Programming Abstractions

CS106B

Cynthia Bailey
Chris Gregg

pollev.com/cs106b

Topics:

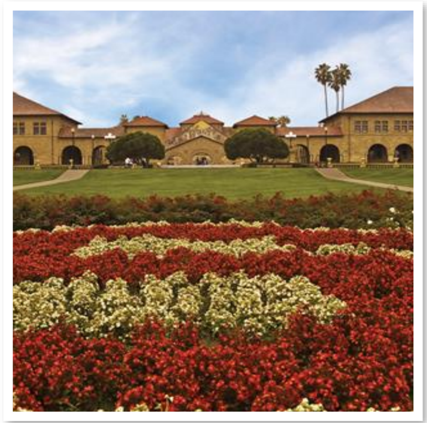
- **Midterm Review**
  - Overview of what to expect
  - Going over solutions to a couple new practice problems
  - Q&A

- *Additional resources:*
  - For even more review, see review session led by Jonathan last Thursday! Video and slides available, see pinned Ed post.

- **For important announcements, be sure to see the weekly announcements post on the Ed Q&A board! https://edstem.org**
- **Also on Ed: live lecture Q&A with Chris & Jonathan**

Midterm Exam

What to expect



The Midterm Exam

- **Time: 2 Hours**
  - Very first thing you should do: **write your SUID on every page.**
    - Forgetting this would be the saddest reason to lose points! And you will NOT be given extra time after time is called to do this. No writing at all after time is called.
  - 4 questions.
  - I'd plan **25 minutes per problem**, which leaves you 20 minutes at the end to re-check your work etc.

The Midterm Exam

- **Format: Handwritten on paper**
  - I would be sure to solve some practice problems with pen & paper so this doesn't feel *completely* new and strange!
  - In grading, we try to ignore minor "typos" in handwritten code that likely would have been prevented or caught by an IDE
    - Just make sure intent is clear—i.e. we are happy to ignore that you wrote "stepcount" instead of "stepCount," but not if you have variables with both names in your code and we actually do need to know which one it is on a given line. ☺
  - You won't be able to actually compile, run, and test your paper-written code, but you'll very much want to be thinking in terms of what you would put in "STUDENT_TEST"s for the code you're writing.
    - To simulate a test: pick a concrete example, then try to set your intent aside and painstakingly trace through the example input line by line to check. Repeat for 1-2 more concrete examples.
    - (Note: familiarity with our STUDENT_TEST code format is expected knowledge on the exam.)

The Midterm Exam

- **Problem Topics:**
  - *(Same general categories as the practice exam)*

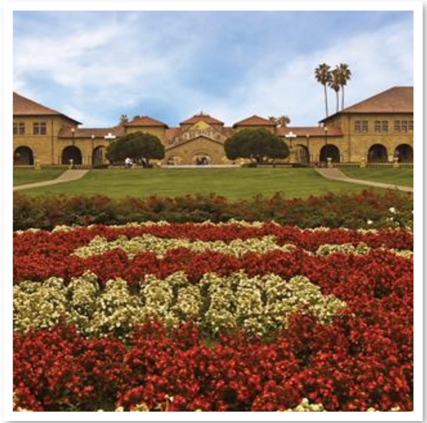
  1. C++ Fundamentals
  2. ADTs
  3. Big-O
  4. Recursion

The Midterm Exam

- **Rules: Closed book, no devices**
  - This is actually to make the exam easier, I promise! ☺
- **Reference Sheet (included in exam)**
  - It's on the course website, so study it ahead of time!
- **Cheat Sheet (your own to bring with you)**
  - One sheet of paper (8.5x11), both sides, handwritten

Sample Problem 1: C++ Fundamentals

What to expect, solution

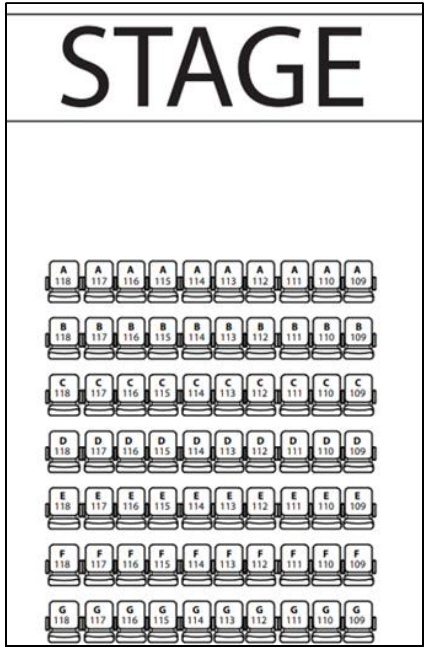


What to expect

- Strings
- Loops, nested loops
- Multi-part conditionals inside a loop
- Thinking carefully about edge cases

Sample Q1: Grid of theater seat reservations

- You are given a `Grid<string>` representing the seating chart of a theater.
- Your function will help customers looking to make group reservations by taking the grid and looking for a row with at least n consecutive open seats.
  - If the seat has been reserved, the string is that person's name, otherwise the string is empty string.
- Return the `GridLocation` of the leftmost seat of the matching block of seats.
  - You may assume a satisfying location exists.


STAGE
A 118, A 117, A 116, A 115, A 114, A 113, A 112, A 111, A 110, A 109
B 118, B 117, B 116, B 115, B 114, B 113, B 112, B 111, B 110, B 109
C 118, C 117, C 116, C 115, C 114, C 113, C 112, C 111, C 110, C 109
D 118, D 117, D 116, D 115, D 114, D 113, D 112, D 111, D 110, D 109
E 118, E 117, E 116, E 115, E 114, E 113, E 112, E 111, E 110, E 109
F 118, F 117, F 116, F 115, F 114, F 113, F 112, F 111, F 110, F 109
G 118, G 117, G 116, G 115, G 114, G 113, G 112, G 111, G 110, G 109

Sample Q1: Strategy

- We need to loop over the Grid
  - Should we use `for (string entry : grid)` style loop?
    - Not in this case, because we need to be aware of our current row (blocks can't cross from one row to another)
    - Use int row/col nested loops
- Within a row, we need to count free seats
  - Start count at 0 again each row
  - Iterate over cols, incrementing if empty
    - If not empty, reset count to 0
  - If we find something that works, immediately store location and return

Sample Q1: Solution

```
GridLocation findSeats(Grid<string>& seatGrid, int n) {
   GridLocation loc(0, 0);
   for (int r = 0; r < seatGrid.numRows(); r++) {
      int count = 0; // count consecutive empty seats
      for (int c = 0; c < seatGrid.numCols(); c++) {
         if (seatGrid[r][c] == "") {
            count++;
            if (count == n) {
               loc = { r, c - n + 1 };
               return loc;
            }
         } else {
            count = 0; // if non-empty string, need to reset counter
         }
      }
   }
   return loc; // shouldn’t reach this but keep compiler happy
}
```

Stanford University

Your Turn

Q: What's something you should be sure to include on your cheat sheet to help with a C++ fundamentals question?

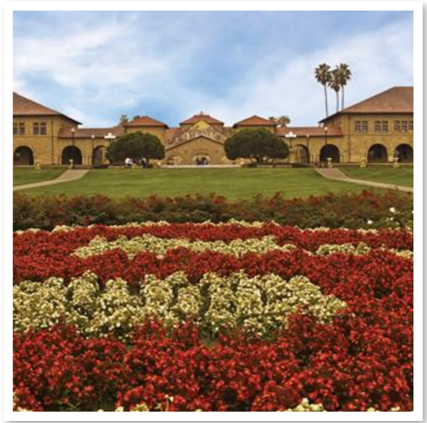
- **Give one idea per pollev response**
- Feel free to give multiple responses

pollev.com/cs106b

Stanford University

Sample Problem 2: ADTs

What to expect, solution



What to expect

- Uses of the ADTs that really “fit” the situation
  - Either by our design, or we ask you to choose the best design
- Code that is much simpler (!!) if you use elegant ADT features
  - Range-based for loop (example: `for (string s : themap)`)
  - Push/pop on a Stack vs manual equivalent on a Vector
  - Set operations vs manual equivalent
- One kind of ADT alongside or inside another kind of ADT
- *Theme*: really let the ADTs “shine”

Sample Q2: ADTs problem

Write a function that takes a `Queue<int>` and returns a `Queue<int>` where the elements are in reverse of the original order.

The challenge: Your function may declare only one local variable: your choice of either a `Stack` or a `Queue`.

```
Queue<int> reverse(Queue<int> q);
```

pollev.com/cs106b

Would you choose a Stack or Queue for this?

Sample Q2 Solution

```
Queue<int> reverse(Queue<int> q) {
    Stack<int> theStack;
    while (!q.isEmpty()) {
        int x = q.dequeue();
        theStack.push(x);
    }
    while (!theStack.isEmpty()) {
        q.enqueue(theStack.pop());
    }
    return q;
}
```

Your Turn

Q: What's something you should be sure to include on your cheat sheet to help with a <u>ADTs</u> question?

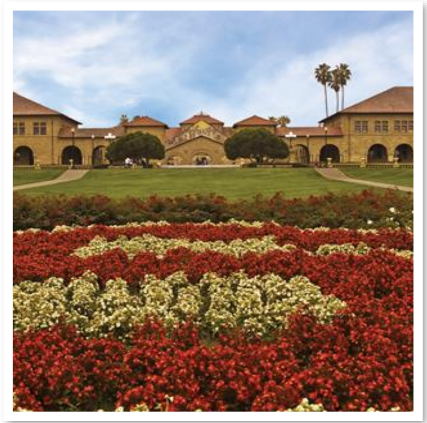
- **Give <u>one</u> idea per pollev response**
- Feel free to give multiple responses

pollev.com/cs106b

Stanford University

Sample Problem 3: Big O

What to expect, solution



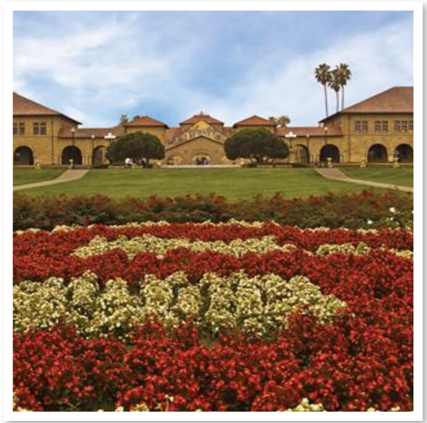
What to expect

- ADT operations with known costs, inside loops and recursive function calls
  - Your job is mainly to analyze the loop or recursion structure, then you can just plug in what you see for that ADT operation, using the Reference Sheet
  - Loops one after the other: additive cost
  - Loops nested in one another: multiplicative cost
  - Remember to simplify! (O(N), not O(3N + 5))

- (no additional samples for this one—refer to practice exams)

Practice Exam Problem 4: Recursion

What to expect, solution



What to expect

- Similar to pre-backtracking recursion examples from lecture: Fibonacci, Coin Flip, Fractals, permutations

Sample Q4: Roller Coaster seating

- This roller coaster at the Santa Cruz Beach Boardwalk is a train of rows that are two seats wide.
- There are N amusement park guests in line for the roller coaster, and they must board in order. The train is loaded from front to back.
- When guests reach the front of the line, they tell the ride attendant whether they want to ride alone (1 person that row) or share with the next person in line (2 people in that row).
  - Assume that the train has at least N rows, so it is long enough to accommodate all N riders in any configuration (rows aren't skipped except there are possibly extra empty rows after all riders have been seated).
- **How many different rider configurations are possible?**

Sample Q1: Strategy

- N is the number of riders guests, and recursively will track how many remain to be seated
- Base cases: 0 guests, 1 guest, 2 guests
- Recursive case: Each recursive call we either seat 1 or 2 guests (decreasing the remaining guests by 1 or 2). For each choice, recursively explore how many options from there

Sample Q1: Solution

```
int countConfigs(int n) {
    if (n < 3) {
        return n;
    }
    return countConfigs(n - 1) + countConfigs(n - 2);
}
```