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

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
  while i <= 8:
    x = x + i
    i = i + 1
  # x = sum of squares of 2..8

  **Invariant**:
  - x = sum of squares of 2..i
  - in terms of the range of integers that have been processed so far

Preconditions & Postconditions

<table>
<thead>
<tr>
<th>n</th>
<th>1 2 3 4 5 6 7 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>contains the sum of these (6)</td>
</tr>
<tr>
<td>n = n + 1</td>
<td></td>
</tr>
</tbody>
</table>

**Precondition**: assertion placed before a segment

**Postcondition**: assertion placed after a segment

Relationship Between Two

If precondition is true, then postcondition will be true

Solving a Problem

<table>
<thead>
<tr>
<th>n</th>
<th>1 2 3 4 5 6 7 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>contains the sum of these (10)</td>
</tr>
</tbody>
</table>

**Relationship Between Two**

If precondition is true, then postcondition will be true

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
while i <= 8:
    x = x + i
    i = i + 1
# x = sum of squares of 2..i

**Invariant**:
- x = sum of squares of 2..i
- in terms of the range of integers that have been processed so far

Integers that have been processed:
- Range 2..i-1: 2, 3, 4, 5
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
    process integer k
k = k + 1
# post: integers in a..b have been processed

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..n-1 divides n, False otherwise

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 = x + 1
# x = # adjacent equal pairs in s[0..len(s)-1]
k: next integer to process.
What is initialization for k?
A: k = 0
B: k = 1
C: k = -1
D: I don’t know

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 = x + 1
# x = # adjacent equal pairs in s[0..len(s)-1]
k: next integer to process.
What is initialization for k?
A: k = 0
B: k = 1
C: k = -1
D: I don’t know

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 = x + 1
# x = # adjacent equal pairs in s[0..len(s)-1]
k: next integer to process.
What is initialization for k?
A: k = 0
B: k = 1
C: k = -1
D: I don’t know

Reason carefully about initialization

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

E: None of the above