CS1110 23 October 2008

The while loop and assertions

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

Quotes for the Day:

Instead of trying out computer programs on test cases until they are debugged, one should prove that they have the desired properties.
John McCarthy, 1961, A basis for a mathematical theory of computation.

Testing may show the presence of errors, but never their absence.
Dijkstra, Second NATO Conf. on Software Engineering, 1969.

A week of hard work on a program can save you 1/2 hour of thinking.
Paul Gries, CS, University of Toronto, 2005.

1

The while loop: syntax

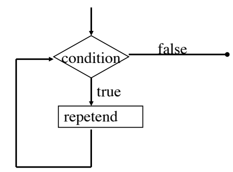
while (*<condition>*)
<repetend>

<condition>: a boolean expression.

<repetend>: a statement.

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

BUT: We almost always make the *<repetend>* a block.



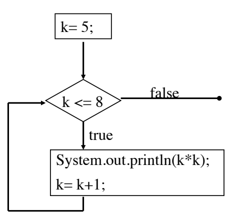
2

The while loop

```
System.out.println(5*5);
System.out.println(6*6);
System.out.println(7*7);
System.out.println(8*8);
```

To execute the while loop:
(1) Evaluate condition k <= 8;
if false, stop execution.
(2) Execute the repetend.
(3) Repeat again from step (1).

```
int k= 5;
while ( k <= 8) {
    System.out.println(k*k);
    k= k+1;
}
```



Trace execution of the loop: Study section 7.1.2 shows you how to “trace” execution of a loop.

3

For loop, corresponding while loop

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

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

4

4 loopy questions

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

1. How does it start? (what is the initialization?)
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?

5

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

```
// Set c to the number of 'e's in String s.
int n= s.length();
c= 0;  k= 0;
//  invariant: c = number 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? (what is the initialization?)
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?

6

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

```
    0    3     k          8
v  | ≥ C  |  ?  |   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.

```
    0    3     k          8
v  | ≥ C  |  ?  |   all Z's  |     k  | 5 |

    0          k          8
v  | ≥ C       |   all Z's  |     k  | 6 |

    0        k            8
v  | ≥ W      |A|C|  all Z's |     k  | 4 |
```

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

7

The four loopy questions

Suppose we have this while loop, with initialization:

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

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

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

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

1. How does loop start? Initialization must truthify inv 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.

8

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

Four loopy questions

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 result? When **k == 101**

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

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

```
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 }
```

9

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 the roaches < aptVol
      while (aptVol > pop * roachVol ) {
         pop= (int) (pop * growthRate);
         w= w + 1;
      }
      return w;
  }
```

10

Iterative version of logarithmic algorithm to calculate b**c.

```
/** = b**c, given c ≥ 0 */
public static int exp(int b, int  c) {
    if (c == 0) return 1;
    if (c%2 = 0) return exp(b*b, c/2);
    return b * exp(b, c–1);
}
```

```
/** 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 }
```

Rest on identities:

b**0 = 1

b**c = b * b**(c-1)

for even c, b**c = (b*b)**(c/2)

3*3 * 3*3 * 3*3 * 3*3 = 3**8

(3*3)*(3*3)*(3*3)*(3*3) = 9**4

Algorithm is *logarithmic in c*, since time is proportional to log c

11

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  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 }
```

12