Welcome to

CS 240:

Intro to

Computer Systems

- Program, Application
- Programming Language
- Compiler/Interpreter
- Operating System
- Instruction Set Architecture
- Microarchitecture
- Digital Logic
- Devices (transistors, etc.)
- Solid-State Physics

Today

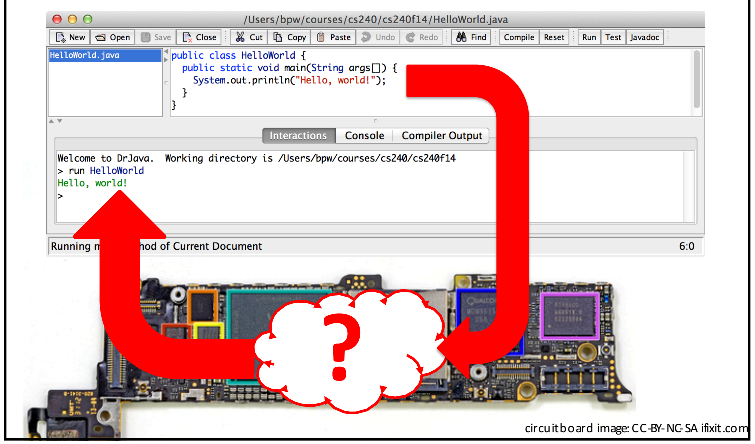
1. **What is CS 240?**
2. Why take CS 240?
3. How does CS 240 work?
4. Dive into foundations of computer hardware.

CS 111, 230, 231, 235, 251:

- What can a program do?
- How can a program 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...

1 CS 240: How do computers work?



Software

Hardware

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

Big Idea:
Abstraction

interface

implementation

Layers of virtual machines manage complexity.

Algorithm, Data Structure, Application

Programming Language

Compiler/Interpreter

Operating System

Instruction Set Architecture

Microarchitecture

Digital Logic

Devices (transistors, etc.)

Solid-State Physics

Big Idea: Abstraction

with a few recurring subplots

Simple, general interfaces:

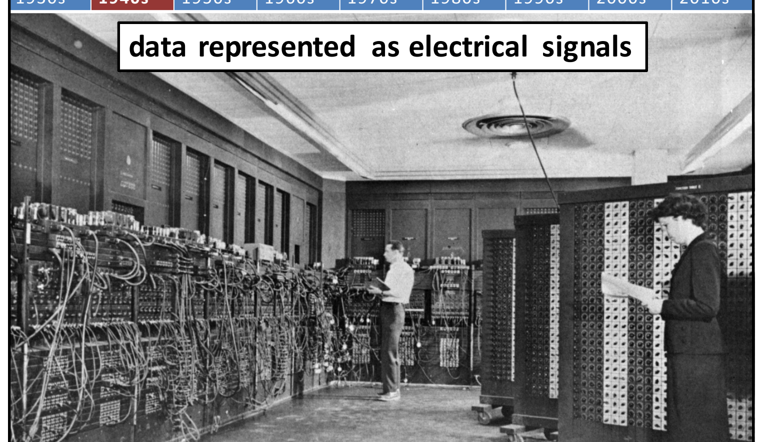
- Hide complexity of efficient implementation.
- Make higher-level systems easy to build.
- **But they are not perfect.**

Representation of data and programs — 0s and 1s, electricity

Translation of data and programs — compilers, assemblers, decoders

Control flow within/across programs — branches, procedures, OS

1930s | **1940s** | 1950s | 1960s | 1970s | 1980s | 1990s | 2000s | 2010s

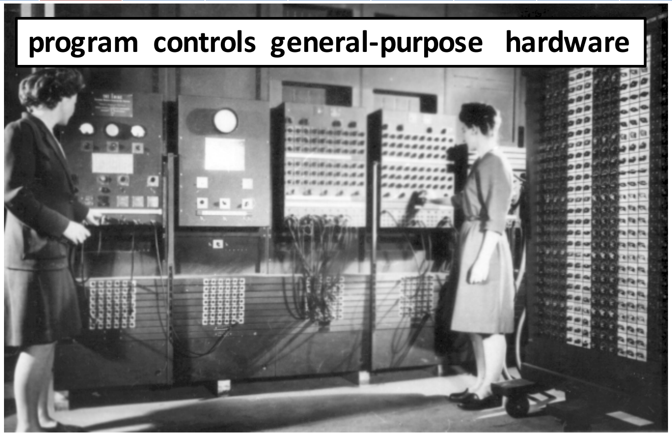


ENIAC (Electronic Numerical Integrator and Computer),
First Turing-complete all-electronic programmable digital computer.
University of Pennsylvania, 1940s

Image: public domain

1930s | **1940s** | 1950s | 1960s | 1970s | 1980s | 1990s | 2000s | 2010s

program controls general-purpose hardware

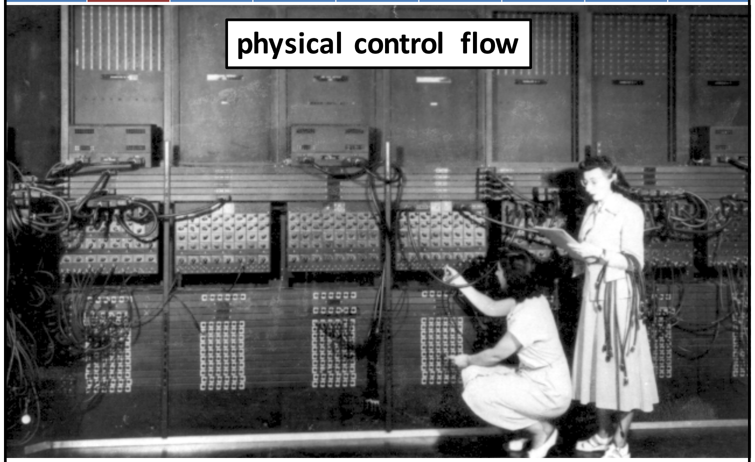


Jean Jennings Bartik and Frances Bilas Spence with part of ENIAC.
The programmers of ENIAC were six women.
http://eniacprogrammers.org/, http://sites.temple.edu/topsecretrosies/

Image: public domain

1930s | **1940s** | 1950s | 1960s | 1970s | 1980s | 1990s | 2000s | 2010s

physical control flow

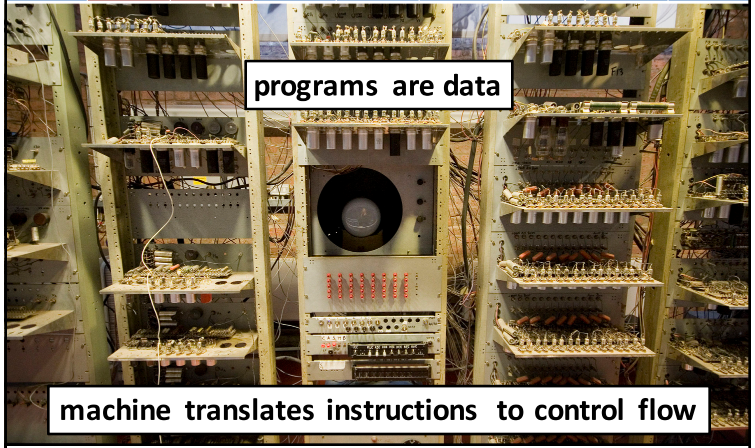


Programming 1940s-style *with switches and cables.*

Image: public domain

1930s | **1940s** | **1950s** | 1960s | 1970s | 1980s | 1990s | 2000s | 2010s

programs are data



machine translates instructions to control flow

Manchester "Baby" SSEM (Small-Scale Experimental Machine), replica
first **stored-program computer** -- University of Manchester (UK), 1948

Image: "SSEM Manchester museum close up" by Parrot of Doom - Own work. Licensed under Creative Commons Attribution-Share Alike 3.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:SSEM_Manchester_museum_close_up.jpg

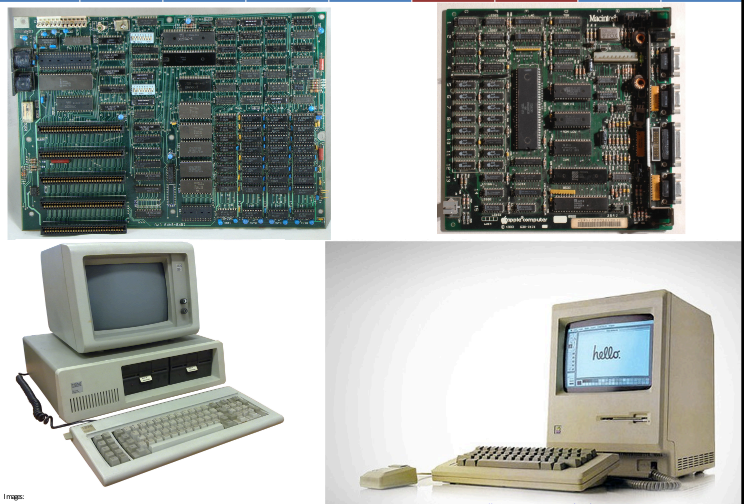
1930s | 1940s | 1950s | **1960s** | **1970s** | 1980s | 1990s | 2000s | 2010s

PDP-11 "**minicomputers**"

http://simh.trailing-edge.com/

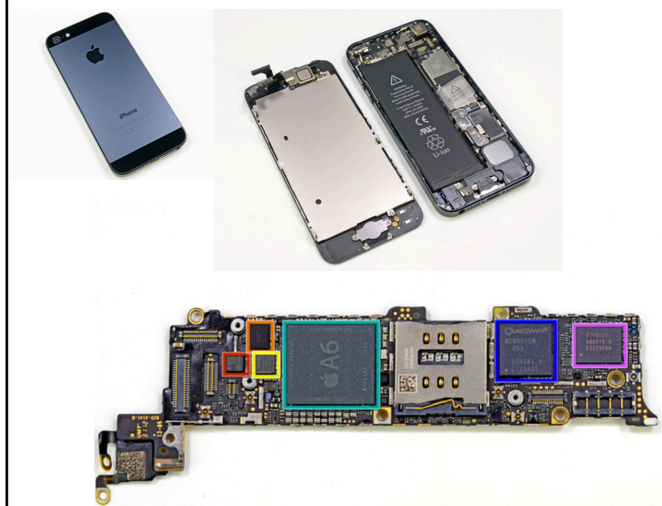
http://www.pcworld.com/article/249951/if_it_aint_broke_dont_fix_it_ancient_computers_in_use_today.html?page=2

1930s | 1940s | 1950s | 1960s | 1970s | **1980s** | **1990s** | 2000s | 2010s



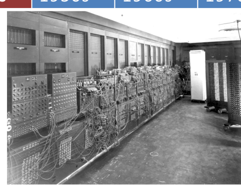
Images:
"Ibm pc 5150" by Ruben de Rijcke - Own work. Licensed under Creative Commons Attribution-Share Alike 3.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Ibm_pc_5150.jpg
"IBM PC Motherboard (1981)" by German - Own work. Licensed under Creative Commons Attribution-Share Alike 3.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:IBM_PC_Motherboard_(1981).
"Macintosh-motherboard" by Shieldforyoureyes Dave Fischer - Own work. Licensed under Creative Commons Attribution-Share Alike 3.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Macintosh-m

1930s | 1940s | 1950s | 1960s | 1970s | 1980s | 1990s | **2000s** | **2010s**



images: CC-BY-NC-SA ifixit.com

1930s | **1940s** | 1950s | 1960s | 1970s | 1980s | 1990s | 2000s | **2010s**

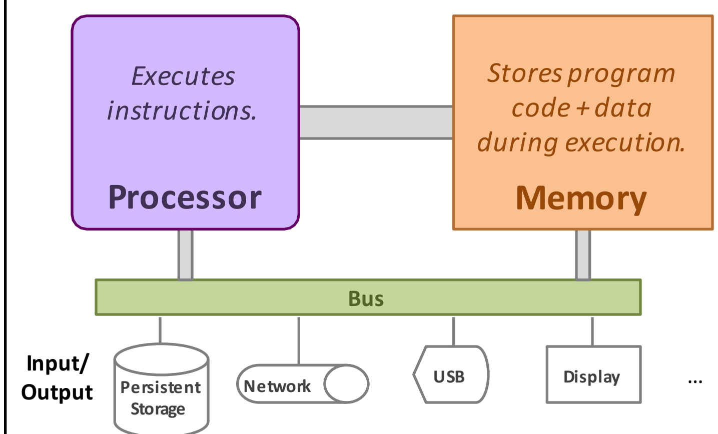


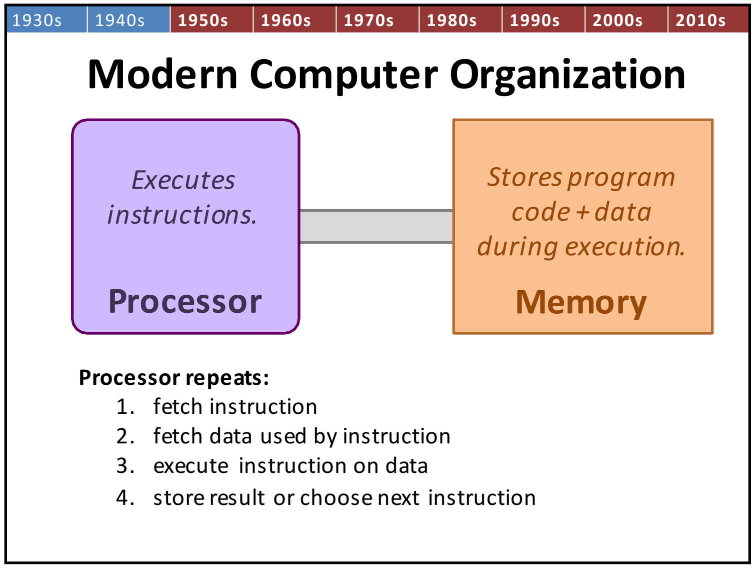
ENIAC image: public domain; iPhone image: CC-BY-NC-SA ifixit.com

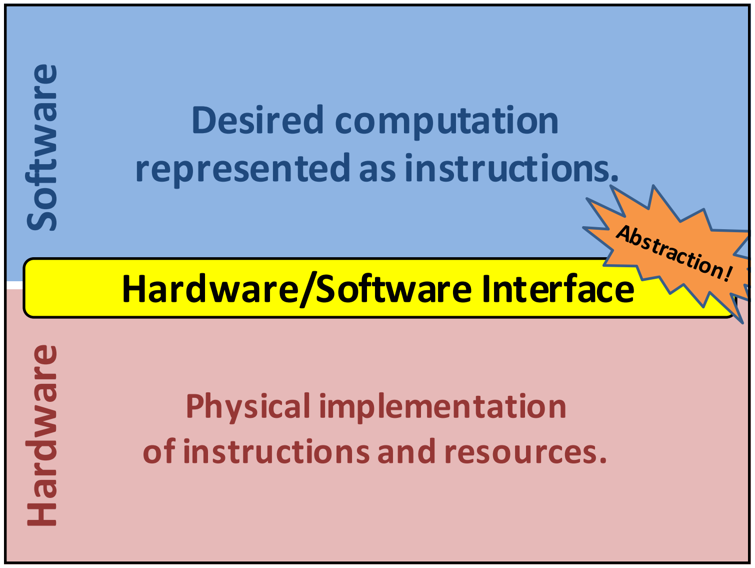
	ENIAC	iPhone 5
Year	1946	2012
Weight	30 tons	4 oz
Volume	2,400 ft^3	3.4 in^3
Cost (USD, 2014)	$6,000,000	$600
Speed	few 1000 ops/sec	2,500,000,000 ops/sec
Memory	~100 bytes	1,073,741,824 bytes (1 GB)
Power	150,000 W	<5W
Input/Output	Switches, lights, later punchcards	Touchscreen, audio, camera, wifi, cell, ...
Production	1	5,000,000 sold in first 3 days

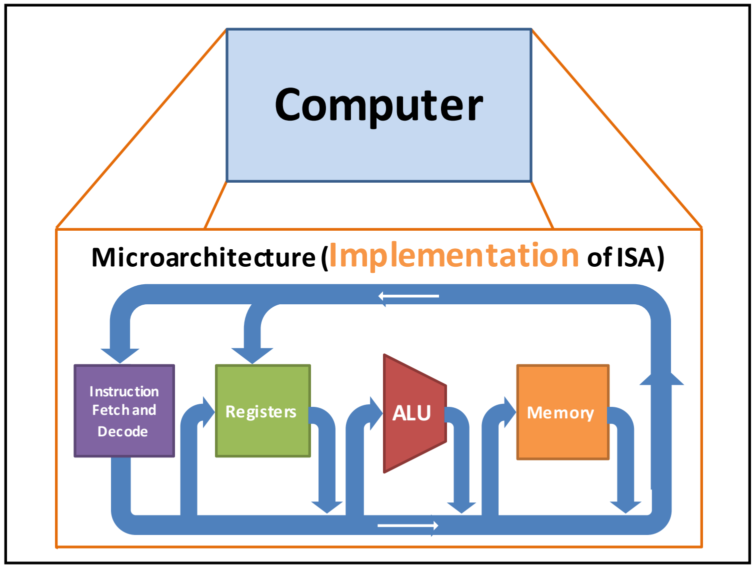
1930s | 1940s | **1950s** | **1960s** | **1970s** | **1980s** | **1990s** | **2000s** | **2010s**

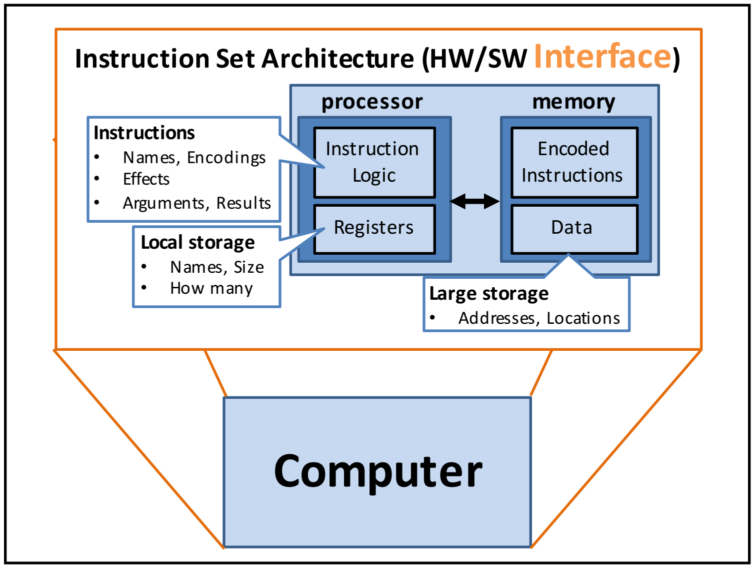
Modern Computer Organization




1930s 1940s 1950s 1960s 1970s 1980s 1990s 2000s 2010s
Modern Computer Organization
Executes instructions.
Processor
Stores program code + data during execution.
Memory
Processor repeats:
1. fetch instruction
2. fetch data used by instruction
3. execute instruction on data
4. store result or choose next instruction


Software
Desired computation represented as instructions.
Abstraction!
Hardware/Software Interface
Hardware
Physical implementation of instructions and resources.


Computer
Microarchitecture (Implementation of ISA)
Instruction Fetch and Decode
Registers
ALU
Memory

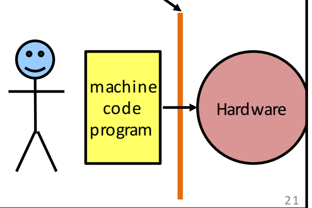

Instruction Set Architecture (HW/SW Interface)
processor
memory
Instructions
• Names, Encodings
• Effects
• Arguments, Results
Instruction Logic
Registers
Encoded Instructions
Data
Local storage
• Names, Size
• How many
Large storage
• Addresses, Locations
Computer

Machine Instructions

(adds two values and stores the result)

00000010100010101100100000010000 0

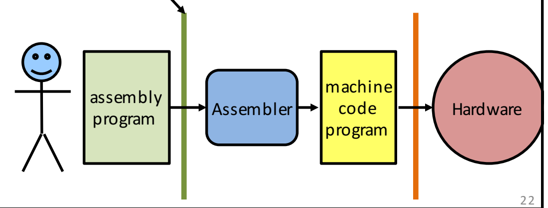
Instruction Set Architecture specification



Assemblers and Assembly Languages

`addl %eax, %ecx` → 00000010100010101100100000010000

Assembly Language specification

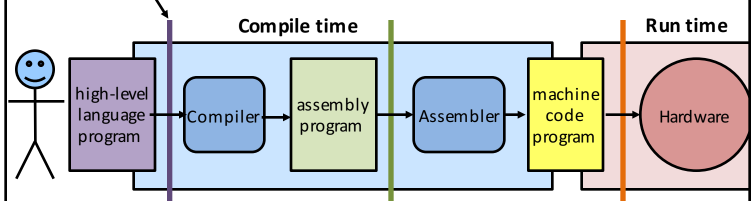


Higher-Level Programming Languages

```
x = x + y;
```

`addl %eax, %ecx` → 00000010100010101100100000010000

Programming Language specification



A-0: first compiler, by Grace Hopper

Early 1950s
Maybe closer to assembler/linker/loader

Later:
- B-0 → FLOW-MATIC
  → COBOL, late 50s

Jean Sammet also involved

1930s	1940s	1950s	1960s	1970s	1980s	1990s	2000s	2010s

More and more layers…

- Operating systems
- Virtual machines
- Hypervisors
- Web browsers
- …

25

CS 240: a 3-stage sprint
(4-5 weeks each)

Hardware *implementation*

From transistors to a simple computer

Hardware-software *interface*

From instruction set architecture to C

Abstraction for practical systems

Memory hierarchy

Operating systems

Higher-level languages

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

It's fascinating, great for critical thinking.

System design principles apply to software too.

Sometimes system abstractions "leak."
Implementation details affect your programs.

`int` ≠ integer
`float` ≠ real

```
int x=…;
x*x >= 0 ?
  40000 * 40000  ==  1600000000
  50000 * 50000  == -1794967296

float a=…, b=…, c=…;
(a + b) + c  ==  a + (y + c) ?
   (-2.7e23 +  2.7e23) + 1.0  == 1.0
    -2.7e23 + (2.7e23  + 1.0) == 0.0
```

28

Reliability

Ariane 5 Rocket, 1996

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

Boeing 787, 2015

"... 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

Arithmetic Performance

x / 973 **x / 1024**

Memory Performance

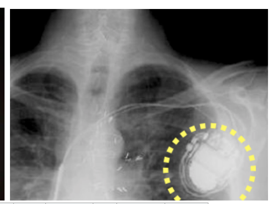
```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

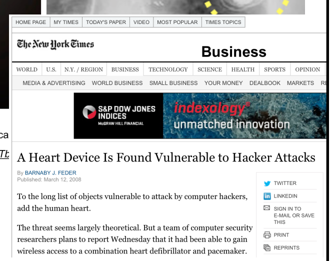
```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

several times faster due to hardware caches

30

Security





Why take CS 240?

- **Learn how computers execute programs.**
- **Build software tools** and appreciate the value of those you use.
- Deepen your appreciation of **abstraction.**
- Learn enduring **system design principles.**
- Improve your **critical thinking** skills.
- Become a **better programmer**:
  - Think rigorously about execution models.
  - Program carefully, defensively.
  - Debug and reason about programs effectively.
  - 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!

Also: C programming language

- Invented to build UNIX operating system, 1970s
  - OS manages hardware, C close to machine model
- Simple pieces look like Java:
  - if, while, for, local variables, assignment, etc.
- Other pieces do not:
  - structs vs. objects, functions vs. methods
  - addresses, pointers
  - no array bounds checks
  - weak type system
- Important language, still widely used, but many better PL ideas have come along since.

https://cs.wellesley.edu/~cs240/

3

Everything is here.
Please read it.