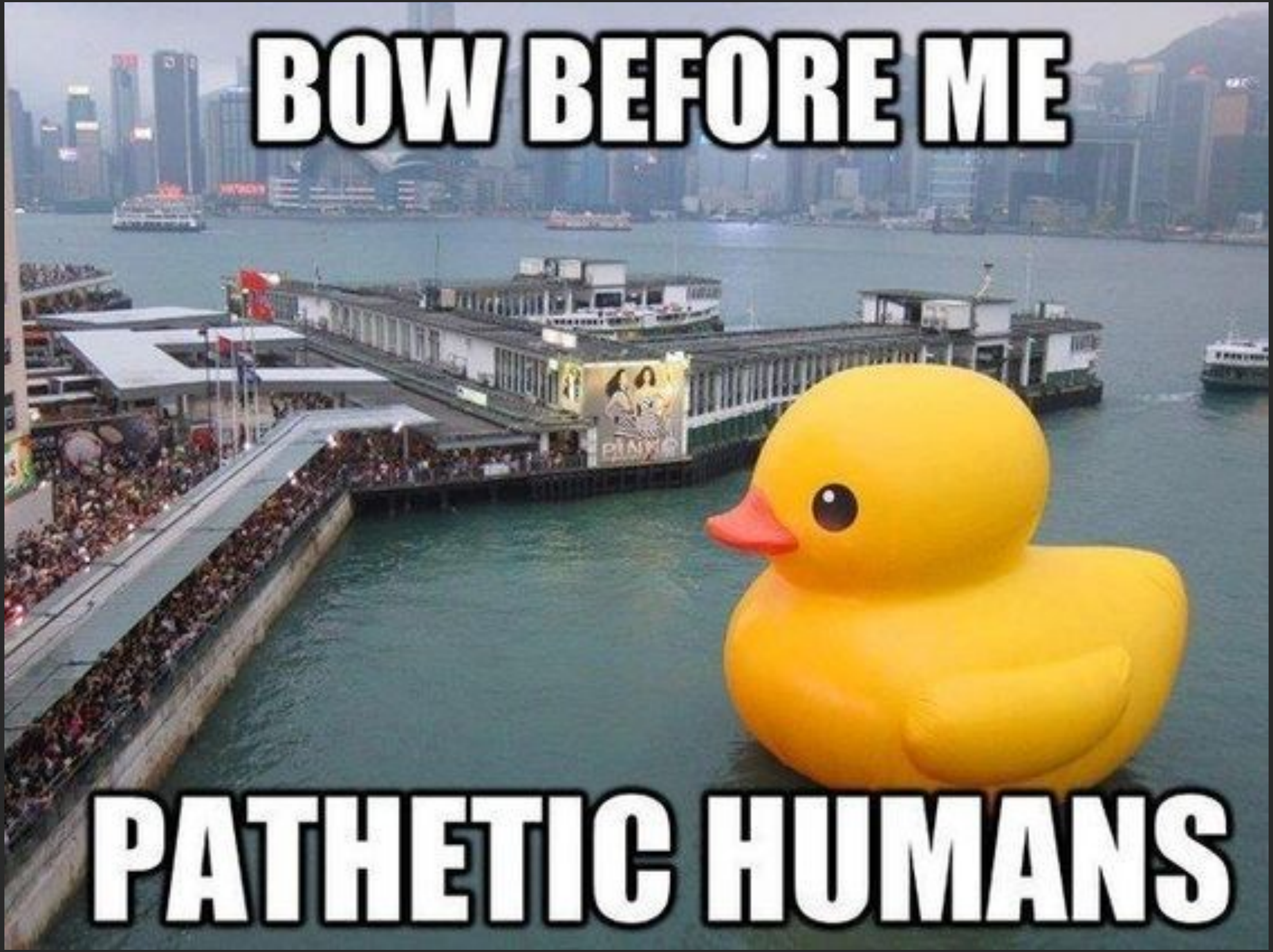
# Welcome to CS 106L!

Avery Wang: averywang@stanford.edu
Anna Zeng: aszeng@stanford.edu

Testing the Poll feature!

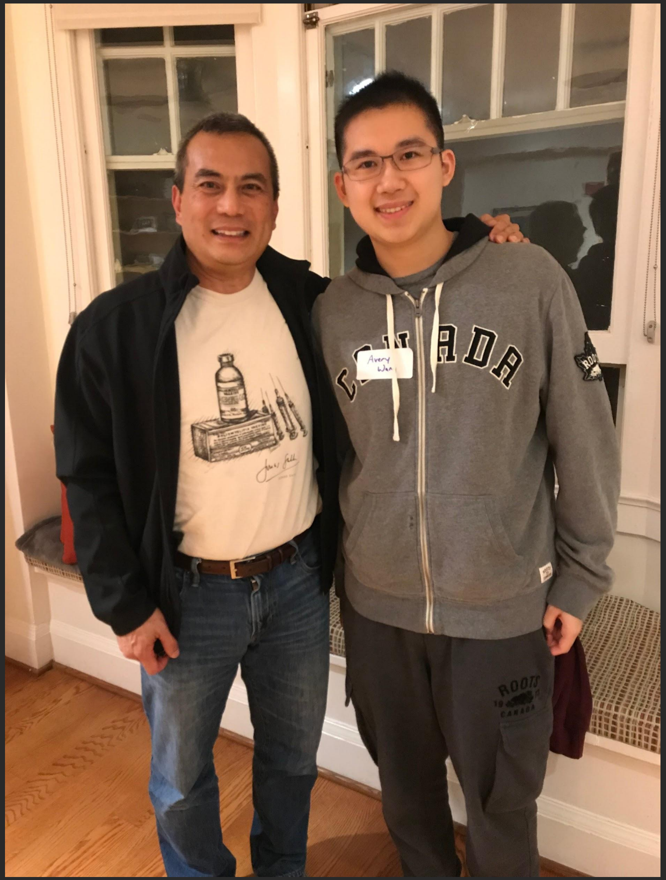

BOW BEFORE ME
PATHETIC HUMANS

## Game Plan

- Welcome
- Logistics
- History and Philosophy of C++
- C++ Basics
- Command-Line Compilation

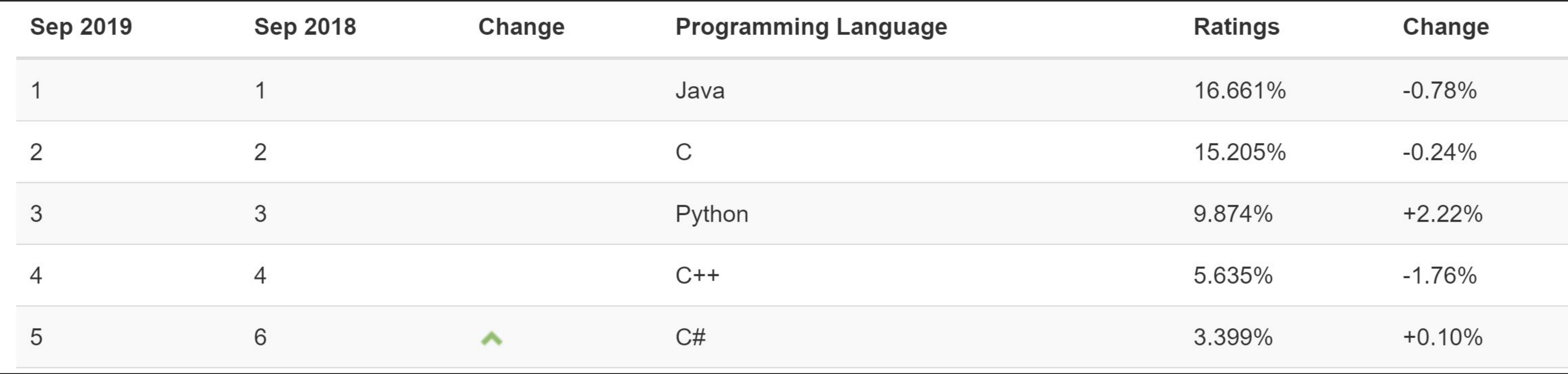
## Introduction

## Instructors



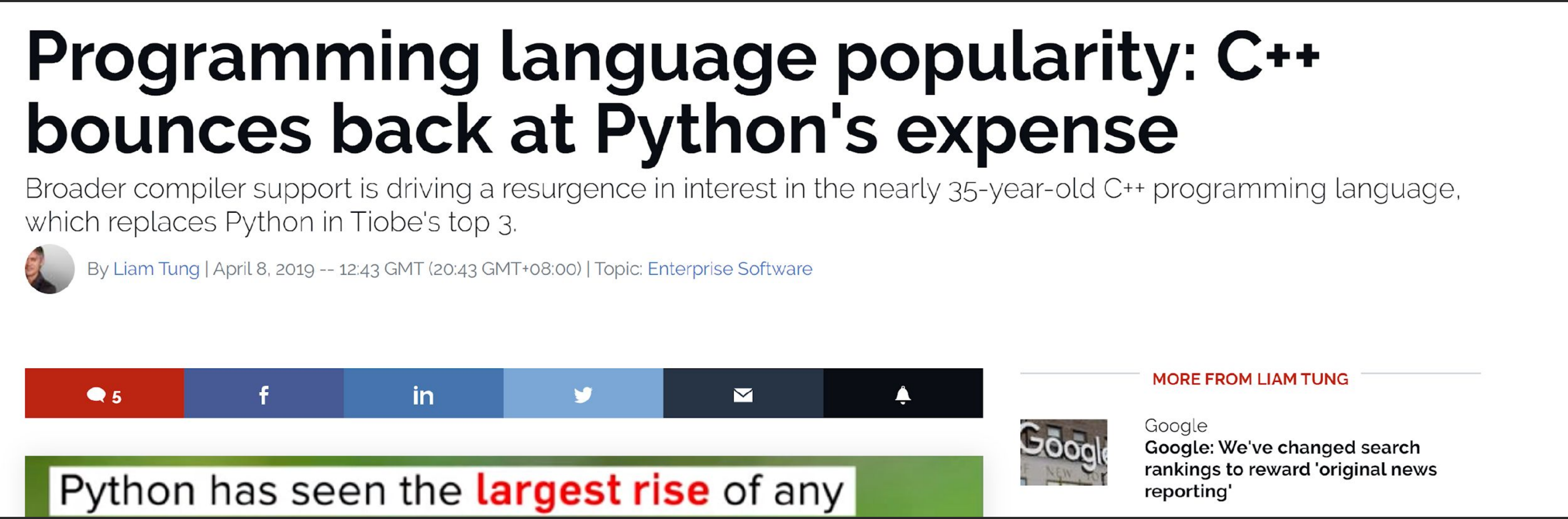
## Why C++?

# C++ is still a very popular language.



| Sep 2019 | Sep 2018 | Change | Programming Language | Ratings | Change |
|---|---|---|---|---|---|
| 1 | 1 | | Java | 16.661% | -0.78% |
| 2 | 2 | | C | 15.205% | -0.24% |
| 3 | 3 | | Python | 9.874% | +2.22% |
| 4 | 4 | | C++ | 5.635% | -1.76% |
| 5 | 6 | ▲ | C# | 3.399% | +0.10% |

## Take that, Python!



Programming language popularity: C++ bounces back at Python's expense

Broader compiler support is driving a resurgence in interest in the nearly 35-year-old C++ programming language, which replaces Python in Tiobe's top 3.

By Liam Tung | April 8, 2019 -- 12:43 GMT (20:43 GMT+08:00) | Topic: Enterprise Software

MORE FROM LIAM TUNG

Google

Google: We've changed search rankings to reward 'original news reporting'

Python has seen the **largest rise** of any

## Classes that use C++

BIOE 215: Physics-Based Simulation of Biological Structure
CME 213: Introduction to parallel computing using MPI
CS 144: Introduction to Computer Networking
CS 231N: Convolutional Neural Networks for Visual Recognition
GENE 222: Parallel Computing for Healthcare
ME 328: Medical Robotics
MUSIC 256A: Music, Computing, Design I
MUSIC 420A: Signal Processing Models in Musical Acoustics

# Companies that use C++

amazon.com

intel

IBM

facebook

Google

Microsoft

Adobe

## Browsers written in C++

## Software written in C++

Java™

Microsoft® Office™

Ps

Windows

BlackBerry

## Games written in C++

ASSASSIN'S CREED

CALL OF DUTY®

MASS EFFECT™

WORLD OF WARCRAFT®

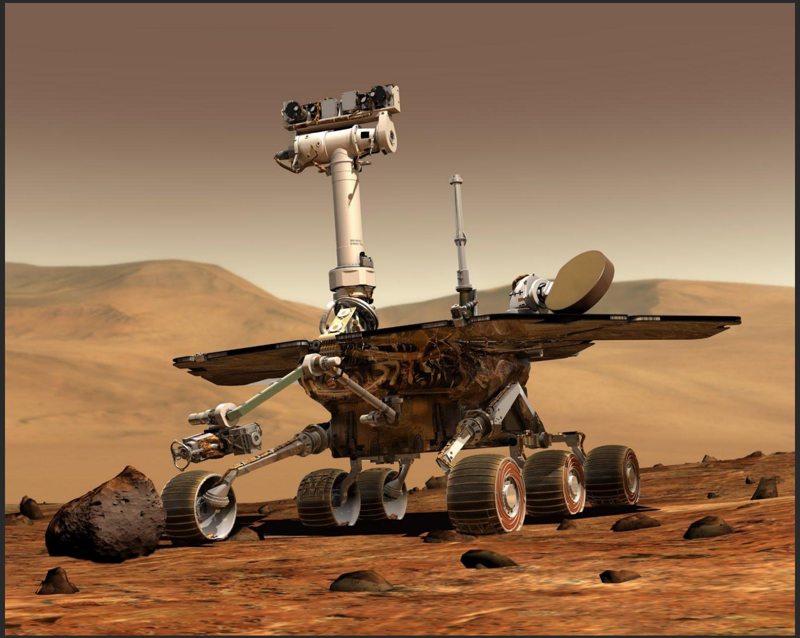
STARCRAFT II

HALO

## Cool stuff written in C++

The F-35 Lightning II (Joint Strike Fighter) relies extensively on C++

The Spirit rover was operational for over 6 years when the mission was only planned to run for around 3 months



## Why CS 106L?

## Goals of CS 106L

1. Learn what features are out there in C++ and why they exist.

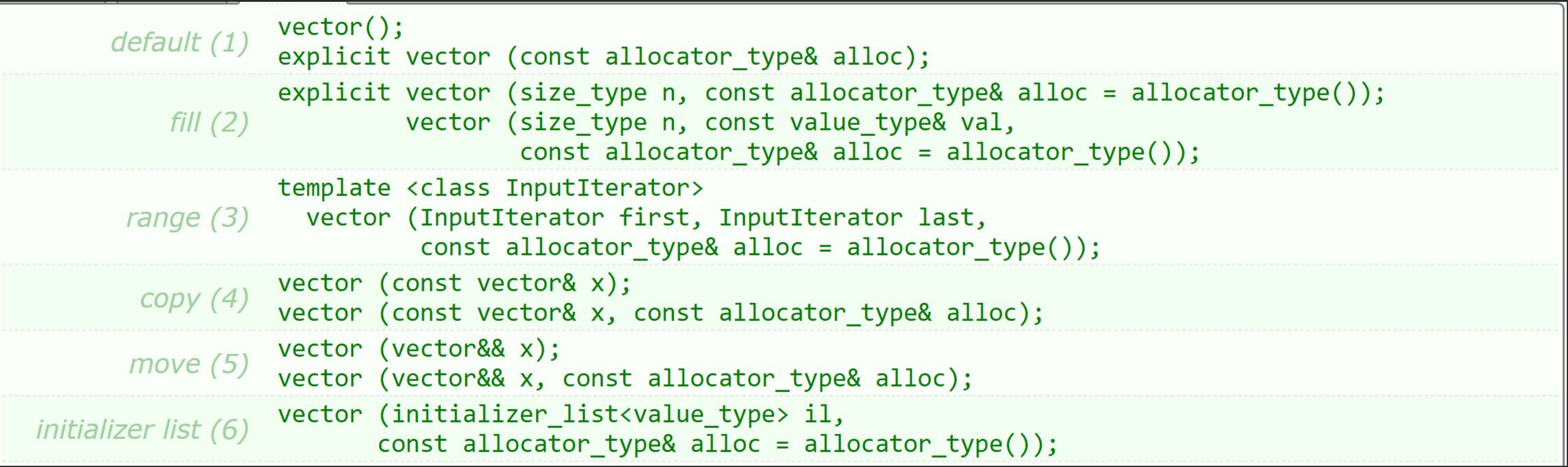
2. Become comfortable with reading C++ documentation.

3. Become familiar with the design philosophy of modern C++.

NOT: memorize the syntax of C++.

# C++ documentation is very "expert friendly".

```
vector<int> nums; // the first default constructor
```



| | |
|---|---|
| *default (1)* | `vector();`<br>`explicit vector (const allocator_type& alloc);` |
| *fill (2)* | `explicit vector (size_type n, const allocator_type& alloc = allocator_type());`<br>`vector (size_type n, const value_type& val, const allocator_type& alloc = allocator_type());` |
| *range (3)* | `template <class InputIterator> vector (InputIterator first, InputIterator last, const allocator_type& alloc = allocator_type());` |
| *copy (4)* | `vector (const vector& x);`<br>`vector (const vector& x, const allocator_type& alloc);` |
| *move (5)* | `vector (vector&& x);`<br>`vector (vector&& x, const allocator_type& alloc);` |
| *initializer list (6)* | `vector (initializer_list<value_type> il, const allocator_type& alloc = allocator_type());` |

## Class Schedule Outline

| | |
|---|---|
| Basics Week 1: | Compilation and Structures |
| Basics Week 2: | References and Streams |
| STL Week 3: | Containers and Iterators |
| STL Week 4-5: | Templates and Algorithms |
| Templates Week 5: | Template Classes |
| Templates Week 6: | Metaprogramming |
| Class Design Week 6: | Const Correctness and Operators |
| Class Design Week 7: | Special Member Functions |
| Class Design Week 8: | RAII |
| Bonus Topics Week 9: | Multithreading and C++20 |

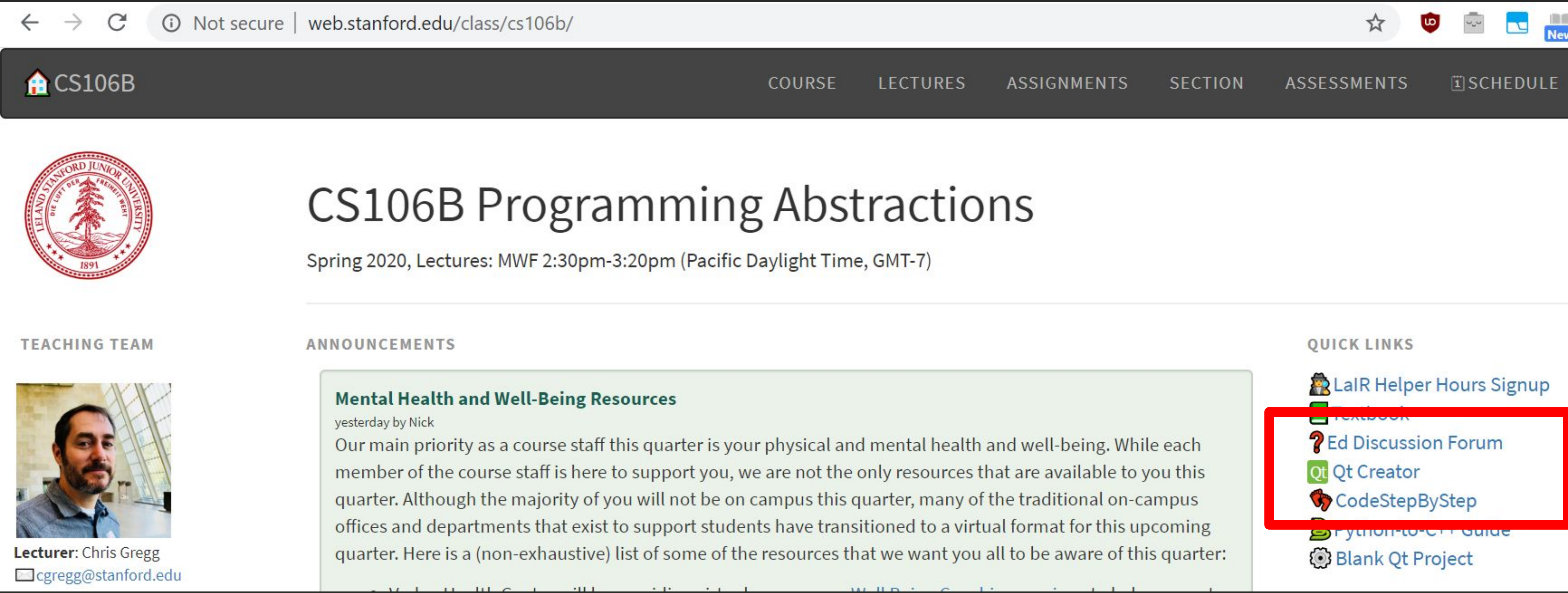
## Logistics

## Logistics

| | |
|---|---|
| Lecture: | M/W 4:30-5:50 on Zoom, weeks 1-9 |
| Website: | https://cs106l.stanford.edu |
| Getting Help: | Office Hours, Piazza, do not use LaIR |
| Assignments: | 2 assignments, complete both for credit |
| Late Days: | Earn 24-hour late days through surveys |
| Development: | Qt Creator (from CS 106B) |
| Honor Code: | Don’t cheat. Same rules as CS 106B. |

Piazza: https://piazza.com/stanford/spring2020/cs106l

# QT Creator Setup



Not secure | web.stanford.edu/class/cs106b/

CS106B

COURSE LECTURES ASSIGNMENTS SECTION ASSESSMENTS SCHEDULE

CS106B Programming Abstractions

Spring 2020, Lectures: MWF 2:30pm-3:20pm (Pacific Daylight Time, GMT-7)

TEACHING TEAM

**Lecturer**: Chris Gregg
cgregg@stanford.edu

ANNOUNCEMENTS

**Mental Health and Well-Being Resources**

yesterday by Nick

Our main priority as a course staff this quarter is your physical and mental health and well-being. While each member of the course staff is here to support you, we are not the only resources that are available to you this quarter. Although the majority of you will not be on campus this quarter, many of the traditional on-campus offices and departments that exist to support students have transitioned to a virtual format for this upcoming quarter. Here is a (non-exhaustive) list of some of the resources that we want you all to be aware of this quarter:

QUICK LINKS

- LaIR Helper Hours Signup
- Textbook
- Ed Discussion Forum
- Qt Creator
- CodeStepByStep
- Python-to-C++ Guide
- Blank Qt Project

## Survey

https://forms.gle/MahBUdB54mfqWnQQ6

= +1 late day!

## History of C++

## Some C++ Code

```cpp
#include <iostream>

int main() {
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

## Also Some C++ Code

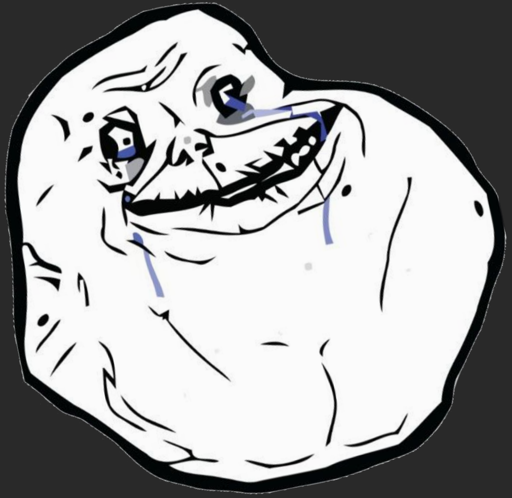
```
#include "stdio.h"
#include "stdlib.h"

int main(int argc, char *argv) {
    printf("%s", "Hello, world!\n");
    return EXIT_SUCCESS;
}
```

## ...Also (Technically) Some C++ Code

```
#include "stdio.h"
#include "stdlib.h"

int main(int argc, char *argv) {
   asm(  "sub    $0x20,%rsp\n\t"
         "movabs $0x77202c6f6c6c6548,%rax\n\t"
         "mov    %rax,(%rsp)\n\t"
         "movl   $0x646c726f, 0x8(%rsp)\n\t"
         "movw   $0x21, 0xc(%rsp)\n\t"
         "movb   $0x0,0xd(%rsp)\n\t"
         "leaq    (%rsp),%rax\n\t"
         "mov    %rax,%rdi\n\t"
         "call  __Z6myputsPc\n\t"
         "add    $0x20, %rsp\n\t"
     );
    return EXIT_SUCCESS;
}
```



# C++ History: Assembly

```
section     .text
global      _start                       ;must be declared for linker (ld)

_start:                                  ;tell linker entry point

    mov     edx,len                      ;message length
    mov     ecx,msg                      ;message to write
    mov     ebx,1                        ;file descriptor (stdout)
    mov     eax,4                        ;system call number (sys_write)
    int     0x80                         ;call kernel
    mov     eax,1                        ;system call number (sys_exit)
    int     0x80                         ;call kernel

section     .data
msg     db  'Hello, world!',0xa          ;our dear string
len     equ $ - msg                      ;length of our dear string
```

## C++ History: Assembly

Benefits:

- Unbelievably simple instructions
- Extremely fast (when well-written)
- Complete control over your program

Why don’t we always use assembly?

# C++ History: Assembly

```
section     .text
global      _start                       ;must be declared for linker (ld)

_start:                                  ;tell linker entry point

    mov     edx,len                      ;message length
    mov     ecx,msg                      ;message to write
    mov     ebx,1                        ;file descriptor (stdout)
    mov     eax,4                        ;system call number (sys_write)
    int     0x80                         ;call kernel
    mov     eax,1                        ;system call number (sys_exit)
    int     0x80                         ;call kernel

section     .data
msg     db  'Hello, world!',0xa          ;our dear string
len     equ $ - msg                      ;length of our dear string
```

## C++ History: Assembly

Drawbacks:

- A lot of code to do simple tasks
- Hard to understand
- Extremely unportable

## C++ History: Invention of C

Problem: computers only understand assembly.*

Idea:

- Source code can be written in a more intuitive language
- An additional program can convert it into assembly

$\uparrow$

This is called a compiler!

## C++ History: Invention of C

T&R created C in 1972, to much praise.

C made it easy to write code that was

- Fast
- Simple
- Cross-platform

Learn to love it in CS107!

Ken Thompson and Dennis Ritchie, creators of the C language.

## C++ History: Invention of C

C was popular since it was simple.

This was also its weakness:

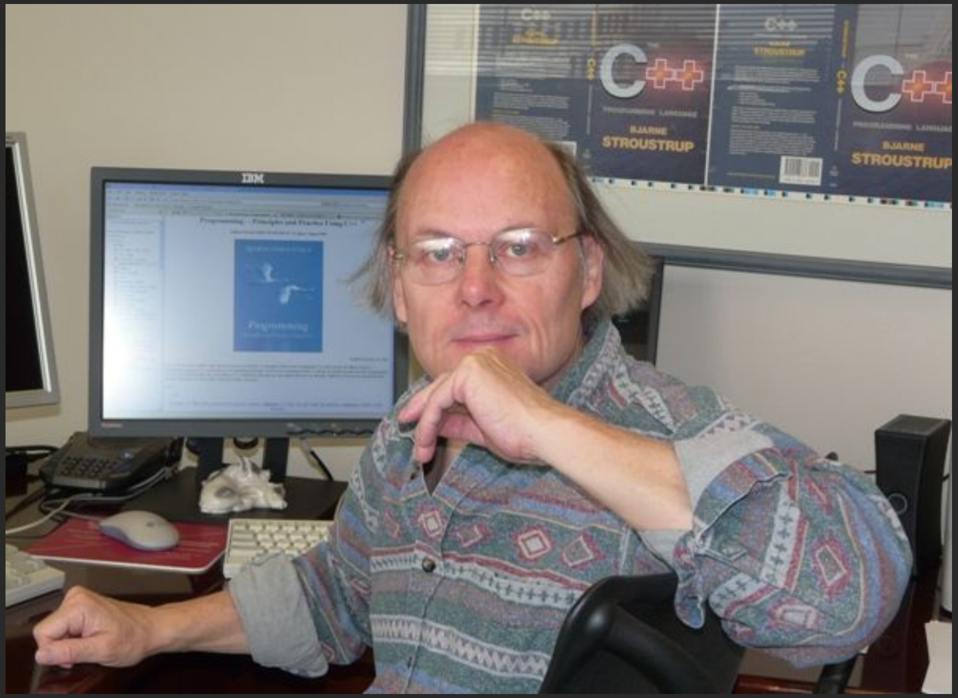
- No objects or classes
- Difficult to write code that worked generically
- Tedious when writing large programs

## C++ History: Welcome to C++!

In 1983, the first vestiges of C++ were created by Bjarne Stroustrup.

He wanted a language that was:

- Fast
- Simple to Use
- Cross-platform
- Had high level features

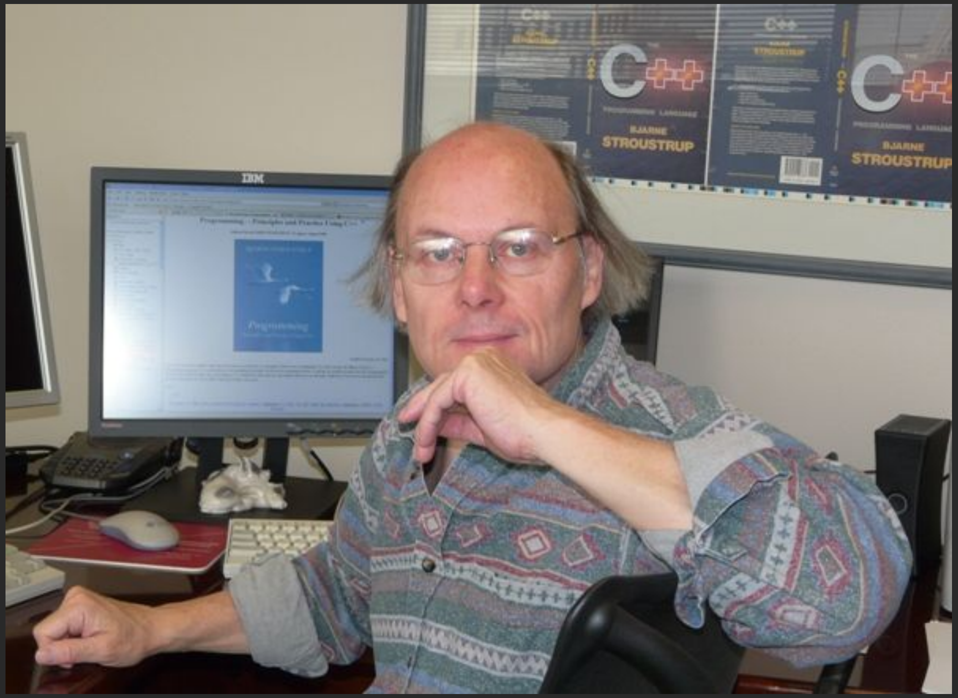


## C++ History: Welcome to C++!

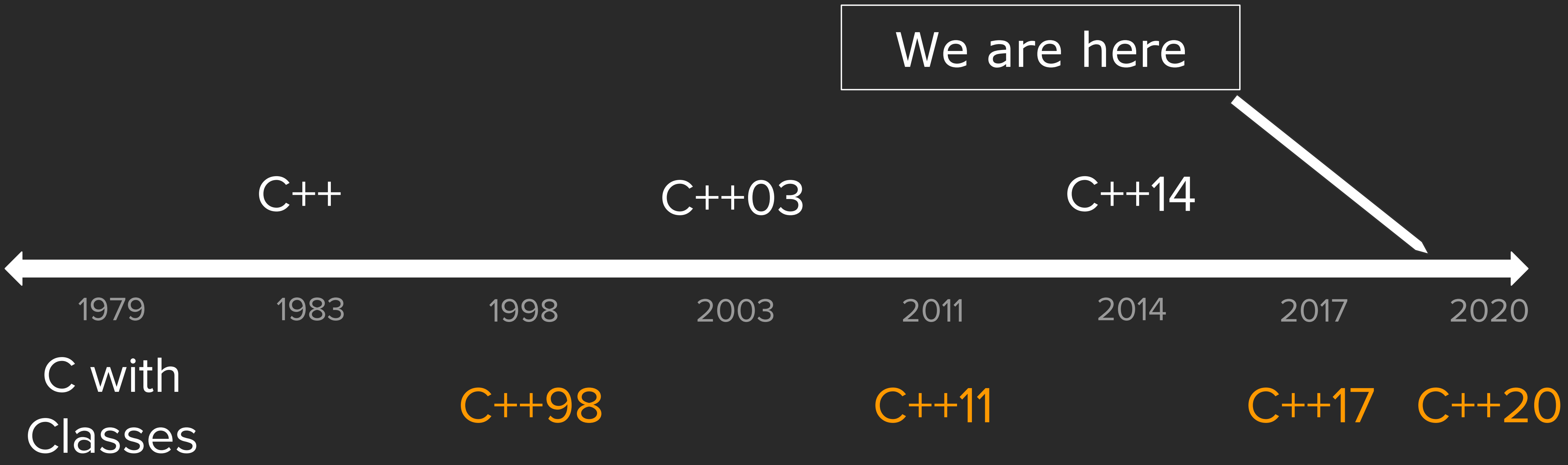
In 1983, the first vestiges of C++ were created by Bjarne Stroustrup.

He wanted a language that was:

- Fast
- Simple to Use
- Cross-platform
- Had high level features



# C++ History: Evolution of C++



## Design Philosophy of C++

## Design Philosophy of C++

- Allow the programmer full control, responsibility, and choice if they want it.
- Express ideas and intent directly in code.
- Enforce safety at compile time whenever possible.
- Do not waste time or space.
- Compartmentalize messy constructs.

## Design Philosophy of C++

- Multi-paradigm
- Express ideas and intent directly in code.
- Safety
- Efficiency
- Abstraction

## Questions so far?

2-min stretch break!

## Example

Our first C++ program

## Today:
## Command Line Compilation

## CL Compilation

For our assignments and in CS106B, you'll use QT Creator to compile your code. However, QT Creator isn't the only way to compile C++ code!

Today we will briefly cover how to do this in the terminal.

First we should understand how C++ compilation works.

## CL Compilation

1. Preprocessor - Deals with `#include`, `#define`, etc directives
2. Compiler - Converts C++ source code into assembly
3. Assembler - Turns assembled code into object code (.o files)
4. Linker - Object files are linked together to make an executable program

## Preprocessor

Responsible for everything starting with a `#`

```
#include

#define

#ifndef

#pragma
```

## Compilers

Converts each .cpp source file into assembly.

This process is localised to each file.

Outputs .s files

## Assembler

Turns previously generated assembly code into object code.

Outputs .o files.

Still no intercommunication between separate cpp files.

## Linker

Combines all the separate object files into one executable file.

In previous phases we only looked at one file at a time.

The linker is the first place where files are combined.

## Linker

Linker checks that every declared function has an implementation.

This is why you get errors like:

```
Linker error: symbols not found for
architecture x86
Linker error: duplicate symbols found
for architecture x86
```

## Let’s try it ourselves!

We will use `g++` as our compiler.

Basic usage:

```
g++ main.cpp otherFile.cpp -o execFileName
```

## Let’s try it ourselves!

We will use three common compiler flags:

-std=c++14

Enable C++14 support

-g

Add debugging information to the output

-Wall

Turn on most compiler warnings

## Example

Command-Line Compilation in Action

# Next time

Structures