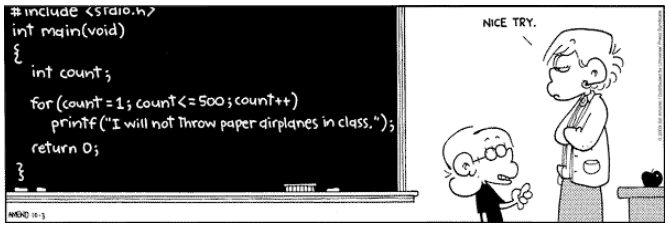
Lecture 3: Loops



Copyright 2004, FoxTrot by Bill Amend, http://www.ucomics.com/foxtrot/2003/10/03

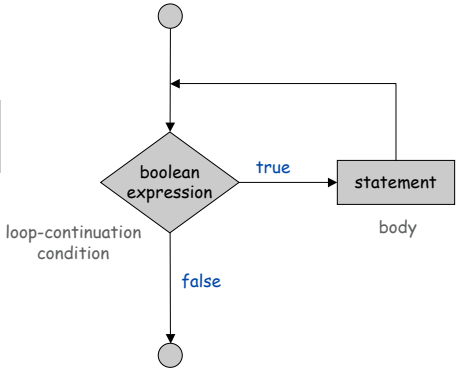
While Loops

The `while` loop. A common repetition structure.

- Check loop-continuation condition.
- Execute a sequence of statements.
- Repeat.

```
while (boolean expression)
   statement;
```

while loop syntax



while loop flow chart

While Loops: Powers of Two

Ex. Print powers of 2.

- Increment `i` from 1 to 6 by 1.
- Double `N` each time.

```
int i = 0;
int N = 1;
while (i <= 6) {
   System.out.println(N);
   i = i + 1;
   N = 2 * N;
}
```

a block statement

i	N	i <= 6
0	1	true
1	2	true
2	4	true
3	8	true
4	16	true
5	32	true
6	64	true
7	128	false

```
% java Powers
1
2
4
8
16
32
64
```

Click for demo

While Loops: Newton-Raphson Method

Q. How might we implement `Math.sqrt()` ?

A. To compute the square root of c:

- Initialize t = c.
- Replace t with the average of t and c / t, and repeat until t = c / t, up to desired precision.

```
public class Sqrt {
   public static void main(String[] args) {
      double EPS = 1E-15;
      double c = Double.parseDouble(args[0]);
      double t = c;
      while (Math.abs(t - c/t) > t*EPS) {
         t = (c/t + t) / 2.0;
      }
      System.out.println(t);
   }
}
```

error tolerance

```
% java Sqrt 2.0
1.414213562373095
```

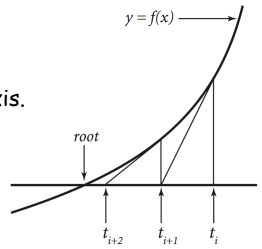
15 decimal digits of accuracy in 5 iterations

While Loops: Newton-Raphson Method

Newton-Raphson method explained.

- Goal: find root of function f(x).
- Start with estimate t_0.
- Draw line tangent to curve at x= t_i.
- Set t_{i+1} to be x-coordinate where line hits x-axis.
- Repeat until desired precision.

f(x) = x^2 - c



Applications and extensions.

- Find roots of a differentiable function.
- Optimize a twice differentiable function.

of one or several variables

check where derivative is zero

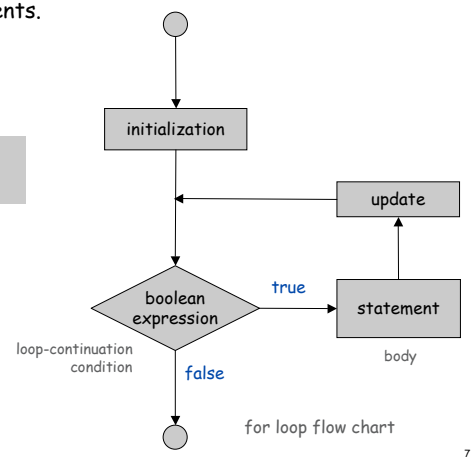
For Loops

The `for` loop. Another common repetition structure.

- Initialize variable.
- Check loop-continuation condition.
- Execute sequence of statements.
- Increment variable.
- Repeat.

```
for (init; boolean; update)
   statement;
```

for loop syntax



for loop flow chart

For Loops: Subdivisions of a Ruler

Create subdivision of a ruler.

- Initialize ruler to empty string.
- For each value i = 1 to N.
- Sandwich two copies of the ruler on either side of i.

```
int N = 3;
String ruler = " ";
for (int i = 1; i <= N; i++) {
   ruler = ruler + i + ruler;
}
System.out.println(ruler);
```

i	ruler
	" "
1	" 1 "
2	" 1 **2** 1 "
3	" 1 2 1 **3** 1 2 1 "

For Loops: Subdivisions of a Ruler

Observation.

- Program produces $2^N - 1$ integers.
- Loops can produce a huge amount of output!

```
% java Ruler 1
1

% java Ruler 2
1 2 1

% java Ruler 3
1 2 1 3 1 2 1

% java Ruler 4
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

% java Ruler 100
Exception in thread "main"
java.lang.OutOfMemoryError
```

Nesting Conditionals and Loops

Conditionals enable you to do one of 2^n sequences of operations with n lines.

```
if (a0 > 0) System.out.print(0);
if (a1 > 0) System.out.print(1);
if (a2 > 0) System.out.print(2);
if (a3 > 0) System.out.print(3);
if (a4 > 0) System.out.print(4);
if (a5 > 0) System.out.print(5);
if (a6 > 0) System.out.print(6);
if (a7 > 0) System.out.print(7);
if (a8 > 0) System.out.print(8);
if (a9 > 0) System.out.print(9);
```

2^{10} = 1024 possible results, depending on input

Loops enable you to do an operation n times using only 2 lines of code.

```
double sum = 0.0;
for (int i = 1; i <= 1024; i++)
   sum = sum + 1.0 / i;
```

computes 1/1 + 1/2 + ... + 1/1024

More sophisticated programs.

- Nest conditionals within conditionals.
- Nest loops within loops.
- Nest conditionals within loops within loops.

Nested If-Else

Ex. Pay a certain tax rate depending on income level.

Income	Rate
0 - 47,450	22%
47,450 - 114,650	25%
114,650 - 174,700	28%
174,700 - 311,950	33%
311,950 -	35%

```
double rate;
if      (income <  47450) rate = 0.22;
else if (income < 114650) rate = 0.25;
else if (income < 174700) rate = 0.28;
else if (income < 311950) rate = 0.33;
else                      rate = 0.35;
```

graduated income tax calculation

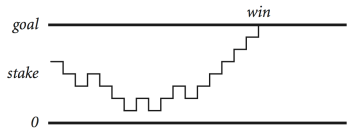
Gambler's Ruin

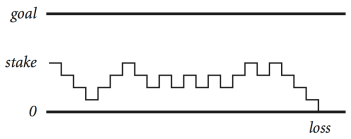
Gambler's ruin. Gambler starts with $stake and places $1 even bets until going broke or reaching $goal.

- What are the chances of winning?
- How many bets will it take?

One approach. Numerical simulation.

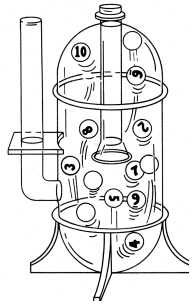
- Flip digital coins and see what happens.
- Repeat and compute statistics.





Library Functions: Math.random

`Math.random()` returns numbers between 0 and 1.



Q. How is `Math.random()` implemented?

- Linear feedback shift register? Cosmic rays?
- User doesn't need to know details.
- User doesn't want to know details.

Caveats.

- "Random" numbers are not really random.
- Don't use for crypto or Internet gambling!
- Check assumptions about library function before using.

Gambler's Ruin

```
public class Gambler {
   public static void main(String[] args) {
      int stake = Integer.parseInt(args[0]);
      int goal  = Integer.parseInt(args[1]);
      int N     = Integer.parseInt(args[2]);
      int wins  = 0;
      // repeat simulation N times
      for (int i = 0; i < N; i++) {
         // do gambler's ruin simulation
         int t = stake;
         while (t > 0 && t < goal) {
            // flip coin and update
            if (Math.random() < 0.5) t++;
            else                     t--;
         }
         if (t == goal) wins++;
      }
      System.out.println(wins + " wins of " + N);
   }
}
```


Simulation and Analysis

stake goal N

```
% java Gambler 10 20 1000
513 wins of 1000

% java Gambler 10 20 1000
492 wins of 1000

% java Gambler 500 2500 100
24 wins of 100
```

after a few minutes of computing....

Fact. Probability of winning = stake ÷ goal.
Fact. Expected number of bets = stake × desired gain.
Ex. 20% chance of turning $500 into $2500, but expect to make one million $1 bets.

Remark. Both facts can be proved mathematically; for more complex scenarios, computer simulation is often the best plan of attack.

Debugging a Program: Syntax Errors

Factor. Given an integer N, compute its prime factorization.
Application. Break RSA cryptosystem.

$168 = 2^3 \times 3 \times 7$

Syntax error. Illegal Java program.
- Compiler error messages help locate problem.
- Eventually, a file named `Factors.class`.

```
public class Factors1 {
   public static void main(String[] args) {
      long N = Long.parseLong(args[0])
      for (i = 0; i < N; i++) {
         while (N % i == 0)
            System.out.print(i + " ")
            N = N / i
      }
   }
}
```

Does not compile

Check if i is a factor.

As long as i is a factor, divide it out.

Debugging a Program: Semantic Errors

Semantic error. Legal but wrong Java program.
- Use "`System.out.println`" method to identify problem.

```
public class Factors2 {
   public static void main(String[] args) {
      long N = Long.parseLong(args[0]);
      for (long i = 2; i < N; i++) {
         while (N % i == 0)
            System.out.print(i + " ");
            N = N / i;
      }
   }
}
```

No output (17) or infinite loop (49)

Check if i is a factor.

As long as i is a factor, divide it out.

Debugging a Program: Performance Errors

Performance bug. Correct program but too slow.

- Use profiling to discover bottleneck.
- Devise better algorithm.

```
public class Factors3 {
   public static void main(String[] args) {
      long N = Long.parseLong(args[0]);
      for (long i = 2; i <= N; i++) {
         while (N % i == 0) {
            System.out.print(i + " ");
            N = N / i;
         }
      }
   }
}
```

Too slow for large N (999,999,937)

Check if i is a factor.

As long as i is a factor, divide it out.

Debugging a Program: Success

Fact. If N has a factor, it has one less than or equal to its square root.
Impact. Many fewer iterations of `for` loop.

Check if i is a factor.

```
public class Factors {
   public static void main(String[] args) {
      long N = Long.parseLong(args[0]);
      for (long i = 2; i <= N/i; i++) {
         while (N % i == 0) {
            System.out.print(i + " ");
            N = N / i;
         }
      }
      if (N > 1) System.out.println(N);
      else       System.out.println();
   }
}
```

As long as i is a factor, divide it out.

Corner case: biggest factor occurs once.

Debugging a Program: Trace

```
% java Factors 3757208
2 2 2 7 13 13 397
```

i	N	output
2	3757208	2 2 2
3	469651	
4	469651	
5	469651	
6	469651	
7	469651	7
8	67093	

i	N	output
9	67093	
10	67093	
11	67093	
12	67093	
13	67093	13 13
14	397	
15	397	

i	N	output
16	397	
17	397	
18	397	
19	397	
20	397	
		397

Debugging a Program: Analysis

Q. How big an integer can I factor?

```
% java Factors 168
2 2 2 3 7

% java Factors 3757208
2 2 2 7 13 13 397

% java Factors 9201111169755555703
9201111169755555703
```

after a few minutes of computing....

largest factor

Digits	i <= N	i <= N / i	i * i <= N
3	instant	instant	instant
6	0.15 seconds	instant	instant
9	77 seconds	instant	instant
12	21 hours †	0.21 seconds	0.16 seconds
15	2.4 years †	4.5 seconds	2.7 seconds
18	2.4 millennia †	157 seconds	92 seconds

† estimated

Programming in Java

Programming in Java. [a slightly more realistic view]

1. Write the program.

2. Compile the program.
   Compiler says: That's not a legal program.
   Back to step 1 to fix your errors of syntax.

3. Execute the program.
   Result is bizarrely (or subtly) wrong.
   Back to step 1 to fix your errors of semantics.

4. Enjoy the satisfaction of a working program!

Flow Of Control Summary

Flow of control.

- Sequence of statements that are actually executed in a program.

Flow-Of-Control	Description	Examples
Straight-line programs	All statements are executed in the order given.	
Conditionals	Certain statements are executed depending on the values of certain variables.	`if` `if-else`

Loops	Certain statements are executed repeatedly until certain conditions are met.	`while`
		`for`
		`do-while`

Conditionals and loops.

- Simple, but powerful tools.
- Enables us to harness power of the computer.