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
equivalent
to a calculator

to infinity and beyond!

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

The *if* statement. A common branching structure.
- Evaluate a boolean expression.
- If true, execute some statements.
- If false, execute other statements.

```java
public class Flip {
   public static void main(String[] args) {
      if (Math.random() < 0.5) System.out.println("Heads");
      else System.out.println("Tails");
   }
}
```

Ex. Take different action depending on value of variable.
### If Statement Examples

<table>
<thead>
<tr>
<th>absolute value</th>
<th>if (x &lt; 0) x = -x;</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x and y into sorted</td>
<td>if (x &gt; y) {</td>
</tr>
<tr>
<td>order</td>
<td>int t = x;</td>
</tr>
<tr>
<td></td>
<td>y = x;</td>
</tr>
<tr>
<td></td>
<td>x = t;</td>
</tr>
<tr>
<td>maximum of x and y</td>
<td>if (x &gt; y) max = x;</td>
</tr>
<tr>
<td></td>
<td>else max = y;</td>
</tr>
<tr>
<td>error check for division</td>
<td>if (den == 0) System.out.println(&quot;Division by zero&quot;);</td>
</tr>
<tr>
<td>operation</td>
<td>else System.out.println(&quot;Quotient = &quot; + num/den);</td>
</tr>
<tr>
<td>error check for quadratic</td>
<td>double discriminant = b<em>b - 4.0</em>c;</td>
</tr>
<tr>
<td>formula</td>
<td>if (discriminant &lt; 0.0) {</td>
</tr>
<tr>
<td></td>
<td>System.out.println(&quot;No real roots&quot;);</td>
</tr>
<tr>
<td></td>
<td>} else {</td>
</tr>
<tr>
<td></td>
<td>System.out.println((-b + Math.sqrt(discriminant))/2.0);</td>
</tr>
<tr>
<td></td>
<td>System.out.println((-b - Math.sqrt(discriminant))/2.0);</td>
</tr>
</tbody>
</table>

---

### The While Loop

- **The while loop.** A common repetition structure.
  - Evaluate a **boolean** expression.
  - If true, execute some statements.
  - Repeat.

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

- **While Loop: Powers of Two**

**Ex.** Print powers of 2 that are \( \leq 2^N \).
- Increment \( i \) from 0 to \( N \).
- Double \( v \) each time.

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

<table>
<thead>
<tr>
<th>i</th>
<th>v</th>
<th>i &lt;= N</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>true</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>true</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>true</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>true</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>true</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>true</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>true</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>false</td>
</tr>
</tbody>
</table>

\( N = 6 \)
Powers of Two

```java
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(i + " " + v);
            i = i + 1;
            v = 2 * v;
        }
    }
}
```

% java PowersOfTwo 4
0 1
1 2
2 4
3 8
% java PowersOfTwo 6
0 1
1 2
2 4
3 8
4 16
5 32
6 64

While Loop Challenge

Q. Anything wrong with the following code for printing powers of 2?

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

While Loops: Square Root

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

A. 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 \( c / t_i \).

```
t_0 = 2.0

\[ t_1 = \frac{1}{2}(t_0 + \frac{c}{t_0}) = 1.5 \]
\[ t_2 = \frac{1}{2}(t_1 + \frac{c}{t_1}) = 1.416666666666665 \]
\[ t_3 = \frac{1}{2}(t_2 + \frac{c}{t_2}) = 1.4142158662745097 \]
\[ t_4 = \frac{1}{2}(t_3 + \frac{c}{t_3}) = 1.4142135623746899 \]
\[ t_5 = \frac{1}{2}(t_4 + \frac{c}{t_4}) = 1.414213562373095 \]
```

A Wonderful Square Root

"A wonderful square root. Let's hope it can be used for the good of mankind."

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

```
% java Sqrt 60481729
7777.0
```

While loops: Square Root

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

A. 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 \( c / t_i \).
While Loops: Square Root

Q. How might we implement Math.sqrt()?
A. 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 \( c / t_i \).

```java
class Sqrt {
    public static void main(String[] args) {
        double epsilon = 1e-15;
        double c = Double.parseDouble(args[0]);
        double t = c;
        while (Math.abs(t - c / t) > t * epsilon) {
            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.
- Goal: find root of any 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.

Caveat. \( f(x) \) must be smooth; \( t_0 \) must be good estimate.

For Loops

The for loop. Another common repetition structure.
- Execute initialization statement.
- Evaluate a boolean expression.
- If true, execute some statements.
- And then the increment statement.
- Repeat.

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

loop continuation condition

Copyright 2004, FoxTrot by Bill Amend
www.ucomics.com/foxtrot/2003/10/03
Anatomy of a For Loop

Q. What does it print?
A. For Loops: Subdivisions of a Ruler

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

```java
public class RulerN {
    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!

<table>
<thead>
<tr>
<th></th>
<th>ruler</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot; 1 &quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot; 1 2 1 &quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot; 1 2 1 3 1 2 1 &quot;</td>
</tr>
</tbody>
</table>

For Loops: Subdivisions of a Ruler

Loop Examples

- **print largest power of two less than or equal to N**
  ```java
  int v = 1;
  while (v < N/2) {
      v = 2*v;
  }
  System.out.println(v);
  ```

- **compute a finite sum \((1 + 2 + \ldots + N)\)**
  ```java
  int sum = 0;
  for (int i = 1; i <= N; i++) {
      sum += i;
  }
  System.out.println(sum);
  ```

- **compute a finite product \((N! = 1 \times 2 \times \ldots \times N)\)**
  ```java
  int product = 1;
  for (int i = 1; i <= N; i++) {
      product *= i;
  }
  System.out.println(product);
  ```

- **print a table of function values**
  ```java
  for (int i = 0; i <= N; i++) {
      System.out.println(i + " + 2*Math.PI*i/N");
  }
  ```

- **print the ruler function (see Program 1.2.1)**
  ```java
  String ruler = " ";
  for (int i = 1; i <= N; i++) {
      ruler = ruler + i + ruler;
  }
  System.out.println(ruler);
  ```
Nesting Conditionals and Loops

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

More sophisticated programs.
- Nest conditionals within conditionals.
- Nest loops within loops.
- Nest conditionals within loops within loops.

```java
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);
```

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

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

$2^9 = 1024$ possible results, depending on input

More sophisticated programs.
- Nest conditionals within conditionals.
- Nest loops within loops.
- Nest conditionals within loops within loops.

**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 - 55%</td>
<td></td>
</tr>
</tbody>
</table>

5 mutually exclusive alternatives

**Nested If Statements**

```java
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 if (income < 311950) 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 if (income < 311950) rate = 0.35;
        }
    }
}
```

Be careful when nesting if-else statements. (See Q+A on p. 75.)
### Nested If Statement Challenge

**Q.** Anything wrong with the following for income tax calculation?

<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>

```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;
```

### Monte Carlo Simulation

**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) {
        int stake = Integer.parseInt(args[0]);
        int goal = Integer.parseInt(args[1]);
        int T = Integer.parseInt(args[2]);
        int wins = 0;
        // repeat experiment N times
        for (int t = 0; t < T; t++) {
            // do one gambler's ruin experiment
            int cash = stake;
            while (cash > 0 && cash < goal) {
                // flip coin and update
                if (Math.random() < 0.5) cash++;
                else cash--;
            }
            if (cash == goal) wins++;
        }
        System.out.println(wins + " wins of " + T);
    }
}
```
**Digression: Simulation and Analysis**

Probability of winning = \( \frac{\text{stake}}{\text{goal}} \).

**Fact.** [see ORF 309] Expected number of bets = \( \frac{\text{stake} \times \text{desired gain}}{\text{goal}} \).

**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.

---

**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 programs</td>
<td>all statements are executed in the order given</td>
<td></td>
</tr>
<tr>
<td>conditionals</td>
<td>certain statements are executed depending on the values of certain variables</td>
<td>if, if-else</td>
</tr>
<tr>
<td>loops</td>
<td>certain statements are executed repeatedly until certain conditions are met</td>
<td>while, for, do-while</td>
</tr>
</tbody>
</table>

---

**Program Development**

95% of Program Development

**Program development.** Creating a program and putting it to good use.

**Def.** A bug is a mistake in a computer program.

Programming is primarily a process of finding and fixing bugs.

**Good news.** Can use computer to test program.

**Bad news.** Cannot use computer to automatically find all bugs.
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.” — Maurice Wilkes

“ If I had eight hours to chop down a tree, I would spend six hours sharpening an axe.” — Abraham Lincoln

Debugging Example

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

Brute-force algorithm. For each putative factor i = 2, 3, 4, …, check if N is a multiple of i, and if so, divide it out.

<table>
<thead>
<tr>
<th>i</th>
<th>N</th>
<th>output</th>
<th>i</th>
<th>N</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3757208</td>
<td>2 2 2</td>
<td>9</td>
<td>67093</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>469651</td>
<td></td>
<td>16</td>
<td>397</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>469651</td>
<td></td>
<td>17</td>
<td>397</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>469651</td>
<td></td>
<td>18</td>
<td>397</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>469651</td>
<td></td>
<td>19</td>
<td>397</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>469651</td>
<td>7 397</td>
<td>20</td>
<td>397</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>67093</td>
<td>15 397</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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


Debugging: 95% 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 many bugs!
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;
            }
        }
    }
}
```

```java
% javac Factors.java
Factors.java:6: ';' expected
    for (i = 2; i < N; i++) {
    ^
1 error
```

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

```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;
            }
        }
    }
}
```

```java
% javac Factors.java
% java Factors
Exception in thread "main"
java.lang.ExceptionInInitializerError
    at Factors.main(Factors.java:8)
% java lang ArrayIndexOutOfBoundsException: 0
```

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;
            }
        }
    }
}
```

```java
% javac Factors.java
% java Factors
Exception in thread "main"
java.lang.ArithmeticException: / by zero
    at Factors.main(Factors.java:8)
```
**Debugging: Semantic Errors**

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

```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;
            }
        }
        System.out.println("TRACE: " + i + " "+ N);
    }
}
```

**Debugging: The Beat Goes On**

**Success.** Program factors $98 = 2 \times 7^2$.
- But that doesn’t mean it works for all 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.print(i + " ");
                N = N / i;
            }
        }
        System.out.println("TRACE: " + i + " "+ N);
    }
}
```

**Debugging: Success?**

**Success.** Program seems to work.

```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;
            }
        }
        System.out.println("TRACE: " + i + " "+ N);
    }
}
```
Performance error. Correct program, but too slow.

```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;
            }
        }
        if (N > 1) System.out.println(N);
        else System.out.println();
    }
}
```

Solution. 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; i++) {
            while (N % i == 0) {
                System.out.print(i + " ");
                N = N / i;
            }
        }
        if (N > 1) System.out.println(N);
        else System.out.println();
    }
}
```

Q. How large an integer can I factor?

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

% java Factors 920111169755555703
9201111697555555703
```

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