Poll Everywhere

pollev.com/cse333

What was your favorite set of 333 topics?

A. C Programming
B. Build Tools: Preprocessor, Linking, Makefiles
C. File I/O: C Std Lib, POSIX, System Calls
D. Memory Management: malloc/free, new/delete, Smart Pointers
E. C++ Classes, Inheritance
F. C++ Templates, STL
G. Network Programming and HTTP
H. Concurrency: Threads, Processes, Synchronization

Course Wrap-Up

CSE 333 Autumn 2021

Instructor: Chris Thachuk

Teaching Assistants:

Arpad (John) Depaszthory
Ian Hsiao
Logan Gnanapragasam
Mengqi (Frank) Chen
Angela Xu
Khang Vinh Phan
Maruchi Kim
Cosmo Wang

Administrivia

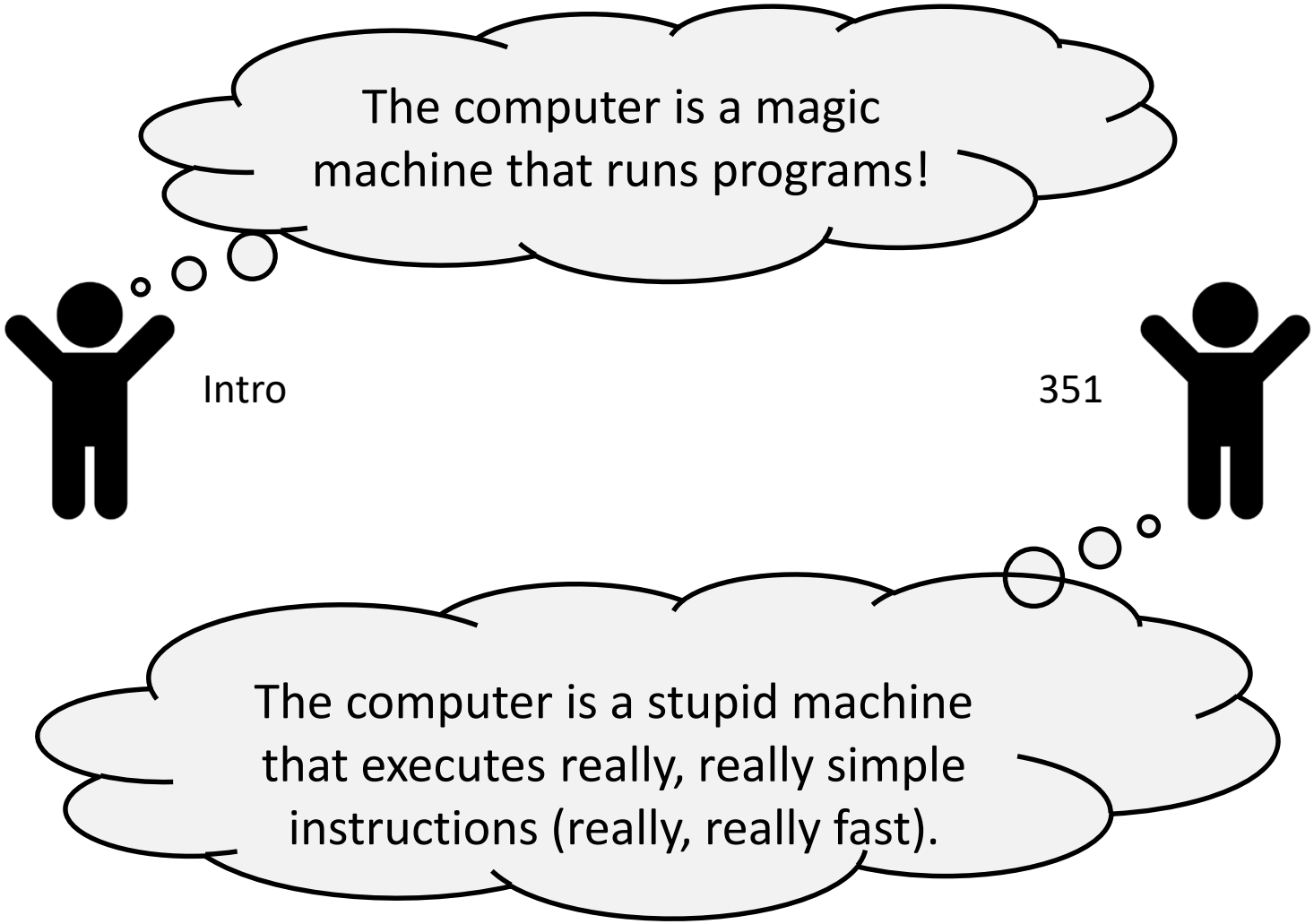
- Homework 4 due yesterday (Dec. 9)
  - Submissions accepted until Sunday (Dec. 12 @ 11pm)
- Course evaluations (Ed Discussion #875) due Sunday night
- Final starts Monday (Dec. 13 @ 8am) and runs until Wednesday (Dec. 15 @ 4:20pm)
  - **Topics:** everything from lecture, exercises, project, etc.
    - (Nothing on Rust, embedded systems, vectorized programming)
  - Written, short-answer questions
  - Gradescope quiz – can open, close, & submit as much as you want
  - Must be completed independently
- Check grades as Canvas assignments are released

What have we been up to for the last 10.5 weeks?

- Ideally, you would have "learned" everything in this course, but we'll use red stars ☆ today to highlight the ideas that we hope stick with you beyond this course

Course Goals

- Explore the gap between:



Systems Programming: The Why

- The programming skills, engineering discipline, and knowledge you need to build a system
  1) Understanding the "layer below" makes you a better programmer at the layer above
  2) Gain experience working with and designing more complex "systems"
  3) Learning how to handle the unique challenges of low-level programming allows you to work directly with the countless "systems" that take advantage of it

So What is a System?

- "A **system** is a group of interacting or interrelated entities that form a unified whole. A system is delineated by its spatial and temporal boundaries, surrounded and influenced by its environment, described by its structure and purpose and expressed in its functioning."
  - https://en.wikipedia.org/wiki/System
  - Still vague, maybe still confusing

- But hopefully you have a better idea of what a system in CS is now
  - What kinds of systems have we seen…?

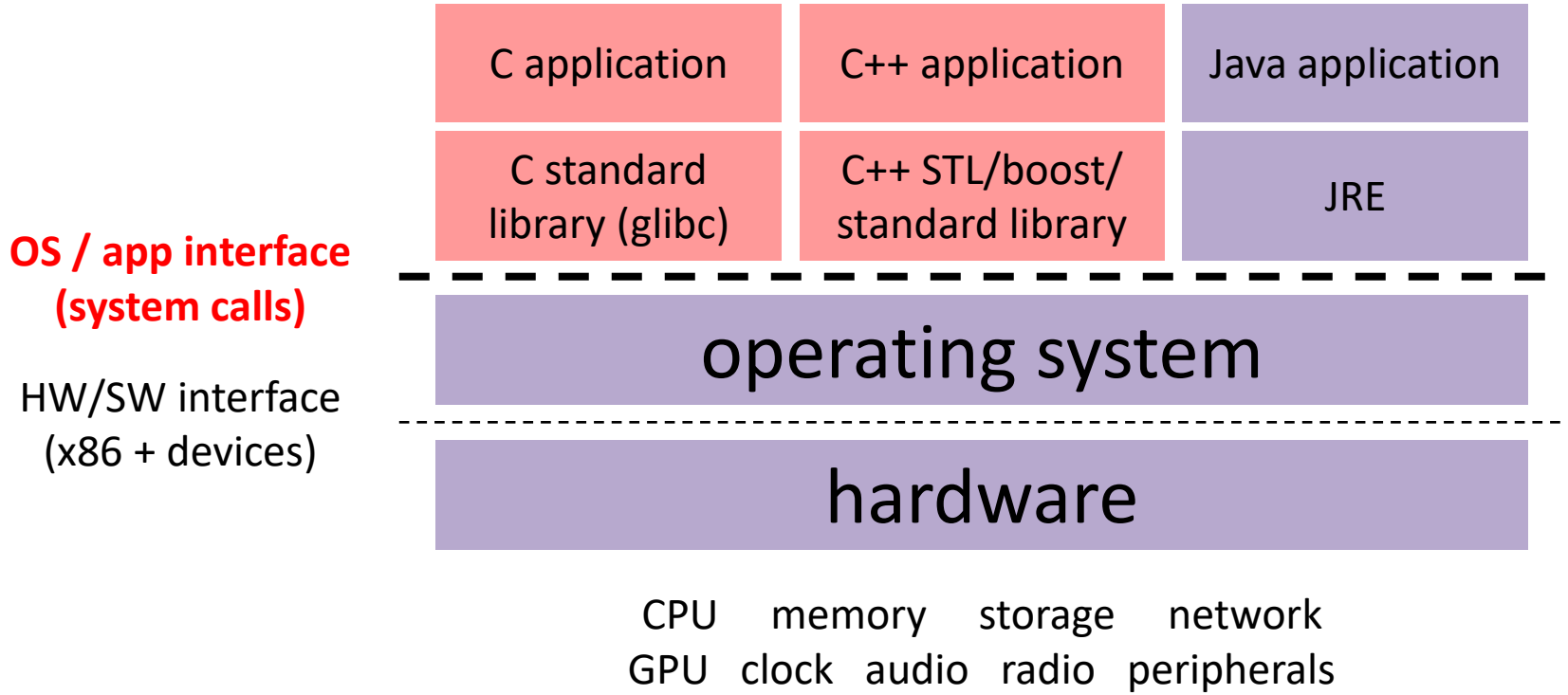
Software System

- Writing complex software systems is *difficult*!
  - Modularization and encapsulation of code
  - Resource management
  - Documentation and specification are critical
  - Robustness and error handling
  - Must be user-friendly and maintained (not write-once, read-never)

- **Discipline:** cultivate good habits, encourage clean code
  - Coding style conventions
  - Unit testing, code coverage testing, regression testing
  - Documentation (code comments, design docs)

The Computer as a System

- Modern computer systems are increasingly complex!
  - Networking, concurrency/parallelism, distributed systems
  - Buffered vs. unbuffered I/O, blocking calls vs. polling, latency



A Network as a System

- A networked system relies heavily on its connectivity
  - Depends on materials, physical distance, network topology, protocols

- Conceptual abstraction layers
  - Physical, data link, network, transport, session, presentation, application
  - Layered *protocol* model
    - We focused on IP (network), TCP (transport), and HTTP (application)
- Network addressing
  - MAC addresses, IP addresses (IPv4/IPv6), DNS (name servers)
- Routing
  - Layered packet payloads, security, and reliability

Systems Programming: The What

- The programming skills, engineering discipline, and knowledge you need to build a system
  -  **Programming:** C / C++
  - **Discipline:** design, testing, debugging, performance analysis
  - **Knowledge:** long list of interesting topics
    - Concurrency, OS interfaces and semantics, techniques for consistent data management, distributed systems algorithms, …
    -  Most important: a deep understanding of the "layer below"

Main Topics

- C
  - Low-level programming language
- C++
  - The 800-lb gorilla of programming languages
  - "better C" + classes + STL + smart pointers + …
- Memory management
- System interfaces and services
- Networking basics – TCP/IP, sockets, …
- Concurrency basics – POSIX threads, synchronization

Topic Theme: Abstraction

- C: `void*` as a generic data type
- C: abstracted data types to hide system-specific details
  - *e.g.*, `size_t`, `int32_t`, `sa_family_t`, `pthread_mutex_t`
- C++: hide execution complexity in simple-looking code
  - *e.g.*, operator overloading, dispatch, containers & algorithms
- C++: templates to generalize code
- OS: abstract away details of interacting with system resources via system call interface
- Networking: 7-layer OSI model hides details of lower layers
  - *e.g.*, DNS abtracts away IP addresses, IP addresses abstract away MAC addresses

Topic Theme: Using Memory

- Variables, scope, and lifetime
  - *Static*, *automatic*, and *dynamic* allocation / lifetime
    - C++ objects and destructors; C++ containers and copying
- Pointers and associated operators (`&`, `*`, `->`, `[]`)
  - Can be used to link data or fake "call-by-reference"
- Dynamic memory allocation
  - **`malloc`**/**`free`** (C), **`new`**/**`delete`** (C++), smart pointers (C++)
  - Who is responsible? Who owns the data? What happens when (not if) you mess this up? (dangling pointers, memory leaks, …)
- Tools
  - Debuggers (`gdb`), monitors (`valgrind`)
  - Most important tool: thinking!

Topic Theme: Data Passing

- C: output parameters
- Processes: status codes (*e.g.*, `EXIT_SUCCESS`)
- Threads: return values or shared memory/resources
  - Leads to synchronization concerns
- I/O to send and receive data from outside of your program (*e.g.*, disk/files, network, streams)
  - Linux/POSIX treats all I/O similarly
  - Takes a LONG time relative to other operations
  - Blocking vs. polling
- Buffers can be used to temporarily hold passed data
  - Buffering can be used to reduce costly I/O accesses, depending on access pattern

Topic Theme: Optimize for your User

- Readability:
  - Properly **modularize** your code using functions, classes, namespaces, and header files
    - Takes advantage of the preprocessor and linker
  - **Documentation** should be thorough, up-to-date, and easy to find (*e.g.*, public interface)
  - Error reporting behaviors should be documented properly
- Usability:
  - Use proper linkage and encapsulation to avoid namespace collisions
  - Make building easy and efficient via build tools (*e.g.*, Makefile)
  - Your programs should be **robust** – no unexpected or unexplained crashes

Congratulations!

- Look how much we learned!
- Lots of effort and work, but lots of useful takeaways:
  - Debugging practice and metacognition (`gdb`, bug journals)
  - Reading documentation
  - Tools (`git`, `valgrind`, makefiles)
  - C and C++ familiarity, including multithreaded and networked code
- Go forth and build cool systems!

Future Courses

- Systems Courses
  - CSE 451: Introduction to Operating Systems
  - CSE 452: Introduction to Distributed Systems
  - CSE 461: Introduction to Computer Communication Networks
  - CSE 401: Introduction to Compiler Construction
  - CSE 444: Database Systems Internals
- Courses in C/C++
  - EE/CSE 474: Introduction to Embedded Systems
  - CSE 455: Computer Vision
  - CSE 457: Computer Graphics
- Otherwise related courses

CSE 451: Intro to Operating Systems

- Pre-reqs: CSE 332, CSE 333
- How do you manage all of the computer's resources?
  - Must do this while being efficient, portable, "fair", etc.
  - Handles concurrency between users/processes
- C programming (and lots of it)
  - Projects come with a LOT of code to build on top of
  - A lot goes into designing what you will do
- The "obvious" follow-up to 333 and often considered a "must-take"

CSE 452: Intro to Distributed Systems

- Pre-reqs: CSE 332, CSE 333; Recommended: CSE 451
- How do you get large collections of computers to collaborate (correctly!)?
  - Reliably, efficiently, to scale, with high availability
  - Fundamentally must deal with concurrency between nodes
- Java programming
- "Probably the most thought per line of code for any course at UW" – Hal Perkins

CSE 461: Intro to Comp Comm Networks

- Pre-reqs: CSE 332, CSE 333
- How to design a network to transmit data?
  - Reliably, to massive scale, securely and "fairly"
  - More than just the 7-layer OSI model we talked about
  - Data encoding, transmission, routing, and security
- Python programming + whatever language(s) you like!
- Networks are involved in almost everything nowadays and knowing their limitations is useful

CSE 401: Intro to Compiler Construction

- Pre-reqs: CSE 332, CSE 351; Recommended: CSE 331
- How does a compiler work?
  - Scan and parse source code, generate a symbol tree, check semantics, optimize, and output assembly
- Java programming
  - Project creates a working compiler
- Theory meets programming meets systems
  - CSE 311, CSE 331, and CSE 351 collide
  - Will always be relevant as new languages and new architectures arise

```
1          .data
2  DisplayOverloading2$$$$:        .quad 0
3          .quad DisplayOverloading2$disp$
4          .quad DisplayOverloading2$disp$
5
6  Overloading$$$$:        .quad 0
7          .quad Overloading$run$0$$$
8
9  DisplayOverloading3$$$$:        .quad D
10         .quad DisplayOverloading2$disp$
11         .quad DisplayOverloading2$disp$
12         .quad DisplayOverloading3$disp$
13         .quad DisplayOverloading3$disp$
14
15         .text
16 DisplayOverloading2$disp$0$$:
17         pushq   %rbp
```

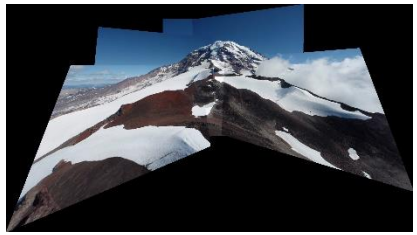
CSE 444: Database Systems Internals

- Pre-reqs: CSE 332, CSE 344; Recommended: CSE 331
- How to build a database management system?
  - Handle (big) data reliably & efficiently
  - Handle parallel accesses to the database
  - Handle errors and crash recovery
- Heavy Java programming
- Design and algorithms for developing a system.
  - Applications of CSE 332 and CSE 333
  - Knowledge carries over into non-database topics

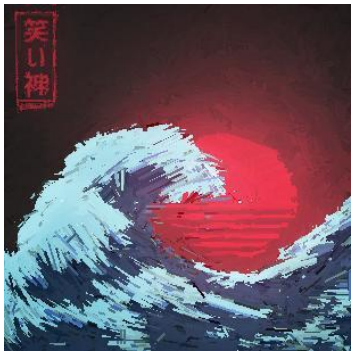
Courses in C/C++

- EE/CSE 474: Intro to Embedded Systems
  - How to interact with computers with limited resources (*e.g.*, RAM) and "real time" requirements
  - Entirely in C, though only pre-req is CSE 143

- CSE 455: Computer Vision
  - Theory-heavy course on the representation and analysis of images (*e.g.*, colors, object recognition)
  - C first half, Python in second half



- CSE 457: Computer Graphics
  - Theory- and coding-heavy course on creating digital art
  - Graphics almost always use C++



Otherwise Related Courses

- CSE 331: Software Design and Implementation
  - Dedicated to good software practices, design, modularity, and more – "core" knowledge for being a good software dev
- CSE 332: Data Structure and Parallelism
  - Use parallelism (a form of concurrency) in Java at the end
- Various Web Programming Courses:
  - CSE 154: Web Programming
  - INFO 340: Client-Side Development
  - INFO 441: Server-Side Development
    - Website design and building

Thanks for a great quarter!

- Special thanks to the course content creators!!!


Steve Gribble


Hal Perkins


John Zahorjan


Hannah Tang


Justin Hsia

- Huge thanks to your awesome TAs!

Thanks for a great quarter!

- And thanks to… **YOU**!
  - It's been a tough quarter as we all adjust to "back in person"
  - You had to deal with a new instructor (me) and our ongoing course development – restructured exercises, new exams format, buggy autograders
  - Be proud of your resilience and what you've still managed to accomplish, even in unideal circumstances

    (Robustness is a hallmark of good systems and good systems programmers.)

- Please take care of yourselves, your friends, and your community – **a lot problems still remain and we all need to be a part of the solution**