

CSCI-C 212 Midterm Exam Fall 2025 (80 points)  
Oct 8, 2025

### **C212 Midterm Exam Rubric**

1. (20 points) You're writing a method that computes the score of a two-part exam that you have written. The exam contains a multiple-choice section and an essay section. Here's what you should do:

Design the `double scoreExam(int c, int t, String e, double a, boolean g)` method that receives the number of multiple-choice questions the student got correct  $c$ , the number of multiple-choice questions on the exam  $t$ , the essay response  $e$ , the class average as a ratio  $a$  (i.e.,  $0 \leq a \leq 1$ ), and whether the student is a graduate student  $g$ . It returns a student's grade on the exam, as a ratio, based on the following criteria:

- (i) The student's raw score  $r$  is the ratio of the number of questions they got correct  $c$  and the total number of questions  $t$ . You may assume  $c \leq t$ .
- (ii) If the `essay`  $e$  either starts or ends with `"chicken nugget"`, award 0.1 extra points to  $r$ .
- (iii) If the `essay`  $e$  does *not* contain `"forty-two"` anywhere, then subtract 0.2 points from  $r$ .
- (iv) If the score, after applying the preceding rules, is above or equal to the class average  $a$ , then add 0.05 to  $r$ . For example, if their current score is  $r = 8/10 = .80$  and the class average is 0.42, then their score becomes 0.85.
- (v) If the student is a graduate student  $g$ , then if their score after applying the above rules is less than a 70%, they automatically receive a 0.
- (vi) The score is clamped to between 0 and 1, inclusive.

In designing this method, follow the design recipe from class: write the signature, purpose statement, testing, and *then* do the implementation. You should probably use simple numbers for the inputs so you can calculate the values in your head. You may assume that all inputs are well-formed.

For testing, write TWO different tests that test TWO distinct conditions.

**The skeleton code is on the next page. You must fill in the Java documentation comment, the tests, and the method to receive full credit.**

*Rubric:*

- (1 pt) Two tests exist that at least call the method. Each is worth 0.5 point.
- (4 pts) Two correct tests exist. Each is worth 2 points.
- (2 pts) The purpose statement for `scoreExam` exists and is correct/coherent.
- (3 pts) Each annotation is populated and is correct. Each annotation is worth 0.5 point.
- (2 pts) The ratio  $r$  is correctly computed. This point is not earned if *only* integer division is used.
- (2 pts) The check and consequent for  $e$  starting or ending with "chicken nugget" is correct. This is all-or-nothing.
- (1 pt) The check and consequent for  $e$  not containing "forty-two" is correct. This is all or nothing.
- (1 pt) The check and consequent for  $r \leq a$  is correct. This is all or nothing.
- (2 pts) The check and consequent for being a graduate student and  $r < 0.7$  is correct. This is all or nothing.
- (2 pts) The score is clamped from above and below (1 point each). If they forget the cast on `Math.max` or `Math.min`, it's fine.

```
/* Tests and javadocs are omitted for brevity. */

class ScoreExam {

    static double scoreExam(int c, int t, String e, double a, boolean g) {
        double r = (double) c / t;
        if (e.startsWith("chicken-nugget") || e.endsWith("chicken-nugget")) {
            r += 0.1;
        }
        if (!e.contains("forty-two")) { r -= 0.2; }
        if (r <= a) { r += 0.05; }
        if (g && r < 0.7) {
            return 0;
        } else {
            return (int) Math.max(0, Math.min(r, 1));
        }
    }
}
```

2. (30 points) This question has three parts, with each part weighted equally.

- (a) Design the *standard recursive* `String construct(String A, String B, int[] l)` method that, when given two strings  $A$  and  $B$  and an array of integers  $l$ , returns a new string that is the concatenation the characters from alternating between  $A$  and  $B$ , but selected by the elements of  $l$ .

For example, consider  $A = \text{"hi"}$  and  $B = \text{"hey"}$  and  $l = [0, 2, 1, 1, 0]$ . The returned string would be `"hyieh"`, because we start from  $A$  and get the character at index 0, which is `'h'`. Then we move to  $B$  and get the character at index 2, which is `'y'`. We then move back to  $A$  and get the character at index 1, which is `'i'`. We then move back to  $B$  and get the character at index 1, which is `'e'`. We then finally move back to  $A$  again and get the character at index 0, which is `'h'`.

You may assume that all of the entries of  $l$  are valid indices into the strings and that  $|l| = |A| + |B|$ . (That is, the length of  $l$  is equal to the sum of the lengths of  $A$  and  $B$ .) Moreover, you may assume that each character is used exactly once. (This property doesn't matter for the context of the problem, but maybe it'll help you reason through the question.)

*Hint: design a standard recursive helper method!*

*Rubric:*

- (1 pt) Helper method is private.
- (3 pts) Base case condition is correct.
- (3 pts) Branch for selecting character from A is correct.
- (3 pts) Branch for selecting character from B is correct.

**Note: if the method is not standard recursive, no points are awarded.**

```
static String construct(String A, String B, int[] l) {
    return constructHelper(A, B, l, 0);
}

private static String construct(String A, String B, int[] l,
                                int i) {
    if (i >= l.length) {
        return acc;
    } else {
        if (i % 2 == 0) {
            return A.charAt(l[i]) + construct(A, B, l, i + 1);
        } else {
            return B.charAt(l[i]) + construct(A, B, l, i + 1);
        }
    }
}
```

- (b) Design the `String constructTR(String A, String B, int[] l)` method that uses tail recursion to solve the problem. You will need to design a helper method. Remember to include the relevant access modifiers!

*Rubric:*

- (1 pt) Driver method correctly initializes all variables.
- (1 pt) Helper method is private.
- (2 pts) Base case condition is correct.
- (3 pts) Branch for selecting character from A is correct.
- (3 pts) Branch for selecting character from B is correct.

**Note: if the method is not tail recursive, no points are awarded.**

```
static String constructTR(String A, String B, int[] l) {
    return constructTRHelper(A, B, l, 0, "");
}

private static String constructTR(String A, String B, int[] l,
                                   int i, String acc) {
    if (i >= l.length) {
        return acc;
    } else {
        if (i % 2 == 0) {
            return constructTR(A, B, l, i + 1, acc + A.charAt(l[i]));
        } else {
            return constructTR(A, B, l, i + 1, acc + B.charAt(l[i]));
        }
    }
}
```

- (c) Finally, design the `String constructLoop(String A, String B, int[] l)` method that uses a loop to solve the problem.

*Rubric:*

- (2 pts) Local variable(s) are declared correctly for accumulating the string.
- (3 pts) Loop condition is correct.
- (2 pts) Branch for selecting character from A is correct.
- (2 pts) Branch for selecting character from B is correct.
- (1 pt) Correct value is returned.

**Note: if the method is recursive at all, no points are awarded.**

```
static String constructLoop(String A, String B, int[] l) {  
    int i = 0;  
    String acc = "";  
    while (!(i >= l.length)) {  
        if (i % 2 == 0) {  
            acc = acc + A.charAt(l[i]);  
            i = i + 1;  
        } else {  
            acc = acc + B.charAt(l[i]);  
            i = i + 1;  
        }  
    }  
}
```

3. (30 points) Design the static `<V> Set<V> process(List<String> l, Map<String, V> M)` method that, when given a list of strings  $l$  and a map of strings to values of type  $V$  called  $M$ , returns an insertion-ordered set of the values from  $M$  via the following property: For every string  $str$  in  $l$ , if there exists a key in  $M$  that starts with  $str$  or  $str$  is lexicographically less than some key in  $M$ , retrieve its value and add it to a set  $S$ .

Let's see an example. Consider the following inputs to `process`:  $l = ["hi", "howdy", "yo"]$  and  $M = \langle "hi \text{ there}" : 42, "what's up" : 5 \rangle$ . For every string  $str$  in  $l$ , we check to see if there's a key in  $M$  that either starts with  $str$  or  $str$  is lexicographically less than it. The first string `"hi there"` starts with `"hi"`, so we return its value of 42 and add it to a set. The second string `"howdy"` is lexicographically less than `"what's up"`, so we return its value of 5 and add it to the set. The third string `"yo"` is not the start of any key nor is it lexicographically less than any key, so nothing happens. The returned set is  $\{42, 5\}$ .

If there are multiple keys that satisfy some string in  $l$ , then it does not matter which is used.<sup>1</sup>

**The skeleton code is below. You do not need to write tests, but doing so may help you in your design.**

*Rubric:*

- Each blank is worth 3 points. If the created map is *not* a `LinkedHashMap`, award 0 points.

```
static <V> Set<V> process(List<String> l, Map<String, V> M) {
    Set<V> resSet = new LinkedHashSet<>();
    Set<String> keys = M.keySet();
    for (String s : l) {
        for (String keyInSet : keys) {
            if (keyInSet.startsWith(s) || s.compareTo(keyInSet) < 0) {
                resSet.add(keyInSet);
            }
        }
    }
    return resSet;
}
```

---

<sup>1</sup>If you want, assume that  $M$  is instantiated as a `LinkedHashMap`.

Scratch work