Announcements for This Lecture

Assignments

- A6 due in one week
  - Dataset should be done
  - Get on track this weekend
  - **Next Week**: ClusterGroup
- A7 will be last assignment
  - Due after classes over
  - Posted before Thanksgiving
  - Lab next week
- **No lab** week of Turkey Day

Prelim 2

- Thursday, 7:30-9pm
  - A–Sh (Statler Aud)
  - Si–X (Statler 196)
  - Y–Z (Statler 198)
  - SDS received e-mail
- **Make-up** is Friday
  - Only if submitted conflict
  - Also received e-mail
- Graded on Saturday

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: Preconditions & Postconditions

precondition

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

postcondition

n

1 2 3 4 5 6 7 8

x contains the sum of these (6)

n

1 2 3 4 5 6 7 8

x contains the sum of these (10)

- **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

precondition

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

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

postcondition

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

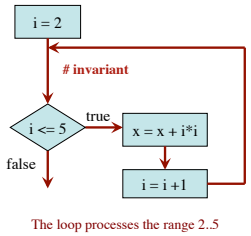
- **Loop Invariant**: an assertion that is true before and after each iteration (execution of repetend)

```
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

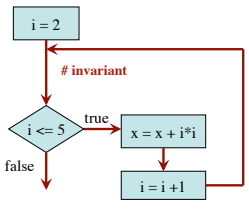
```
x = 0; i = 2
# Inv: x = sum of squares of 2..i-1
while i <= 5:
    x = x + i*i
    i = i +1
# Post: x = sum of squares of 2..5
```

Integers that have been processed:

Range 2..i-1:

x 0

i ?



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
```

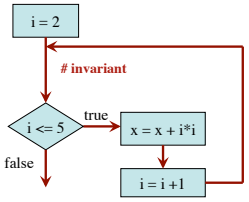
Integers that have been processed: 2, 3, 4, 5

Range 2..i-1: 2..5

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

x: 0 4 13 29 54

i: 2 3 4 5 6



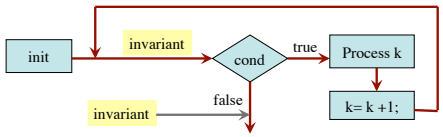
The loop processes the range 2..5

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

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 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

Command to do something

```
# 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
```

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]
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]
```

Command to do something

for s = 'ebeee', x = 2

Equivalent postcondition

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 = ??
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

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