

Exam Info

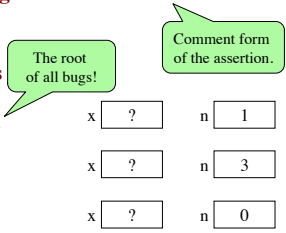
- **Today, 7:30–9:00PM**
 - Last name **A – G** in Olin 155
 - Last name **H – K** in Olin 165
 - Last name **L – R** in Olin 255
 - Last name **S – Z** in Upson B17
 - **Extra-time Students:** Upson 5130 at 6:30 pm
- **Makeup:** Friday, 6:30-8pm in Upson 5130
 - Only if you have contacted me for permission
- Exams will be graded over the weekend

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
- **precondition:** assertion placed before a statement
 - Same idea as **function precondition**, but more general
- **postcondition:** assertion placed after a statement
- **loop invariant:** assertion supposed to be true before and after each iteration of the loop
 - Distinct from **attribute invariant**
- **iteration of a loop:** one execution of its body

Assertions versus Asserts

- **Assertions prevent bugs** # x is the sum of 1..n
 - 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

```

# x = sum of 1..n-1
x = x + n
n = n + 1
# x = sum of 1..n-1
    
```

precondition

1 2 3 4 5 6 7 8
x contains the sum of these (6)

1 2 3 4 5 6 7 8
x contains the sum of these (10)

postcondition

- **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

```

# x = sum of 1..n
n = n + 1
# x = sum of 1..n
    
```

precondition

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

postcondition

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 repetend)

```

x = 0; i = 2
while i <= 5:
    x = x + i*i
    i = i + 1
# x = sum of squares of 2..5
    
```

Invariant:
x = sum of squares of 2..i-1

in terms of the range of integers that have been processed so far

The loop processes the range 2..5

Invariants: Assertions That Do Not Change

```

x = 0; i = 2
# Inv: x = sum of squares of 2..i-1
while i <= 5:
    x = x + i*i
    i = i + 1
# Post: x = sum of squares of 2..5

```

Integers that have been processed: 2, 3, 4, 5
Range 2..i-1: 2..5

Invariant was always true just before test of loop condition. So it's true when loop terminates

The loop processes the range 2..5

Designing Integer while-loops

```

# 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

```

Command to do something
Equivalent postcondition

Designing Integer while-loops

- Recognize that a range of integers b..c has to be processed
- Write the command and equivalent postcondition
- Write the basic part of the for-loop
- Write loop invariant
- Figure out any initialization
- Implement the 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

```

Finding an Invariant

```

# Make b True if no int in 2..n-1 divides n, 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

```

Command to do something
Equivalent postcondition

What is the invariant? 1 2 3 ... k-1 k k+1 ... n

Finding an Invariant

```

# set x to # adjacent equal pairs in s[0..len(s)-1]
for s = 'ebee', x = 2
while k < len(s):
    # Process k
    k = k + 1
# x = # adjacent equal pairs in s[0..len(s)-1]

```

Command to do something
Equivalent postcondition

k: next integer to process.
Which have been processed?

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

Reason carefully about initialization

```

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

```

Command to do something
Equivalent postcondition

- What is the invariant?
- 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