**CS1110 Lecture 16, 26 Oct 2010**

**While-loops**

Reading for next time: Ch. 8.1-8.3 (arrays)

**Prelim 2:** Tu Nov 9th, 7:30-9pm.
Last name A-Lewis: Olin 155
Last name L-Z: Olin 255

**Conflicts?** Submit CMS "assignment" "P2 conflicts" by today.

**Review session:** Sun Nov 7th, 1-3pm, Phillips 101. (Set your clocks back the night before!)

Reminder: A5 due Sat. Oct 30th. See assignments pg for hints on snowflake geometry.

A mystery (due to von Neumann): suppose you have a biased coin with unknown probability of heads $p$, $0 < p < 1$.

How can you use this coin to simulate the output of a fair coin?

(Answer: while-loops …)

---

**Beyond ranges of integers: the while loop**

```java
while (<condition>) {
    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.

---

**Interesting while loops (showing why they can be hard to understand)**

```java
// open question in mathematics: is there an n such that this function never returns a value (i.e., it doesn’t "return to 1")?
public static boolean unfairFlip() {
    boolean f1 = new unfair flip;
    boolean f2 = new unfair flip;
    while (true) // loop "forever"
        if (f1 && !f2) {
            return true; // escape the loop
        } else if (!f1 && f2) { // TH
            return false;
        } else {
            n = n/2;
        }
    return true;
}
```

---

**How to analyze loops: understanding assertions about lists**

<table>
<thead>
<tr>
<th>v index</th>
<th>Character</th>
<th>is C?</th>
<th>all Cs?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>is not C</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>Y</td>
<td>is C</td>
<td>?</td>
</tr>
<tr>
<td>2</td>
<td>Z</td>
<td>is C</td>
<td>?</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>is C</td>
<td>?</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td>is C</td>
<td>?</td>
</tr>
<tr>
<td>5</td>
<td>C</td>
<td>is C</td>
<td>?</td>
</tr>
<tr>
<td>6</td>
<td>C</td>
<td>is C</td>
<td>?</td>
</tr>
<tr>
<td>7</td>
<td>C</td>
<td>is C</td>
<td>?</td>
</tr>
</tbody>
</table>

This is an assertion about v and k, thus explaining the meaning of these variables. It is true because v[0..3] are not C and v[6..8] are C's.

This falsely asserts that v[0..3] are C's, v[5..8] are C's.

This is a true assertion about v and k, since v[0..3] are not C's, v[1..5] is A, v[5..8] are D & v[6..8] are C's.

---

**Counting characters:** Store in n the number of ‘/’ in string s.

// Store in n to truthify diagram R

k = 0; n = 0;

// inv: See diagram P below

while (s.length() > 0) {
    if (s.charAt(k) == '/') {
        n = n + 1;
        k = k + 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 repend keep invariant true?
Suppose we are thinking of this while loop:

```java
while (B) {
    // Collect  stuff
}
```

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}

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;

The four loopy questions

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

1. How does loop start? Initial value of i 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.

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}

The four loopy questions

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

1. How does loop start? Initial value of i 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.

Linear search. Character c is in string s. Find its first position.

```
// Store in k to trutify diagram R
Idea: Start at beginning of s, looking for c; step when found.
```

```
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?
```

```
1. How does loop start? Make range 1..k–1 empty:

```java
k= 1;  x= 0;
```

We'll keep this definition of x and k true:

```
x = sum of 1..(k–1)
```

```
while ( k != 101) {
    x=  x + k;
    k= k + 1;
}
```

```
// { x = sum of 1..100 }
```

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?

```
R:  s      c not here     c         ?
   0                     k                      s.length()
```

```
// Store in k to truthify diagram R
Idea: Start at beginning of s, looking for c; stop when found.
```

```
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?
```

```
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?
```

```
R:  s      c not here     c         ?
   0                     k                      s.length()
```

```
// Store in k to truthify diagram R
Idea: Start at beginning of s, looking for c; stop when found.
```

```
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?
```

```
R:  s      c not here     c         ?
   0                     k                      s.length()
```

```
// Store in k to truthify diagram R
Idea: Start at beginning of s, looking for c; stop when found.
```

```
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?
```

```
R:  s      c not here     c         ?
   0                     k                      s.length()
```

```
// Store in k to truthify diagram R
Idea: Start at beginning of s, looking for c; stop when found.
```

```
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?
```

```
R:  s      c not here     c         ?
   0                     k                      s.length()
```

```
// Store in k to truthify diagram R
Idea: Start at beginning of s, looking for c; stop when found.
```

```
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?
```

```
R:  s      c not here     c         ?
   0                     k                      s.length()
```

```
// Store in k to truthify diagram R
Idea: Start at beginning of s, looking for c; stop when found.
```

```
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?
```

```
R:  s      c not here     c         ?
   0                     k                      s.length()
```

```
// Store in k to truthify diagram R
Idea: Start at beginning of s, looking for c; stop when found.
```

```
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?
```

```
R:  s      c not here     c         ?
   0                     k                      s.length()
```

```
// Store in k to truthify diagram R
Idea: Start at beginning of s, looking for c; stop when found.
```

```
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?
```

```
R:  s      c not here     c         ?
   0                     k                      s.length()
```

```
// Store in k to truthify diagram R
Idea: Start at beginning of s, looking for c; stop when found.
```

```
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?
```

```
R:  s      c not here     c         ?
   0                     k                      s.length()
```

```
// Store in k to truthify diagram R
Idea: Start at beginning of s, looking for c; stop when found.
```

```
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?
```

```
R:  s      c not here     c         ?
   0                     k                      s.length()
```

```
// Store in k to truthify diagram R
Idea: Start at beginning of s, looking for c; stop when found.
```

```
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?
```

```
R:  s      c not here     c         ?
   0                     k                      s.length()
```

```
// Store in k to truthify diagram R
Idea: Start at beginning of s, looking for c; stop when found.
```

```
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?
```

```
R:  s      c not here     c         ?
   0                     k                      s.length()
```

```
// Store in k to truthify diagram R
Idea: Start at beginning of s, looking for c; stop when found.
```

```
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?
```

```
R:  s      c not here     c         ?
   0                     k                      s.length()
```

```
// Store in k to truthify diagram R
Idea: Start at beginning of s, looking for c; stop when found.
```

```
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?
```

```
R:  s      c not here     c         ?
   0                     k                      s.length()
```