EECS 281
Data Structures and Algorithms

Prof. Marcus Darden
Dr. David Paoletti
Spring 2024

eecs281admin@umich.edu

eecs281admin Email

- Seen by faculty and a few select staff
- You can feel safe contacting us with medical issues, kept entirely confidential
  - Everyone goes through FERPA training
- If you don't get a sufficient response or sufficiently quickly, come to Proffice Hours
  - Offered almost every weekday
  - Ask to speak to us individually

Grading

- Grading Policy
  - 20% Labs (10)
  - 40% Projects (4)
  - 20% Midterm Exam
  - 20% Final Exam

What Guarantees that I Pass?

- Achieve minimal competency
- If you earn **ALL OF**:
  - (>= 50% on Exams)
  - **AND** (>= 55% on Projects)
  - **AND** (>= 75% on Labs)
- You **WILL** pass this course
- A total of 68, with 30% projects, 100% labs and 90% exams is **NOT PASSING**

Labs (20%)

- 10 lab assignments
- **Can work with other students**
- Submit on paper (in lab), electronically via Canvas, and/or autograder machine
- Late submissions for **Quizzes** & **AG** assignments are accepted at 50% credit
  - Up to midnight of the day before each exam
  - Do not ask for extensions
  - Cannot use late days

Lab Times

- Labs meet Tuesday – Monday every week
- You do not have to attend the lab that you're enrolled in
- We would like you to attend the same lab each week
  - Make contacts and consistent partnerships

Lab Written Portion

- Every lab has a "written" problem
  - Done on paper, during lab
  - This is practice for the exams
  - It is graded by effort
  - You can miss up to two at full credit
- These problems will prepare you for writing code by hand on the exams, in interviews, etc.

Projects (40%)

- 4 projects
- **Individual work**
- Submitted electronically to autograder
  - Details to follow
- Approximately 3 weeks per project
  - Less in Spring
- Late submissions: USE LATE DAYS WISELY (see "Policy on Deadlines")

Policy on Deadlines

- Autograder: 2 Late Days per semester
- Use them as you want
- Project 0 late days are “free”, use them for practice!
  - Before any “real” assignment is due, everyone will have their late days restored
- Example: if a Project was due Tuesday, today is Thursday; you didn’t submit yesterday = 2 late days to submit today (submitting 2 days late)

27

Projects (40%)

- C++ (International C++11 Standard)
  - https://en.wikipedia.org/wiki/C++11
- CAEN Linux Computing Environment
  - g++ (GCC) 11.3.0
- Beware if you are doing development in any other environment
  - May compile/run perfectly for you, then not even compile on the autograder

28

Autograder

- We will grade projects with an autograder
  - Correctness, timing and memory usage
- Immediate feedback on most test cases
- ~3 submissions per day
  - Some projects have more, some have two parts
  - +1 submit per day for finding enough bugs!
  - More in Spring (due to double speed)

29

Exams (40%)

- Midterm Exam (20%)
- Final Exam (20%)
- Will test your **understanding** of material and **problem-solving skills**
- Both a multiple choice section and a long answer section
- Must notify instructor 2 weeks ahead if conflict
- Cannot miss exam without documented serious medical or personal emergency

34

Will Solutions Be Posted?

- Yes - for labs (see Canvas after the due date)
- No
  - For in-class exercises (some yes, some no)
  - For projects
  - For exams
- Midterm solutions may be outlined in class
- Clarifications on Piazza and office hours

36

Lectures

- Not all material presented in lecture will appear in the lecture slides
  - Explanations on a tablet
  - Additional practice questions
- If you are not following lecture material, don’t wait until just before the exam
  - Ask questions, attend office hours

38

Useful tools

- Automated compilations
  - make
- Editors for “power users”
  - Vim, Emacs
- Version control system
  - Git (use private repositories only!)
    - https://github.com/
    - https://gitlab.eecs.umich.edu

44

Partial List of IDEs

Proprietary

- Visual Studio 20XX, Enterprise or Community
  - Enterprise edition
  - Community edition
  - C++, C#
  - PC only
- Xcode (free)
  - apple.com
  - C++, Swift, Objective-C
  - Mac only

Multiple Platforms*

- NetBeans (free)
  - netbeans.org
  - C++, Java, etc.
- Eclipse (free)
  - eclipse.org
  - C++, Java, etc.
- VS Code (free)

*Need a separate g++ compiler such as Cygwin or Min-GW

48

Plotting Tools

- Useful for plotting algorithm statistics
  - Runtimes
  - Memory Usage
  - Other parameters
- Gnuplot (installed on CAEN Unix)
  - http://www.gnuplot.info/
- Google Sheets
- Excel (installed on CAEN Windows)
  - http://www.usd.edu/trio/tut/excel/
- Matlab (installed on CAEN Windows)
  - http://www.math.ufl.edu/help/matlab-tutorial/

Pre-Midterm: Foundational Skills & Techniques

- Complexity analysis of algorithms
- Building blocks – elementary algorithms & data structures
  - Sorting, searching, stacks and queues, priority queues (+ possibly more)
- Implementation in C++11 using STL
  - How to be efficient, what to avoid
- Time measurement and optimization
- Algorithmic problem-solving
- Examples for how to select the best algorithm for a problem

After the Midterm: Sophisticated Algorithms

- Binary search trees (dictionaries)
- Hashing and hash tables
- Graph algorithms
- Algorithm types
  - Divide-and-conquer
  - Greedy
  - Dynamic programming
  - Backtracking and branch-and-bound

Data Structures and ADTs

- Need a way to store and organize data in order to facilitate access and modifications
- An **abstract data type (ADT)** defines a collection of valid operations and their behaviors on stored data
  - e.g., insert, delete, access
  - ADTs define an interface
- A **data structure** provides a concrete implementation of an ADT

Algorithms

- An **algorithm** is a well-defined procedure that solves a computational problem
  - Transforms given input data into desired output or answer
- "Recipe" or "Set of Directions"
- Algorithms are tools for solving problems
  - Sort a list of data, find the shortest path between classes, pack as many boxes as possible in a delivery truck

Analyzing Data Structures and Algorithms

- When designing algorithms and DSs, we care about:
  - How long does an operation take (# of steps)?
  - How much space is used?
- Predict answers before running the code
  - Avoid wasting time on bad designs
- **Complexity analysis** answers these questions relative to the size/quantity of input data