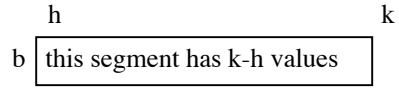
Recitation 4. Developing loops

Introduction. This recitation concerns developing loops using their invariants and bound functions. Your recitation instructor will develop the loops as presented in this handout. During the recitation, it's best that you concentrate on what the instructor is saying and participate in the loop developments without simply reading from this document. This document is meant for you to study at a later time.

Remember this about ranges! The number of values in a segment b[h..k-1] is k-h:



So, if you want the segment to have 0 values, set k to h (or h to k). You can discover the number of values in other segments, e.g. b[h..k], based on the above fact.

Memorize this loopy questions! To prove

// {Q}

init;

// {invariant: P; bound function: t}

while (B) S

// {R}

do the following.

Checklist

1. Prove that P is true initially: {Q} init {P}
2. Upon termination R holds: P and !B imply R
3. Each iteration makes progress toward termination.
4. P is invariant: {P and B} S {P}

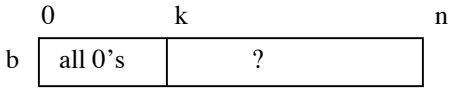
In the examples below, we use the checklist to **develop** loops. Four loops will be developed for the same problem, using different invariants, so that you begin to see how invariants are used.

Problem 1. Write a loop (with initialization) to set all elements of array segment b[0..n-1] to 0.

Precondition Q: n >= 0

Postcondition R: b[0..n-1] is all 0

Invariant P: b[0..k-1] is all 0, and 0 <= k <= n, i.e.



Initialization: To make invariant P true, use k= 0;. (Why does this work?)

Condition B: The loop can stop only when the ? segment of the invariant is empty, which happens when k=n. Therefore, the loop condition is k != n. Another way to see this is to look at P and see what extra information is needed to imply R. This extra information is k = n, so complement to get B.

Progress toward termination: Do k= k+1;

Keep invariant true: Before the increment of k, introduce the statement b[k]= 0;

The loop is: k= 0; **while** (k != n)

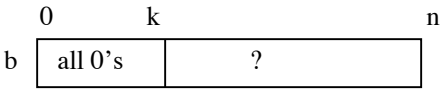
{b[k]= 0; k= k+1;}

Problem 2. Write a loop (with initialization) to set the elements of array segement b[0..n-1] to 0.

Precondition Q: n >= 0

Postcondition R: b[0..n-1] is all 0

Invariant P: b[0..k] is all 0, and -1 <= k < n, i.e.



The only difference between this and the previous problem is that k is used differently. Compare the two invariants (in their picture form)!

Initialization: To make invariant P true, use k= -1;

Condition B: The loop can stop only when the ? segment of the invariant is empty, which happens when k=n-1. Therefore, the loop condition is k != n-1. Another way to see this is to look at P and see what extra information is needed to imply R. This extra information is k = n-1, so complement to get B.

Progress toward termination: Do k= k+1;

Keep invariant true: Before the increment of k, introduce the statement b[k+1]= 0;. However, it is better to exchange the two statements in the body and use

The loop is: k= -1; **while** (k != n-1)

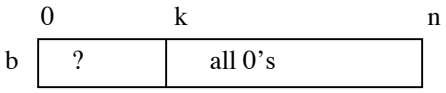
{k= k+1; b[k]= 0;}

Problem 3. Write a loop (with initialization) to set the elements of array segment b[0..n-1] to 0.

Precondition Q: n >= 0

Postcondition R: b[0..n-1] is all 0

Invariant P: b[k..n-1] is all 0, and 0 <= k <= n, i.e.



The only difference between this and the previous problem is that k is used differently. Compare the two invariants! Here, the invariant was obtained from postcondition R by replacing expression 0 by k.

Initialization: To make invariant P true, use k= n;

Condition B: The loop can stop only when the ? segment of the invariant is empty, which happens when k=0. Therefore, the loop condition is k != 0.

Progress toward termination: Do k= k-1;

Keep invariant true: Before the decrement of k, introduce the statement b[k-1]= 0;. However, it is better to exchange the two statements in the body and use

The loop is: k= n; **while** (k != 0)

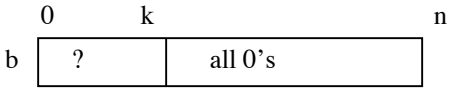
{k= k-1; b[k]= 0;}

Problem 4. Write a loop (with initialization) to set array b[0..n-1]

Precondition Q: n >= 0

Postcondition R: b[0..n-1] is all 0

Invariant P: b[k-1..n-1] is all 0, and -1 <= k < n, i.e.



The only difference between this and the previous problem is that k is used differently.

Initialization: To make invariant P true, use k= n-1;

Condition B: The loop can stop only when the ? segment of the invariant is empty, which happens when k=-1. Therefore, the loop condition is k != -1. Another way to see this is to look at P and see what extra information is needed to imply R. This extra information is k = -1, so complement to get B.

Progress toward termination: Do k= k-1;

Keep invariant true: Before the decrement of k, introduce the statement b[k]= 0;.

The loop is: k= n-1; **while** (k != -1)

{b[k]= 0; k= k-1;}

Summary. We developed four loops (with initialization) for the same problem. The solutions differed only in the meaning of variable k:

Problem 1. b[0..k-1] is all zeros

Problem 2. b[0..k] is all zeros

Problem 3. b[k..n-1] is all zeros

Problem 4. b[k-1..n-1] is all zeros

This exercise should alert you to the need to define variables precisely and to refer to those definitions repeatedly when reading or developing a program.

Problem 5. Produce a String that contains the decimal representation of nonnegative **int** n.

Consider an integer in decimal notation, like 5647. This number has the value

$5*10^3 + 6*10^2 + 4*10^1 + 7*10^0$.

A nonnegative integer n is written in decimal as a sequence of digits $d_k d_{k-1} \ldots d_1 d_0$ for some k, where:

(1) Each d_i satisfies $0 <= d_i < 10$

(2) d_k, the most significant digit, is > 0.

(3) $n = d_k*10^k + \ldots + d_1*10^1 + d_0*10^0$

Note that with this definition, the decimal representation for 0 is the empty String, "".

How do we calculate $d_k \ldots d_1 d_0$ as a sequence of characters? Consider n = 5647. We have:

$d_0 = 5647 \% 10 = 7.$

so we can get the last digit easily. Suppose we compute

$m = 5647 / 10 = 564.$

Now we can repeat the process using m, picking off first the 4, then the 6, then the 5.

The above expresses the idea. Lets now formalize.

Precondition: n >= 0, and in decimal, n is

$d_k d_{k-1} \ldots d_1 d_0$ for some k

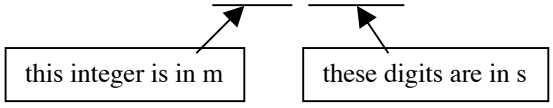
Postcondition: s = "$d_k d_{k-1} \ldots d_1 d_0$" is the decimal representation of n.

Invariant P: n = $m*10^k$ + (integer represented by s)
(for some k)

Here's another view of the invariant, which may be easier for you to understand. Suppose n, in decimal, is a series of digits, n = d d d d d d d d d d.

Here's the invariant:

Invariant P: n = d d ... d d … d d



Note that k is not needed in this little algorithm. It's a "thought" variable --in our heads but not in the program. The "for some k" alerts us to this fact.

Note: m will decrease with each iteration.

1. Initialization. We make the invariant true by setting s to "" and m to n.

2. Loop Condition. From the invariant, we see that when m = 0, n is the integer represented by s. that's the termination condition. So, the loop condition is m != 0.

3. Progress and maintaining the invariant. Progress is made by reducing m, but the invariant has to be maintained using the idea presented earlier. We calculate with $m*10^k$

$m*10^k$ (for some k)
= $((m/10)*10 + m\%10)*10^k$
= $(m/10)*10^{k+1} + (m\%10)*10^k$

Therefore we can manipulate P as follows:

n = $m*10^k$ + (integer represented by s)
= n = $(m/10)*10^{k+1} + (m\%10)*10^k$ +
(integer represented by s)

Thus, the digit m%10 has to be prepended to s and m has to be set to m/10. Here's the loop (with initialization):

```
int m= n; String s= "";
// inv: n = m*10**k + (int represented by s)
//              (for some k)
while (m != 0) {
   // Prepend next digit to s
      int d= m%10; // Note: 0 <= d <= 9
      s= (char)(d + '0') + s;
      m= m/10;
}
```

Problem 6. Find the quotient and remainder when x is divided by y (for y > 0 and x >= 0). We can't use division or multiplication.

Discussion. Here is the formula:

x = q*y + r where 0 <= r < y

We call q the quotient and y the remainder. For example, we know that dividing 14 by 4 yields 3 with a remainder of 2:

14 = 3*4 + 2, i.e.

14 = q*4 + r, where q = 3 and r = 2, 0 <= r < 4.

Precondition Q: y > 0 and x >= 0

Store in q and r to truthify R:

Postcondition R: x = q*y + r and 0 <= r < y

Invariant P: We can truthify x = q*y + r by setting q to 0 and r to x. This also truthifies 0 <= y. But it does not necessarily truthify r < y. So let's take as P the following:

P: x = q*y + r and 0 <= r

Thus, the only part of the postcondition that is not always true is r < y, so we take its negation as the condition. Thus far we have:

```
q= 0; r= x;
while (y <= r) { …}
```

To make progress, we have to reduce r. Subtracting 1 from it is a possibility, but it would be hard to maintain P. Instead, let's investigate P and loop condition y <= r:

x = q*y + r and 0 <= r and y <= r
= <We need to reduce r, and r-y is a good guess>
x = (q+1)*y + r-y and 0 <= r and 0 <= r-y

So adding 1 to q and subtracting y from r keeps the invariant true. The loop is then:

```
q= 0; r= x;
// inv P: x = q*y + r and 0 <= r
while (y <= r) {
   q= q+1; r= r-y;
}
```