CS 1110, LAB 14: LOOPS AND LOOP INVARIANTS

http://www.cs.cornell.edu/courses/cs1110/2018sp/labs/lab14/lab14.pdf

First Name: _____ **Last Name**: _____ **NetID**: _____

Getting Credit: **As always, strive to finish during the lab session** — it's the best way to stay on track in this course.[1]

Reference material

You may wish to consult the reading on worked examples regarding loops and invariants.

Exercise 1: Range notation

Ranges of Integers. What values are in the following ranges? Remember that in the notation `h..k`, we require `k` $\geq$ `h`-1. For example, `5..4` is OK, but `5..3` is not allowed.

The ranges in the right-hand column require an answer in terms of the variable `h`.

We've provided a few answers to help you.

Given Range	Contents	Given Range	Contents
`5..7`	`5, 6, 7`	`h..h+1`	`h, h+1`
`5..6`		`h..h`	
`5..5`		`h..h-1`	
`5..4`	Empty	`h-1..h`	
`4..4`	`4`	`h-1..h+1`	

Lab authors: D. Gries, L. Lee, S. Marschner, W. White

[1] But if you don't manage finish during lab, here are the alternate checkoff opportunities: (a) at ACCEL Green room consulting hours, listed at http://www.cs.cornell.edu/courses/cs1110/2018sp/about/staff.php , **from today until Tue May 8 inclusive**, (b) at non-professorial TA office hours **from today to Wed May 9 3:45pm inclusive**, although at TA office hours, questions about course material or assignments take precedence over lab check-offs; or (c) during the **first 10 minutes of your next scheduled lab (Tue May 8 or Wed May 9)**. Beyond that time, the staff have been instructed not to give you credit.
Labs are graded on effort, not correctness. We just want to see that you tried all the exercises, and to clarify any misunderstandings or questions you have.

Preserving Invariants. In each of (a)-(d) below, you are given an assertion labeled `P` and an assignment statement. Where indicated, write code so that if the assertion `P` is true initially, it is *also* true after your code and the assignment statement are executed. We have done the first one for you.

You may not use any function calls in your code (so, don't just call min()).

In all of these exercises, `v` is a list of `int`s.

(a)
```
# P: x is the sum of 1..n
#
    #  1                  n   n+1
    #  ---------------------------
    # | x is sum of these |       |
    # ---------------------------
    # STUDENTS: Put 1-3 lines here:

    x = x + (n+1)

    n = n + 1
    # x is (once again) the sum of 1..n
    #
    #  1                       n
    #  -------------------------
    # | now x is sum of these   |
    # ---------------------------
```

(b)
```
# P: x is the sum of h..100
    #
    #        h                 100
    #  ---------------------------
    # |    | x is sum of these    |
    # ---------------------------
    # STUDENTS: Put 1-3 lines here:

    h = h - 1
    # x is (once again) the sum of h..100
    #
    #  h                       100
    #  -------------------------
    # |  now x is sum of these   |
    # ---------------------------
```

(c) diagram now says "x is" instead of "v is"

```
(c) # P: x is the minimum of v[0..k-1]
    #
    #   0                  k-1   k
    #   ------------------------------
    # v | x is min of these    |      |
    #   ------------------------------
    # STUDENTS: Put 1-3 lines here:



    k = k + 1
    # x is (once again) the min of v[0..k-1]
    #
    #  0                        k-1  k
    #  ------------------------------
    # | now x is min of these       |
    # -------------------------------
```

(d)
```
# P: x is the minimum of v[h..100]
    #
    #         h                  100
    #    ---------------------------
    # v |    | x is min of these    |
    #    ---------------------------
    # STUDENTS: Put 1-3 lines here:



    h = h - 1
    # x is (again) the min of v[h..100]
    #
    #  h                       100
    #  -------------------------
    # |  now x is min of these   |
    # ---------------------------
```

Exercise 2: Functions With Loop Invariants

The following pages provides "stub" (skeletons) for several functions, all of which have the same specification, but are to be implemented based on different invariants. Complete **two of the three** skeletons with while-loops that constitute effective use of the given invariants, by writing in your code on this handout.

This is optional, but if you want to code up and test your solutions, or see test cases for the function to be implemented, get the Lab 14 files from the Labs section of the course web page, http://www.cs.cornell.edu/courses/cs1110/2018sp/labs .

```
def num_space_runs1(s):
    """Returns: The number of runs of spaces in the string s.

    Examples (we've put ^ markers to "underline" spaces):
            num_space_runs('  a  f   g    ') returns 4
                            ^^ ^^ ^^^ ^^^^

            num_space_runs('a  f   g') returns 2
                             ^^ ^^^

            num_space_runs('  a  bc   d') returns 3
                            ^^ ^^  ^^^
    Precondition: s is a nonempty string with letters and spaces"""

    # STUDENTS: The invariant for you to work with is:
    #     s[0..i-1] has n runs of spaces, AND:
    #     in_a_run is a boolean:
    #          True if i-1 is a valid index and s[i-1] is a space
    #          False otherwise
    #
    # In other words, s[i..len(s)-1] still needs to be checked;
    # and in_a_run tells us whether a new space would be part of an old run.


    # REPLACE THE FOLLOWING WITH CORRECT INITIALIZATION CODE:
    i = None
    n = None
    in_a_run = None

    # PUT YOUR WHILE LOOP HERE
    # Hint1: only increment n when you find a space and are not currently in a run.
    # Hint2: you need to change in_a_run when:
    #   (a) you have found a space and you are not currently in a run, or
    #   (b) you found a non-space and you currently in a run
    # Hint3: don't forget to increment your loop variable, if you have one!









    # post: s[0..len(s)-1] contains n runs of spaces
    # PUT THE RETURN STATEMENT HERE
```

```
def num_space_runs2(s):
    """Same spec as above"""

    # invariant: s[0..i] contains n runs of spaces. So if i+1 is a legal index,
    # s[i+1] is the next thing to check, or the unknowns are s[i+1..len(s)-1].


    # WE ARE GIVING YOU THE FOLLOWING INITIALIZATION. DON'T CHANGE IT.
    i = 0
    if s[0] == ' ': # this initialization "peeks" at the data to see
                    # whether s[0] starts a run or not.
        n = 1
    else:
        n = 0

    # PUT YOUR WHILE LOOP HERE.
    # Hint: you only need to increment n when you have a space following a
    # non-space.









    # post: s[0..len(s)-1] contains n runs of spaces

    # PUT THE RETURN STATEMENT HERE
```

```
def num_space_runs3(s):
    """Same spec as above"""

    # The invariant for you to work with is:
    #     s[0..i] has n runs of spaces
    #
    # In other words, s[i+1..len(s)-1] still needs to be checked.

    # WE ARE GIVING YOU THE FOLLOWING INITIALIZATION. DON'T CHANGE IT.
    i = -1
    n = 0

    # PUT YOUR WHILE LOOP HERE
    # Hint: you only need to increment n when i is a valid index and s[i] is
    # not a space but s[i+1] is a space,
    # OR when i is -1 and the very first character in s is a space









    # post: s[0..len(s)-1] contains n runs of spaces
    # PUT THE RETURN STATEMENT HERE
```