Review 6

Developing Loops from Invariants

Outline

- 4 questions for loop
- How to develop loops from invariants
- What is on the exam
- Common mistakes

Feel free to ask questions at any time

Four Loopy Questions

1. How does it start?
   - Does the initialization make the invariant true?
2. When does it stop?
   - Invariant + falsity of condition => postcondition
3. Does the repetend make progress toward termination?
4. Does the repetend keep the invariant true?

Developing a Loop on a Range of Integers

- Given a range of integers a..b to process.
- Possible alternatives
  - Could use a for-loop: `for x in range(a,b+1):`
  - Or could use a while-loop: `x = a; while a <= b:`
  - Which one you can use will be specified
- But does not remove the need for invariants
  - **Invariants**: properties of variables outside loop (as well as the loop counter x)
  - If repetend has any variables that are accessed outside of loop, you need an invariant

Developing an Integer Loop (a)

Suppose you are trying to implement the command

Process a..b

Write the command as a postcondition:

post: a..b has been processed.

Developing an Integer Loop (b)

Set-up using for:

```
for k in range(a,b+1):
    # Process k
# post: a..b has been processed.
```

Developing an Integer Loop (b)

Set-up using while:

```
while k <= b:
    # Process k
    k = k + 1
# post: a..b has been processed.
```

Developing an Integer Loop (c)

Add the invariant (for):

```
# invariant: a..k-1 has been processed
for k in range(a,b+1):
    # Process k
# post: a..b has been processed.
```

Note it is post condition with the loop variable

Developing an Integer Loop (c)

Add the invariant (while):

```
# invariant: a..k-1 has been processed
while k <= b:
    # Process k
    k = k + 1
# post: a..b has been processed.
```

Note it is post condition with the loop variable

Developing a For-Loop (d)

Fix the initialization:

Nothing to do unless invariant has variables **other** than loop variable

```
init to make invariant true
# invariant: a..k-1 has been processed
for k in range(a,b+1):
    # Process k
# post: a..b has been processed.
```

Why did not use loop invariants with for loops

Developing a For-Loop (d)

Fix the initialization:

Has to handle the loop variable (and others)

```
init to make invariant true
# invariant: a..k-1 has been processed
while k <= b:
    # Process k
    k = k + 1
# post: a..b has been processed.
```

Developing a For-Loop (e)

Figure out how to "Process k":

```
init to make invariant true
# invariant: a..k-1 has been processed
for k in range(a,b+1):
    # Process k
    implementation of "Process k"
# post: a..b has been processed.
```

Developing a For-Loop (e)

Figure out how to "Process k":

```
init to make invariant true
# invariant: a..k-1 has been processed
while k <= b:
    # Process k
    implementation of "Process k"
    k = k + 1
# post: a..b has been processed.
```

Range

- Pay attention to range:

  a..b or a+1..b or a…b-1 or …

- This affects the loop condition!
  - Range a..b-1, has condition k < b
  - Range a..b, has condition k <= b
- Note that a..a-1 denotes an empty range
  - There are no values in it

Modified Question 3 from Spring 2008

- A magic square is a square where each **row and column adds up to the same number** (often this also includes the diagonals, but for this problem, we will not). For example, in the following 5-by-5 square, 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
```

```
def are_magic_rows(square, value):
    """Returns: True if all rows of square sum to value
    Precondition: square is a 2d list of numbers"""

    # invariant: each row 0..i-1 sums to value
    while            :
        # Return False if row i is does sum to value

    # invariant: each row 0..len(square)-1 sums to value
    return
```

```python
def are_magic_rows(square, value):
    """Returns: True if all rows of square sum to value
    Precondition: square is a 2d list of numbers"""
    i = 0
    # invariant: each row 0..i-1 sums to value
    while i < len(square) :
        # Return False if row i is does sum to value
        rowsum = 0
        # invariant: elements 0..k-1 of square[i] sum to rowsum
        for k in range(len(square)):    # rows == cols
            rowsum = rowsum + square[i][k]
        if rowsum != value:
            return False
        i = i+1
    # invariant: each row 0..len(square)-1 sums to value
    return True
```

```
def are_magic_rows(square, value):
    """Returns: True if all rows of square sum to value
    Precondition: square is a 2d list of numbers"""
    i = 0
    # invariant: each row 0..i-1 sums to value
    while i < len(square) :
        # Return False if row i is does sum to value
        rowsum = 0
        # invariant: elements 0..k-1 of square[i] sum to rowsum
        for k in range(len(square)):    # rows == cols
            rowsum = rowsum + square[i][k]
        if rowsum != value:
            return False
        i = i+1
    # invariant: each row 0..len(square)-1 sums to value
    return True
```

Inner invariant was not required

Invariants and the Exam

- We **will not** ask you for an invariant without both giving you precondition/postcondition
  - So we will give you every extra variable other than the loop variables
  - You just need to reword with the loop variable
- We will try to keep it simple
  - Will only have one loop variable unless it is one of the five required algorithms
  - Only need box diagrams for required algorithms
  - If more complicated, will **give you the invariant**

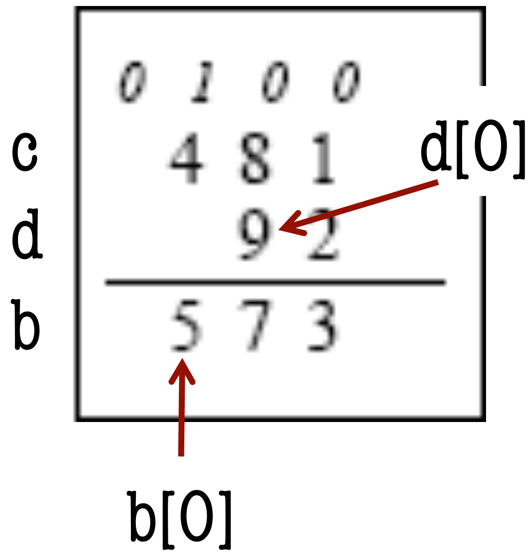
Modified Question 4 from Spring 2007

```
# Given lists b, c, d which with single digit elements
# len(b) = len(c) >= len(d)
# Want to 'add' c and d and put result in b
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 at the beginning
```

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

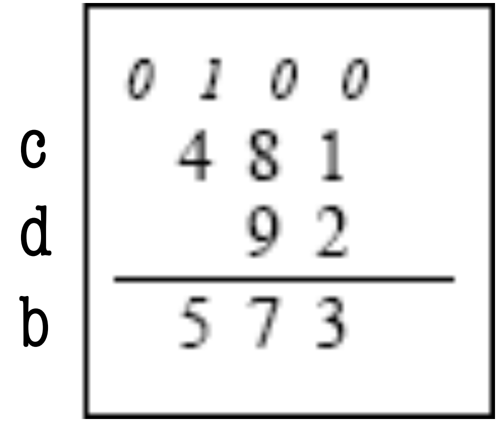
Modified Question 4 from Spring 2007

```
# Given lists b, c, d which with single digit elements
# len(b) = len(c) >= len(d)
# Want to 'add' c and d and put result in b
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 at the beginning
```



Modified Question 4 from Spring 2007

```
h = len(c)
k = len(d)
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 = h -1; k = k – 1 # Easier if decrement first
    x = d[k] if k>= 0 else 0
    b[h] = c[h]+x+carry
    if b[h] >= 10:
        carry = 1; b[h] = b[h]-10
    else:
        carry = 0
# postcondition: b contains the sum of c and d
# except that the carry contains the 0 or 1 at the beginning
```



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 ____________ :
    # Okay to use b, c, d, h, k, and carry
    # Anything else should be 'local' to while
```

Will cost you points on the exam!

DOs and DON'Ts #2

DO double check corner cases!

- h = len(c)
- while h > 0:
  - What will happen when h=1 and h=len(c)?
  - If you use h in c (e.g. c[x]) can you possibly get an error?

```
# 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:
    ...
```

Range is off by 1.
How do you know?

Questions?