CSE 331 Spring 2021 <u>Example</u> Final Exam

Name _____Solution_____

The exam should only take about 1 hour.

Score: _____ / 60

1. _____ / 12

2. _____ / 10

3. _____ / 10

4. _____ / 10

5. _____ / 10

6. _____ / 8

Problem 1 (Reasoning)

Fill in an implementation of the method `runLengthEncode` on the **next page**. It takes as input a string, `str`, an array of characters, `chars`, and an array of ints, `lens`. You can assume the string and both arrays are of length at least `n`. You can assume that `str` is non-empty and that it does not contain the character '\0'.

Your method will **write its output** into the arrays `chars` and `lens`, and it should **return** a number t such that (after returning) str = chars[0] * lens[0] + ... + chars[t-1] * lens[t-1], where a char * int means a string containing that many copies of the char. For example, if str = "aaabbccccaaddd", it would return t = 5 and leave chars[0..4] = [a, b, c, a, d] and lens[0..4] = [3, 2, 4, 2, 3].

The invariant for the loop is already provided. ***Do not add any additional loops.***

You do not need to *turn in* a complete proof of correctness, but you should complete one since your code will be graded on correctness.

(Continued on the next page...)

{{ P: 0 < n <= str.length chars.length, lens.length }}

```
int runLengthEncode(String str, int n, char[] chars, int[] lens) {

    int i = 0;
    int j = -1;
    char cur = '\0';
```

{{ Inv: P and str[0..i-1] = chars[0] * lens[0] + … + chars[j] * lens[j] and
        chars[0] != chars[1], ..., chars[j-1] != chars[j] and
        (i = 0 or cur = str[i-1]) }}

```
    while (i != n) {
        if (str.charAt(i) == chr) {
            lens[j] = lens[j] + 1;
        } else {
            j = j + 1
            cur = str.charAt(i);
            chars[j] = cur;
            lens[j] = 1;
        }
        i = i + 1;



    }
```

{{ str[0..n-1] = chars[0] * lens[0] + … + chars[j] * lens[j] and
    chars[0] != chars[1], ..., chars[j-1] != chars[j]}}

```
    return j+1;
}
```

Problem 2 (Testing)

Describe three test cases for the `runLengthEncode` method on the previous pages. The three tests should fall into different subdomains, i.e., they should be from subsets of the input where the expected or actual behavior is fundamentally different.

1. **Input**:   str = _____"abc"_____ and n = _____3_____

   Output: returns _____3_____

   `chars` starts with _____ ['a', 'b', 'c'] _____

   `lens` starts with _____ [1, 1, 1] _____

2. **Input**:   str = _____ "aabbbcc" _____ and n = _____6_____

   Output: returns _____3_____

   `chars` starts with _____ ['a', 'b', 'c'] _____

   `lens` starts with _____ [2, 3, 2] _____

   If it's not obvious, why is this testing a different behavior[1] from the case above?

   Testing cases where characters are repeated

3. **Input**:   str = _____"abc"_____ and n = _____2_____

   Output: returns _____2_____

   `chars` starts with _____ ['a', 'b'] _____

   `lens` starts with _____ [1, 1] _____

   If it's not obvious, why is this testing a different behavior[1] from the cases above?

   Testing cases where not all characters in str should be examined.

---

[1] You can define behavior, e.g., in terms of expected (black box) or actual (clear box) execution equivalence using either implementation of `runLengthEncode`.

Problem 3 (ADTs)

Suppose that we created a CharList ADT whose abstract value is a string but whose concrete representation was the run-length encoding used in the previous problems:

```
/** Represents an immutable sequence of characters like "abc" or "". */
class CharList {

  private char[] chars;
  private int[] lens;
  private int count;  // number of entries used in above arrays
...
```

(Note: `count` corresponds to the return value of runLengthEncode.)

What would the representation invariant[2] be for this ADT?

> chars != null and lens != null and 0 <= count <= chars.length, lens.length and
> chars[i-1] != chars[i] for i in 1 .. count-1

What would the abstraction function[2] be for this ADT?

> chars[0] * lens[0] + chars[1] * lens[1] + … + chars[count-1] * lens[count-1]

Fill in the implementation of the following method:

```
public void checkRep() {
  assert chars != null;
  assert lens != null;
  assert 0 <= count;
  assert count <= chars.length;
  assert count <= lens.length;


}
```

---

[2] While `CharList` uses the same representation as the `runLengthEncode` methods from before, you cannot use those methods to define your RI or AF here. You should define both directly in terms of the fields, as usual.

Problem 4 (Reasoning II)

Fill in the implementation of the following method:

```java
/** Returns the abstract value as a string. */
public String toString() {

  StringBuilder buf = new StringBuilder();
  for (int i = 0; i < count; i++) {
    for (int j = 0; j < lens[i]; j++) {
      buf.append(chars[i]);
    }
  }
  return buf.toString();



}
```

Fill in the implementation of the following method (include the loop invariant):

```java
/** @return Length of the list. */
public int size() {

  int s = 0;
  // s = length of chars[0]*lens[0] + … + chars[i-1]*lens[i-1]
  for (int i = 0; i < count; i++) {
    s += lens[i];
  }
  return s;



}
```

Problem 5 (Testing II)

Describe three test cases for the `CharList` ADT defined on the previous pages. Each case should be described by specifying the state of the fields of `CharList`. The three tests should fall into different subdomains, i.e., they should be from subsets of the input where the expected or actual behavior is fundamentally different.

1. **Setup**:   chars = _____ ['a', 'b', 'c'] _____
       lens   = _____ [1, 1, 1] _____
       count = __3__

   Outputs: toString() returns ____"abc"_____
       size() returns   _____3_____

2. **Setup**:   chars = _____ ['a', 'c'] _____
       lens   = _____ [2, 1] _____
       count = __2__

   Outputs: toString() returns _____"aac"_____
       size() returns   _____3_____

   If it's not obvious, why is this testing a different behavior from the case above?

   Tests a case where characters are repeated.

3. **Setup**:   chars = _____ [] _____
       lens   = _____ [] _____
       count = __0__

   Outputs: toString() returns _____""_____
       size() returns   _____0_____

   If it's not obvious, why is this testing a different behavior from the case above?

   Test the case where the string is empty.

Problem 6 (Miscellaneous)

a. Which is the best movie by the Cohen brothers (circle one)?

      The Big Lebowski   Fargo

      Burn After Reading   No Country for Old Men

b. Which is the most underrated movie by the Cohen brothers (circle one)?

      Burn After Reading   The Hudsucker Proxy

      O Brother Where Art Thou   The Ballad of Buster Scruggs

c. O Brother Where Art Thou is a retelling of which book?

      The Iliad   The Odyssey

      The Epic of Gilgamesh   Beowulf

d. The main characters of The Big Lebowski enjoy which of these the most?

      yoga   bowling

      debugging   singing

Of course, these questions are all fake. The actual test will include a half dozen or so multiple-choice / short answer questions on the topics from the second half of the course: equals & hashCode, exceptions, subtypes, generics, event-driven programs, and design patterns.