CS100J October 16, 2003
More on Loops
Reading: Secs 7.1–7.4

I have graded Q1, Q2, and A2. Monday morning, they will be placed in the Carpenter basement, to be picked up when a consultant is there.

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.

On “fixing the invariant”

```
// {s is the sum of 1..h}
s=  s + (h+1);
h= h+1;
// {s is the sum of 1..h}
```

On "fixing the invariant"

// {s is the sum of h..n}   s = 5 + 6 + 7 + 8   h = 5, n = 8

s= s + (h-1);

h= h-1;

// {s is the sum of h..n} s = 4 + 5 + 6 + 7 + 8   h = 4, n = 8

Loop pattern to process a range m..n–1
(if m = n, the range is empty)

```
int h= m;
// invariant: m..h–1 has been processed
while (h != n) {
    Process h;
    h= h+1;
}
// { m..n–1 has been processed }
```

5..7

5..6

5..5

5..4

Loop pattern to process a range m..n
(if m = n+1, the range is empty)

```
int h= m;
// invariant:  m..h–1 has been processed
while (h != n+1) {
    Process h;
    h= h+1;
}
// { m..n has been processed }
```

Loop pattern to process a range m..n in reverse order
(if m = n+1, the range is empty)

```
int h= n+1;
// invariant:  h..n has been processed (in reverse)
while (h != m) {
    Process h–1;
    h= h–1;
}// { m..n has been processed (in reverse)}
```

Logarithmic algorithm to calculate b**c, for c >= 0 (i.e. b multiplied by itself c times)

```
/** 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 }
```

Decimal	Binary	
001	1	= 2**0
002	10	= 2**1
003	11	
004	100	= 2**2
005	101	
006	110	
007	111	
008	1000	= 2**3
009	1001	
010	1010	
011	1011	
012	1100	
013	1101	
014	1110	
015	1111	
016	10000	= 2**4
…		
099		
100		
…		
256	100000000	= 2**8

2**n in binary is: 1 followed by n zeros. 2**15 is 32768 (in decimal).

n is called the *logarithm* of 2**n. The logarithm of 32768 = 2**15 is 15.

Logarithmic algorithm to calculate b**c, for c >= 0
(i.e. b multiplied by itself c times)

```
/** 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 }
```

The algorithm looks at the binary representation of y.

- Testing if y is even means testing whether it rightmost bit is 0.
- y= y/2; is done by deleting the rightmost bit.
- y= y–1; in the algorithm is done by changing the rightmost bit from 1 to 0.

Logarithmic algorithm to calculate b**c, for c >= 0 (i.e. b multiplied by itself c times)

```
/** 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 }
```

The algorithm is

“logarithmic in c”

which means that if c = 2**k, it takes time proportional to k

E.g. if c = 2**15, i.e. 32768, loop

takes at most 2*15 + 1 iterations!