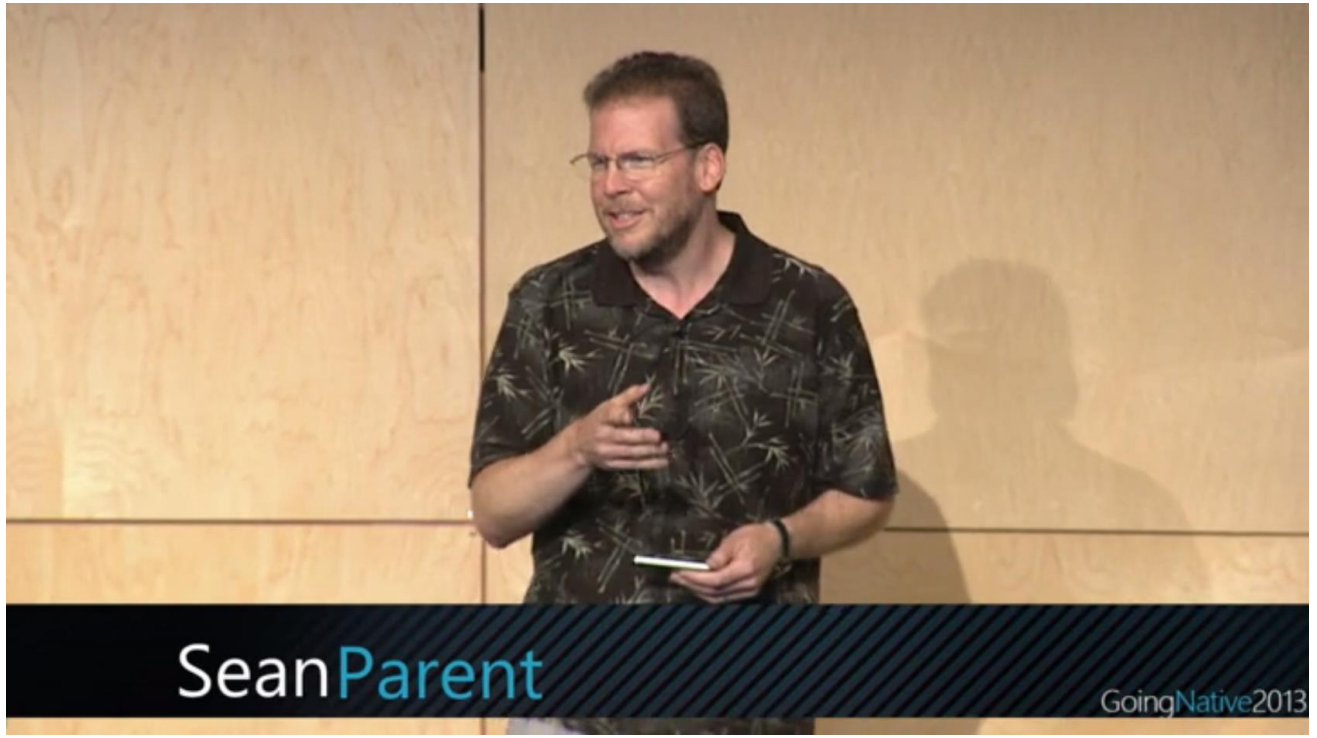
```cpp
int main()
{
    std::vector<int> v = { 1, 2, 3, 4, 1000, 5, 6, 7, 8 };

    // Find the maximum value:
    int max_value(INT_MIN);

    for (size_t i(0); i != v.size(); ++i)
    {
        if (v[i] > max_value)
            max_value = v[i];
    }

    std::cout << "Maximum:  " << max_value << "\n";
}
```

```cpp
int main()
{
    std::vector<int> v = { 1, 2, 3, 4, 1000, 5, 6, 7, 8 };

    // Find the maximum value:
    int max_value(INT_MIN);

    for (auto it(v.begin()); it != v.end(); ++it)
    {
        if (*it > max_value)
            max_value = *it;
    }

    std::cout << "Maximum:  " << max_value << "\n";
}
```

```cpp
int main()
{
    std::vector<int> v = { 1, 2, 3, 4, 1000, 5, 6, 7, 8 };

    // Find the maximum value:
    int max_value(INT_MIN);

    for (auto&& x : v)
    {
        if (x > max_value)
            max_value = x;
    }

    std::cout << "Maximum:  " << max_value << "\n";
}
```

```cpp
int main()
{
    std::vector<int> v = { 1, 2, 3, 4, 1000, 5 };

    auto max_value_it = std::max_element(v.begin(), v.end());

    std::cout << "Maximum:  " << *max_value_it << "\n";
}
```

C++ Seasoning



http://channel9.msdn.com/Events/GoingNative/2013/Cpp-Seasoning

A Real-World Example

There's Just One Problem...

We've gone and replaced really fast C code with C++ code.

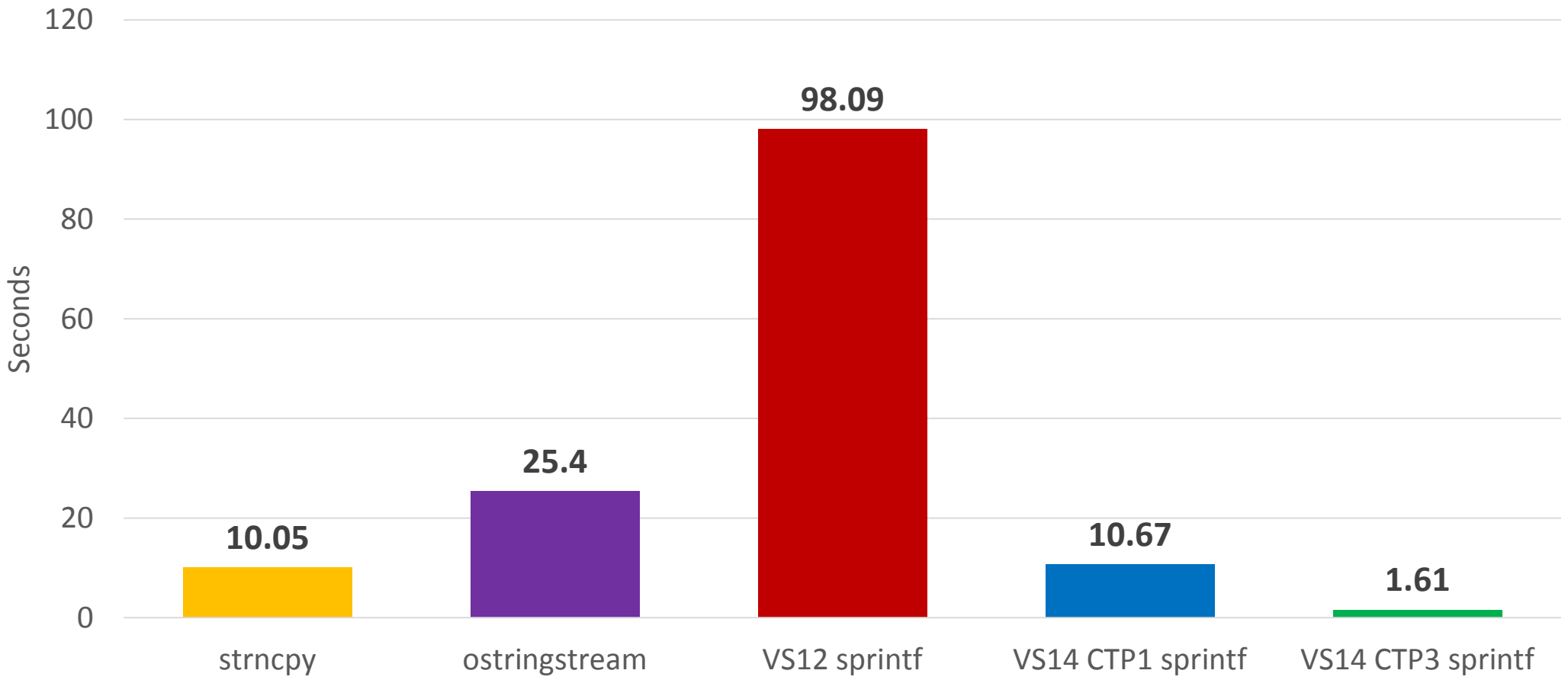
Everyone knows C++ is slower than C.

…Right?

…Right?

LOL

Visual C++ sprintf %s Performance



Recap

Recommendations

Eliminate complexity introduced by the preprocessor

Refactor functions to linearize and shorten them

Update any manual resource management to use RAII

Litter your code with the const qualifier

Convert C casts to C++ casts

Use algorithms instead of loops


\0