Course Overview & Logistics

CSCI 237: Computer Organization
1st Lecture, September 8, 2023

Bill Jannen
Today’s Plan

- Welcome and Sign-in
- Course theme
- Five “realities”
- How the course fits into the CS curriculum
- Course logistics and academic integrity
- Hello world, C, and the environment
Course Theme: Abstraction Is Good But Don’t Forget Reality

- Most CS courses emphasize abstraction
  - CS136: Abstract data types
  - CS136, 256, 361: Asymptotic analysis

- These abstractions have limits
  - Especially in the presence of bugs
  - Need to understand details of underlying implementations

- Useful outcomes from taking CS237
  - 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
    - Security (331), Operating Systems (432), Distributed Systems (339), Storage Systems (333), Graphics (373), Applied Algorithms (358), Robotics and Digital Fabrication (345), Parallel Processing (338), …
Great Reality #1: HW Precision is Limited (Ints are not Integers, Floats are not Reals)

- Example 1: Is $x^2 \geq 0$?
  - Floats: Yes!
  - Ints:
    - $40,000 \times 40,000 \rightarrow 1,600,000,000$
    - $50,000 \times 50,000 \rightarrow ??$

- Example 2: Is $(x + y) + z = x + (y + z)$?
  - Unsigned & Signed Int’s: Yes!
  - Float’s:
    - $(1e20 + -1e20) + 3.14 \rightarrow 3.14$
    - $1e20 + (-1e20 + 3.14) \rightarrow ??$

Source: xkcd.com/571
Great Reality #2: Even if you never write it, you’ve got to understand 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
  - Implementing system software
    - Compiler has machine code as target
    - Operating systems must manage process state
  - Tuning program performance
    - Understand optimizations done / not done by the compiler
    - Understanding sources of program inefficiency
  - Creating / fighting malware
    - x86 assembly is the language of choice!
Great Reality #3: Memory Matters
Random Access Memory Is an Abstraction

- Memory is bounded
  - It must be allocated and managed
  - Many applications are memory dominated

- Memory referencing bugs are 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) → 2.0000006103516
- fun(4) → 3.14
- fun(6) → 3.14, 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) ➞ 3.14, Segmentation fault

Program memory:

<table>
<thead>
<tr>
<th>Critical Prog. State</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td>d7 ... d4</td>
<td>3</td>
</tr>
<tr>
<td>d3 ... d0</td>
<td>2</td>
</tr>
<tr>
<td>a[1]</td>
<td>1</td>
</tr>
<tr>
<td>a[0]</td>
<td>0</td>
</tr>
</tbody>
</table>

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 a bug has any effect depends on system and compiler
  - Bug effects may manifest far from the actual bug -> difficult to identify
    - (Space) Corrupted object logically unrelated to one being accessed
    - (Time) Effect of bug may be first observed long after it is generated

- How can I deal with this?
  - Write bug-free code (HAHAHA)
  - Program in managed languages (Java, Python, Ruby, ML, ...)
  - Understand what possible interactions may occur (learned in this course)
  - 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 is written
  - To write fast code, must optimize at multiple levels: algorithm, data representations, procedures, and loops
  - Even system environment variables significantly affect performance

- Must understand whole system to optimize performance
  - How programs are 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];
}

48ms  2.9 GHz Intel Core i5  167ms

- Why? The memory hierarchy!
- Memory performance depends on access patterns
  - Including how to step through a multi-dimensional array (row or column?)
Why The Performance Differs: Locality

- **copyij**
- **copyji**
Great Reality #5: Program code is not the only factor in a program’s execution

- Computers transfer data in and out
  - The I/O system is critical to program reliability and performance

- Computers 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

This Course is Programmer-Centric

- Purpose is to show that by knowing more about the underlying system, one can be more effective as a programmer
- Enable you to:
  - Write programs that are more reliable and efficient
  - Incorporate features that require cooperation with OS
    - E.g., concurrency, signal handlers
- We 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!
  - Even if you plan to write phone apps in Java, this knowledge will help
Role within CS Curriculum

CS 339 Distributed Systems
- Network Prog Concurrency

CS 336 Networks
- Network Protocols

CS 432 Operating Systems
- Processes Mem. Mgmt

CS 434 Compilers
- Machine Code

CS 337 Digital Design/Modern Arch
- Execution Model Memory System
- Mem. Mgmt Efficiency

CS 333 Storage Systems
- Mem Mgmt

CS 334 Program. Lang.

237

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

CS 134/136 Imperative Programming
Administrative Details

- (Next few slides can be found on syllabus, and slides will be posted online to review...)
Administrative Details

- Course syllabus (see PDF on website)

- All sections have the same content, outcomes, and resources

- Labs: Weds 1-2:25 & 2:35-4 in TCL 312
  Thurs 1-2:25 & 2:35-4 in TCL 312

- Instructor: 09wkj@williams.edu, TPL 304

- TA and office hours are posted on course webpage
Textbooks

- Randal E. Bryant and David R. O’Hallaron,
  - [http://csapp.cs.cmu.edu](http://csapp.cs.cmu.edu)
  - This book really matters for the course!
    - How to solve labs
    - Practice problems typical of exam problems

- Brian Kernighan and Dennis Ritchie,
  - Still (arguably) the best book about C, from the originators
  - Page 93!
Course Components

- Lectures
  - Mostly higher level concepts, some info about important tools and skills for labs

- Labs
  - Challenging, but very rewarding and (hopefully) fun
  - 1-2 weeks each, mostly due on Tuesdays
  - Provide in-depth understanding of an aspect of systems
  - Often involve both programming and measurement

- Practice Problems & In-lab “Quizzes”
  - Practice for important concepts to help prepare for exams

- Exams (midterm + final)
  - Test your understanding of concepts & mathematical principles
  - Midterm: evening exam on October 19
  - Final: scheduled (information TBA)
Getting Help

- Class Web page: http://www.cs.williams.edu/~cs237/
  - Complete schedule of lectures, exams, and assignments
  - Copies of lectures, labs, exams, sample solutions (?)
  - Clarifications to labs

- GLOW
  - We won’t be using GLOW for the course
  - ...but we will make the Facebook available to get to know classmates

- Mailing list/Google Group
  - Forum for asking/answering questions
  - We will post announcements and hints here
Getting Help

- **Office hours**
  - Held in TCL 312 on Monday and Tuesday afternoons

- **1:1 appointments on Thursday**
  - If you are stuck, the earlier you reach out the better!

- **TA hours**
  - Most nights except Wednesday & Friday/Saturday

- **Peer tutoring program**

- **YOU WILL NEVER BE PENALIZED FOR SEEKING HELP!!!**
Policies: Labs And Exams

■ Work groups
  ▪ You must work alone on all lab assignments unless otherwise noted
  ▪ Allowed types of collaboration will be detailed in lab handouts

■ Handins
  ▪ Labs due at 10pm on date specified (often 2 parts per lab)
  ▪ Electronic handins using GitLab

■ Exams
  ▪ Closed book
  ▪ Midterm October 19 (evening)
  ▪ Final exam details to come later

■ Appealing grades
  ▪ We all make mistakes! Come see me ASAP.
Cheating: Description

- Please pay close attention! We take this VERY seriously.

- What is cheating?
  - Sharing code: by copying, retyping, looking at, or supplying a file
  - Describing: verbal description of code from one person to another
  - Coaching: helping your friend to write a lab, line by line
  - Searching the Web for solutions
  - Copying code from a previous course or online solution
    - You are only allowed to use code we supply, or from the CS:APP website

- What is NOT cheating?
  - Explaining how to use systems or tools, Unix environment
  - Helping others with high-level design issues (be sure to give credit)
  - Discussing the documentation or asking clarifying questions about course materials

- See the course syllabus/assignment description for more details.
  - Ignorance is not an excuse. When in doubt, ask!
Cheating: Consequences

- **Penalty for cheating:**
  - According to our Honor Code, if we suspect cheating, we must notify the honor committee
  - If found guilty, probably will fail course (less serious outcomes possible, but less common)
  - Permanent mark on your record

- **Detection of cheating:**
  - We have sophisticated tools for detecting code plagiarism
  - We (unfortunately) catch students every semester

- **Just don’t do it!**
  - Start early
  - Ask for help when you get stuck
Facilities

- Labs work on environment we’ve created on the CS lab machines
- Use SSH to work remotely, or VM to work on laptop
  - Many command line tools, so emacs/vim/nano are great editors
  - Git let’s us move around among machines
    - commit and push *often*
Facilities (continued)

- Try to log in soon. If you have any trouble, Email Lida Doret and I for account setup/password reset help
Timeliness

- **Grace days**
  - 3 grace days for the semester
  - Limit of 2 grace days per any single lab
  - Covers scheduling crunch, out-of-town trips, illnesses, minor setbacks

- **Lateness penalties**
  - Once grace days are used up, we will grade the latest version

- **Catastrophic events**
  - Major illness, death in family, ... 
  - Come talk to us (and probably get help from a dean—they are great!)

- **Advice**
  - Once you start running late/falling behind, it’s really hard to catch up, so reach out early
Other Rules of the Classroom

- Laptops permitted for note taking, but please sit side/back

- Electronic communications
  - No email, instant messaging, texting, cell phone calls, etc, during class
  - Distracting to everyone around you

- Presence in course meetings and labs is required
  - Let me know if you need to miss
Policies: Grading

- Homework (0%): Practice problems covering important topics to help prepare for exams and quizzes. TAs available to help answer questions.

- Quizzes (5%): announced in advance, given at start of lab.

- Labs (40%): weighted (slightly) according to effort

- Exams (55%): midterm (25%), final (30%)
Any Questions?

- I have questions for you!
- Getting to know you form...
  
  www.cs.williams.edu/~cs237/schedule.html
Hello World, Compiling, and Running C

Tasks:

- Print hello world
- Print sum of two numbers
- Print sum of two command line arguments
- Print sum of N command line arguments