CS1110 5 March 2009

Read: Sec. 2.3.8 and chapter 7 on loops.
The lectures on the ProgramLive CD can be a big help.

1. Prelim 2 next Thursday evening, 12 March, 7:30PM, *Uris Auditorium*.
   For-loops are **not** on this prelim.
   Conflict? Email Maria Witlox, mwitlox@cs.cornell.edu, by this evening.
   One handout is the prelim study guide.
   Review session 1-3pm Sunday March 8 in Phillips 101.

2. Please complete an online questionnaire concerning your TA.
http://www.engineering.cornell.edu/TAEval/menu.cfm
This is a midterm evaluation. It is important because your constructive comments are used to help the TA improve, which may help you in this course.

3. Assignment 5 "due" Sunday (although actually due Wed. March 11)

Assertions

Assertions are true-false statements (comments) asserting your beliefs about (the current state of) your program.

// x is the sum of 1..n <- asserts a specific relationship between x and n

Assertions help prevent bugs by helping you keep track of what you're doing…
… and **they help track down bugs** by making it easier to check belief/code mismatches.

Assertions can help with **loop bugs**: initialization errors, termination errors, and processing errors.

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

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)

Solving a problem

precondition

```
// x  = sum of 1..n
x= x + n+1;
n= n + 1;
// x =  sum of 1..n
```

What statement do you put here so that segment is correct? (if precondition is true, execution of segment should make postcondition true.)

postcondition

@ precondition:

n
1 … i-2 i-1 i i+1 i+2…

x contains the sum of these
(but we only know "i" through variable n)

Solving a problem

precondition

```
// x  = sum of 1..n-1
x= x + n;
n= n + 1;
// x =  sum of 1..n-1
```

What statement do you put here so that segment is correct? (if precondition is true, execution of segment should make postcondition true.)

postcondition

@ precondition:

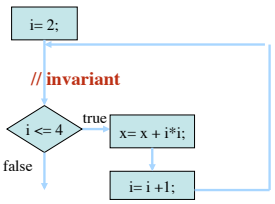
n
1 … i-2 i-1 i i+1 i+2…

x contains the sum of these
(but we only know "i" through variable n)

Invariants: another type of assertion

An invariant is an assertion about the variables that is true before and after each iteration (execution of the repetend).

```
for (int i= 2; i <= 4; i= i +1) {
   x= x + i*i;
}
```



```
// Process integers in a..b
 // inv: the integers in a..k-1 have been processed
for (int k= a;  k <= b;  k= k + 1) {
        Process integer k;
}
// post: the integers in a..b have been processed
```

Command to do something

equivalent post-condition

Methodology for developing a for-loop

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 for-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
for (int k= b;  k <= c;  k= k+1) {
   // Process k
}
// Postcondition: range b..c has been processed
```


Finding an invariant

```
// Store in double variable v the sum
//   1/1 + 1/2 + 1/3 + 1/4 + 1/5 + … + 1/n
 v= 0;
 // invariant: v = sum of 1/i for i in 1..k-1
for (int k= 1; k <= n;  k= k +1) {
        Process k;
}
// v =1/1 + 1/2 + … + 1/n
```

Command to do something and

equivalent postcondition

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

Finding an invariant

```
// set x to no. of adjacent equal pairs in s[0..s.length()-1]
```

for s = 'ebeee', x = 2.

Command to do something and equivalent post-condition

```
// invariant:  x = no. of adjacent equal pairs in s[0..k-1]
for (int k= 0; k < s.length();  k= k +1) {
        Process k;
}
// x = no. of adjacent equal pairs in s[0..s.length()-1]
```

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

A. 0..k C. a..k

B. 0..k–1 D. a..k–1

What is the invariant?

A. x = no. adj. equal pairs in s[1..k]

B. x = no. adj. equal pairs in s[0..k]

C. x = no. adj. equal pairs in s[1..k–1]

D. x = no. adj. equal pairs in s[0..k–1]

A warning

```
// { Vector of Integers v has > 0 items }
// Set m to maximum Integer in v

// inv:  m is largest Integer in v[0..k–1]
m= ??;
for (int k=  ??;  k < v.size();  k= k + 1) {
      // Process k;
}
//  m = largest Integer in v[0..v.size()–1]
```

Although this invariant is "typical", the "typical" initialization "k= 0; m= v.get(0);" isn't appropriate:

An empty set (of Integers) has no maximum. Therefore, be sure that 0..k–1 is not empty. Therefore, start with k = 1.