**Some 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

---

**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

---

**Invariants: Assertions That Do Not Change**

- **Loop Invariant**: an assertion that is true before and after each iteration (execution of repeat)
  \( x = 0; i = 2 \)
  **while** \( i <= 5: \)
  \( x = x + 1\)
  \( i = i + 1 \)

---

**Invariants: Assertions That Do Not Change**

- **Loop Invariant**: an assertion that is true before and after each iteration (execution of repeat)
  \( x = 0; i = 2 \)
  **while** \( i <= 5: \)
  \( x = x + 1\)
  \( i = i + 1 \)

---
Designing Integer while-loops

Command to do something

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

Equivalent postcondition

Finding an Invariant

Command to do something

# 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..n-1 divides n, False otherwise

Finding an Invariant

Command to do something

# 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;
    k = k + 1
# x = # adjacent equal pairs in s[0..len(s)-1]

Finding an Invariant

Command to do something

# s is a string; len(s) >= 1
# Set c to largest element in s
# e: ??

Finding an Invariant

Command to do something

# 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;
    k = k + 1
# x = # adjacent equal pairs in s[0..len(s)-1]

Finding an Invariant

Command to do something

# s is a string; len(s) >= 1
# Set c to largest element in s
# e: ??

Reason carefully about initialization

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

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