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

PLEASE READ ALL DIRECTIONS BEFORE STARTING YOUR EXAM. DO NOT OPEN UNTIL YOU ARE TOLD TO DO SO.

This is a closed-note exam aside from your aid sheet. You may not use any electronic devices to complete this exam, nor can you communicate with anyone besides the proctors and professor. *If you are caught cheating, you will receive an F in the course.*

For any question, unless specified otherwise, you may use any class without a corresponding `import`. E.g., if you want to use `HashMap`, you do not need to also import `java.util.HashMap`.

Unless otherwise stated, you do not need to spell out the "full design recipe," i.e., write out documentation comments and tests. Of course, doing so may aid you in creating your solution.

If you find a mistake, please raise your hand and let one of the proctors know; we will determine whether or not this is the case.

The exam has 80 total points. Answer all three problems for full credit.

When you are finished, turn in your exam and aid sheet if you have one, then quietly exit.

You have 75 minutes to complete the exam.

Good luck!

Question	Points	Score
1	20	
2	30	
3	30	
Total:	80	

Name: _____

IU Email: _____

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.

```
class ScoreExamTester {

  @Test
  void testScoreExam() {
    assertEquals(_______,
      scoreExam(_____, _____, ___________________________________, _______, _______));
    assertEquals(_______,
      scoreExam(_____, _____, ___________________________________, _______, _______));
  }
}

class ScoreExam {

  /**
   *
   * @param c
   * @param t
   * @param e
   * @param a
   * @param g
   * @return
   */
  static double scoreExam(int c, int t, String e, double a, boolean g) {









  }
}
```

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 $A =$ `"hi"` and $B =$ `"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!

(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!

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

Before proceeding...

For the last question, did you use the translation pipeline to solve the problem? It was not required; I am just curious!

_____ Yes

_____ No

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 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.[1]

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

```
static <V> Set<V> process(List<String> l, Map<String, V> M) {
  Set<V> S = ______________________________________;
  Set<String> keys = M._________________;

  for (__________ str : _________) {
    for (String keyInSet : keys) {
      if (_______________________________.startsWith(_______________)
        || str._____________(_____________________) < 0) {
        S._________(_____________________);
      }
    }
  }

  return ________________________;
}
```

[1] If you want, assume that M is instantiated as a `LinkedHashMap`.

Scratch work