LIGHTNING TALKS
ACCU2018
Thursday 12th April

electricity is, really just organized lightning?
— George Carlin
THE RULES

subjects are open!
five minutes (max)
have fun
Peter Sommerlad - FOOL
Michel Grootjans - Crafting Guitars
Rob Smallshire - The Gender Equality Paradox
Florian Gilcher - Trains
Graham Haynes - On Automati
Marshall Clow - Fuzzing Your Code
Chris Oldwood - The Far Side
Jon Kalb - This is Why We Can't Have Nice Things
Phil Nash - East All The Things
Jim Hague - A Brief of one-line abuses
Mike Seymour - Sparsity Parsery
SLASH
OR
DASH
Peter Sommerlad - FOOL
Michel Grootjans - Crafting Guitars
Rob Smallshire - The Gender Equality Paradox
Florian Gilcher - Trains
Graham Haynes - On Automati
Marshall Clow - Fuzzing Your Code
Chris Oldwood - The Far Side
Jon Kalb - This is Why We Can't Have Nice Things
Phil Nash - East All The Things
Jim Hague - A Brief of one-line abuses
Mike Seymour - Sparsity Parsery
{The Problem

```cpp
std::vector v{1,2,3,4,5,6};
auto p=accumulate(begin(v), end(v), 1, std::multiplies<>{});
```

looks very ugly: `<>{}`

or: `<>()`

or: `<int>{}`:}
What else wrong in <functional>?

• Many operators are missing:
  unary operators: *, &, +
  member access: .*, ->*
  shifts, assignments, ternary, …
• Arity is fixed: 1 or 2
• Must instantiate objects…
A Better Way!

P0984R0 - Functions Objects Obsoleted by Lambdas

---

**Document Number:** P0984R0  
**Date:** 2018-04-01  
**Project:** Programming Language C++  
**Audience:** EWG/LEWG  
**Target:** C++20


```cpp
// unary operators
constexpr inline auto Deref = see below;  // unary *
constexpr inline auto Address = see below;  // unary &
constexpr inline auto Negate = see below;  // unary -
constexpr inline auto Posate = see below;  // unary +
constexpr inline auto Not = see below;  // ! not
constexpr inline auto Cmpl = see below;  // ~ cmpl

// left associative binary operators
constexpr inline auto PtrMemb = see below;  // ->*
constexpr inline auto RefMemb = see below;  // .*

constexpr inline auto Plus = see below;  // +
constexpr inline auto Minus = see below;  // -
constexpr inline auto Times = see below;  // *
constexpr inline auto Divide = see below;  // /
constexpr inline auto Remainder = see below;  // &
```
More FOOL!

```cpp
constexpr inline auto Equal = see below; // ==
using equalTea = decltype(Equal); // to replace equal_to<>;
constexpr inline auto Not_eq = see below; // !=
constexpr inline auto Bigger = see below; // >
using moreTea = decltype(Bigger); // to replace greater<>;
constexpr inline auto Smaller = see below; // <
using lessTea = decltype(Smaller); // to replace less<>;
constexpr inline auto Maybe_bigger = see below; // >=
constexpr inline auto Sometimes_smaller = see below; // <=
constexpr inline auto Spaceship = see below; // <=>
constexpr inline auto And = see below; // && and
constexpr inline auto Or = see below; // || or

constexpr inline auto Bitand = see below; // & bitand
constexpr inline auto Bitor = see below; // | bitor
constexpr inline auto Xor = see below; // ^ xor
constexpr inline auto Lshift = see below; // right associative binary operators
constexpr inline auto Rshift = see below; //
```
Further Usage

```cpp
auto endl=static_cast<std::ostream&(*)(std::ostream&>)(std::endl);
Lshift(std::cout,"Hello","World!",endl,"The Answer is: ",6*7,endl);

std::set<int,lessTee> s({3,1,4,1,5,9,2,6},Smaller); // no C++20 compiler
std::set<int,lessTee> s({3,1,4,1,5,9,2,6}); // C++20 allows creating from decltype(lambda)
```
See below?

• Lambda constexpr variable
• Variadic Lambda
• Deduced Noexcept
• Deduced Return Type
• Fold Expression
• No specializations
• No overloads
How?

```cpp
std::vector v{1,2,3,4,5,6};
auto res=accumulate(begin(v),end(v),1,Times);
```
std::vector v{1,2,3,4,5,6};
auto res=accumulate(begin(v),end(v),1,Times);

variadic lambdas & folds
How?

```cpp
std::vector v{1,2,3,4,5,6};
auto res=accumulate(begin(v),end(v),1,Times);
```

variadic lambdas & folds

```cpp
constexpr auto Times=[](auto&&... l) constexpr
```
std::vector v{1,2,3,4,5,6};
auto res=accumulate(begin(v),end(v),1,Times);

```cpp
constexpr auto Times=[](auto&&... l) constexpr noexcept(noexcept((... * std::declval<decltype(l)>())))
```

variadic lambdas & folds
How?

```cpp
std::vector v{1,2,3,4,5,6};
auto res=accumulate(begin(v),end(v),1,Times);
```

`std::vector v{1,2,3,4,5,6};`  
`auto res=accumulate(begin(v),end(v),1,Times);`  

**variadic lambdas & folds**

```cpp
constexpr auto Times=[](auto&&... l) constexpr noexcept(noexcept((... * std::declval<decltype(l)>())))  
-> decltype((... * std::forward<decltype(l)>(l) ))
```
How?

```cpp
std::vector v{1,2,3,4,5,6};
auto res=accumulate(begin(v),end(v),1,Times);
```

**variadic lambdas & folds**

```cpp
constexpr auto Times=[](auto&&... l) constexpr
  noexcept(noexcept((... * std::declval<decltype(l)>())))
-> decltype((... * std::forward<decltype(l)>(l) ))
{
```

```cpp
} // End of variadic lambdas & folds
```
How?

```cpp
test::vector v{1,2,3,4,5,6};
auto res=accumulate(begin(v),end(v),1,Times);
```

**variadic lambdas & folds**

```cpp
constexpr auto Times=[](auto&&... l) constexpr noexcept(noexcept((... * std::declval<decltype(l)>())))
-> decltype((... * std::forward<decltype(l)>(l) ))
{
    return (...*std::forward<decltype(l)>(l));
}
```
How?

```cpp
std::vector v{1,2,3,4,5,6};
auto res=accumulate(begin(v),end(v),1,Times);
```

**variadic lambdas & folds**

```cpp
constexpr auto Times=[](auto&&... l) constexpr noexcept(noexcept((... * std::declval<decltype(l)>())))
    -> decltype((... * std::forward<decltype(l)>(l) ))
{
    return (...*std::forward<decltype(l)>(l));
};
```
Unary Ops:
Unary Ops:

```cpp
constexpr inline auto Posate = [] (auto&& l) constexpr
```
Unary Ops:

```cpp
constexpr inline auto Posate = [](auto&& l) constexpr noexcept(noexcept(+ std::declval<decltype(l)>())))
```
Unary Ops:

```cpp
constexpr inline auto Posate = [](auto&& l) constexpr noexcept(noexcept(+ std::declval<decltype(l)>()))
    -> decltype(+ std::forward<decltype(l)>(l))
```

Unary Ops:

```cpp
constexpr inline auto Posate = [] (auto&& l) constexpr noexcept(noexcept(+ std::declval<decltype(l)>()()))
  -> decltype(+ std::forward<decltype(l)>(l))
{

```
Unary Ops:

```cpp
constexpr inline auto Posate = [](auto&& l) constexpr noexcept(noexcept(+ std::declval<decltype(l)>()))
  -> decltype(+ std::forward<decltype(l)>(l))
  {
      return + std::forward<decltype(l)>(l);
  }
```
Unary Ops:

```cpp
constexpr inline auto Posate = [] (auto&& l) constexpr
    noexcept(noexcept( + std::declval< decltype(l) >()))
    -> decltype(+ std::forward< decltype(l) >(l))
{
    return + std::forward< decltype(l) >(l);
};
```
Ternary Op:
Ternary Op:

```cpp
constexpr auto Wtf=[](auto&& c, auto&& l, auto&& r) constexpr
```

```cpp
constexpr auto Wtf=[](auto&& c, auto&& l, auto&& r) constexpr
```
Ternary Op:

```cpp
constexpr auto Wtf=[](auto&& c, auto&& l, auto&& r) constexpr noexcept(noexcept(std::declval<decltype(c)>()?std::declval<decltype(l)>():std::declval<decltype(r)>()))
```
Ternary Op:

```cpp
constexpr auto Wtf = [](auto&& c, auto&& l, auto&& r) constexpr
    noexcept(noexcept(std::declval<decltype(c)>()?std::declval<decltype(l)>():std::declval<decltype(r)>()))
    -> decltype(std::declval<decltype(c)>()?std::declval<decltype(l)>():std::declval<decltype(r)>())
```
Ternary Op:

```cpp
constexpr auto Wtf = [](auto&& c, auto&& l, auto&& r) constexpr noexcept(noexcept(std::declval<decltype(c)>() ? std::declval<decltype(l)>() : std::declval<decltype(r)>()))
    -> decltype(std::declval<decltype(c)>() ? std::declval<decltype(l)>() : std::declval<decltype(r)>())
{

};
```
Ternary Op:

```cpp
constexpr auto Wtf = [](auto&& c, auto&& l, auto&& r) constexpr noexcept(noexcept(std::declval<decltype(c)>()?std::declval<decltype(l)>():std::declval<decltype(r)>()))
    -> decltype(std::declval<decltype(c)>()?std::declval<decltype(l)>():std::declval<decltype(r)>())
{
    return std::forward<decltype(c)>(c) ?
```

Ternary Op:

```cpp
constexpr auto Wtf = [](auto&& c, auto&& l, auto&& r) constexpr
    noexcept(noexcept(std::declval<decltype(c)>() ? std::declval<decltype(l)>() : std::declval<decltype(r)>()))
    -> decltype(std::declval<decltype(c)>() ? std::declval<decltype(l)>() : std::declval<decltype(r)>())
{
    return std::forward<decltype(c)>(c) ?
        std::forward<decltype(l)>(l) : std::forward<decltype(r)>(r);
};
```
Ternary Op:

```cpp
constexpr auto Wtf=[](auto&& c, auto&& l, auto&& r) constexpr
    noexcept(noexcept(std::declval<decltype(c)>()?std::declval<decltype(l)>():std::declval<decltype(r)>()))
    -> decltype(std::declval<decltype(c)>()?std::declval<decltype(l)>():std::declval<decltype(r)>())
    {
        return std::forward<decltype(c)>(c) ?
            std::forward<decltype(l)>(l) : std::forward<decltype(r)>(r);
    };
```
Ternary Op:

```cpp
constexpr auto Wtf=[](auto&& c, auto&& l, auto&& r) constexpr noexcept(noexcept(std::declval<decltype(c)>()?std::declval<decltype(l)>():std::declval<decltype(r)>()))
    -> decltype(std::declval<decltype(c)>()?std::declval<decltype(l)>():std::declval<decltype(r)>())
{
    return std::forward<decltype(c)>(c) ?
        std::forward<decltype(l)>(l) : std::forward<decltype(r)>(r);
};
```
Sum up

Function Objects can/should be Obsoleted by Lambdas
SLASH OR DASH
NETSTAT
DASH
The craft of building guitars

Random musings by @michelgrootjans
So... what is the difference between work and craft?
Care, attention to detail, functionality, durability
Continuous education, sharing, accepting criticism
Experimenting
Should every guitar be finely crafted?
Should every guitar be finely crafted?
It depends on what you plan to do with it
The first shitty draft ...
The throw-away guitar
/// BOTH ---
Peter Sommerlad - FOOL
Michel Grootjans - Crafting Guitars
**Rob Smallshire** - The Gender Equality Paradox
Florian Gilcher - Trains
Graham Haynes - On Automati
Marshall Clow - Fuzzing Your Code
Chris Oldwood - The Far Side
Jon Kalb - This is Why We Can't Have Nice Things
Phil Nash - East All The Things
Jim Hague - A Brief of one-line abuses
Mike Seymour - Sparsity Parsery
The Gender Equality Paradox
The Gender-Equality Paradox in Science, Technology, Engineering, and Mathematics Education

Gijsbert Stoet\(^1\) and David C. Geary\(^2\)
\(^1\)School of Social Sciences, Leeds Beckett University, and \(^2\)Department of Psychological Sciences, University of Missouri

Abstract
The underrepresentation of girls and women in science, technology, engineering, and mathematics (STEM) fields is a continual concern for social scientists and policymakers. Using an international database on adolescent achievement in science, mathematics, and reading \((N = 472,242)\), we showed that girls performed similarly to or better than boys in science in two of every three countries, and in nearly all countries, more girls appeared capable of college-level STEM study than had enrolled. Paradoxically, the sex differences in the magnitude of relative academic strengths and pursuit of STEM degrees rose with increases in national gender equality. The gap between boys’ science achievement and girls’ reading achievement relative to their mean academic performance was near universal. These sex differences in academic strengths and attitudes toward science correlated with the STEM graduation gap. A mediation analysis suggested that life-quality pressures in less gender-equal countries promote girls’ and women’s engagement with STEM subjects.
\[ N = 472242 \]
Global Gender Gap Index (GGGI)

Women Among STEM Graduates (%)
<table>
<thead>
<tr>
<th>Country</th>
<th>Science</th>
<th>Maths</th>
<th>Reading</th>
</tr>
</thead>
<tbody>
<tr>
<td>Albania</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algeria</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Australia</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Austria</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>China</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Belgium</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brazil</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bulgaria</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Canada</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chile</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Czech Republic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Denmark</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estonia</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finland</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>France</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Georgia</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Germany</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greece</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hong Kong</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hungary</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iceland</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Indonesia</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ireland</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Israel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Italy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Japan</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jordan</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Korea</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Latvia</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lebanon</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lithuania</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Luxembourg</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Macao</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Macedonia</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Malta</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mexico</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Moldova</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Montenegro</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Netherlands</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>New Zealand</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Norway</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Peru</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Portugal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Puerto Rico</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Qatar</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Romania</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Russian Federation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Singapore</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slovak Republic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slovenia</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spain</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sweden</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Switzerland</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thailand</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trinidad and Tobago</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tunisia</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turkey</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>United Arab Emirates</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>United Kingdom</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>United States</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uruguay</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vietnam</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
intra-individual achievement

Girls

Boys
intra-individual achievement

Girls

science

Boys

science
intra-individual achievement
intra-individual achievement

Girls

reading maths science

high social security
high gender equality
economic freedom
to follow interests

low social security
low gender equality
high economic premium for STEM careers
SLASH
OR
DASH
RFC 3986
UNIFORM RESOURCE IDENTIFIER (URI): GENERIC SYNTAX

/// SLASH ///
Peter Sommerlad - FOOL
Michel Grootjans - Crafting Guitars
Rob Smallshire - The Gender Equality Paradox
Florian Gilcher - Trains
Graham Haynes - On Automati
Marshall Clow - Fuzzing Your Code
Chris Oldwood - The Far Side
Jon Kalb - This is Why We Can't Have Nice Things
Phil Nash - East All The Things
Jim Hague - A Brief of one-line abuses
Mike Seymour - Sparsity Parsery
There’s an issue I’d like to speak about.
$SL(1)$
sl(1) is in danger
appreciation is sinking
distributions ship it with options allowing to stop the train (debian)
in outdated versions (debian)
I’m pretty sure someone is rebuilding it as a systemd module.
Outstanding bugs are left unfixed!
$ man sl
BUGS

It rarely shows contents of current directory.
A revival!
Rebuild your own $sl(1)$, as close to the original as possible, in your favourite language, system, whatever.
Tweet to: @argorak
I’ve got another issue to talk about!
Error messages
$ rustc --version
Rust C++ Linter, version 1.25
fn main() {
    let mut vec = vec![1,2,3];

    let foo = &vec[2];

    // copious amount of work

    vec[1] = 2;
}

error[E0502]: cannot borrow `vec` as mutable because it is also borrowed as immutable
    -> src/main.rs:8:5
     |
 4  |     let foo = &vec[2];
     |
     --- immutable borrow occurs here
...
 8  |     vec[1] = 2;
     |     ^^^ mutable borrow occurs here
 9  | }
     | - immutable borrow ends here
Great for readability,
bad for colleagues.
$ rustc --error-format verbal
Rust prides itself in bringing ideas good ideas from the past to use.
This one goes hundreds of years back!
Format:

• 2 lines for context

• 2 lines for explaining the problem

• a call to action (my designer told me to!)
$ rustc --error-format verbal vec.rs
That `vec` which you wanted to borrow
Is giving my checker much sorrow
for meanwhile you mutate
it in line eight
let’s fix it until tomorrow!
Finally! Compilers for humans!
Also: Compilers for the IoT world!
echo "Alexa, `rustc --error-format verbal vec.rs`" | say
rust-lang/rfcs/pull/2378
/// SLASH ///

(OR NO SLASH)
Peter Sommerlad - FOOL
Michel Grootjans - Crafting Guitars
Rob Smallshire - The Gender Equality Paradox
Florian Gilcher - Trains
**Graham Haynes** - On Automati
Marshall Clow - Fuzzing Your Code
Chris Oldwood - The Far Side
Jon Kalb - This is Why We Can't Have Nice Things
Phil Nash - East All The Things
Jim Hague - A Brief of one-line abuses
Mike Seymour - Sparsity Parsery
...and Ariston?

Pastiche of Bertha,
By Woodland Animations/BBC
Automate all the things?
./configure
make
#!/bin/sh

git clone http://github.com/project/my_project.git &&
    cd my_project &&
cmake . &&
make -j
make test
#!/bin/env sh -e

git clone http://github.com/project/my_project.git project

cd project

mkdir build

cd build

mkdir build

cmake ..

make

nice make -j4

make test
#!/bin/env sh -e
apt-get install -y boost
git clone https://github.com/project/my_project.git project
mkdir build
cd build
cmake ../project
nice cmake --build . -- -j4
make test
cpack -G deb
cp *.deb /releases/
#!/bin/env sh -e

curl -X POST http://build/status -d @in_progress
apt-get install -y boost
git clone https://github.com/project/my_project.git project
git submodule update --init

mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=RELWITHDEBINFO ../project

nice cmake --build . -- -j4
valgrind --tool=memcheck ./tests

cpack -G deb
cp *.deb /releases/
curl -X POST http://build/status -d @successful
#!/bin/env sh -e

curl -X POST -u username:password http://build/status -d @in_progress
apt-get install -y boost
git clone https://github.com/project/my_project.git project
git submodule update -init

for compiler in clang gcc; do
    mkdir $compiler
    cd $compiler
    cmake -DCMAKE_BUILD_TYPE=RELWITHDEBINFO -DCMAKE_EXPORT_COMPILE_COMMANDS=ON -DCMAKE_CXX_COMPILER=/usr/bin/${compiler} ../project
    nice cmake --build . -- -j4
    cd..
    done

cd clang
find ../project -name "*.cpp" -print | xargs clang-tidy
valgrind --tool=memcheck ./tests
cpack -G deb
mkdir /releases/$version
cp *.deb /releases/$version
cp *.deb /releases/$version
curl -X POST -u username:password http://build/status -d @successful
#!/bin/env sh -e
POST="curl -x POST -u username:password"
${POST} http://build/status -d @in_progress
apt-get install -y boost
git clone https://github.com/project/my_project.git project
git submodule update --init
for compiler in clang gcc; do
  for bits in 32 64; do
    mkdir ${compiler}-${bits}
cd ${compiler}-${bits}
cmake -DCMAKE_BUILD_TYPE=RELWITHDEBINFO \
      -DCMAKE_EXPORT_COMPILE_COMMANDS=ON \
      -DCMAKE_CXX_COMPILER=/usr/bin/${compiler} \
      -DCMAKE_CXX_FLAGS=-m${bits} ../project
    nice cmake --build . -- -j4
cd ..
done
done
cd clang
find ../project -name "*.cpp" -print | xargs clang-tidy
valgrind --tool=memcheck ./tests
valgrind --tool=cachegrind ./soak
cpack -G deb
mkdir -p /releases/${version}
cp *.deb /releases/${version}/
chmod 777 /releases/${version}/
${POST} http://build/status -d @successful
#!/bin/env sh -e
POST="curl -x POST -u username:password"
${POST} https://build/status -d @in_progress
${POST} https://slack/post -d "Build of ${version} in progress!"
apt-get install -y boost
git clone https://github.com/project/my_project.git project
git submodule update --init
for compiler in clang gcc; do
  for bits in 32 64; do
    BUILD_DIR="${compiler}-${bits}"
mkdir "${BUILD_DIR}"
cd "${BUILD_DIR}"
cmake -DCMAKE_BUILD_TYPE=RELWITHDEBINFO \ 
  -DCMAKE_EXPORT_COMPILE_COMMANDS=ON \ 
  -DCMAKE_CXX_COMPILER="/usr/bin/${compiler}" \ 
  -DCMAKE_CXX_FLAGS="-m${bits}" .. /project
    nice cmake --build . -- -j4
cd ..
  done
done
cd clang
find .. /project -name "*.cpp" -print | xargs clang-tidy
valgrind --tool=memcheck ./tests
valgrind --tool=cachegrind ./soak
cpack -G deb
RELEASE_DIR="/releases/${version}"
mkdir -p "${RELEASE_DIR}"
cp *.deb "${RELEASE_DIR}/*
chmod -R ugo+r "${RELEASE_DIR}"
cd ..
git log >"${RELEASE_DIR}/release_notes.txt"
${POST} https://build/status -d @successful
${POST} https://slack/post -d "Build of ${version} succeeded!"
Configure, make, done :-D
Oops – somebody cares about running the tests
Oops – somebody cares about the return code from the tests
Publish build success to Slack channel
Publish build success to Slack channel only if build succeeded
Cross-compile on half a dozen platforms
Support parallel builds
Throttle Slack channel spam
Use HTTPS. Oh certificate pain!
Set up ccache and distcc
Build on Windows too
Escape backslashes
Clean workspace between builds
Rewrite scripts in Python
Code coverage? Profile builds?
Rewrite scripts as Jenkins pipelines. Or is it Huson? Travis?
Valgrind, sanitizers, static analysis – no dodgy code shall pass!
Fix dodgy static analysis on dodgy code
git submodule for the win!
git submodule – never again!
Roll our own git submodule mechanism
Wrap all the CMake Functions!
Provision prerequisites before building
Update dependency! (This change is coupled to the other repo)
Clone before setting env or set env before cloning?
Use CPack and deploy packages
RPMs Pkgs, Debs, oh my!
Let's try Conan, it can't do any harm
Docker will solve everything
That was not the right way to do containers
Can I get back to coding again please?
Yet Another Commit Comment /yawn
INSTANT LEGACY!
Some problems...

Products can become **unmaintainable** when their **supporting infrastructure** rots

Development becomes **coupled** to automated tooling – making it hard to develop, debug, or run manually

People rely on it working - but **forget** how it works and are unfamiliar with the technologies used

Running automation is **not handed over** or is locked to users who have left

Changes **untested**? Untestable? Self-testing?

Some problems...
Considered...

**Harmful**

Automation

**Harmless**

Planet Earth
Harmful

Automation

(bad, excessive, overly complex, premature)

Harmless

Planet Earth

(mostly)
Is it **viable** (or possible) to automate?

Do I **understand** the kind of solution required?

Do I know how it could **fail**?

How can I ensure it gives me the required levels of **assurance**?
Build on the experience of others

- Draw on people's collective wisdom
- Favour mature, open and compatible tooling over esoteric and bespoke
- Use DevOps practices, e.g. Infrastructure as Code

Learn by trying it out

- You won't get it right first time
- Build up your own experience
  - Deliberate Practice

Mind the human

- Reduce the opportunity for conflict between human and automated processes
- Don't run under individuals' accounts – use an automation account to avoid single points of failure
Blib blib

Prip prip
Blib prip

Blib prip
**Design automation to evolve**

- **Refactor** for readability and (re)use
  - Comment the intent: *say what the code can't*, especially if nuanced or infrequently maintained
- Set and apply **Coding Standards**, review changes against them and combat rot
- **Test** what you can

**Exercise good coding practices**

- Don't write and forget - *let it live*, budget for it to; waterfall deliveries will rot
- Beware vendor lock-in, immature and unsupported tools – **use DSLs with care**
- **Keep it simple** - more dependencies and complexity means more frequent maintenance
Automate the solution

only once you've begun to understand the problem
SLASH OR DASH
EITHER
Peter Sommerlad - FOOL
Michel Grootjans - Crafting Guitars
Rob Smallshire - The Gender Equality Paradox
Florian Gilcher - Trains
Graham Haynes - On Automati
Marshall Clow - Fuzzing Your Code
Chris Oldwood - The Far Side
Jon Kalb - This is Why We Can't Have Nice Things
Phil Nash - East All The Things
Jim Hague - A Brief of one-line abuses
Mike Seymour - Sparsity Parsery
Fuzzing your code

Marshall Clow
Qualcomm
marshall@idio.com
What is fuzzing?

• Pass random (ish) inputs to your code, look for misbehavior. (Over and over)

• Initial fuzzers just generated random inputs

• More modern fuzzers are “guided”
Profile-guided fuzzing

• Build your program with code coverage enabled.

• After each run, the fuzzer examines the coverage data, and uses the information to generate the next test case.

• American Fuzzy Lop - http://lcamtuf.coredump.cx/afl/

• Clang libfuzzer - https://llvm.org/docs/LibFuzzer.html
OSS-Fuzz
Fuzzing as a service

• Continuous fuzzing for open source projects

• You write glue code that tells OSS-Fuzz how to test your code. It takes some data (ptr, length), and returns an int. 0 \Rightarrow \text{success}.

• You write a config that tells OSS-Fuzz how to get/build your code

• And… that’s it!
What do you get?

• Ideally … nothing!

• Whenever OSS-Fuzz finds a problem, it opens an issue in a bugzilla, and sends you an email with the details.

• After the bug is fixed, OSS-Fuzz pulls the new version, builds and tests it, and closes the bug.

• If it doesn’t get fixed, after 90 days, the bug is made public.
extern "C" int LLVMFuzzerTestOneInput
(const uint8_t *data, size_t size)
{
    std::vector<uint8_t> working(data, data + size);
    std::sort(working.begin(), working.end());

    if (!std::is_sorted(working.begin(), working.end()))
        return 1;
    if (!std::is_permutation(data, data + size, working.cbegin()))
        return 99;
    return 0;
}
Peter Sommerlad - FOOL
Michel Grootjans - Crafting Guitars
Rob Smallshire - The Gender Equality Paradox
Florian Gilcher - Trains
Graham Haynes - On Automati
Marshall Clow - Fuzzing Your Code
Chris Oldwood - The Far Side
Jon Kalb - This is Why We Can't Have Nice Things
Phil Nash - East All The Things
Jim Hague - A Brief of one-line abuses
Mike Seymour - Sparsity Parsery
The __far* Side

Chris Oldwood
Keep apart 2 chevrons
5 CLEAR 49999: BORDER 0: PAPER 0: BRIGHT 1: INK 7: CLS
10 FOR i=10 TO 20: BEEP .001+i/300,i: NEXT i: PRINT AT 10,10: INK 1: PAPER 7: BRIGHT 1:
20 FOR i=USR "a" TO USR "t"+7: READ a: POKE i,a: NEXT i
30 DATA 10,0,83,168,85,170,85,138,240,0,222,31,127,255,255,51,10,5,10,0,42,85,42,85,240,240
40 DATA 42,85,42,21,0,5,10,5,254,254,254,254,254,252,0,240,240,240,10,5,10,5,10,5,240,240,240
50 DATA 255,231,195,193,97,51,31,0,255,231,195,131,134,204,248
60 DATA 0,255,85,255,195,129,195,255,0,255,0,255,0,255,0,223,223,223,223,0,251,251,251
70 DATA 0,251,251,219,251,56,59,0,0,224,224,0,224,224,224,0,126,90,231,126,102,60,24,255,255
90 DATA 15616 TO 16383: POKE x+48384,PEEK x: NEXT x
100 FOR x=64264 TO 64479: READ y: POKE x,y: NEXT x
110 RESTORE 200: FOR x=64264 TO 64479: READ y: POKE x,y: NEXT x
200 DATA 124,254,246,254,254,246,246,0,252,254,230,252,230,254,252,0,124,254,246,240,246,250
210 DATA 124,254,240,252,240,254,124,0,124,254,240,252,240,240
211 DATA 0,124,254,240,240,246,246,246,246,246,124,0,246,246,254,254,254,246,0,254,254,56,56,56
220 DATA 30,30,30,222,222,222,254,124,0,238,254,252,248,252,254,254,254,238,0,240,240,240,240,240,254
230 DATA 198,238,254,254,254,214,214,0,124,254,254,246,246,246,246,0,124,254,238,238,238,254,254,254
240 DATA 124,254,254,254,246,246,250,124,0,124,254,230,254,252,238,0,126,254,248,124,30,254
250 DATA 246,246,246,246,246,254,254,124,0,246,246,246,246,254,124,56,0,214,214,214,254,254,238
260 DATA 222,222,254,254,30,254,252,0,254,254,62,124,248,254,254,0
280 DATA 220,220,220,0,220,220,220,0
300 RESTORE 310: FOR x=64128 TO 64207: READ y: POKE x,y: NEXT x

@chrisoldwood / gort@cix.co.uk / chrisoldwood.com
Merge in turn
THE CLOSE
SLASH OR DASH
M.1677 : INTERNATIONAL MORSE CODE

--- DASH ---
Peter Sommerlad - FOOL
Michel Grootjans - Crafting Guitars
Rob Smallshire - The Gender Equality Paradox
Florian Gilcher - Trains
Graham Haynes - On Automati
Marshall Clow - Fuzzing Your Code
Chris Oldwood - The Far Side
Jon Kalb - This is Why We Can't Have Nice Things
Phil Nash - East All The Things
Jim Hague - A Brief of one-line abuses
Mike Seymour - Sparsity Parsery
This is Why We Can’t Have Nice Things

Jon Kalb
ACCU 2018
East const

Simon Brand
@TartanLlama

Tag yourself

West-const  East-const

const T&  vs  T const&

11:16 AM - 6 Nov 2017

22 Retweets  81 Likes

34  22  81
Simplifying const Syntax

By Dan Saks, September 26, 2011

2 Comments

The simplest way to read and write const declarations correctly is to use an unconventional style

When support for the const qualifier appeared in compilers some 20 years ago, I learned to
Consistent const

The rule for const is:
Consistent `const`

The rule for `const` is:

```
const applies to what is on its left,
```
**Consistent const**

The rule for const is:

const applies to what is on its left, unless there is nothing to its left,
Consistent const

The rule for const is:

const applies to what is on its left, unless there is nothing to its left, then it applies to what’s on its right.
Consistent const

My rule for const is:
My rule for const is:

const applies to what is on its left
**Consistent const**

*My* rule for const is:

const applies to what is on its left.
Read Order

When declaring an integer const, don’t you want to read it as “a constant integer?”
Read declarations “inside out” and “right to left.”
Read Order

Read declarations “inside out” and “right to left.”

\[\text{void } (*\text{fn})(\text{int});\]
Read declarations “inside out” and “right to left.”

```
void (*fn)(int);
```

“fn is a pointer to a function that takes an int and returns void.”
Read Order

Read declarations “inside out” and “right to left.”
Read Order

Read declarations “inside out” and “right to left.”

```c
long (&ra)[3] = a;
```
Read Order

Read declarations “inside out” and “right to left.”

```
long (&ra)[3] = a;
```

“ra is a reference to an array of three longs.”
Read Order

Read declarations “inside out” and “right to left.”
Read Order

Read declarations “inside out” and “right to left.”

```c
char * const pc = str;
```
Read Order

Read declarations “inside out” and “right to left.”

`char * const pc = str;`

“pc is a constant pointer to character.”
Read Order

Read declarations “inside out” and “right to left.”
Read Order

Read declarations “inside out” and “right to left.”

```c
char const * const pc = str;
```
Read Order

Read declarations “inside out” and “right to left.”

```c
char const * const pc = str;
```

“pc is a constant pointer to constant character.”
Read Order

Read declarations “inside out” and “right to left.”
Read Order

Read declarations “inside out” and “right to left.”

```cpp
int const & ri = i;
```
Read Order

Read declarations “inside out” and “right to left.”

```cpp
int const & ri = i;
```

“ri is a reference to a constant integer.”
Required East const Cases

Constant pointers

```c
char * const pc = str;
```

Constant member functions

```c
size_type size() const;
```
What does this mean?

This case may be confusing to for programmers that don’t know the const rule.

```c
char const * pc = str;
```
What does this mean?

This case may be confusing to for programmers that expect the const to apply to the “Widget” in the alias.

```cpp
using WidgetPtr = Widget*;

const WidgetPtr wp = &w;
```
What does this mean?

This case may be confusing to for programmers that expect the const to apply to the “Widget” in the alias.

```
using WidgetPtr = Widget*;

const WidgetPtr wp = &w;
```

```
WidgetPtr const wp = &w;
```
A Foolish Consistency

The Hobgoblin of Little Minds

Ralph Waldo Emerson famously said, “A foolish consistency is the hobgoblin of little minds, adored by little statesmen and philosophers and divines.” I don’t think he was talking about code, but that statement couldn’t be more relevant to software engineers.

I’ve experienced a scenario like this a number of time in my career:

I’m sharing a new approach to writing code that offers some clear improvements to what we’ve been doing. Perhaps it is more readable, more efficient, or safer. But the response that I hear from colleagues is, “But we can’t do that here. We have <some large number> lines of code where we didn’t do it that way, so it wouldn’t be consistent.”

This is Why We Can’t Have Nice Things
NL.26: Use conventional **const** notation

**Reason**

Conventional notation is more familiar to more programmers. Consistency in large code bases.

**Example**

```cpp
const int x = 7;     // OK
int const y = 9;     // bad
const int *const p = nullptr; // OK, constant pointer to constant int
int const *const p = nullptr; // bad, constant pointer to constant int
```

**Note**

We are well aware that you could claim the "bad" examples more logical than the ones marked "OK", but they also confuse more people, especially novices relying on teaching material using the far more common, conventional OK style.

As ever, remember that the aim of these naming and layout rules is consistency and that aesthetics vary immensely.

**Enforcement**

Flag **const** used as a suffix for a type.
NL.26: Use conventional `const` notation

Reason

Conventional notation is more familiar to more programmers. Consistency in large code bases.

Example

```c
const int x = 7; // OK
int const y = 9; // bad
const int *const p = nullptr; // OK, constant pointer to constant int
int const *const p = nullptr; // bad, constant pointer to constant int
```

Note

We are well aware that you could claim the "bad" examples more logical than the ones marked "OK", but they also confuse more people, especially novices relying on teaching material using the far more common, conventional OK style.

As ever, remember that the aim of these naming and layout rules is consistency and that aesthetics vary immensely.

Enforcement

Flag `const` used as a suffix for a type.
NL.26: Use conventional \texttt{const} notation

\textbf{Reason}

Conventional notation is more familiar to more programmers. Consistency in large code bases.

\textbf{Example}

\begin{verbatim}
const int x = 7; // OK
int const y = 9; // bad
const int *const p = nullptr; // OK, constant pointer to constant int
int const *const p = nullptr; // bad, constant pointer to constant int
\end{verbatim}

\textbf{Note}

We are well aware that you could claim the "bad" examples more logical than the ones marked "OK", but they also confuse more people, especially novices relying on teaching material using the far more common, conventional OK style.

As ever, remember that the aim of these naming and layout rules is consistency and that aesthetics vary immensely.

\textbf{Enforcement}

Flag \texttt{const} used as a suffix for a type.
NL.26: Use conventional \texttt{const} notation

Reason

Conventional notation is more familiar to more programmers. Consistency in large code bases.

Example

\begin{verbatim}
const int x = 7; // OK
int const y = 9; // bad
const int *const p = nullptr; // OK, constant pointer to constant int
int const *const p = nullptr; // bad, constant pointer to constant int
\end{verbatim}

Note

We are well aware that you could claim the "bad" examples more logical than the ones marked "OK", but they also confuse more people, especially novices relying on teaching material using the far more common, conventional OK style.

As ever, remember that the aim of these naming and layout rules is consistency and that aesthetics vary immensely.

Enforcement

Flag \texttt{const} used as a suffix for a type.
NL.26: Use conventional \texttt{const} notation

Reason

Conventional notation is more familiar to more programmers. Consistency in large code bases.

Example

\begin{verbatim}
const int x = 7;       // OK
int const y = 9;       // bad
const int *const p = nullptr;  // OK, constant pointer to constant int
int const *const p = nullptr;  // bad, constant pointer to constant int
\end{verbatim}

Note

We are well aware that you could claim the "bad" examples more logical than the ones marked "OK", but they also confuse more people, especially novices relying on teaching material using the far more common, conventional OK style.

As ever, remember that the aim of these naming and layout rules is consistency and that aesthetics vary immensely.

Enforcement

Flag \texttt{const} used as a suffix for a type.
This is why we can’t have nice things
This is why we can’t have nice things

If you must remain consistent with the past, you can never improve.
Join the Revolution
Join the Revolution

Petition for const Consistency

// info
comments on c++ and issues of interest to c++ programmers
SLASH
OR
DASH
COMMENTS
COMMENTS

C++
/// SLASH ///
COMMENTS
SQL
Peter Sommerlad - FOOL
Michel Grootjans - Crafting Guitars
Rob Smallshire - The Gender Equality Paradox
Florian Gilcher - Trains
Graham Haynes - On Automati
Marshall Clow - Fuzzing Your Code
Chris Oldwood - The Far Side
Jon Kalb - This is Why We Can't Have Nice Things
Phil Nash - East All The Things
Jim Hague - A Brief of one-line abuses
Mike Seymour - Sparsity Parsery
We have always been at war with West Constia
auto someFunc( int i ) -> std::string;

// instead of

std::string someFunc( int i );
auto someFunc( int i ) const -> std::string;

// instead of

std::string someFunc( int i ) const;
template <typename Lhs, typename Rhs>
auto add( Lhs const& lhs, Rhs const& rhs ) -> decltype( lhs + rhs ) {
    return lhs + rhs;
}
auto lambda = [] -> double { return 0; }
Trailing return types everywhere
Trailing return types are an oddity in C++ – we should use them only when necessary.

A few days ago one of my coworkers asked me to explain an odd line of code he had encountered in an open source library. The line was similar to this:

```cpp
auto getMulticastHops() const -> int;
```

Some people will know that this is a way of declaring functions that came into the language with C++11. The part `-> int` is called “trailing return type”, and the line is exactly the same as

```cpp
int getMulticastHops() const;
```
Trailing return types are an oddity in C++ – we should use them only when necessary.

A few days ago one of my coworkers asked me to explain an odd line of code he had encountered in an open source library. The line was similar to this:

```cpp
auto getMulticastHops() const -> int;
```

Some people will know that this is a way of declaring functions that came into the language with C++11. The part `-> int` is called “trailing return type”, and the line is exactly the same as

```cpp
int getMulticastHops() const;
```
But...
func factorial(of : Int) -> Int
Swift

```
func factorial(of : Int) -> Int
```

Haskell

```
factorial :: (Integral a) => a -> a
```
Swift  
func factorial(of : Int) -> Int

Haskell  
factorial :: (Integral a) => a -> a

Maths  
f(x) -> y
auto doesItBlend() -> bool;
auto whatsYourFavouriteNumber() -> int;
auto add( double a, double b ) -> double;
void setTheControls();
auto doesItBlend() -> bool;
auto whatsYourFavouriteNumber() -> int;
auto add( double a, double b ) -> double;
void setTheControls();

virtual void foo();
virtual auto bar() -> int;
A Foolish Consistency

The Hobgoblin of Little Minds

Ralph Waldo Emerson famously said, “A foolish consistency is the hobgoblin of little minds, adored by little statesmen and philosophers and divines.” I don’t think he was talking about code, but I like to think of it as a metaphor for the same thing. When writing code, it’s important to be mindful of the trade-offs between consistency and flexibility. A foolhardy consistency can lead to brittle code that is difficult to maintain and evolve over time. On the other hand, too much flexibility can lead to a lack of cohesion and predictability, making it difficult to reason about the system as a whole. The key is to find a balance that strikes the right balance between these two extremes.
template <typename Lhs, typename Rhs>
auto add( Lhs const& lhs, Rhs const& rhs ) -> decltype( lhs + rhs ) {
    return lhs + rhs;
}

auto lambda = []-> double { return 0; }
#define func auto
#define func auto

#define var auto

#define let auto const
```c++
#define func auto
#define var auto
#define let auto const

func len( std::string s ) -> size_t {
    let length = s.size();
    return length;
}
```
East End Functions

6th April 2018 at 17:40

There has been a recent stirring of attention, in the C++ community, for the practice of always placing the const modifier to the right of the thing it modifies. The practice has even been gifted a catchy name: East Const (which, I think, is what has stirred up the interest).

As purely a matter of style it's fascinating that it seems to have split the community so strongly! There are cases for and against, but both sides seem to revolve around the idea of "consistency". For the East Const believers the consistency is in the sense that you can always apply one, simple, rule about what const means and where it goes. For the West Consters the consistency is with the majority of existing code out there - as
SLASH OR DASH
FRIDAY THE 13TH

/// SLASH ///
A Brief History Of Online Abuses

Jim Hague
jim.hague@acm.org
@banburybill
SPAM
Green Card Lottery 1994 May Be The Last One!
THE DEADLINE HAS BEEN ANOUNCED.

The Green Card Lottery is a completely legal program giving away a
certain annual allotment of Green Cards to persons born in certain
countries. The lottery program was scheduled to continue on a
permanent basis. However, recently, Senator Alan J Simpson
introduced a bill into the U. S. Congress which could end any future
lotteries. THE 1994 LOTTERY IS SCHEDULED TO TAKE PLACE
SOON, BUT IT MAY BE THE VERY LAST ONE.

PERSONS BORN IN MOST COUNTRIES QUALIFY, MANY FOR
FIRST TIME.

The only countries NOT qualifying are: Mexico; India; P. R. China;
Taiwan, Philippines, North Korea, Canada, United Kingdom (except
Northern Ireland), Jamaica, Dominican Republic, El Salvador and
Vietnam.

Lottery registration will take place soon. 55,000 Green Cards will be
given to those who register correctly. NO JOB IS REQUIRED.

THERE IS A STRICT JUNE DEADLINE. THE TIME TO START IS
NOW!!

For FREE information via Email, send request to
cs考察@indirect.com

--

******************************************************************************
Canter & Siegel, Immigration Attorneys
3333 E Camelback Road, Ste 250, Phoenix AZ 85018 USA
cs考察@indirect.com telephone (602)661-3911 Fax (602) 451-7617
29th March 1864
TO THE EDITOR OF THE TIMES.

Sir,—On my arrival home late yesterday evening a "telegram," by "London District Telegraph," addressed in full to me, was put into my hands. It was as follows:—

"Messrs. Gabriel, dentists, 27, Harley-street, Cavendish-square. Until October Messrs. Gabriel’s professional attendance at 27, Harley-street, will be 10 till 5."

I have never had any dealings with Messrs. Gabriel, and beg to ask by what right do they disturb me by a telegram which is evidently simply the medium of advertisement? A word from you would, I feel sure, put a stop to this intolerable nuisance. I enclose the telegram, and am,

Your faithful servant,


M. P.
Le réseau Chappe en France

Directions (date de création)
- 1793-1800
- 1800-1815
- 1815-1830
- Après 1830

Lignes (date de création)
- 1793-1800
- 1800-1815
- 1815-1830
- Après 1830
180?
Dear Sir,

I write to inform you of the sudden death of Mrs. Bourne, my former wife. She passed away peacefully in her sleep last night. I am deeply saddened by this loss and extend my condolences to her family and friends.

Mrs. Bourne was a kind and gentle soul, always ready to lend a listening ear and offer words of comfort. She had a warm heart and a loving nature, and her absence will be greatly felt by all who knew her.

I will keep you updated on any arrangements for the funeral and the wake. In the meantime, please accept my deepest sympathies.

Yours sincerely,

[Signature]

[Date]
What has been will be again, what has been done will be done again; there is nothing new under the sun.

Ecclesiastes 1:9
SLASH
OR
DASH
Peter Sommerlad - FOOL
Michel Grootjans - Crafting Guitars
Rob Smallshire - The Gender Equality Paradox
Florian Gilcher - Trains
Graham Haynes - On Automati
Marshall Clow - Fuzzing Your Code
Chris Oldwood - The Far Side
Jon Kalb - This is Why We Can't Have Nice Things
Phil Nash - East All The Things
Jim Hague - A Brief of one-line abuses
Mike Seymour - Sparsity Parsery
Sparsity Parsery

Or

Compile-time trickery for dealing with sparse key sets

Mike Seymour  
github.com/mikeseymour/wocca
Problem

- Receiving key-value tags with integer keys
- Keys cover a large range of values
- Messages might contain many tags
- We're only interested in a small, fixed subset
- Examples:
  - Music tagging (like ID3v2)
  - Financial protocols (like FIX)
Solution

• Parse the interesting tags into a small array:
  \[
  \text{parser<title,album,artist> } \ p(\text{reader});
  \]

• Read them by key, calculating the array index at compile time:
  \[
  \text{out} \ll \text{“Title: “} \ll p.\text{at<title>}(); \\
  \text{out} \ll \text{“Album: “} \ll p.\text{at<album>}(); \\
  \]
  
  \[
  p.\text{at<bpm>}(); \ // \ \text{ERROR! unspecified tag}
  \]
Basic types

- Values: perhaps a view over received data
  
  ```cpp
  using view = std::string_view;
  ```

- Tags: nullable pairs (philosophically awkward)
  ```cpp
  struct tag {
    explicit operator bool() const;
    int key;
    view value;
  };
  ```

- Reader: functor returning sequential tags
The Parser

- Contains an array of values
- Initialised from a reader
  ```cpp
  while (tag t = reader()) {
      int i = index(t.key);
      if (i >= 0) values[i] = t.value;
  }
  ```
- Read by key
  ```cpp
  static_assert(index(key) >= 0);
  return values[index(key)];
  ```
Gory details: Key sets

• A compile-time set of integer keys:

```cpp
template <int... Keys> struct keyset {
    static constexpr int keys[] {Keys...};
};
```

• Operations, including

```cpp
// gory details omitted
template <class Keys> using sort = keyset<???>;
```
Gory details: Finding the index

- Binary search in a sorted key-set's array

```cpp
using sorted = sort<keyset<Key...>>;
int first = 0, last = std::size(sorted::keys);
while (first != last) {
    int mid = first + (last-first)/2;
    if (sorted::keys[mid] == key)
        return mid;
    if (sorted::keys[mid] < key)
        first = mid+1;
    else
        last = mid;
}
return -1;
```
Gory details: Sorting the keys

template <int Key, class Keys> struct prepend_;
template <int Key, class Keys> using prepend = typename prepend_<Key, Keys>::result;

template <int Key, class Keys> struct prepend_ {using result = keyset<Key>;
};
template <int Key, int... Keys> struct prepend_<Key, keyset<Keys...>>
  {using result = keyset<Key, Keys...>;
};

template <int Key, class Keys> struct remove_;
template <int Key, class Keys> using remove = typename remove_<Key, Keys>::result;

template <int Key, class Keys> struct remove_ {using result = keyset<>;
};
template <int Key, int... Tail> struct remove_<Key, keyset<Key, Tail...>>
  {using result = keyset<Tail...>;
};
template <int Key, int Head, int... Tail> struct remove_<Key, keyset<Head, Tail...>>
  {using result = prepend<Head, remove<Key, keyset<Tail...>>>
};

template <class Keys> struct min_;
template <int Single> struct min_<keyset<Single>> {static constexpr int result = Single;
};
template <int Head, int... Tail> struct min_<keyset<Head, Tail...>>
  { static constexpr int tail = min_<keyset<Tail...>>::result;
    static constexpr int result = Head < tail ? Head : tail;
  };
template <class Keys> static constexpr int min = min_<Keys>::result;

template <class Keys> struct sort_;
template <class Keys> using sort = typename sort_<Keys>::result;

template <> struct sort_<keyset<> > {using result = keyset<>;
};
template <int... Keys> struct sort_<keyset<Keys...>>
  { static constexpr int first = min<keyset<Keys...>>;
    using result = prepend<first, sort<remove<first, keyset<Keys...>>>;
  };

SLASH
OR
DASH
C++
- > EITHER //
LISP
(NAH)
NO
SLASH OR DASH
CALC.EXE
NEITHER
THANKS!