1.3 Conditionals and Loops

Conditionals and Loops

Control flow.
- Sequence of statements that are actually executed in a program.
- Conditionals and loops: enable us to choreograph control flow.

Statement 1
statement 2
statement 3
statement 4

Boolean 1
true
false

Boolean 2
true
false

Statement 1
Statement 2
Statement 3

Straight-line control flow
Control flow with conditionals and loops
If Statement

The **if** statement. A common branching structure.

- Evaluate a boolean expression.
- If **true**, execute some statements.
- **else** option: If **false**, execute other statements.

```java
if (x > y)
{
  int t = x;
  x = y;
  y = t;
}
```

**Ex.** Take different action depending on value of variable.

```java
public class Flip
{
  public static void main(String[] args)
  {
    if (Math.random() < 0.5)
      System.out.println("Heads");
    else System.out.println("Tails");
  }
}
```
The **while loop**. A common repetition structure.

- Check a boolean expression.
- Execute a sequence of statements.
- Repeat.

```java
while (boolean expression) {
    statement 1;
    statement 2;
}
```

**While Loop Example: Powers of Two**

Ex. Print powers of 2 that are ≤ 2^n.
- Increment i from 0 to n.
- Double v each time.

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

**Powers of Two (full program)**

```java
public class PowersOfTwo {
    public static void main(String[] args) {
        // last power of two to print
        int n = Integer.parseInt(args[0]);
        int i = 0; // loop control counter
        int v = 1; // current power of two
        while (i <= n)
        {
            System.out.println(v);
            i = i + 1;
            v = 2 * v;
        }
    }
}
```

**While Loop Challenge**

Anything wrong with the following code?

```java
public class PowersOfTwo {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int i = 0; // loop control counter
        int v = 1; // current power of two
        while (i <= N)
        {
            System.out.println(v);
            i = i + 1;
            v = 2 * v;
        }
    }
}
```
Goal. Implement Math.sqrt().

Newton-Raphson method to compute the square root of c:
• Initialize \( t_0 = c \).
• Repeat until \( t_i = c / t_i \), up to desired precision:
  set \( t_{i+1} \) to be the average of \( t_i \) and \( c / t_i \).

<table>
<thead>
<tr>
<th>i</th>
<th>( t_i )</th>
<th>( 2 / t_i )</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.0</td>
<td>1.0</td>
<td>1.5</td>
</tr>
<tr>
<td>1</td>
<td>1.5</td>
<td>1.3333333</td>
<td>1.4166667</td>
</tr>
<tr>
<td>2</td>
<td>1.4166667</td>
<td>1.4117647</td>
<td>1.4142157</td>
</tr>
<tr>
<td>3</td>
<td>1.4142157</td>
<td>1.4142114</td>
<td>1.4142136</td>
</tr>
<tr>
<td>4</td>
<td>1.4142136</td>
<td>1.4142136</td>
<td>1.4142135</td>
</tr>
</tbody>
</table>

computing the square root of 2 to seven places

Newton-Raphson Method

Square root method explained (some math omitted).
• 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.

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);
    }
}

% java Sqrt 2.0
1.414213562373095

15 decimal digits of accuracy in 5 iterations

The For Loop

public class Sqrt {
    public static void main(String[] args) {
        int main() {
            int count = 0;
            for (count = 0; count < 500; count++) {
                printf("I will not throw paper airplanes in class,");
            }
            return 0;
        }
    }
}

% include stdio.h
#include stdio.h
int main(void) {
    int count = 0;
    for (count = 0; count < 500; count++) {
        printf("I will not throw paper airplanes in class,");
    }
    return 0;
}
The For Loop

The for loop. Another common repetition structure.
- Execute initialization statement.
- Check boolean expression.
- Execute sequence of statements.
- Execute increment statement.
- Repeat.

```
for (init; boolean expression; increment)
{
    statement 1;
    statement 2;
}
```

Anatomy of a for Loop

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

Every for loop has an equivalent while loop

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

For Loops: Subdivisions of a Ruler

Create subdivision of a ruler.
- Initialize ruler to single space.
- For each value i from 1 to N:
  sandwich two copies of ruler on either side of i.

```
public class Ruler
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        String ruler = " ";
        for (int i = 1; i <= N; i++)
            ruler = ruler + i + ruler;
        System.out.println(ruler);
    }
}
```

Why for loops? Can provide more compact and understandable code.
For Loops: Subdivisions of a Ruler

```
% 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 5
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 5 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

```
% 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 5
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 5 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

---

Observation. Loops can produce a huge amount of output!

---

Nesting

```
Nesting Conditionals and Loops

Nesting. Use a conditional or a loop within a conditional or a loop

• Enables complex control flows.
• Adds to challenge of debugging.

Any "statement" within a conditional or loop
may itself be a conditional or a loop statement
```

```java
int v = 1;
while (v <= N/2)
  v = 2*v;
System.out.println(v);
```

print largest power of 2 less than or equal to N

```
for (int i = 0; i < trials; i++)
  {     int t = stake;
   while (t > 0 && t < goal)
    if (Math.random() < 0.5) t++;
    else                     t--;
   if (t == goal) wins++;
  }
```

if-else statement within a while loop within a for loop
### Nested If Statements

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

<table>
<thead>
<tr>
<th>Income</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 47,450</td>
<td>22%</td>
</tr>
<tr>
<td>47,450 - 114,650</td>
<td>25%</td>
</tr>
<tr>
<td>114,650 - 174,700</td>
<td>28%</td>
</tr>
<tr>
<td>174,700 - 311,950</td>
<td>33%</td>
</tr>
<tr>
<td>311,950 -</td>
<td>35%</td>
</tr>
</tbody>
</table>

5 mutually exclusive alternatives

### Nested If-Else Statements

Need all those braces? Not always:

```java
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;
```

is shorthand for

```java
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;
        }
    }
}
```

but BE CAREFUL when nesting if-else statements (see Q&A p. 75).

### Nested If Statement Challenge

Anything wrong with the following code?

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

### Nested for loops

Ex. Visit each location in a two-dimensional table (stay tuned for arrays).

```java
for (x = 0; x < N; x++)
    for (y = 0; y < M; y++)
        Do something at entry (x,y);
```
Nesting Example: Gambler’s Ruin

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

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

One approach. Monte Carlo simulation.

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

```
public class Gambler {
    public static void main(String[] args) {
        // Get parameters from command line.
        int stake = Integer.parseInt(args[0]);
        int goal = Integer.parseInt(args[1]);
        int trials = Integer.parseInt(args[2]);

        int wins = 0;
        for (int i = 0; i < trials; i++) {
            // Do one gambler's ruin experiment.
            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 " + trials);
        }
    }
}
```

Digression: Simulation and Analysis

```
% java Gambler 5 25 1000
191 wins of 1000

% java Gambler 5 25 1000
203 wins of 1000

% java Gambler 500 2500 1000
197 wins of 1000
```

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
Debugging Example

**Factor.** Given an integer \( N > 1 \), compute its prime factorization.

\[ 3,757,208 = 2^3 \times 7 \times 13^2 \times 397 \]

\[ 98 = 2 \times 7^2 \]

\[ 17 = 17 \]

11,111,111,111,111,111 = 2,071,723 \times 5,363,222,357


Debugging: Syntax Errors

**Syntax error.** Illegal Java program.
- Compiler error messages help locate problem.
- Goal: no errors and a file named `Factors.class`.

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

Debugging: 99% of Program Development

**Programming.** A process of finding and fixing mistakes.
- Compiler error messages help locate syntax errors.
- Run program to find semantic and performance errors.

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

This program has bugs!

Debugging: Syntax Errors

```java
% javac Factors.java
Factors.java:6: ';' expected
for (i = 2; i < N; i++)
  ^
1 error ← the FIRST error
```
Debugging: Syntax Errors

Syntax error. Illegal Java program.
- Compiler error messages help locate problem.
- Goal: no errors and a file named Factors.class.

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

Semantic error. Legal but wrong Java program.
- Run program to identify problem.
- Add print statements if needed.

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

% javac Factors.java
% java Factors 98
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Factors.main(Factors.java:8)

need to start at 2 since 0 and 1 cannot be factors
Debugging: Semantic Errors

Semantic error. Legal but wrong Java program.
• Run program to identify problem.
• Add print statements if needed.

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

Success? Program factors 98 = 2 7 7.
• Time to try it for other inputs.
• Add trace to find and fix (minor) problems.

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

Print out N after for loop (if it is not 1)
Debugging: Success?

Success? Program seems to work
• Add code for corner case, add comments.
• Remove trace to try larger inputs

```java
public class Factors {
    public static void main(String[] args) {
        long N = Long.parseLong(args[0]);
        for (int i = 2; i < N; i++) {
            // Check whether i is a factor.
            if (N % i == 0)
                System.out.print(i + " ");
        }
        System.out.print(N + " ");
    }
}
```

Debugging: Performance Errors

Performance error. Correct program, but too slow.
• Are all iterations of inner loop necessary?
• Improve or change underlying algorithm.

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

Fresh semantic error. Fast program (now), but new error.
• Was performance fix exactly right?
• Again, consider (possibly new) corner cases.

```java
public class Factors {
    public static void main(String[] args) {
        long N = Long.parseLong(args[0]);
        for (int i = 2; i < N; i++) {
            // Check whether i is a factor.
            while (N % i == 0)
                System.out.print(i + " ");
            if (N > 1) System.out.println(N);
            else System.out.println();
        }
    }
}
```
Debugging: Back to Semantic Errors!

**Fresh semantic error.** Fast program (now), but new error.
- Was performance fix exactly right?
- Again, consider (possibly new) corner cases.

```java
public class Factors {
    public static void main(String[] args) {
        long N = Long.parseLong(args[0]);
        for (int i = 2; i * i <= N; i++) {
            // Check whether i is a factor.
            while (N % i == 0) {
                // If so, print and divide.
                System.out.print(i + " ");
                N = N / i;
            }
        }
        if (N > 1) System.out.println(N);
    }
}
```

Execute loop body if \(i^2 \leq N\)

Q. How large an integer can I factor?

<table>
<thead>
<tr>
<th>digits</th>
<th>(i &lt; N)</th>
<th>((i^2 \leq N))</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>instant</td>
<td>instant</td>
</tr>
<tr>
<td>6</td>
<td>0.15 seconds</td>
<td>instant</td>
</tr>
<tr>
<td>9</td>
<td>77 seconds</td>
<td>instant</td>
</tr>
<tr>
<td>12</td>
<td>21 hours (^\dagger)</td>
<td>0.16 seconds</td>
</tr>
<tr>
<td>15</td>
<td>2.4 years (^\dagger)</td>
<td>2.7 seconds</td>
</tr>
<tr>
<td>18</td>
<td>2.4 millennia (^\dagger)</td>
<td>92 seconds</td>
</tr>
</tbody>
</table>

\(^\dagger\) estimated, using analytic number theory

Note. Can’t break RSA this way (experts are still trying)

Program Development: Analysis

99% of program development

**Debugging.** Cyclic process of editing, compiling, and fixing errors.
- Always a logical explanation.
- What would the machine do?
- Explain it to the teddy bear.

You will make many mistakes as you write programs. It’s normal.

“As soon as we started programming, we found out to our surprise that it wasn’t as easy to get programs right as we had thought. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs.”

Sir Maurice Wilkes

Good news: Can use computer to test program.

Bad news: Conditionals/loops open up huge number of possibilities.

Really bad news: Cannot use computer to automatically find all bugs. **Stay tuned**
## The First Bug?

![Photo of Lieutenant Grace Murray Hopper](http://www.history.navy.mil/photos/images/h96000/h96566kc.htm)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debugging bugs in the computer</td>
<td>Cleaning a 60-year-old fly with a mosquito aspirator</td>
</tr>
</tbody>
</table>

*Lieutenant Grace Murray Hopper*