CSCI 1103: Loops

Chris Kauffman

Last Updated:
Mon Oct 2 17:19:46 CDT 2017
Logistics

Reading
Eck Ch 3 on loops (while/for)

Goals
- Finish up conditionals
- Loop basics
- while()
- for()
- Nesting loops

Project 2
- Now Posted
- Conditionals and loops
- Array problem
  (Monday Lecture)

Lab04: Loops
Will cover what we’ve been up to with while and for
Backwards Branching

- Conditionals allow forward branching
- Loops allow backward branching
  - Repetition, iteration
- With the addition of loops, have a Turing Complete language
  - Anything that can be computed, can be computed with loops (!)
  - Most general programming languages are Turing Complete but add lots of conveniences for humans
- Java has a several kinds of loops
  - while()
  - for()
  - do/while()
while(): The Simple Loop

- while() is the simplest looping structure
- Repeatedly do something until given condition is false
- Condition is any boolean value, often numeric checks

```c
while(this condition is true){
    do this;
    and this;
    and this other thing;
    probably change condition;
    do this too;
}
```

do this once;
Simple while() loops

```java
1 public class SimpleWhile{
2   public static void main(String args[]){
3       System.out.println("Loop for 10 iterations");
4       int i = 0;
5       while (i < 10) {
6           System.out.printf("i is %d\n", i);
7           i = i + 1;
8           // or i += 1;
9           // or i++;
10       }
11       System.out.println("Done with loop");
12   }
13 }
```

Worthwhile now to introduce "op-equals" operators: get a lot of use in loops

<table>
<thead>
<tr>
<th>Op-Eq</th>
<th>Means</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>x += 5;</td>
<td>add 5 onto x</td>
<td>x = x + 5;</td>
</tr>
<tr>
<td>x -= 4;</td>
<td>subtract 5 from x</td>
<td>x = x - 5;</td>
</tr>
<tr>
<td>x *= 3;</td>
<td>triple x</td>
<td>x = x * 3;</td>
</tr>
<tr>
<td>x /= 2;</td>
<td>halve x</td>
<td>x = x / 2;</td>
</tr>
<tr>
<td>x++;</td>
<td>increment x</td>
<td>x = x + 1;</td>
</tr>
<tr>
<td>x--;</td>
<td>decrement x</td>
<td>x = x - 1;</td>
</tr>
</tbody>
</table>
Understanding Memory in Loops

Understanding flow of execution through code now requires a very good mental model of working memory

```java
public class DemoLoop{
    public static void main(String args[])
    {
        int x = 0;
        int a = 5;
        while(x != a){
            x++;
            a = a/2 + 1;
        }
        System.out.println("x: "+x);
        System.out.println("a: "+a);
    }
}
```

1st ITER
CPU: Line 7
| Box | Value |
|-----+-------|
| x   | 0     |
| a   | 5     |

CPU: Line 8
| Box | Value |
|-----+-------|
| x   | 0     |
| a   | 5     |

CPU: Line 9-10
| Box | Value |
|-----+-------|
| x   | 1     |
| a   | 5->3  |

2nd ITER
CPU: Line 7
| Box | Value |
|-----+-------|
| x   | 1     |
| a   | 3     |

CPU: Line 8
| Box | Value |
|-----+-------|
| x   | 1     |
| a   | 3     |

CPU: Line 9-10
| Box | Value |
|-----+-------|
| x   | 2     |
| a   | 3->2  |

3rd ITER -> Condition no longer true
CPU: Line 7
| Box | Value |
|-----+-------|
| x   | 2     |

CPU: Line 11
| Box | Value |
|-----+-------|
| x   | 2     |

SCREEN
| Box | Value |
|-----+-------|
| x   | 2     |
| a   | 2     |
Exercise: Show Loop Output

- Show output of the adjacent loops
- Describe what they are doing in English

```java
public class LoopExercises{
    public static void main(String args[]){
        // LOOP A
        int n = 18;
        int mf = n-1;
        while(n % mf != 0){
            mf--;
        }
        System.out.printf("n: %d mf: %d\n", n,mf);

        // LOOP B
        int p = 1;
        int e = 0;
        while(e < 10){
            System.out.printf("2^%d = %d\n",e,p);
            p *= 2;
            e++;
        }
    }
}
```
Answer: Show Loop Output

Loop A finds maximum factor of 18 (largest int that evenly divides 18)

n: 18 mf: 9

Loop B prints powers of 2

\[
\begin{align*}
2^0 &= 1 \\
2^1 &= 2 \\
2^2 &= 4 \\
2^3 &= 8 \\
2^4 &= 16 \\
2^5 &= 32 \\
2^6 &= 64 \\
2^7 &= 128 \\
2^8 &= 256 \\
2^9 &= 512
\end{align*}
\]

```java
public class LoopExercises{
    public static void main(String args[]){
        // LOOP A
        int n = 18;
        int mf = n-1;
        while(n % mf != 0){
            mf--;
        }
        System.out.printf("n: %d mf: %d\n", n,mf);
        // LOOP B
        int p = 1;
        int e = 0;
        while(e < 10){
            System.out.printf("2^%d = %d\n", e,p);
            p *= 2;
            e++;
        }
    }
}
```
Exercise: Immediate Trouble

```
1  public class LongLoop{
2      public static void main(String args[]){
3
4          int twoPow = 1;
5          int exponent = 0;
6          while(exponent < 10){
7              System.out.printf("2^%d = %d
",
8                                                  exponent,twoPow);
9              twoPow *= 2;
10          }
11      }
12  }
```

- Show the output of the following loop
- Does it behave strangely? If so why and any suggested corrections?
Answer: Immediate Trouble

```java
1  public class LongLoop{
2     public static void main(String args[]){
3
4         int twoPow = 1;
5         int exponent = 0;
6         while(exponent < 10){
7             System.out.printf("2^%d = %d\n", exponen
twoPow);
8             exponent = twoPow;
9             twoPow *= 2;
10         }
11     }
12 }
```

- This is an infinite loop
- Will run forever
- Forgot to increment exponent each loop iteration
Classic Exercise: Integer Exponentiator

▶ Prompt for a base and power
▶ Use a while() loop to compute exponentiated number

Start with

```java
public class Exponentiator{
    public static void main(String args[]){
        System.out.println("Enter base (int): ");
        int base = TextIO.getInt();
        System.out.println("Enter power (int): ");
        int power = TextIO.getInt();
        // ADDITIONAL CODE BELOW
    }
}
```

> javac Exponentiator.java

> java Exponentiator
Enter base (int):
2
Enter power (int):
5
2^5 is 32

> java Exponentiator
Enter base (int):
3
Enter power (int):
8
3^8 is 6561

> java Exponentiator
Enter base (int):
9
Enter power (int):
4
9^4 is 6561
public class Exponentiator {
    public static void main(String args[]) {
        System.out.println("Enter base (int): ");
        int base = TextIO.getInt();
        System.out.println("Enter power (int): ");
        int power = TextIO.getInt();

        int curVal = 1;
        int curPow = 0;

        while (curPow < power) {
            curVal *= base;  // curVal = curVal * base;
            curPow++;
        }
        System.out.printf("%d^%d is %d
",
                base, power, curVal);
    }
}
Loops and Conditionals

Can nest loops in conditions and vice-versa

**Condition in Loop**

```plaintext
while(num != ans){
    num++;
    if(num > limit){
        num = 0;
    }
    else if(num < 0){
        num = ans;
    }
}
```

**Loop inside Conditional**

```plaintext
if(x>0 && y==7){
    while(z > 5){
        z /= 2;
    }
}
else{
    while(q != 0){
        q = q % 3;
    }
}
```

Above code is syntax only, does nonsense. Opens up lots of possibilities though...
User Input in Loops

Common to provide a user input loop for interactive experiences

```java
// Demonstrate how user input can affect loops
public class InteractiveSum{
    public static
    void main(String args[])
    {
        System.out.println("How many integers?");
        int count = TextIO.getInt(); // number of iterations
        int i = 0;
        int sum = 0;
        while(i < count){ // loop until reach count
            System.out.println("Enter an integer:");
            int value = TextIO.getInt();
            sum += value; // update sum
            i++;
        }
        System.out.printf("Sum is %d\n",sum);
    }
}
```

- Used fixed # iterations
- Could use different method for this
Original Code

```java
// Demonstrate how user input can affect loops
public class InteractiveSum{
    public static void main(String args[])
    {
        System.out.println("How many integers?");
        int count = TextIO.getInt();// number of iterations
        int i = 0;
        int sum = 0;
        while(i < count){ // loop until reach count
            System.out.println("Enter an integer:");
            int value = TextIO.getInt(); // update sum
            sum += value;
            i++;
        }
        System.out.printf("Sum is %d\n",sum);
    }
}
```

Modify this to produce the interactions shown

- Don’t ask for iterations up front
- Terminate when 0 is entered
- Will need a condition in loop
Answer: User Input with "Quit" Value

- Note use of 0 as quit value: Magic Constant
- Would be clearer to do:

```java
int quit = 0;
while(value != quit){

    // Demonstrate interactive loop with a "quit" value which causes the
    // loop to terminate
    public class InteractiveSum2{
        public static
        void main(String args[]){
            int i = 0;
            int sum = 0;
            int value = -1;  // Must declare before loop
            while(value != 0){  // Check for quit value
                System.out.println("Enter an integer (0 to quit): ");
                value = TextIO.getInt();
                if(value != 0){  // Don’t do anything with
                    sum += value;  // quit value
                }
            }
            System.out.printf("Sum is %d\n",sum);  // After loop
        }
    }
}
```

**Variant:** Modify the above solution to use a quit value is -1. Is your solution easy to modify using quit of -1?
for() loops

In many cases loops follow a regular pattern:

- Start at some counter value (like \( i=0 \))
- Loop until value goes out of bounds (like \( i < \text{count} \))
- At the end of each iteration, move counter forward (like \( i++ \))

So common that there is a special for() syntax for it.

Example: Loop 0 to 19

```java
int count = 20;
for(int i=0; i<count; i++){
    System.out.println("Looping");
    System.out.printf("i is %d\n",i);
}
```

Output

```
Looping
i is 0
Looping
i is 1
Looping
i is 2 ...
```

General for() Syntax

```java
for(initialize; condition; update){
    do some stuff repeatedly;
    and some other stuff repeatedly;
}
then do this;
```

Start Elsewhere, Double

```java
for(int twoPow=1; twoPow<1024; twoPow*=2){
    System.out.printf("%d ",twoPow);
}
// Prints: 1 2 4 8 16 32 64 128 256 512
```
Equivalence of for() and while()

- Loop types are interchangeable if you know what you are doing.
- Typical parts are arranged as follows.

```java
initialize; for(initialize; condition; update){
    while(condition){
        body;
    }
}
```

**Example: Equivalent Powers of 2**

```java
System.out.println("WHILE LOOP VERSION");
twoPow=1;
exponent=0;
while(exponent < 10){
    System.out.printf("2^%d = %d
", exponent, twoPow);
    twoPow *= 2;
    exponent++;
}
```

```java
System.out.println("FOR LOOP VERSION");
twoPow=1;
for(exponent=0; exponent < 10; exponent++){
    System.out.printf("2^%d = %d\n", exponent, twoPow);
    twoPow *= 2;
}
```
Exercise: Easy Exam Questions to Write

**Convert to for**

```java
double tol = 1e-4;
double S = 45.0;
double x = 45.0/2;
double err;

err = (S - x*x)*(S - x*x);
while(err > tol){
    x = (x + S/x) / 2.0;
    err = (S - x*x)*(S - x*x);
}
```

**Convert to while**

```java
int x = 48;
int f = -1;
boolean found = false;

for(int i=x-1; i>1 && !found; i--){
    if(x % i == 0){
        f = i;
        found = true;
    }
}
```

**Answers in code pack**
Nested Loops

- Like conditionals, loops can be nested
- Often done with nested for() loops to create tabular output

Table.java Code

```java
// Prints a small table of numbers
public class Table{
    public static void main(String args[]){
        int rows=5, cols=7;
        System.out.printf("%d by %d table\n", rows, cols);

        // OUTER LOOP: print a whole
        // row per iteration
        for(int i=0; i<rows; i++){
            // Print at beginning of row
            System.out.printf("Row %d : ",i);

            // INNER LOOP: Print rest of row,  // 1 iteration per column
            for(int j=0; j<cols; j++){
                System.out.printf("%d",i,j);
            }
        }

        // Print end of row
        System.out.printf(" : done\n");
    }
}
```

Output

```bash
> javac Table.java
> java Table
5 by 7 table
Row 0 : 00 01 02 03 04 05 06 : done
Row 1 : 10 11 12 13 14 15 16 : done
Row 2 : 20 21 22 23 24 25 26 : done
Row 3 : 30 31 32 33 34 35 36 : done
Row 4 : 40 41 42 43 44 45 46 : done
```

- Very common to use i for outer loop and j for inner loop counters
- Note position of printing start/end of each row
- Lab/Project have table problems
Note on Scoping

- All names in Java have a **scope**
- Dictates the parts of code in which name is visible and usable
- Easiest scope rule: declare variables before use

  ```java
  // YES                          // NO!
  double x = 5;                 System.out.println(x);
  System.out.println(x);        double x = 5;
  ```

- Every set of curly braces `{ stuff }` sets up a scope:
  - Variables declared before `{` are visible inside
  - Variables declared inside `{` go **out of scope** at `}`

- Common problems involve declaring variables inside a scope then trying to use it outside of that scope

- Java compiler reports this as

  ```java
  error: cannot find symbol
  ```
Common Scope Errors

// GOOD
int x = 0;
if(condition){
    x = 10;
}
System.out.println(x);

int i;
for(i=0; i<5; i++){
    blah blah;
}
System.out.println(i);

int v=10;
for(int i=0; i<5; i++){  
    if(condition){
        v *= 2;
    }
}
System.out.println(v);

// ERROR
if(condition){
    int x = 10;
}
System.out.println(x);

for(int i=0; i<5; i++){  
    blah blah;
}
System.out.println(i);

for(int i=0; i<5; i++){  
    if(condition){
        v *= 2;
    }
}
System.out.println(v);
Exercise: Fix this Scope problem

Code: Max of 5

```java
1 // Find the max of 5 numbers given by
2 // user using a loop
3 //
4 // This program has a scope problem
5 public class ScopeProblem{
6    public static void main(String args[]){
7        System.out.println("Enter 5 numbers:");
8        for(int i=0; i<5; i++){
9            int max = 0;
10           int value = TextIO.getInt();
11           if( value > max ){
12               max = value;
13           }
14        }
15        System.out.printf("max is %d\n",max);
16    }
17 }
```

Compiler Errors

Note that the error below directs attention to line 15 via

```
ScopeProblem.java:15: error:>
```

```bash
> javac ScopeProblem.java
ScopeProblem.java:12: error: cannot find symbol
    System.out.printf("max is %d\n",max);
        ^
symbol: variable max
location: class ScopeProblem
1 error
```

One answer in code pack as Max5Loop.java
Numerical Loops: Square Root

- Initial purpose of computers: crunch numbers
- "Computers" were humans that crunched numbers, WWII era artillery tables, mostly women with Math backgrounds
- Gradually replaced by machines which were faster, cheaper, more accurate

An al-Khwārizmī for Square Roots

The Babylonian Algorithm to computer the square root of $S$

- Initialize: Set $x$ to a guess
- Repeat
  - Calculate $x_{next} = \frac{1}{2} \left( x + \frac{S}{x} \right)$
  - Set $x$ to be $x_{next}$
  - If $x^2$ is close enough to $S$, quit

Let’s see if it works, try calculating $\sqrt{18}$
Computing $\sqrt{18}$

\[
s = 18 \quad \# \text{Find my square root}
\]
\[
x = 4 \quad \# \text{A guess}
\]
\[
\# \text{Repeat these steps}
\]
\[
x_{\text{next}} = \left(\frac{1}{2}\right) \times (x + \frac{18}{x})
\]
\[
x = x_{\text{next}}
\]
\[
x
\]
\[
4.25000000000000000000
\]
\[
x^2 - 18
\]
\[
.06250000000000000000
\]
\[
\# \text{Pretty close, but can we get closer?}
\]
\[
x_{\text{next}} = \left(\frac{1}{2}\right) \times (x + \frac{18}{x})
\]
\[
x = x_{\text{next}}
\]
\[
4.24264705882352941176
\]
\[
x = x_{\text{next}}
\]
\[
x^2 - 18
\]
\[
.00005406574394463663
\]
al-Khwa-what?

Abū ʿAbdallāh Muḥammad ibn Mūsā al-Khwārizmī (780-850 AD)

► Say that 5 times fast
► Well, *algorithm* is close enough

In the twelfth century, Latin translations of his work on the Indian numerals introduced the decimal positional number system to the Western world. His *Compendious Book on Calculation by Completion and Balancing* presented the first systematic solution of linear and quadratic equations in Arabic. In Renaissance Europe, he was considered the original inventor of *algebra*, although it is now known that his work is based on older Indian or Greek sources.

► Wikipedia
break statements

break;: immediately jump outside the current loop

- Often used in conjunction with seemingly infinite loops
- Avoid when possible but use when it makes sense
- Not needed for Lab04 or Project 2

1 // Demonstrate interactive loop with a "quit" value which causes the
2 // loop to terminate; a break statement is used for this
3 public class SumBreak{
4   public static
5   void main(String args[]){
6     int i = 0;
7     int sum = 0;
8     int quitVal = -1;
9     while(true){ // Apparently loop forever...
10       System.out.printf("Enter an integer (%d to quit):
", quitVal);
11       int value = TextIO.getInt();
12       if(value == quitVal){ // Check for quit val
13         break; // jump out of loop
14       }
15       sum += value;
16     }
17     System.out.printf("Sum is %d\n",sum); // After loop
18   }
19 }
20 }
Looping Potpourri

Various other loop capabilities exist in Java which deserve mention but not much attention.

```java
for (el : collection)
    ▶ For-Each Loop
    ▶ Iterate over each elements in a collection,
    ▶ Automatically bounds sets up loop variable, bounds
    ▶ Will talk about this later with arrays and ArrayList

continue;
    ▶ Skip remainder of loop body, do update,
    ▶ Start back at the beginning

Labeled break, continue
    ▶ Direct jumping to specific portions of the code,
    ▶ Similar to goto,
    ▶ Generally hard to read
    ▶ Used only when computing efficiency is the driving force
```

do/while()

Do a single iteration, then check the loop condition
Classic Exercise: Guessing Games

Approach

- Set up secret = 42
- Check user input in loop
- Way Too Big when > secret + 10
- Little Big when > secret
- Way Too Small when < secret - 10
- Little Small when < secret
- End when guess is correct
- Print # of guesses

Start your code

```java
public class GuessingGame {
    public static void main(String args[]) {
        int secret = 42;
        int nGuesses = 0;
        int guess = -1;
        System.out.println("Guess between 1 and 100:");
```
Answer: Guessing Games

// Demonstrate how user input can affect loops
public class GuessingGame{
    public static
    void main(String args[]){
        int secret = 42;
        int nGuesses = 0;
        int guess = -1;
        System.out.println("Guess between 1 and 100:");
        while(guess != secret){
            guess = TextIO.getInt();
            nGuesses++;
            if(guess > secret +10){
                System.out.println("Way too big");
            }
            else if(guess > secret){
                System.out.println("A little too big");
            }
            else if(guess < secret-10){
                System.out.println("Way too small");
            }
            else if(guess < secret){
                System.out.println("A little too small");
            }
        }
        System.out.println("Correct! The secret number is "+secret);
        System.out.printf("It took you %d guesses\n",nGuesses);
    }
}