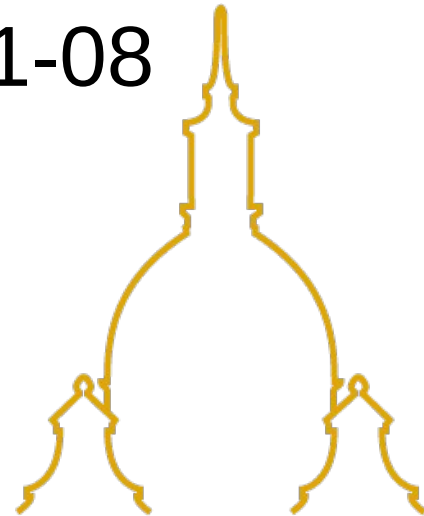


any optional variant

C++17 vocabulary types

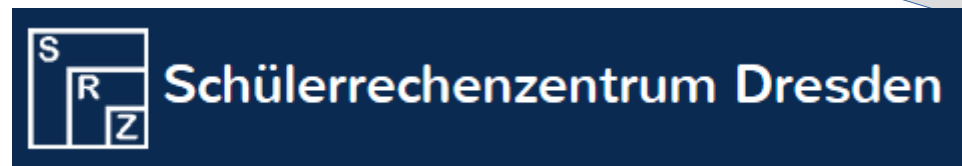
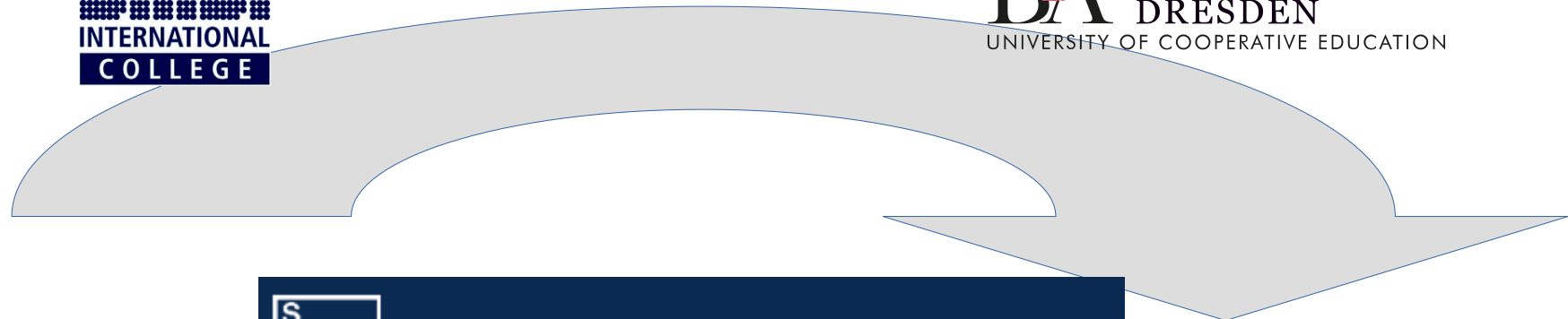
How did we get there
and how to use them

René Richter
2018-11-08



C + +
usergroup
D R E S D E N

about me



PH Dresden
Teacher: Math, Physics
(+ computer science)



agenda

- where do we come from?
- any optional variant // C++17
- how do they fit into the type system?
- type system complete?

expected<Things,E> discussion = { ... };

the name game: what is in a name?

```
auto something = 0b0010'1010;  
                // C++?? version bingo
```

- size : number of bits or bytes
 - data : $[0|1]^*$ (bit representation)
 - what : interpretation (value)
 - how : to deal with (operations)
 - where: location in memory (address)
- } type

type theory?

$\text{card}(V) \sim \text{sizeof}(\text{something})$

/

$\text{type} = (\{\text{values}\dots\}, \{\text{operations}\dots\})$

- $U \times V$ cross product (product type)
- $U + V$ (sum type)
- type hierarchies
- names of types, types of types $\rightarrow \infty$

caveat!

names can lie

- meaning (semantics)?

let's talk about...

```
auto talk() {  
    auto nothing = []{};  
    nothing();  
    return nothing();  
}  
  
int main() {  
    talk();  
}
```

nothing

must something have

- an address in memory?
- a name?
- a value?
- some type?
- some data?

rvalue, register

rvalue, operation result

uninitialized memory,
garbage

```
void* black_hole
```

```
void, []{}()
```


everything

'*' == char(0b0010'1010)

time + space (i.e. memory) constraints

many things

cardinality (ER model in database design)

- 0, 1, many (N) or any (*) data units
- 0, 1, many (N) or any (*) types

where

- N = fixed number
- * = 0 to ... (or 1 to ...)

FORTRAN

number of data	0	number of types	==>	*
0				
0/1				
1		T		
N		T[N]		
*				

let's C ...

number of data		number of types	==>	
0	void	1	N	*
0/1				
1		T		
N		T[N]	<code>struct S{T1 t1;...TN tn;}</code>	
*				

the million dollar mistake

invention of the null reference (1965)

comprehensive type system
for references in an object oriented language
(ALGOL W) [C.A.R. Hoare]

Because pointers are such dangerous things,
it's better to never generate one. [Herbert Schildt]

dangerous pointers

```
if (!p) // road to nowhere?  
p = somewhere // uninitialized?  
p[index] // object or range?  
++p // out of bounds?  
delete p // owner?  
delete[] p // zombie area?
```

C disaster area

number of data		number of types	==>	
	0	1	N	*
0	void			
0/1		T*	union U{T1 t1;...TN tn;}	void*
1		T		
N		T[N]	struct S{T1 t1;...TN tn;}	void*
*		dynamic T*	dynamic U*	dyn. void*

C++98

number of data	number of types	==>	
0	void	1	*
0/1	T*	union U{T1 t1;...TN tn;}	void*
1	T		
N	T[N] bitset<N>	struct S{T1 t1;...TN tn;} pair<T1,T2>	void*
*	dynamic T* vector<T>	dynamic U*	dyn. void*

C++11

number of data		number of types	==>	
0	void		N	*
0/1		T*	union U{T1 t1;...TN tn;}	void*
		unique_ptr<T> shared_ptr<T>		
1		T		
N		T[N] array<T,N> bitset<N>	struct S{T1 t1;...TN tn;} tuple<T1,...TN> pair<T1,T2>	void*
*		dynamic T*	dynamic U*	dyn. void*
		vector<T>		

C++17

number of data	number of types	==>	
0	void	1	*
0/1	void	T*	union U{T1 t1;...TN tn;} void*
		unique_ptr<T> shared_ptr<T> optional<T>	variant<monostate, T1, ..., TN> any
1		T	variant<T1, ..., TN>
N		T[N] array<T, N> bitset<N>	struct S{T1 t1;...TN tn;} tuple<T1, ...TN> pair<T1, T2>
*	void	dynamic T*	dynamic U* dyn. void*
		vector<T>	vector<variant<T1, ...TN>> vector<any>

state of the union in C++

```
union PF { Dog d; Pig p; Sheep s; };
```

- non-static data with non-trivial constructors / destructors?
- RAI (AC/DC)?
 - C++98 : forbidden
 - C++11 : roll yer own
 - C++17 : `std::variant<Dog, Pig, Sheep>`

C++17 vocabulary types

```
#include <any>  
// typesafe replacement for void*
```

```
#include <optional>  
// like a nullable type in database
```

```
#include <variant>  
// a more perfect(?) union
```

any optional variant of interface

```
any a  
make_any<T>(args)
```

```
a = 42  
a.emplace<T>(args)
```

```
a.has_value()  
any_cast<T>(a)  
any_cast<T>(&a)  
    // != nullptr?
```

```
a.type()  
    // == typeid(T)?
```

```
a.reset()
```

```
// may throw  
bad_any_cast  
: bad_cast
```

```
optional<T> o  
make_optional<T>(args)
```

```
o = 42  
o.emplace(args)
```

```
if (o) | o.has_value()  
    *o | o.value()  
    UBA^! | o.valueOr(42)
```

```
o = null_opt  
o.reset()
```

```
// may throw  
bad_optional_access  
: exception
```

```
variant<Ts...> v  
variant<monostate,...>
```

```
v = 42  
v.emplace<T>(args)  
v.emplace<Index>(args)
```

```
get<T>(v)  
get<Index>(v)  
visit(visitor, v)
```

```
v.index()  
holds_alternative<T>(v)
```

```
// may throw  
bad_variant_access  
: exception
```

Ability to Learn & Teach?

-

+

--

how did we get here?

If we can supply a feature as a library, we should do so [...]
However, a library design should not be an excuse for inelegant interfaces, for irregular interfaces, for stylistic differences from built-in language features, or overelaboration. It is always easy to add another function to a class, so library components have a tendency to bloat. Note [...] the dramatic differences in the interfaces to `std::any`, `std::optional`, and `std::variant`.

[...] `std::variant`, `std::optional`, and `std::any` have a long history as independent proposals. That wouldn't be too bad if the reason was that significant improvements were added during the process.

B.Dawes/H.Hinnant/B.Stroustrup/D.Vandevoorde/M.Wong:
P0939R0: Direction for ISO C++ (2018-02-10).

visit a variant: have a visitor

```
struct MyVisitor
{
    void operator()(char c)           { std::cout << c; }
    void operator()(int i)           { std::cout << i; }
    void operator()(std::string s)   { std::cout << s; }

    void operator()(auto x)         { std::cout << x; }
};

void demo1()
{
    using myvariant = std::variant<char, int, double, std::string>;
    auto v = std::vector<myvariant>{ 'C', "++", 17, "->", 20.17 };

    for (auto e : v)
        std::visit(MyVisitor{}, e);
}
```

context aware visitor

```
// http://en.cppreference.com/w/cpp/utility/variant/visit
```

```
template<class... Ts>  
struct overloaded : Ts... { using Ts::operator()...; };  
template<class... Ts> overloaded(Ts...) -> overloaded<Ts...>;
```

```
void demo2()  
{  
    using myvariant = std::variant<char, int, double, std::string>;  
    auto v = std::vector<myvariant>{ 'C', "++", 17, "->", 20.17 };  
  
    for (auto e: v)  
    {  
        std::visit(overloaded  
        { [&](char c)          { std::cout << c; },  
          [&](int i)           { std::cout << i; },  
          [&](std::string s)   { std::cout << s; },  
          [&](auto x)          { std::cout << x; },  
        }, e);  
    }  
}
```


variant use case

- github.com/mpusz/fsm-variant

Finite State Machines:
states as types, not values

any T_{ype} V_{alue} casting show

```
std::any o; // empty
o = 1;
o = 3.14;
o = "hello world";
```

```
if (auto ptr = any_cast<int>(&o))      cout << *ptr;
if (auto ptr = any_cast<double>(&o))  cout << *ptr;
if (auto ptr = any_cast<const char*>(&o)) cout << *ptr;
```

```
if (o.has_value()) {
    try {
        cout << any_cast<const char*>(o);
        cout << any_cast<double>(o);
        cout << any_cast<int>(o);
    }
    catch (bad_any_cast& err) {
        cerr << err.what();
    }
}
o.reset();
```

void* to any thing

- www.bfilipek.com/2018/06/any.html

optional<result>

```
auto find_smallest(std::vector<int> v)
-> std::optional<int>
{
    if (v.empty()) return {};
    return *min_element(begin(v), end(v));
}
// ---8<-----
if (auto m = find_smallest({2018, 11, 8}))
{
    std::cout << *m << '\n';
}
```

now it is Xmas...


```
hollywar =  
optional<Elefant&>{inTheRoom};
```

[www.fluentcpp.com/2018/10/05/
pros-cons-optional-references/](http://www.fluentcpp.com/2018/10/05/pros-cons-optional-references/)

[thephd.github.io/2018/10/25/
Big-Papers-Optional.html](http://thephd.github.io/2018/10/25/Big-Papers-Optional.html)

```
P0798R2 Monadic operations for std::optional  
o.map(f).and_then(g).orElse(h)
```

Result<T,E> // Rust

- optional<T> vs. (Boost.)Outcome ...
- foonathan.net/blog/2017/12/04/exceptions-vs-expected.html
- Niall Douglas Meeting C++ 2017
- experimental  expected<T,E>
P0323R7 (2018-06-22)
? Library Fundamentals TS v3
- Zero-overhead deterministic exceptions P0709R2
(2018-10-06) Herb Sutter

::

— <=> —;

any result =
past→present→future();

these are the voyages^{...} of the starship C++
to boldly go where no^{...} one has gone before

(C++2a) ^{...}cpp-ug-dresden.blogspot.com