

CSE 331 Fall 2017 Midterm Exam

Name _____

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

There is one bonus problem (part of problem 5). Only complete this **if you have time**. It will not be included in your score for the exam but will instead go into your bonus score. The latter will only be considered if your overall score for the course is close to the dividing line between grades.

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

Score: _____ / 58

1. _____ / 12

2. _____ / 12

3. _____ / 12

4. _____ / 10

5. _____ / 4

(Bonus: _____ / 4)

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:

```
/**
 *
 *
 *
 *
 *
 */
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?

- 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?

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

weaker

incomparable

stronger

Problem 2 (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) $str = chars[0] * lens[0] + \dots + chars[t-1] * lens[t-1]$, where a `char * int` means a string containing that many copies of the char. For example, if `str = "aaabbccccaadd"`, 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) {

    {{ 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 ( _____ ) {

    }

    {{ 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) {
```

```
    {{ 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 ( _____ ) {
```

```
        {{ Inv: Inv and str[i+1] = str[i+2] = ... = str[k+1] }}
```

```
        while ( _____ ) {
```

```
        }
```

```
    }
```

```
    {{ 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` = _____ and `n` = _____

Output: returns _____

`chars` starts with _____

`lens` starts with _____

2. **Input:** `str` = _____ and `n` = _____

Output: returns _____

`chars` starts with _____

`lens` starts with _____

If it's not obvious, why is this testing a different behavior¹ from the case above?

3. **Input:** `str` = _____ and `n` = _____

Output: returns _____

`chars` starts with _____

`lens` starts with _____

If it's not obvious, why is this testing a different behavior¹ from the cases above?

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 invariant² be for this ADT?

What would the abstraction function² be for this ADT?

Bonus (only if you have time): Fill in the implementation of the following method:

```
@Override // (returns the abstract value, which is a string)
public String 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?

- 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