Lecture 24:
Loop Invariants

Recall: Important Terminology

- **assertion**: true-false statement placed in a program to assert that it is true at that point
  - Can either be a comment, or an `assert` command
- **invariant**: assertion supposed to always be true
  - If temporarily invalidated, must make it true again
  - **Example**: class invariants and class methods
- **loop invariant**: assertion supposed to be true before and after each iteration of the loop
- **iteration of a loop**: one execution of its body

Recall: The while-loop

- **Precondition**: assertion placed before a segment
- **Postcondition**: assertion placed after a segment

Task 1: Setting the table for more people

```python
precondition: n_forks are needed @ table
k = 0
while k < n_more_guests:
    # body goes here
    ...
    k = k + 1
postcondition: n_forks are needed @ table
```

Relationship Between Two
If precondition is true, then postcondition will be true

- **Precondition**: before we start, we should have 2 forks for each guest (dinner fork & salad fork)
- **Postcondition**: after we finish, we should still have 2 forks for each guest

Q: Completing the Loop Body

```python
precondition: n_forks are needed @ table
k = 0
while k < n_more_guests:
    k = k + 1
postcondition: n_forks are needed @ table
```

What statement do you put here to make the postcondition true?

A: n_forks +=2
B: n_forks += 1
C: n_forks = k
D: None of the above
E: I don’t know

Invariants: Assertions That Do Not Change

**Loop Invariant**: an assertion that is true before and after each iteration (execution of body)

```python
precondition: n_forks are needed @ table
k = 0
#INV: n_forks = num forks needed with k more guests
while k < n_more_guests:
    # INV: n_forks += 2
    k += 1
postcondition: n_forks are needed @ table
```

invARIANT: invariant holds before loop
invARIANT: invariant still holds here
Designing Integer while-loops

1. Recognize that a range of integers b..c has to be processed
2. Write the command and equivalent postcondition
3. Write the basic part of the while-loop
4. Write loop invariant
5. Figure out any initialization
6. Implement the body (aka repetend) (# Process k)

# Process b..c
Initialize variables (if necessary) to make invariant true
# Invariant: range b..k-1 has been processed
while k <= c:
    # Process k
    k = k + 1
# Postcondition: range b..c has been processed

Q: What range of s has been processed?

1. Recognize that a range of integers b..c has to be processed
2. Write the command and equivalent postcondition
3. Write the basic part of the while-loop

# set n_pair to number of adjacent equal pairs in s
while k < len(s):
    k = k + 1
# POST: n_pair = # adjacent equal pairs in s[0..len(s)-1]

Q: What is the loop invariant?

1. Recognize that a range of integers b..c has to be processed
2. Write the command and equivalent postcondition
3. Write the basic part of the while-loop
4. Write loop invariant

# set n_pair to number of adjacent equal pairs in s
while k < len(s):
    k = k + 1
# POST: n_pair = # adjacent equal pairs in s[0..len(s)-1]

Q: What do we compare to “process k”?

1. Recognize that a range of integers b..c has to be processed
2. Write the command and equivalent postcondition
3. Write the basic part of the while-loop
4. Write loop invariant
5. Figure out any initialization
6. Implement the body (aka repetend) (# Process k)

# set n_pair to # adjacent equal pairs in s
while k < len(s):
    k = k + 1
# POST: n_pair = # adjacent equal pairs in s[0..len(s)-1]

Task 3: count num adjacent equal pairs

1. Recognize that a range of integers b..c has to be processed

Approach:
Will need to look at characters 0…len(s)-1
Beyond that… not sure yet!

s = 'ebeee', n_pair = 2
s = 'xxxxbee', n_pair = 4

Q: What range of s has been processed?

A: 0..k
B: 1..k
C: 0..k-1
D: 1..k-1
E: I don’t know

Q: how to initialize k?

A: 0
B: 1
C: 0
D: ?

Q: What is the loop invariant?

A: n_pair = # num adj. equal pairs in s[1..k-1]
B: n_pair = # num adj. equal pairs in s[0..k]
C: n_pair = # num adj. equal pairs in s[1..k-1]
D: n_pair = # num adj. equal pairs in s[0..k-1]
E: I don’t know

Q: What do we compare to “process k”?

A: s[k] and s[k+1]
B: s[k-1] and s[k]
C: s[k] and s[k+1]
D: s[k] and s[n]    E: I don’t know