The purpose of this problem set is for you to get practice developing loop invariants and loops. As long as this is a good attempt, with most things right, you get 100%. This practice will help you understand later lectures that cover searching, sorting, and related algorithms that manipulate arrays.

We hope you will work with other students, in groups of 1, 2, or 3, as in previous recitations.

Try to complete and submit this by the end of the recitation. Remember to form a group before submitting. All group members must do something to form a group: one invites and the other(s) accept.

If it helps you to draw a mathematical assertion as an array diagram, do it.

**1. 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.

!B && P

init

Q

S

B

P

B && P

!B && P => R

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

In the exercises below, don’t be concerned with declaring variables. Assume they are all declared.

**2. 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]](#footnote-1)

2A. Q: true

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

Init: (a). b = 0; k = 0;

(b). Because the range of c could be [0,-1], which is 0, equals to b.

2B. Q: true

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

Init: (a). b = 0; k = c.length();

(b). Because the range of c could be [c.length, c.length-1], which is 0, equals to b.

**3. 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.

3A. P: b = all elements of c[0..k-1] are 0

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

loop condition B: a). k <= c.length – 1

b). The first unstatisfactory is k=c.length, replace k-1 by k=c.length, R is true

3B. P: b = all elements of c[k..c.length-1] are 0

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

loop condition B: a). k > 0

b). The first unstatisfactory is k=0.length, replace k by k=0, R is true

**4. 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.

4A. B: k < c.length

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

a). Repetend should increase, k should increase

b). The increment of k could make progress to k = c.length

4B. B: 0 < k

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

a). Repetend should decrease, k should decrease

b). The decrement of k could make progress to k = 0

**5. Linear search.** 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.

?

Q: b

0 b.length

? v is not here

R: b

0 h b.length

and (h = -1 or b[h] = v)

? v is not here

P: b

0 h b.length

Answer:

**int h = -1;**

**int k = 0;**

**while ( k <= b.length – 1){**

**if ( b[k] == v){ h = k;}**

**k ++;**

**}**

**6. Generalizing array diagrams.** 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 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; draw diagrams for 2 of these invariants. Add extra variables to mark boundaries of two adjacent array segments if necessary.

reds whites blues

R: b

0 b.length

?

Q: b

0 b.length

1).

? reds whites blues

P: b

0 h g i b.length

2).

reds whites blues ?

P: b

0 h g i b.length

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. [↑](#footnote-ref-1)