Programming Abstractions

CS106B

Cynthia Bailey Lee
Julie Zelenski

Topics:

- **Midterm Review**
  - Overview of what to expect
  - Going over solutions to the practice exam
  - Q&A

- *Additional resources:*
  - For more of a general topics review (as opposed to practice exam solutions review), see the SL-led review session from Sunday night! Video and slides available, see pinned Ed post.

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.
  - I'd plan **20 minutes per problem**, which leaves you 20 minutes at the end to re-check your work etc.
    - Like the practice exam, there are 5 questions, so 120/5 = 24 min/q.

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: actually having you write test cases in our STUDENT_TEST format is also fair game for the exam.)

The Midterm Exam

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

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

Often includes ADTs (use reference sheet to know what methods "cost")

Often includes ADTs, but *be careful!* If you are given instructions like, "do not use auxiliary data structures" or "do not store the results, only count them," then you should not add ADTs such as Vector in your recursive helper function design.

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!
    - What is/isn't on there, how it's organized
    - Make sure you know how to read it, what it means
    - Don't waste real estate on your cheat sheet with anything that's on the reference sheet
  - **Cheat Sheet (your own to bring with you)**
    - One sheet of paper (8.5x11), both sides, could be printed or handwritten
    - **Studies show students who make a high-quality note sheet not only score better on that one test, but have deeper learning and long-term retention!** The process of reviewing all the course material and thoughtfully prioritizing/sifting key points is GREAT for learning!
    - Keep it organized! Pay attention to layout. You might even do things like highlighting in different colors, etc.

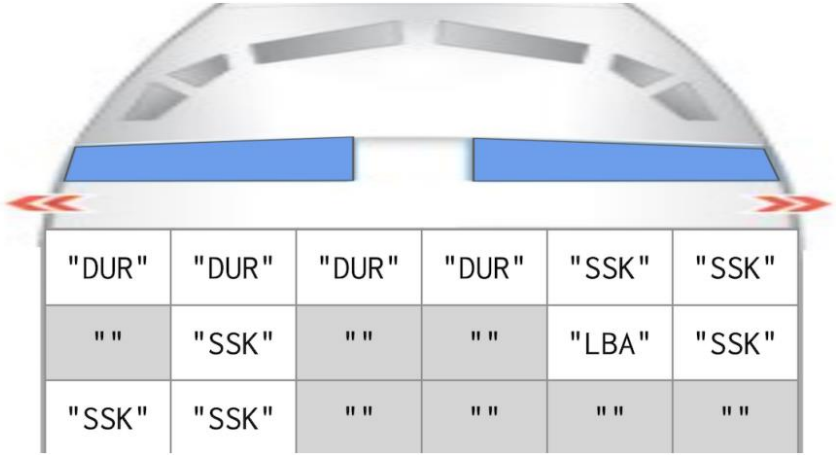
Practice Exam Problem 1: C++ Fundamentals

What to expect, solution

What to expect

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

Practice Exam Q1



"DUR"	"DUR"	"DUR"	"DUR"	"SSK"	"SSK"
""	"SSK"	""	""	"LBA"	"SSK"
"SSK"	"SSK"	""	""	""	""

A *free block* is a sequence of adjacent empty seats within a row. In the diagram above:

- row 0 has no free block
- row 1 has two free blocks, a block of size 1 starting at location **`{1, 0}`** and a block of size 2 starting at location **`{1, 2}`**
- row 2 has a free block of size 4 starting at location **`{2, 2}`**

Write the function **`findFreeBlock`** that searches **`seatGrid`** for a free block of at least size **`k`**.

Practice Exam 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

Practice Exam Q1: Solution

```
bool findFreeBlock(Grid<string>& seatGrid, int k, GridLocation& loc) {
    for (int r = 0; r < seatGrid.numRows(); r++) {
        int count = 0; // count of num consecutive empty seats in current row
        for (int c = 0; c < seatGrid.numCols(); c++) {
            if (seatGrid[r][c].empty()) {
                count++;
                if (count == k) {
                    loc = { r, c - k + 1 };
                    return true;
                }
            } else {
                count = 0;
            }
        }
    }
    return false;
}
```

Could also do: `seatGrid[r][c] == ""`

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

Practice Exam Problem 2: ADTs

What to expect, solution

What to expect

- One kind of ADT inside another kind of ADT
- 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: `for (string s : themap)`
  - Push/pop on a Stack vs manual equivalent on a Vector
  - Set operations vs manual equivalent
- *Theme*: really let the ADTs "shine"

Practice Exam Q2: Solution

```
int reseatGroup(Grid<string>& seatGrid, Map<string, Set<GridLocation>>& reservation[
    // Start by "unlocking" the seat assignments from existing reservation
    Set<GridLocation> oldSeats = reservationDB[groupCode];
    int k = oldSeats.size();
    for (GridLocation seat : oldSeats) {
        seatGrid[seat] = "";
    }

    GridLocation loc;
    if (findFreeBlock(seatGrid, k, loc)) {
        // Update reservationDB with new block of seats
        reservationDB[groupCode] = getSeatsForBlock(loc, k);
    }

    // mark seat assignments (will restore old or set new)
    for (GridLocation seat : reservationDB[groupCode]) {
        seatGrid[seat] = groupCode;
    }

    // Take a set difference between seatsNow and oldSeats.
    // Elements that are in both will be removed.
    Set<GridLocation> changedSeats = oldSeats.difference(reservationDB[groupCode]);
    return changedSeats.size();
}
```

Note cool use of Set operation

Your Turn

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

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

Practice Exam Problem 3: Big O

What to expect, solution

What to expect

- ADT operations with known costs, inside loops and other structures

Strategy

- Your job is mainly to analyze the loop/structure, then you can just plug in what you see for that ADT operation, using the Reference Sheet

Practice Exam Q3: Part 1 Solution

- We have a loop over a vector: N steps
  - Inside the loop, we have a fetch of vec[i] (O(1))
  - Inside the loop, we have a remove (O(N))
- N * N = O(N^2)

```
void expungeVector(Vector<int>& vec) {
   if (!vec.isEmpty()) {
      int first = vec[0];
      for (int i = 0; i < vec.size(); i++) { // changed line 13
         int cur = vec[i];
         if (cur == first) vec.remove(i);
      }
   }
}
```

Practice Exam Problem 4 and 5: Recursion

What to expect, solution

What to expect

- Similar to homework and lecture examples
  - But be careful—the same scenario may have very different code based on the question being asked (e.g., do you need to gather all the vote tallies, or just count them)
- You'll need to use the function we provide as a wrapper, and make your own recursive helper
  - Always fine to do this, but pay attention to rules like "no auxiliary storage" etc.
- For backtracking, use the template from lecture!!

Backtracking template

bool backtrackingRecursiveFunction*(args)* {

- Base case test for success: return true
- Base case test for failure: return false
- Loop over several options for "what to do next":
  1. Tentatively "**choose**" one option
  2. if ("**explore**" with recursive call returns true) return true
  3. else that tentative idea didn't work, so "**un-choose**" that option,
     but don't return false yet!--let the loop explore the other options before giving up!
- None of the options we tried in the loop worked, so return false

}

Remember this "if" structure—
don't return false here!

Your Turn

Q: What's something you should be sure to include on your cheat sheet to help with a Recursion or Backtracking question?

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