

cppcon 2017

# cppcon 2017

- Attended eighteen talks

# cppcon 2017

- Attended eighteen talks
- Saw twenty-one speakers

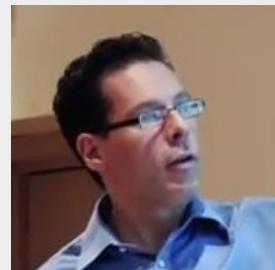
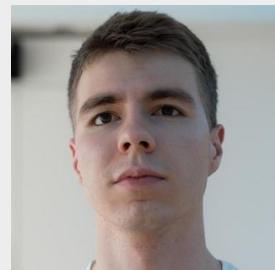
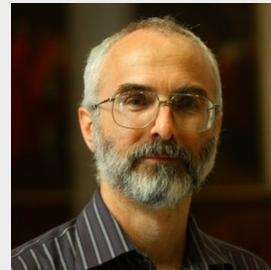
# cppcon 2017

- Attended eighteen talks
- Saw twenty-one speakers



# cppcon 2017

- Some were boring



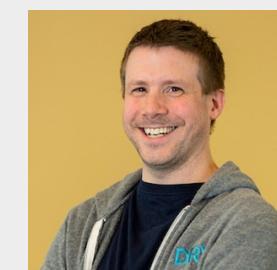
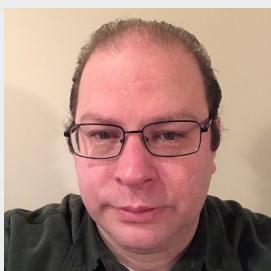
# cppcon 2017

- Some were boring



# cppcon 2017

- Here are the talks I liked



# From security to performance to GPU programming: exploring modern allocators

# From security to performance to GPU programming: exploring modern allocators

- Sergey Zubkov



# From **security** to **performance** to GPU programming: exploring modern **allocators**

- Sergey Zubkov



- Allocators allow you to do magic in legacy situations

# From **security** to **performance** to GPU programming: exploring modern **allocators**

- Sergey Zubkov



- Allocators allow you to do magic in legacy situations
- Memory pools can prevent fragmentation

# From **security** to **performance** to GPU programming: exploring modern **allocators**

- Sergey Zubkov



- Allocators allow you to do magic in legacy situations
- Memory pools can prevent fragmentation



# From security to performance to GPU programming: exploring modern allocators

- Sergey Zubkov



- Allocators allow you to do magic in legacy situations
- Memory pools can prevent fragmentation



# **dynamic\_cast** From Scratch

# **dynamic\_cast** From Scratch

- Arthur O'Dwyer



# **dynamic\_cast** From Scratch

- Arthur O'Dwyer
- Inheritance is cuckoo bananas



# **dynamic\_cast** From Scratch

- Arthur O'Dwyer
- Inheritance is cuckoo bananas
- Informative talk. Takeaway:



# dynamic\_cast From Scratch

- Arthur O'Dwyer



- Inheritance is cuckoo bananas
- Informative talk. Takeaway:
- If you want to understand your objects, use neither multiple nor virtual inheritance

# Using Functional Programming Patterns to build a clean and simple HTTP routing API

# Using Functional Programming Patterns to build a clean and simple HTTP routing API

- Jeremy Demeule, Quentin Duval



# Using Functional Programming Patterns to build a clean and simple HTTP routing API

- Jeremy Demeule, Quentin Duval



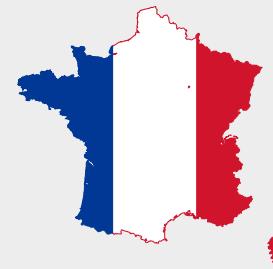
# Using Functional Programming Patterns to build a clean and simple HTTP routing API

- Jeremy Demeule, Quentin Duval



# Using Functional Programming Patterns to build a clean and simple HTTP routing API

- Jeremy Demeule, Quentin Duval



- Use expression templates to define *declarative* domain specific languages in your API

# Using Functional Programming Patterns to build a clean and simple HTTP routing API

- Jeremy Demeule, Quentin Duval



- Use expression templates to define *declarative* domain specific languages in your API

```
auto router = RequestRouter(
    GET(Uri("stocks")/"current" / Int("id")) , LastPrice(),
    GET(Uri("stocks")/Ticker("ticker")) , History(),
    GET(Uri("status")/Regex("(exchange|ticker) [0 - 9]+") , Status()));

cout << typeid(router).name() << '\n'; // something insane
```

# Agent based class design, C++ with a robot glue gun

# Agent based class design, C++ with a robot glue gun

- Odin Holmes



# Agent based class design, C++ with a robot glue gun

- Odin Holmes



- Odin is a C++ wizard

# Agent based class design, C++ with a robot glue gun

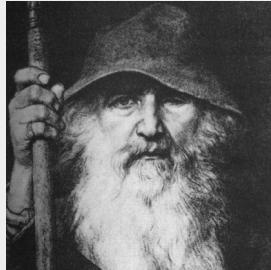
- Odin Holmes



- Odin is a C++ wizard

# Agent based class design, C++ with a robot glue gun

- Odin Holmes



- Odin is a C++ wizard
- I don't understand what Odin is doing

# Agent based class design, C++ with a robot glue gun

- Odin Holmes



- Odin is a C++ wizard
- I don't understand what Odin is doing
- Something about assembling capabilities of electronic components at compile time in a modular fashion

# Agent based class design, C++ with a robot glue gun

- Odin Holmes



- Odin is a C++ wizard
- I don't understand what Odin is doing
- Something about assembling capabilities of electronic components at compile time in a modular fashion
- Statically defined user interfaces!

# Meta: Thoughts on generative C++

# Meta: Thoughts on generative C++

- Herb Sutter



# Meta: Thoughts on generative C++

- Herb Sutter



- Proposed an idea of what “meta-classes” might look like

# Meta: Thoughts on generative C++

- Herb Sutter



- Proposed an idea of what “meta-classes” might look like
- I’m not turning C++ into Lisp, I promise!

# Meta: Thoughts on generative C++

- Herb Sutter

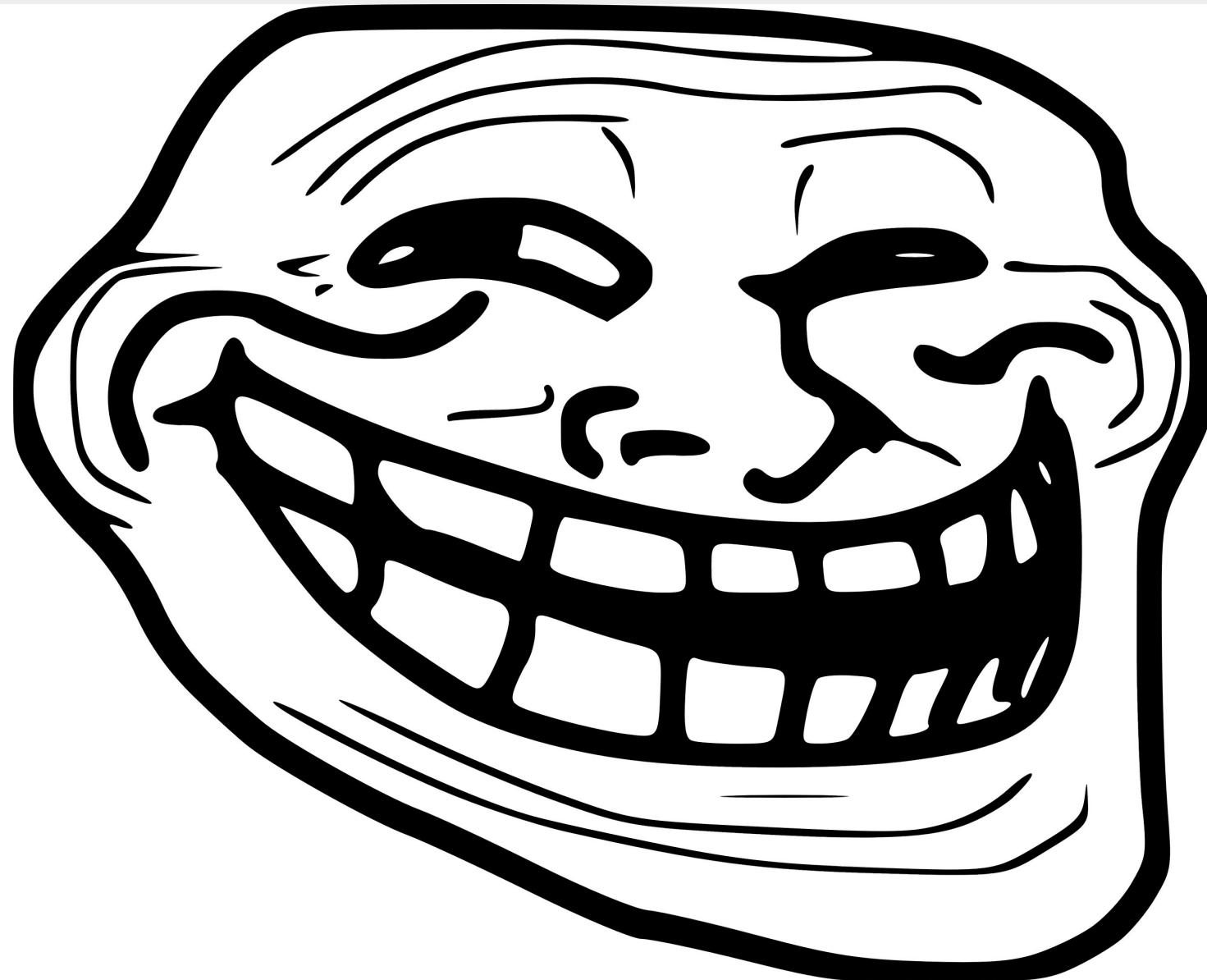


- Proposed an idea of what “meta-classes” might look like
- I’m not turning C++ into Lisp, I promise!
- I’m just simplifying what we already do, honest!

# Meta: Thoughts on generative C++

```
(define-syntax define-record-type
  (syntax-rules ()
    ((define-record-type type
      (constructor constructor-tag ...))
     predicate
     (field-tag accessor . more) ...))
  (begin
    (define type
      (make-record-type 'type '(field-tag ...)))
    (define constructor
      (record-constructor type ' (constructor-tag ...)))
    (define predicate
      (record-predicate type)))
    (define-record-field type field-tag accessor . more)
    ...))))
```

# Meta: Thoughts on generative C++



# Reader-Writer Lock versus Mutex – Understanding a Lost Bet

# Reader-Writer Lock versus Mutex – Understanding a Lost Bet

- Jeffrey Mendelsohn



# Reader-Writer Lock versus Mutex – Understanding a Lost Bet

- Jeffrey Mendelsohn



- Just use a mutex

# Reader-Writer Lock versus Mutex – Understanding a Lost Bet

- Jeffrey Mendelsohn



- Just use a mutex
- Always label your axes

# Reader-Writer Lock versus Mutex – Understanding a Lost Bet

- Jeffrey Mendelsohn



- Just use a mutex
- Always label your axes
- Measure benchmarks *in isolation*

# An Interesting Lock-free Queue – Part 2 of N

# An Interesting Lock-free Queue – Part 2 of N

- Tony Van Eerd



# An Interesting Lock-free Queue – Part 2 of N

- Tony Van Eerd



- Lock-free queues are interesting



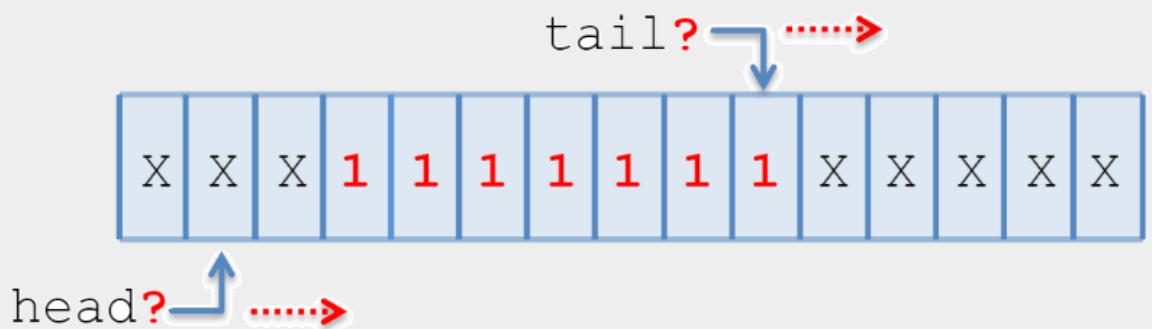
# An Interesting Lock-free Queue – Part 2 of N

- Tony Van Eerd



- Lock-free queues are interesting

- Subtle



# An Interesting Lock-free Queue – Part 2 of N

- Tony Van Eerd



- Lock-free queues are interesting

- Subtle

- Fun to think about



# An Interesting Lock-free Queue – Part 2 of N

- Tony Van Eerd



- Lock-free queues are interesting

- Subtle

- Fun to think about

- You should never use one



# Objects, Lifetimes, and References

# Objects, Lifetimes, and References oh my: the C++ Object Model, and Why it Matters to You

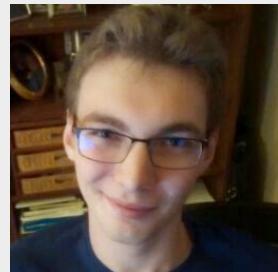
# Objects, Lifetimes, and References oh my: the C++ Object Model, and Why it Matters to You

- Nicole Mazzuca



# Objects, Lifetimes, and References oh my: the C++ Object Model, and Why it Matters to You

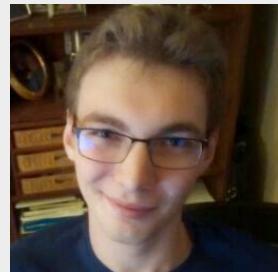
- Nicole Mazzuca



- Rebind reference members when assigning to object

# Objects, Lifetimes, and References oh my: the C++ Object Model, and Why it Matters to You

- Nicole Mazzuca



- Rebind reference members when assigning to object
- Value/object relationship

# Objects, Lifetimes, and References oh my: the C++ Object Model, and Why it Matters to You

- Nicole Mazzuca



- Rebind reference members when assigning to object
- Value/object relationship



**So, you inherited a large code base...**

# So, you inherited a large code base...

- David Sankel



# So, you inherited a large code base...

- David Sankel



- How to improve BAS

# So, you inherited a large code base...

- David Sankel



- How to improve BAS
- Migration paths

# So, you inherited a large code base...

- David Sankel



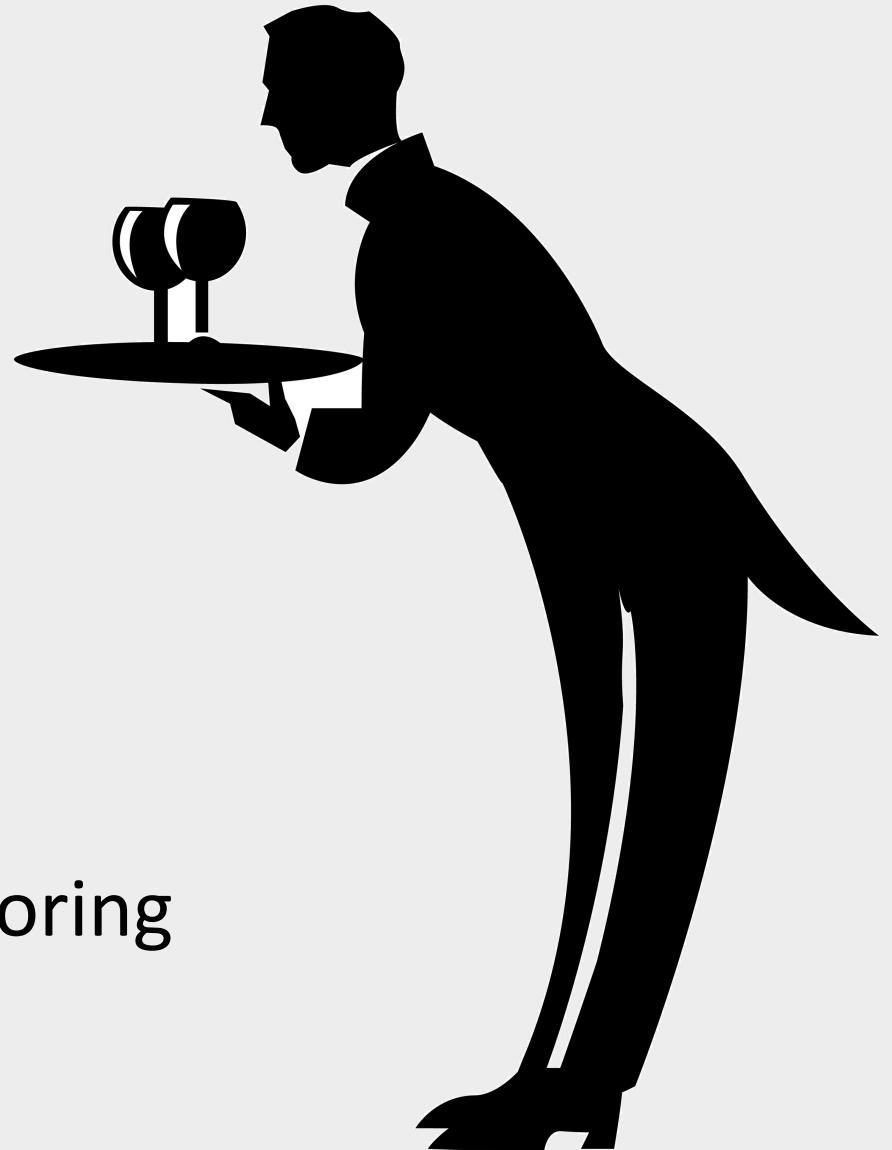
- How to improve BAS
- Migration paths
- Mnemonics, automated refactoring

# So, you inherited a large code base...

- David Sankel



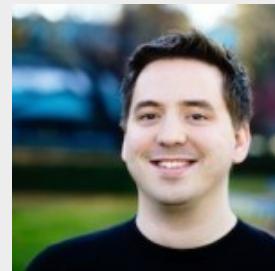
- How to improve BAS
- Migration paths
- Mnemonics, automated refactoring



# Going Nowhere Faster

# Going Nowhere Faster

- Chandler Carruth



# Going Nowhere Faster

- Chandler Carruth



- X86 code is compiled (in the chip) into microcode

# Going Nowhere Faster

- Chandler Carruth



- X86 code is compiled (in the chip) into microcode
- Microcode is pipelined, scheduled, reordered

# Going Nowhere Faster

- Chandler Carruth

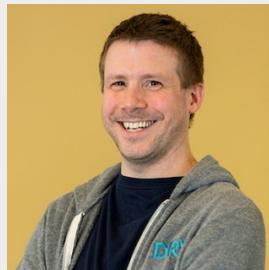


- X86 code is compiled (in the chip) into microcode
- Microcode is pipelined, scheduled, reordered
- Code does not do what you think

# Unbolting the Compiler's Lid: What Has My Compiler Done for Me Lately?

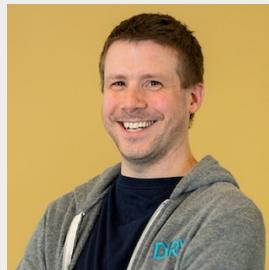
# Unbolting the Compiler's Lid: What Has My Compiler Done for Me Lately?

- Matt Godbolt



# Unbolting the Compiler's Lid: What Has My Compiler Done for Me Lately?

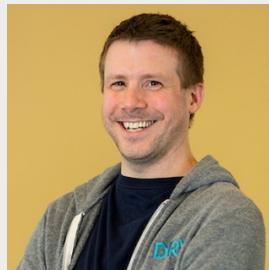
- Matt Godbolt



- godbolt.org is cool

# Unbolting the Compiler's Lid: What Has My Compiler Done for Me Lately?

- Matt Godbolt



- godbolt.org is cool
- Matt is cool

# Unbolting the Compiler's Lid: What Has My Compiler Done for Me Lately?

- Matt Godbolt



- godbolt.org is cool
- Matt is cool
- The -O3 optimization setting is cool

# Unbolting the Compiler's Lid: What Has My Compiler Done for Me Lately?

The screenshot shows the Compiler Explorer interface with three panes displaying assembly code for the same C++ program:

```
Compiler Explorer - C++ x
Secure | https://godbolt.org
GOGL Phabricator grok Code Tools Jira Gists bbgithub Jenkins colorize ib Hypersheet bbpy bbpy-pages BASH BAS internal Splunk Prod Splunk Dev ENGX Other bookmarks
Compiler Explorer C++ Editor Diff View More ...
C++ source #1 x
x86-64 clang (trunk) (Editor #1, Compiler #1) x
x86-64 clang (trunk) -O1
main:
    test edi, edi
    jle .LBB0_1
    lea eax, [rdi - 1]
    lea ecx, [rdi - 2]
    imul rax, rax
    shr rax
    lea eax, [rcx + rdi]
    add eax, -1
    ret
.LBB0_1:
    xor eax, eax
    ret
x86-64 gcc (trunk) (Editor #1, Compiler #2) x
x86-64 gcc (trunk) -O3
main:
    test edi, edi
    jle .L8
    lea eax, [rdi - 1]
    cmp eax, 17
    jbe .L9
    pxor xmm0, xmm0
    mov edx, edi
    xor eax, eax
    movdqa xmm1, XMMWORD PTR .LC0[rip]
    shr edx, 2
    movdqa xmm2, XMMWORD PTR .LC1[rip]
.L5:
    add eax, 1
    paddd xmm0, xmm1
    paddd xmm1, xmm2
    cmp eax, edx
    jne .L5
    movdqa xmm1, xmm0
    mov edx, edi
    and edx, -4
    psrlq xmm1, 8
    paddd xmm0, xmm1
    movdqa xmm1, xmm0
    cmp edi, edx
    psrlq xmm1, 4
    paddd xmm0, xmm1
    movd eax, xmm0
    je .L11
.L7:
    add eax, edx
    add edx, 1
    cmp edi, edx
    jg .L7
    ret
.L11:
    ret
.L8:
    xor eax, eax
    ret
.L9:
    xor edx, edx
    xor eax, eax
    jmp .L7
.LC0:
    .long 0
    .long 1
    .long 2
    .long 3
.LC1:
    .long 4
    .long 4
    .long 4
    .long 4
```

The left pane shows the C++ source code:

```
1 int main(int argc, char *argv[])
2 {
3     int sum = 0;
4     for (int i = 0; i < argc; ++i) {
5         sum += i;
6     }
7     return sum;
8 }
9
10 }
```

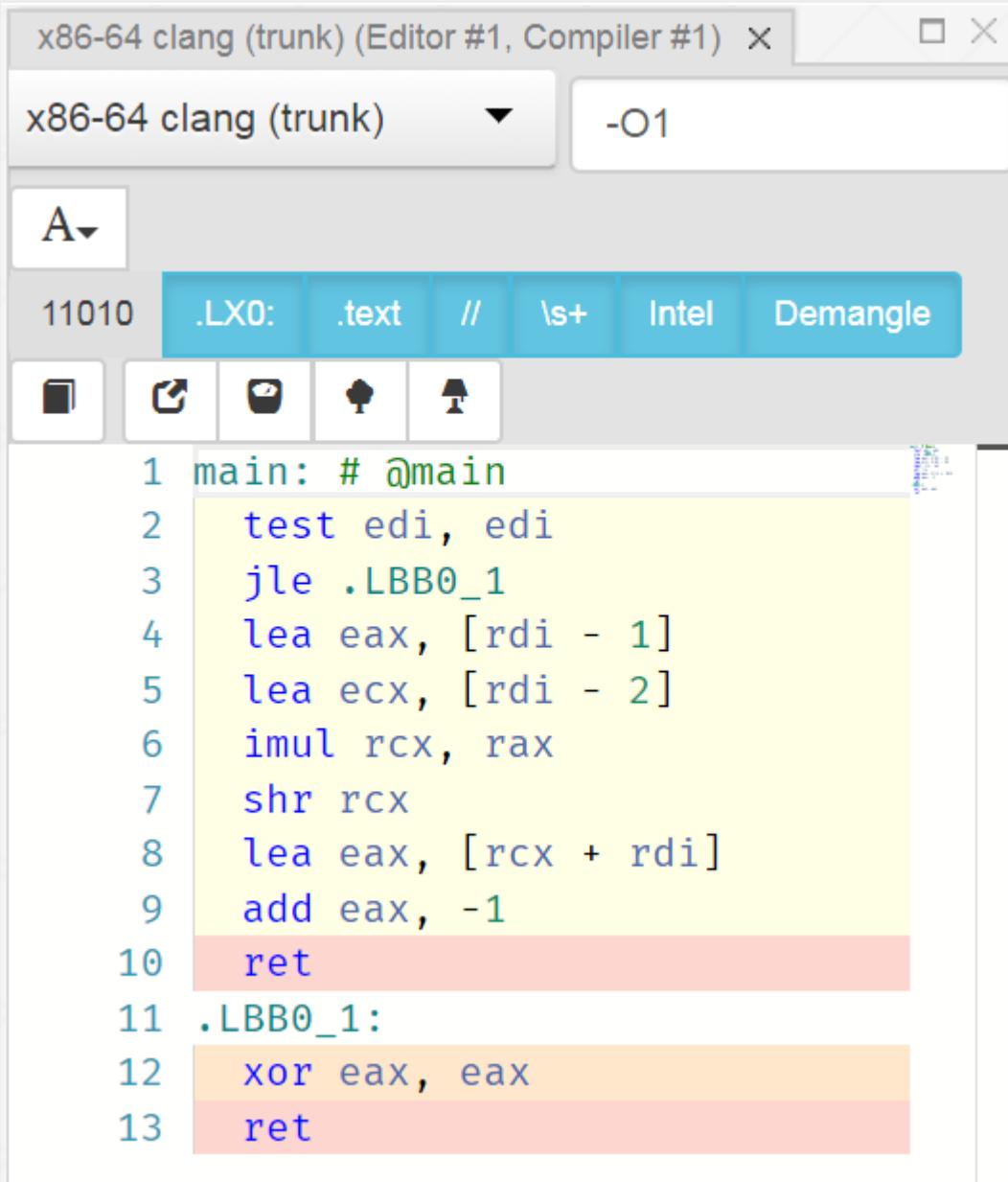
The middle pane shows the assembly output for clang with optimization level -O1.

The right pane shows the assembly output for gcc with optimization level -O3.

# Unbolting the Compiler's Lid: What Has My Compiler Done for Me Lately?

```
C++ source #1 x
A- H ↑ ☰
1
2 int main(int argc, char *argv[])
3 {
4     int sum = 0;
5     for (int i = 0; i < argc; ++i) {
6         sum += i;
7     }
8
9     return sum;
10}
11
```

# Unbolting the Compiler's Lid: What Has My Compiler Done for Me Lately?



The screenshot shows the assembly editor of the GDB debugger interface. The title bar indicates it is for an x86-64 clang (trunk) build, with Editor #1 and Compiler #1. The assembly code is being viewed for the main function, with optimization level -O1 selected. The assembly code is as follows:

```
1 main: # @main
2     test edi, edi
3     jle .LBB0_1
4     lea eax, [rdi - 1]
5     lea ecx, [rdi - 2]
6     imul rcx, rax
7     shr rcx
8     lea eax, [rcx + rdi]
9     add eax, -1
10    ret
11 .LBB0_1:
12    xor eax, eax
13    ret
```

The assembly code is color-coded: comments and labels are green, registers and memory addresses are blue, and instructions are black. The first few lines of the main function are highlighted in yellow, while the loop body and the .LBB0\_1 label are highlighted in orange. The final RET instruction is highlighted in pink.

# Unbolting the Compiler's Lid: What Has My Compiler Done for Me Lately?

x86-64 gcc (trunk) (Editor #1, Compiler #2) x

x86-64 gcc (trunk) -O3

A 11010 .LX0: .text // \s+ Intel Demangle

1 main:  
2 test edi, edi  
3 jle .L8  
4 lea eax, [rdi-1]  
5 cmp eax, 17  
6 jbe .L9  
7 pxor xmm0, xmm0  
8 mov edx, edi  
9 xor eax, eax  
10 movdqa xmm1, XMMWORD PTR .LC0[rip]  
11 shr edx, 2  
12 movdqa xmm2, XMMWORD PTR .LC1[rip]  
.L5:  
14 add eax, 1  
15 paddd xmm0, xmm1  
16 paddd xmm1, xmm2  
17 cmp eax, edx  
18 jne .L5  
19 movdqa xmm1, xmm0  
20 mov edx, edi  
21 and edx, -4  
22 psrlcq xmm1, 8  
23 paddd xmm0, xmm1

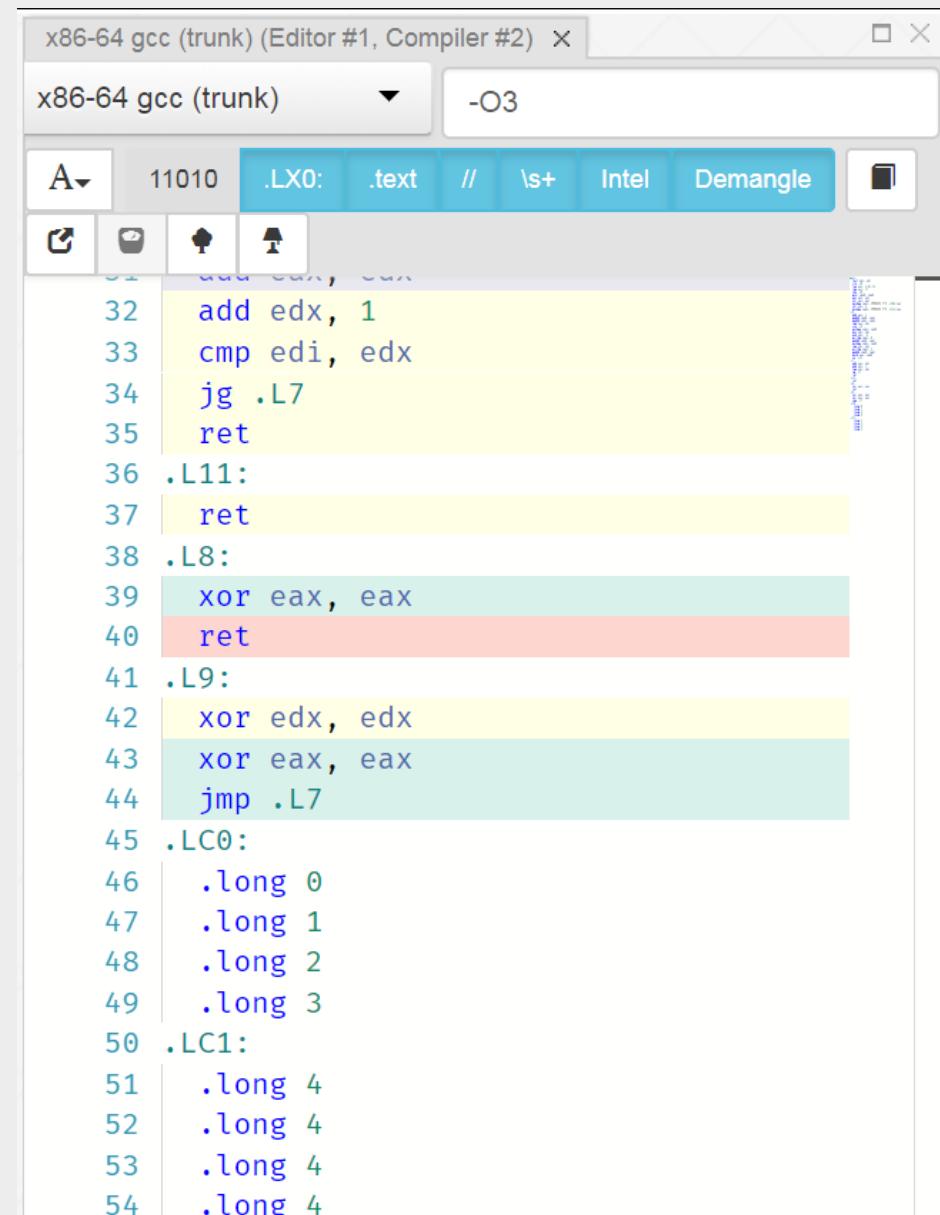
x86-64 gcc (trunk) (Editor #1, Compiler #2) x

x86-64 gcc (trunk) -O3

A 11010 .LX0: .text // \s+ Intel Demangle

24 movdqa xmm1, xmm0  
25 cmp edi, edx  
26 psrlcq xmm1, 4  
27 paddd xmm0, xmm1  
28 movd eax, xmm0  
29 je .L11  
.L7:  
31 add eax, edx  
32 add edx, 1  
33 cmp edi, edx  
34 jg .L7  
35 ret  
.L11:  
37 ret  
.L8:  
39 xor eax, eax  
40 ret  
.L9:  
42 xor edx, edx  
43 xor eax, eax  
44 jmp .L7  
.LC0:  
46 .long 0

# Unbolting the Compiler's Lid: What Has My Compiler Done for Me Lately?



The screenshot shows the Immunity Debugger interface with the following configuration:

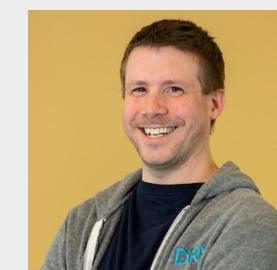
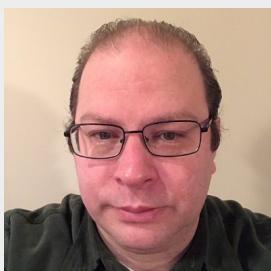
- Compiler: x86-64 gcc (trunk)
- Optimization Level: -O3
- Architecture: A (Assembly)
- File Format: 11010
- Section: .LX0: (selected)
- Text View: .text
- Comment View: //
- Registers View: \s+
- Intel View: Intel
- Demangle View: Demangle

The assembly code listing is as follows:

```
31    add eax, eax
32    add edx, 1
33    cmp edi, edx
34    jg .L7
35    ret
36 .L11:
37    ret
38 .L8:
39    xor eax, eax
40    ret
41 .L9:
42    xor edx, edx
43    xor eax, eax
44    jmp .L7
45 .LC0:
46    .long 0
47    .long 1
48    .long 2
49    .long 3
50 .LC1:
51    .long 4
52    .long 4
53    .long 4
54    .long 4
```

# cppcon 2017

- That's it for the talks I liked



# Overall Impression of the Conference

# Overall Impression of the Conference

- Reflection/generation as language features

# Overall Impression of the Conference

- Reflection/generation as language features
  - What we're going towards is hygienic macros

# Overall Impression of the Conference

- Reflection/generation as language features
  - What we're going towards is hygienic macros
  - But *not*, since C++ isn't (and shouldn't be) homoiconic

# Overall Impression of the Conference

- Reflection/generation as language features
  - What we're going towards is hygienic macros
  - But *not*, since C++ isn't (and shouldn't be) homoiconic
  - Meta-programming to build domain-tailored APIs

# Overall Impression of the Conference

- Reflection/generation as language features
  - What we're going towards is hygienic macros
  - But *not*, since C++ isn't (and shouldn't be) homoiconic
- Meta-programming to build domain-tailored APIs
  - Templates, SFINAE, constexpr, traits, constexpr if...

# Overall Impression of the Conference

- Reflection/generation as language features
  - What we're going towards is hygienic macros
  - But *not*, since C++ isn't (and shouldn't be) homoiconic
- Meta-programming to build domain-tailored APIs
  - Templates, SFINAE, constexpr, traits, constexpr if...
- Make everything go faster

# Overall Impression of the Conference

- Reflection/generation as language features
  - What we're going towards is hygienic macros
  - But *not*, since C++ isn't (and shouldn't be) homoiconic
- Meta-programming to build domain-tailored APIs
  - Templates, SFINAE, constexpr, traits, constexpr if...
- Make everything go faster
  - Measurement is *assumed* but how often *overlooked*?

# Overall Impression of the Conference

- Reflection/generation as language features
  - What we're going towards is hygienic macros
  - But *not*, since C++ isn't (and shouldn't be) homoiconic
- Meta-programming to build domain-tailored APIs
  - Templates, SFINAE, constexpr, traits, constexpr if...
- Make everything go faster
  - Measurement is *assumed* but how often *overlooked*?
- Concurrency still treated as a niche

# Overall Impression of the Conference

- Reflection/generation as language features
  - What we're going towards is hygienic macros
  - But *not*, since C++ isn't (and shouldn't be) homoiconic
- Meta-programming to build domain-tailored APIs
  - Templates, SFINAE, constexpr, traits, constexpr if...
- Make everything go faster
  - Measurement is *assumed* but how often *overlooked*?
- Concurrency still treated as a niche
  - No more free lunch, blah blah, co-routines, blah blah

# Overall Impression of the Conference

- Reflection/generation as language features
  - What we're going towards is hygienic macros
  - But *not*, since C++ isn't (and shouldn't be) homoiconic
- Meta-programming to build domain-tailored APIs
  - Templates, SFINAE, constexpr, traits, constexpr if...
- Make everything go faster
  - Measurement is *assumed* but how often *overlooked*?
- Concurrency still treated as a niche
  - No more free lunch, blah blah, co-routines, blah blah



#define

I aspire to a  
more perfect  
enum

```
enum Color { e_RED, e_GREEN, e_PURPLE };
```

```
Color favorite = e_PURPLE;
```

```
cout << "I like " << favorite;
```

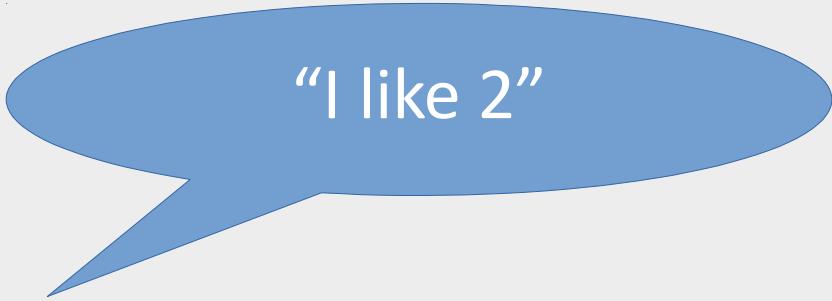
```
Color boring = Color::e_RED;
```

```
enum Color { e_RED, e_GREEN, e_PURPLE };
```

```
Color favorite = e_PURPLE;
```

```
cout << "I like " << favorite;
```

```
Color boring = Color::e_RED;
```



"I like 2"

```
enum Color { e_RED, e_GREEN, e_PURPLE };
```

```
Color favorite = e_PURPLE;
```

```
cout << "I like " << favorite;
```

```
Color boring = Color::e_RED;
```

“I like 2”

“Error: WTF is Color::e\_RED?”

```
enum class Color { e_RED, e_GREEN, e_PURPLE } ;  
  
Color favorite = Color::e_PURPLE;  
  
cout << "I like " << favorite;
```

```
enum class Color { e_RED, e_GREEN, e_PURPLE };  
  
Color favorite = Color::e_PURPLE;  
  
cout << "I like " << favorite;
```



“I like 2”

```
struct Color {  
    enum Value { e_RED, e_GREEN, e_PURPLE };  
};  
  
Color favorite = Color::e_PURPLE;
```

```
struct Color {  
    enum Value { e_RED, e_GREEN, e_PURPLE };  
};  
  
Color favorite = Color::e_PURPLE;
```

Error: No conversion

```
struct Color {  
    enum Value { e_RED, e_GREEN, e_PURPLE };
```

```
};
```

```
Color favorite = Color::e_PURPLE;
```

Error: No conversion

```
Color::Value best = Color::e_PURPLE;
```

```
cout << "I like " << best;
```

```
struct Color {  
    enum Value { e_RED, e_GREEN, e_PURPLE };
```

```
};
```

```
Color favorite = Color::e_PURPLE;
```

Error: No conversion

```
Color::Value best = Color::e_PURPLE;
```

```
cout << "I like " << best;
```

“I like 2”

## *Part One*

bas\_codegen.pl

/home/dgoffred/git/bas-codegen ! \$ \

```
/home/dgoffred/git/bas-codegen ! $ \
> git ls-files '*.*' | \
```

```
/home/dgoffred/git/bas-codegen ! $ \
> git ls-files '*.*' | \
> sed 's/.*\.\.\([^\.]\+\)\$/\1/' | \
```

```
/home/dgoffred/git/bas-codegen ! $ \
> git ls-files '*.*' | \
> sed 's/.*\.\.\([^\.]\+\)\$/\1/' | \
> sort | \
```

```
/home/dgoffred/git/bas-codegen ! $ \
> git ls-files '*.*' | \
> sed 's/.*\.\.\([^\.]\+\)\$/\1/' | \
> sort | \
> uniq -c | \
```

```
/home/dgoffred/git/bas-codegen ! $ \
> git ls-files '*.*' | \
> sed 's/.*\.\.\([^\.]\+\)\$/\1/' | \
> sort | \
> uniq -c | \
> sort -r
```

```
/home/dgoffred/git/bas-codegen ! $ \
> git ls-files '*.*' | \
> sed 's/.*\.\.\([^.]\+\)\$/\1/' | \
> sort | \
> uniq -c | \
> sort -r

 90 pm          8 dd
 58 t           6 pl
 57 mk          6 h
 57 mem         6 d
 57 dep          5 sundev1
 57 cap          5 defs
 52 xml          2 md
 19 opts         1 pod
 18 gitignore    1 ini
 10 xsd          1 depends
 10 sh
 10 cpp
```

```
/home/dgoffred/git/bas-codegen ! $ cat \
```

```
/home/dgoffred/git/bas-codegen ! $ cat \
> <(git ls-files '*.pl') \
```

```
/home/dgoffred/git/bas-codegen ! $ cat \
> <(git ls-files '*.pl') \
> <(git ls-files '*.pm') \
```

```
/home/dgoffred/git/bas-codegen ! $ cat \
> <(git ls-files '*.pl') \
> <(git ls-files '*.pm') \
> <(git ls-files '*.t') | \
```

```
/home/dgoffred/git/bas-codegen ! $ cat \
> <(git ls-files '*.pl') \
> <(git ls-files '*.pm') \
> <(git ls-files '*.t') | \
> xargs wc -l | \
```

```
/home/dgoffred/git/bas-codegen ! $ cat \
> <(git ls-files '*.pl') \
> <(git ls-files '*.pm') \
> <(git ls-files '*.t') | \
> xargs wc -l | \
> tail -1
```

```
/home/dgoffred/git/bas-codegen ! $ cat \
> <(git ls-files '*.pl') \
> <(git ls-files '*.pm') \
> <(git ls-files '*.t') | \
> xargs wc -l | \
> tail -1
68761 total
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:bdem="http://bloomberg.com/schemas/bdem">

    <xs:simpleType name="Color">
        <xs:restriction base="xs:string">
            <xs:enumeration value="RED"      bdem:id="0" />
            <xs:enumeration value="GREEN"    bdem:id="1" />
            <xs:enumeration value="PURPLE"   bdem:id="2" />
        </xs:restriction>
    </xs:simpleType>

</xs:schema>
```

**bas-codegen XSD**

# Basic Application Services XSD

# Basic Application Services XML Schema Definition

# Basic Application Services Extensible Markup Language Schema Definition

# Basic Application Services Extensible Markup Language Schema Definition Language

# Bloomberg Application Services Extensible Markup Language Schema Definition Language

# BAS Application Services Extensible Markup Language Schema Definition Language

# BAS Application Services

## Application Services Extensible

### Markup Language Schema

#### Definition

##### Language

BAS Application Services  
Application Services Application  
Services Extensible Markup  
Language Schema Definition  
Language

BAS Application Services  
Application Services Application  
Services Application Services  
Extensible Markup Language  
Schema Definition  
Language

BAS Application Services  
Application Services Application  
Services Application Services  
Application Services Extensible  
Markup Language Schema  
Definition  
Language

BAS Application Services  
Application Services Application  
Services Application Services  
Application Services Application  
Services Extensible Markup  
Language Schema Definition  
Language

BAS Application Services  
Application Services Application  
Services Application Services  
Application Services Application  
Services Application Services  
Extensible Markup Language  
Schema Definition  
Language

BAS Application Services  
Application Services Application  
Services Application Services  
Application Services Application  
Services Application Services  
Application Services Extensible  
Markup Language Schema  
Definition  
Language

BAS Application Services  
Application Services Application  
Services Application Services  
Application Services Application  
Services Application Services  
Application Services Application  
Services Extensible Markup  
Language Schema Definition  
Language

BAS Application Services  
Application Services Application  
Services Application Services  
Application Services Application  
Services Application Services  
Application Services Application  
Services Application Services  
Extensible Markup Language  
Schema Definition  
Language

BAS Application Services Application  
Services Application Services  
Application Services Application  
Services Application Services  
Application Services Application  
Services Application Services  
Application Services Application  
Services Extensible Markup Language  
Schema Definition  
Language

BAS Application Services Application  
Services Application Services  
Extensible Markup Language Schema  
Definition  
Language

BAS Application Services Application  
Services Application Services  
Application Services Extensible Markup  
Language Schema Definition  
Language

BAS Application Services Application  
Services Extensible Markup Language  
Schema Definition  
Language

BAS Application Services Application  
Services Application Services Extensible  
Markup Language Schema Definition  
Language

BAS Application Services Application  
Services Extensible Markup Language  
Schema Definition  
Language

BAS Application Services Application  
Services Application Services Application  
Markup Language Schema Definition  
Language

BAS Application Services Application  
Services Application Services Application  
Extensible Markup Language  
Schema Definition  
Language

BAS Application Services Application  
Services Application Services Application  
Markup Language Schema Definition  
Language

BAS Application Services Application Services

Extensible Markup Language Schema Definition  
Language

BAS Application Services Application Services  
Application Services Extensible Markup  
Language Schema Definition  
Language

BAS Application Services Application Services  
Extensible Markup Language Schema Definition  
Language

BAS Application Services Application Services  
Application Services Extensible Markup  
Language Schema Definition  
Language

BAS Application Services Application Services

Extensible Markup Language Schema Definition  
Language





**Windows**

exception 0E has occurred at 0028:C562F1B7 in UXD\_ctpci9x(0  
853. The current application will be terminated.

Press any key to terminate the current application.  
Press CTRL+ALT+DEL again to restart your computer. You will  
lose any unsaved information in all applications.

Press any key to continue \_

1T1-14-3030V]H> ED1oA1119 dL7AL9t8i 801"3 L0/05118501/6  
UioS0R+~^83?3t+89?JuTC&97uBi 341 H0tBa>e?ZY[ii0]-+ UioW, 6A Yn(03A  
i>B &iH>31fPi&0?>u 8ell 8*i*u+i>1HAL8e& i  
d4Gj H&A+9 8u) NEUL+JuE&i" 8iB\_ n0k93+  
rJ0egkiH+~+t P8+PL ~IASJ u98i+> 818 K [~Tu#0e+ , 69 Yn(X48 6 3H  
u+&e&9J u8c=3 D1/0G1 l8c+1 8g+8T\_818i+> 2EJ u3V9ti .8i69 .iBB ;+,e6H  
;+ut&e69 0U &i+  
rt:3|832 tt92 t>Se[2 2phi7Pi 4-777-89" 3\_8+G+Bt+8i[8+8t9d7R& w88U+S  
8904008 w & w95+3|ht7Pi8i d u+89g+8i=81+> 818  
rt13|832 tt92 t  
241-3418723860+0u"89w u-LJ u98iG&|G4=288&eG 848eA 3H7-78  
43003 tLi^4|9K u+SR0871A SP32o6+ u+8k0i 4308i0V]H=+ 9+8iX7>ai9u6.  
EUli94A1-31t721818 471"41120+0 u8 c B 86 0 82M6314D<10081B 98









```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:bdem="http://bloomberg.com/schemas/bdem">

    <xs:simpleType name="Color">
        <xs:restriction base="xs:string">
            <xs:enumeration value="RED"      bdem:id="0" />
            <xs:enumeration value="GREEN"    bdem:id="1" />
            <xs:enumeration value="PURPLE"   bdem:id="2" />
        </xs:restriction>
    </xs:simpleType>

</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:bdem="http://bloomberg.com/schemas/bdem">

    <xs:simpleType name="Color">
        <xs:restriction base="xs:string">
            <xs:enumeration value="RED"      bdem:id="0" />
            <xs:enumeration value="GREEN"    bdem:id="1" />
            <xs:enumeration value="PURPLE"   bdem:id="2" />
        </xs:restriction>
    </xs:simpleType>

</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:bdem="http://bloomberg.com/schemas/bdem">

    <xs:simpleType name="Color">
        <xs:restriction base="xs:string">
            <xs:enumeration value="RED"      bdem:id="0" />
            <xs:enumeration value="GREEN"    bdem:id="1" />
            <xs:enumeration value="PURPLE"   bdem:id="2" />
        </xs:restriction>
    </xs:simpleType>

</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:bdem="http://bloomberg.com/schemas/bdem">

    <xs:simpleType name="Color">
        <xs:restriction base="xs:string">
            <xs:enumeration value="RED"      bdem:id="0" />
            <xs:enumeration value="GREEN"    bdem:id="1" />
            <xs:enumeration value="PURPLE"   bdem:id="2" />
        </xs:restriction>
    </xs:simpleType>

</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:bdem="http://bloomberg.com/schemas/bdem">

    <xs:simpleType name="Color">
        <xs:restriction base="xs:string">
            <xs:enumeration value="RED"      bdem:id="0" />
            <xs:enumeration value="GREEN"    bdem:id="1" />
            <xs:enumeration value="PURPLE"   bdem:id="2" />
        </xs:restriction>
    </xs:simpleType>

</xs:schema>
```

```
$ bas_codegen.pl \
```

```
$ bas_codegen.pl \
> --mode msg \
> --msgExpand \
> --noAggregateConversion \
> --noExternalization \
> --package msgs \
```

```
$ bas_codegen.pl \
> --mode msg \
> --msgExpand \
> --noAggregateConversion \
> --noExternalization \
> --package msgs \
> color.xsd
```

```
$ bas_codegen.pl \
> --mode msg \
> --msgExpand \
> --noAggregateConversion \
> --noExternalization \
> --package msgs \
> color.xsd
```

The schema is valid

Generating ./msgs\_color.cpp

Generating ./msgs\_color.h

```
$ wc -l msgs_color.{h,cpp}
```

```
$ wc -l msgs_color.{h,cpp}
167 msgs_color.h
119 msgs_color.cpp
286 total
```

```
$ less msgs_color.{h,cpp}
```

```
$ less msgs_color.{h,cpp}
```

```
// msgs_color.h          *DO NOT EDIT*      @generated -*-C++-*-  
#ifndef INCLUDED_MSGS_COLOR  
#define INCLUDED_MSGS_COLOR  
  
#ifndef INCLUDED_BDES_IDENT  
#include <bdes_ident.h>  
#endif  
BDES_IDENT_RCSID(msgs_color_h, "$Id$ $CSID$")  
BDES_IDENT_PRAGMA_ONCE  
  
//@PURPOSE: Provide value-semantic attribute classes  
//  
//@AUTHOR: David GOFFREDO (dgoffredo@bloomberg.net)  
  
#ifndef INCLUDED_BSLALG_TYPETRAITS  
#include <bslalg_typetraits.h>  
#endif  
  
#ifndef INCLUDED_BDEAT_ATTRIBUTEINFO  
#include <bdeat_attributeinfo.h>  
#endif  
  
#ifndef INCLUDED_BDEAT_ENUMERATORINFO  
#include <bdeat_enumeratorinfo.h>  
#endif
```

```
$ less msgs_color.{h,cpp}

#ifndef INCLUDED_BDEAT_TYPETRAITS
#include <bdeat_typetraits.h>
#endif

#ifndef INCLUDED_BSLS_ASSERT
#include <bsls_assert.h>
#endif

#ifndef INCLUDED_BSL_IOSFWD
#include <bsl_iosfwd.h>
#endif

#ifndef INCLUDED_BSL_OSTREAM
#include <bsl_ostream.h>
#endif

#ifndef INCLUDED_BSL_STRING
#include <bsl_string.h>
#endif
```

```
$ less msgs_color.{h,cpp}
```

```
namespace BloombergLP {

namespace msgs {

        // =====
        // class Color
        // =====

struct Color {

public:
    // TYPES
    enum Value {
        RED      = 0
    , GREEN    = 1
    , PURPLE  = 2
    } ;

    enum {
        NUM_ENUMERATORS = 3
    } ;

    // CONSTANTS
    static const char CLASS_NAME[];

    static const bdeat_EnumeratorInfo ENUMERATOR_INFO_ARRAY[];
}
```

```
$ less msgs_color.{h,cpp}
```

```
// CLASS METHODS

static const char *toString(Value value);
    // Return the string representation exactly matching the enumerator
    // name corresponding to the specified enumeration 'value'.


static int fromString(Value      *result,
                      const char   *string,
                      int          stringLength);
    // Load into the specified 'result' the enumerator matching the
    // specified 'string' of the specified 'stringLength'.  Return 0 on
    // success, and a non-zero value with no effect on 'result' otherwise
    // (i.e., 'string' does not match any enumerator).

static int fromString(Value      *result,
                      const bsl::string& string);
    // Load into the specified 'result' the enumerator matching the
    // specified 'string'.  Return 0 on success, and a non-zero value with
    // no effect on 'result' otherwise (i.e., 'string' does not match any
    // enumerator).

static int fromInt(Value *result, int number);
    // Load into the specified 'result' the enumerator matching the
    // specified 'number'.  Return 0 on success, and a non-zero value with
    // no effect on 'result' otherwise (i.e., 'number' does not match any
    // enumerator).
```

```
$ less msgs_color.{h,cpp}

static bsl::ostream& print(bsl::ostream& stream, Value value);
    // Write to the specified 'stream' the string representation of
    // the specified enumeration 'value'.  Return a reference to
    // the modifiable 'stream'.
};

// FREE OPERATORS
inline
bsl::ostream& operator<<(bsl::ostream& stream, Color::Value rhs);
    // Format the specified 'rhs' to the specified output 'stream' and
    // return a reference to the modifiable 'stream'.

} // close package namespace

// TRAITS

BDEAT DECL ENUMERATION TRAITS(msgs::Color)

// =====
//           INLINE FUNCTION DEFINITIONS
// =====
```

```
$ less msgs_color.{h,cpp}
```

```
namespace msgs {  
  
    // -----  
    // class Color  
    // -----  
  
// CLASS METHODS  
inline  
int Color::fromString(Value *result, const bsl::string& string)  
{  
    return fromString(result,  
                      string.c_str(),  
                      static_cast<int>(string.length()));  
}  
  
inline  
bsl::ostream& Color::print(bsl::ostream& stream,  
                           Color::Value value)  
{  
    return stream << toString(value);  
}  
} // close package namespace
```

```
$ less msgs_color.{h,cpp}

// FREE FUNCTIONS

inline
bsl::ostream& msgs::operator<<(
    bsl::ostream& stream,
    msgs::Color::Value rhs)
{
    return msgs::Color::print(stream, rhs);
}

} // close enterprise namespace
#endif

// GENERATED BY BLP_BAS_CODEGEN_3.8.26 Fri Oct 13 21:00:46 2017
// USING bas_codegen.pl --mode msg --msgExpand -noAggregateConversion
--noExternalization --package msgs generated.xsd
// -----
// NOTICE:
//     Copyright (C) Bloomberg L.P., 2017
//     All Rights Reserved.
//     Property of Bloomberg L.P. (BLP)
//     This software is made available solely pursuant to the
//     terms of a BLP license agreement which governs its use.
// ----- END-OF-FILE -----
```

```
$ less msgs_color.{h,pp}
```

```
// msgs_color.cpp
```

```
*DO NOT EDIT*
```

```
@generated -*-C++-*-
```

```
#include <bdes_ident.h>
BDES_IDENT_RCSID(msgs_color_cpp, "$Id$ $CSID$")
```

```
#include <msgs_color.h>
```

```
#include <bdeat_formattingmode.h>
#include <bdeat_valuetypefunctions.h>
#include <bdeu_print.h>
#include <bdeu_printmethods.h>
#include <bdeu_string.h>
```

```
#include <bdlb_print.h>
#include <bslim_printer.h>
#include <bsls_assert.h>
```

```
#include <bsl_iomanip.h>
#include <bsl_limits.h>
#include <bsl_ostream.h>
```

```
namespace BloombergLP {
namespace msgs {
```

```
$ less msgs_color.{h,cpp}

// -----
// class Color
// -----


// CONSTANTS

const char Color::CLASS_NAME[] = "Color";

const bdeat_EnumeratorInfo Color::ENUMERATOR_INFO_ARRAY[] = {
{
    Color::RED,
    "RED",
    sizeof("RED") - 1,
    ""
},
{
    Color::GREEN,
    "GREEN",
    sizeof("GREEN") - 1,
    ""
},
{
    Color::PURPLE,
    "PURPLE",
    sizeof("PURPLE") - 1,
    ""
}
};
```

```
$ less msgs_color.{h,cpp}

// CLASS METHODS

int Color::fromInt(Color::Value *result, int number)
{
    switch (number) {
        case Color::RED:
        case Color::GREEN:
        case Color::PURPLE:
            *result = (Color::Value)number;
            return 0;
        default:
            return -1;
    }
}
```

```
$ less msgs_color.{h,cpp}
```

```
int Color::fromString(
    Color::Value *result,
    const char      *string,
    int              stringLength)
{
    for (int i = 0; i < 3; ++i) {
        const bdeat_EnumeratorInfo& enumeratorInfo =
            Color::ENUMERATOR_INFO_ARRAY[i];

        if (stringLength == enumeratorInfo.d_nameLength
        && 0 == bsl::memcmp(enumeratorInfo.d_name_p, string, stringLength))
        {
            *result = (Color::Value)enumeratorInfo.d_value;
            return 0;
        }
    }

    return -1;
}
```

```
$ less msgs_color.{h,cpp}
```

```
const char *Color::toString(Color::Value value)
{
    switch (value) {
        case RED: {
            return "RED";
        } break;
        case GREEN: {
            return "GREEN";
        } break;
        case PURPLE: {
            return "PURPLE";
        } break;
    }

    BSLS_ASSERT(!"invalid enumerator");
    return 0;
}

} // close package namespace
} // close enterprise namespace
```

```
$ less msgs_color.{h,cpp}
```

```
// GENERATED BY BLP_BAS_CODEGEN_3.8.26 Fri Oct 13 21:00:46 2017
// USING bas_codegen.pl --mode msg --msgExpand --noAggregateConversion --package
msgs generated.xsd
// -----
// NOTICE:
//     Copyright (C) Bloomberg L.P., 2017
//     All Rights Reserved.
//     Property of Bloomberg L.P. (BLP)
//     This software is made available solely pursuant to the
//     terms of a BLP license agreement which governs its use.
// ----- END-OF-FILE -----
```

```
#include <msgs_color.h>

Color::Value favorite = Color::PURPLE;

cout << "I like " << favorite;
```

```
#include <msgs_color.h>  
  
Color::Value favorite = Color::PURPLE;  
  
cout << "I like " << favorite;
```



“I like PURPLE”

```
Color favorite;
cin >> favorite;

if (!cin) {
    cerr << "That's not a color.";
    return 1;
}

if (favorite == Color::e_PURPLE) {
    cout << "Good choice.";
}
else {
    cout << favorite << " is in poor taste.";
}
```

```
Color favorite;
cin >> favorite;

if (!cin) {
    cerr << "That's not a color.";
    return 1;
}

if (favorite == Color::e_PURPLE) {
    cout << "Good choice.";
}
else {
    cout << favorite << " is in poor taste.";
}
```

“RED is in poor taste”

```
// ... insert magic here ...

Color favorite;
cin >> favorite;

if (!cin) {
    cerr << "That's not a color.";
    return 1;
}

if (favorite == Color::e_PURPLE) {
    cout << "Good choice.";
}
else {
    cout << favorite << " is in poor taste.";
}
```



“RED is in poor taste”

```
bslmc::enumeration Color { RED, GREEN, PURPLE } ;  
  
Color favorite;  
cin >> favorite;  
  
if (!cin) {  
    cerr << "That's not a color.";  
    return 1;  
}  
  
if (favorite == Color::e_PURPLE) {  
    cout << "Good choice.";  
}  
else {  
    cout << favorite << " is in poor taste.";  
}
```



“RED is in poor taste”

```
DEFINE_ENUM(Color, RED, GREEN, PURPLE);  
  
Color favorite;  
cin >> favorite;  
  
if (!cin) {  
    cerr << "That's not a color.";  
    return 1;  
}  
  
if (favorite == Color::e_PURPLE) {  
    cout << "Good choice."  
}  
else {  
    cout << favorite << " is in poor taste."  
}
```



“RED is in poor taste”

## *Part Two*

```
DEFINE_ENUM(NAME, ...);
```

## *Part Two*

```
DEFINE_ENUM(NAME, ...);
```

*Journey to the Center of the C++*

When I type this:

When I type this:

```
DEFINE_ENUM(Color, RED, GREEN, PURPLE);
```

I want it to expand to this:

I want it to expand to this:

```
class Color {
public:
    enum Value { e_RED, e_GREEN, e_PURPLE };

    enum { k_NUM_VALUES = 3 };

    static const StringRef(&names()) [k_NUM_VALUES] // { ... }

private:
    Value d_value;

public:
    Color() // { .. }
    Color(Value) // { ... }

    operator Value() const // { ... }

    StringRef toString() const // { ... }
    friend ostream& operator<<(ostream&, const Color&) // { ... }

    int fromString(const StringRef&) // { ... }
    friend ostream& operator>>(ostream&, Color&) // { ... }

};
```

I want it to expand to this:

```
class Color {  
public:  
    enum Value { e_RED, e_GREEN, e_PURPLE };  
  
    enum { k_NUM_VALUES = 3 };  
  
    static const StringRef(&names[]);  
  
private:  
    Value d_value;  
  
public:  
    Color() // { .. }  
    Color(Value) // { ... }  
  
    operator Value() const // { ... }  
  
    StringRef toString() const // { ... }  
    friend ostream& operator<<(ostream&, const Color&) // { ... }  
  
    int fromString(const StringRef&) // { ... }  
    friend ostream& operator>>(ostream&, Color&) // { ... }  
};
```



constants with the e\_

where the strings are in the names () like this:

where the strings are in the names () like this:

```
static const StringRef(&names()) [k_NUM_VALUES]
{
    static const StringRef(*dataPtr) [k_NUM_VALUES] = 0;

BSLMT_ONCE_DO
{
    static const StringRef data[] = {
        "RED", "GREEN", "PURPLE"
    };
    dataPtr = &data;
}

return *dataPtr;
}
```

where the strings are in the names () like this:

```
static const StringRef(&names()) [k_NUM_VALUES]
{
    static const StringRef(*dataPtr) [k_NUM_VALUES] = 0;

BSLMT_ONCE_DO
{
    static const StringRef data[] = {
        "RED", "GREEN", "PURPLE"
    };
    dataPtr = &data;
}
return *dataPtr;
}
```



each of the arguments “quoted”



Say you want to overload a macro on arity, so that:

Say you want to overload a macro on arity, so that:

**MY\_MACRO(arg1);**

**MY\_MACRO(arg1, arg2);**

**MY\_MACRO(arg1, arg2, arg3);**

invoke three totally different macro definitions.

Say you want to overload a macro on arity, so that:

**MY\_MACRO(arg1);**

**MY\_MACRO(arg1, arg2);**

**MY\_MACRO(arg1, arg2, arg3);**

invoke three totally different macro definitions.

Can that be done?

**YES**

# YES

*but you're not going to like it*

A macro that expands to *how many arguments* were passed to it:

A macro that expands to *how many arguments* were passed to it:

**NUM\_ARGS**(foo, bar) → 2

#define **NUM\_ARGS**(...) // ... ?

A macro that expands to *how many arguments* were passed to it:

**NUM\_ARGS**(foo, bar) → 2

#define **NUM\_ARGS**(...) // ... ?

How can such a macro be defined?

For any finite maximum supported arity (say, 3):

For any finite maximum supported arity (say, 3):

```
#define NUM_ARGS_HELPER(A3, A2, A1, N, ...) \
    N

#define NUM_ARGS(...) \
    NUM_ARGS_HELPER(__VA_ARGS__, 3, 2, 1)
```

For any finite maximum supported arity (say, 3):

```
#define NUM_ARGS_HELPER(A3, A2, A1, N, ...) \
    N

#define NUM_ARGS(...) \
    NUM_ARGS_HELPER(__VA_ARGS__, 3, 2, 1)
```

Then the expansion of

```
NUM_ARGS(foo, bar)
```

For any finite maximum supported arity (say, 3):

```
#define NUM_ARGS_HELPER(A3, A2, A1, N, ...) \
    N

#define NUM_ARGS(...) \
    NUM_ARGS_HELPER(__VA_ARGS__, 3, 2, 1)
```

Then the expansion of

`NUM_ARGS(foo, bar)`

is 2, because:

```
NUM_ARGS_HELPER(foo, bar, 3, 2, 1)
// NUM_ARGS_HELPER(A3, A2, A1, N, ...) N
```

Then you can have:

Then you can have:

```
#define MY_MACRO_1(A1) "one"
#define MY_MACRO_2(A1, A2) "two"
#define MY_MACRO_3(A1, A2, A3) "three"

#define MY_MACRO_HELPER_(N, ...) \
MY_MACRO_##N(__VA_ARGS__)

#define MY_MACRO_HELPER(N, ...) \
MY_MACRO_HELPER_(N, __VA_ARGS__)

#define MY_MACRO(...) \
MY_MACRO_HELPER(NUM_ARGS(__VA_ARGS__), \
__VA_ARGS__)
```

and so

```
cout << MY_MACRO(foo, bar);
```

and so

```
cout << MY_MACRO(foo, bar);
```

expands to

```
cout << "two";
```



For our enumeration class, we need to turn this

**RED , GREEN , PURPLE**

For our enumeration class, we need to turn this

**RED, GREEN, PURPLE**

into this

**e\_RED, e\_GREEN, e\_PURPLE**

For our enumeration class, we need to turn this

**RED, GREEN, PURPLE**

into this

**e\_RED, e\_GREEN, e\_PURPLE**

and also into this

**"RED", "GREEN", "PURPLE"**

We can use token pasting for the enum literals:

```
#define ENUMIFY(x) e_ ## x
```

We can use token pasting for the enum literals:

```
#define ENUMIFY(X) e_ ## X
```

and we can use token stringification for the strings:

```
#define QUOTE(X) #X
```

We can use token pasting for the enum literals:

```
#define ENUMIFY(X) e_ ## X
```

and we can use token stringification for the strings:

```
#define QUOTE(X) #X
```

except due to the order of operations we need helpers:

```
#define ENUMIFY_(X) e_ ## X
#define ENUMIFY(X) ENUMIFY_(X)
```

```
#define QUOTE_(X) #X
#define QUOTE(X) QUOTE_(X)
```

# That's because of the way the pre-processor works:



The relevant steps of macro expansion are (per C 2011 [n1570] 6.10.3.1 and C++ 1998 16.3.1):



1. Process tokens that are preceded by `#` or `##`.
2. Apply macro replacement to each argument.
3. Replace each parameter with the corresponding result of the above macro replacement.
4. Rescan for more macros.



Thus, with `xstr(foo)`, we have:

1. The replacement text, `str(s)`, contains no `#` or `##`, so nothing happens.
2. The argument `foo` is replaced with `4`, so it is as if `xstr(4)` had been used.
3. In the replacement text `str(s)`, the parameter `s` is replaced with `4`, producing `str(4)`.
4. `str(4)` is rescanned. (The resulting steps produce `"4"`.)

<https://stackoverflow.com/a/16990634>

```
#define QUOTE_(X) #X
#define QUOTE(X) QUOTE_(X)
```

# That's because of the way the pre-processor works:



The relevant steps of macro expansion are (per C 2011 [n1570] 6.10.3.1 and C++ 1998 16.3.1):



1. Process tokens that are preceded by `#` or `##`.
2. Apply macro replacement to each argument.
3. Replace each parameter with the corresponding result of the above macro replacement.
4. Rescan for more macros.



Thus, with `xstr(foo)`, we have:

1. The replacement text, `str(s)`, contains no `#` or `##`, so nothing happens.
2. The argument `foo` is replaced with `4`, so it is as if `xstr(4)` had been used.
3. In the replacement text `str(s)`, the parameter `s` is replaced with `4`, producing `str(4)`.
4. `str(4)` is rescanned. (The resulting steps produce `"4"`.)

<https://stackoverflow.com/a/16990634>

```
#define QUOTE_(X) #X
#define QUOTE(X) QUOTE_(X)
```

# That's because of the way the pre-processor works:



The relevant steps of macro expansion are (per C 2011 [n1570] 6.10.3.1 and C++ 1998 16.3.1):

41

1. Process tokens that are preceded by `#` or `##`.
2. Apply macro replacement to each argument.
3. Replace each parameter with the corresponding result of the above macro replacement.
4. Rescan for more macros.



Thus, with `xstr(foo)`, we have:

1. The replacement text, `str(s)`, contains no `#` or `##`, so nothing happens.
2. The argument `foo` is replaced with `4`, so it is as if `xstr(4)` had been used.
3. In the replacement text `str(s)`, the parameter `s` is replaced with `4`, producing `str(4)`.
4. `str(4)` is rescanned. (The resulting steps produce `"4"`.)

<https://stackoverflow.com/a/16990634>

```
#define QUOTE_(X) #X
#define QUOTE(X) QUOTE_(X)
```

So now, for example, this:

```
cout << QUOTE(RED) << ENUMIFY(RED);
```

So now, for example, this:

```
cout << QUOTE(RED) << ENUMIFY(RED);
```

expands to:

```
cout << "RED" << e_RED;
```

# GREAT

# GREAT

*now how do we operate on lists of arguments?*

To avoid repeating ourselves, let's write MAP:

To avoid repeating ourselves, let's write MAP:

```
#define MAP(MACRO, LIST) // ?
```

To avoid repeating ourselves, let's write MAP:

```
#define MAP(MACRO, LIST) // ?
```

so, for example, these invocations:

```
MAP(QQUOTE, (foo, bar, baz))  
MAP(ENUMIFY, (foo, bar, baz))
```

To avoid repeating ourselves, let's write MAP:

```
#define MAP(MACRO, LIST) // ?
```

so, for example, these invocations:

```
MAP(QQUOTE, (foo, bar, baz))  
MAP(ENUMIFY, (foo, bar, baz))
```

would expand to these expressions:

```
"foo", "bar", "baz"  
e_foo, e_bar, e_baz
```

The idea is to use this algorithm:

The idea is to use this algorithm:

```
(define (map func items)
  (prepend (func (first items))
           (map func (rest items)))))
```

The idea is to use this algorithm:

```
(define (map func items)
  (prepend (func (first items))
           (map func (rest items)))))
```

except we don't have recursion in the pre-processor,

The idea is to use this algorithm:

```
(define (map func items)
  (prepend (func (first items))
           (map func (rest items)))))
```

except we don't have recursion in the pre-processor,  
so we'll have to write a macro for each arity n:

The idea is to use this algorithm:

```
(define (map func items)
  (prepend (func (first items))
           (map func (rest items)))))
```

except we don't have recursion in the pre-processor,  
so we'll have to write a macro for each arity n:

```
(define (mapn func items)
  (prepend (func (first items))
           (mapn-1 func (rest items)))))
```

Here's how it looks in the pre-processor:

Here's how it looks in the pre-processor:

```
#define EXPAND(X) X
```

```
#define REST(X, ...) (_VA_ARGS_)
```

```
#define FIRST(X, ...) (X)
```

```
#define MAP_3(MACRO, LIST) \
    EXPAND(MACRO FIRST LIST) \
    , MAP_2(MACRO, REST LIST)
```

Let's do an example looking at only the first part:

Let's do an example looking at only the first part:

**MAP\_3** (QUOTE , (killer , macro , stuff) )

Let's do an example looking at only the first part:

**MAP\_3 (QUOTE , (killer, macro, stuff) )**

Let's do an example looking at only the first part:

MAP\_3 (QUOTE , (killer, macro, stuff) )

EXPAND (QUOTE FIRST (killer, macro, stuff) )

Let's do an example looking at only the first part:

MAP\_3 (QUOTE , (killer, macro, stuff) )

EXPAND (QUOTE FIRST (killer, macro, stuff) )

Let's do an example looking at only the first part:

MAP\_3 (QUOTE , (killer, macro, stuff) )

EXPAND (QUOTE FIRST (killer, macro, stuff) )

EXPAND (QUOTE (killer) )

Let's do an example looking at only the first part:

MAP\_3 (QUOTE , (killer, macro, stuff) )

EXPAND (QUOTE FIRST (killer, macro, stuff) )

EXPAND (QUOTE (killer) )

Let's do an example looking at only the first part:

MAP\_3 (QUOTE , (killer, macro, stuff) )

EXPAND (QUOTE FIRST (killer, macro, stuff) )

EXPAND (QUOTE (killer) )

EXPAND ("killer")

Let's do an example looking at only the first part:

MAP\_3 (QUOTE , (killer, macro, stuff) )

EXPAND (QUOTE FIRST (killer, macro, stuff) )

EXPAND (QUOTE (killer) )

EXPAND ("killer")

Let's do an example looking at only the first part:

MAP\_3 (QUOTE , (killer, macro, stuff) )

EXPAND (QUOTE FIRST (killer, macro, stuff) )

EXPAND (QUOTE (killer) )

EXPAND ("killer")

"killer"

Let's do an example looking at only the first part:

MAP\_3 (QUOTE , (killer, macro, stuff) )

EXPAND (QUOTE FIRST (killer, macro, stuff) )

EXPAND (QUOTE (killer) )

EXPAND ("killer")

"killer"

and then the rest of the definition gives us:

Let's do an example looking at only the first part:

MAP\_3 (QUOTE , (killer, macro, stuff) )

EXPAND (QUOTE FIRST (killer, macro, stuff) )

EXPAND (QUOTE (killer) )

EXPAND ("killer")

"killer"

and then the rest of the definition gives us:

"killer" , MAP\_2 (QUOTE , (macro, stuff) )

Now combine that with the `NUM_ARGS` trick,

Now combine that with the `NUM_ARGS` trick,  
and you've got a multi-arity `MAP` macro:

Now combine that with the NUM\_ARGS trick,  
and you've got a multi-arity MAP macro:

```
#define MAP_(N, MACRO, LIST) \
    MAP_##N(MACRO, LIST)
```

```
#define MAP_(N, MACRO, LIST) \
    MAP__(N, MACRO, LIST)
```

```
#define MAP(MACRO, LIST) \
    MAP_(NUM_ARGS LIST, MACRO, LIST)
```

# TADA

# TADA

*now how do we apply this to `DEFINE_ENUM`?*

```
#define DEFINE_ENUM(NAME, . . .)

class NAME {
public:
    enum Value { MAP(ENUMIFY, (__VA_ARGS__)) } ;

    enum { k_NUM_VALUES = NUM_ARGS(__VA_ARGS__) } ;

    static const StringRef(&names()) [k_NUM_VALUES]

    ...
}
```

```
#define DEFINE_ENUM(NAME, ...)

class NAME {
public:
    enum Value { MAP(ENUMIFY, __VA_ARGS__)) } ;
    enum { k_NUM_VALUES = NUM_ARGS(__VA_ARGS__) } ;
    static const StringRef(&names()) [k_NUM_VALUES]
...
}
```

```
#define DEFINE_ENUM(NAME, ...)

class NAME {
public:
    enum Value { MAP(ENUMIFY, __VA_ARGS__)) } ;
    enum { k_NUM_VALUES = NUM_ARGS(__VA_ARGS__) } ;
    static const StringRef(&names()) [k_NUM_VALUES]
};

...
```

```
#define DEFINE_ENUM(NAME, ...)

class NAME {
public:
    enum Value { MAP(ENUMIFY, __VA_ARGS__)) };
    enum { k_NUM_VALUES = NUM_ARGS(__VA_ARGS__) };
    static const StringRef(&names()) [k_NUM_VALUES]
};

...
```

```
static const StringRef(&names()) [k_NUM_VALUES]
{
    static const StringRef (*Ptr) [k_NUM_VALUES] = 0;

BSLMT_ONCE_DO
{
    static const StringRef data[] = {
        MAP(QQUOTE, (__VA_ARGS__))
    };
    Ptr = &data;
}

return *Ptr;
}
```

```
static const StringRef(&names()) [k_NUM_VALUES]
{
    static const StringRef(*Ptr) [k_NUM_VALUES] = 0;

BSLMT_ONCE_DO
{
    static const StringRef data[] = {
        MAP(QQUOTE, __VA_ARGS__)
    };
    Ptr = &data;
}

return *Ptr;
}
```



Now this is working code:

Now this is working code:

```
DEFINE_ENUM(Color, RED, GREEN, PURPLE);

Color favorite;
cin >> favorite;

if (!cin) {
    cerr << "That's not a color.";
    return 1;
}

if (favorite == Color::e_PURPLE) {
    cout << "Good choice.";
}
else {
    cout << favorite << " is in poor taste.";
}
```

And the macro can appear where generated code cannot:

And the macro can appear where generated code cannot:

```
class Session {
public:
    // PUBLIC TYPES
    DEFINE_ENUM(Rcode,
        SUCCESS,
        BAD_ADDRESS,
        DISCONNECT,
        BAD_MESSAGE,
        TIMEOUT);

    // MANIPULATORS
    Rcode open(...);
};
```

But there's still something missing...

# But there's still something missing...

```
static bsl::ostream& print(bsl::ostream& stream, Value value);
    // Write to the specified 'stream' the string representation of
    // the specified enumeration 'value'.  Return a reference to
    // the modifiable 'stream'.
};

// FREE OPERATORS
inline
bsl::ostream& operator<<(bsl::ostream& stream, Color::Value rhs);
    // Format the specified 'rhs' to the specified output 'stream' and
    // return a reference to the modifiable 'stream'.

} // close package namespace

// TRAITS

BDEAT DECL ENUMERATION TRAITS(msgs::Color)
```

# But there's still something missing...

```
static bsl::ostream& print(bsl::ostream& stream, Value value);
    // Write to the specified 'stream' the string representation of
    // the specified enumeration 'value'.  Return a reference to
    // the modifiable 'stream'.
};

// FREE OPERATORS
inline
bsl::ostream& operator<<(bsl::ostream& stream, Color::Value rhs);
    // Format the specified 'rhs' to the specified output 'stream' and
    // return a reference to the modifiable 'stream'.

} // close package namespace

// TRAITS
BDEAT_DECL_ENUMERATION_TRAITS(msgs::Color)
```

Remember this?

*Part Three*

# Codecs

*Part Three*

# Codecs

*Getting `bdlat` to work with `DEFINE_ENUM`*

I want to be able to write this:

I want to be able to write this:

```
DEFINE_ENUM(Color, RED, GREEN, PURPLE);  
  
istringstream input(  
    "["RED\", \"GREEN\", \"PURPLE\"]");  
  
vector<Color> colors;  
baljson::Decoder().decode(input, &colors);  
  
ostringstream output;  
baljson::Encoder().encode(output, colors);  
  
assert(output.str() == input.str());
```



# **bdlat** enumerations

A type **Enum** is a **bdlat** enumeration if:

# **bdlat** enumerations

A type **Enum** is a **bdlat** enumeration if:

- overloads of the **bdlat\_enumfunctions** exist for **Enum** and are accessible via argument dependent lookup, and

# **bdlat** enumerations

A type **Enum** is a **bdlat** enumeration if:

- overloads of the **bdlat\_enumfunctions** exist for **Enum** and are accessible via argument dependent lookup, and
- the **IsEnumeration** trait is specialized for **Enum** in the **bdlat\_EnumFunctions** namespace.

# **bdlat** enumerations

These are the functions to overload:

# bdlat enumerations

These are the functions to overload:

```
int bdlat_enumFromInt(Enum *result,  
                      int number);
```

```
int bdlat_enumFromString(Enum *result,  
                        const char *string,  
                        int stringLength);
```

```
void bdlat_enumToInt(int *result,  
                     const Enum& value);
```

```
void bdlat_enumToString(string *result,  
                        const Enum& value);
```

# bdlat enumerations

This trait is to be specialized for **TYPE=Enum**:

# bdlat enumerations

This trait is to be specialized for **TYPE=Enum**:

```
namespace BloombergLP {  
namespace bdlat_EnumFunctions {  
  
template <typename TYPE>  
struct IsEnumeration;  
  
}  
}
```

# bdlat enumerations

Like this:

```
namespace BloombergLP {  
namespace bdlat_EnumFunctions {  
  
template <>  
struct IsEnumeration<Enum>  
{  
    enum { VALUE = 1 } ;  
};  
}  
}
```

# **bdlat** enumerations

This is how **bas\_codegen.pl** handles the trait:

# bdlat enumerations

This is how `bas_codegen.pl` handles the trait:

```
#define BDLAT_DECL_ENUMERATION_TRAITS( ClassName )
```

**Value:**

```
namespace bs1mf {  
    template <>  
    struct IsBitwiseMoveable<ClassName::Value> : bs1::true_type { };  
}  
template <>  
struct bdlat_IsBasicEnumeration<ClassName::Value> : bs1::true_type { };  
template <>  
struct bdlat_BasicEnumerationWrapper<ClassName::Value> : ClassName {  
    typedef ClassName Wrapper;  
};
```

# bdlat enumerations

This is how `bas_codegen.pl` handles the trait:

```
#define BDLAT_DECL_ENUMERATION_TRAITS( ClassName )
```

**Value:**

```
namespace bs1mf {  
    template <>  
    struct IsBitwiseMoveable<ClassName::Value> : bs1::true_type { };  
}  
template <>  
struct bdlat_IsBasicEnumeration<ClassName::Value> : bs1::true_type { };  
template <>  
struct bdlat_BasicEnumerationWrapper<ClassName::Value> : ClassName {  
    typedef ClassName Wrapper;  
};
```

There's that macro from the generated code

# bdlat enumerations

But `bas_codegen.pl` can pick its namespace

```
#define BDLAT_DECL_ENUMERATION_TRAITS( ClassName )
```

# bdlat enumerations

But `bas_codegen.pl` can pick its namespace

```
#define BDLAT_DECL_ENUMERATION_TRAITS( ClassName )
```

and `DEFINE_ENUM` cannot,

# bdlat enumerations

But `bas_codegen.pl` can pick its namespace

```
#define BDLAT_DECL_ENUMERATION_TRAITS( ClassName )
```

and `DEFINE_ENUM` cannot,

so we'll have to do something else.

# **bdlat** enumerations

Let's go back to the functions

# bdlat enumerations

Let's go back to the functions

```
int bdlat_enumFromInt(Enum *result,  
                      int number);
```

```
int bdlat_enumFromString(Enum *result,  
                        const char *string,  
                        int stringLength);
```

```
void bdlat_enumToInt(int *result,  
                     const Enum& value);
```

```
void bdlat_enumToString(string *result,  
                        const Enum& value);
```

# bdlat enumerations

We can't include these in the macro,

```
int bdlat_enumFromInt(Enum *result,  
                      int number);
```

```
int bdlat_enumFromString(Enum *result,  
                        const char *string,  
                        int stringLength);
```

```
void bdlat_enumToInt(int *result,  
                     const Enum& value);
```

```
void bdlat_enumToString(string *result,  
                        const Enum& value);
```

# bdlat enumerations

We can't include these in the macro,

int bdlat\_enumFromInt(Enum \*result,  
because they could end up in a class scope.

```
int bdlat_enumFromString(Enum *result,
                         const char *string,
                         int stringLength);
```

```
void bdlat_enumToInt(int *result,
                      const Enum& value);
```

```
void bdlat_enumToString(string *result,
                        const Enum& value);
```

# bdlat enumerations

We can't include these in the macro,

```
int bdlat_enumFromInt(Enum *result,
```

because they could end up in a class scope.

```
int bdlat_enumFromString(Enum *result,
```

Instead, how do we define these functions

```
void bdlat_enumToInt(int *result,  
                     const Enum& value);
```

```
void bdlat_enumToString(string *result,  
                        const Enum& value);
```

# INHERITANCE

# INHERITANCE

*let's try it*

# bdlat enumerations

Say each class created by **DEFINE\_ENUM**

had a common base class **EnumBase**:

```
int bdlat_enumFromInt(Enum      *result,
                      int       number);

int bdlat_enumFromString(Enum      *result,
                         const char *string,
                         int       stringLength);

void bdlat_enumToInt(int      *result,
                     const Enum& value);

void bdlat_enumToString(string      *result,
                        const Enum& value);
```

# bdlat enumerations

Say each class created by **DEFINE\_ENUM**

```
int bdlat_enumFromInt(Enum *result,  
                      int number);
```

had a common base class **EnumBase**:

```
#define DEFINE_ENUM(NAME, ...) \  
class NAME : public EnumBase { \  
public: \  
    void bdlat_enumToInt(int *result,  
                          const Enum& value); \  
    void bdlat_enumToString(string *result,  
                           const Enum& value);
```

# bdlat enumerations

Then we could generally define these functions:

```
int bdlat_enumFromInt(EnumBase *result,  
                      int      number);  
  
int bdlat_enumFromString(EnumBase *result,  
                         const char *string,  
                         int          stringLength);  
  
void bdlat_enumToInt(int           *result,  
                     const EnumBase& value);  
  
void bdlat_enumToString(string        *result,  
                        const EnumBase& value);
```



# **bdlat** enumerations

and...

# **bdlat** enumerations

and...

the functions would know nothing about the  
actual enumerations



# bdlat enumerations

The base class needs knowledge of the derived

class

```
int bdlat_enumFromInt(Enum      *result,
                      int       number);

int bdlat_enumFromString(Enum      *result,
                         const char *string,
                         int       stringLength);

void bdlat_enumToInt(int      *result,
                     const Enum& value);

void bdlat_enumToString(string      *result,
                        const Enum& value);
```

# bdlat enumerations

The base class needs knowledge of the derived

class

```
int bdlat_enumFromInt(Enum *result,  
                      int number);
```

Can it be done?

```
int bdlat_enumFromString(Enum *result,  
                         const char *string,  
                         int stringLength);
```

```
void bdlat_enumToInt(int *result,  
                     const Enum& value);
```

```
void bdlat_enumToString(string *result,  
                        const Enum& value);
```

# The Curiously Recurring Template Pattern

# The Curiously Recurring Template Pattern

*let's try it*

# bdlat enumerations

Each class **Enum** created by **DEFINE\_ENUM**

```
int bdlat_enumFromInt(Enum *result,  
                      int number);  
  
int bdlat_enumFromString(Enum *result,  
                         const char *string,  
                         int stringLength);  
  
void bdlat_enumToInt(int *result,  
                     const Enum& value);  
  
void bdlat_enumToString(string *result,  
                        const Enum& value);
```

# bdlat enumerations

Each class **Enum** created by **DEFINE\_ENUM**

```
int bdlat_enumFromInt(Enum      *result,
                      int       number);
```

has a common base class **EnumBase<Enum>**:

```
#define DEFINE_ENUM(NAME, ...)
class NAME : public EnumBase<NAME> {
    void bdlat_enumToInt(int      *result,
                         const Enum& value);
    void bdlat_enumToString(string   *result,
                            const Enum& value);
```

# bdlat enumerations

Then we could generally define these functions:

# bdlat enumerations

Then we could generally define these functions:

```
template <typename ENUM>
int bdlat_enumFromInt(EnumBase<ENUM> *result,
                      int number);
```

...

# bdlat enumerations

Then we could generally define these functions:

```
template <typename ENUM>
int bdlat_enumFromInt(EnumBase<ENUM> *result,
                      int number)

{
    if (number < 0 || number >= ENUM::k_NUM_VALUES)
        return -1;

    ENUM& derived = static_cast<ENUM&>(*result);
    derived = ENUM::Value(number);
    return 0;
}
```

# SUCCESS

# SUCCESS

*Now what about that trait...*

# bdlat enumerations

How can we specialize this trait  
for all classes derived from `EnumBase<T>`?

```
namespace BloombergLP {  
namespace bdlat_EnumFunctions {  
  
template <typename TYPE>  
struct IsEnumeration;  
  
}  
}
```

# bdlat enumerations

How can we specialize this trait  
for all classes derived from **EnumBase<T>**?

---

```
namespace BloombergLP {  
namespace bdlat_EnumFunctions {  
  
template <typename TYPE>  
struct IsEnumeration;  
  
}  
}
```

# bdlat enumerations

Does the compiler accept this?

```
namespace BloombergLP {  
namespace bdlat_EnumFunctions {  
  
template <  
    typename T,  
    typename=enable_if<  
        has_enum_base<T>::value>::type>  
struct IsEnumeration {  
    ...  
}
```

**NO**

# NO

*Error: **IEnumeration** previously defined with **one** template parameter, but here is defined with **two***



# The Curiously Recurring Template Pattern

# ~~The Curiously Recurring Template Pattern~~

# ~~The Curiously Recurring Template Pattern~~

*not going to work*

# bdlat enumerations

What are we going to do about this trait?

```
namespace BloombergLP {  
namespace bdlat_EnumFunctions {  
  
template <typename TYPE>  
struct IsEnumeration;  
  
}  
}
```



# **bdlat** enumerations

What if each type generated by **DEFINE\_ENUM** were  
an instantiation of some template, **BasicEnum**?

# bdlat enumerations

What if each type generated by `DEFINE_ENUM` were an instantiation of some template, `BasicEnum`?

```
template <typename UTIL>
class BasicEnum {
    typename UTIL::Value d_value;
public:
    BasicEnum(typename UTIL::Value)
    ...
};
```

# bdlat enumerations

Then we could have this:

```
#define DEFINE_ENUM(NAME, . . .) \
class NAME ## _Util { \
    . . . \
}; \
typedef BasicEnum<NAME ## _Util> NAME
```

# **bdlat enumerations**

For example, this:

```
DEFINE_ENUM(Color, RED, GREEN, PURPLE);
```

# bdlat enumerations

For example, this:

```
DEFINE_ENUM(Color, RED, GREEN, PURPLE);
```

would expand to this:

```
class Color_Util {  
    ...  
};
```

```
typedef BasicEnum<Color_Util> Color;
```

# bdlat enumerations

Then the trait is just a partial specialization

# bdlat enumerations

Then the trait is just a partial specialization

```
namespace BloombergLP {
namespace bdlat_EnumFunctions {

template <typename UTIL>
struct IsEnumeration<BasicEnum<UTIL> >

{
    enum { VALUE = 1 } ;
} ;

} }
```

# bdlat enumerations

The functions are similar to before:

```
template <typename UTIL>
int bdlat_enumFromInt(BasicEnum<UTIL> *result,
                      int number);
```

...

# bdlat enumerations

The functions are similar to before:

```
template <typename UTIL>
int bdlat_enumFromInt(BasicEnum<UTIL> *result,
                      int number);

{
    if (number < 0 || number >= UTIL::k_NUM_VALUES)
        return -1;

    *result = typename UTIL::Value(number);
    return 0;
}
```

**IT WORKS**

This is now working code:

This is now working code:

```
DEFINE_ENUM(Color, RED, GREEN, PURPLE);

stringstream input(
    "["RED\", \"GREEN\", \"PURPLE\"]");

vector<Color> colors;
baljsn::Decoder().decode(input, &colors);

ostringstream output;
baljsn::Encoder().encode(output, colors);

assert(output.str() == input.str());
```



We've created a macro

```
DEFINE_ENUM(Color, RED, GREEN, PURPLE);
```

that expands to the definition of a type that:

We've created a macro

```
DEFINE_ENUM(Color, RED, GREEN, PURPLE);
```

that expands to the definition of a type that:

- behaves like a built-in `enum`,

We've created a macro

```
DEFINE_ENUM(Color, RED, GREEN, PURPLE);
```

that expands to the definition of a type that:

- behaves like a built-in **enum**,
- doesn't export its constant values,

We've created a macro

```
DEFINE_ENUM(Color, RED, GREEN, PURPLE);
```

that expands to the definition of a type that:

- behaves like a built-in **enum**,
- doesn't export its constant values,
- can be inserted into and extracted from streams,

We've created a macro

```
DEFINE_ENUM(Color, RED, GREEN, PURPLE);
```

that expands to the definition of a type that:

- behaves like a built-in `enum`,
- doesn't export its constant values,
- can be inserted into and extracted from streams,
- can appear in namespace, class, or function scope,

We've created a macro

```
DEFINE_ENUM(Color, RED, GREEN, PURPLE);
```

that expands to the definition of a type that:

- behaves like a built-in `enum`,
- doesn't export its constant values,
- can be inserted into and extracted from streams,
- can appear in namespace, class, or function scope,
- is compatible with `bdlat`-based codecs,

We've created a macro

```
DEFINE_ENUM(Color, RED, GREEN, PURPLE);
```

that expands to the definition of a type that:

- behaves like a built-in `enum`,
- doesn't export its constant values,
- can be inserted into and extracted from streams,
- can appear in namespace, class, or function scope,
- is compatible with `bdlat`-based codecs,
- adheres to BDE typographical conventions, and

We've created a macro

```
DEFINE_ENUM(Color, RED, GREEN, PURPLE);
```

that expands to the definition of a type that:

- behaves like a built-in `enum`,
- doesn't export its constant values,
- can be inserted into and extracted from streams,
- can appear in namespace, class, or function scope,
- is compatible with `bdlat`-based codecs,
- adheres to BDE typographical conventions, and
- requires no external tool or language.

But wouldn't this have been easier?

But wouldn't this have been easier?

```
bslmc::enumeration Color {  
    RED,  
    GREEN,  
    PURPLE  
};
```

fin