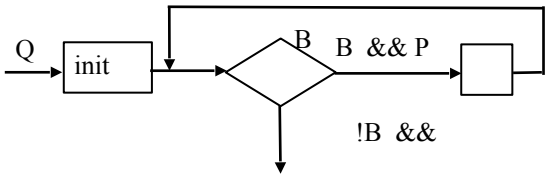
CS2110 Spring 2017 Recitation: Loop-invariant Problems

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 lecture that cover searching, sorting, and related algorithms that manipulate arrays.

It may help to do this with another person or two, together. If you do that, put both names and netids at the top. If you have a question, ask the TA or the people around you. Go ahead, discuss things with those around you. Some of these questions are mechanical, asking for a definition or something like that. Some ask you to write code or execute a method call. You can look at notes or a book, get on the internet and read the exception-handling webpage, watch the videos, google something, whatever.

1. The four loopy questions. Understanding and *knowing* the four loopy questions is key to understanding all this loop 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.



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

(2) Does it end right? Is P && !B => R true?

(3) Does the repetend S make progression towards the termination the loop?

(4) Does the repetend S maintain precondition P true? Is {P and B} S {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. See these footnotes.[1]

2A. Q: true

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

k=0; b = (c[0] == 0);

2B. Q: true

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

k=c.length()-1; b = (c[0] == 0);

2C. Q: true

P: s is the product of c[k+1..c.length-1]

k=c.length()-1; s = 1;

---

[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.

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 && 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

k<= c.length();

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

k != 0;

3C. P: s is the product of c[k+1..c.length-1]

R: s is the product of c[0..c.length-1]

k >= 0 ;

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, 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

Repetend should increase k

```
if (c[k] !=0){
    b=false;
}
k+=1;
```

4B. B: 0 < k

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

Repetend should decrease k

```
if (c[k-1] !=0){
    b=false;
}
k-=1;
```

4C. B: 0 <= k

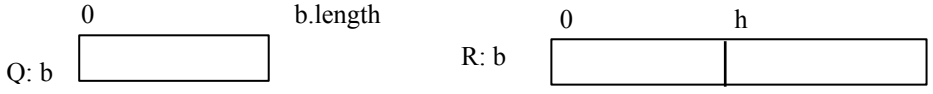
P: s is the product of c[k+1..c.length-1]

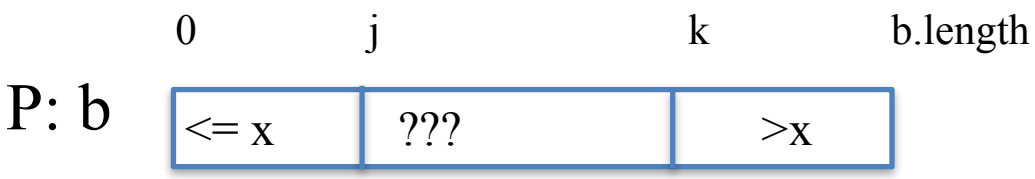
Repetend should decrease k

```
s=s*(c[k]);
k-=1;
```

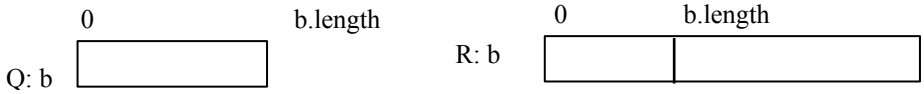
5. Generalizing array diagrams. Below are pairs of assertions given as array diagrams. For each pair, draw a diagram that generalizes both. Add extra variables to mark boundaries of two adjacent array segments if necessary

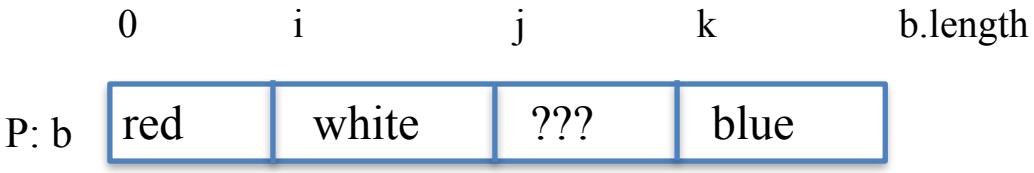
5A. This pair is used in an algorithm called *binary search*, which will be developed in lecture:



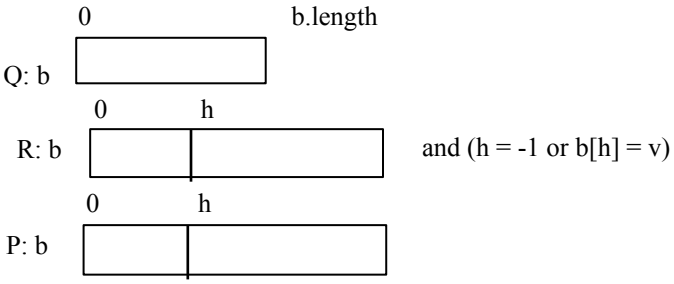


5B. Dutch National Flag. Array b contains red, white, and blue balls. The idea is to swap array elements so that the red ones are first, then the white ones, and then the blue ones. There are four possible invariants that have 4 segments; draw them all.





6. 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 to the right of the array diagrams, using the four loopy questions. Hint: The loop body will be very simple.

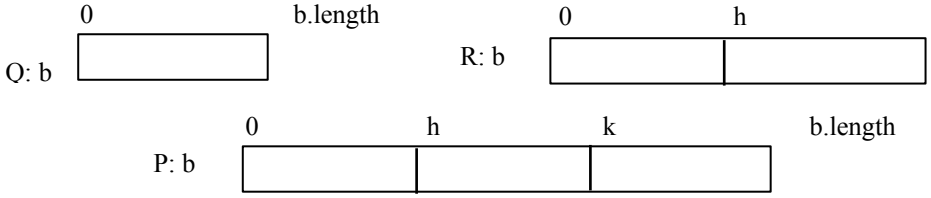



```
h = b.length()-1;
while ( h != v || h== -1){
        h- -;
}
```

7. Another linear search but with a different specification

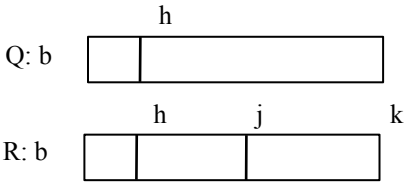
Assume array b is sorted. We want a loop with initialization that stores a value in h to truthify R below. The algorithm finds the rightmost occurrence of x in b if it is in the array. If it is not in, tells you where x belongs, right after b[h]. We also give a loop invariant. In all three diagrams, we omit the information that b is sorted simply to make them easier to see. But it is sorted and you can use that fact.

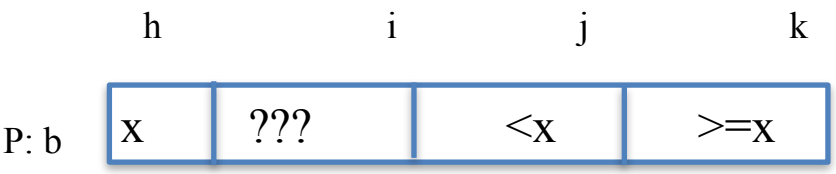
Below the diagrams, write the loop with initialization, using the four loopy questions. Have each iteration increase h by 1 or decrease k by one, but not both. Remember, this is a search, not a sort; don't change the array.




```
h=-1;
k=b.length();
while (h<k){
      if (b[h]<=x){
            h++;
      }
      if (b[k] > x) {
            k- -;
      }
}
```

8. Partition algorithm. b[h] contains a value x. x is not a variable and cannot be used in the program; it just denotes what is in b[h]. The algorithm is supposed to swap values of b[h+1..k] around until the postcondition is true, using a variable j (which has to be given an initial value). First, generalize Q and R to an invariant and then write the algorithm using the four loopy questions. Do NOT change h and k!






```
j=k; i=k;
while (i>h){
      if (b[h+1] >= x){swap b[h+1] with b[k]; j- -; i- -;}
      else{ swap b[h+1] with b[j]; i- -;}
}
```