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.

Conditionals
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;
}
```

---

If Statement Examples

- **absolute value**
  ```java
  if (x < 0) x = -x;
  ```
- **maximum**
  ```java
  if (x > y) max = x;
  else max = y;
  ```
- **error check for division operation**
  ```java
  if (den == 0) System.out.println("Division by zero");
  else System.out.println("Quotient = "+ num/den);
  ```
- **error check for quadratic formula**
  ```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);
  }
  ```

---

Loops

- **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");
    }
}
```
While Loop

The while loop. A common repetition structure.

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

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

boolean expression
true
false

Ex. Print powers of 2 that are ≤ 2^n.

- Increment i from 0 to n.
- Double v each time.

While Loop Example: Powers of Two

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

Powers of Two (full program)

```
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?

```
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 )</th>
<th>( 2/t )</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>1</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>1.417647</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>1.4142136</td>
</tr>
</tbody>
</table>

computing the square root of 2 to seven places

**The For Loop**

```
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);
    }
}
```

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

15 decimal digits of accuracy in 5 iterations
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;
}
```

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);
    }
}
```
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
Nested If Statements

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

Ex.

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

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

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;
```

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;
```

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

Nested If-Else Statements

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 (int i = 0; i < N; i++)
    for (int j = 0; j < M; j++)
        Do something at entry (i,j);
```
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.

```java
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);
    }
}
```

Nesting Example: Gambler’s Ruin Simulation

```java
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

**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.

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

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

\[
17 = 17
\]

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

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

% javac Factors.java
Factors.java:6: ';' expected
`for (i = 0; i < N; i++)` ^
1 error — the FIRST error

**Debugging: Semantic Errors**

**Semantic error.** Legal but wrong 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;
            }
        }
    }
}
```

% javac Factors.java
Factors.java:6: ';' expected
`for (i = 0; i < N; i++)` ^
1 error — the FIRST error

% java Factors
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 0
at Factors.main(Factors.java:5)

(oops, need argument)

Need terminating semicolons

Syntax (compile-time) errors

Need to declare variable i
Semantic errors. Legal but wrong Java program.
• Run program to identify problem.
• Two kinds: runtime (program crashes) and logic (program gets wrong answer).

```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.print(i + " ");
                N = N / i;
            }
        }
    }
}
```

Semantic error. Legal but wrong Java program.
• Run program to identify problem.
• Two kinds: runtime (program crashes) and logic (program gets wrong answer).

```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.print(i + " ");
                N = N / i;
            }
        }
    }
}
```

Semantic (logic) error: indents do not imply braces

```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.print(i + " ");
                N = N / i;
            }
        }
    }
}
```

Semantic (logic) error: indents do not imply braces

```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.print(i + " ");
                N = N / i;
            }
        }
    }
}
```
Debugging: The Beat Goes On

Success? Program factors 98 = 2 7 7.
• Time to try it for other 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.
            while (N % i == 0) {
                // If so, print and divide.
                System.out.println(i + " ");
                N = N / i;
            }
        }
        // System.out.println("TRACE " + i + " "+ N);
        if (N > 1) System.out.println(N);
        else System.out.println();
    }
}
```

Success? Program factors 98 = 2 7 7.
• Time to try it for other inputs.
• Add print statements to produce a trace.

```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.println(i + " ");
                N = N / i;
            }
        }
        System.out.println("TRACE " + i + " "+ N);
        if (N > 1) System.out.println(N);
        else System.out.println();
    }
}
```

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.
            while (N % i == 0) {
                // If so, print and divide.
                System.out.println(i + " ");
                N = N / i;
            }
        }
        // System.out.println("TRACE " + i + " "+ N);
        if (N > 1) System.out.println(N);
        else System.out.println();
    }
}
```

Performance error. Apparently 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) {
                // If so, print and divide.
                System.out.println(i + " ");
                N = N / i;
            }
        }
        // System.out.println("TRACE " + i + " "+ N);
        if (N > 1) System.out.println(N);
        else System.out.println();
    }
}
```
Debugging: Performance Errors

**Performance error.** Apparently 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 * 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: Back to Semantic Errors!

**Fresh semantic error.** Fast program (now), but new logic 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);
        else System.out.println();
    }
}
```

Program Development: Analysis

**Q.** How large an integer can I factor?

```plaintext
- 2 3 7 11 31 24 64 127 107 257 233 277 359 2047 4096 65537
```

- largest (a Mersenne prime: $2^{64} - 1$)
- 2nd-largest int
- largest long (but not a Mersenne prime)
- largest prime long

**Oh no! Another semantic error?**
Debugging: Back to Semantic Errors Again!

Another semantic error. Very big prime (another corner case) has logic error.

Q: How big can candidate factor i be?
A: Too big to be an int!

```
public class Factors {
    public static void main(String[] args) {
        long N = Long.parseLong(args[0]);
        for (long 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);
        else System.out.println();
    }
}
```

97

Program Development: Back to Analysis

Q. Once again, how large an integer can I factor?

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

```
% java Factors 9201111169755555703
9201111169755555703
```

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

```
in largest factor    digits   (1 < N)    (1*i <= N)
3                    instant  instant
6                    0.15 seconds  instant
9                    77 seconds  instant
12                   21 hours†  0.16 seconds
15                   2.4 years†  2.7 seconds
18                   2.4 millennia†  92 seconds

† estimated, using analytic number theory
```

Debugging Your Program

Debugging Your Program. [summary]

1. Edit the program (type in code).
2. Compile it.
   Compiler says: That’s not a legal program!
   Back to step 1 to fix your syntax errors.
3. Run it.
   Result is bizarrely (or subtly) wrong!
   Back to step 1 to fix your semantic (runtime and/or logic) errors.
4. Test it.
   Too slow?
   Back to step 1 to try a different algorithm.

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.

99
The First Bug?

Lieutenant Grace Murray Hopper


Lieutenant Grace Murray Hopper