The Plan
Welcome to CS 240: Foundations of Computer Systems!
Your lecture instructor: **Alexa VanHattum**

Note: you can call me “Alexa”, “Prof. Alexa”, or “Prof. VanHattum”

- New to Wellesley this year!
- Research focus: programming languages & systems

Before Wellesley:
- PhD in Computer Science at Cornell
- Software engineer for Apple health (heart monitoring)
  - **THIS CLASS** one of the most helpful across industry *and* research
Today’s preview

1. What is CS 240?

2. Why take CS 240? (in brief)

3. How does CS 240 work? (in brief)
CS 111, 230, 231, 235, 251:

- How do you use programming to solve a problem?
- How do you structure a program?
- How do you know it is correct or efficient?
- How hard is it to solve a problem?
- How is computation expressed?
- What does a program mean?
- ...

A BIG question is missing...
Software

CS 111, 230, 231, 235, 251

CS 240

Algorithm, Data Structure, Application

Programming Language

Compiler/Interpreter

Operating System

Instruction Set Architecture

Microarchitecture

Digital Logic

Devices (transistors, etc.)

Solid-State Physics

Hardware
Big Idea: Abstraction

Layers manage complexity.
Big Idea: Abstraction

with a few recurring subplots

Simple, general interfaces:
Hide complexity of efficient implementation.
Make higher-level systems easy to build.

**Representation** of data and programs

**Translation** of data and programs

**Control flow** within/across programs

0s and 1s, electricity

compilers, assemblers, decoders

branches, procedures, operating system
Desired computation in a programming language

Physical implementation with circuits and electricity.

Hardware/Software Interface
CS 240 in 3 acts  (4-5 weeks each)

1. Hardware *implementation*
   From transistors to a simple computer

2. Hardware-software *interface*
   From instruction set architecture to programming in C

3. Abstraction for practical systems
   Memory hierarchy
   Operating system basics
   Higher-level languages and tools
Machine Instructions

(adds two values and stores the result)

00000010100010101100100000010000

Instruction Set Architecture specification

Hardware

machine code program
Assemblers and Assembly Languages

addl %eax, %ecx \rightarrow 00000010100010101100100000010000

Assembly Language specification

assembly program \rightarrow Assembler \rightarrow machine code program

Hardware
Higher-Level Programming Languages

\[
x = x + y;
\]

\[
\text{addl } \%eax, \%ecx \quad 0000001010001011001000010000
\]

Programming Language specification

<table>
<thead>
<tr>
<th>1940s</th>
<th>1950s</th>
<th>1960s</th>
<th>1970s</th>
<th>1980s</th>
<th>1990s</th>
<th>2000s</th>
<th>2010s</th>
<th>2020s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile time</td>
<td>Run time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
More and more layers...

- Operating systems
- Virtual machines
- Hypervisors
- Web browsers
- ...
Microarchitecture (Implementation of ISA)

Computer

Instruction Fetch and Decode

Registers

ALU

Memory
I just like to program.

Why study the implementation?
I just like to program. Why study the implementation?

Most system abstractions "leak."

Implementation details affect your programs:

- Their performance
- Their correctness
- Their security
Performance

\[ \frac{x}{973} \quad \frac{x}{1024} \]
Which of these code snippets is faster? `x / 973` or `x / 1024`?

- `x / 973` is faster
- `x / 1024` is faster
- The same
Which of these code snippets is faster? `x / 973` or `x / 1024`
Which of these code snippets is faster? `x / 973` or `x / 1024`?
void copyji(int src[2048][2048],
     int dst[2048][2048])
{
    int i, j;
    for (j = 0; j < 2048; j++)
        for (i = 0; i < 2048; i++)
            dst[i][j] = src[i][j];
}

void copyij(int src[2048][2048],
     int dst[2048][2048])
{
    int i, j;
    for (i = 0; i < 2048; i++)
        for (j = 0; j < 2048; j++)
            dst[i][j] = src[i][j];
}

several times faster
due to hardware caches
Correctness ✗ ✔

int ≠ integer
float ≠ real

Exploded due to cast of 64-bit floating-point number to 16-bit signed number. Overflow.

"... a Model 787 airplane ... can lose all alternating current (AC) electrical power ... caused by a software counter internal to the GCUs that will overflow after 248 days of continuous power. We are issuing this AD to prevent loss of all AC electrical power, which could result in loss of control of the airplane."
--FAA, April 2015
The **GHOST vulnerability** is a buffer overflow condition that can be easily exploited locally and remotely, which makes it extremely dangerous. This vulnerability is named after the `GetHOSTbyname` function involved in the exploit.

All computers are flawed -- and the fix will take years

by Selena Larson  @selenalarson

January 26, 2018, 12:07 PM ET

Meltdown and Spectre

A Heart Device Is Found Vulnerable to Hacker Attacks

by Barnaby J. Feder
Published March 12, 2018

To the long list of objects vulnerable to attack by computer hackers, add the human heart.

The threat seems largely theoretical. But a team of computer security researchers plans to report Wednesday that it had been able to gain wireless access to a combination heart defibrillator and pacemaker.
Why take CS 240?

Learn *how* computers execute programs.
Deepen your appreciation of *abstraction*.
Improve your *critical thinking* skills.

Become a **better programmer**:
Think rigorously about execution models.
Identify limits and impacts of abstractions and representations.
Learn to use software development tools.

**Foundations** for:
Compilers, security, computer architecture, operating systems, ... 

Have fun and feel accomplished!
https://cs.wellesley.edu/~cs240/

3 Long but necessary!