CS1110 27 October 2009
while loops

Reading: today: Ch. 7 and ProgramLive sections.

For next time: Ch. 8.1-8.3

Prelim 2. Tuesday, 10 November, 7:30PM
If you have a conflict, and if you haven't been contacted about the conflict, please email Maria Witlox mwitlox@cs.cornell.edu by Friday!!!!

Make sure you give her your last name, first name, Cornell netid. State clearly and completely what the conflict is. (E.g. don't just say "I have another test." State what the course (or whatever) is.)

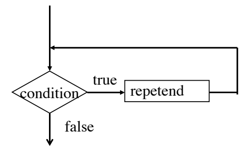
Watch the lectures on www.videonote.com/cornell

Beyond ranges of integers: the while loop

```
while (<condition>) {
  sequence of declarations
  and statements
}
```

<condition>: a boolean expression.

<repetend>: sequence of statements.



In comparison to for-loops: we get a broader notion of "there's still stuff to do" (not tied to integer ranges), but *we* must ensure that "condition" stops holding (since there's no explicit increment).

Canonical while loops

```
// Process b..c
for (int k= b; k <= c; k= k+1) {
    Process k;
}
```

scope of k: the loop; k can't be used after the loop

```
// Process b..c
int k= b;
while (k <= c) {
    Process k;
    k= k+1;
}
```

Here's one way to use the while loop

```
// process a sequence of input not of fixed size
<initialization>;
while (<still input left>) {
    Process next piece of input;
    make ready for the next piece of input;
}
```


Understanding assertions about lists

	0	1	2	3	4	5	6	7	8
v	X	Y	Z	X	A	C	Z	Z	Z

This is a list of Characters

v: 0..3 $\geq$ C | ? | k..8 all Z's — k = 6

This is an assertion about v and k. It is **true** because chars of v[0..3] are greater than 'C' and chars of v[6..8] are 'Z's.

v: 0..3 $\geq$ C | ? | k..8 all Z's — k = 5

v: 0.. $\geq$ C | k..8 all Z's — k = 6

v: 0.. $\geq$ W | A | C | k..8 all Z's — k = 4

Indicate whether each of these 3 assertions is true or false.

The while loop: 4 loopy questions. Allows us to focus on one thing at a time and thus separate our concerns.

```
// Set c to the number of 'e's in String s.
int n= s.length();
k= 0; c= 0;
// inv: c = #. of 'e's in s[0..k-1]
while (k < n) {
    if (s.charAt(k) == 'e')
        c= c + 1;
    k= k+ 1;
}
// c = number of 'e's in s[0..n-1]
```

1. How does it start? ((how) does init. make inv true?)

2. When does it stop? (From the invariant and the falsity of loop condition, deduce that result holds.)

3. (How) does it make progress toward termination?

4. How does repetend keep invariant true?

The four loopy questions

Suppose we are thinking of this while loop:

```
initialization;
while ( B ) {
    repetend
}
```

We add the postcondition and also show where the invariant must be true:

```
initialization;
// invariant: P
while ( B ) {
    // { P and B}
    repetend
    // { P }
}
// { P and !B }
// { Result R }
```

Second box helps us develop four loopy questions for developing or understanding a loop:

1. How does loop start? Initialization must truthify invariant P.

2. When does loop stop?

At end, P and !B are true, and these must imply R. Find !B that satisfies
P && !B => R.

3. Make progress toward termination? Put something in repetend to ensure this.

4. How to keep invariant true? Put something in repetend to ensure this.

Appendix examples: Develop loop to store in x the sum of 1..100.

We'll keep this definition of x and k true:
x = sum of 1..k–1

1. How should the loop start? Make range 1..k–1 empty: **k= 1; x= 0;**

2. When can loop stop? What condition lets us know that x has desired result? When **k == 101**

3. How can repetend make progress toward termination? k= k+1;

4. How do we keep def of x and k true? x= x + k;

Four loopy questions

```
k= 1;  x= 0;
// invariant: x = sum of 1..(k–1)
while ( k != 101) {
    x=  x + k;
    k= k + 1;
}
// { x = sum of 1..100 }
```


Building a fair coin from an unfair coin

```
/** = result of flipping a fair coin
      (heads/tails is true/false) */
public static boolean fairFlip() {
    boolean f1= new unfair flip;
    boolean f2= new unfair flip;
    /* invariant P: f1, f2 contain results
            of 2 unfair flips, and
         in all previous flips, f1 and f2
         were the same */
     while (f1 == f2) {
         f1= new unfair flip;
         f2= new unfair flip;
     }
    // R: P  and  f1 != f2
    return !f1   &&  f2
}
```

John von Neumann: building a "fair coin" from an unfair coin

loopy questions:

1. P is true initially
2. When it stops, R is true
4. Repetend keeps P true
3. But we can't prove that the loop makes progress toward termination!

Can't get something for nothing

Unfair flip produces heads with some probability p, 0 < p < 1

Roach infestation

```
/** = number of weeks it takes roaches to fill the apartment --see p 244 of text*/
public static int roaches() {
      double roachVol= .001;     // Space one roach takes
      double aptVol= 20*20*8;  // Apartment volume
      double growthRate= 1.25; // Population growth rate per week

      int w=    0    ;        // number of weeks
      int pop= 100;         // roach population after w weeks

      // inv: pop = roach population after w weeks   AND
      //       before week w, volume of roaches < aptVol
      while ( aptVol > pop * roachVol) {
         pop= (int) (pop * growthRate);
         w= w + 1;
      }
      // Apartment is filled, for the first time, at week w.
      return w;
  }
```


Iterative version of logarithmic algorithm to calculate b**c (we've seen a recursive version before).

```
/** set z to b**c, given c ≥ 0 */
int x= b; int y= c; int z= 1;
// invariant:  z * x**y = b**c  and 0 ≤ y ≤ c
while (y != 0) {
    if (y % 2 == 0)
           { x= x * x; y= y/2;  }
    else { z= z * x; y= y – 1; }
}
// { z = b**c }
```


Calculate quotient and remainder when dividing x by y

x/y = q + r/y   **21/4= 4 + 3/4**

Property: x = q * y + r and 0 ≤ r < y

```
/** Set q to quotient  and r to remainder.
    Note: x >= 0 and y > 0 */
int q= 0; int r= x;
// invariant:  x = q * y + r    and 0 ≤ r
while (r >= y) {
        r= r – y;
        q= q + 1;
}
// { x = q * y + r    and    0 ≤ r < y }
```