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
}
```

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);
    }
}
```

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.

Applications and extensions.
- Find roots of a differentiable function.
- Optimize a twice differentiable function.

For Loops: Subdivisions of a Ruler

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

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

<table>
<thead>
<tr>
<th>i</th>
<th>ruler</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1 2 1</td>
</tr>
<tr>
<td>3</td>
<td>1 2 1 3 1 2 1</td>
</tr>
<tr>
<td>4</td>
<td>1 2 1 3 1 2 1</td>
</tr>
</tbody>
</table>

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

Nesting Conditionals and Loops

**Conditionals** enable you to do one of \(2^N\) sequences of operations with \(N\) lines.

\[
\begin{align*}
\text{if } (a0 > 0) & \Rightarrow \text{System.out.print(0);} \\
\text{if } (a1 > 0) & \Rightarrow \text{System.out.print(1);} \\
\text{if } (a2 > 0) & \Rightarrow \text{System.out.print(2);} \\
\text{if } (a3 > 0) & \Rightarrow \text{System.out.print(3);} \\
\text{if } (a4 > 0) & \Rightarrow \text{System.out.print(4);} \\
\text{if } (a5 > 0) & \Rightarrow \text{System.out.print(5);} \\
\text{if } (a6 > 0) & \Rightarrow \text{System.out.print(6);} \\
\text{if } (a7 > 0) & \Rightarrow \text{System.out.print(7);} \\
\text{if } (a8 > 0) & \Rightarrow \text{System.out.print(8);} \\
\text{if } (a9 > 0) & \Rightarrow \text{System.out.print(9);} \\
\end{align*}
\]

2\(^N\) = 1024 possible results, depending on input

**Loops** enable you to do an operation \(N\) times using only 2 lines of code.

\[
\begin{align*}
\text{double sum = 0.0;} \\
\text{for (int i = 1; i <= 1024; i++)} \\
\quad \text{sum = sum + 1.0 / i;} \\
\end{align*}
\]

computes \(1/1 + 1/2 + \ldots + 1/1024\)

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.

<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>

\[
\begin{align*}
\text{double rate;} \\
\text{if } (\text{income} < 47450) \text{ rate} = 0.22; \\
\text{else if } (\text{income} < 114650) \text{ rate} = 0.25; \\
\text{else if } (\text{income} < 174700) \text{ rate} = 0.28; \\
\text{else if } (\text{income} < 311950) \text{ rate} = 0.33; \\
\text{else } \text{ rate} = 0.35; \\
\end{align*}
\]

graduated income tax calculation

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.
### 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`.

```java
public class Factors1 {
    public static void main(String[] args) {
        long N = Long.parseLong(args[0]);
        for (i = 2; i < N; i++) {
            while (N % i == 0) {
                System.out.print(i + " ");
                N = N / i;
            }
        }
    }
}
```

Does not compile

### Debugging a Program: Semantic Errors

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

```java
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)

### Simulation and Analysis

<table>
<thead>
<tr>
<th>stake</th>
<th>goal</th>
<th>N</th>
<th>wins</th>
<th>after a few minutes of computing...</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1000</td>
<td></td>
<td>513</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>2500</td>
<td>100</td>
<td>492</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>2500</td>
<td>1000</td>
<td>24</td>
<td></td>
</tr>
</tbody>
</table>

**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.

---

```java
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++) {
            int t = stake;
            while (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);
        }
    }
}
```

Gambler’s Ruin

---

**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 a Program: Performance Errors

**Performance bug.** Correct program but too slow.
- Use profiling to discover bottleneck.
- Devise better algorithm.

```java
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)

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

```java
% java Factors 168
2 2 2 3 7
% java Factors 3757208
2 2 2 7 13 13 397
% java Factors 9201111169755555703
9201111169755555703
```

Fact. If N has a factor, it has one less than or equal to its square root.

Impact. Many fewer iterations of `for` loop.

---

**Fact.** If N has a factor, it has one less than or equal to its square root.

Impact. Many fewer iterations of `for` loop.

```java
public class Factors1 {
    public static void main(String[] args) {
        long N = Long.parseLong(args[0]);
        for (long i = 2; i <= Math.sqrt(N); 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: Analysis

### Q. How big an integer can I factor?

```java
% 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...

<table>
<thead>
<tr>
<th>Digits</th>
<th>i &lt;= N</th>
<th>i &lt;= N / i</th>
<th>i * i &lt;= N</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>instant</td>
<td>instant</td>
<td>instant</td>
</tr>
<tr>
<td>6</td>
<td>0.15 seconds</td>
<td>instant</td>
<td>instant</td>
</tr>
<tr>
<td>9</td>
<td>77 seconds</td>
<td>instant</td>
<td>instant</td>
</tr>
<tr>
<td>12</td>
<td>21 hours</td>
<td>0.21 seconds</td>
<td>0.16 seconds</td>
</tr>
<tr>
<td>15</td>
<td>2.4 years</td>
<td>4.5 seconds</td>
<td>2.7 seconds</td>
</tr>
<tr>
<td>18</td>
<td>2.4 millennia</td>
<td>157 seconds</td>
<td>92 seconds</td>
</tr>
</tbody>
</table>

† 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.

<table>
<thead>
<tr>
<th>Flow-Of-Control</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, 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>

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

Debugging a Program

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 8 hours to chop down a tree, I would spend 6 hours sharpening an axe." - Anonymous