Review 6

Developing Loops from Invariants

Outline

- Creating loops from invariants
- What is on the exam
- Common mistakes

Feel free to ask questions at any time

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 body has any variables 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 does not 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 does not 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 does not 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**

Example from Lab 10

```
def num_space_runs(s):
    """The number of runs of spaces in the string s.
    Examples: ' a f g ' is 4 'a f g' is 2 ' a bc d' is 3.
    Precondition: len(s) >= 1"""
    i  = _______
    n = _______
    # invariant: s[0..i] contains n runs of spaces
    while _____________ :



    # postcondition: s contains n runs of spaces
    return n
```

Example from Lab 10

```
def num_space_runs(s):
    """The number of runs of spaces in the string s.
    Examples: ' a f g ' is 4 'a f g' is 2 ' a bc d' is 3.
    Precondition: len(s) >= 1"""
    i  = 0
    n = 1 if s[0] = ' ' else 0
    # invariant: s[0..i] contains n runs of spaces
    while i < len(s)-1 :
        if s[i+1] == ' ' and s[i] != ' ':

            n += 1
        i = i+1
    # postcondition: s contains n runs of spaces
    return n
```

MUST check START of run to satisfy invariant

Example from Lab 10

```
def num_space_runs(s):
    """The number of runs of spaces in the string s.
    Examples: ' a f g ' is 4 'a f g' is 2 ' a bc d' is 3.
    Precondition: len(s) >= 1"""
    i  = -1
    n = 0
    # invariant: s[0..i] contains n runs of spaces
    while i < len(s) -1:
        if s[i+1] == ' ' and (i == -1 or s[i] != ' '):

            n += 1
        i = i+1
    # postcondition: s contains n runs of spaces
    return n
```

Also works

This initialization changes the loop body

DOs and DON'Ts #1

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

```
# invariant: s[0..i] contains n runs of spaces
while ____________ :
    # Okay to use s, i, and n
    # No other loop variables allowed
    # Anything else should be 'local' to while
```

Will cost you points on the exam!

DOs and DON'Ts #2

DO double check corner cases!

- i = -1
- while i < len(s)-1:
  - Why did we choose len(s)-1 instead of len(s)
  - What is problem with "looking ahead" for runs?

```
# invariant: s[0..i] contains n runs of spaces
while i < len(s):
    if s[i+1] == ' ' and s[i] != ' ':
        ...
```

Crashes when i = len(s)-1.
How do you know this?

Example from Lab 10

```
def split(s):
    """Returns a list of words (separated by spaces) in s

    Words are indicated by spaces; there is always a space after each word.

    Example: split('a b c d ') returns ['a','b','c','d']
             split('a ') returns ['a']

    Parameter s: The string to parse
    Precondition: s is a nonempty string with no adjacent spaces.  There is
    no space at the beginning, but there is a single space at the end
    """
```

Example from Lab 10

```
def split(s):
    """Returns a list of words (separated by spaces) in s
    Precondition: s is a string with no adjacent spaces; space at end, not beginning
    """
    pos  = ________
    result = ________
    # invariant: result contains the words in s[0..pos-1], and s[pos-1] is a space
    while _______________ :




    # postcondition: result contains the words in s[0..len(s)-1], and s[-1] is a space
    return result
```

Example from Lab 10

```
def split(s):
    """Returns a list of words (separated by spaces) in s
    Precondition: s is a string with no adjacent spaces; space at end, not beginning
    """
    pos  = s.find(' ')+1
    result = [s[:pos-1]]
    # invariant: result contains the words in s[0..pos-1], and s[pos-1] is a space
    while _______________ :




    # postcondition: result contains the words in s[0..len(s)-1], and s[-1] is a space
    return result
```

Example from Lab 10

```
def split(s):
    """Returns a list of words (separated by spaces) in s
    Precondition: s is a string with no adjacent spaces; space at end, not beginning
    """
    pos  = s.find(' ')+1
    result = [s[:pos-1]]
    # invariant: result contains the words in s[0..pos-1], and s[pos-1] is a space
    while pos < len(s) :




    # postcondition: result contains the words in s[0..len(s)-1], and s[-1] is a space
    return result
```

Example from Lab 10

```
def split(s):
    """Returns a list of words (separated by spaces) in s
    Precondition: s is a string with no adjacent spaces; space at end, not beginning
    """
    pos  = s.find(' ')+1
    result = [s[:pos-1]]
    # invariant: result contains the words in s[0..pos-1], and s[pos-1] is a space
    while pos < len(s) :
        pos2 = s.find(' ',pos)+1
        result.append(s[pos:pos2-1])
        pos = pos2
    # postcondition: result contains the words in s[0..len(s)-1], and s[-1] is a space
    return result
```

Questions?