Loops, part 1
or, more fun with booleans

Round and around and around and around.
– Chubby Checker, 1960, “The Twist”

Example “for” loop

• static void drawStars(int nStars) {
  •   for (int i=1; i <= nStars; ++i) {
    •     System.out.print('*');
    •   }
  •   System.out.println();
  • }

Pieces of the for loop (1)

• for (int i=1; i <= nStars; ++i) {
  •   do this some number of times
  • }

• Purpose: repeat the part in { ... } some number of times.
• Often a “counting” or “definite” loop.

Pieces of the for loop (2)

• for (int i=1; i <= nStars; ++i) {
  •   // create a counter
  •   // keep going?
  •   // next counter value
  •   
  • }

• do this some number of times
• }
for (int i=1; 
i <= nStars;
++i)
{
  System.out.print("*");
}

Some Typical Loop Patterns

- **Do to each** ...
  - Example: Print each integer from 1 to n
- **Accumulate** ...
  - Example: Sum a set of integers
- **Select**
  - Example: Print longest word in dictionary
- **Filter**
  - Example: Print dictionary words with length > 7

Do to each ...

/** Print x^2 for x in [min, max] */
static void printSquares (int min, int max) {
  for (int i= min; i <= max; ++i) {
    int sq = i * i;
    System.out.println( i + ": " + sq);
  }
}

Accumulate ...

/** Sum integers in range [min,max] */
static int sumIntRange(int min, int max) {
  int sum = 0;
  for (int i=min; i <= max; ++i) {
    sum = sum + i;
  }
  return sum;
}
Select

- Example: Print longest word in dictionary
- Pseudocode:
  longest = ""; // anything should be longer
  for (each word in dictionary) {
    if (word is longer than longest) {
      longest = word;
    }
  }

What does this print?

for (int rep=1; rep <= 10; ++rep) {
  for (int m=1; m <= 10; ++m) {
    for (int n=1; n <= m; ++n) {
      System.out.print("*");
    }
    System.out.println();
  }
}

Filter

- Example: Print dictionary words with length > 7
- Pseudocode:
  for (each word in dictionary) {
    if (length of word > 7) {
      print word;
    }
  }

Prospective: Control Flow

- Power tools: We’re moving into algorithm design
  - We can make beautiful furniture, or cut off our hands
- We’ll spend a little effort on syntax, and a lot of effort on how to design and reason about loops and conditions
Successive Approximation

- Another important pattern in loops:
  
  Make a rough guess
  While (not good enough) {
    Improve the guess
  }

  // The guess is now the answer

Square root by successive approximation

- We know \( \sqrt{x} \) is between 0 and x
  - Our initial guess: Half way between. (Bad, but good enough to start)

- Improving the guess:
  - If our guess is too large or too small, we can reduce the range of possibilities

- We will quickly converge on a pretty good estimate

(Using integers to keep the numbers simple)

Looking for \( \sqrt{81} \)
0  \( \downarrow \)  Guess 40  81

Too big!
(And anything bigger than 40 is also too big)

Looking for \( \sqrt{81} \)
0  \( \downarrow \)  Guess 40  81

Looking for \( \sqrt{81} \)
0  \( \downarrow \)  Guess 20  40

Still too big!
Looking for $\sqrt{81}$

Guess 40

0

$\triangleleft$

40

0

$\triangleleft$

20

Too big

0

$\triangleleft$

10

Too small

5

$\triangleleft$

10

Too small

Recall the pattern ...

Make a rough guess
While (not good enough) {
    Improve the guess
}

// The guess is now the answer

Yippee!
Parts of the pattern

Make a rough guess
We guess root(81) is half way between 0 and 81
While (not good enough) {
The range we are guessing in is still too large
Improve the guess
Narrow the range of possibilities
}

// The guess is now the answer

Root finding in Java

• Doing it with floating point numbers, so ...
• Don’t expect an exact answer
  • type “double” is just a good approximation of real numbers
• We can set an error bound for our answer
  • the bound can be very small, but must not be zero
  • what if we chose an error bound of zero? what goes wrong?

In Java ...

double low = 0.0;  // Low end of guess range
double high = x;  // High end of guess range
double guess = x / 2.0;  // See why this is needed?

while (high - low > ERROR_BOUND) {  // Close enough?
  guess = (high + low) / 2.0;
  if (guess * guess > x) {  // Too high
    high = guess;
  } else if (guess * guess < x) {  // Too low
    low = guess;
  }
}

A classic algorithm ...

• The root algorithm is a “binary search”
  – One of the classic algorithms of computer science.
  We’ll see many variations on the theme, in this class and later.
• Efficient!
  – What can we say about how long it takes the binary search to narrow down the answer to within a given error bound?