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

### Assertions versus Asserts

- **Assertions prevent bugs**
  * Help you keep track of what you are doing
- **Also track down bugs**
  * Make it easier to check belief/code mismatches
- **The `assert` statement is a (type of) assertion**
  * One you are enforcing
  * Cannot always convert a comment to an assert

### Preconditions & Postconditions

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

### Invariants: Assertions That Do Not Change

- **Loop Invariant**: an assertion that is true before and after each iteration (execution of repetend)
  \[
  x = 0; i = 2 \\
  \text{while } i <= 5: \\
  \quad x = x + 1 \times i \\
  \quad i = i + 1 \\
  \# x = \text{sum of squares of 2..i} \\
  \text{Invariant:} \\
  x = \text{sum of squares of 2..i} \\
  \text{in terms of the range of integers that have been processed so far}
  \]

### Invariants: Assertions That Do Not Change

- **Loop Invariant**: an assertion that is true before and after each iteration (execution of repetend)
  \[
  x = 0; i = 2 \\
  \text{while } i <= 5: \\
  \quad x = x + 1 \times i \\
  \quad i = i + 1 \\
  \# x = \text{sum of squares of 2..i} \\
  \text{Invariant:} \\
  x = \text{sum of squares of 2..i} \\
  \text{in terms of the range of integers that have been processed so far}
  \]

### Solving a Problem

- What statement do you put here to make the postcondition true?
  A: `x = x + 1`  
  B: `x = x + n`  
  C: `x = x + n+1`  
  D: None of the above
  E: I don’t know

### Loop Invariant Example

- The loop processes the range 2..5
- Invariant was always true just before test of loop condition. So it’s true when loop terminates

### Loop Invariant Example

- The loop processes the range 2..5
- Invariant was always true just before test of loop condition. So it’s true when loop terminates
Designing Integer while-loops

# Process integers in a..b
# inv: integers in a..k have been processed
k = a
while k <= b:
    process integer k
    k = k + 1
# post: integers in a..b have been processed

Command to do something

Equivalent postcondition

Finding an Invariant

# Make b True if n is prime, False otherwise
b = True
k = 2
# invariant: b is True if no int in 2..k-1 divides n, False otherwise
while k < n:
    # Process k;
    if n % k == 0:
        b = False
    k = k + 1
# b is True if no int in 2..k-1 divides n, False otherwise
What is the invariant?  1  2  3  …  k-1  k  k+1 …  n

Command to do something

Equivalent postcondition

Finding an Invariant

# set x to # adjacent equal pairs in s
x = 0
# inv: x = # adjacent equal pairs in s[0..k-1]
while k < len(s):
    # Process k;
    x = # adjacent equal pairs in s[0..k-1]
    k = k + 1
# x = # adjacent equal pairs in s[0..len(s)-1]

Command to do something

Equivalent postcondition

Finding an Invariant

# s is a string; len(s) >= 1
# Set c to largest element in s
# inv: c is largest element in s[0..k-1]
c = ??
while k < len(s):
    # Process k;
    k = k + 1
# c = largest char in s[0..len(s)-1]

Command to do something

Equivalent postcondition

Reason carefully about initialization

# is a string; len(s) >= 1
# Set c to largest element in s
# inv: c is largest element in s[0..k-1]
c = ??
while k < len(s):
    # Process k;
    k = k + 1
# c = largest char in s[0..len(s)-1]

Command to do something

Equivalent postcondition

1. What is the invariant?
2. How do we initialize c and k?

A: k = 0; c = s[0]
B: k = 1; c = s[0]
C: k = 1; c = s[1]
D: k = 0; c = s[1]
E: None of the above