Welcome to EECS/CS 281!

Me: Sugih Jamin  
email: jamin@eecs.umich.edu  
Office: 4737 CSE  
Hours: Tue/Thu 3:00-3:45, Fri 3:00-4:00, and by appt.  
(but *never* right before lecture)  
Tel: +1 734 763 1583

Prereqs: EECS 203 and 280. Strict

GSIs and IA:

- Brian Kirby (kirbyb)
- TBD (): TBD
- Christopher Peplin (peplin)

GSI office hours will be held at the NE corner of the Dude 3rd flr.
Readings

Required:

- Preiss, *Data Structures and Algorithms with Object-Oriented Design Patterns in C++*
- Class handouts
- Web site and Forum(!)

Recommended:

- Soulie, J., *C++ Tutorial* (online)
- C++ *stdlib and iostream Reference* (online)
- Sebern, *string class* (online)
- Josuttis, *The C++ Standard Library*
Web site and Forum

Web site: http://irl.eecs.umich.edu/jamin/courses/eecs281/

Forum (not up yet):

- register first
- **DON’T** use your CAEN passwd
- either use your uniqname as the forum account or use your umich email, so that we can verify membership
- **DON’T** post solutions! (Honor Code violation)
  No: here’s my code, why doesn’t it compile/run
  No: this problem can be solved using data structure X, or algorithm Y
  If in doubt, err on caution, and ask by email
What 281 covers

How to build various *tools* from existing ones (data structures)

How some tools are better for certain tasks than other tools (performance analysis)
 (“When you have a hammer every problem looks like a nail.” Not)

How to go about solving a task using the tools (algorithms)

How to improve the tools and solutions, and to hone your programming skills

Sugih Jamin (jamin@eecs.umich.edu)
Data Structures and Algorithms

Data structures: collections of variables, possibly of different data types, connected in various ways

Algorithms: methods for solving problems that are suited for computer applications

Algorithm + Data Structures = Programs [N. Wirth]

Algorithmic Thinking: “Conceptualize problems with digital representations and seeks algorithms that express or find solutions” – P.J. Denning
Discussion Session

Make sure you register for a discussion session!

- W 3:30-4:30 2150 DOW
- W 3:30-4:30 1003 EECS
- F 1:30-2:30 1006 DOW

(there will be NO discussion this week)
Grading Policy

Course grade composition:

- Final: 20%
- Midterm: 20%
- Homeworks (HWs): 12%
- Programming Assignments (PAs): 46%
- Class Participation: 2%

Four free late days (incl. weekends and holidays)

You must keep track of your own late days
Regrade

- except for arithmetic error in grade computation, all regrade requests must be made in writing
- must explain the technical reasons requiring regrade
- must be made 5 working days after the work is returned to the class (except same day for final exam)
- the whole work will be regraded
Cheating and Collaboration

You are encouraged to learn from each other

We look forward to lively discussions in class and on the forum

However, cheating is not tolerated and will be reported to the Honor Council

Cheating is when you copy, with or without modification, someone else’s work

It is also considered cheating to knowingly expose your solutions or to use some other publicly available solutions

NOTE: It is also considered cheating to have a different algorithm implemented than the one specified in the programming assignment

When in doubt, ask

Sugih Jamin (jamin@eecs.umich.edu)
Exams

Midterm: Thu, Oct. 25th, 6:00-8:00 pm
Final Exam: Wed, Dec. 19th, 4:00-6:00 pm

Scheduling conflicts:

- only **documented** medical or personal emergency allowed
- if you need extra time to complete an exam due to personal disability, please inform us **1 week** in advance
- other scheduling conflicts will not be considered **two weeks** after **the start of the term (today)**
- outside commitments, e.g., job interviews, marching band trips, top-coder contests, are **not** considered valid reasons for missing an exam

Sugih Jamin (jamin@eecs.umich.edu)
Assignments

Homeworks to be turned in hardcopy in the 281 lockbox in 2420 EECS

Programming assignments:

- no group project
- no STL
- must not include external materials (e.g., open-source code downloads) unless requested
- must compile and run on CAEN’s Linux with the latest version of g++

Put your uniqname on EVERYTHING you turn in

Do NOT submit ANYTHING by email
Programming Assignment Grading

Grading criteria:

- code compiles
- code runs correctly
- code runs fast
- code is readable, well-documented, e.g.,
  - no use of literals
  - code re-use instead of cut-and-paste
- algorithm is efficient
- implementation is efficient, e.g.,
  - no unnecessary copying
  - no loop invariant statement in loop
Magic Show
Performance Analysis of Name Search

Best case

Worst case

Average case

Common case

(What’s the difference between average and common cases?)
Execution Cost

Algorithms take resources to execute

Resources:

- Space: memory, bandwidth
- Time

Two types of resource cost:

- fixed cost
- variable cost
Space Time Tradeoff

If you can load all your data into memory, you can sometimes come up with very fast algorithm

Where space matters:

- handheld devices
- embedded systems
- large data-set problems, examples:
Timing Cost

Fixed cost (one-time cost):

- coding time
- compile time
- variable initializations
- etc.

Variable cost:

- run time: depends on the size of the problem
Runtimes of our name search problem

Assume can search 10 names/ms

<table>
<thead>
<tr>
<th>Population (size)</th>
<th>Linear</th>
<th>Binary</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EECS 281 (100)</td>
<td>10 ms</td>
<td>0.7 ms (lookup 7 names)</td>
<td></td>
</tr>
<tr>
<td>UM (40000)</td>
<td>4 secs</td>
<td>1.5 ms</td>
<td></td>
</tr>
<tr>
<td>Washtenaw</td>
<td>35 secs</td>
<td>1.8 ms</td>
<td></td>
</tr>
<tr>
<td>MI (9 mil)</td>
<td>15 mins</td>
<td>2.3 ms</td>
<td></td>
</tr>
<tr>
<td>US (290 mil)</td>
<td>8 hours</td>
<td>2.8 ms</td>
<td></td>
</tr>
<tr>
<td>World (6 billion)</td>
<td>7 days</td>
<td>3.3 ms</td>
<td></td>
</tr>
</tbody>
</table>

Implication: there is usually more than one way to solve a problem

Want: the most efficient way
Algorithm Design

Typical approach:

- define problem to be solved
- understand its complexity
- decompose it into smaller subtasks

Algorithmic Thinking: “Conceptualize problems with digital representations and seeks algorithms that express or find solutions” – P.J. Denning
Algorithm Design

Typical approach:

• define problem to be solved
• understand its complexity
• decompose it into smaller subtasks

Creative refinements:

• think outside the box: exploit any particular characteristics of the workload or problem
• don’t lose the forest for the tree:
  - find the real performance bottleneck of the whole program (how?)
  - also note the human labor cost
• optimize for the common case

Sugih Jamin (jamin@eecs.umich.edu)
Mirror, mirror on the wall, . . . .

Given two algorithms, how do you determine which is more efficient?

Example: roofing problem

- A new house is being built down the road
- The builders are currently working on the roof
- There are three different methods for moving shingles from the shingle truck to the roof
Three Roofing Methods

Method 1:

A builder can carry two shingles on his shoulder as he climbs up the ladder. He then climbs down and carries two more shingles up the ladder. Each round trip (up and down the ladder) costs $2.

Method 2:

The builder rents a lift for $10. The lift can move 20 shingles up the roof at one time, at a cost of $1 per round trip.

Method 3:

The builder rents a super-lift for $40. Unfortunately, the lift has a slow leak in its hydraulic system. It is able to lift half of the necessary shingles to the roof on the first round-trip. However, on the second trip, it is only able to lift half of the remaining shingles, then half of the remaining . . . down to the minimum of one shingle per round trip. Each round trip costs $2.

In all three methods, it costs $4/shingle to nail it to the roof.
Back-of-the-Envelope “Analysis”

Questions:

1. Which is the cheapest of the three methods to build a tree house (8 shingles)?
2. Which is the cheapest for a shed (128 shingles)?
3. Which is the cheapest for a house (2048 shingles)?