Name 1: _____ NetID 1: _____
Name 2: _____ NetID 2: _____
Name 3: _____ NetID 3: _____

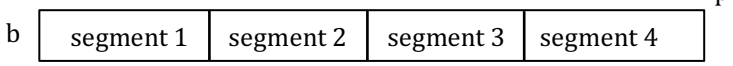
The purpose of this problem set is for you to get practice developing loop invariants and loops. This practice will help you understand later lectures that cover searching, sorting, and related algorithms that manipulate arrays. As with previous labs, you can work with other students. You can submit this lab with groups of up to 3 students.

This lab should be handed in on paper. Try to complete and hand this in by the end of the lab. If you don't finish during lab on Wednesday, you can bring your paper to Thursday's class or Professor VanHattum's office hours Thursday afternoon.

Part 1: Correctness and Hoare triples

Question 1. Write the formula for the number of values in the range b..c: _____

Question 2. Below are four array segments. To the right, write the number of values in each segment in terms of the relevant variables.



b[h..k] _____

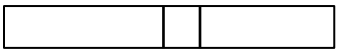
b[k+1..m] _____

b[m+1..n–1] _____

Question 3. State a formula (relating p and q) that makes segment a[p..q] empty: _____

Question 4. Below, draw an array diagram that represents this assertion:

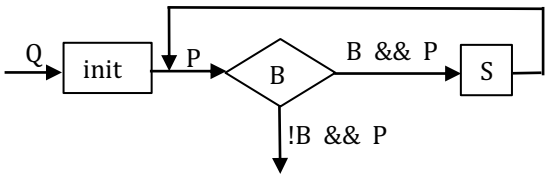
b[h..j-1] ≤ x && b[j] = x && b[j+1..k ≥ x]



Question 5. Write down the meaning of the Hoare triple {B} C {D} :

Part 2: Loop invariants

Question 6. The four loopy questions. Understanding and *knowing* the four loopy questions is key to understanding all this loopy stuff. Below, write the four loopy questions, giving (a) the general idea and (b) a precise statement of what that means. The annotated flow chart to the right should help you. We do the first loopy question for you. You can continue on the next page.



(1) Does it start right? Is {Q} init {P} true?

!B && P => R

In the exercises below, you do not need to be concerned with declaring variables. Assume they are all declared.

Question 7. Does it start right? Below are preconditions Q and loop invariants P. To the right of each pair, write initialization init so that {Q} init {P} is true. The initialization should store values in b and k. Do not store anything in array c. See these footnotes.[1]

7A. Q: true
P: b = all elements of c[0..k-1] are 0

7B. Q: true
P: b = all elements of c[k..c.length-1] are 0

Question 8. Does it stop right? Below are loop invariants P and postconditions R. To the right of each pair, write the loop condition B such that !B and P => R.

8A. P: b = all elements of c[0..k-1] are 0
R: b = all elements of c[0..c.length-1] are 0

8B. P: b = all elements of c[k..c.length-1] are 0
R: b = all elements of c[0..c.length-1] are 0

Question 9. Does the repetend do what it is supposed to do? Repetend S of the loop has to make progress toward termination and keep the invariant true, i.e. {B && P} S {P} must be true. Below, to the right of each question, write the loop body given B, P, and the expression that must be decreased to make progress toward termination.

9A. B: k < c.length

P: b = all elements of c[0..k-1] are 0

Repetend should increase k

9B. B: 0 < k

P: b = all elements of c[k..c.length-1] are 0

Repetend should decrease k

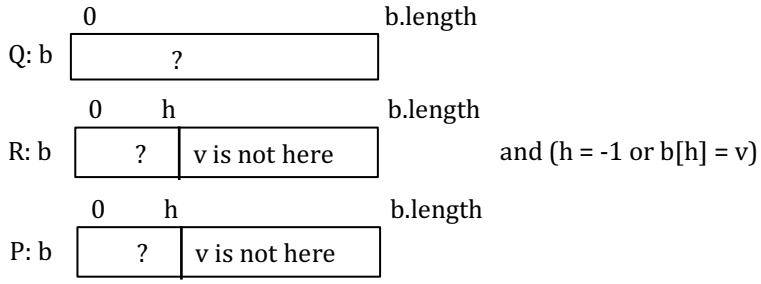
---

[1] {true} S {P} means that in any starting state, execution of S will terminate with P true.
b = "all elements of c[0..k-1] are 0" means: if all elements of c[0..k-1] are 0, b is true; otherwise it is false.
The sum of an empty set is 0; its product 1. This is because 0 + anything = anything and 1 * anything = anything.

Question 10. We want a loop (with initialization) that finds the last occurrence b[h] of a value v in array b, setting h to -1 if v is not in b. Below, we give the precondition Q, postcondition R, and loop invariant P.

Look at the postcondition. If h = -1, then the array diagram tells you that b[0..h] is empty and v is not in the array. P arises by deleting part of R—the assertion h = -1 or b[h] = v . Write the loop with initialization to the right of the array diagrams, using the four loopy questions. Hint: The loop body will be very simple.

Note: Initially, probably you should make b[h+1..b.length-1] empty.



Question 11. Generalizing array diagrams.

Note: the solution to this problem is described in JavaHyperText. Please attempt this problem with your group **before** looking up the solution. Once you have tried it, you can find the solution by searching JavaHyperText for "dutch" and selecting the corresponding PDF.

Below is a pair of assertions—given as array diagrams—for the Dutch National Flag algorithm.

In this algorithm, array b contains red, white, and blue balls. The desired postcondition is for the array to have red balls first, then white balls, then blue balls, which must be accomplished by switching any two balls (repeatedly).

That is, the idea is to swap array elements so that the red ones are first, then the white ones, and then the blue ones. There are four possible invariants that have 4 segments each; draw diagrams for 2 of these invariants. Add extra variables to mark the boundaries of two adjacent array segments if necessary. Then, read the JavaHyperText page for deciding between invariants.