1.3 Conditionals and Loops

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.

straight-line control flow

control flow with conditionals and loops
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;
}
```

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

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

Diagram for the if statement: [Diagram Image]

Diagram for the if statement: [Diagram Image]
If Statement

Ex. Take different action depending on value of variable.

```java
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

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

maximum

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

error check for division operation

double discriminant = b*b - 4.0*c;
if (discriminant < 0.0)
{
    System.out.println("No real roots");
}
else
{
    int t = x;
    x = y;
    y = t;
}

2-sort

x > y before

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>t</td>
</tr>
<tr>
<td>1234</td>
<td>99</td>
<td>undefined</td>
</tr>
<tr>
<td>99</td>
<td>99</td>
<td>1234</td>
</tr>
<tr>
<td>99</td>
<td>1234</td>
<td>1234</td>
</tr>
</tbody>
</table>

x < y after

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>t</td>
</tr>
<tr>
<td>1234</td>
<td>99</td>
<td>1234</td>
</tr>
<tr>
<td>99</td>
<td>99</td>
<td>1234</td>
</tr>
<tr>
<td>99</td>
<td>1234</td>
<td>1234</td>
</tr>
</tbody>
</table>

equal 2-sort

equal

x y t
1234 99 undefined
1234 99 1234
99 99 1234
99 1234 1234

error check for quadratic formula
Loops
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 \( \leq 2^n \).
- Increment \( i \) from 0 to \( n \).
- Double \( v \) each time.

```java
int i = 0;
int v = 1;
while (i <= n) {
    System.out.println(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>

\( n = 6 \)
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 )</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>1.4117647</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></td>
</tr>
</tbody>
</table>

computing the square root of 2 to seven places

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

Copyright 2004, Sidney Harris
http://www.sciencecartoonsplus.com
While Loop Example: Square Root

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 \( c / t_i \).

```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.

$f(x) = x^2 - c$ to compute $\sqrt{c}$
The For Loop

```c
#include <stdio.h>
int main(void)
{
    int count;
    for (count = 1; count <= 500; count++)
        printf("I will not throw paper airplanes in class.\n");
    return 0;
}
```
The **for** loop. Another common repetition structure.

- Execute initialization statement.
- Check boolean expression.
- Execute sequence of statements.
- Execute increment statement.
- Repeat.

```cpp
for (init; boolean expression; increment)
{
    statement 1;
    statement 2;
}
```
Anatomy of a for Loop

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

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

**initialize another variable in a separate statement**
**declare and initialize a loop control variable**
**loop continuation condition**
**increment**

**body**

prints table of powers of two
Anatomy of a for Loop

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

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

Every for loop has an equivalent while loop.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0 1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1 2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2 4</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>3 8</td>
</tr>
</tbody>
</table>

Why for loops? *Can provide more compact and understandable code.*
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 \( i \).

```java
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);
    }
}
```

<table>
<thead>
<tr>
<th>( i )</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>

end-of-loop trace
For Loops: Subdivisions of a Ruler

\[
2^{100} - 1 = 1,267,650,600,228,229,401,496,703,205,375 \text{ integers in output}
\]

**Observation.** Loops can produce a huge amount of output!
# Loop Examples

```java
int sum = 0;
for (int i = 1; i <= N; i++)
    sum += i;
System.out.println(sum);
```

**compute sum \((1 + 2 + 3 + \ldots + N)\)**

<table>
<thead>
<tr>
<th>sum</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
</tbody>
</table>

trace at end of loop for \(N = 4\)

```java
int product = 1;
for (int i = 1; i <= N; i++)
    product *= i;
System.out.println(product);
```

**compute \(N! (1 * 2 * 3 * \ldots * N)\)**

<table>
<thead>
<tr>
<th>product</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>24</td>
<td>4</td>
</tr>
</tbody>
</table>

```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 \(N\)**

<table>
<thead>
<tr>
<th>v</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
</tr>
</tbody>
</table>

**print a table of function values**

```java
for (int i = 0; i <= N; i++)
    System.out.println(i + " " + 2*Math.PI*i/N);
```

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.57079632...</td>
</tr>
<tr>
<td>2</td>
<td>3.14159265...</td>
</tr>
<tr>
<td>3</td>
<td>4.71238898...</td>
</tr>
<tr>
<td>4</td>
<td>6.28318530...</td>
</tr>
</tbody>
</table>

**N = 4**

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

trace at end of loop for \(N = 23\)
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++;
        else t--;
        if (t == goal) wins++;
    }
}
```
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

```cpp
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;
        }
    }
}
```
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

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?

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

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 $\text{stake}$ and places $1$ fair bets until going broke or reaching $\text{goal}$.

• 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);
    }
}

Nesting Example: Gambler's Ruin Simulation
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);
    }
}
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 = \( \frac{\text{stake}}{\text{goal}} \).

Fact. Expected number of bets = \( \text{stake} \times \text{desired gain} \).

Ex. 20\% chance of turning $500 into $2500, but expect to make one million $1 bets.

\[
\frac{500}{2500} = 20\% \\
500 \times (2500 - 500) = 1,000,000
\]

Remark. Both facts can be proved mathematically. For more complex scenarios, computer simulation is often the best plan of attack.
Debugging
Factor. Given an integer $N > 1$, compute its prime factorization.

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

$98 = 2 \times 7^2$

$17 = 17$

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


Note: 1 is not prime. (else it would have to be in every factorization)
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: 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++)
        {
            while (N % i == 0)
                System.out.print(i + " ")
            N = N / i
        }
    }
}
```
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++)
            { 
                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: 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;
        }
    }
}
```

- **Syntax (compile-time) errors**
  - need terminating semicolons
  - need to declare variable `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 = 0; i < N; i++) {
            while (N % i == 0) {
                System.out.print(i + " ");
                N = N / i;
            }
        }
    }
}
```

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

(oops, need argument)

you will see this message!
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 = 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)
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;
            }
        }
    }
}
```

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

% javac Factors.java
% java Factors 98
```
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```
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
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.print(i + " ");
                N = N / i;
            }
        }
    }
}
```

% java Factors 98
2 7 7 %  need newline
% java Factors 5
%
% java Factors 6
2 %  ??? where's the 3?
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++) {
            while (N % i == 0) {
                System.out.println(i + " ");
                N = N / i;
            }
            System.out.println("TRACE " + i + " " + N);
        }
    }
}
```

% javac Factors.java
% java Factors 5
TRACE 2 5
TRACE 3 5
TRACE 4 5
% java Factors 6
2
TRACE 2 3

AHA! Print out N after for loop (if it is not 1)
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.print(i + " ");
                N = N / i;
            }
        }
        // System.out.println("TRACE " + i + " " + N);
        if (N > 1) System.out.println(N);
        else System.out.println();
    }
}
```

Time to document code (if not earlier).

"Comment out" trace code (may need it later)

Corner case: print largest factor (and new line)
Performance error. Apparently correct program, but too slow.

• Are all iterations of inner loop necessary?
• Improve or change underlying algorithm.

```
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();
    }
}
```

very long wait!
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();
    }
}
```

Fixes performance error: terminate when $i^2 > N$ since no larger factors left
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();
    }
}
```

% java Factors 24
2 2 2 3
% java Factors 25
25
% java Factors 49
49
%

Can't handle perfect squares!
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();
    }
}
```

% java Factors 24
2 2 2 3
% java Factors 25
5 5
% java Factors 49
7 7
%

Execute loop body if \( i \times i \leq N \)
Q. How large an integer can I factor?

```java
% java Factors 2147483647
2147483647

% java Factors 2147483646
2 3 3 7 11 31 151 331

% java Factors 9223372036854775807
7 7 73 127 337 92737 649657

% java Factors 9201111169755555703
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

Oh no! Another semantic error?
```
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!

```java
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();
    }
}
```

```
long i now big enough to be square root of biggest N
```

```
% java Factors 920111169755555703
9201111697555555703
```
Q. Once again, how large an integer can I factor?

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

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.

stay tuned
The First Bug?

Photo # NH 96566-KN  First Computer "Bug", 1945

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

Lieutenant Grace Murray Hopper