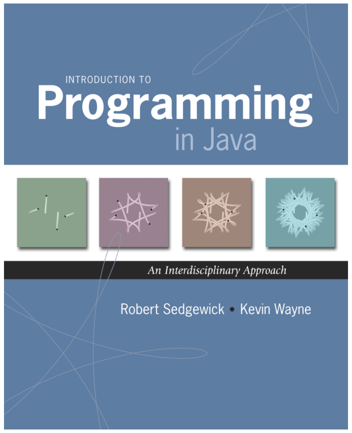
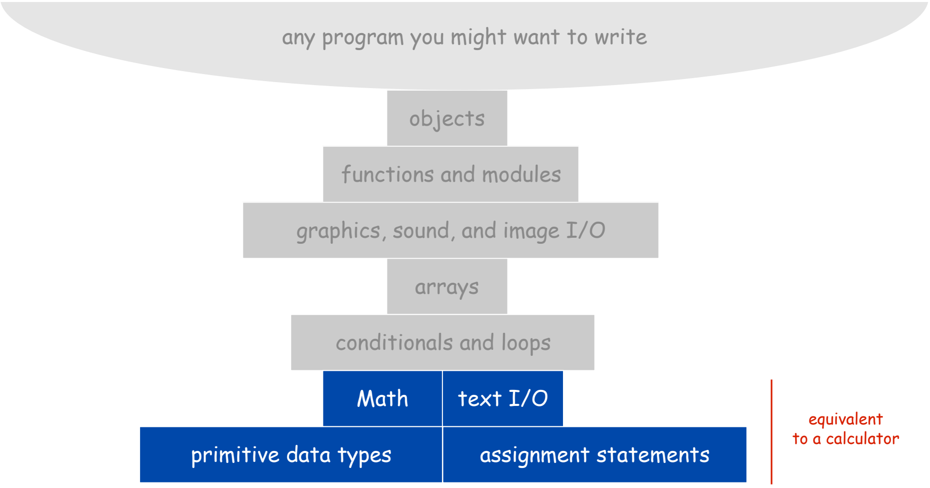
1.3 Conditionals and Loops



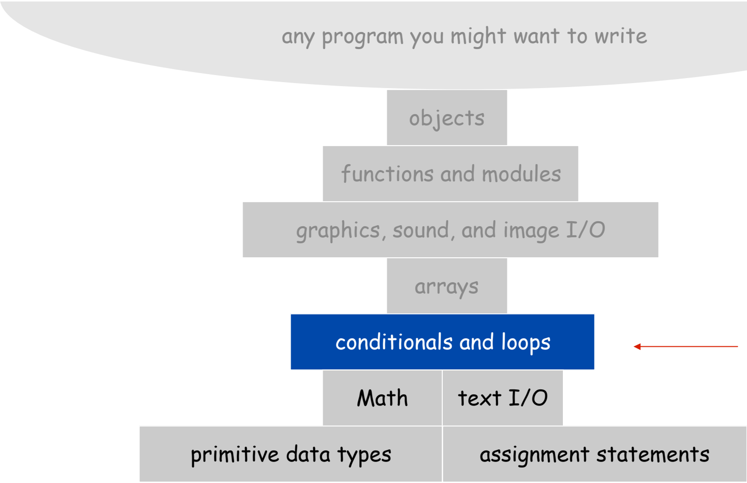
Introduction to Programming in Java: An Interdisciplinary Approach · Robert Sedgewick and Kevin Wayne · Copyright © 2008 · February 10, 2009 10:07 PM

A Foundation for Programming



equivalent to a calculator

A Foundation for Programming

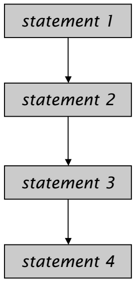


to infinity and beyond!

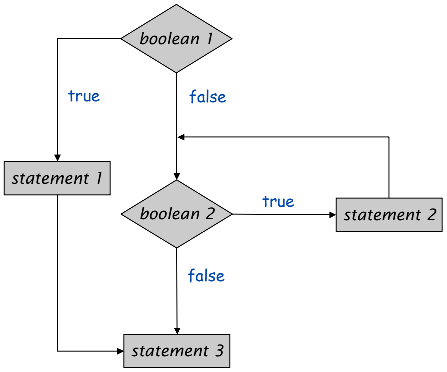
Control Flow

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

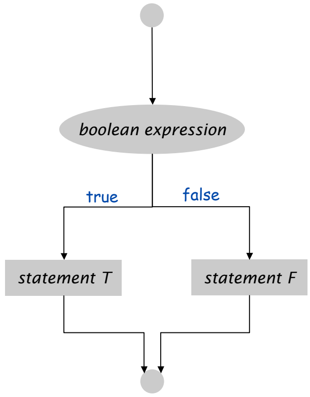
If Statement

The `if` statement. A common branching structure.

- Evaluate a `boolean` expression.
- If `true`, execute some statements.
- If `false`, execute other statements.

```
if (boolean expression) {
    statement T;
}
else {
    statement F;
}
```

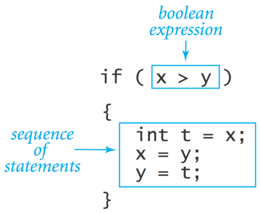
can be any sequence of statements



If Statement

The `if` statement. A common branching structure.

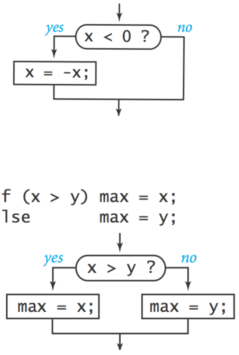
- Evaluate a `boolean` expression.
- If `true`, execute some statements.
- If `false`, execute other statements.



```
if ( x > y )
{
   int t = x;
   x = y;
   y = t;
}
```

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

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



If Statement

Ex. Take different action depending on value of variable.

```
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

absolute value	`if (x < 0) x = -x;`

put x and y into sorted order	`if (x > y)`
	`{`
	`   int t = x;`
	`   y = x;`
	`   x = t;`
	`}`

maximum of x and y	`if (x > y) max = x;`
	`else       max = y;`

error check for division operation	`if (den == 0) System.out.println("Division by zero");`
	`else          System.out.println("Quotient = " + num/den);`

error check for quadratic formula	`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);`
	`}`

The While Loop

While Loop

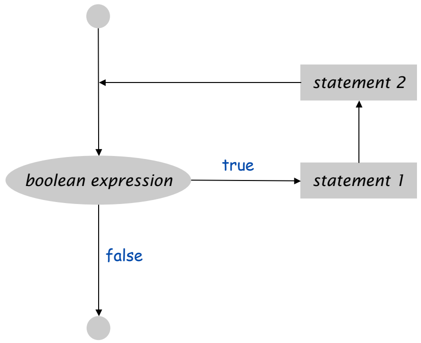
The `while` loop. A common repetition structure.

- Evaluate a `boolean` expression.
- If `true`, execute some statements.
- Repeat.

loop continuation condition

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

loop body



While Loop: Powers of Two

Ex. Print powers of 2 that are ≤ 2^N.

- Increment `i` from `0` to `N`.
- Double `v` each time.

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

i	v	i <= N
0	1	true
1	2	true
2	4	true
3	8	true
4	16	true
5	32	true
6	64	true
7	128	false

```
0 1
1 2
2 4
3 8
4 16
5 32
6 64
```

N = 6

Click for demo

Powers of Two

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

print i and ith power of two

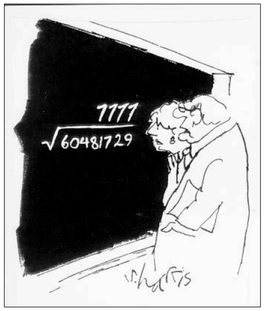
```
% java PowersOfTwo 4
0 1
1 2
2 4
3 8

% 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?

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

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:

- 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.

$$\begin{aligned}
t_0 & & &= 2.0 \\
t_1 &= \tfrac{1}{2}(t_0 + \tfrac{2}{t_0}) &&= 1.5 \\
t_2 &= \tfrac{1}{2}(t_1 + \tfrac{2}{t_1}) &&= 1.4166666666666665 \\
t_3 &= \tfrac{1}{2}(t_2 + \tfrac{2}{t_2}) &&= 1.4142156862745097 \\
t_4 &= \tfrac{1}{2}(t_3 + \tfrac{2}{t_3}) &&= 1.4142135623746899 \\
t_5 &= \tfrac{1}{2}(t_4 + \tfrac{2}{t_4}) &&= 1.414213562373095
\end{aligned}$$

computing the square root of 2

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.

```
public 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);
   }
}
```

relative error tolerance

```
% java Sqrt 2.0
1.414213562373095
```

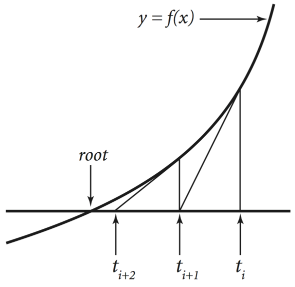
15 decimal digits of accuracy in 5 iterations

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.

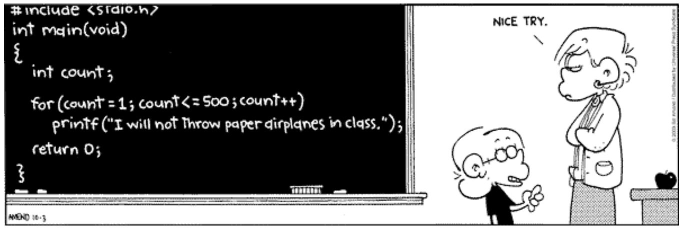
$f(x) = x^2 - c$ to compute $\sqrt{c}$



$$t_{i+1} = t_i - \frac{f(t_i)}{f'(t_i)}$$

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

The For Loop



```
#include <stdio.h>
int main(void)
{
   int count;

   for (count = 1; count <= 500; count++)
      printf("I will not Throw paper airplanes in class.");

   return 0;
}
```

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

For Loops

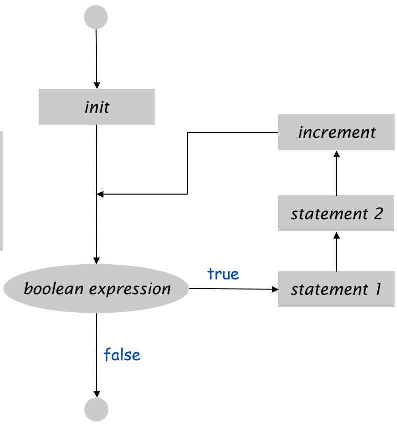
The `for` loop. Another common repetition structure.

- Execute initialization statement.
- Evaluate a `boolean` expression.
- If `true`, execute some statements.
- And then the increment statement.
- Repeat.

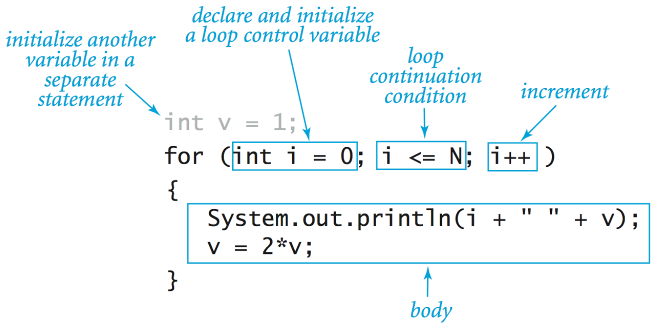
loop continuation condition

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

body



Anatomy of a For Loop



```
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

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`.

```
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);
   }
}
```

i	ruler
	" "
1	" **1** "
2	" 1 **2** 1 "
3	" 1 2 1 **3** 1 2 1 "

For Loops: Subdivisions of a Ruler

```
% java RulerN 1
 1

% java RulerN 2
 1 2 1

% java RulerN 3
 1 2 1 3 1 2 1

% java RulerN 4
 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

% java RulerN 5
 1 2 1 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 RulerN 100
Exception in thread "main"
java.lang.OutOfMemoryError
```

Observation. Loops can produce a huge amount of output!

Loop Examples

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

compute a finite sum (1 + 2 + . . . + N)	`int sum = 0;`
	`for (int i = 1; i <= N; i++)`
	`   sum += i;`
	`System.out.println(sum);`

compute a finite product (N! = 1 × 2 × . . . × N)	`int product = 1;`
	`for (int i = 1; i <= N; i++)`
	`   product *= i;`
	`System.out.println(product);`

print a table of function values	`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)	`String ruler = " ";`
	`for (int i = 1; i <= N; i++)`
	`   ruler = ruler + i + ruler;`
	`System.out.println(ruler);`

Nesting

Nesting Conditionals and Loops

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

```
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);
```

2^{10} = 1024 possible results, depending on input

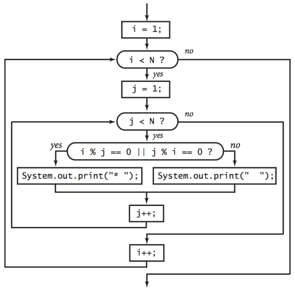
Loops enable you to do an operation n times using only 2 lines of code.

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

computes 1/1 + 1/2 + ... + 1/1024

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.

Income	Rate
0 - 47,450	22%
47,450 - 114,650	25%
114,650 - 174,700	28%
174,700 - 311,950	33%
311,950 -	35%

5 mutually exclusive alternatives

```
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                      rate = 0.35;
```

graduated income tax calculation

Nested If Statements

```
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

```
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?

Income	Rate
0 - 47,450	22%
47,450 - 114,650	25%
114,650 - 174,700	28%
174,700 - 311,950	33%
311,950 -	35%

```
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;
```

wrong graduated income tax calculation

Monte Carlo Simulation

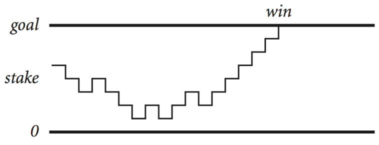
Gambler's Ruin

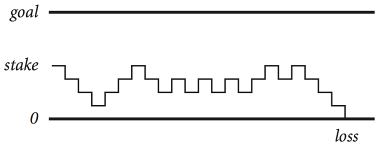
Gambler's ruin. Gambler starts with $stake and places $1 fair bets until going broke or reaching $goal.

- 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.





Gambler's Ruin

```
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

stake goal T

```
% java Gambler 5 25 1000
191 wins of 1000

% java Gambler 5 25 1000
203 wins of 1000

% java Gambler 500 2500 1000
197 wins of 1000
```

after a substantial wait….

Fact. [see ORF 309] Probability of winning = stake ÷ goal.
Fact. [see ORF 309] Expected number of bets = stake × desired gain.
Ex. 20% chance of turning $500 into $2500,
but expect to make one million $1 bets.

500/2500 = 20%
500 * (2500 - 500) = 1 million

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.

Control Flow	Description	Examples
straight-line programs	all statements are executed in the order given	
conditionals	certain statements are executed depending on the values of certain variables	`if` `if-else`

loops	certain statements are executed repeatedly until certain conditions are met	`while`
		`for`
		`do-while`

Program Development

Ada Lovelace

Admiral Grace Murray Hopper

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.

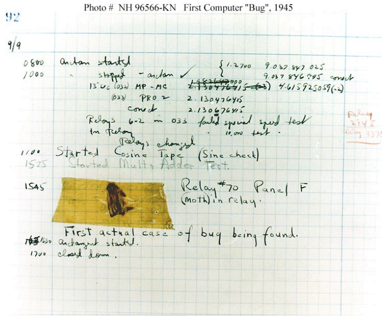


Photo # NH 96566-KN First Computer "Bug", 1945

92

9/9

0800 Antan started
1000 " stopped - antan ✓ {1.2700 9.037 847 025
9.037 846 995 conect
13" uc (032) MP - MC ~~1.982147000~~ ~~2.130476415~~ ~~(-3)~~ 4.615925059(-2)
(033) PRO 2 2.130476415
conect 2.130676415
Relays 6-2 in 033 failed special speed test
In Relay " 10,000 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1630 antangent started.
1700 closed down.

Good news. Can use computer to test program.
Bad news. Cannot use computer to automatically find all bugs.

95% 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. "* — 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.

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

$98 = 2 \times 7^2$

$17 = 17$

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

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

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.

i	N	*output*	i	N	*output*	i	N	*output*
2	3757208	2 2 2	9	67093		16	397	
3	469651		10	67093		17	397	
4	469651		11	67093		18	397	
5	469651		12	67093		19	397	
6	469651		13	67093	13 13	20	397	
7	469651	7	14	397				397
8	67093		15	397				

3757208/8

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.

check if i is a factor

as long as i is a factor, divide it out

```
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 many bugs!

Debugging: Syntax Errors

Syntax error. Illegal Java program.

- Compiler error messages help locate problem.
- Goal: no errors and a file named `Factors.class`.

```
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 = 2; 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`.

```
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;
      }
   }
}
```

need to declare variable i

need terminating semicolons

syntax (compile-time) errors

Debugging: Semantic Errors

Semantic error. Legal but wrong Java program.

- Run program to identify problem.
- Add print statements if needed to produce trace.

```
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, no argument

Debugging: Semantic Errors

Semantic error. Legal but wrong Java program.

- Run program to identify problem.
- Add print statements if needed to produce trace.

```
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;
      }
   }
}
```

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

```
% javac Factors.java
% java Factors 98
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.

```
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;
      }
   }
}
```

indents do not imply braces

```
% 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 2 2 2 …
```

infinite loop!

Debugging: The Beat Goes On

Success. Program factors 98 = 2 × 7^2.

- But that doesn't mean it works for all 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.print(i + " ");
            N = N / i;
         }
      }
   }
}
```

```
% java Factors 98
2 7 %

% java Factors 5

% java Factors 6
2 %
```

need newline

??? no output

??? missing the 3

Debugging: The Beat Goes On

Success. Program factors 98 = 2 × 7^2.

- But that doesn't mean it works for all 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.println(i + " ");
            N = N / i;
         }
         System.out.println("TRACE: " + i + " " + N);
      }
   }
}
```

```
% 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)

Debugging: Success?

Success. Program seems to work.

```
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();
   }
}
```

"corner case"

```
% java Factors 5
5

% java Factors 6
2 3

% java Factors 98
2 7 7

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

Debugging: Performance Error

Performance error. Correct program, but too slow.

```
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();
   }
}
```

```
% java Factors 11111111
11 73 11 137

% java Factors 11111111111
21649 51329

% java Factors 11111111111111
11 239 4649 909091

% java Factors 11111111111111111
2071723
```

very long wait
(with a surprise ending)

Debugging: Performance Error

Performance error. Correct program, but too slow.

Solution. Improve or change underlying algorithm.

fixes performance error:
if N has a factor, it has one
less than or equal to its square root

```
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();
   }
}
```

```
% java Factors 11111111
11 73 11 137

% java Factors 11111111111
21649 51329

% java Factors 11111111111111
11 239 4649 909091

% java Factors 11111111111111111
2071723 5363222357
```

Program Development: Analysis

Q. How large an integer can I factor?

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

% java Factors 9201111169755555703
9201111169755555703
```

after a few minutes of computing….

largest factor

digits	(i <= N)	(i*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

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

Debugging

Programming. A process of finding and fixing mistakes.

1. Create the program.

2. Compile it.
   Compiler says: That's not a legal program.
   Back to step 1 to fix syntax errors.

3. Execute it.
   Result is bizarrely (or subtly) wrong.
   Back to step 1 to fix semantic errors.

4. Enjoy the satisfaction of a working program!

5. Too slow? Back to step 1 to try a different algorithm.