1989 30 2019

GARMIN®

ANNIVERSARY

1989 30 2019

GARMIN

ANNIVERSARY

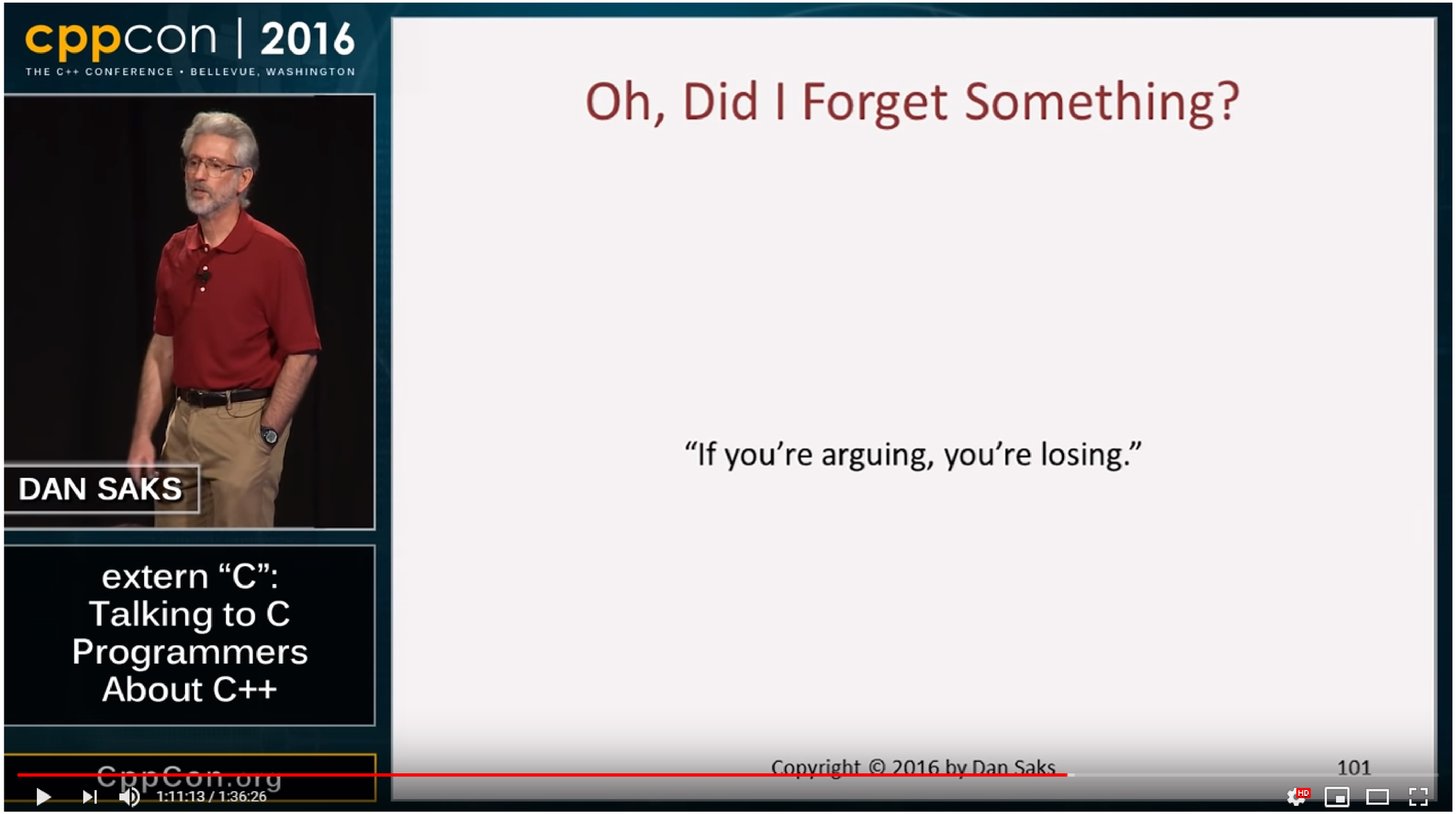
Infiltrating a Codebase:

Moving Toward a Better C

Brian Ruth, Senior Software Enginer: Brian.Ruth@Garmin.com

Motivation

Why a C talk at a C++ conference?



Why a C talk at a C++ conference?

- Like it or not, C/C++ is still a thing
- Understand the historical constraints of C
- Use code familiar to C developers
- Address fundamental concerns

You don’t always get to start from scratch

- Existing code has value
- No one is sure how the code does what it does, only that it does it
- Tests aren’t always available to ensure functionality
- The extension may be CPP, but the code is C
- Old projects need to be supported

Why not C++, an informal Poll of C programmers:

- “C++ is all object-oriented”
- “Templates are just as bad as macros for debugging”
- “The standard library is bloated”
- “I don’t want to bring in a bunch of stuff I don’t use”
- “The C code works, and has for years, why touch it?”
- “C++ hides what it is doing from you.”
- “My team knows C, they don’t know C++.”

Know your users

- Can’t immediately switch a C codebase to C++, especially modern C++
- The tools may not support anything newer than C++98 or just be really bad at C++
- The existing programmers may not be familiar with C++
- May introduce problems (memory/performance) for older or resource limited hardware
- Want to make safer and maintainable code the default

What steps can we take?

- Make code easy to read
  - Minimize macros use
  - Utilize `const` and scoping
  - Isolate globals
- Reduce duplication
  - Reducing function size
- Don’t introduce regressions
  - Ensure equal performance
- Speak the same language
  - Use C compatible subset of C++

Minimize Macro Use

(C++: type system)

When are macros necessary?

- Generic code (though not type safe)

```
#define max_val(a, b) ( (a < b) ? a : b )
static const uint8_t MAX_WIDTH = 44;

int calculate_width(double desired_width)
{
  return max_val(desired_width, MAX_WIDTH);
}
```

- Code generation

```
#define create_image_constants( _name, _path )\
   static const bmp_handle BMP_##_name##_HANDLE = create_file_handle( _path ); \
   static const int BMP_##_name##_WIDTH = get_width( BMP_##_name##_HANDLE ); \
   static const int BMP_##_name##_HEIGHT = get_height( BMP_##_name##_HANDLE );
```

When are macros necessary?

- Conditional compilation

```
void some_fun()
{
#ifdef ENABLE_LOGGING
printf("I'm here");
#endif
}
```

- "Inheritance"

```
#define BASE_PKT_INFO int id;\
                      int type;

typedef struct{
    BASE_PKT_INFO
} small_packet;

typedef struct{
    BASE_PKT_INFO
    int size;
    uint8_t* data;
} large_packet;
```

Compile time constants

- C does not allow constant lvalues to be used at compile time

```
static const int NUM_VALS = 7;
int val_array [ NUM_VALS ]; // won't compile

switch(some_val)
{
    case NUM_VALS: //won't compile
}

static const int SOME_OTHER_CONST = NUM_VALS * 14; //nope
```

Compile time constants (C edition)

- `#define` for constants

```
#define NUM_VALS 7
int val_array[ NUM_VALS ]; //size of array is 7
```

- The `enum` trick

```
enum
{
 NUM_VALS = 7
};
int val_array[ NUM_VALS ]; //size of array is 7
```

Run-time Constants

- Not all constants need to be used at compile time)

C source #1

```c
#include <stdint.h>

#define DISPLAY_WIDTH 7

void draw_bigger_than_width();
void draw_smaller_than_width();

void draw(uint8_t width)
{
    if(width < DISPLAY_WIDTH)
    {
        draw_bigger_than_width();
    } else
    {
        draw_smaller_than_width();
    }
}
```

C source #2

```c
#include <stdint.h>

static const uint8_t DISPLAY_WIDTH = 7;

void draw_bigger_than_width();
void draw_smaller_than_width();

void draw(uint8_t width)
{
    if(width < DISPLAY_WIDTH)
    {
        draw_bigger_than_width();
    } else
    {
        draw_smaller_than_width();
    }
}
```

x86-64 gcc 9.1 (Editor #1, Compiler #1) C — x86-64 gcc 9.1 — -O1

```asm
draw:
        sub     rsp, 8
        cmp     dil, 6
        ja      .L2
        mov     eax, 0
        call    draw_bigger_than_width
.L1:
        add     rsp, 8
        ret
.L2:
        mov     eax, 0
        call    draw_smaller_than_width
        jmp     .L1
```

x86-64 gcc 9.1 (Editor #2, Compiler #2) C — x86-64 gcc 9.1 — -O1

```asm
draw:
        sub     rsp, 8
        cmp     dil, 6
        ja      .L2
        mov     eax, 0
        call    draw_bigger_than_width
.L1:
        add     rsp, 8
        ret
.L2:
        mov     eax, 0
        call    draw_smaller_than_width
        jmp     .L1
```

https://godbolt.org/z/Ff4Q_P

Run-time Constants

- Not all constants need to be used at compile time)

C source #1

```c
#include <stdint.h>

#define DISPLAY_WIDTH 7

void draw_bigger_than_width();
void draw_smaller_than_width();

void draw(uint8_t width)
{
    if(width < DISPLAY_WIDTH)
    {
        draw_bigger_than_width();
    } else
    {
        draw_smaller_than_width();
    }
}
```

C source #2

```c
#include <stdint.h>

static const uint8_t DISPLAY_WIDTH = 7;

void draw_bigger_than_width();
void draw_smaller_than_width();

void draw(uint8_t width)
{
    if(width < DISPLAY_WIDTH)
    {
        draw_bigger_than_width();
    } else
    {
        draw_smaller_than_width();
    }
}
```

x86-64 gcc 9.1 (Editor #1, Compiler #1) C — x86-64 gcc 9.1 — -O1

```asm
draw:
        sub     rsp, 8
        cmp     dil, 6
        ja      .L2
        mov     eax, 0
        call    draw_bigger_than_width
.L1:
        add     rsp, 8
        ret
.L2:
        mov     eax, 0
        call    draw_smaller_than_width
        jmp     .L1
```

x86-64 gcc 9.1 (Editor #2, Compiler #2) C — x86-64 gcc 9.1 — -O1

```asm
draw:
        sub     rsp, 8
        cmp     dil, 6
        ja      .L2
        mov     eax, 0
        call    draw_bigger_than_width
.L1:
        add     rsp, 8
        ret
.L2:
        mov     eax, 0
        call    draw_smaller_than_width
        jmp     .L1
```

https://godbolt.org/z/Ff4Q_P

Pseudo-functions

- Sometimes C code is broken out into separate “functions” using macros

```
#define write_2_lines( str1, str2 ) set_colors( str_color.bg, str_color.fg ); \
                                    move_origin_to( text_crd.x + 3, text_crd.y - 10 );  \
                                    write_text(unpack_text_font( USE_ALT_FONT | SMALL_FONT ), str1, 30, ( BG_FILL_BLANK | \
                                      LEFT_JST ), pack_xbrdr_attr( 1, 1, 1, 1 ) ); \
                                    move_origin_to( text_crd.x + 3, text_crd.y + 3 );   \
                                    write_text( unpack_text_font( USE_ALT_FONT | SMALL_FONT ), str2, 20, ( BG_FILL_BLANK | \
                                      LEFT_JST ), pack_xbrdr_attr( 1, 1, 1, 1 ) )

void draw()
{
write_2_lines( “Line1”, “Line2”);
}
```

- This forces the compiler to inline the macro code everywhere the “function” is called which can lead to code bloat
- There is no type checking on the arguments
- Debuggers tend to have a hard time stepping through macros

- Prone to maintenance issues:
  - Correct `()`, `{}`, `;` (do `{…}while(0)` trick)
  - Multi-line code needs \
  - Can’t have a pre-processor check within a macro if you need to conditionally compile
  - Local variables can be hidden within the macro

Make real functions

```
static void write_2_lines(const xy_type text_crd, const colors_type str_color,
    const char* str1, const char* str2)
{
    set_colors(str_color.bg, str_color.fg);
    move_origin_to(text_crd.x + 3, text_crd.y - 10);
    write_text(unpack_text_font(USE_ALT_FONT | SMALL_FONT), str1, 30,
        (BG_FILL_BLANK | LEFT_JST), pack_xbrdr_attr(1, 1, 1, 1));
    move_origin_to(text_crd.x + 3, text_crd.y + 3);
    write_text(unpack_text_font(USE_ALT_FONT | SMALL_FONT), str2, 20,
        (BG_FILL_BLANK | LEFT_JST), pack_xbrdr_attr(1, 1, 1, 1));
}

void draw()
{
    xy_type loc;
    colors_type color;
…
    write_2_lines(loc, color, "blah", "blahs");
}
```

- Allow the compiler to decide if it wants to inline or not
- Arguments are type checked
- Easier to maintain:
    - Code reads as normal source, no need to think about the “calling” context since anything could be put in the arguments
    - Can have preprocessor macros in the function if we need to conditionally compile a portion
    - Expose hidden dependencies. The above function assumed `text_crd` and `str_color` were local variables of types `xy_type` and `colors_type`, now we explicitly show the function depends on them

Compilers like to inline

C source #2

```c
#define write_2_lines( str1, str2 ) set_colors( str_color.bg, str_color.fg ); \
                                    move_origin_to( text_crd.x + 3, text_crd.y - 10 );  \
                                    write_text(unpack_text_font( USE_ALT_FONT | SMALL_FONT ), str1, 30, ( BG_FILL_BLANK | \
                                        LEFT_JST ), pack_xbrdr_attr( 1, 1, 1, 1 ) ); \
                                    move_origin_to( text_crd.x + 3, text_crd.y + 3 );   \
                                    write_text( unpack_text_font( USE_ALT_FONT | SMALL_FONT ), str2, 20, ( BG_FILL_BLANK |
                                        LEFT_JST ), pack_xbrdr_attr( 1, 1, 1, 1 ) )

void draw()
{
    colors_type str_color;
    xy_type text_crd;
    write_2_lines("blah", "blahs");
}
```

C source #1

```c
static void write_2_lines(const xy_type text_crd, const colors_type str_color,
    const char* str1, const char* str2)
{
    set_colors(str_color.bg, str_color.fg);
    move_origin_to(text_crd.x + 3, text_crd.y - 10);
    write_text(unpack_text_font(USE_ALT_FONT | SMALL_FONT), str1, 30,
        (BG_FILL_BLANK | LEFT_JST), pack_xbrdr_attr(1, 1, 1, 1));
    move_origin_to(text_crd.x + 3, text_crd.y + 3);
    write_text(unpack_text_font(USE_ALT_FONT | SMALL_FONT), str2, 20,
        (BG_FILL_BLANK | LEFT_JST), pack_xbrdr_attr(1, 1, 1, 1));
}

void draw()
{
    colors_type str_color;
    xy_type text_crd;
    write_2_lines(text_crd, str_color, "blah", "blahs");
}
```

x86-64 gcc 9.2 (Editor #2, Compiler #1) C — x86-64 gcc 9.2 — -O1

```asm
.LC0:
        .string "blah"
.LC1:
        .string "blahs"
draw:
        push    r12
        push    rbp
        push    rbx
        mov     edi, 0
        mov     rsi, rdi
        sar     rsi, 32
        call    set_colors
        mov     r12d, 3
        mov     ebx, 0
        lea     esi, [rbx-10]
        mov     edi, r12d
        call    move_origin_to
        mov     ecx, 1
        mov     edx, 1
        mov     esi, 1
        mov     edi, 1
        call    pack_xbrdr_attr
        mov     ebp, eax
        mov     edi, 1
        call    unpack_text_font
        mov     edi, eax
        mov     r8d, ebp
        mov     ecx, 3
```

x86-64 gcc 9.2 (Editor #1, Compiler #2) C — x86-64 gcc 9.2 — -O1

```asm
.LC0:
        .string "blah"
.LC1:
        .string "blahs"
draw:
        push    r12
        push    rbp
        push    rbx
        mov     esi, 0
        mov     edi, 0
        call    set_colors
        mov     r12d, 3
        mov     ebx, 0
        lea     esi, [rbx-10]
        mov     edi, r12d
        call    move_origin_to
        mov     ecx, 1
        mov     edx, 1
        mov     esi, 1
        mov     edi, 1
        call    pack_xbrdr_attr
        mov     ebp, eax
        mov     edi, 1
        call    unpack_text_font
        mov     edi, eax
        mov     r8d, ebp
        mov     ecx, 3
        mov     edx, 30
```

https://godbolt.org/z/08PEjG

Compilers like to inline

6-64 gcc 9.2 (Editor #2, Compiler #1) C

x86-64 gcc 9.2 | -O1

11010 | ./a.out | .LX0: | lib.f: | .text | // | \s+ | Intel | Demangle

```
.LC0:
        .string "blah"
.LC1:
        .string "blahs"
draw:
        push    r12
        push    rbp
        push    rbx
        mov     edi, 0
        mov     rsi, rdi
        sar     rsi, 32
        call    set_colors
        mov     r12d, 3
        mov     ebx, 0
        lea     esi, [rbx-10]
        mov     edi, r12d
        call    move_origin_to
        mov     ecx, 1
        mov     edx, 1
        mov     esi, 1
        mov     edi, 1
        call    pack_xbrdr_attr
        mov     ebp, eax
        mov     edi, 1
```

x86-64 gcc 9.2 (Editor #1, Compiler #2) C

x86-64 gcc 9.2 | -O1

A | 11010 | ./a.out | .LX0: | lib.f: | .tex

```
.LC0:
        .string "blah"
.LC1:
        .string "blahs"
draw:
        push    r12
        push    rbp
        push    rbx
        mov     esi, 0
        mov     edi, 0
        call    set_colors
        mov     r12d, 3
        mov     ebx, 0
        lea     esi, [rbx-10]
        mov     edi, r12d
        call    move_origin_to
        mov     ecx, 1
        mov     edx, 1
        mov     esi, 1
        mov     edi, 1
        call    pack_xbrdr_attr
        mov     ebp, eax
        mov     edi, 1
        call    unpack_text_font
```

https://godbolt.org/z/08PEjG

Utilize `const` and Scoping

(C++: RAII, constexpr, type system)

Scoping – historically speaking

- C89/90 (A.K.A. ANSI-C)
  - All variables be declared at the top of a scope
  - Declarations and executable statements not be intermixed

```
int florb()
{
    int i = 0; /* ok */
    int j; /* ok */
    j = 4; /* ok – 1st executable statement */
    int r = 7; /* fails to compile */

    if( some_condition() )
    {
        int var = 4; /* ok – new scope */
    }

    for (int k = 0; k < get_length(); ++k) /* won't compile, for is not new scope */
    {
        int l = 7; /* but this is a new scope */
    }
}
```

- C99 (A.K.A. ISO-C) removed this requirement

An `int` is an `int` is an `int`…

- ANSI-C declaration syntax led to variable reuse
- Variables change meaning during a function
- In order for variables to be generic they usually have names like `tmp_int`

```
extern int get_width();
extern int get_orbital_velocity();
extern void layout_page(int i);
extern void change_delta_v(int i);

int main()
{
    int tmp_int = get_width();
    layout_page(tmp_int);
    tmp_int = get_orbital_velocity();
    change_delta_v(tmp_int);
}
```

Compilers are used to this

- No extra memory, it's reused

C source #2

```c
extern int get_width();
extern int get_orbital_velocity();
extern void layout_page(int i);
extern void change_delta_v(int i);

int main()
{
    int tmp = get_width();
    layout_page(tmp);
    tmp = get_orbital_velocity();
    change_delta_v(tmp);
}
```

C source #1

```c
extern int get_width();
extern int get_orbital_velocity();
extern void layout_page(int i);
extern void change_delta_v(int i);

int main()
{
    const int widget_width = get_width();
    layout_page(widget_width);
    const int velocity = get_orbital_velocity();
    change_delta_v(velocity);
}
```

x86-64 gcc 9.2 (Editor #2, Compiler #2) C — x86-64 gcc 9.2 — -O1

```asm
main:
        sub     rsp, 8
        mov     eax, 0
        call    get_width
        mov     edi, eax
        call    layout_page
        mov     eax, 0
        call    get_orbital_velocity
        mov     edi, eax
        call    change_delta_v
        mov     eax, 0
        add     rsp, 8
        ret
```

x86-64 gcc 9.2 (Editor #1, Compiler #1) C — x86-64 gcc 9.2 — -O1

```asm
main:
  sub rsp, 8
  mov eax, 0
  call get_width
  mov edi, eax
  call layout_page
  mov eax, 0
  call get_orbital_velocity
  mov edi, eax
  call change_delta_v
  mov eax, 0
  add rsp, 8
  ret
```

https://godbolt.org/z/q9Z1jY

Even for structs

C source #2

```c
typedef struct
{
    int i;
    double j;
    char some_name[44];
} S;

extern void get_struct(int ID, S* s);
extern void send_struct(S* s);
int main()
{
    S s;
    for (int i = 0; i < 22; ++i)
    {
        get_struct(i, &s);
        send_struct(&s);
    }
}
```

C source #1

```c
typedef struct
{
    int i;
    double j;
    char some_name[44];
} S;

extern void get_struct(int ID, S* s);
extern void send_struct(S* s);
int main()
{
    for (int i = 0; i < 22; ++i)
    {
        S s;
        get_struct(i, &s);
        send_struct(&s);
    }
}
```

x86-64 gcc 9.2 (Editor #2, Compiler #1) C

x86-64 gcc 9.2 -O1

```asm
main:
        push    rbx
        sub     rsp, 64
        mov     ebx, 0
.L2:
        mov     rsi, rsp
        mov     edi, ebx
        call    get_struct
        mov     rdi, rsp
        call    send_struct
        add     ebx, 1
        cmp     ebx, 22
        jne     .L2
        mov     eax, 0
        add     rsp, 64
        pop     rbx
        ret
```

x86-64 gcc 9.2 (Editor #1, Compiler #2) C

x86-64 gcc 9.2 -O1

```asm
main:
  push rbx
  sub rsp, 64
  mov ebx, 0
.L2:
  mov rsi, rsp
  mov edi, ebx
  call get_struct
  mov rdi, rsp
  call send_struct
  add ebx, 1
  cmp ebx, 22
  jne .L2
  mov eax, 0
  add rsp, 64
  pop rbx
  ret
```

https://godbolt.org/z/XuwSEh

An aside (macro scoping)

- macros are scoped to a translation unit after first declaration
- This leads to some maintenance issues

```
#include <stdio.h>

void blorp()
{
    #define MY_VAL 7

    printf("%s: %d\r\n", __FUNCTION__, MY_VAL);
}

void other_fun(void);

int main()
{
    printf("%s: %d\r\n", __FUNCTION__, MY_VAL);
    blorp();
    other_fun();
    printf("%s: %d\r\n", __FUNCTION__, MY_VAL);
    return 0;
}

void other_fun(void)
{
    #define MY_VAL 9 // Generates a warning, maybe
    printf("%s: %d\r\n", __FUNCTION__, MY_VAL);
}
```

Output:

```
main: 7
blorp: 7
other_fun: 9
main: 7
```

An aside (macro scoping)

- macros are scoped to a translation unit after first declaration
- This leads to some maintenance issues

```
#include <stdio.h>

void blorp()
{
    #define MY_VAL 7

    printf("%s: %d\r\n", __FUNCTION__, MY_VAL);
}

void other_fun(void)
{
    #define MY_VAL 9 // Generates a warning, maybe
    printf("%s: %d\r\n", __FUNCTION__, MY_VAL);
}

int main()
{
    printf("%s: %d\r\n", __FUNCTION__, MY_VAL);
    blorp();
    other_fun();
    printf("%s: %d\r\n", __FUNCTION__, MY_VAL);
    return 0;
}
```

Output:

main: 9
blorp: 7
other_fun: 9
main: 9

`const` in C

- C89 did not have `const`
- C90 added `const` and `volatile`
- `const` in C does not mean constant expression, it means read only*

- Use const in C for same reasons in C++
  - Code contracts
  - Readability
  - Compiler hints
  - Ensure assumptions

* by software, see const volatile

`const` all the things

- `const` all function arguments
  - Many times argument variables are either reused locally and when code is rearranged, odd things happen
  - clang-tidy warns on this, but it conveys useful information
- Make `const` locals for complicated expressions, especially conditionals
- Compilers generally inline `const` values even at low optimization levels
- Making something `const` enables some very powerful optimizations

Casting away `const` (Another Aside)

- Don't ever cast away `const`, just like in C++, the compiler is free to ignore you (or not)

```
#include <stdio.h>
void some_legacy_func(int* val)
{
   *val += 11;
}
int my_new_func(const int value)
{
    const int local_value = 10;
    some_legacy_func((int*)&value);
    some_legacy_func((int*)&local_value);
    printf("arg: %d\r\n", value);
    printf("local: %d\r\n", local_value);
    return value;
}

int main()
{
    return my_new_func( 7 );
}
```

```
Output:
arg: 18      modified
local: 10    ignored
```

Reduce function size

(C++: algorithms, classes)

The 1,000 line function, why?

- State machine or message driven systems
- Convenience
- Hooks
- Adapters

Too much to do

- Long functions usually to do many things, not all of them related
- Tend to have generic names like `message_processor`
- Can contain unnecessary control flow and duplicated code
- Very similar code is copy-paste-modified instead of refactored
- Variables change meaning throughout the function

Refactoring seams – easy candidates for functions

- Look for `switch` nested inside other control statements
- Long `if..else if.. else` chains
- Long section within control
- Comments indicating a section of code
- Code shape
  - Arrow code
  - Similar look

```
if (rowCount > rowIdx)
{
    if (drc[rowIdx].Table.Columns.Contains("avalId"))
    {
        do
        {
            if (Attributes[attrVal.AttributeClassId] == null)
            {
                // do stuff
            }
```

More subtle seams

- Same code, slight differences

```
static status_t get_employees_from_dept(employee* filtered_list,
    int* cnt, const department dep)
{
    status_t status = NOT_FOUND;
    int num_found = 0;
    for (int i = 0; i < MAX_ITEMS; i++)
    {
        employee rec;
        status = get_rec(i, &rec);
        if (status == NOT_FOUND)
        {
            break;
        }

        if( rec.dep == dep )
        {
            filtered_list[num_found] = rec;
            num_found++;
        }

    }
    *cnt = num_found;
    return status;
}
```

```
static status_t get_employees_with_gt_years(employee*
filtered_list, int* cnt, const int years)
{
    status_t           status = NOT_FOUND;
    int num_found = 0;
    for (int i = 0; i < MAX_ITEMS; i++)
    {
        employee rec;
        status = get_rec(i, &rec);
        if (status == NOT_FOUND)
        {
            break;
        }

        if (rec.years_experience > years )
        {
            filtered_list[num_found] = rec;
            num_found++;
        }

    }
    *cnt = num_found;
    return status;
}
```

More subtle seams

- Same code, slight differences

```
static status_t get_employees_from_dept(employee* filtered_list,
    int* cnt, const department dep)
{
    status_t status = NOT_FOUND;
    int num_found = 0;
    for (int i = 0; i < MAX_ITEMS; i++)
    {
        employee rec;
        status = get_rec(i, &rec);
        if (status == NOT_FOUND)
        {
            break;
        }

        if( rec.dep == dep )
        {
            filtered_list[num_found] = rec;
            num_found++;
        }

    }
    *cnt = num_found;
    return status;
}
```

```
static status_t get_employees_with_gt_years(employee*
filtered_list, int* cnt, const int years)
{
    status_t           status = NOT_FOUND;
    int num_found = 0;
    for (int i = 0; i < MAX_ITEMS; i++)
    {
        employee rec;
        status = get_rec(i, &rec);
        if (status == NOT_FOUND)
        {
            break;
        }

        if (rec.years_experience > years )
        {
            filtered_list[num_found] = rec;
            num_found++;
        }

    }
    *cnt = num_found;
    return status;
}
```

More subtle seams

- Pass function pointer

```
typedef _Bool (*filter_predicate)(const employee * const);

static status_t filter_list(employee* filtered_list, int* cnt,
    filter_predicate filter)
{
    status_t           status = NOT_FOUND;
    int num_found = 0;
    for(int i = 0; i < MAX_ITEMS; i++)
    {
        employee rec;
        status = get_rec(i, &rec);
        if (status == NOT_FOUND)
        {
            break;
        }

        if(filter(&rec))
        {
            filtered_list[num_found] = rec;
            num_found++;
        }
    }
    *cnt = num_found;
    return status;
}
```

```
static _Bool is_engineer(const employee *const employee)
{
    return employee->dep == DEPT_ENGINEERING;
}

static _Bool is_4wk_vacation(const employee* const employee)
{
    return employee->years_experience > 10;
}

int main()
{
    employee engineering_dept[MAX_ITEMS];
    int num_engineers = 0;
    filter_list(engineering_dept, &num_engineers, is_engineer);
    print_list(engineering_dept, num_engineers);

    employee vacation_4weeks[MAX_ITEMS];
    int num_4wk = 0;
    filter_list(vacation_4weeks, &num_4wk, is_4wk_vacation);
    print_list(vacation_4weeks, num_4wk);
    return 0;
}
```

If you can’t refactor right now, rename

- Consider a function updateDatabase:

```
_Bool updateDatabase(request* req)
{
    /* common code to setup DB */
    if ( request_exists(req))
    {
        /* update a record */
    }
    else
    {
        /* add new record */
    }
}
```

If you can't refactor right now, rename

- Consider a function updateDatabase:

```
_Bool updateDatabase(request* req)
{
    /* common code to setup DB */
    if ( request_exists(req))
    {
        /* update a record */
    }
    else
    {
        /* add new record */
    }
}
```

Rename to:

UpdateOrAddRequestToDatabase

Isolate Globals

(C++: Classes)

Hidden dependencies are hard to track

- C programs routinely use many constructs that qualify as globals
  - Macros
  - extern functions/variables
  - Implicit functions
  - Global variables

An exercise…

- This snippet relies on the global `current_canvas`

```
extern canvas current_canvas;
void set_column_count(const int cols)
{
    num_columns = cols;
    current_canvas.dirty = true;
}

void draw()
{
    if (current_canvas.dirty)
    {
        const int col_width = current_canvas.width / num_columns;
        int col_x = 0;
        for (int i = 0; i < num_columns; ++i)
        {
            draw_column(col_x, col_width);
            col_x += col_width;
        }
        current_canvas.dirty = false;
    }
}
```

Isolation steps

1. Isolate the variable in a function that returns a `const canvas *`

```
extern canvas current_canvas;
static const canvas * get_current_canvas()
{
    return &current_canvas;
}

void set_column_count(const int cols)
{
    num_columns = cols;
    get_current_canvas()->dirty = true; //won’t compile
}

void draw()
{
    if (get_current_canvas()->dirty)
    {
        const int col_width = get_current_canvas()->width /
num_columns;
        int col_x = 0;
        for (int i = 0; i < num_columns; ++i)
        {
            draw_column(col_x, col_width);
            col_x += col_width;
        }
        get_current_canvas()->dirty = false; // won’t compile
    }
}
```

Isolation steps

1. Isolate the variable in a function that returns a `const canvas *const`
2. Make a `*_mutable` version for any modification

```
extern canvas current_canvas;
static const canvas* get_current_canvas()
{
    return &current_canvas;
}

static canvas* get_current_canvas_mutable()
{
    return &current_canvas;
}

void set_column_count(const int cols)
{
    num_columns = cols;
    get_current_canvas_mutable()->dirty = true;
}

void draw()
{
    if (get_current_canvas()->dirty)
    {
        const int col_width = get_current_canvas()->width / num_columns;
        int col_x = 0;
        for (int i = 0; i < num_columns; ++i)
        {
            draw_column(col_x, col_width);
            col_x += col_width;
        }
        get_current_canvas_mutable()->dirty = false;
    }
}
```

Isolation steps

1. Isolate the variable in a function that returns a `const canvas *const`
2. Make a `*_mutable` version for any modification
3. Create a function to isolate modifications

```
extern canvas current_canvas;
static const canvas* get_current_canvas()
{
    return &current_canvas;
}

static canvas* get_current_canvas_mutable()
{
    return &current_canvas;
}

static void set_current_canvas_dirty(const _Bool is_dirty)
{
    get_current_canvas_mutable()->dirty = is_dirty;
}

void set_column_count(const int cols)
{
    num_columns = cols;
    set_current_canvas_dirty(true);
}

void draw()
{
    if (get_current_canvas()->dirty)
    {
        const int col_width = get_current_canvas()->width / num_columns;
        int col_x = 0;
        for (int i = 0; i < num_columns; ++i)
        {
            draw_column(col_x, col_width);
            col_x += col_width;
        }
        set_current_canvas_dirty(false);
    }
}
```

Isolated, and identical

C source #1

```c
void set_column_count(const int cols)
{
    num_columns = cols;
    current_canvas.dirty = true;
}

void draw()
{
    if (current_canvas.dirty)
    {
        const int col_width = current_canvas.width / num_columns;
        int col_x = 0;
        for (int i = 0; i < num_columns; ++i)
        {
            draw_column(col_x, col_width);
            col_x += col_width;
        }
        current_canvas.dirty = false;
    }
}
```

C source #2

```c
static const canvas* const get_current_canvas()
{
    return &current_canvas;
}

static canvas* get_current_canvas_mutable()
{
    return &current_canvas;
}

static void set_current_canvas_dirty(const _Bool is_dirty)
{
    get_current_canvas_mutable()->dirty = is_dirty;
}

void set_column_count(const int cols)
{
    num_columns = cols;
    set_current_canvas_dirty(true);
}

void draw()
{
    if (get_current_canvas()->dirty)
    {
        const int col_width = get_current_canvas()->width / num_columns;
        int col_x = 0;
        for (int i = 0; i < num_columns; ++i)
        {
            draw_column(col_x, col_width);
            col_x += col_width;
        }
        set_current_canvas_dirty(false);
    }
}
```

x86-64 gcc 9.2 (Editor #1, Compiler #1) C — x86-64 gcc 9.2 — -O1

```asm
set_column_count:
        mov     DWORD PTR num_columns[rip], edi
        mov     BYTE PTR current_canvas[rip+4], 1
        ret
draw:
        cmp     BYTE PTR current_canvas[rip+4], 0
        je      .L8
        push    r12
        push    rbp
        push    rbx
        mov     ecx, DWORD PTR num_columns[rip]
        mov     eax, DWORD PTR current_canvas[rip]
        cdq
        idiv    ecx
        mov     r12d, eax
        test    ecx, ecx
        jle     .L4
        mov     ebp, 0
        mov     ebx, 0
.L5:
        mov     esi, r12d
        mov     edi, ebx
        call    draw_column
        add     ebx, r12d
        add     ebp, 1
        cmp     DWORD PTR num_columns[rip], ebp
        jg      .L5
.L4:
        mov     BYTE PTR current_canvas[rip+4], 0
        pop     rbx
        pop     rbp
        pop     r12
        ret
.L8:
        ret
num_columns:
        .long   4
```

x86-64 gcc 9.2 (Editor #2, Compiler #2) C — x86-64 gcc 9.2 — -O1

```asm
set_column_count:
        mov     DWORD PTR num_columns[rip], edi
        mov     BYTE PTR current_canvas[rip+4], 1
        ret
draw:
        cmp     BYTE PTR current_canvas[rip+4], 0
        je      .L8
        push    r12
        push    rbp
        push    rbx
        mov     ecx, DWORD PTR num_columns[rip]
        mov     eax, DWORD PTR current_canvas[rip]
        cdq
        idiv    ecx
        mov     r12d, eax
        test    ecx, ecx
        jle     .L4
        mov     ebp, 0
        mov     ebx, 0
.L5:
        mov     esi, r12d
        mov     edi, ebx
        call    draw_column
        add     ebx, r12d
        add     ebp, 1
        cmp     DWORD PTR num_columns[rip], ebp
        jg      .L5
.L4:
        mov     BYTE PTR current_canvas[rip+4], 0
        pop     rbx
        pop     rbp
        pop     r12
        ret
.L8:
        ret
num_columns:
        .long   4
```

https://godbolt.org/z/cU--XJ

Isolated, and identical

```
set_column_count:
        mov     DWORD PTR num_columns[rip], edi
        mov     BYTE PTR current_canvas[rip+4], 1
        ret
draw:
        cmp     BYTE PTR current_canvas[rip+4], 0
        je      .L8
        push    r12
        push    rbp
        push    rbx
        mov     ecx, DWORD PTR num_columns[rip]
        mov     eax, DWORD PTR current_canvas[rip]
        cdq
        idiv    ecx
        mov     r12d, eax
        test    ecx, ecx
        jle     .L4
        mov     ebp, 0
        mov     ebx, 0
.L5:
        mov     esi, r12d
        mov     edi, ebx
        call    draw_column
        add     ebx, r12d
        add     ebp, 1
        cmp     DWORD PTR num_columns[rip], ebp
        jg      .L5
.L4:
        mov     BYTE PTR current_canvas[rip+4], 0
        pop     rbx
        pop     rbp
        pop     r12
        ret
.L8:
        ret
num_columns:
        .long   4
```

```
set_column_count:
        mov     DWORD PTR num_columns[rip], edi
        mov     BYTE PTR current_canvas[rip+4], 1
        ret
draw:
        cmp     BYTE PTR current_canvas[rip+4], 0
        je      .L8
        push    r12
        push    rbp
        push    rbx
        mov     ecx, DWORD PTR num_columns[rip]
        mov     eax, DWORD PTR current_canvas[rip]
        cdq
        idiv    ecx
        mov     r12d, eax
        test    ecx, ecx
        jle     .L4
        mov     ebp, 0
        mov     ebx, 0
.L5:
        mov     esi, r12d
        mov     edi, ebx
        call    draw_column
        add     ebx, r12d
        add     ebp, 1
        cmp     DWORD PTR num_columns[rip], ebp
        jg      .L5
.L4:
        mov     BYTE PTR current_canvas[rip+4], 0
        pop     rbx
        pop     rbp
        pop     r12
        ret
.L8:
        ret
num_columns:
        .long   4
```

https://godbolt.org/z/cU--XJ

Isolation steps

1. Isolate the variable in a function that returns a `const canvas *const`
2. Make a `*_mutable` version for any modification
3. Create a function to isolate modifications
4. Move functions to another files, remove `static` and make `_mutable` inaccessible.

```
#include "current_canvas.h“

void set_column_count(const int cols)
{
    num_columns = cols;
    set_current_canvas_dirty(true);
}

void draw()
{
    if (get_current_canvas()->dirty)
    {
        const int col_width = get_current_canvas()->width / num_columns;
        int col_x = 0;
        for (int i = 0; i < num_columns; ++i)
        {
            draw_column(col_x, col_width);
            col_x += col_width;
        }
        set_current_canvas_dirty(false);
    }
}
```

current_canvas.h

```
const canvas* get_current_canvas();
void set_current_canvas_dirty(const _Bool is_dirty);
```

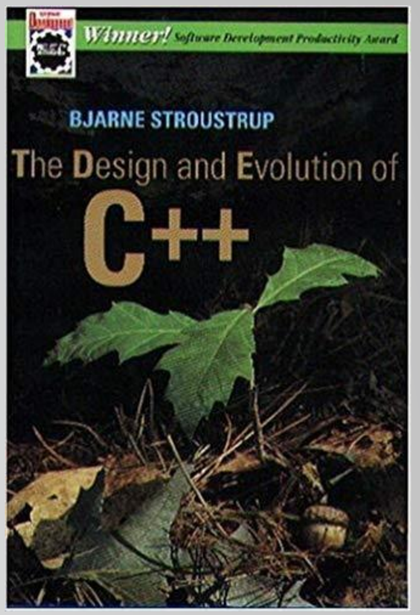
current_canvas.c

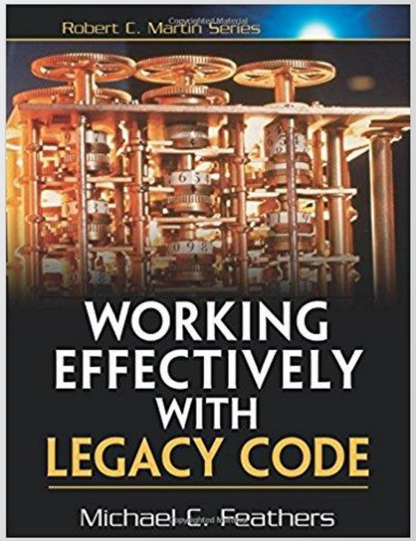
```
static canvas current_canvas;
const canvas* get_current_canvas()
{
    return &current_canvas;
}

static canvas* get_current_canvas_mutable()
{
    return &current_canvas;
}

void set_current_canvas_dirty(const _Bool is_dirty)
{
    get_current_canvas_mutable()->dirty = is_dirty;
}
```

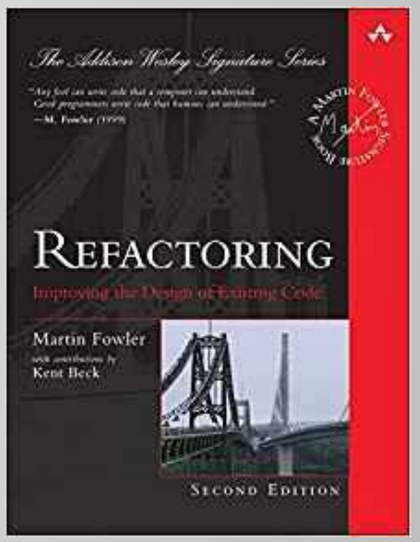
If you're arguing, you're losing

Resources

- Trust your compiler
  - Matt Godbolt: What Everyone Should Know About How Amazing Compilers Are: https://www.youtube.com/watch?v=w0sz5WbS5AM
  - Jason Turner: Rich Code for Tiny Computers: A Simple Commodore 64 Game in C++17: https://www.youtube.com/watch?v=zBkNBP00wJE
- Communication
  - Dan Saks: extern “C”: Talking to C Programmers about C++: https://www.youtube.com/watch?v=D7Sd8A6_fYU
- Inlining
  - Jason Turner: C++ Weekly - Ep 136 - How `inline` Might Affect The Optimizer: https://www.youtube.com/watch?v=GldFtXZkgYo
- Naming
  - Arlo Belshee: Good Naming is a Process: http://arlobelshee.com/good-naming-is-a-process-not-a-single-step/







1989 30 2019
GARMIN
ANNIVERSARY



1989 30 2019

GARMIN®

ANNIVERSARY

We’re hiring worldwide: https://careers.garmin.com