1.3 Conditionals and Loops

A Foundation for Programming

any program you might want to write

objects
functions and modules
graphics, sound, and image I/O
arrays
conditionals and loops
Math
text I/O
primitive data types
assignment statements

Introduction to Programming in Java: An Interdisciplinary Approach · Robert Sedgewick and Kevin Wayne · Copyright © 2008 · February 04, 2008 10:00 AM

any program you might want to write

objects
functions and modules
graphics, sound, and image I/O
arrays
conditionals and loops
Math
text I/O
primitive data types
assignment statements

to infinity and beyond!
Conditionals and Loops

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

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.

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");
        }
    }
}
```

% java Flip
Heads
% java Flip
Heads
% java Flip
Tails
% java Flip
Heads
If Statement Examples

```java
if (x > 0) x = -x;
```

absolute value

```java
if (x > y) max = x;
else max = y;
```

maximum

```java
if (den == 0) System.out.println("Division by zero");
else System.out.println("Quotient = " + num/den);
```

error check for division operation

```java
double discriminant = b*b - 4.0*c;
if (discriminant < 0.0)
{
    System.out.println("No real roots");
}
else
{
    System.out.println((-b + Math.sqrt(discriminant))/2.0);
    System.out.println((-b - Math.sqrt(discriminant))/2.0);
}
```

everror check for quadratic formula

Loops

While Loop

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;
}
```

loop continuation condition

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

Ex. Print powers of 2 that are \( \leq 2^n \).
- Increment \( i \) from 0 to \( n \).
- Double \( v \) each time.
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;
        }
    }
}
```

Any mistakes 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;
        }
    }
}
```

### While Loop Example: Square Root

**Goal.** Implement `Math.sqrt()`.

Newton-Raphson method to compute the square root of `c`:

1. Initialize `t_0 = c`.
2. 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>c / 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>1.333333</td>
<td>1.416667</td>
</tr>
<tr>
<td>2</td>
<td>1.416667</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>

“A wonderful square root. Let’s hope it can be used for the good of mankind.”

Copyright 2004, Sidney Harris
http://www.sciencecartoonsplus.com

---

TEQ on While Loops

### While Loop Example: Square Root

**Goal.** Implement `Math.sqrt()`.

Newton-Raphson method to compute the square root of `c`:

1. Initialize `t_0 = c`.
2. 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`.

```java
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
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.

\[ f(x) = x^2 - c \]

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
Anatomy of a for Loop

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

![End of loop trace]

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++;
}

Why for loops? Can provide more compact and understandable code.

For Loops: Subdivisions of a Ruler

Create subdivision of a ruler.
• 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);
    }
}

![Loop Examples]

% java Ruler 1
1
% java Ruler 2
1 2 1
% java Ruler 3
1 2 3 1 2 1
% java Ruler 4
1 2 3 1 2 1 4 1 2 1 3 1 2 1
% java Ruler 5
1 2 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!
TEQ on For Loops
[easy if you read Exercise 1.3.13]

What does the following program print?

```java
public class Mystery {
    public static void main(String[] args) {
        int f = 0, g = 1;
        for (int i = 0; i <= 10; i++) {
            System.out.println(f);
            f = f + g;
            g = f - g;
        }
    }
}
```

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

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 Statements

Use nested if statements to handle multiple alternatives

```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;
```

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).

TEQ on If-Else

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;
```

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);
    }
}

Control Flow Summary

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

<table>
<thead>
<tr>
<th>Control Flow</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Straight-line</td>
<td>All statements are executed in the order given.</td>
<td></td>
</tr>
<tr>
<td>programs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conditionals</td>
<td>Certain statements are executed depending on the values of certain variables.</td>
<td>if-else</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loops</td>
<td>Certain statements are executed repeatedly until certain conditions are met.</td>
<td>while</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for</td>
</tr>
<tr>
<td></td>
<td></td>
<td>do-while</td>
</tr>
</tbody>
</table>

Digression: Simulation and Analysis

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

Admiral Grace Murray Hopper

http://www.history.navy.mil/photos/images/h96000/h96566kc.htm
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.

**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.

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++) {
            while (N % i == 0) {
                System.out.print(i + " ")
                N = N / i
            }
        }
    }
}
```

This program has bugs!

Debugging Example

**Factor.** Given an integer $N > 1$, compute its prime factorization.

- $3,757,208 = 2^3 \times 7 \times 13 \times 397$
- $98 = 2 \times 7^2$
- $17 = 17$
- $11,111,111,111,111 = 2,071,723 \times 5,363,222,357$

**Application.** Break RSA cryptosystem (factor 200-digit numbers).

```
public class Factors {
    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
            }
        }
    }
}
```

This program has bugs!
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 + " ");
        }
        System.out.println(N = N / i);
    }
}
```

% javac Factors.java
% java Factors

```
Exception in thread "main"
java.lang.ArithmeticException: / by zero
  at Factors.main(Factors.java:8)
```
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++) {
            // Check whether i is a factor.
            while (N % i == 0) {
                // If so, print and divide.
                System.out.print(i + " ");
                N = N / i;
            }
        }
    }
}
```

Semantic (run-time) error: indents do not imply braces

need to start at 2 since 0 and 1 cannot be factors

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

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

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.printf(i + " ");
                N = N / i;
            }
            System.out.println("TRACE " + i + " " + N);
        }
    }
}

Success? Program seems to work.
• Remove trace to try larger inputs.
• [stay tuned].

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) {
                // If so, print and divide.
                System.out.print(i + " ");
                N = N / i;
            }
        }
        if (N > 1) System.out.println(N); else System.out.println();
    }
}

Debugging Your Program

Debugging Your Program. [summary]

1. Create the program.
2. Compile it.
   Compiler says: That’s not a legal program.
   Back to step 1 to fix your errors of syntax.
3. Execute it.
   Result is bizarrely (or subtly) wrong.
   Back to step 1 to fix your errors of semantics.
4. Enjoy the satisfaction of a working program!
   [but stay tuned for more debugging]