























C++ Club Meeting Notes

Gleb Dolgich

2018-08-09

IEEE Most popular programming languages of 2018

Article, [Reddit](#)

Language Rank	Types	Spectrum Ranking
1. Python	  	100.0
2. C++	  	98.4
3. C	  	98.2
4. Java	  	97.5
5. C#	  	89.8
6. PHP		85.4
7. R		83.3
8. JavaScript	 	82.8
9. Go	 	76.7
10. Assembly		74.5

Follow-up: Toby Allsopp “An Introduction to the Proposed Coroutine Support for C++”

Video

How to use coroutines today

- ▶ Visual Studio 2017 (/await)
- ▶ Clang 5.0 with libc++ 5.0 (-fcoroutines-ts -stdlib=libc++)

Abstraction libraries

- ▶ **cppcoro**
 - ▶ task, generator, async_generator, async_mutex, ...
- ▶ **range-v3**
 - ▶ generator

Clang Concepts is feature-complete

- ▶ [Announcement](#)
- ▶ [Compiler Explorer](#)
- ▶ [Code](#)
 - ▶ Andrew Sutton's [reply](#): “If I had not gone on vacation, I might have beaten you to the punch in GCC ;) I’m in the process of working through older TS tests.”

The optimal way to return from a function, by Jason Turner

Video

Single return (20%):

```
1 string val(const bool b) {  
2     string ret;  
3     if (b) ret = "Hello"; else ret = "World";  
4     return ret;  
5 }
```

versus multiple return (61%):

```
1 string val(const bool b) {  
2     if (b) return "Hello"; else return "World";  
3 }
```

versus:

```
1 string val(const bool b) {  
2     return b ? "Hello" : "World";  
3 }
```

- ▶ [Video](#)
- ▶ [Slides](#)

Class Template Deduction Guides

Which helps for cases like this:

```
1  template<typename First, typename Second>
2  struct Pair {
3      template<typename P1, typename P2>
4      Pair(P1 &&p1, P2 &&p2)
5          : first(std::forward<P1>(p1)), second(std::forward<P2>(p2))
6      {}
7
8      First first;
9      Second second;
10 };
11
12 template<typename P1, typename P2>
13 Pair(P1 &&p1, P2 &&p2) -> Pair<std::decay_t<P1>, std::decay_t<P2>>;
14
15 int main() {
16     Pair p{1, 2.3};
17 }
```

Copyright Jason Turner

@lefticus

8.4



JASON TURNER

Practical C++17

CppCon.org

Fold Expressions

```
1 { ... <op> <pack expression> } // unary left fold
2 { <pack expression> <op> ... } // unary right fold
3 { <init> <op> ... <op> <pack expression> } // binary left fold
4 { <pack expression> <op> ... <op> <init> } // binary right fold
```

Allows for "folding" of a variadic parameter pack into a single value.

Allowed operations:

+ - * / % ^ & | < > += -= *= /= %= ^= &= |= << >> == != < >
<= >= && || , .* ->*

Copyright Jason Turner

@lefticus

10.2



Practical C++17

CppCon.org

`noexcept` In The Type System

```
1 void use_func(void (*func)() noexcept);  
2 void my_func();  
3  
4 use_func(&my_func);
```

This will no longer compile in C++17, a new overload is needed:

```
1 void use_func(void (*func)());
```

`noexcept` function pointer can be converted to non-`noexcept`, but not the other way around.

Copyright Jason Turner

@lefticus

11.2



Practical C++17

CppCon.org

C++17: `std::apply` and `std::invoke`

- ▶ `std::apply`
- ▶ `std::invoke`

C++17 in libsigc++ : invoke, apply, and constexpr if

- ▶ [Post](#)
- ▶ [Code](#) (LGPL)
- ▶ [Docs](#)

libsigc++ implements a typesafe callback system for standard C++. It allows you to define signals and to connect those signals to any callback function, either global or a member function, regardless of whether it is static or virtual.

Modules TS example

- ▶ Post
- ▶ Code

Modules TS example

pet.cpp:

```
1 module pets.pet;
2 import std.core;
3
4 export class Pet
5 {
6 public:
7     virtual char const* pet() = 0;
8 };
```

Modules TS example

dog.cpp:

```
1 module pets.dog;
2 import std.core;
3 import pets.pet;
4
5 export class Dog : public Pet
6 {
7     public:
8         char const* pet() override;
9 };
10
11 char const* Dog::pet()
12 {
13     return "Woof!";
14 }
```

Modules TS example

interface.cpp (or maybe pets.cpp?):

```
1 module pets;  
2  
3 export module pets.pet;  
4 export module pets.dog;
```

Modules TS example

main.cpp:

```
1 import pets;
2 import std.core;
3 import std.memory;
4
5 int main()
6 {
7     std::unique_ptr<Pet> pet = std::make_unique<Dog>();
8     std::cout << "Pet says: "<< pet->pet() << std::endl;
9 }
```


- ▶ C++ modules and why we need them desperately
- ▶ Using modules in Visual C++
- ▶ Migrating existing C++ code to use modules
- ▶ Compiling boost on QNX: a tale of why modules are needed in C++
 - ▶ QNX Demo Floppy (1999)

Video

Question

Write a portable C++ (or C) program that displays:

Hello World

on the standard output when executed, WITHOUT USING ANY SEMICOLONS (;)

- ▶ Don't worry what's in standard header files (and in the C version you don't need *any* headers)
- ▶ No preprocessor directives are required (aside from `#include` for C++)
- ▶ No assembly language required

A C++ Puzzle by Leor Zolman

Solutions

C++:

```
1 #include <iostream>
2 int main() {
3     if (std::cout << "Hello World") {}
4 }
```

C:

```
1 int main() {
2     if (printf("Hello World")) {}
3 }
```

CppChat: Volatile Is the Embedded Keyword

Episode

- ▶ For many embedded or kernel developers using C++ for anything is anathema: “Here is a thing I made in C++ which solves this problem in the kernel/embedded system” – “Why are you even using C++? You should use C!”
- ▶ C and C++ compiler defaults differ, so compiling C code with a C++ compiler will make it slower. When you disable certain C++ defaults (RTTI, exceptions) it becomes faster than C.
- ▶ Freestanding proposals by Ben Craig:
 - ▶ [Library](#)
 - ▶ [Language](#)
- ▶ [Static exceptions by Herb Sutter](#)

Volatile

- ▶ volatile is needed:
 - ▶ John Regehr's [tweet](#): "I think it's 100% clear the C++ committee should remove volatile"
 - ▶ JF Bastien's [reply](#): "No! I used volatile recently, and advocated for its use too!!! It's great for signals, and [TOCTOU](#), at a minimum. I'm wondering if we should deprecate volatile-qualified functions though. I don't think they're useful anymore."
 - ▶ JF Bastien's [tweet](#): "<...> it defines the member function to call if the this pointer is volatile. That's been standard C++ forever. Same for const member function overloads. Don't forget about *ref* and *rvalue* member functions! <...>"

```
1 class Foo {  
2     void bar() volatile;  
3 };
```

A very fast header-only C++ logging library

- ▶ [Code](#)
- ▶ [V1.0](#)

Interview

