Developing loops from invariants

CS1110 Final Review

Info

- My name:
- Feel free to ask questions at any time
- Slides will be posted online

Outline

- 4 questions for loop
- How to develop loops from invariants
- Common mistakes
- What's in the exam

Four loopy questions

1. How does it start (does the initialization make the invariant true?
2. When does it stop (the invariant together with falsity of the guard should imply the postcondition)?
3. Does the repetend make progress toward termination?
4. Does the repetend keep the invariant true?

Developing a for-loop

Develop a for-loop when you recognize that a range of integers a..b has to be processed.

The steps we propose for developing the loop allows it to be developed with a minimum of fuss and a maximum chance of success. The development of most of the loop is almost mechanical, allowing you to focus on the difficult, creative parts.

Developing a for-loop-(a)

Suppose the command you are trying to implement is

Process a..b

- Write the command as a postcondition:

post: a..b has been processed.

Developing loops-(b)

- write the loop:

```
for (int k= a; k <= b; k= k+1) {
    //Process k
}
// post: a..b has been processed.
```

Developing loops-(c)

- Fill in the invariant

```
// invariant: a..k-1 has been processed
for (int k= a; k <= b; k= k+1) {
    Process k
}
// post: a..b has been processed.
```

Developing loops-(d)

- Fix the initialization

```
init to make invariant true;
// invariant: a..k-1 has been processed
for (int k= a; k <= b; k= k+1) {
    Process k
}
// post: a..b has been processed.
```

Developing loops-(e)

- Figure out how to "Process k"

```
init to make invariant true;
// invariant: a..k-1 has been processed
for (int k= a; k <= b; k= k+1) {
    // Process k
   implementation of "Process k"
}
// post: a..b has been processed.
```

Range

- Pay attention to range:
  a..b or a+1..b or a…b-1 or …
- To process a range a..b-1, change the loop condition to k < b
- Note that a..a-1 denotes an empty range —no values in it

Spring'08-Prelim3 Question 3

- A magic square is a square where each **row and column adds up to the same number** (sometimes, one also includes the diagonals, but for this problem, we won't). For example, in the following 5-by-5 square, the elements in each row and column add up to 70:

```
18 25  2  9 16
24  6  8 15 17
 5  7 14 21 23
11 13 20 22  4
12 19 26  3 10
```

```
/** = "all the rows of square array sq sum to sum".
      Precondition: sq is not null and indeed is a square array. */
public static boolean areMagicRows(int[][] sq, int sum) {

    // invariant: each row 0..k-1 of sq sums to sum.
    for (int k= 0; k < sq.length; k = k + 1) {
        // Return false if row k does not sum to sum.
        int rowsum= 0;
        for (int j= 0; j < sq.length; j= j + 1) {
            rowsum= rowsum + sq[k][j];
        }
        if (rowsum != sum)
            return false;
    }
    // postcondition: each row 0..sq.length-1 sums to sum
    return true;
}
```

While-Loop

for (**int** k= a; k <= b; k= k+1) {…S…}

```
int k= a;
while (k <= b) {
    S;
    k= k+1;
}
```

You will not be asked to develop the invariant for a while-loop (unless it is one of the algorithms you must know). You may be asked to develop one given the invariant.

Another example

- Prelim 3 Question 4. Addition of 2 numbers

```
h=?
k=?
carry=?

//invariant: b[h..] contains the sum of c[h..] and d[k..],
//           except that the carry into position k-1 is in
'carry'
While (_________) {


}
//postcondition: b contains the sum of c and d
//          except that the carry contains the 0 or 1 that
//          belongs in the beginning
```

```
 0 1 0 0
   4 8 1
   9 2
 -------
   5 7 3
```

DOs and DON'Ts #1

- DO use variables given in the invariant.
- DON'T Use other variables.

```
//invariant: b[h..] contains the sum of c[h..] and d[k..],
//           except that the carry into position k-1 is in
'carry'
While (_________) {
    //use h, k, carry
    // you can declare variables here

}
```

DOs and DON'Ts #2

- DO understand invariants first
  - x = some function of b[A..k-1]
    - For (int k=A;….;k=k+1)
  - X= some function of b[k..b.length-1]
    - For (int k=b.length;…;k=k-1)

```
h=c.length
k=d.length
carry=0
//invariant: b[h..] contains the sum of c[h..] and d[k..],
//           except that the carry into position k-1 is in
'carry'
While (h>0)   //why is not >=0?
```

DOs and DON'Ts #3

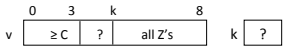
- DO double check corner cases!
  - for (x=0; x<b.length; x++)
    - What will happen when x=0 and x=b.length-1?
    - If you use x in array, e.g. a[x], will x always be between 0…a.length-1?

```
h=c.length
k=d.length
carry=0
//invariant: b[h..] contains the sum of c[h..] and d[k..],
//           except that the carry into position k-1 is in
'carry'
While (h>0) {
    h--; k--;  //directly using b[h] or c[h] or d[k] is wrong!
}
```

DOs and DON'Ts #4

- DON'T put the variable directly above the vertical line.

```
// Where is k?
```



What's in the exam

- Array diagrams, invariant from pre- and post-condition, initialization, body of loop, etc.
- We will NOT ask you to write a loop invariant for a while-loop EXCEPT:

  Linear search, binary search, dutch national flag, partition algorithm of quicksort, insertion sort, and selection sort:

Questions?