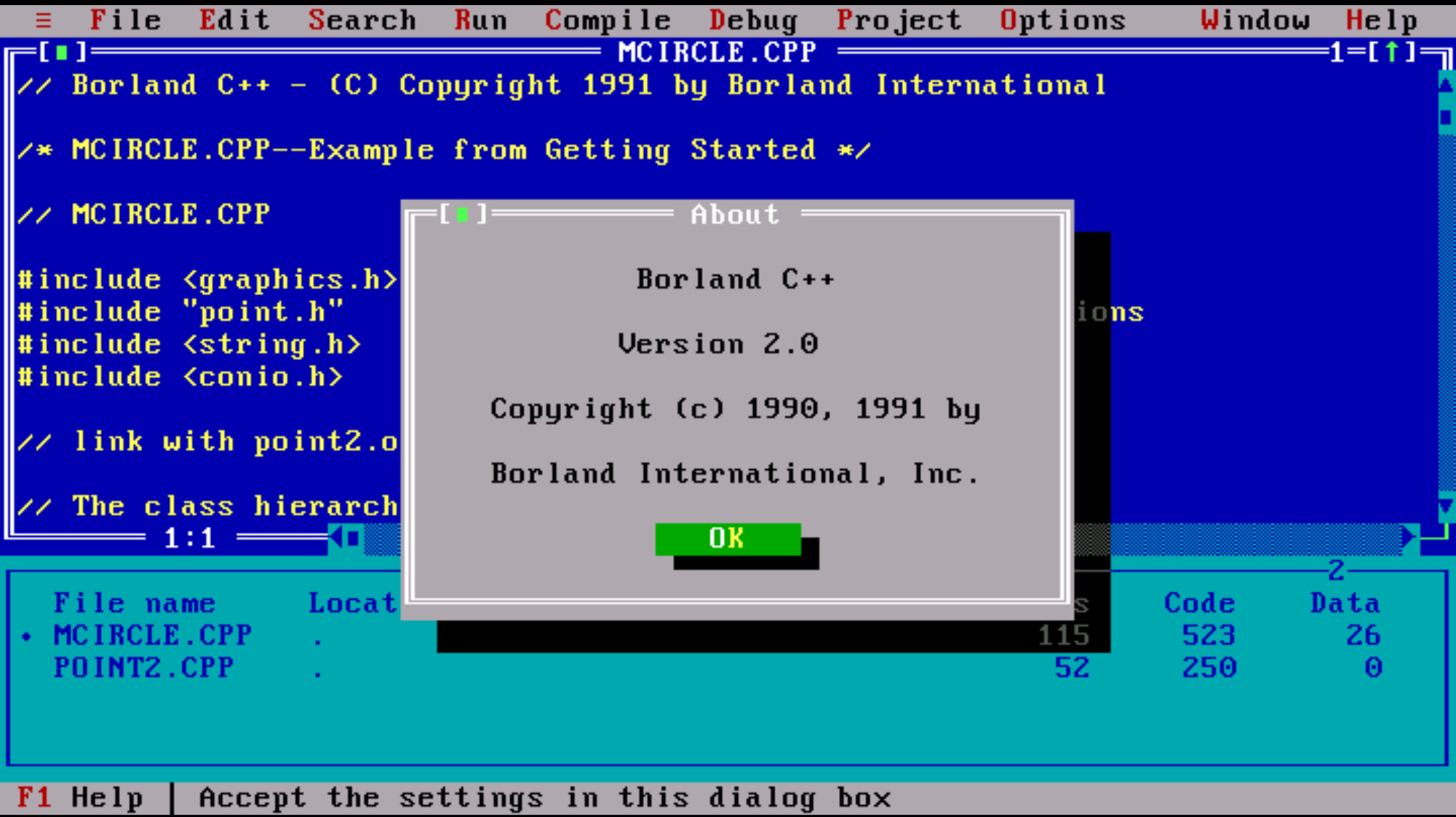
C++ Goodies

Timo Bingmann – 8. März 2017 @ Karlsruhe C++ Meetup

```
alignas alignof and and_eq asm auto bitand bitor
bool break case catch char char16_t char32_t class
compl const const_cast constexpr continue decltype
default delete do double dynamic_cast else enum
explicit export extern false final float for
friend goto if inline int long mutable namespace
new noexcept not not_eq nullptr operator or
or_eq override private protected public register
reinterpret_cast return short signed sizeof static
static_assert static_cast struct switch template
this thread_local throw true try typedef typeid
typename union unsigned using virtual void
volatile wchar_t while xor xor_eq
```



Roadmap (even though we may not get through)

1. Rvalue References and Move Semantics
   (with Excursions into Lambdas and std::function)

2. Virtual Final Override

3. Variadic Template Parameter (Un-)Packing

4. Random Bits of Thrill

R-val·ue

noun

The capacity of an insulating material to resist heat flow. The higher the R-value, the greater the insulating power.

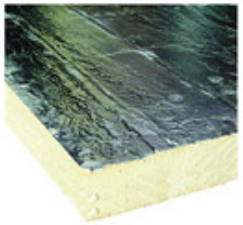
Insulation R-Values

TYPE OF INSULATION	R-VALUE
Fiberglass Batt	3.14
Fiberglass Blown (attic)	2.2
Fiberglass Blown (wall)	3.2
Rock Wool Batt	3.14
Rock Wool Blown (attic)	3.1
Rock Wool Blown (wall)	3.03
Cellulose Blown (attic)	3.13
Cellulose Blown (wall)	3.7
Vermiculite	2.13
Air-entrained Concrete	3.9
Urea terpolymer foam	4.48
Rigid Fiberglass (> 4lb/ft^3)	4
Expanded Polystyrene (beadboard)	4
Extruded Polystyrene	5
Polyurethane (foamed-in-place)	6.25
Polyisocyanurate (foil-faced)	7.2

3.14
Fiberglass Batt

3.13
Cellulose Blown (attic)

6.25
Polyurethane (foamed-in-place)



7.2
Polyisocyanurate (foil-faced)

Lvalues and Rvalues – from the Standard

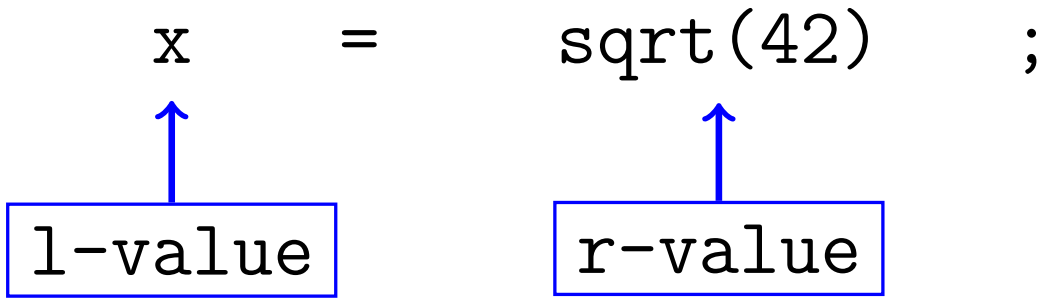
- An lvalue (so called, historically, because lvalues could appear on the left-hand side of an assignment expression) designates a function or an object.
- An xvalue (an "eXpiring" value) also refers to an object, usually near the end of its lifetime (so that its resources may be moved, for example). An xvalue is the result of certain kinds of expressions involving rvalue references (8.3.2).
- A glvalue ("generalized" lvalue) is an lvalue or an xvalue.
- An rvalue (so called, historically, because rvalues could appear on the right-hand side of an assignment expressions) is an xvalue, a temporary object (12.2) or subobject thereof, or a value that is not associated with an object.
- A prvalue ("pure" rvalue) is an rvalue that is not an xvalue.

Lvalues and Rvalues – from MS Visual C++

"Every C++ expression is either an lvalue or an rvalue."

"An lvalue refers to an object that persists beyond a single expression. You can think of an lvalue as an object that has a name. All variables, including nonmodifiable (const) variables, are lvalues."

"An rvalue is a temporary value that does not persist beyond the expression that uses it."

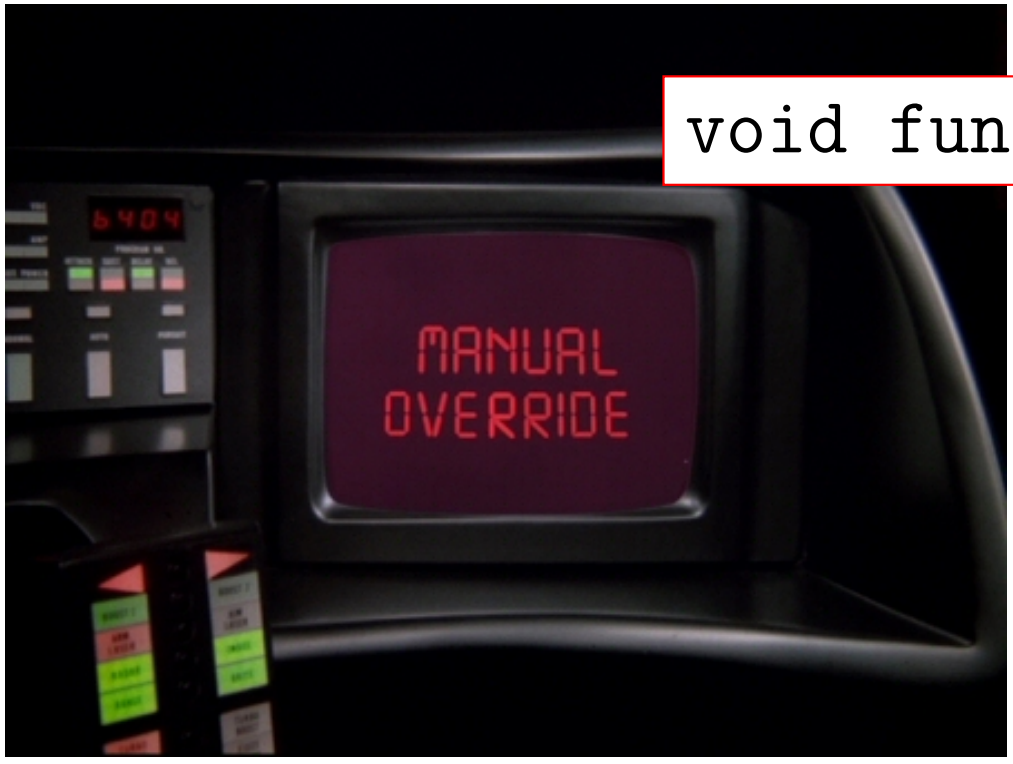


C++11 allows us to capture **rvalues by reference**: `Type&&`.

virtual void function();

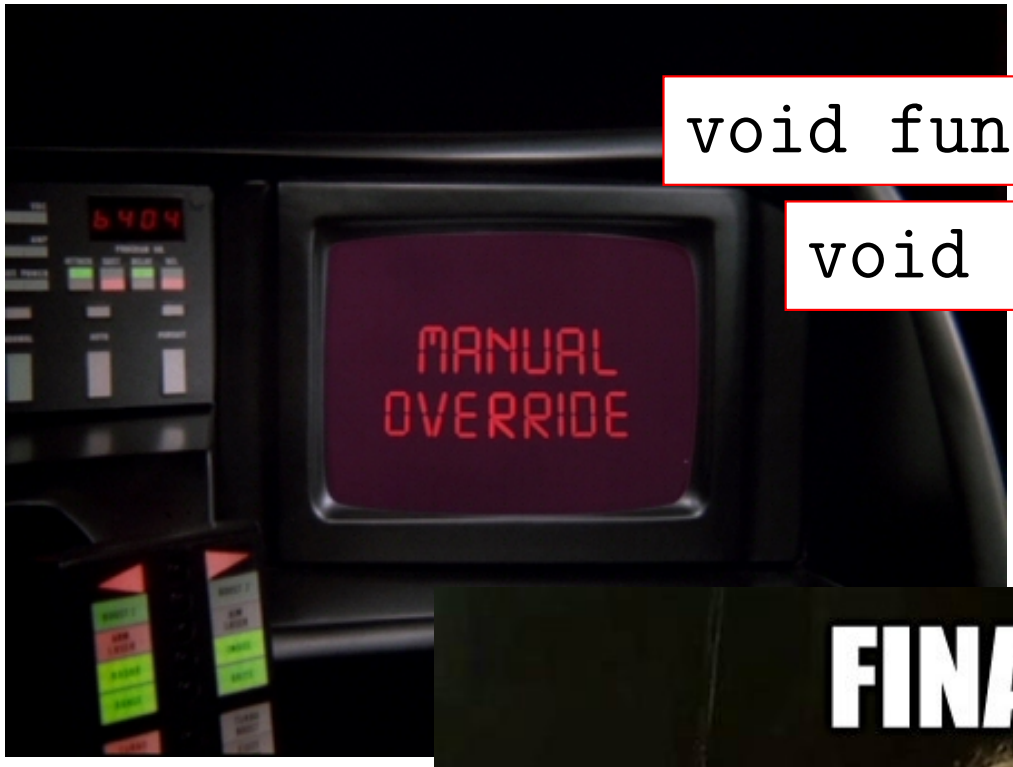
CC by Minecraftpsyco @ Wikipedia

virtual void function();



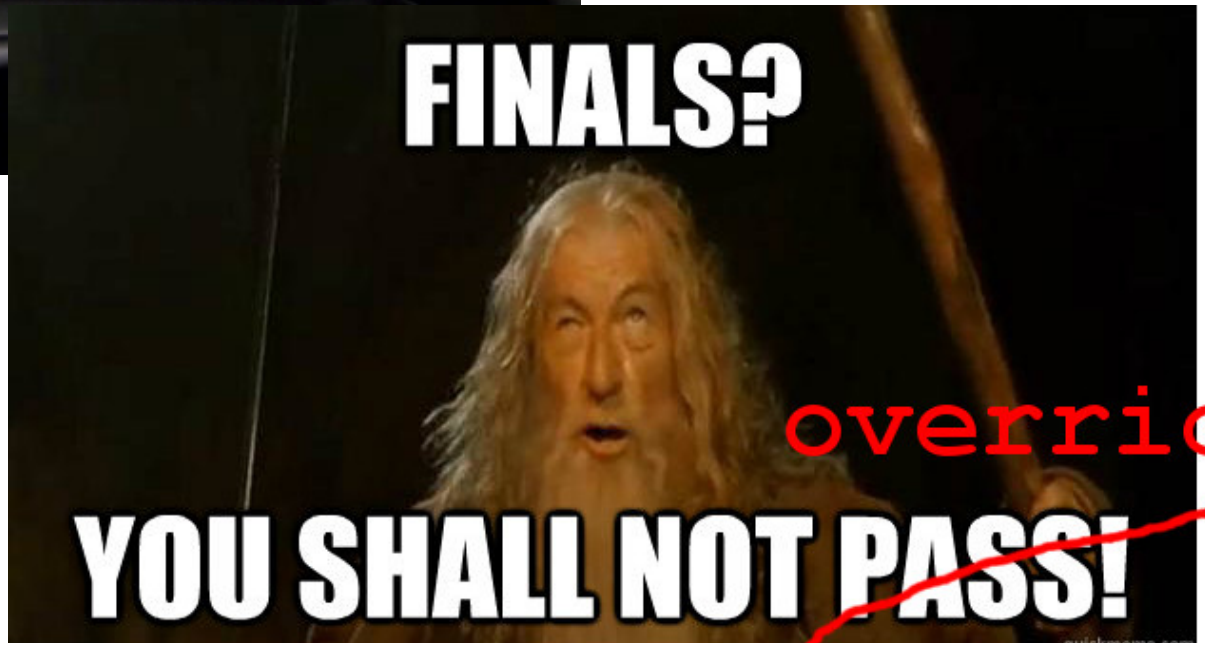

```
void function() override;
```

virtual void function();




```
void function() override;
```

```
void function() final;
```

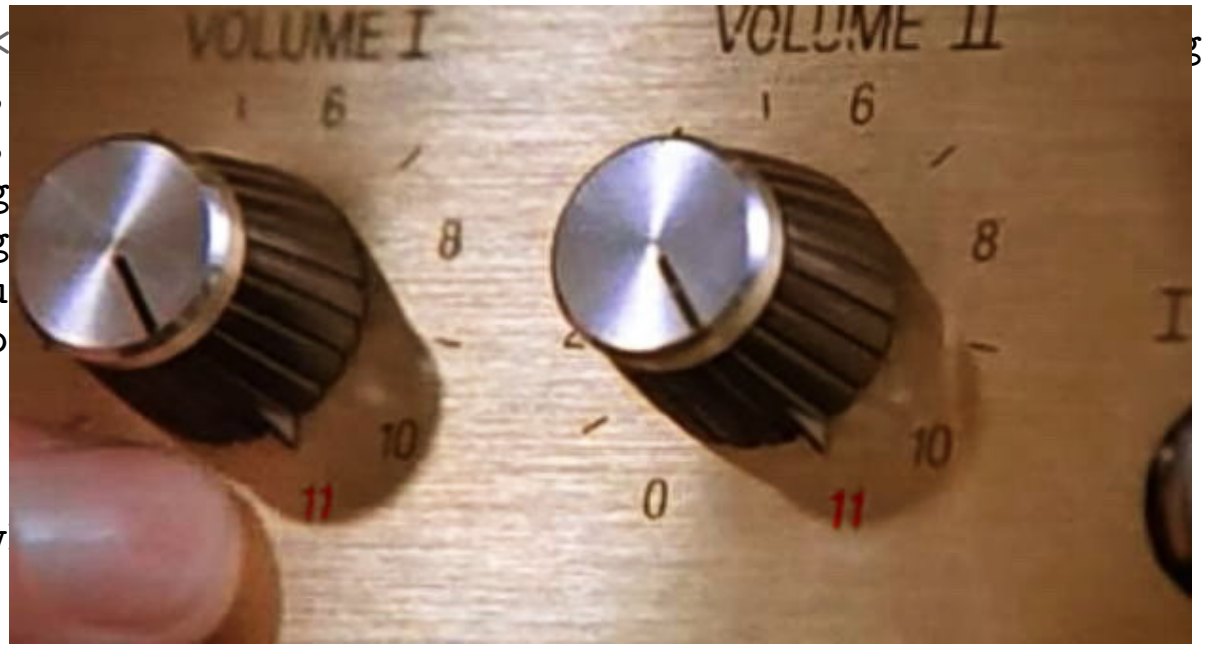


C++ Variadic Templates

```
using ca = int64_t;
template <typename ml, ml ta> struct qu { const static ml dw = ta; };
template <bool vl, ca fy, ca xa> struct fl : fl<(xa * 10> fy), fy, xa * 10> {};
template <ca fy, ca xa> struct fl<true, fy, xa> { using tg = qu<ca, xa>; };
template <ca fy> using ha = typename fl<(10 > fy), fy, 10>::tg;
template <bool yj, bool fk, ca ta, ca ls, ca af, ca lm>
    struct vo : vo < af<10, ta % lm == ls, ta / 10, ls, af / 10, lm> {};
template <bool fk, ca ta, ca ls, ca af, ca lm>
struct vo<true, fk, ta, ls, af, lm> { using tg = qu<bool, false>; };
template <ca ta, ca ls, ca af, ca lm> struct vo<false, true, ta, ls, af, lm>
{ using tg = qu<bool, true>; };
template <ca ta, ca ls>
using qe = typename vo <
           ha<ta>::dw<ha<ls>::dw, false, ta, ls, ha<ta>::dw, ha<ls>::dw>::tg;
template <bool ao, ca ta, ca... gg> struct di { using tg = qu<bool, ao>; };
template <bool ao, ca ta, ca scq, ca... gg>
struct di<ao, ta, scq, gg...> : di<ao | qe<ta, scq>::dw, ta, gg...> {};
template <ca ta, ca... gg> using zw = typename di<false, ta, gg...>::tg;
template <ca pl> struct uo {
  const static ca dw = uo<pl - 1>::dw + uo<pl - 2>::dw;
};
template <> struct uo<0> { const static ca dw = 0; };
template <> struct uo<1> { const static ca dw = 1; };
template <ca pl> struct yk { const static ca dw = yk<pl-1>::dw + uo<pl>::dw; };
```

C++ Variadic Templates

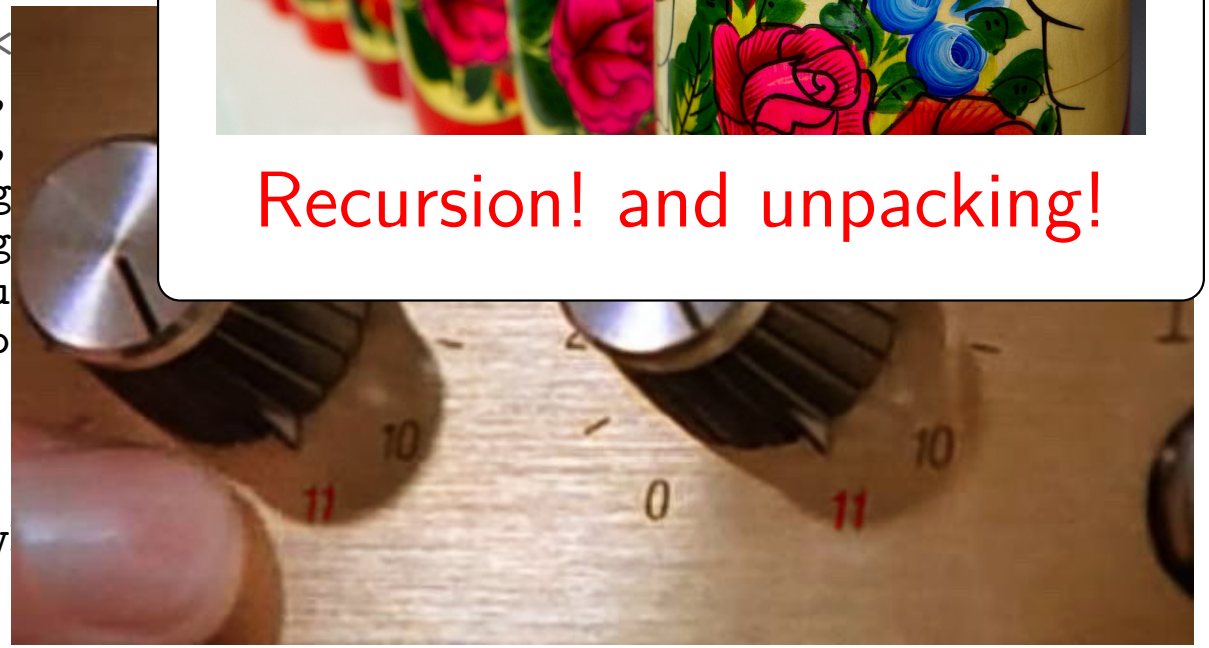
```
using ca = int64_t;
template <typename ml, ml ta> struct qu { const static ml dw = ta; };
template <bool vl, ca fy, ca xa> struct fl : fl<(xa * 10> fy), fy, xa * 10> {};
template <ca fy, ca xa> struct fl<true, fy, xa> { using tg = qu<ca, xa>; };
template <ca fy> using ha = typename fl<(10 > fy), fy, 10>::tg;
template <bool yj, bool fk, ca ta, ca ls, ca af, ca lm>
    struct vo : vo < af<10, ta % lm == ls, ta / 10, ls, af / 10, lm> {};
template <bool fk, ca ta, ca ls, ca af, ca lm>
struct vo<true, fk, ta, ls, af, lm> { using tg = qu<bool, false>; };
template <ca ta, ca ls, ca af, ca lm> struct vo<false, true, ta, ls, af, lm>
{ using tg = qu<bool, true>; };
template <ca ta, ca ls>
using qe = typename vo <
           ha<ta>::dw<ha<                                                  ;
template <bool ao, ca ta,
template <bool ao, ca ta,
struct di<ao, ta, scq, gg
template <ca ta, ca... gg
template <ca pl> struct u
  const static ca dw = uo
};
template <> struct uo<0>
template <> struct uo<1>
template <ca pl> struct y                                                  };
```



C++ Variadic Templates

```
using ca = int64_t;
template <typename ml, ml ta> st
template <bool vl, ca fy, ca xa
template <ca fy, ca xa> struct
template <ca fy> using ha = type
template <bool yj, bool fk, ca
    struct vo : vo < af<10, ta
template <bool fk, ca ta, ca ls
struct vo<true, fk, ta, ls, af,
template <ca ta, ca ls, ca af,
{ using tg = qu<bool, true>; };
template <ca ta, ca ls>
using qe = typename vo <
           ha<ta>::dw<ha<
template <bool ao, ca ta,
template <bool ao, ca ta,
struct di<ao, ta, scq, gg
template <ca ta, ca... gg
template <ca pl> struct u
  const static ca dw = uo
};
template <> struct uo<0>
template <> struct uo<1>
template <ca pl> struct y
```

Recursion! and unpacking!



Questions?

Thank you for your attention.

Questions?

Source code examples used in talk available at

`https://github.com/bingmann/2017-cpp-goodies`

for self study.

More of my work: `http://panthema.net`