CSE 331 Fall 2017 Midterm Exam

Name _____

The exam is closed book and closed electronics. One page of notes is allowed.

Please **wait to turn the page** until everyone is told to begin.

Score: _____ / 62

1. _____ / 12
2. _____ / 12
3. _____ / 12
4. _____ / 10
5. _____ / 8
6. _____ / 8

Problem 1 (Specifications)

a. Alice is writing a function to find the minimum of an array of numbers. She intends to implement it by sorting the array, but she does not want clients to depend on that fact. Write a specification for her function:

```
/**
 * @requires vals != null
 * @modifies vals
 * @effects
 * @returns minimum number in vals
 * @throws
 */
int findMin(int[] vals) { ...
```

b. Suppose that Alice decides to change her implementation to no longer sort the array. How should she change the specification above?

Remove vals from @modifies.

c. This new specification would be (circle one):

weaker incomparable stronger

d. Suppose that Alice decides instead to stick with the version that sorts the array but will now allow clients to depend on that behavior. How should she change the specification above?

Add "vals is sorted" to @effects.

e. This new specification would be (circle one):

weaker incomparable stronger

Problem 2 (Reasoning)

Fill in an implementation of the method `runLengthEncode`. 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.

{{ 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) == cur) {
      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 3 (Reasoning II)

Fill in another implementation for `runLengthEncode` below.

For this version, you will use nested loops. The invariants for both loops are 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.

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

```
int runLengthEncode(String str, int n, char[] chars, int[] lens) {
  int i = -1;
  int j = -1;

  {{ Inv: P and str[0..i] = chars[0] * lens[0] + … + chars[j] * lens[j] and
        chars[0] != chars[1], ..., chars[j-1] != chars[j] and
        (i < 0 or n <= i+1 or str[i] != str[i+1]) }}
  while (i != n-1) {
    int k = i;

    {{ Inv: Inv and str[i+1] = str[i+2] = ... = str[k+1] }}
    while (k+1 != n-1 && str.charAt(i+1) == str.charAt(k+2)) {
      k = k + 1;
    }

    j = j + 1;
    chars[j] = str.charAt(i+1);
    lens[j] = k + 1 - i;
    i = k + 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 4 (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 = _____7_____

   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?

   The method should return a number smaller than str's length.

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?

   The method should ignore the last character of the input.

[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 5 (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

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

chars[0] * lens[0] + ... + chars[count-1] * lens[count-1]

Fill in the implementation of the following method:

```
@Override // (returns the abstract value, which is 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();
}
```

[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 6 (Miscellaneous)

a. Suppose that Alice has written a method and has a postcondition that correctly describes what she wants it to do but does not yet have a precondition. How could she come up with one that is certain to make her code correct?

Use backward reasoning

b. When is it safe to mutate an object being used as a key in a HashMap?

i. when the key and value are the same object

ii. when the associated value is immutable

iii. never

c. Which of the following types of operations are **NOT** usually included in the results of a requirements analysis? (Circle one.)

i. operations explicitly mentioned in relevant use cases

ii. operations we can infer will be necessary to complete use cases

iii. operations on which our chosen representation is most efficient

iv. operations clients would strongly expect to see based on conventions in the language or its standard libraries

d. Which of the following is the most important reason that it is difficult to write arbitrarily large software programs?

i. software tends to become out-of-date with hardware changes as it grows

ii. software tends to become too complex to understand as it grows

iii. compilers are too asymptotically slow to use on large amounts of code

iv. computers would not have enough memory to load the bytecode

e. Which of the following is a most often a symptom of writing poor quality code?

i. lots of time spent adding assertions

ii. lots of time spent testing

iii. lots of time spent debugging