http://www.cs.cornell.edu/courses/cs1110/2018sp

Lecture 24: Loop Invariants

[Online Reading]

CS 1110

Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]

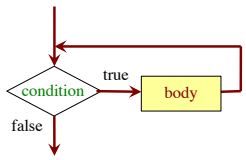
Recall: Important Terminology

- **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
while <condition>:
    statement 1
    ...
    statement n
postcondition
```

body



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

Task 1: Setting the table for more people

```
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

```
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)

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

invariant holds before loop

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
```

19

Task 3: count num adjacent equal pairs

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

s = 'ebeee', n_pair = 2

s = 'xxxxbee', n_pair = 4

Approach:

Will need to look at characters 0...len(s)-1

Beyond that… not sure yet!

20

Q: What range of s has been 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]
```

A: 0..k
B: 1..k
C: 0..k–1
D: 1..k–1
E: I don't know

k: next integer to process.
What range of s has been processed?

22

Q: What is the loop invariant?

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

A: n_pair = num adj. equal pairs in s[1..k]
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

24

Q: how to initialize k?

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

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

A: k = 0
B: k = 1
C: k = –1
D: I don't know

26

Q: What do we compare to "process k"?

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

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

28