CS1110 21 October 2010

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

Some anagrams

A decimal point Debit card Bad credit Dormitory Dirty room Schoolmaster The classroom Statue of liberty The Morse code Western Union No wire unsent Parishioners I hire parsons

Animosity Is no amity
Desperation A rope ends it
Funeral Real fun
Slot machines Cash lost in 'em
Snooze alarms Alas! No more Z's
Vacation times I'm not as active
George Bush He bugs Gore
The earthquakes That queen shake

Circumstantial evidence Can ruin a selected victim
Victoria, England's queen Governs a nice quiet land
Eleven plus two Twelve plus one (and they have 13 letters!)

llus two Twelve plus one (and they have 13 letters!)

A4: A4 max times: A4 times: mean: 96.7 mean: 5.6 15 (2 people) median: 98 median: 5 12 (2 people) std dev: 2.5 10 (4 people) std dev: 6.2 09 (5 people) Only 107 people gave us the time spent 08 (5 people) Today's terminology: assertion: true-false statement, sometimes placed in a program to assert that it is true at that point. precondition: assertion placed before a statement postcondition: assertion placed after a statement loop invariant: assertion supposed to be true before and after each iteration of the loop iteration of a loop: one execution of its repetend And we give you a methodology for developing for-loops.

Assertion: true-false statement (comment) asserting a belief about (the current state of) your program.

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

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

x ? n 1

x ? n 0

x ? n 3

assert <boolean expression> ;

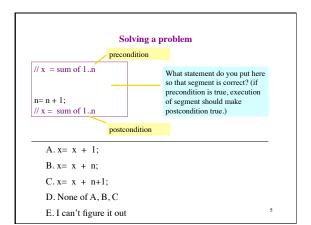
Assertions help prevent bugs by helping you keep track of what you're doing \dots

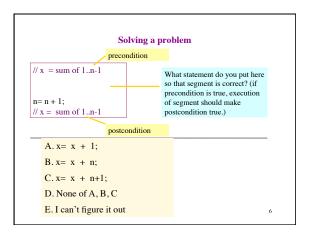
... and they help track down bugs by making it easier to check belief/code mismatches

Java assert statement. To

execute: if the bool exp is

false, stop with an error message





```
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 \le 4; i = i + 1) {
                                            i= 2;
    x=x+i*i;
                                            // invariant
  Invariant:
                                                    true
                                                         x = x + i*i;
   x = sum of squares of 2..i-1
                                        false
                                                           i= i +1;
in terms of the range of integers
that have been processed so far
                                  The loop processes the range 2..4 7
```

```
// Process integers in a..b Command to do something

// 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 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
```

```
// Store in b the value of:

"no int in 2..n-1 divides n

b= true;

// invariant: b = no int in 2..k-1 divides n

for (int k= 2; k < n; k= k+1) {

// Process k;
} if (n%k == 0) b= false;

// b = "no int in 2..n-1 divides n"

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] — Command
                    for s = \text{`ebeee'}, x = 2.
                                                                  something
                                                                        and
                                                                  equivalent
// invariant:
                                                                   post-
condition
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.
                             What is the invariant?
Which ones have been
                              A. x = no. adj. equal pairs in s[1..k]
processed?
                              B. x = no. adj. equal pairs in s[0..k]
A. 0..k
               C. a..k
                              C. x = no. adj. equal pairs in s[1..k-1]
B. 0..k-1
              D. a..k-1
                             D. x = \text{no. adj. equal pairs in } s[0..k-1]
```

