CS1110 30 March 2012. while-loops

Haikus (5-7-5) seen on Japanese computer monitors

Reading: today: Ch. 7 and ProgramLive sections.

Yesterday it worked.
Today it is not working.
Windows is like that.

A crash reduces
Your expensive computer
To a simple stone.

Three things are certain:
Death, taxes, and lost data.
Guess which has occurred?

Serious error.
All shortcuts have disappeared.
Screen. Mind. Both are blank.

The Web site you seek
Cannot be located, but
Countless more exist.

Chaos reigns within.
Reflect, repent, and reboot.
Order shall return.

Beyond ranges of integers: the while loop

while (<*condition*>) {

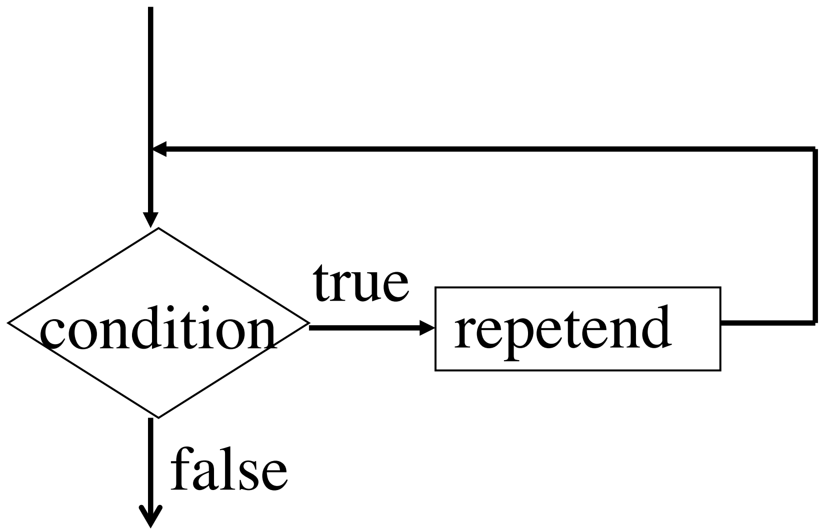
sequence of declarations

and statements

}

<*condition*>: a boolean expression

<*repetend*>: sequence of statements



In comparison with for-loops: a broader notion of "still stuff to do" (not tied to integer ranges), but we must ensure that the condition becomes false (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;
}
```

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

scope of k: from its declaration to end of block in which declaration occurs. k can be used after the loop.

```
// Precondition: 1 <= n
// Set s to the largest power of 2 that is at most n.

    s= 1;
    // Keep this true: s is a power of 2 and
    //                          s <= n
    while (2 * s  <=  n) {
        s= 2*s;     // Make progress toward termination
                    //  and keep assertion true
    }

// R: s is a power of 2 and s <= n and 2*s > n
```

Example: n = 1. 2^0 =   1 but 2^1 =   2. So set s to   1.

Example: n = 31. 2^4 = 16 but 2^5 = 32. So set s to 16.

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 item of input;
    make ready for next item of input;
}
```

```
// Set n to number of lines in file that have "/" in them.
String s= first line of file (null if none);
int n= 0;
while (s != null) {
    if (s.contains("/"))
        n= n+1;
    s= next line of file (null if none);
}
```

You will learn how to read/write files on your hard drive in a few weeks

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	$\geq$ C	?	all Z's

k: 6

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	$\geq$ C	?	all Z's

k: 5

	0	k ... 8
v	$\geq$ C	all Z's

k: 6

	0	k		... 8
v	$\geq$ W	A	C	all Z's

k: 4

This is:

A. true

B. False

C. I don't know

Set t to number of times the first char appears at beginning of s.
Precondition: s not empty

s	t
"bbbcgbb"	3
"$b$$$"	1
"hh"	2

```
t= 1;
while (t < s.length() &&
        s.charAt[t] == s.charAt[t-1]) {
    t= t + 1;
}


// { R1 and R2 }   i.e. the postcondition
```

Question: how can we know that this works –without having to execute it on several cases?

	0	t	s.length
R1:	these are all the same		

R2: either t = s.length or s[t] != s[t-1]

Set t to number of times the first char appears at beginning of s.
Precondition: s not empty

s	t
“bbbcgbb”	3
“$b$$$”	1
“hh”	2

```
t=  1;
// invariant: R1
while (t < s.length() &&
         s.charAt[t] == s.charAt[t-1]) {
     t= t + 1;


}
// { R1 and R2 }   i.e. the postcondition
```

Invariant will be true before and after each iteration

	0	t	s.length
R1:	all the same		

R2: either t = s.length or s[t] != s[t-1]

1. Initialization right?
2. Condition right?
3. Repetend keep invariant true?
4. Repetend make progress toward termination?

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

// Store in k to truthify diagram R

Idea: Start at beginning of s, looking for c; stop when found.
How to express as an invariant?

```
k= 0;

// invariant: See diagram P, below

while (    s.charAt(k) != c     ) {

    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 repetend keep invariant true?

P : s

0	k	s.length()
c not here	?	

R: s

0	k		s.length()
c not here	c	?	

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?

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 }
```

The four loopy questions

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.

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 *(1 + growthRate));
        w= w + 1;
    }
    // Apartment is filled, for the first time, at week w.
    return w;
}
```