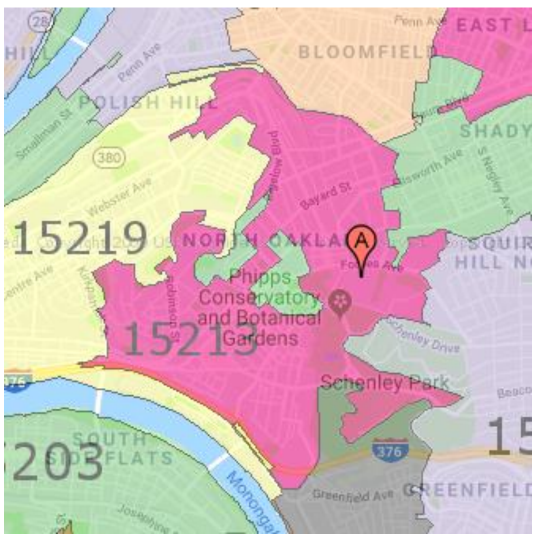
Course Overview

15-213/15-513:
Introduction to Computer Systems
1st Lecture, May 14, 2024



Instructors:

Brian Railing

The course that gives CMU its "Zip"!

Overview

- **Introductions**
- **Big Picture**
  - Course theme
  - Five realities
  - How the course fits into the CS/ECE/INI curriculum
- **Academic integrity**
- **Logistics and Policies**

Instructors

Brian Railing

The Big Picture

Course Theme:
(Systems) Knowledge is Power!

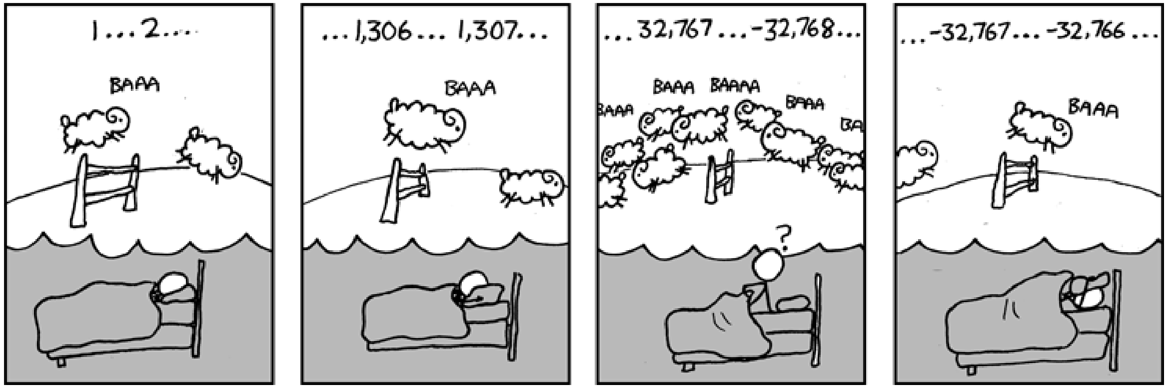
- **Systems Knowledge**
  - How hardware (processors, memories, disk drives, network infrastructure) plus software (operating systems, compilers, libraries, network protocols) combine to support the execution of application programs
  - How you as a programmer can best use these resources
- **Useful outcomes from taking 213/513**
  - Become more effective programmers
    - Able to find and eliminate bugs efficiently
    - Able to understand and tune for program performance
  - Prepare for later "systems" classes in CS & ECE
    - Compilers, Operating Systems, Networks, Computer Architecture, Embedded Systems, Storage Systems, etc.

It's Important to Understand How Things Work

- **Why do I need to know this stuff?**
  - Abstraction is good, but don't forget reality
- **Most CS and CE courses emphasize abstraction; so do we!**
  - Abstract data types
  - Asymptotic analysis
- **But it's helpful to understand what abstractions build upon and their limits!**
  - Especially in the presence of bugs
  - Need to understand details of underlying implementations
  - Sometimes the abstract interfaces don't provide the level of control or performance you need

Great Reality #1:
Ints are not Integers, Floats are not Reals

- **Example 1: Is $x^2 \geq 0$?**
  - Float's: Yes!



  - Int's:
    - 40000 * 40000 --> 1600000000
    - 50000 * 50000 --> ?
- **Example 2: Is (x + y) + z = x + (y + z)?**
  - Unsigned & Signed Int's: Yes!
  - Float's:
    - (1e20 + -1e20) + 3.14 --> 3.14
    - 1e20 + (-1e20 + 3.14) --> ??

Source: xkcd.com/571

Thinking Face …

- **1e20 + (-1e20 + 3.14) --> ??**

Chat GPT says:
The result of **1e20 + (-1e20 + 3.14)** being evaluated as 0 is due to the limitations of floating-point arithmetic.
In floating-point arithmetic, numbers are represented with a fixed number of bits, which means there is a limit to the precision and range of numbers that can be accurately represented. This representation introduces rounding errors and can lead to loss of precision.
When you perform the calculation **1e20 + (-1e20 + 3.14)**, the subtraction **-1e20 + 3.14** evaluates to **3.14** since the magnitude of **-1e20** is much larger than **3.14**, so the smaller value is effectively negligible in terms of precision. Therefore, the expression simplifies to **1e20 + 3.14**.
However, when you add **1e20** and **3.14**, the magnitude of **1e20** is significantly larger than **3.14**. As a result, the addition of **3.14** does not contribute to the value of **1e20** due to the limited precision of floating-point arithmetic. Therefore, the resulting sum is effectively equal to **1e20**.
In mathematical terms, this cancellation of significant digits is known as "catastrophic cancellation." It occurs when there is a subtraction of two nearly equal numbers, leading to the loss of precision in the result.
As a consequence, the total of **1e20 + (-1e20 + 3.14)** is approximated as 1e20, which is effectively zero when compared to the magnitude of **1e20**.

Computer Arithmetic

- **Does not generate random values**
  - Arithmetic operations have important mathematical properties
- **Cannot assume all “usual” mathematical properties**
  - Due to finiteness of representations
  - Integer operations satisfy “ring” properties
    - Commutativity, associativity, distributivity
  - Floating point operations satisfy “ordering” properties
    - Monotonicity, values of signs
- **Observation**
  - Need to understand which abstractions apply in which contexts
  - Important issues for compiler writers and serious application programmers

Great Reality #2: You've Got to Know Assembly

- **Chances are, you'll never write programs in assembly**
  - Compilers are much better & more patient than you are
- **But: Understanding assembly is key to machine-level execution model**
  - Behavior of programs in presence of bugs
    - High-level language models break down
  - Tuning program performance
    - Understand optimizations done / not done by the compiler
    - Understanding sources of program inefficiency
  - Implementing system software
    - Compiler has machine code as target
    - Operating systems must manage process state
  - Creating / fighting malware
    - x86 assembly is the language of choice!

Great Reality #3: Memory Matters

Random Access Memory Is an Unphysical Abstraction

- **Memory is not unbounded**
  - It must be allocated and managed
  - Many applications are memory dominated
- **Memory referencing bugs especially pernicious**
  - Effects are distant in both time and space
- **Memory performance is not uniform**
  - Cache and virtual memory effects can greatly affect program performance
  - Adapting program to characteristics of memory system can lead to major speed improvements

Memory Referencing Bug Example

```
typedef struct {
  int a[2];
  double d;
} struct_t;

double fun(int i) {
  volatile struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741824; /* Possibly out of bounds */
  return s.d;
}
```

```
fun(0)  -->     3.14
fun(1)  -->     3.14
fun(2)  -->     3.1399998664856
fun(3)  -->     2.00000061035156
fun(4)  -->     3.14
fun(6)  -->     Segmentation fault
```

- Result is system specific

Memory Referencing Bug Example

```
typedef struct {
  int a[2];
  double d;
} struct_t;
```

```
fun(0)  -->     3.14
fun(1)  -->     3.14
fun(2)  -->     3.1399998664856
fun(3)  -->     2.00000061035156
fun(4)  -->     3.14
fun(6)  -->     Segmentation fault
```

Explanation:

Critical State	6
?	5
?	4
`d7 ... d4`	3
`d3 ... d0`	2
`a[1]`	1
`a[0]`	0

`struct_t`

Location accessed by `fun(i)`

Memory Referencing Errors

- **C and C++ do not provide any memory protection**
  - Out of bounds array references
  - Invalid pointer values
  - Abuses of malloc/free
- **Can lead to nasty bugs**
  - Whether or not bug has any effect depends on system and compiler
  - Action at a distance
    - Corrupted object logically unrelated to one being accessed
    - Effect of bug may be first observed long after it is generated
- **How can I deal with this?**
  - Program in Java, Ruby, Python, ML, …
  - Understand what possible interactions may occur
  - Use or develop tools to detect referencing errors (e.g. Valgrind)

Great Reality #4: There's more to performance than asymptotic complexity

- **Constant factors matter too!**
- **And even exact op count does not predict performance**
  - Easily see 10:1 performance range depending on how code written
  - Must optimize at multiple levels: algorithm, data representations, procedures, and loops
- **Must understand system to optimize performance**
  - How programs compiled and executed
  - How to measure program performance and identify bottlenecks
  - How to improve performance without destroying code modularity and generality

Memory System Performance Example

```
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];
}
```

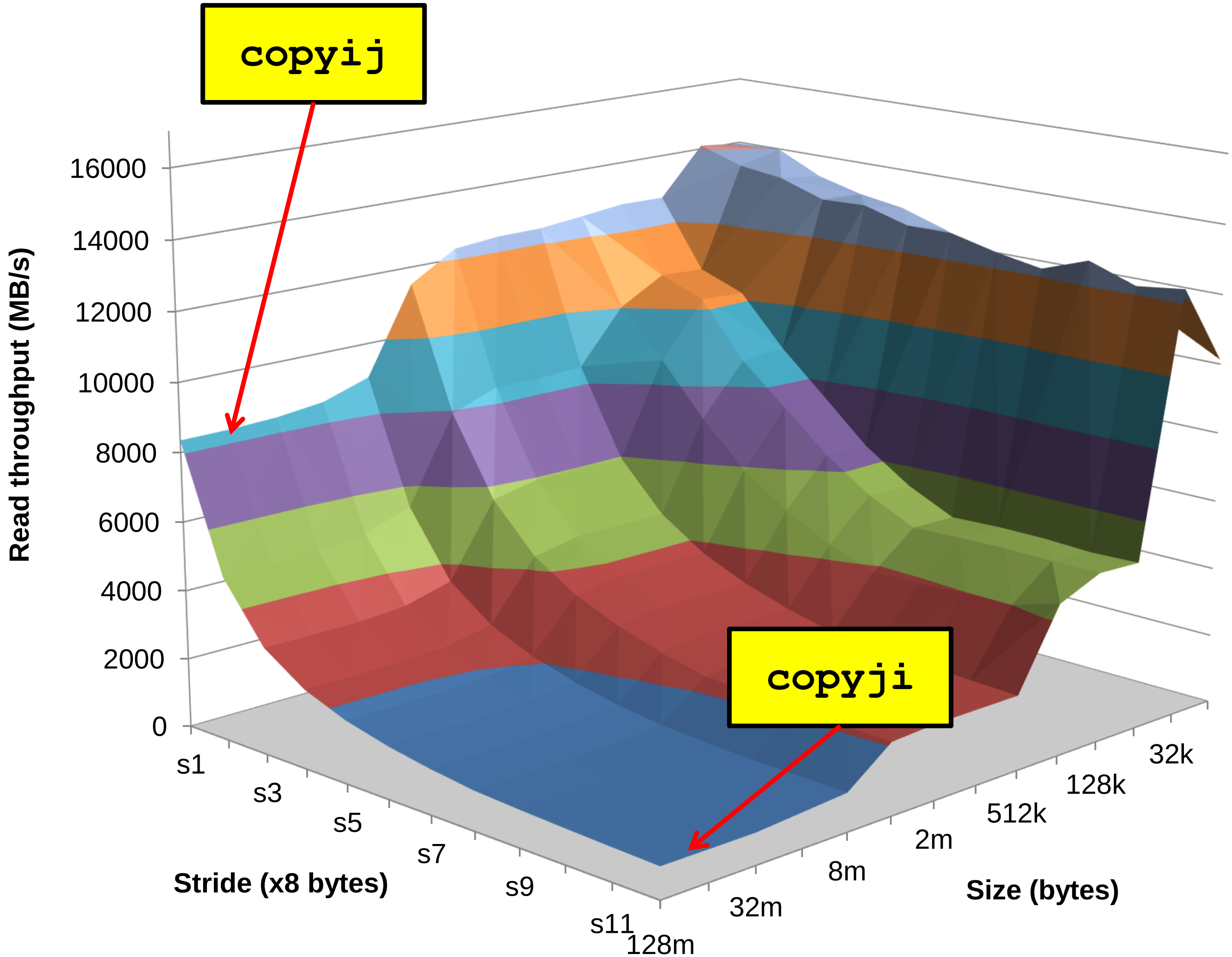
```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

4.3ms **81.8ms**

2.0 GHz Intel Core i7 Haswell

- **Hierarchical memory organization**
- **Performance depends on access patterns**
  - Including how step through multi-dimensional array

Learn why on June 6: Memory Hierarchy



Great Reality #5:
Computers do more than execute programs

- **They need to get data in and out**
  - I/O system critical to program reliability and performance

- **They communicate with each other over networks**
  - Many system-level issues arise in presence of network
    - Concurrent operations by autonomous processes
    - Coping with unreliable media
    - Cross platform compatibility
    - Complex performance issues

Course Perspective

- **Most Systems Courses are Builder-Centric**
  - Computer Architecture
    - Design pipelined processor in Verilog
  - Operating Systems
    - Implement sample portions of operating system
  - Compilers
    - Write compiler for simple language
  - Networking
    - Implement and simulate network protocols

Course Perspective (Cont.)

- **Our Course is Programmer-Centric**
  - By knowing more about the underlying system, you can be more effective as a programmer
  - Enable you to
    - Write programs that are more reliable and efficient
    - Incorporate features that require hooks into OS
      - E.g., concurrency, signal handlers
  - Cover material in this course that you won't see elsewhere
  - Not just a course for dedicated hackers
    - **We bring out the hidden hacker in everyone!**

Role within CS/ECE Curriculum

CS 122
Imperative
Programming

Foundation of Computer Systems
Underlying principles for hardware, software, and networking

213/513

CS Systems
- 15-319 Cloud Computing
- 15-330 Computer Security
- 15-346 Computer Architecture
- 15-410 Operating Systems
- 15-411 Compiler Design
- 15-415 Database Applications
- 15-418 Parallel Computing
- 15-440 Distributed Systems
- 15-441 Computer Networks
- 15-445 Database Systems

ECE Systems
- 18-330 Computer Security
- 18-349 Intro to Embedded Systems
- 18-441 Computer Networks
- 18-447 Computer Architecture
- 18-452 Wireless Networking
- 18-451 Cyberphysical Systems

CS Graphics
- 15-462 Computer Graphics
- 15-463 Comp. Photography

Academic Integrity

Please pay close attention, especially if this is your first semester at CMU

Cheating/Plagiarism: Description

- **Unauthorized use of information**
  - **Borrowing code:** by copying, retyping, ***looking at*** a file
  - **Describing:** verbal description of code from one person to another
    - ***Even if you just describe/discuss how to put together CS:APP code snippets to solve the problem***
  - **Searching the Web** for solutions, discussions, tutorials, blogs, other universities' 213 instances,… in English or any other language
  - **Copying code** from a previous course or online solution
  - **Reusing *your* code** from a previous semester (here or elsewhere)
    - ***If you take the course this semester, all work has to be done this semester (unless you have special permission from the instructors)***
  - ***Using AI to generate your code (e.g., GitHub CoPilot)***

Cheating/Plagiarism: Description (cont.)

- **Unauthorized supplying of information**
  - Providing copy: Giving a copy of a file to someone
  - Providing access:
    - Putting material in unprotected directory
    - Putting material in unprotected code repository (e.g., Github)
      - Or, letting protections expire
  - Applies to this term and the future
    - There is no statute of limitations for academic integrity violations
- **Collaborations beyond high-level, strategic advice**
  - Anything more than block diagram or a few words
  - Code / pseudo-code is NOT high level
  - Coaching, arranging blocks of allowed code is NOT high level
  - Code-level debugging is NOT high level

Cheating/Plagiarism: Description

- **What is NOT cheating?**
  - Explaining how to use systems or tools
  - Helping others with *high-level* design issues
    High means VERY high
  - Using code supplied by us
  - Using code from the CS:APP web site
  - Using books from the library, Unix manpages, other published material
    - Except the "Solutions Manual for CS:APP"
  - Using *general* online references
    - OK: The GNU C Library Manual, Beej's Guide to C, cplusplus.com
    - Not OK: searching for "213 malloc solution"

Cheating/Plagiarism: Attribution

- **If you copy code (that you're allowed to copy), you MUST credit the author**
  - Starter code: No
  - Any other allowed code (course, CS:APP, etc): Yes
  - Indicate source, beginning, and end

- **Here's what it should look like:**

```
/** Hash a 4-byte integer.
 *  This is the "6 shifts" function from
 *  https://burtleburtle.net/bob/hash/integer.html
 *  (second from the top on that page)
 */
uint32_t hash( uint32_t a)
{
    a = (a+0x7ed55d16) + (a<<12);
    // ...
}
```

Cheating: Consequences

- **Penalty for cheating:**
  - Best case: -100% for assignment
    - ***You would be better off to turn in nothing***
  - Worst case: Removal from course with failing grade
    - This is the default
  - University-level involvement (from notification to serious things)
  - Loss of respect by you, the instructors and your colleagues
  - ***If you do cheat – come clean asap!***

- **Detection of cheating:**
  - We have sophisticated tools for detecting code plagiarism
  - In Fall 2015, 20 students were caught cheating and failed the course.
    - Some were **expelled** from the University
  - In January 2016, 11 students were penalized for cheating violations that occurred as far back as Spring 2014.
  - In May 2019, we gave an AIV to a student who took the course in Fall 2018 for unauthorized coaching of a Spring 2019 student. His grade was changed retroactively.

- **Don't do it!**
  - Manage your time carefully
  - Ask the staff for help when you get stuck
  - We will help you! We will give you extensions! We want you to succeed.

Cheating Notes

- **We have written over 100 letters for cheating cases**
  - Don’t add to this total
  - Some have been for years earlier
- **Your work is sophisticated enough that there are many different solutions**
  - Things that look the same are very suspicious
  - If you do your own work and commit regularly, your work is unique
- **We use PhD-level research to detect similarities**
  - Inputs include: multiple tools, online searches, archives of online solutions such as wdxtub, past semester submissions

Some Concrete Examples:

- **This is Cheating:**
  - Searching the internet with the phrase 15-213, 15213, 213, 18213, 513, malloclab, **etc.**
    - That's right, just entering it in a search engine
  - Looking at someone's code on the computer next to yours
  - Giving your code to someone else, now or in the future
  - Posting your code in a publicly accessible place on the Internet, now or in the future
  - Hacking the course infrastructure
- **This is OK (and encouraged):**
  - Googling a man page for fputs
  - Asking a friend for help with gdb
  - Asking a TA or course instructor for help, showing them your code, …
  - Looking in the textbook for a code example
  - Talking about a (high-level) approach to the lab with a classmate

How it Feels: Student and Instructor

- **Fred is desperate. He can't get his code to work and the deadline is drawing near. In panic and frustration, he searches the web and finds a solution posted by a student at U. Oklahoma on Github. He carefully strips out the comments and inserts his own. He changes the names of the variables and functions. Phew! Got it done!**
- **The course staff run checking tools that compare all submitted solutions to the solutions from this and other semesters, along with ones that are on the Web.**
  - Remember: We are as good at web searching as you are
- **Meanwhile, Fred has had an uneasy feeling: Will I get away with it? Why does my conscience bother me?**
- **Fred gets email from an instructor: "Please see me tomorrow at 9:30 am."**
  - Fred does not sleep well that night

How it Feels: Student and Instructor

- **The instructor feels frustrated. His job is to help students learn, not to be police. Every hour he spends looking at code for cheating is time that he cannot spend providing help to students. But, these cases can't be overlooked**
- **At the meeting:**
  - Instructor: "Explain why your code looks so much like the code on Github."
  - Fred: "Gee, I don't know. I guess all solutions look pretty much alike."
  - Instructor: "I don't believe you. I am going to file an academic integrity violation."
    - Fred will have the right to appeal, but the instructor does not need him to admit his guilt in order to penalize him.
- **Consequences**
  - Fred may (most likely) will be given a failing grade for the course
  - Fred will be reported to the university
  - A second AIV will lead to a disciplinary hearing
  - Fred will go through the rest of his life carrying a burden of shame
  - The instructor would have rather spent that time *teaching* Fred in office hours!

Why It's a Big Deal

- **This material is best learned by doing**
  - Even though that can, at times, be difficult and frustrating
  - Starting with a copy of a program and then tweaking it is very different from writing from scratch
    - Planning, designing, organizing a program are important skills
- **We are the gateway to other system courses**
  - Want to make sure everyone completing the course has mastered the material
- **Industry appreciates the value of this course**
  - We want to make sure anyone claiming to have taken the course is prepared for the real world
- **Working in teams and collaboration is an important skill**
  - But only if team members have solid foundations
  - This course is about foundations, not teamwork

Version Control: Your Good Friend

- **Starting with cache lab, labs will be distributed via GitHub Classroom**
- **Must be used by all students**
- **Students must commit early and often**
- **If a student is accused of cheating (plagiarism), we will consult the GIT server and look for a reasonable commit history**
- **Missing GIT history will count against you**
- **Please make sure you have one!**

- **Note: Posting your work for this class in a *public* Git repo (on your personal GitHub account, for instance) is considered an AIV**
  - You're making it too easy for other people to copy from you
  - Use a *private* repo shared with specific people if you want to e.g. show your work to a potential employer

Version Control: Quick Tips

- **Always commit your changes and only your changes**

- **Do not do:**
  - git add .
    - This adds / commits all files that git is either not tracking or have changed
  - git commit –a
    - This commits all files that have changed, which might include accidental edits

- **Do:**
  - git commit <list of files>
    - This commits a specific list of files (usually one) and allows you to write a multi-line commit message

How to Avoid AIVs

- **Start early**
- **Don’t rely on marathon programming sessions**
  - Your brain works better in small bursts of activity
  - Ideas / solutions will come to mind while you’re doing other things
- **Plan for stumbling blocks**
  - Assignment is harder than you expected
  - Code doesn’t work
  - Bugs hard to track down
  - Life gets in the way
    - Minor health issues
    - Unanticipated events
- **Reach out to the faculty! We will help you, give extensions, etc. Our goal is for you to learn & succeed, not be cops.**

Logistics

Education research in this course

- We (faculty) use this course as a basis for formal research studies into how we can teach more effectively.
- Nothing we do for research purposes will affect your grade.
- You have the right to request to be excluded from the studies.

Primary Textbook

- **Randal E. Bryant and David R. O'Hallaron,**
  - *Computer Systems: A Programmer's Perspective*, **Third Edition** (CS:APP3e), Pearson, 2016
  - https://csapp.cs.cmu.edu
  - This book really matters for the course!
    - How to solve labs
    - Practice problems typical of exam problems
  - Electronic editions available (*Don't get paperback/international version*!)
  - On reserve in Sorrells Library

- **Note: All textbooks have errors**
  - Don't panic if you see something that seems wrong
  - Come talk to us about it if you can't make it make sense

Recommended reading

- **Brian Kernighan and Dennis Ritchie,**
  - *The C Programming Language*, Second Edition, Prentice Hall, 1988
  - Everyone calls this book "K&R"
  - Guide to C by the designers of the language
  - Well-written, concise
  - A little dated
    - Doesn't cover additions to C since 1988 (that's thirty years ago…)
    - Casual about issues we consider serious problems now
  - On reserve in Sorrells Library

If you want more books about C

- **C for Programmers with an introduction to C11**
  - Paul and Harvey Deitel
  - Opposite of K&R: modern, verbose
  - Lots of worked-out examples
  - Ugly code style (compare readability to K&R)
- **21st Century C**
  - Ben Klemens
  - Supplement to full C textbooks: goes into the corners of the language
  - Opinionated
  - First half is about how to *build* C programs in the Unix environment
    - So, if you want to understand the Makefiles we give you…
- **Learn C the Hard Way**
  - Zed A. Shaw
  - Extremely opinionated
  - Also has lots of worked-out examples
  - Only book I can find that takes “undefined behavior” seriously enough
- **These books are not on reserve**
  - The library may still have them

Course Components

- **Lectures**
  - Higher level concepts
  - In-class quizzes (except 15-513) could tilt you to a higher grade if borderline
  - New this semester: more involved in-class activities for some lectures
- **Labs (8)**
  - 1-2+ weeks each
  - Provide in-depth understanding of an aspect of systems
  - Programming and measurement
- **Written Assignments (best 10 of 12)**
  - Reinforce concepts
  - You earn 1/4 of score by grading your peers' work according to our rubric
  - Due Wednesdays at 11:59pm ET with peer grades due the next Wednesday
- **Final Exam**
  - Test your understanding of concepts & mathematical principles
  - Covers content from the whole semester

Programs and Data

- **Topics**
  - Bit operations, arithmetic, assembly language programs
  - Representation of C control and data structures
  - Includes aspects of architecture and compilers
- **Assignments**
  - L0 (C programming Lab): Test/refresh your C programming abilities
  - L1 (datalab): Manipulating bits
  - L2 (bomblab): Defusing a binary bomb
  - L3 (attacklab): The basics of code injection attacks

The Memory Hierarchy

- **Topics**
  - Memory technology, memory hierarchy, caches, disks, locality
  - Includes aspects of architecture and OS

- **Assignments**
  - L4 (cachelab): Building a cache simulator and optimizing for locality.
    - Learn how to exploit locality in your programs.

Virtual Memory

- **Topics**
  - Virtual memory, address translation, dynamic storage allocation
  - Includes aspects of architecture and OS
- **Assignments**
  - L5 (malloclab): Writing your own malloc package
    - Get a real feel for systems-level programming

Exceptional Control Flow

- **Topics**
  - Hardware exceptions, processes, process control, Unix signals, nonlocal jumps
  - Includes aspects of compilers, OS, and architecture

- **Assignments**
  - L6 (tshlab): Writing your own Unix shell.
    - A first introduction to concurrency

Networking, and Concurrency

- **Topics**
  - High level and low-level I/O, network programming
  - Internet services, Web servers
  - concurrency, concurrent server design, threads
  - I/O multiplexing with select
  - Includes aspects of networking, OS, and architecture

- **Assignments**
  - L7 (proxylab): Writing your own Web proxy
    - Learn network programming and more about concurrency and synchronization.

Parallelism and Files

- **Topics**
  - Low-level I/O
  - Parallelism and sychronization
  - Includes aspects of OS and architecture

- **Assignments**
  - L8 (sfslab): Extend a basic file system
    - Learn about parallelism and performance
    - Understand further how files / file systems work

Lab Rationale

- **Each lab has a well-defined goal such as solving a puzzle or winning a contest**
- **Doing the lab should result in new skills and concepts**
- **We try to use competition in a fun and healthy way**
  - Set a reasonable threshold for full credit
  - Post intermediate results (anonymized) on Autolab scoreboard for glory!

Lab0: C Programming

- **You can start tonight: see 213 schedule page**
  - (waiting on IT to update the ~213 link)
- **It should all be review:**
  - Basic C control flow, syntax, etc.
  - Explicit memory management, as required in C.
  - Creating and manipulating pointer-based data structures.
  - Implementing robust code that operates correctly with invalid arguments, including NULL pointers.
  - Creating rules in a Makefile
- **If this lab takes you more than 10 hours, please think hard about taking the course.**

Policies: Lab

- **Work groups**
  - You must work alone on all lab assignments

- **Handins**
  - Labs due at 11:59pm ET
  - Electronic handins using **Autolab** (no exceptions!)

Timeliness

- **Grace days**
  - **5 grace days for the semester**
  - **Limit of 0, 1, or 2 grace days per lab used automatically**
  - Covers scheduling crunch, out-of-town trips, illnesses, minor setbacks
- **Lateness penalties**
  - Once grace day(s) used up, get penalized **15% per day**
  - No handins later than **3 days after due date**
- **Catastrophic events**
  - Major illness, death in family, …
  - Fill in form (link on Piazza) to request more time
  - Formulate a plan (with your academic advisor) to get back on track
- **Advice**
  - Once you start running late, it's **really hard** to catch up
  - Try to save your grace days until the last few labs

Facilities

- **Labs will use the Intel Computer Systems Cluster**
  - The “shark machines”
  - `linux> ssh shark.ics.cs.cmu.edu`

  - Servers donated by Intel for 213/513
  - Login using your Andrew ID and password
  - Storage shared with general-purpose Andrew clusters (AFS)

  - Please don’t use the general-purpose Andrew clusters for 213 work
    - They get overloaded
    - They don’t have the right compilers
  - You can try to do the labs on your local machine
    - But it probably doesn’t have the right compilers either
    - `make submit` definitely won’t work

Autolab (https://autolab.andrew.cmu.edu)

- **Labs are provided by the CMU Autolab system**
  - Project page: http://autolab.andrew.cmu.edu
  - Developed by CMU faculty and students
  - Key ideas: Autograding and Scoreboards
    - **Autograding:** Providing you with instant feedback.
    - **Scoreboards:** Real-time, rank-ordered, and anonymous summary.
  - Used by over 3,000 students each semester
- **With Autolab you can use your Web browser to:**
  - Download the lab materials
  - Handin your code for autograding by the Autolab server
  - View the class scoreboard
  - View the complete history of your code handins, autograded results, instructor's evaluations, and gradebook.
  - View the TA annotations of your code for Style points.

Your grade on autolab is your grade for the lab

- **Even if you got a better grade on the sharks**
  - Autolab does more stringent tests for some labs
  - Screenshots can be faked
  - File timestamps can be faked

- **Even if you got a better grade earlier**
  - Don’t submit borderline code over and over
  - All labs are calibrated so it’s possible to get 100% *consistently*

- **One more reason to do the labs early!**

Autolab accounts

- **Students enrolled on Tues, May 14 have Autolab accounts**
- **You must be enrolled to get an account**
  - Autolab is not tied into the Hub’s rosters
  - If you add in, sign up with Google form (check on Piazza)
  - We will update the autolab accounts once a day, so check back in 24 hours.
- **For those who are waiting to add in, request an extension after you have been added, if needed**

Getting Help

- **Class Web page:**
  - **http://www.cs.cmu.edu/~213** for 15-213/15-513
  - Complete schedule of lectures, exams, and assignments
  - Copies of lectures, assignments, exams, solutions
  - FAQ
- **Piazza**
  - Best place for questions about assignments
  - We will fill the FAQ and Piazza with answers to common questions
  - Be careful about public posts: Remember the AIV policy
- **Canvas**
  - Recorded lectures
  - In-class quizzes
  - Written assignments

Getting Help

- **Email**
  - Send email to individual instructors or TAs only to schedule appointments

- **Office hours**
  - TAs: See separate slide for 15-cohort vs. 18-cohort
  - Instructors: See course home page

- **Walk-in Tutoring**
  - Details TBA. Will put information on class webpage.

- **1:1 Appointments**
  - You can schedule 1:1 appointments with any of the teaching staff

Recitations

- **Ad-hoc schedule**

Office Hours

- **TA office hours begin next week (week of May 20)**
- **Both in-person and remote (Zoom) will be offered**
  - Only one or the other, at any particular time
- **Schedule will be posted on Piazza**

- **TAs can spend only 10 minutes per student**
  - Come prepared – have a specific problem you want to ask about
  - Need more time? Schedule a 1:1 session, or **come to faculty office hours**

- **Faculty office hours also begin May 20**
  - Schedule will also be posted on Piazza and the course website
  - Mine are after lecture

Policies: Grading

- **Final Exam (30%)**
- **Labs (50%): weighted according to effort**
- **Written Assignments (20%): drop lowest 2 out of 12**
  - 15/20 points from average of the three scores assigned by the peer graders
  - 5/20 points for completing the peer reviews with constructive feedback
- **In-lecture quizzes and activities are NOT graded**
  - We may look at whether you did the quizzes, for curving only
- **Final grades based on a straight scale (90/80/70/60) with a small amount of curving**
  - Only upward
  - No +/- grades are given (university policy)

Bootcamps

- **Bootcamp #1**
  - Linux, the Command Line and Git
  - TBD, check Piazza for time and zoom link
- **Bootcamp #2**
  - Debugging Fundamentals & GDB
  - See schedule page on the web
- **Bootcamp #3**
  - GCC & Build Automation (makefiles)
  - See schedule page on the web

- **More bootcamps to be announced for specific labs later**

Waitlist questions

- **15-213 / 15-513 (Sara Kuntz <sgolembi@andrew.cmu.edu>)**
  - Waitlist priority is always: are you graduating or does not taking it now directly impact your graduation

- **Please don’t contact the instructors with waitlist questions.**

Managing this course

- **Time management is key**
  - Start early.
  - Office hours are basically empty the first few days an assignment is out.
  - If you feel pressured, do appropriate risk analysis
- **Read the Textbook!**
- **Come to lecture**
- **Go to recitation**

Welcome
and Enjoy!