

CS107, Lecture 1

Welcome to CS107!

reading:

[Course Syllabus](#)

Bryant & O'Hallaron, Ch. 1 (skim)

[Honor Code and Collaboration Page](#)

Asking Questions

- Feel free to raise your hand at any time with a question
- If you prefer, you can anonymously post your question in the Ed forum thread for each day's lecture
- We'll monitor the thread throughout the lecture and answer them as they're posted



Visit Ed:

<https://edstem.org/us/courses/70267/discussion>

Today's thread:

<https://edstem.org/us/courses/70267/discussion/5940864>

What questions best summarize CS107?

How? Why?

CS107: How/Why?

Our CS106 courses teach you how to solve problems using high level programming languages. CS107 digs deeper and allows us to understand how and why programs do what they do.

- **How** is program data represented on the hardware?
- **How** does the heap work, and how is it implemented?
- **How** does a computer actually run our code?
- **How** does a program map onto the various components of a computer?
- **Why** is my program doing one thing when I expected it to do something else?

Studying programming at this level reveals why systems work as they do and helps us become better software engineers.

CS107 and Programming Experience

- CS107 will further develop and improve your programming abilities and provide additional coding mileage.
- CS107 focuses quite a bit on **debugging strategies** that get to the root of why something isn't working as expected.
- For the duration of the quarter, we'll emphasize how to become a better debugger, how to write professional-grade code, and how to further refine your software development skills.

CS107 Learning Goals

The goals for CS107 are for students

to acquire a **fluency** with

- pointers and memory and how to make use of them when coding in C
- an executable's address space and runtime behavior

to acquire **competency** with

- the translation between C and assembly
- the implementation of programs that respect the limits of computer arithmetic
- the ability to identify bottlenecks and improve runtime performance
- the ability to navigate your own Unix development environment
- several ethical frameworks to be considered when designing and implementing software

and, to gain some **exposure** to

- the basics of computer architecture
- compilers and disassemblers and how they work

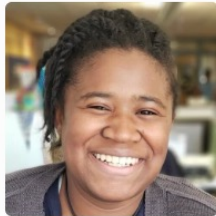
Course Overview

1. **Bits and Bytes** - *How can a computer represent integer values?*
2. **Characters and C Strings** - *How can a computer represent and manipulate more complex data types like text?*
3. **Pointers, Stack Memory and Heap Memory** – *How can we effectively manage all forms of memory in our programs?*
4. **Generics** - *How can we leverage our understanding of memory and data representations to write code that works with any data type?*
5. **Assembly** - *How does a computer compile, interpret, and execute C programs? And what does assembly code actually look like?*
6. **Heap Allocators** - *How do core memory allocation functions like `malloc` and `free` work? Are the built-in versions always good enough?*

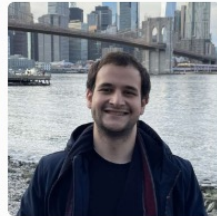
Teaching Team



Jerry Cain



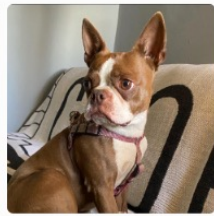
Ola Adekola



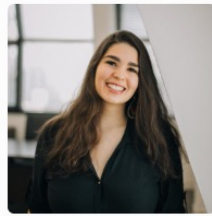
Arman Aydin



Esteban Barrero-Hernandez



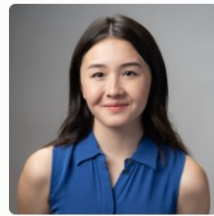
Doris



Lara Franciulli



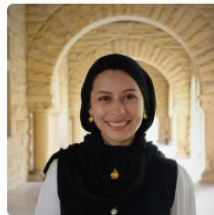
Salman Abdullah



Alison Rogers



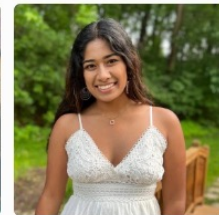
Stephanie Hurtado



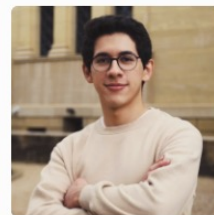
Carolina Borbon Miranda



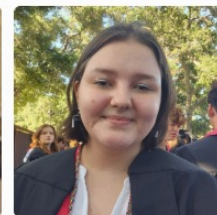
Ben Yan



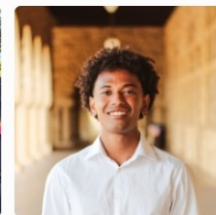
Anushka Rawat



Julian Rodriguez Cardenas



Andreea Jitaru



Abraham Yosef

Jerry Cain biography (jerry@cs.stanford.edu):

- Chemistry undergrad MIT, originally a chemistry Ph.D. student here, switched to CS in 1994
- Taught CS107 for the first time in Autumn 1999, have taught it more than 25 times already

Companion Class: CS107ACE

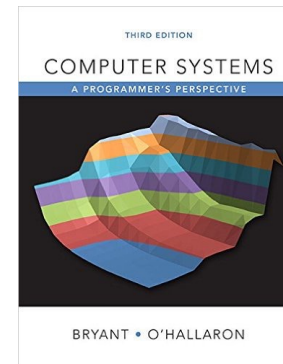
- **CS107ACE** is an extra, single-unit discussion section providing additional support, practice, and instruction.
- Class meets on Thursdays from 11:30am – 1:20pm in STLC 118.
- Admission is by [application](#), and everyone is welcome to apply. In past quarters, most students who've applied have been accepted into CS107ACE.



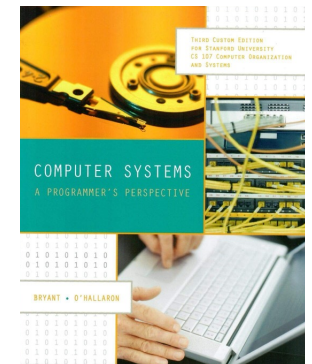
Shray Alag

Textbook(s)

- *Computer Systems: A Programmer's Perspective* by Bryant & O'Hallaron, **3rd Edition**
 - **3rd edition matters** – important updates to content
 - Stanford Library has generously scanned **all** readings for CS107 under "fair use" (private study, scholarship, research). [**Canvas -> Files**]. Please do not distribute.
 - If you want more context, you may want to purchase a full copy.
- A C programming reference of your choice
 - *The C Programming Language* by Kernighan and Ritchie (free link on course website Resources page)
 - Other C programming books, websites, or reference sheets



Full textbook



CS107 full chapters




canvas

CS107-specific readings

The textbook (and C programming references) are **excellent** resources for the course, especially post midterm!

Course Structure

- Lectures: understand concepts, see demos
- Labs: learn tools, study code, discuss with peers  Great preamble to homework!
- Assignments: build programming skills, synthesize lecture/lab content

	Monday	Wednesday	Friday
Week N		Lecture: part A	Lecture: part B
Week N + 1	Lecture: part C		

CS107 labs will meet at various times on Wednesdays and Thursday, and you'll use that time to exercise more fully digest the prior week's material

- **assign0**: out on Wednesday, due the following Wednesday (covers today's, Wednesday's, and some of Friday's material)

Grading

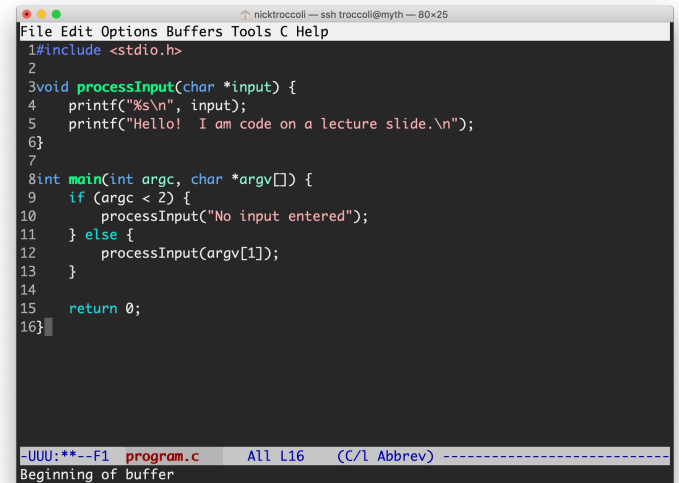
*****	40%	Assignments
**	10%	Lab Participation
****	20%	Midterm Exam
*****	30%	Final Exam

Grading

*****	40%	Assignments
**	10%	Lab Participation
***	20%	Midterm Exam
*****	30%	Final Exam

Assignments

- 7 programming assignments completed individually using **Unix command line tools**
 - Free software, already installed on **myth** machines / available on course website
 - We supply starter projects for each assignment
- Graded on **functionality** and **style**
 - Functionality graded using *automated tools*, given as point score – minimal CA review
 - Style graded via manual CA code review, with *occasional automated tests*, given as bucket score
 - Grades posted on CS107 website and via email



```
File Edit Options Buffers Tools C Help
1#include <stdio.h>
2
3void processInput(char *input) {
4    printf("%s\n", input);
5    printf("Hello! I am code on a lecture slide.\n");
6}
7
8int main(int argc, char *argv[]) {
9    if (argc < 2) {
10        processInput("No input entered");
11    } else {
12        processInput(argv[1]);
13    }
14
15    return 0;
16}

-UUU:**--F1 program.c All L16 (C/l Abbrev) -----
Beginning of buffer
```

The Style Bucket System

+	An outstanding job: rarely given, particularly on early assignments
ok	A solid job: solid effort, but opportunities for improvement.
-	Conveys some effort and understanding but has larger problems that would need to be addressed before checking into a professional code base.
--	Suffers from many issues and represents minimally passing work.
0	No work submitted or barely changes any of the supplied starter code.

Assignment Late Policy

- **All programming assignments are due a minute before midnight, Stanford time, on whatever day they fall due.**
- If you need to submit an assignment after the deadline, you still can. But doing so places a cap on the maximum number of points you can get, depending on how late the submission.
 - Submit an assignment before the published deadline? You can get **100%**. Seems right.
 - Submit after the deadline, but within 24 hours? Functionality score is capped at 95%.
 - What does **capped** mean here? It means that all scores between 96% and 100% are demoted to 95%, but all other scores are left alone.
 - Submit an assignment between 24 and 48 hours post-deadline? Score is capped at 90%.
 - Submit an assignment between 48 and 72 hours post-deadline? Score is capped at 85%.
 - Unless you've made prior arrangements with us, no work is accepted after 72 hours.



Question Break

Any questions about the overall learning goals, textbook or assignments?

Grading

*****	40%	Assignments
**	10%	Lab Participation
****	20%	Midterm Exam
*****	30%	Final Exam

Lab Sections

- Weekly 80-minute, in-person labs led by CAs, starting *next* week, offered on Wednesdays and Thursdays.
- Hands-on practice with lecture material and course concepts.
- Graded on attendance + participation. You're literally for literally showing up.
- SCPD students complete lab work (and take the exams) remotely. More info in [SCPD Handout](#).
- Lab preference submissions open **Wednesday 01/08 at 5PM PST** but are **not first-come/first-serve**, so take your time finalizing your schedule before submitting. You may submit your preferences anytime until **Sunday 01/12 at 5PM PST**. Sign up on the course website after email announcement on Wednesday.

Grading

*****	40%	Assignments
**	10%	Lab Participation
***	20%	Midterm Exam
*****	30%	Final Exam

Exams

- **Midterm exam:** Wednesday, February 12th, 7:00 - 9:00PM outside of class
Wednesday, February 12th, 3:30 - 5:30PM if first time can't work
 - Contact the course staff by 11:59PM on Friday, February 7th if you have an academic or extracurricular conflict with **both** times and absolutely cannot make either.
- **Final exam:** Monday, March 17th, 3:30 - 6:30PM
 - If and only if you hold a conflict with this time because of a competing final exam—that is, you're taking two MWF 10:30AM classes and that other class also have a final—can you take the final on the same day, on March 17th, from 7:00 – 10:00PM instead.
- Both exams are **closed book, closed notes, and closed electronic device**. You'll be provided with a reference sheets with common function prototypes and other pertinent information not easily memorized.
- SCPD students have 48-hour window during which they can complete exams.
- Both are administered as old-school, pencil-and-paper exams.

Grading

*****	40%	Assignments
**	10%	Lab Participation
***	20%	Midterm Exam
*****	30%	Final Exam

Read our full course policies document:
<https://cs107.stanford.edu/syllabus.html>



Question Break

What questions do you have about labs or exams?

Getting Help

- Post on the **Discussion Forum**
 - Online discussion forum for students—post questions, answer other student questions
 - Best for course material discussions, policy questions, debugging questions or general assignment concerns. **Don't publicly post assignment code under any circumstances.**
- Visit **Office Hours**
 - Chat about course topics or just hang out
 - Sign up in queues for 1:1 CA help; schedule will be posted on course website this week
 - Best for **group work, coding/debugging questions (with CAs, of course) or longer discussions about course material**
- **Email** the Course Staff
 - cs107-win2425-staff@lists.stanford.edu or individual CAs/instructor
 - Best for **private concerns** (e.g., grading questions, academic accommodations, etc.)

Updated Stanford Honor Code

- The [Honor Code](#) is an undertaking of the Stanford academic community, individually and collectively. Its purpose is to uphold a culture of academic honesty.
 - Students will support this culture of academic honesty by neither giving nor accepting unpermitted academic aid in any work that serves as a component of grading or evaluation, including assignments, examinations, and research.
 - Instructors will support this culture of academic honesty by providing clear guidance, both in their course syllabi and in response to student questions, on what constitutes permitted and unpermitted aid. Instructors will also not take unusual or unreasonable precautions to prevent academic dishonesty.
 - Students and instructors will also cultivate an environment conducive to academic integrity. While instructors alone set academic requirements, the Honor Code is a community undertaking that requires students and instructors to work together to ensure conditions that support academic integrity.

above drawn verbatim from: <https://communitystandards.stanford.edu/policies-guidance/honor-code>

It is your responsibility to ensure you have read and are familiar with the honor code guidelines shared via the main page of the CS107 course website. Please read them and come speak with us if you have any questions or concerns.

<https://cs107.stanford.edu/collaboration>

Honor Code and CS107

- Do your part to be transparent about how you get your work done:
 - Indicate any assistance received on homework (books, friends, etc.).
 - Do not look at other people's solution code or answers, and do not share your solutions with others or post them on the web or our Ed forum.
 - Report any online solutions you incidentally come across to Jerry.
- Assignment submissions are regularly examined using software tools.
- If you need help, please contact us and we will do what we can to help you make good choices.
 - We do not want you to feel any pressure to violate the Honor Code because you need to comfortably arrive at an outcome that you're happy with and think will impress others.
 - If you realize you've made a mistake, you may retract one or more assignment submissions at any time, no questions asked, up to the time the final exam begins.

<https://cs107.stanford.edu/collaboration>

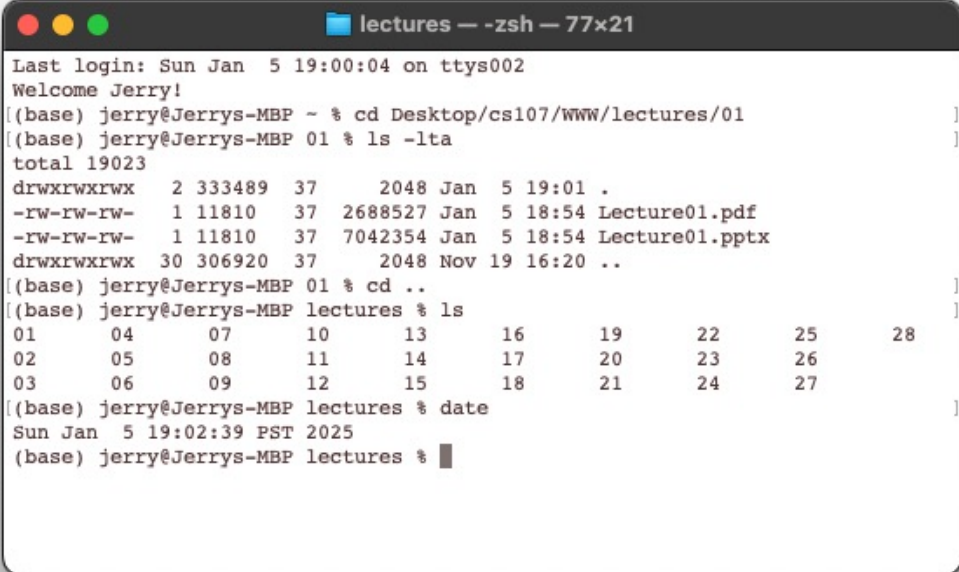


Question Break

What questions do you have about course support or the honor code?

What is Unix?

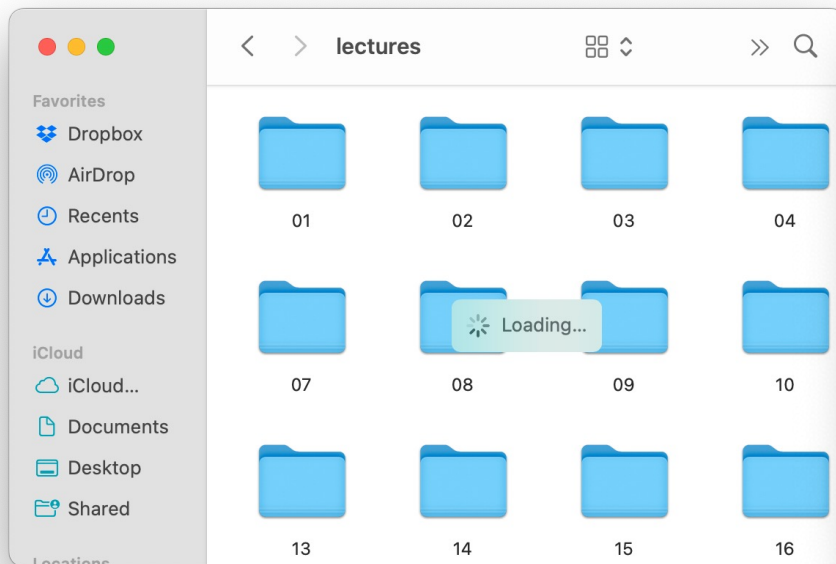
- **Unix:** provides a set of standards and many tools used to write software.
 - **macOS** and **Linux** are operating systems built on top of Unix
- You navigate a Unix system using the **command line** (aka "the terminal")
- All Unix systems work using the same tools and commands



```
lectures — -zsh — 77x21
Last login: Sun Jan  5 19:00:04 on ttys002
Welcome Jerry!
[(base) jerry@Jerrys-MBP ~ % cd Desktop/cs107/www/lectures/01
[(base) jerry@Jerrys-MBP 01 % ls -lta
total 19023
drwxrwxrwx  2 333489  37    2048 Jan  5 19:01 .
-rw-rw-rw-  1 11810  37 2688527 Jan  5 18:54 Lecture01.pdf
-rw-rw-rw-  1 11810  37 7042354 Jan  5 18:54 Lecture01.pptx
drwxrwxrwx 30 306920  37    2048 Nov 19 16:20 ..
[(base) jerry@Jerrys-MBP 01 % cd ..
[(base) jerry@Jerrys-MBP lectures % ls
01    04    07    10    13    16    19    22    25    28
02    05    08    11    14    17    20    23    26
03    06    09    12    15    18    21    24    27
[(base) jerry@Jerrys-MBP lectures % date
Sun Jan  5 19:02:39 PST 2025
(base) jerry@Jerrys-MBP lectures %
```

What is the Command Line?

- Think of the **command line** as a text-based interface to navigate a computer's file system and launch applications. It's a popular, lightweight alternative to the more traditional Graphical User Interface (GUI) you've likely used before.



Graphical User Interface

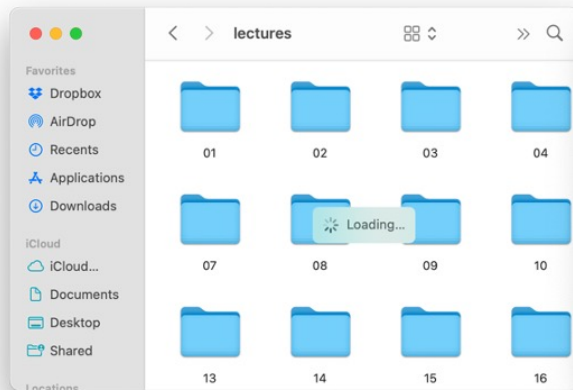
```
Last login: Sun Jan 5 19:00:04 on ttys002
Welcome Jerry!
[(base) jerry@Jerrys-MBP ~ % cd Desktop/cs107/www/lectures/01
[(base) jerry@Jerrys-MBP 01 % ls -lta
total 19023
drwxrwxrwx  2 333489  37      2048 Jan  5 19:01 .
-rw-rw-rw-  1 11810   37 2688527 Jan  5 18:54 Lecture01.pdf
-rw-rw-rw-  1 11810   37 7042354 Jan  5 18:54 Lecture01.pptx
drwxrwxrwx 30 306920  37      2048 Nov 19 16:20 ..
[(base) jerry@Jerrys-MBP 01 % cd ..
[(base) jerry@Jerrys-MBP lectures % ls
01    04    07    10    13    16    19    22    25    28
02    05    08    11    14    17    20    23    26
03    06    09    12    15    18    21    24    27
[(base) jerry@Jerrys-MBP lectures % date
Sun Jan  5 19:02:39 PST 2025
[(base) jerry@Jerrys-MBP lectures %
```

Command-line interface

Command Line Vs. GUI

Just like a GUI file explorer interface, a terminal interface:

- shows you a **specific place** on your computer at any given time.
- lets you go **into folders** and **ascend out of folders**.
- lets you **create new** files and **edit** existing one.
- lets you **execute programs**.



Graphical User Interface

A screenshot of a terminal window titled 'lectures -- zsh -- 77x21'. It shows a sequence of commands and their outputs:

```
Last login: Sun Jan 5 19:00:04 on ttys002
Welcome Jerry!
(base) jerry@Jerrys-MBP ~ % cd Desktop/cs107/WWW/lectures/01
(base) jerry@Jerrys-MBP 01 % ls -lta
total 19023
drwxrwxrwx  2 333489  37   2048 Jan  5 19:01 .
-rw-rw-rw-  1 11810   37 2688527 Jan  5 18:54 Lecture01.pdf
-rw-rw-rw-  1 11810   37 7042354 Jan  5 18:54 Lecture01.pptx
drwxrwxrwx 30 306920  37   2048 Nov 19 16:20 ..
(base) jerry@Jerrys-MBP 01 % cd ..
(base) jerry@Jerrys-MBP lectures % ls
01  04  07  10  13  16  19  22  25  28
02  05  08  11  14  17  20  23  26
03  06  09  12  15  18  21  24  27
(base) jerry@Jerrys-MBP lectures % date
Sun Jan  5 19:02:39 PST 2025
(base) jerry@Jerrys-MBP lectures %
```

Command-line interface

Why Use Unix / the Command Line?

- You can navigate almost any device using the same tools and commands:
 - Laptops and desktops
 - Embedded devices (Raspberry Pi, etc.)
 - Mobile Devices (Android, etc.)
 - Microwaves, dishwashers, air conditioners, cameras
- Used frequently by software engineers:
 - **Web development:** running servers and web tools on servers
 - **Machine learning:** processing data on servers, running algorithms
 - **Systems:** writing operating systems, networking code and embedded software
 - **Mobile Development:** running tools, managing libraries
- We'll use Unix and the command line to implement and execute our programs.

Unix Commands To Try

- **cd** – change directories
- **ls** – list directory contents
- **mkdir** – create a new directory
- **emacs** – open text editor
- **rm** – irreversibly delete file or folder
- **man** – read documentation for standard C functions

See the course website for more commands and a complete reference.

Demo: Using Unix and the Command Line



Get up and running with our guide:
<http://cs107.stanford.edu/getting-started.html>

Learning Unix and the Command Line

- Using Unix and the command line is at first very intimidating.
 - It looks retro! It **is** retro!
 - How do I know what to type?
- It's like learning a new language.
 - At first, you may have to repeatedly look things up (**resources** on course website!)
 - It's important to spend as much time as possible (during labs and assignments) building muscle memory with the tools



Question Break

Get up and running with our guide:
<http://cs107.stanford.edu/getting-started.html>

The C Language

C was created around 1970 to simplify implementation of Unix and core utilities.

- Part of the C/C++/Java family of languages (C++ and Java were created later)
- Design principles:
 - Small, simple abstractions over hardware
 - Minimalist, language is difficult to master, but very small
 - Prioritizes efficiency and simplicity over safety and higher-level abstractions

C vs. C++ and Java

They all share:

- Syntax
- Basic data types
- Arithmetic, relational, and logical operators

C limitations:

- No advanced features like operator overloading, default arguments, true pass by reference, classes and objects, ADTs, etc.
- No extensive libraries (no graphics, networking, etc.) – small language footprint means not much to learn 😊
- Weak compiler and virtually zero runtime checks. This is a double-edged sword.

Programming Language Philosophies

C is procedural: you write functions, rather than define new variable types with classes and invoke methods. **C is small, fast and efficient.**

C++ is procedural, but with objects: you still write functions, and define new variable types with classes, and call methods on objects.

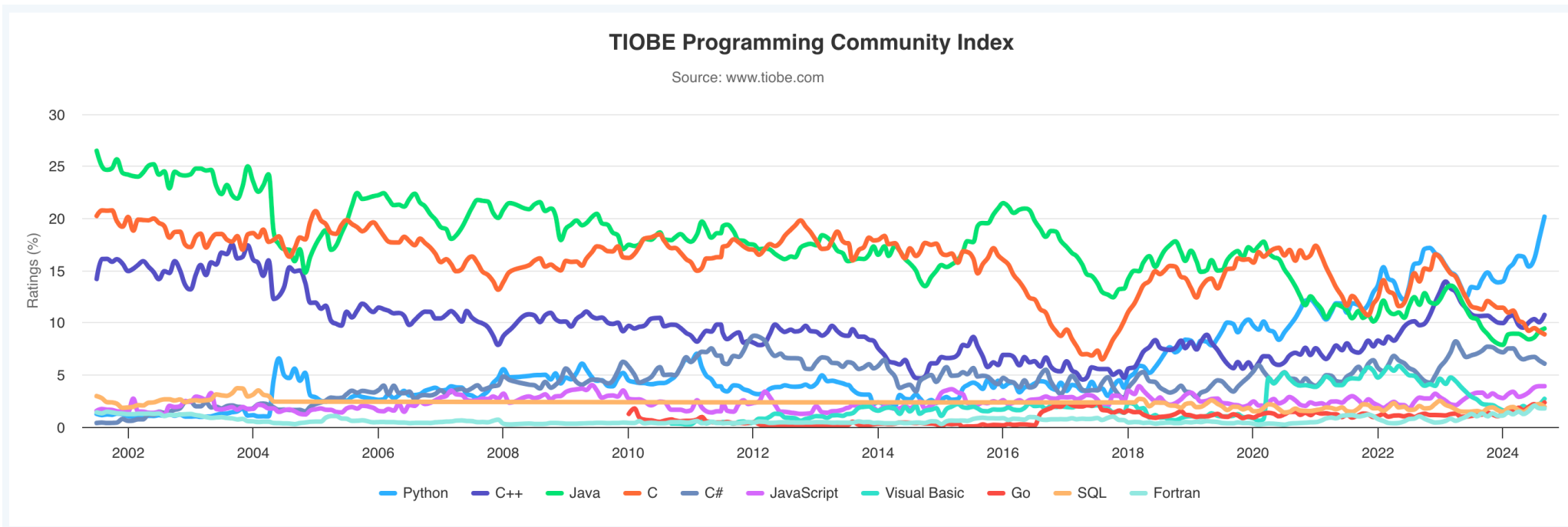
Python is procedural, but dynamically typed: you still write functions and call methods on objects but traditionally omit data types when coding. The development process is very different.

Java is fully object-oriented: virtually everything is an object and everything you write must conform to the object-oriented paradigm.

Why C?

- Many tools (and even other languages, like Python) are written using C.
- C is a language of choice for fast, highly efficient programs.
- C is popular for systems programming (operating systems, networking, etc.)
- C allows you to more easily manipulate the underlying system.
- Alternatives to C are emerging (e.g., Rust, Go), but they're substantially more complicated and not quite right for those new to systems programming.

Programming Language Popularity



<https://www.tiobe.com/tiobe-index/>

Our First C Program

```
/*  
 * hello.c  
 * This program prints a welcome message  
 * to the user.  
 */  
#include <stdio.h> // for printf  
  
int main(int argc, char *argv[]) {  
    printf("Hello, world!\n");  
    return 0;  
}
```

Our First C Program

```
/*  
 * hello.c  
 * This program prints a welcome message  
 * to the user.  
 */
```

```
#include <stdio.h> // for printf
```

```
int main(int argc, char *argv[]) {  
    printf("Hello, world!\n");  
    return 0;  
}
```

Program comments

You can write block or inline comments.

Our First C Program

```
/*  
 * hello.c  
 * This program prints a welcome message  
 * to the user.  
 */  
#include <stdio.h> // for printf  
  
int main(int argc, char *argv[]) {  
    printf("Hello, world!\n");  
    return 0;  
}
```

Import statements

C libraries are written with angle brackets.
Local libraries use quotes instead:
`#include "wordle-utils.h"`

Our First C Program

```
/*  
 * hello.c  
 * This program prints a welcome message  
 * to the user.  
 */  
#include <stdio.h> // for printf
```

```
int main(int argc, char *argv[]) {  
    printf("Hello, world!\n");  
    return 0;  
}
```

Main function – entry point for the program, and should always return a small integer (0 = success)

Our First C Program

```
/*  
 * hello.c  
 * This program prints a welcome message  
 * to the user.  
 */  
#include <stdio.h> // for printf  
  
int main(int argc, char *argv[]) {  
    printf("Hello, world!\n");  
    return 0;  
}
```

Main parameters – **main** takes two parameters, both relating to the *command line arguments* used to execute the program.

argc is the *number* of arguments in **argv**
argv is an *array of arguments* (**char *** is C string)

Our First C Program

```
/*
 * hello.c
 * This program prints a welcome message
 * to the user.
 */
#include <stdio.h> // for printf

int main(int argc, char *argv[]) {
    printf("Hello, world!\n");
    return 0;
}
```

printf – prints output to the screen

Console Output: printf

```
printf(text, arg1, arg2, arg3,...);
```

`printf` makes it easy to print out the values of variables or expressions.

If you include *placeholders* in your printed text, `printf` will replace each placeholder with the values of subsequent parameters, passed after the text.

`%s` (string)

`%d` (integer)

```
// Example
```

```
char *department = "CS";
```

```
int number = 107;
```

```
printf("You are in %s%d", department, number);
```

```
// You are in CS107
```



Familiar Syntax

```
int x = 42 + 7 * -5;           // variables, types
double pi = 3.14159;
char c = 'Q';                 /* two comment styles */

for (int i = 0; i < 10; i++) { // for loops
    if (i % 2 == 0) {         // if statements
        x += i;
    }
}

while (x > 0 && c == 'Q' || b) { // while loops, logic
    x = x / 2;
    if (x == 42) {
        return 0;
    }
}

binky(x, 17, c);             // function call
```


Boolean Variables

To declare Booleans, (e.g. `bool b = ____`), you must include `stdbool.h`:

```
#include <stdio.h>    // for printf
#include <stdbool.h>  // for bool

int main(int argc, char *argv[]) {
    bool x = argc > 2 && argv[argc - 1][0] != 'A';
    if (x) {
        printf("Hello, world!\n");
    } else {
        printf("Howdy, world!\n");
    }
    return 0;
}
```

Boolean Expressions

C treats a nonzero value as true, and a zero value as false:

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    int x = 5;
    if (x) { // true
        printf("Hello, world!\n");
    } else {
        printf("Howdy, world!\n");
    }
    return 0;
}
```



Question Break

Writing, Debugging and Compiling

We will use:

- the **emacs** text editor to write our C programs
- the **make** tool to compile our C programs
- the **gdb** debugger to debug our programs
- the **valgrind** tools to debug memory errors and measure program efficiency



This week



Next week

Working On C Programs

- **ssh** – remotely log in to **myth** computers
- **emacs** – text editor to write and edit C programs
 - Use the mouse to position cursor, scroll, and highlight text
 - Ctl-x Ctl-s to save, Ctl-x Ctl-c to quit
- **make** – compile program using provided Makefile
- **./myprogram** – run executable program (optionally with arguments)
- **make clean** – remove executables and other compiler files
- Lecture code is accessible at **/afs/ir/class/cs107/lecture-code/lect[N]**
 - Make your own copy: **cp -r /afs/ir/class/cs107/lecture-code/lect[N] lect[N]**
 - See the website for even more commands, and a complete reference.

Demo: Compiling And Running A C Program



Get up and running with our guide:

<http://cs107.stanford.edu/getting-started.html>



Question Break

Get up and running with our guide:
<http://cs107.stanford.edu/getting-started.html>

Assign0

Assignment 0 (Intro to Unix and C) goes out on Wednesday and falls due a week later, on **Wednesday, January 15th at 11:59PM PDT**.

There are **5** parts to the assignment, which is meant to get you comfortable using the command line, and editing/compiling/running C programs:

- Visit the website resources to become familiar with different Unix commands
- **Clone** the **assign0** starter project
- **Answer** several questions in **readme.txt**
- **Compile** a provided C program and **modify** it
- **Submit** the assignment

Recap

- CS107 is a programming class in C that teaches you about what goes on under the hood of programming languages and software.
- We'll use Unix and command line tools to write, debug and run our programs.
- Please visit the course website, cs107.stanford.edu, where you can read the lecture schedule, the syllabus, the information we provide about the Honor Code in CS107, and much, much more.

I'm looking forward to an awesome quarter!