cppcon

Making C++ Code Beautiful

JAMES MCNELLIS
MICROSOFT VISUAL C++
@JAMESMCNELLIS

KATE GREGORY
GREGORY CONSULTING LIMITED
@GREGCONS

C++ gets praise…

Powerful

Fast

Flexible

Language for smart people

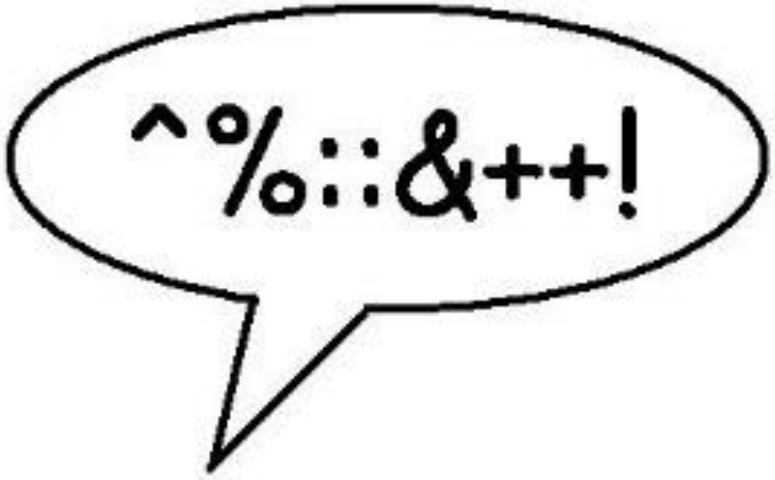
…but C++ also gets criticism

Ugly

Complicated

Hard to read

Language for smart people



This is not C++

```
void f(char [illegible]* const file[illegible]) {
    void* buf[illegible] = malloc(1[illegible] 1024);
    if (buffer[illegible] [illegible]ullptr)
        FILE* fi[illegible] fope[illegible]e_name, "r");
        if (file ![illegible] {
            // ...Re[illegible] into the buffer, or whatever...
            fclose([illegible]
        }
        free(buf[illegible]);
    }
}
```

The Basics (from Monday)

Compile at a high warning level

Stop writing C and try to port existing C code to C++

Avoid #ifdefs wherever possible; when they are necessary, keep them simple

Use RAII everywhere, even in the absence of exceptions

Keep functions linear and don’t write arrow code

Const-qualify everything (or, as much as possible ☺)

Don’t use C casts—eliminate as many casts as possible and use C++ casts where necessary

Agenda

Macros are ugly

Walls of code are ugly

Lambdas are beautiful

Invisible code is beautiful

Removing effort is beautiful

Other people’s code is beautiful

Comments are ugly

Macros are ugly

Object-Like Macros

Often used for values

No encapsulation

No type (literal text)

Enumerators or static const variables can be used for most constants

- …except in headers that may need to be #includable by C code

Function-Like Macros

Used for function-like things

Horrible expansion issues and side effects

Use a function for functions

- Work on many types? Template function
- Wrap around expression or piece of code? Take a lambda

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

```
#define red      0
#define orange   1
#define yellow   2
#define green    3
#define blue     4
#define purple   5
#define hot_pink 6

void f()
{
    unsigned orange = 0xff9900;
}
```

```
warning C4091: '' : ignored on left of 'unsigned int' when no variable is declared
error C2143: syntax error : missing ';' before 'constant'
error C2106: '=' : left operand must be l-value
```

```
#define red      0
#define orange   1
#define yellow   2
#define green    3
#define blue     4
#define purple   5
#define hot_pink 6

void f()
{
    unsigned 2 = 0xff00ff;
}
```

```
warning C4091: '' : ignored on left of 'unsigned int' when no variable is declared
error C2143: syntax error : missing ';' before 'constant'
error C2106: '=' : left operand must be l-value
```

```
#define RED      0
#define ORANGE   1
#define YELLOW   2
#define GREEN    3
#define BLUE     4
#define PURPLE   5
#define HOT_PINK 6

void g(int color);
void f()
{
    g(HOT_PINK); // Ok
    g(9000);     // Not ok, but compiler can’t tell
}
```

```
enum color_type
{
    red      = 0,
    orange   = 1,
    yellow   = 2,
    green    = 3,
    blue     = 4,
    purple   = 5,
    hot_pink = 6
};
```

```
enum color_type
{
    red, orange, yellow, green, blue, purple, hot_pink
};

void g(color_type color);
void f()
{
    g(hot_pink); // Ok
    g(9000);     // Not ok, compiler will report error
}
```

```
error C2664: 'void g(color_type)' : cannot convert argument 1 from 'int' to 'color_type'
```

```
enum color_type
{
    red, orange, yellow, green, blue, purple, hot_pink
};

void f()
{
    int x = red;           // Ugh
    int x = red + orange; // Double ugh
}
```

```
enum color_type
{
    red, orange, yellow, green, blue, purple, hot_pink
};

enum traffic_light_state
{
    red, yellow, green
};
```

```
error C2365: 'red' : redefinition; previous definition was 'enumerator‘
error C2365: 'yellow' : redefinition; previous definition was 'enumerator‘
error C2365: 'green' : redefinition; previous definition was 'enumerator'
```

```
enum class color_type
{
    red, orange, yellow, green, blue, purple, hot_pink
};

void g(color_type color);
void f()
{
    g(color_type::red);
}
```

```
enum class color_type
{
    red, orange, yellow, green, blue, purple, hot_pink
};

void g(color_type color);
void f()
{
    int x = color_type::hot_pink;
}
```

error C2440: 'initializing' : cannot convert from 'color_type' to 'int'

```
enum class color_type
{
    red, orange, yellow, green, blue, purple, hot_pink
};

void g(color_type color);
void f()
{
    int x = static_cast<int>(color_type::hot_pink);
}
```

```
enum : uint32_t
{
    max_uint32_t = static_cast<uint32_t>(-1)
};
```

Named Constants

Prefer enumerators for named constants

- And prefer scoped enumerations where possible

It's also common to see 'static const' variables used

- But an enumerator with an underlying type is often preferable

When interop with C is necessary, macros may also be necessary

Macros that should be Functions

```
#define make_char_lowercase(c) \
    ((c) = (((c) >= 'A') && ((c) <= 'Z')) ? ((c) - 'A' + 'a') : (c))

void make_string_lowercase(char* s)
{
    while (make_char_lowercase(*s++))
        ;
}
```

```
#define make_char_lowercase(c) \
    ((c) = (((c) >= 'A') && ((c) <= 'Z')) ? ((c) - 'A' + 'a') : (c))

void make_string_lowercase(char* s)
{
    while (((*s++) = (((*s++) >= 'A') && ((*s++) <= 'Z'))
              ? ((*s++) - 'A' + 'a') : (*s++)))
        ;
}
```

```
// Old, ugly macro implementation:
#define make_char_lowercase(c) \
    ((c) = (((c) >= 'A') && ((c) <= 'Z')) ? ((c) - 'A' + 'a') : (c))


// New, better function implementation:
inline char make_char_lowercase(char& c)
{
    if (c > 'A' && c < 'Z')
    {
        c = c - 'A' + 'a';
    }

    return c;
}
```

Replace function-like macros with functions

Functions are…

- Easier to read and maintain
- Easier to write (you aren't restricted to a single expression)
- Easier to debug through
- Behave better according to programmer expectations

…and there is generally no performance overhead.

A wall of code is ugly

Wall of Code

Split into functions

- Argument taking rules have changed
- Avoid premature optimization
- Bonus: functions have names, arguments have names

Is it hard because of the types you're using?

- E.g. char* vs std::string
- Ten "related" locals should perhaps be a class?

Look for chances to use standard algorithms

- Less code to explain, test, maintain, optimize

Lambdas are beautiful

Okay, maybe not so beautiful to look at…

[ ] ( ) { }

...but they are so useful!

```
auto vglambda = [](auto printer) {
    return [=](auto&& ... ts) {
        printer(std::forward<decltype(ts)>(ts)...);
        return [=]() {
            printer(ts ...);
        };
    };
};
```

```
std::vector<int> const v = { 1, 2, 3, 4, 5 };

std::for_each(begin(v), end(v), [](int const n)
{
    std::cout << n << '\n';
});
```

```
std::thread t([]{ std::cout << "Hello, CppCon\n"; });
```

```
CreateThread(
    nullptr,
    0,
    [](void*) { std::cout << "Hello, CppCon people\n"; return 0ul; },
    nullptr,
    0,
    nullptr);
```

```
extern "C" errno_t my_amazing_c_function()
{
    return translate_exceptions([&]
    {
        // ... code that may throw ...
    });
}
```

```
database const* target_scope(nullptr);

switch (resolution_scope.table())
{
case table_id::module:
    target_scope = &module.database();
    break;

case table_id::module_ref:
    target_scope = &resolve_module_ref(resolution_scope.as<module_ref_token>());
    break;

case table_id::assembly_ref:
    target_scope = is_windows_runtime_assembly_ref(assembly_ref_scope)
        ? &resolve_namespace(usable_namespace)
        : &resolve_assembly_ref(assembly_ref_scope);
    break;

default:
    assert_unreachable();
}
```

```
database const* target_scope(nullptr);

switch (resolution_scope.table())
{
case table_id::module:
    target_scope = &module.database();
    break;

case table_id::module_ref:
    target_scope = &resolve_module_ref(resolution_scope.as<module_ref_token>());
    break;

case table_id::assembly_ref:
    target_scope = is_windows_runtime_assembly_ref(assembly_ref_scope)
        ? &resolve_namespace(usable_namespace)
        : &resolve_assembly_ref(assembly_ref_scope);
    break;

default:
    assert_unreachable();
}
```

```
database const* target_scope(nullptr);

switch (resolution_scope.table())
{
case table_id::module:
    target_scope = &module.database();
    break;

case table_id::module_ref:
    target_scope = &resolve_module_ref(resolution_scope.as<module_ref_token>());
    break;

case table_id::assembly_ref:
    target_scope = is_windows_runtime_assembly_ref(assembly_ref_scope)
        ? &resolve_namespace(usable_namespace)
        : &resolve_assembly_ref(assembly_ref_scope);
    break;

default:
    assert_unreachable();
}
```

```
database const& target_scope([&]() -> database const&
{
    switch (resolution_scope.table())
    {
    case table_id::module:
        return module.database();

    case table_id::module_ref:
        return resolve_module_ref(resolution_scope.as<module_ref_token>());

    case table_id::assembly_ref:
        return is_windows_runtime_assembly_ref(assembly_ref_scope)
            ? resolve_namespace(usable_namespace)
            : resolve_assembly_ref(assembly_ref_scope);

    default:
        assert_unreachable();
    }
}());
```

Invisible code is beautiful

Invisible code

What happens when flow of control reaches this line of code:

```
}
```

Sometimes, a lot

Look at your cleanup code

How long are your catch blocks?

… longer than the try?

Do your catch blocks have goto statements?

… more than 3?

What is your code trying to tell you?

```
ReConnect:
    if (iReconnectCnt > 3)
    {
        goto exit_thread_loop;
    }

    do
    {
        while( TRUE )
        {
            if ( iRunState == RS_RUN )
                break;
            if ( iRunState == RS_STOPPING )
                goto exit_thread_loop;
            WaitForSingleObject(hChangeEvent, INFINITE);
        }

    } while( iRunState != RS_RUN );

    try
    {
        OpenConnection(outfile, &connectInfo, pInfo, &txnRec, &pTxn );
    }
```

```
catch(CBaseErr *e)
{
    if ( (e->ErrorType() == ERR_TYPE_SOCKET) &&
         ((e->ErrorNum() == WSAECONNRESET) ||
          (e->ErrorNum() == WSAECONNABORTED) ||
          (e->ErrorNum() == ERR_CONNECTION_CLOSED)) )
    {
        CloseHTMLSocket(&connectInfo.socket);
        iReconnectCnt++;
        goto ReConnect;
    }
    goto exit_thread_loop;
}
catch(...)
{
goto exit_thread_loop;
}
```

```
exit_thread_loop:
if (bNeedToReleaseSemaphore)
{
    ReleaseSemaphore( hTxnConcurrency, 1, NULL );
    bNeedToReleaseSemaphore = FALSE;
}
```

Unexpected benefits

Consistent cleanup

Encapsulation

Important code becomes visible

Removing effort is beautiful

Removing effort

Code that is hard to write is hard to read

Also to maintain, test, and fix

Code that states your intent clearly is easier for everyone

Code that cannot have certain mistakes is easier for everyone

Range-based for

Algorithms

auto--sometimes

```
std::vector<int> const v = { 1, 2, 3, 4, 5 };

for (std::vector<int>::const_iterator it = v.begin(); it != v.end(); ++it)
{
    std::cout << *it << '\n';
}
```

```
std::vector<int> const v = { 1, 2, 3, 4, 5 };

for (auto it = v.begin(); it != v.end(); ++it)
{
    std::cout << *it << '\n';
}
```

```
std::vector<int> const v = { 1, 2, 3, 4, 5 };

for (int const n : v)
{
    std::cout << n << '\n';
}
```

```
std::vector<int> const v = { 1, 2, 3, 4, 5 };

for (auto const n : v)
{
    std::cout << n << '\n';
}
```

```
std::vector<int> const v = { 1, 2, 3, 4, 5 };

for (n : v)
{
    std::cout << n << '\n';
}
```

```
std::vector<int> const v = { 1, 2, 3, 4, 5 };

std::copy(begin(v), end(v), std::ostream_iterator<int>(std::cout, "\n"));
```

```
std::vector<int> const v = { 1, 2, 3, 4, 5 };

std::for_each(begin(v), end(v), [](int const n)
{
    std::cout << n << '\n';
});
```

So many useful algorithms…

for_each

transform

all_of

any_of

find

find_if

binary_search

lower_bound

upper_bound

equal_range

sort

stable_sort

copy_if

unique

partition

remove

rotate

and many more…

Other people's code is beautiful

Other people’s code

Calling a function with a name, declaring an instance of a class with a name

Less for you to write, test, and maintain

Are you really the first person to face this?

Start with std::

There is a whole world out there

Finding libraries

http://en.wikipedia.org/wiki/Category:C%2B%2B_libraries

…113 pages – excellent indication of C++ skills at “naming things”

cppcon

Pages in category "C++ libraries"

The following 113 pages are in this category, out of 113 total. This list may not reflect recent changes (learn more).

A

- Active Template Library
- Adaptive Communication Environment
- Algorithmic skeleton
- ANGLE (software)
- Apache C++ Standard Library
- Armadillo (C++ library)
- Artefaktur
- Asio C++ library
- AT&T FSM Library
- ATL Server

B

- BALL
- Blitz++
- Boehm garbage collector
- Boost (C++ libraries)
- Borland Graphics Interface
- Botan (programming library)
- Brig (C++ libraries)

C

- C++/Tcl
- C++ AMP
- CAPD library
- Cassowary (software)

G cont.

- GiNaC
- Google Test
- Gosu (library)
- GPhoto
- Graphics Environment for Multimedia
- Gtkmm

H

- HOOPS 3D Graphics System

I

- Index64
- Integrated Performance Primitives
- Intel Array Building Blocks
- Iterative Template Library

J

- JUCE

K

- Kakadu (software)
- Kyoto Cabinet

L

- LEMON (C++ library)
- LevelDB

O cont.

- Object-oriented Abstract Type Hierarchy
- ODB (C++)
- OGRE
- Open Asset Import Library
- Open Inventor
- OpenH264
- OpenImageIO
- OpenNN
- Oracle Template Library
- Orfeo toolbox

P

- Pantheios
- Parallel Patterns Library
- PLIB
- POCO C++ Libraries
- Podofo
- Poppler (software)
- PTK Toolkit

Q

- Qt (software)

R

- Rich Booleans

Finding libraries

Search for what you want to do

…Image manipulation?

…Face recognition?

…Work with json? Zip files? Pdf files?

Change your default from “nobody else could possibly meet my standards” to “shipping is a feature and other people might be smart enough too”

Comments are ugly

```
/// <summary>
/// Default constructor
/// </summary>
basic_istream();

/// <summary>
/// Copy constructor
/// </summary>
/// <param name="other">The source object</param>
basic_istream(const basic_istream &other);

/// <summary>
/// Assignment operator
/// </summary>
/// <param name="other">The source object</param>
/// <returns>A reference to the object that contains the result of the assignment.</returns>
basic_istream & operator =(const basic_istream &other);
```

```
struct basic_istream
{
    basic_istream();
    basic_istream(basic_istream const& other);
    basic_istream& operator=(basic_istream const& other);
};
```

```
enum class day_of_week
{
    sunday,   // Sunday
    monday,   // Monday
    tuesday,  // Tuesday
    humpday,  // Humpday
    thursday, // Thursday
    friday,   // Friday
    saturday  // Saturday
};
```

```
enum class day_of_week
{
    sunday,   // The first day of the week (value: 1)
    monday,   // The second day of the week (value: 2)
    tuesday,  // The third day of the week (value: 3)
    humpday,  // The fourth day of the week (value: 4)
    thursday, // The fifth day of the week (value: 5)
    friday,   // The sixth day of the week (value: 6)
    saturday  // The seventh day of the week (value: 7)
};
```

```
enum class day_of_week
{
    sunday,
    monday,
    tuesday,
    humpday,
    thursday,
    friday,
    saturday
};
```

```c
int _read_nolock(
    int fh,
    void *inputbuf,
    unsigned cnt
    )
{
    int bytes_read;                 /* number of bytes read */
    char *buffer;                   /* buffer to read to */
    int os_read;                    /* bytes read on OS call */
    char *p, *q;                    /* pointers into buffer */
    wchar_t *pu, *qu;               /* wchar_t pointers into buffer for UTF16 */
    char peekchr;                   /* peek-ahead character */
    wchar_t wpeekchr;               /* peek-ahead wchar_t */
    __int64 filepos;                /* file position after seek */
    ULONG dosretval;                /* o.s. return value */
    char tmode;                     /* textmode - ANSI/UTF-8/UTF-16 */
    BOOL fromConsole = 0;           /* true when reading from console */
    void *buf;                      /* buffer to read to */
    int retval = -2;                /* return value */

    // ...
```

```c
int _read_nolock(
    int fh,
    void *inputbuf,
    unsigned cnt
    )
{
    int bytes_read;                 /* number of bytes read */
    char *buffer;                   /* buffer to read to */
    int os_read;                    /* bytes read on OS call */
    char *p, *q;                    /* pointers into buffer */
    wchar_t *pu, *qu;               /* wchar_t pointers into buffer for UTF16 */
    char peekchr;                   /* peek-ahead character */
    wchar_t wpeekchr;               /* peek-ahead wchar_t */
    __int64 filepos;                /* file position after seek */
    ULONG dosretval;                /* o.s. return value */
    char tmode;                     /* textmode - ANSI/UTF-8/UTF-16 */
    BOOL fromConsole = 0;           /* true when reading from console */
    void *buf;                      /* buffer to read to */
    int retval = -2;                /* return value */

    // ...
```

```
int _read_nolock(
    int fh,
    void *inputbuf,
    unsigned cnt
    )
{
    int bytes_read;                 /* number of bytes read */
    char *buffer;                   /* buffer to read to */
    int os_read;                    /* bytes read on OS call */
    char *p, *q;                    /* pointers into buffer */
    wchar_t *pu, *qu;               /* wchar_t pointers into buffer for UTF16 */
    char peekchr;                   /* peek-ahead character */
    wchar_t wpeekchr;               /* peek-ahead wchar_t */
    __int64 filepos;                /* file position after seek */
    ULONG dosretval;                /* o.s. return value */
    char tmode;                     /* textmode - ANSI/UTF-8/UTF-16 */
    BOOL fromConsole = 0;           /* true when reading from console */
    void *buf;                      /* buffer to read to */
    int retval = -2;                /* return value */

    // ...
```

```c
int _read_nolock(
    int fh,
    void *inputbuf,
    unsigned cnt
    )
{
    int bytes_read;                 /* number of bytes read */
    char *buffer;                   /* buffer to read to */
    int os_read;                    /* bytes read on OS call */
    char *p, *q;                    /* pointers into the buffer */
    wchar_t *pu, *qu;               /* wchar_t pointers into the buffer for UTF16 */
    char peekchr;                   /* peek-ahead character */
    wchar_t wpeekchr;               /* peek-ahead wchar_t */
    __int64 filepos;                /* file position after seek */
    ULONG dosretval;                /* o.s. return value */
    char tmode;                     /* textmode - ANSI/UTF-8/UTF-16 */
    BOOL fromConsole = 0;           /* true when reading from console */
    void *buf;                      /* buffer to read to */
    int retval = -2;                /* return value */

    // ...
```

```
/* lock the file */
_lock_fh(fh);

/* Init stream pointer */
stream = str;

/* flush all streams */
_flushall();

/* unlock the file */
_unlock_fh(fh);
```

```
int main()
{
    std::vector<int> v;

    // Get the data from the input file:
    v = get_data_from_input_file();

    // Find the largest value in the vector:
    int largest_value = INT_MIN;
    for (auto const i : v)
    {
        if (i > largest_value)
            largest_value = i;
    }

    // Print the largest value that we found:
    std::cout << "largest value:  " << largest_value << '\n';
}
```

```
int main()
{
    std::vector<int> v;

    // Get the data from the input file:
    v = get_data_from_input_file();

    // Find the largest value in the vector:
    int largest_value = INT_MIN;
    for (auto const i : v)
    {
        if (i > largest_value)
            largest_value = i;
    }

    // Print the largest value that we found:
    std::cout << "largest value:  " << largest_value << '\n';
}
```

```
int main()
{
    std::vector<int> const v = get_data_from_input_file();

    // Find the largest value in the vector:
    int largest_value = INT_MIN;
    for (auto const i : v)
    {
        if (i > largest_value)
            largest_value = i;
    }

    std::cout << "largest value:  " << largest_value << '\n';
}
```

```
int find_largest_value(std::vector<int> const& v)
{
    int largest_value = INT_MIN;
    for (auto const i : v)
    {
        if (i > largest_value)
            largest_value = i;
    }
    return largest_value;
}

int main()
{
    std::vector<int> const v = get_data_from_input_file();
    int const largest_value = find_largest_value(v);
    std::cout << "largest value:  " << largest_value << '\n';
}
```

```
int find_largest_value(std::vector<int> const& v)
{
    int largest_value = INT_MIN;
    for (auto const i : v)
    {
        if (i > largest_value)
            largest_value = i;
    }
    return largest_value;
}

int main()
{
    std::vector<int> const v = get_data_from_input_file();
    int const largest_value = *std::max_element(begin(v), end(v));
    std::cout << "largest value:  " << largest_value << '\n';
}
```

```
/*
int find_largest_value(std::vector<int> const& v)
{
    int largest_value = INT_MIN;
    for (auto const i : v)
    {
        if (i > largest_value)
            largest_value = i;
    }
    return largest_value;
}
*/

int main()
{
    std::vector<int> const v = get_data_from_input_file();
    int const largest_value = *std::max_element(begin(v), end(v));
    std::cout << "largest value:  " << largest_value << '\n';
}
```

```
#if 0
int find_largest_value(std::vector<int> const& v)
{
    int largest_value = INT_MIN;
    for (auto const i : v)
    {
        if (i > largest_value)
            largest_value = i;
    }
    return largest_value;
}
#endif

int main()
{
    std::vector<int> const v = get_data_from_input_file();
    int const largest_value = *std::max_element(begin(v), end(v));
    std::cout << "largest value:  " << largest_value << '\n';
}
```

```
int main()
{
    std::vector<int> const v = get_data_from_input_file();

    int const largest_value = *std::max_element(begin(v), end(v));

    std::cout << "largest value:  " << largest_value << '\n';
}
```

Ugly Comments

Comments at the top of a file that contain the file history

- You have source control, right? Right?

Comments that record date, time, author, and/or reason for change at the location of the change

- Source control. And, ugh.

Comments that explain what the next block of code does

- Refactor that block into a function with a descriptive name
- Try to avoid explaining *what* code does; but do explain *why* interesting code is the way it is

Commented out code

- Ugh
- Also, source control
- Also, ugh

Recap

Macros are ugly

Walls of code are ugly

Lambdas are beautiful

Invisible code is beautiful

Removing effort is beautiful

Other people’s code is beautiful

Comments are ugly

\0