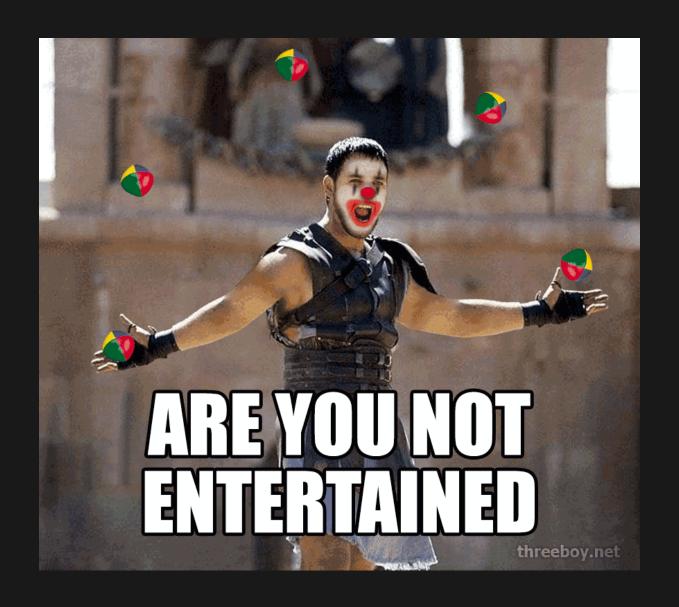
CPPFRONT, CARBON, CIRCLE, WHAT'S NEXT FOR C++?



LIGHTING TALK

Piotr Wierciński

2022 is very exciting year for C++ ecosystem



What is happening in system programming world:

rapid growth of distinct languages (Rust, Zig)

What is happening in system programming world:

- rapid growth of distinct languages (Rust, Zig)
- new languages as direct successors of C++ (Carbon)

What is happening in system programming world:

- rapid growth of distinct languages (Rust, Zig)
- new languages as direct successors of C++ (Carbon)
- evolution of C++ itself, new dialects (CppFront, Circle)

Agenda:

- CppFront
- Carbon
- Circle

Goals of this talk:

- tease the syntax
- what problem does it solve
- what it tries to achieve
- maturity of project

CPP FRONT

CPP FRONT

presented by Herb Sutter on CppCon2022 closing keynote

CPP FRONT

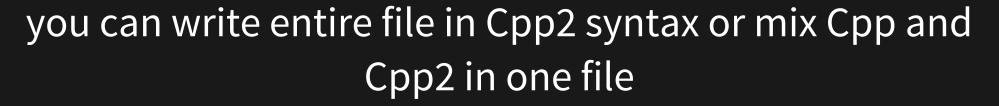
presented by Herb Sutter on CppCon2022 closing keynote

goal: "make C++ 10x simpler & safer"

CppFront introduces a new syntax - Cpp2

CppFront introduces a new syntax - Cpp2
Cpp2 files are transpiled by CppFront to normal C++

CppFront introduces a new syntax - Cpp2
Cpp2 files are transpiled by CppFront to normal C++
Cpp2 for C++ is like TypeScript for JavaScript



the idea is make it easy to innovate and improve language without changes to C++ standard and compilers

A lot can be checked during Cpp2->Cpp transpilation:

 making defaults right (const variables, non-discard functions)

A lot can be checked during Cpp2->Cpp transpilation:

- making defaults right (const variables, non-discard functions)
- remove unsafe parts (unions, raw pointers arithmetic)

A lot can be checked during Cpp2->Cpp transpilation:

- making defaults right (const variables, non-discard functions)
- remove unsafe parts (unions, raw pointers arithmetic)
- improve memory safety (enforce CppCoreGuideliness)

SYNTAX TEASER

UNIVERSAL TYPE DECLARATION

name: type = value

Variables

```
//Cpp1
std::vector<std::string> vec{"hello", "world"};

//Cpp2
vec: std::vector<std::string> = ("hello", "world");
```

Classes

```
//Cpp1
class Shape {};

//Cpp2
shape: type {}
```

Functions

```
//Cpp1
std::string f(int i) {}

//Cpp2
f: (i: int) -> std::string = {}
```

PARAMETER PASSING SYNTAX

implements proposal D0708

```
f: (inout x: string) -> void = {...}
```

IMPLICIT TEMPLATE PARAMS

```
//Cpp1
template<typename T1, typename T2>
int f1(T1 foo, T2 bar) {...}

//Cpp2
f1: (foo: _, bar: _) -> int = {...}
```



Checkout https://github.com/hsutter/cppfront

Ready to be tested on Godbolt

CARBON

CARBON

presented by Chandler Carruth at CppNorth 2022

CARBON

presented by Chandler Carruth at CppNorth 2022 officially called "experimental successor to C++"

comes from Google

comes from Google aspires to be community driven

• difficult to improve C++

- difficult to improve C++
- technical debt

- difficult to improve C++
- technical debt
- backward compatibility holds the language back

- difficult to improve C++
- technical debt
- backward compatibility holds the language back
- ABI stability is a blocker for performance improvements

What Carbon aims for:

• be a high performant language like C++

What Carbon aims for:

- be a high performant language like C++
- be easy to evolve

What Carbon aims for:

- be a high performant language like C++
- be easy to evolve
- break ABI when it needs

What Carbon aims for:

- be a high performant language like C++
- be easy to evolve
- break ABI when it needs
- break API when it needs

"Why not rewrite everything in Rust?"

"Why not rewrite everything in Rust?"

Impossible to gradually migrate huge C++ codebase to Rust.

Killer feature of Carbon:

INTEROPERABILITY with C++

Carbon -> C++

```
//Person.h
struct Person { int iq;}

//Person.carbon
package Person api;

import Cpp library "Person.h"

fn PrintIq(person: Cpp.Person) {
    Print("His/her/their IQ is {0}", person.iq);
}
```

C++ -> Carbon

```
//person.carbon
package Person api;
import Cpp library "Person.h"
fn PrintIq(person: Cpp.Person) {
    Print("His/her/their IQ is {0}", person.iq);
}

//Person.cpp
#include "Person.h"
#include "person.carbon.h"
Person p1;
Person::PrintIq(p1);
```

MATURITY

early stage of language design not fully there yet, no working compiler

CIRCLE

CIRCLE

C++ 20 compiler written from scratch by Sean Baxter

CIRCLE

C++ 20 compiler written from scratch by Sean Baxter

it "extends" a C++ by adding many novel features

Assumption: C++ can evolve incrementally into much better language

SYNTAX TEASER

Pack subscript

```
template<typename ...T>
void f(T... foo) {
   std::cout << T...[0] << std::endl;
   std::cout << T...[-2] << std::endl;
}</pre>
```

Pack slice

```
template<typename ...T>
void f(T... foo) {
   std::cout << T...[: sizeof... foo / 2 ]; //first half
   std::cout << T...[1::2]; //odds
   std::cout << T...[::-1]; //reverse
}</pre>
```

Reflections given struct:

```
struct record_t {
   std::string first;
   std::string last;
   char32_t middle_initial;
   int year_of_birth;
};
```

```
template<typename type_t>
void useReflections() {
  printf("%s\n", @type_string(type_t));
}
useReflections<record_t>();
```

```
template<typename type_t>
void useReflections() {
  printf("%s\n", @type_string(type_t));
}
useReflections<record_t>();
```

produces

```
record_t
```

```
template<typename type_t>
void useReflections() {
    @meta for(int i = 0; i < @member_count(type_t); ++i) {
        printf("%d: %s - %s\n", i, @member_type_string(type_t, i),
          @member_name(type_t, i));
    }
}
useReflections<record_t>();
```

```
template<typename type_t>
void useReflections() {
    @meta for(int i = 0; i < @member_count(type_t); ++i) {
        printf("%d: %s - %s\n", i, @member_type_string(type_t, i),
           @member_name(type_t, i));
    }
}
useReflections<record_t>();
```

produces

```
0: std::string - first
1: std::string - last
2: char32_t - middle_initial
3: int - year_of_birth
```



working compiler

https://www.circle-lang.org/>

also in Godbolt

QUESTIONS?