Lecture 01
Introduction & C

CS211 – Fundamentals of Computer Programming II

Branden Ghena – Spring 2023

Slides adapted from:
Jesse Tov, Sruti Bhagavatula, Joe Hummel

Welcome to CS211

- Course Goal: become a **better** and **broader** programmer

- First half
  - C programming
  - Unix shell

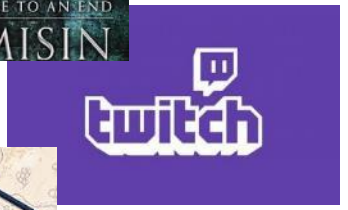
- Second half
  - C++ programming

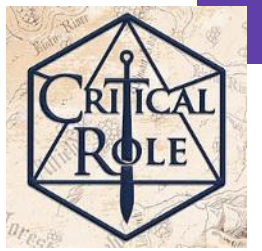
- Introduces students to industry-standard languages and tools
- Builds foundational software design skills at a medium scale

Branden Ghena (he/him)

- Assistant Faculty of Instruction
- Education
  - Undergrad: Michigan Tech
  - Master's: University of Michigan
  - PhD: University of California, Berkeley
- Research
  - Resource-constrained sensing systems
  - Low-energy wireless networks
  - Embedded operating systems
- Teaching
  - Computer Systems
    - CS211: Fundamentals of Programming II
    - CS213: Intro to Computer Systems
    - CS343: Operating Systems
    - CE346: Microprocessor System Design
    - CS397: Wireless Protocols for the IoT

Things I love





Questions in class

- Please ask questions!!!
  - It’s not just you who doesn’t understand something.
- You can always ask questions verbally during class
  - Raise hand whenever
  - I’ll stop for questions too
- Other options
  - Ask me after class
  - Ask on Piazza

Today's Goals

- Discuss **why** we teach (and require) this class
- Describe how this class is going to function
- Introduction to C programming

Outline

- **Why?**
- Course Overview
- Intro to C
  - Hello World
  - Variables
  - Computing Fibonacci

What does CS211 teach?

- **C and C++ Programming**

- Unix Shell

C - the most important programming language

- Old (1972), but nowhere near the first programming language
  - FORTRAN, LISP, ALGOL, COBOL, Basic, B, and many others came first
- Right time, right place, right capability
  - Enables both low-level control and (relatively) high level thinking
  - Fast, efficient, and highly portable
- Inspired everything that has come since
  - C syntax is copied partially or completely in MANY other languages
  - Lessons learned from using C inspired improvements to make programming easier

C++ - an evolutionary addition to C

- Additional features on top of C
  - Most important: classes to support Object Oriented Programming
  - Also includes a significant amount of libraries that C does not

- Enables more complicated software design
  - Manages which part of code can access which things at which times
  - Manages how things are named and referred to
  - Manages errors to help software respond to them

Things written in C/C++

- All major modern operating systems are partially or entirely C
  - Windows, Linux, MacOS, Android, iOS
- Scientific computing (mix of C and C++)
  - Mathematica, MATLAB, various scientific libraries
- Video game engines (often C++)
  - Unreal Engine, Unity, CryEngine
- Embedded control systems (usually C, occasionally C++)
  - Cars, Airplanes, Satellites and Rovers, Thermostats, Webcams, …

Upsides to C and C++

- You are in charge of everything
  - You can do anything you want without constraints

- Capable of directly interacting with hardware (“systems language”)
  - Grab exactly as much memory as you need and manage it yourself
  - Makes it incredibly fast (~100x faster than Python)
  - Makes it incredibly efficient (no memory is wasted)

- These lead to the languages being very widely used
  - Top five programming languages for decades include C and C++

Downsides to C and C++

- You are in charge of everything
  - And nothing is taken care of for you

- Things you "can't" do are often **UNDEFINED BEHAVIOR**
  - To enable portability, the languages just straight-up don't say what happens if you violate the rules
  - The computer could do *anything*

- Backwards compatibility means features are only ever added
  - You'll see this especially in C++, C just has less features total
  - C++ feels like a bunch of things stapled together
    - And there's an amazing programming language hiding in there

Analogies for programming languages

- Racket
  - Generic beginner's car that gets you places
- Python
  - Great car you can drive without a license
  - Unless you want to go really fast or on bad terrain, might be good enough
- C
  - A racing car that goes incredibly fast but breaks down every fifty miles
- C++
  - A souped-up version of the C racing car with dozens of extra features that only breaks down every 250 miles
  - But when it breaks down, nobody can figure out what went wrong

http://users.cms.caltech.edu/~mvanier/hacking/rants/cars.html

So why teach C and C++?

- You'll learn a lot more about programming
  - Syntax and ideas from C inspired a lot of other languages
  - Feels very different from Racket or Python

- You'll become a better programmer
  - You're going to run into a lot of errors and problems in this class
  - Hopefully they teach you to better design and plan your code

- Prepare you to dig deeper into computer systems
  - A "systems language" is needed to interact directly with hardware
  - Major options: Pascal, C, C++, Ada, Rust

What does CS211 teach?

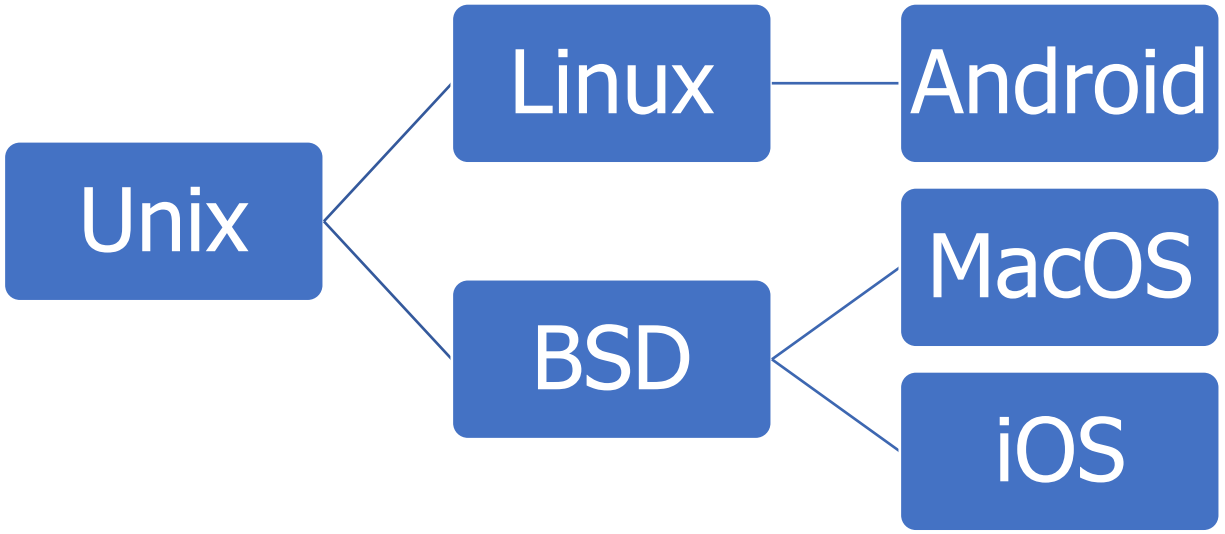
- C and C++ Programming

- **Unix Shell**

Unix

- A wildly popular operating system in the 1970s and 80s

- Today refers to the *family* of operating systems inspired or grown from Unix
  - Particular design style for "everything is a file"
  - Various tools the OS is expected to provide
  - Command line interface, also known as a "shell"



C and Unix were born together

- Operating systems used to be written in assembly
  - Basic instructions specific to a certain processor family (see CS213)
  - So supporting a new computer type meant rewriting all of your software

- Unix development (1969) by Ken Thompson and Dennis Ritchie
  - Developed at Bell Labs, which was a computing research powerhouse

- C language (1972) by Dennis Ritchie to write Unix programs
  - And they quickly rewrote the whole OS in C as well
  - This made the OS simpler to modify and easier to **port** to new systems
  - Unix became *enormously* popular due in part to its portability

Unix shell

- Text-based interface to a computer
  - Compare to graphical interfaces that need a mouse

- Necessary for remote interactions with many computers
  - Cloud servers
  - Specialized “headless” hardware

- Can be incredibly efficient and powerful
  - Find all JPEG files in this folder and convert into PNGs
    `mogrify -format png *.jpg`

  - Replace all instances of CS150 with CS211 across all Markdown files
    `sed -i 's/CS150/CS211/g' *.md`

So why teach Unix shell?

- Many future classes are going to require you to work on a specialized computer that is shared by the class
  - More resources, specific capabilities, etc.

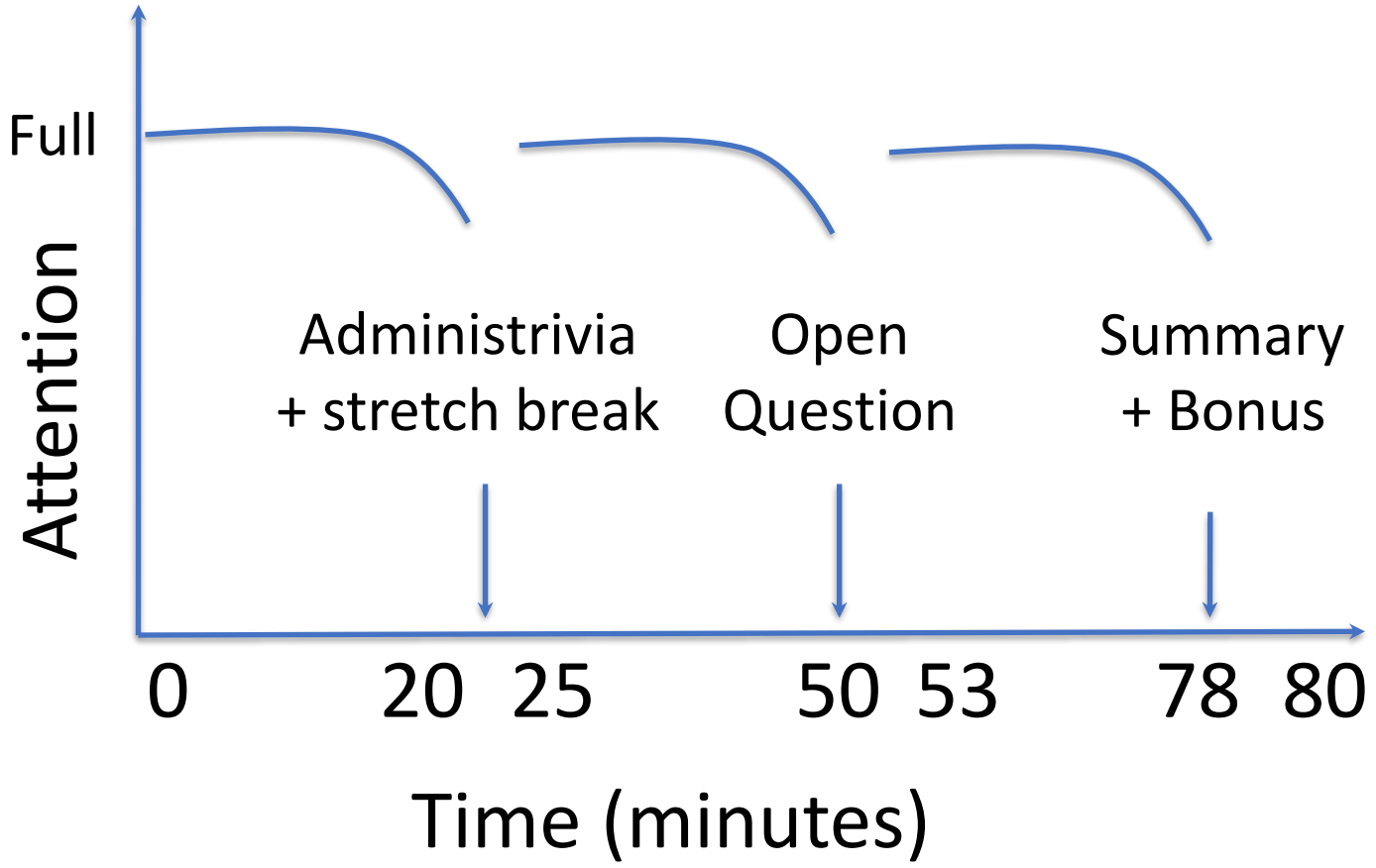
- Add another basic computing tool to your skillset
  - You might not use shell every day
  - But maybe you might

- You get to feel like a "hacker"
  - Using shell isn't the only way to be a programmer, but is a stereotypical way

So, why CS211?

- It’s going to make you a **much** better programmer
- It’s going to teach you a bunch of new skills
- It’s going to enable you to succeed in future classes

Architecture of a lecture



Break + Question

- Why might some software use C instead of Python?

Break + Question

- Why might some software use C instead of Python?

  - Performance!!
    - C is MUCH faster than Python

  - Tricky low-level “systems” behavior
    - Directly manipulating memory and hardware devices
    - E.g., an Operating System or a Game Engine

  - Legacy code
    - C is older than Python is, and is supported on more systems

    - An old Palm Pilot from the 90s: Python won’t work on it but C **will**

Outline

- Why?
- **Course Overview**
- Intro to C
  - Hello World
  - Variables
  - Computing Fibonacci

Course Staff

- TA (1)
  - Sherwin Shen - PhD student in Computer Science

- PMs (14)
  - Sofia Melendez
  - Natalie Hill
  - Chisara Ojiako
  - Mercy Omwoyo
  - Eli Barlow
  - Antonio Rocha
  - Inessa Verbitsky
  - Ethan McAlpin
  - John Sanchez
  - Matt Saperstein
  - Jackie Lin
  - Ben Geduld
  - Liz Yumbla
  - Emily Wei

- Their role: support student questions via office hours and Piazza

How to learn stuff

- Lectures: here in class on Tuesdays and Thursdays
  - Please attend and ask questions!
  - Panopto tab on Canvas will have recordings (a few hours later)

- Textbook
  - Zybooks “Programming in C” and “Programming in C++”
  - Interactive materials covering the basics of C and C++

- Office hours (starting next week)
  - Planning a mix of in-person and online
  - More info will be posted to Piazza when the schedule is ready

Asking questions

- Class and office hours are always an option!
  - I've got time to hang out after lectures and answer questions

- Piazza: (similar to Campuswire)
  - Post questions
  - Answer each other's questions
  - Find posts from the course staff
  - Post private info just to course staff

- Please do not email me! Post to Piazza instead!
  - I'm terrible at email and won't respond when I get busy
  - Exception: email me if you can't access Piazza.
    I'll be updating roster again a few times

Exercises

- Practice labs in the zyBooks textbook
  - Small snippets of code you'll need to write to match some expected output
  - Usually, 1-20 lines of code

- Immediate feedback, infinite retries, graded on completion
  - Can work with others on them if that's helpful

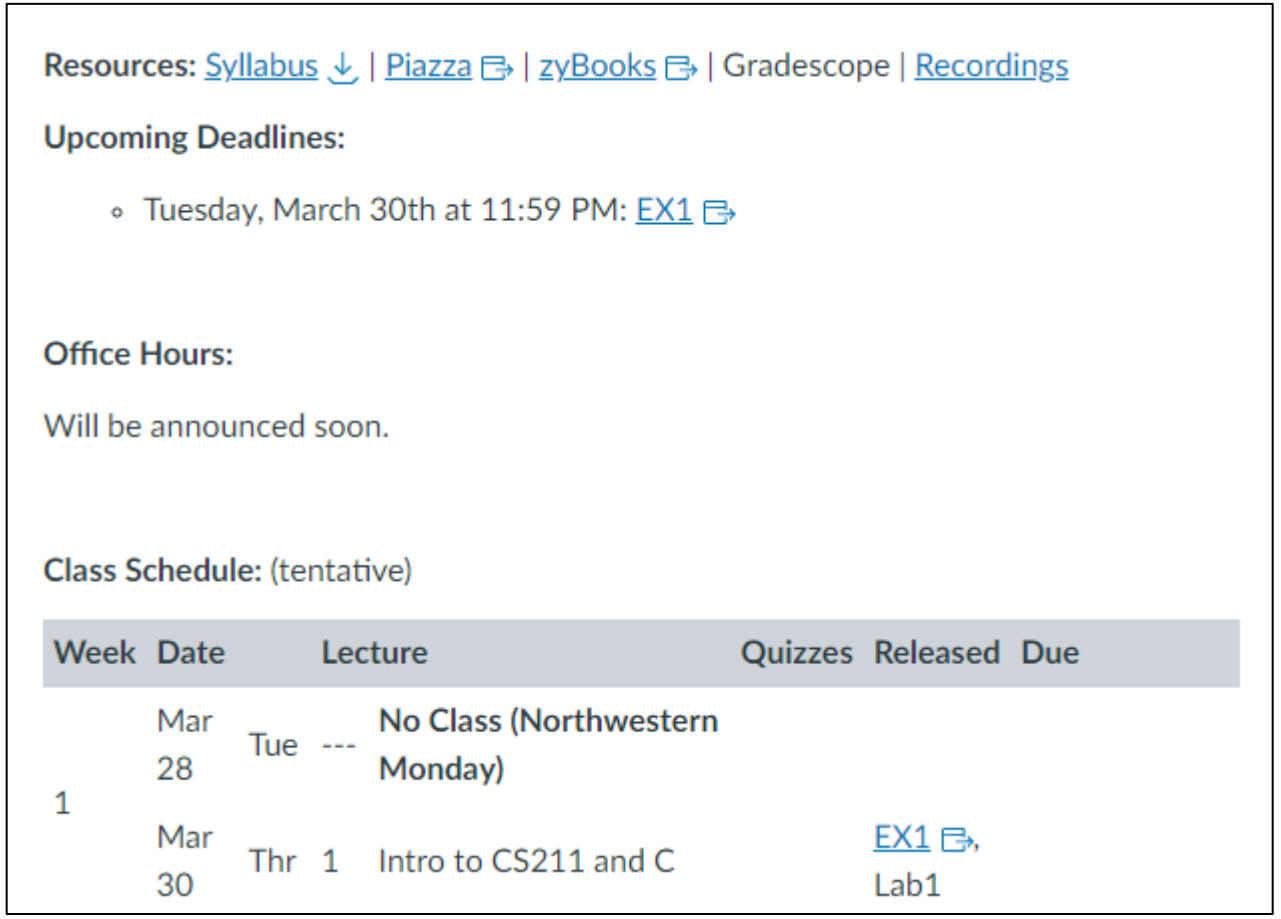
- Provides practice programming in C/C++
  - If you're already comfortable, should be easy
  - If you're uncomfortable, these should help!

Bad news: first assignment is already out

- The first set of exercises ("EX1") is due next week Tuesday

- Posted on Canvas homepage

- Lecture slides are also posted to Canvas right before class



Resources: Syllabus | Piazza | zyBooks | Gradescope | Recordings

Upcoming Deadlines:

- Tuesday, March 30th at 11:59 PM: EX1

Office Hours:

Will be announced soon.

Class Schedule: (tentative)

Week	Date		Lecture	Quizzes	Released	Due
1	Mar 28	Tue ---	**No Class (Northwestern Monday)**			
	Mar 30	Thr 1	Intro to CS211 and C		EX1, Lab1	

Labs

- Small, guided practice sessions to set up a new environment
  1. Setup for SSH access to lab machines (C programming)
  2. CLion IDE setup with game engine (C++)

- These are super important, because without them you won't be able to work on your homework!
  - First lab will be out tonight or tomorrow

- These are not formal assignments or quizzes
  - You may work with others on them
  - Goal is to make sure your setup works *before* the homework starts

Homeworks

- Medium-sized individual programming assignments
  - Around 200-1000 lines of code (50-200 is your solution)
  - About a week to complete them
- First three are C, last two are C++
- These are serious work, but also where the most learning will happen
  - Individual, may NOT work with other students on them

Final Project

- A bigger homework, where you get to choose what you want to do
  - Done with a partner of your choosing

- Make an "interactive program" (usually a game)
  - Examples: Pacman, Tetris, Two-dots, MS Paint, Checkers, Desert Bus

- This is your chance to do something interesting and fun!
- Can be a significant amount of work though

Quizzes

- Multiple quizzes instead of a big exam
  - Should be four total
  - Each is roughly 15-20 minutes
- Quizzes cover mainly material from the last two weeks
  - But build upon knowledge from the entire course
- Only 10% of your grade total (2.5% each)
  - Focus is really on making sure you're caught up on class material
  - Hopefully shouldn't be too stressful
- First quiz isn't until Tuesday of Week 3

Grade composition

Category	Count	Total Value
Exercises	6	5%
Labs	2	5%
Homework	5	55%
Final project	1	25%
Quizzes	4	10%

- Standard letter grade scale
  - 93%+ A
  - 90%+ A-
  - 87%+ B+
  - etc.

Relative homework difficulties

Homework	Difficulty
HW1	5
HW2	7
HW3	11
HW4	6
HW5	9
Final Project	10ish*

HW3 is the last in C

It's a two-part assignment spread over two weeks

* But really it's up to you

Late Policy

- You can submit *homeworks* late
  - Quizzes, exercises, and labs cannot be submitted late

- 10% penalty to maximum grade per day late
  - Example: three days late means maximum grade is 70%

- Final project has a sliding scale
  - 90% for up to 24-hours late
  - 60% and 30% for the two days after that

We will support you if possible and equitable

- We can be flexible with deadlines for problems outside of your control
  - Sick, family emergency, broken computer
  - Contact me (via Piazza) and I'll provide additional extensions

- Also, we support expected accessibility needs
  - Make sure to submit ANU requests if you have any
  - Let me know about anything else you need and we can discuss it

Slip Days

- Slip days let you turn in a homework late and receive no penalty

- Each student gets **4 slip days**
  - Apply to **homeworks only** (not final project, exercises, or labs)
  - You don’t need to tell us you’re using them, we’ll just automatically apply them at the end of the year

- Examples:
  - Turn in HW1 three days late
  - Turn in HW4 two days late and HW5 one day late
  - Turn in HW2 four days late with only a one-day penalty

Getting Help – Office Hours

- Office hours are mostly hosted by the PMs and TA
  - I will have some too! Especially for higher-level questions

- Schedule
  - We’re going to host a TON of office hours
  - With some in-person and some online
  - Details to follow, schedule on Canvas homepage

- Reminder: office hours are meant to augment the class
  - Attend them when you need to!

Getting Help – Request a Meeting

- Lecture is my side gig

- My main job is helping students succeed

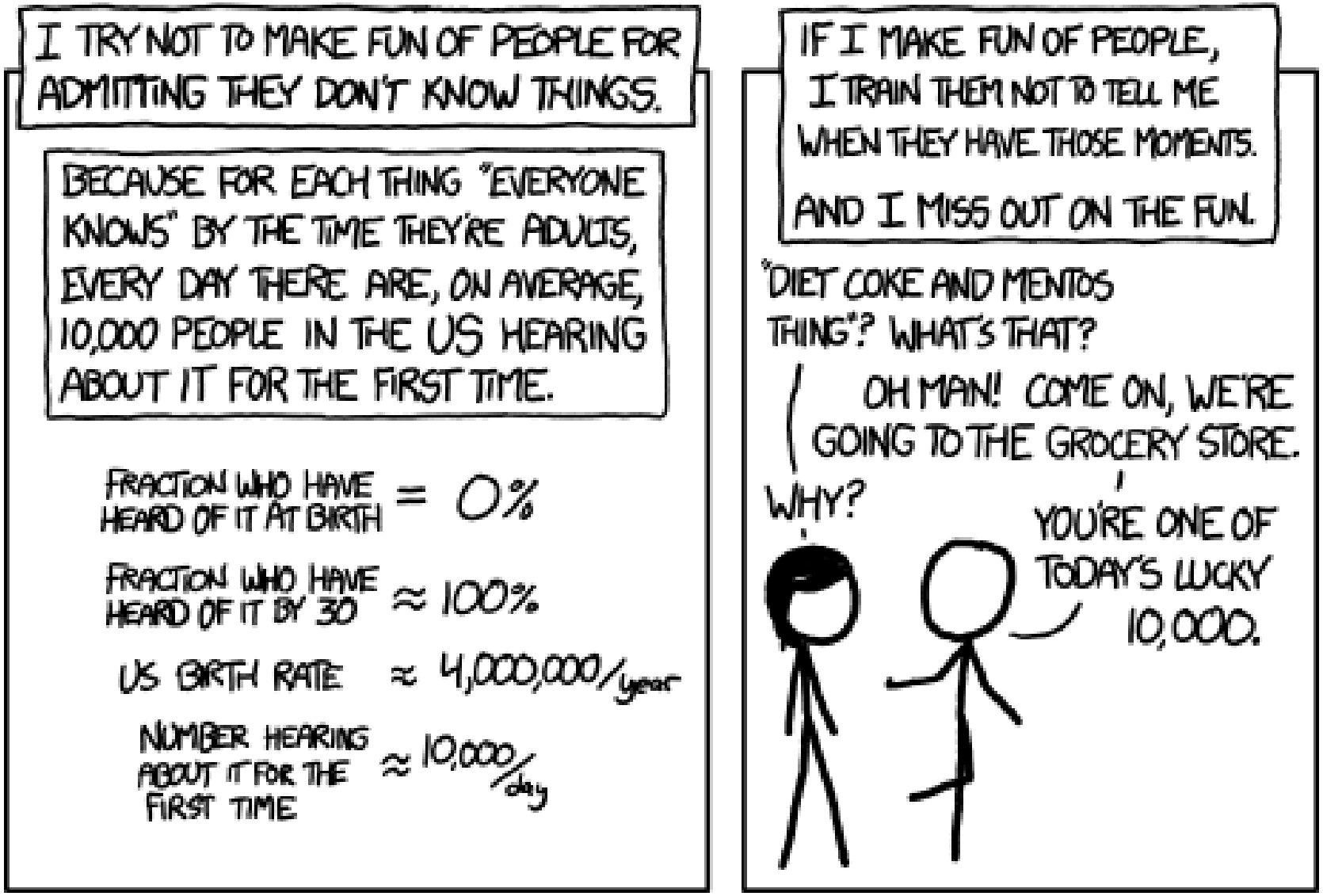
- If you are struggling, reach out and I will meet with you
  - Course material
  - Homework
  - Other stuff going on in your life

Advice

- Submit assignments early and often!

- If you find this course difficult, that's because it **is** difficult.
- However, nobody fails unless they give up.
- You belong here and can succeed here.
- **Be kind to each other.**

Break + relevant xkcd



https://xkcd.com/1053/

Collaboration in CS211, three levels:

1. Partner Collaboration

- Your code and the other student's code are identical because you share it and work on it together
- ONLY for registered partners on final project

2. Close Collaboration

- You communicate about code however you see fit
- ONLY acceptable for labs and exercises

3. Arms-Length Collaboration

- You discuss problems and solutions at a high level
- MAY NOT read, write, look at, record, or transcribe code
- MAY NOT have the code up on screen during collaboration
- MUST cite your sources, both arms-length collaborators and other resources

Refer to syllabus for the official version of this policy

Academic Honesty

- In CS211, we take cheating very seriously

- Cheating is when you:
  - Engage in an inappropriate level of collaboration
    - Such as look at another student's code
  - Enable another student, *present or future*, to cheat
    - Such as letting a CS211 student read your code next year
  - Fail to cite your sources (friend, Stack Overflow, etc.)
    - Such as you get a big hint and don't acknowledge where it came from in a code comment

Academic Honesty

- **Please do not cheat in CS211**

1. If you don’t write code, you won’t learn!

2. Cheating on code is super easy to catch!!
   - No, like really really easy
   - All suspected cheating is reported to the relevant dean for investigation
   - Last time I taught CS211, eight different students were reported

- If you are unsure about a situation, ask the staff on Piazza

Outline

- Why?
- Course Overview
- **Intro to C**
  - **Hello World**
  - Variables
  - Computing Fibonacci

Hello world C program

hello.c

```c
#include <stdio.h>

int main(void) {
    printf("Hello, CS 211!\n");

    return 0;
}
```

Hello world C program

hello.c

This is the code file where you can find this code!

Usually, I'll provide students source code for any in-class examples

```c
#include <stdio.h>

int main(void) {
    printf("Hello, CS 211!\n");

    return 0;
}
```

Hello world C program

hello.c

```c
#include <stdio.h>

int main(void) {
    printf("Hello, CS 211!\n");

    return 0;
}
```

Hello world C program

hello.c

```
#include <stdio.h>

int main(void) {
    printf("Hello, CS 211!\n");

    return 0;
}
```

A function named `main()`

No Arguments (`void`)

Returns an integer

Hello world C program

hello.c

```c
#include <stdio.h>

int main(void) {
    printf("Hello, CS 211!\n");

    return 0;
}
```

Call to the `printf()` function

One argument to the function, the string “Hello, CS211\n”

Hello world C program

hello.c

The `printf()` function is a part of the standard input/output library, included here

Call to the `printf()` function

One argument to the function, the string “Hello, CS211\n”

```
#include <stdio.h>

int main(void) {
    printf("Hello, CS 211!\n");

    return 0;
}
```

Hello world C program

hello.c

```c
#include <stdio.h>

int main(void) {
    printf("Hello, CS 211!\n");

    return 0;
}
```

Returns a value, 0
(which is of type `int`)

Hello world C program

hello.c

```c
#include <stdio.h>

int main(void) {
    printf("Hello, CS 211!\n");

    return 0;
}
```

Two special things going on here:

1. main() is a special function name that is called when the program runs

Hello world C program

hello.c

```c
#include <stdio.h>

int main(void) {
    printf("Hello, CS 211!\n");

    return 0;
}
```

Two special things going on here:

1. main() is a special function name that is called when the program runs

2. main() returns a number that specifies whether the program succeeded or failed and how
   - 0 means success
   - non-zero means failure
   - specific numbers mean different things to different programs

Outline

- Why?
- Course Overview
- **Intro to C**
  - Hello World
  - **Variables**
  - Computing Fibonacci

Program state is preserved in variables

- C is an **Imperative** programming language
  - List of step-by-step statements that modify the program's **state**

- State is information from prior steps that influences future steps
  - Example: TV volume Up/Down apply to prior setting

- In programs, we explicitly keep state in variables

```
int z = 5;
```

Values, objects, and variables

- **Values** are the actual information we want to work with
  - Numbers, Strings, Images, etc.
  - Example: `5` is an `int` value while `'a'` is a `char` value

- An **object** is a chunk of memory that can hold a value of a particular type.
  - Example: function `f` takes an argument `int x`
    - Each time `f` is called, a "fresh" object that can hold an `int` is "created"

- A **variable** is the name of an object

- Assigning to a **variable** changes the *value* stored in the **object** named by the variable

Example of definition and assignment

```
int z = 5;
z = 7;
z = z + 4;
```

- What happens?

Example of definition and assignment

```
int z = 5;
z = 7;
z = z + 4;
```

- What happens?
  1. The first statement is a definition.
     It creates an `int` object,
     names it `z`,
     and initializes it to the value `5`

Example of definition and assignment

```
int z = 5;
z = 7;
z = z + 4;
```

- What happens?
  2. The second statement is an assignment.
     It replaces the value `5`
     stored in the object named by `z`
     with the value `7`.

Example of definition and assignment

```
int z = 5;
z = 7;
z = z + 4;
```

- What happens?

z: 11

3. The third statement is also an assignment.
   It retrieves the current value of `z` (which is `7`),
   then adds `4` to it,
   and then stores the result back in the object named by `z`.

C: Typed imperative programming

- Imperative programming
  - Each line is a **statement** that changes the program's **state**
  - Usually, the values within a variable

- Type System
  - Variables have a type associated with them

  1. The type determines qualities of the *object*
     - Example: how much memory it takes up

  2. The type specifies what kind of *value* the variable holds
     - Example: integers, decimal numbers, strings, etc.

Some types in C

- Hold an integer number (like 5 or 0 or -3)
  - `char`, `short`, `int`, `long`, `size_t`, `int8_t`, `int16_t`, `int32_t`, etc.
  - These can also specify signedness
    - `unsigned`: only 0 and greater
    - `signed`: negative, 0, or positive

- Hold a decimal number (like 6.238 or 0.00001 or -32566.5)
  - `float`, `double`
  - These are always negative, 0, or positive

- Difference between types: how big of a value they can hold
  - `short`: 0 to 65536 OR `signed short`: -32768 to 32767
  - `int`: 0 to 4294967296 OR `signed int`: -2147483648 to 2147483647

  - We'll have a whole future lecture on *why* the types are like this

Signed vs unsigned variables

- All “integer” types in C can be signed or unsigned
  - char, short, int, long, etc.

  - Unsigned: only zero or positive
  - Signed: negative, zero, or positive

- Signed is the default! If it doesn’t say, it’s usually signed
  - An exception is `size_t` which is unsigned

- Comparing signed and unsigned numbers generates a warning
  - Should make sure they’re the same before comparing

Temporarily changing types while comparing

- You can cast a variable to another type during an expression
  - To cast, put a type in parentheses before the variable name

- Example

```
int i = 0;                  //int is signed by default
size_t length = 5;          //size_t is unsigned

if (i > length) {           // warning here!
   printf("Too big!\n");
}
```

Temporarily changing types while comparing

- You can cast a variable to another type during an expression
  - To cast, put a type in parentheses before the variable name

- Example

```
int i = 0;                  //int is signed by default
size_t length = 5;          //size_t is unsigned

if (i > (int)length) { // no warning anymore!
   printf("Too big!\n");
}
```

More complicated example

```
int prev;
int curr = 5;
int next = 8;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;
```

More complicated example

```
int prev;
int curr = 5;
int next = 8;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;
```

prev: 

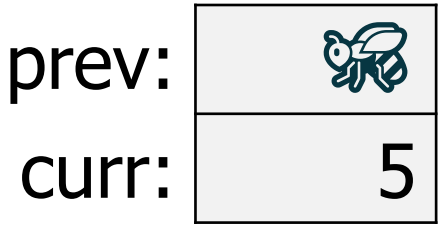
More complicated example

```
int prev;
int curr = 5;
int next = 8;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;
```



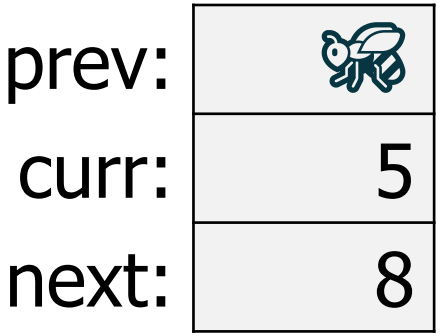
More complicated example

```
int prev;
int curr = 5;
int next = 8;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;
```



More complicated example

```
int prev;
int curr = 5;
int next = 8;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;
```

prev:	5
curr:	5
next:	8

More complicated example

```
int prev;
int curr = 5;
int next = 8;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;
```

prev:	5
curr:	8
next:	8

More complicated example

```
int prev;
int curr = 5;
int next = 8;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;
```

prev:	5
curr:	8
next:	13

More complicated example

```
int prev;
int curr = 5;
int next = 8;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;
```

prev:	8
curr:	8
next:	13

More complicated example

```
int prev;
int curr = 5;
int next = 8;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;
```

prev:	8
curr:	13
next:	13

More complicated example

```
int prev;
int curr = 5;
int next = 8;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;
```

prev:	8
curr:	13
next:	21

More complicated example

```
int prev;
int curr = 5;
int next = 8;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;
```

prev:	13
curr:	13
next:	21

More complicated example

```
int prev;
int curr = 5;
int next = 8;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;
```

prev:	13
curr:	21
next:	21

More complicated example

```
int prev;
int curr = 5;
int next = 8;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;

prev = curr;
curr = next;
next = prev + curr;
```

prev:	13
curr:	21
next:	34

Outline

- Why?
- Course Overview
- **Intro to C**
  - Hello World
  - Variables
  - **Computing Fibonacci**

Definition of Fibonacci Function

- $fib(n) = \begin{cases} n, & if\ n < 2; \\ fib(n-2) + fib(n-1), & otherwise \end{cases}$

n	fib(n)
0	0
1	1
2	1
3	2
4	3
5	5
6	8
7	13
8	21

Implementing Fibonacci in C

fib.c

```
long fib(int n){
   if (n < 2) {
      return n;
   } else {
      return fib(n - 2) + fib(n - 1);
   }
}
```

$$fib(n) = \begin{cases} n, & if\ n < 2; \\ fib(n-2) + fib(n-1), & otherwise \end{cases}$$

Recursion works in C!

Implementing Fibonacci in C

fib.c

```
long fib(int n){
  if (n < 2) {
    return n;
  } else {
    return fib(n - 2) + fib(n - 1);
  }
}
```

$$fib(n) = \begin{cases} n, & if\ n < 2; \\ fib(n-2) + fib(n-1), & otherwise \end{cases}$$

```
if (⟨test-expr⟩) { // evaluate ⟨test-expr⟩; then…
  ⟨then-stms⟩ // do these if ⟨test-expr⟩ was true
} else {
  ⟨else-stms⟩ // do these if ⟨test-expr⟩ was false
}
```

Any statements can be nested in C

```
if (⟨first-test-expr⟩) {
  if (⟨second-test-expr⟩) {
    ⟨A-stms⟩
  } else {
    ⟨B-stms⟩
  }
} else {
  if (⟨third-test-expr⟩) {
    ⟨C-stms⟩
  } else {
    ⟨D-stms⟩
  }
}
```

C ignores most whitespace

$$fib(n) = \begin{cases} n, & if\ n < 2; \\ fib(n-2) + fib(n-1), & otherwise \end{cases}$$

```
long fib(int n){
   if (n < 2) {
      return n;
   } else {
      return fib(n - 2) + fib(n - 1);
   }
}
```

C ignores most whitespace

$$fib(n) = \begin{cases} n, & if\ n < 2; \\ fib(n-2) + fib(n-1), & otherwise \end{cases}$$

```
long fib(int n){
  if (n < 2) {
    return n;
  } else {
    return fib(n - 2) +
           fib(n - 1);
  }
}
```

C doesn't care about whitespace

C ignores most whitespace

$$fib(n) = \begin{cases} n, & if\ n < 2; \\ fib(n-2) + fib(n-1), & otherwise \end{cases}$$

```
long fib(int n){if(n<2){return n;}else{return
fib(n-2)+fib(n-1);}}
```

C **really doesn't care** about whitespace

C ignores most whitespace

$$fib(n) = \begin{cases} n, & if\ n < 2; \\ fib(n-2) + fib(n-1), & otherwise \end{cases}$$

```
long fib(int n){if(n<2){return n;}else{return
fib(n-2)+fib(n-1);}}
```

C **really doesn't care** about whitespace

But humans do!

So don't write your code this way!!!!!!!!!!

A note on style

- A lot of things are *possible* in C, but bad ideas
  - They can make things hard to read
  - They can be a source of bugs in code

- We try to provide you with what we think of as "good" C code

- We have a guide to how you should write your C code
  - This is a (small) portion of your grade on each homework!
  - https://nu-cs211.github.io/cs211-files/cstyle.html

Outline

- Why?
- Course Overview
- Intro to C
  - Hello World
  - Variables
  - Computing Fibonacci