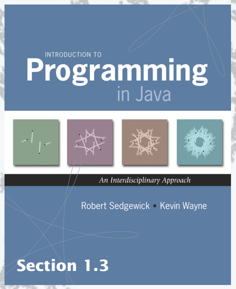
COMPUTER SCIENCE
SEDGEWICK / WAYNE



INTRODUCTION TO
Programming
in Java

An Interdisciplinary Approach

Robert Sedgewick · Kevin Wayne

Section 1.3

http://introcs.cs.princeton.edu

3. Conditionals and loops

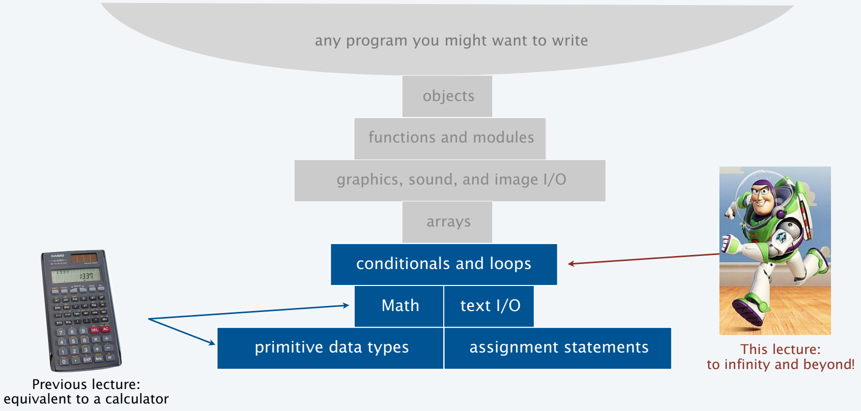
COMPUTER SCIENCE
SEDGEWICK / WAYNE

3. Conditionals & Loops

- Conditionals: the `if` statement
- Loops: the `while` statement
- An alternative: the `for` loop
- Nesting
- Debugging

CS.3.A.Loops.If

Context: basic building blocks for programming



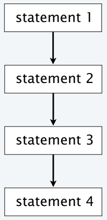
Previous lecture:
equivalent to a calculator

This lecture:
to infinity and beyond!

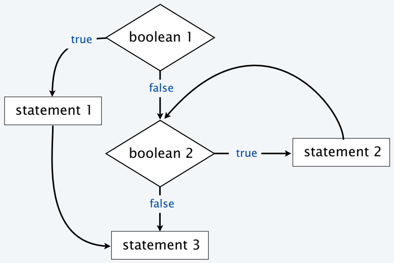
Conditionals and Loops

Control flow

- The sequence of statements that are actually executed in a program.
- Conditionals and loops enable us to choreograph control flow.



straight-line control flow
[previous lecture]



control flow with conditionals and a loop
[this lecture]

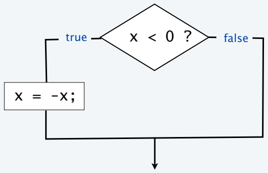
The if statement

Execute certain statements depending on the values of certain variables.

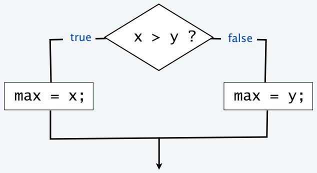
- Evaluate a boolean expression.
- If true, execute a statement.
- The else option: If false, execute a different statement.

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

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



Computes the absolute value of x

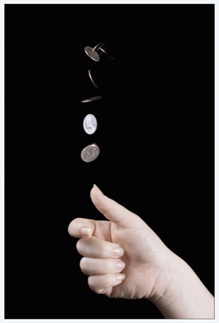


Computes the maximum of x and y

Example of if statement use: simulate a coin flip

```
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
```



Example of if statement use: 2-sort

Q. What does this program do?

```
public class TwoSort
{
   public static void main(String[] args)
   {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      if (b < a)
      {
         int t = a;
         a = b;
         b = t;
      }
      StdOut.println(a);
      StdOut.println(b);
   }
}
```

alternatives for if and else can be a *sequence* of statements, enclosed in braces

```
% java TwoSort 1234 99
99
1234

% java TwoSort 99 1234
99
1234
```

A. Reads two integers from the command line, then prints them out in numerical order.

Pop quiz on if statements

Q. Add code to this program that puts a, b, and c in numerical order.

```
public class ThreeSort
{
   public static void main(String[] args)
   {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      int c = Integer.parseInt(args[2]);




      StdOut.println(a);
      StdOut.println(b);
      StdOut.println(c);
   }
}
```

```
% java ThreeSort 1234 99 1
1
99
1234

% java ThreeSort 99 1 1234
1
99
1234
```

Example of `if` statement use: error checks

```
public class IntOps
{
   public static void main(String[] args)
   {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      int sum  = a + b;
      int prod = a * b;
      System.out.println(a + " + " + b + " = " + sum);
      System.out.println(a + " * " + b + " = " + prod);
      if (b == 0) System.out.println("Division by zero");
      else        System.out.println(a + " / " + b + " = " + a / b);
      if (b == 0) System.out.println("Division by zero");
      else        System.out.println(a + " % " + b + " = " + a % b);
   }
}
```

```
% java IntOps 5 2
5 + 2 = 7
5 * 2 = 10
5 / 2 = 2
5 % 2 = 1

% java IntOps 5 0
5 + 0 = 5
5 * 0 = 0
Division by zero
Division by zero
```

Good programming practice. Use conditionals to check for *and avoid* runtime errors.

COMPUTER SCIENCE
SEDGEWICK / WAYNE

3. Conditionals & Loops

- Conditionals: the `if` statement
- **Loops: the `while` statement**
- An alternative: the `for` loop
- Nesting
- Debugging

CS.3.B.Loops.While

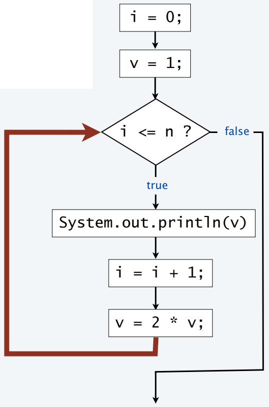
The `while` loop

Execute certain statements repeatedly until certain conditions are met.

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

Example:

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



Prints the powers of two from 2^0 to 2^n .

[stay tuned for a trace]

Example of `while` loop use: print powers of two

```
public class PowersOfTwo
{
   public static void main(String[] args)
   {
      int n = Integer.parseInt(args[0]);
      int i = 0;
      int v = 1;
      while (i <= n)
      {
         System.out.println(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

```
% java PowersOfTwo 6
1
2
4
8
16
32
64
```

Prints the powers of two from 2^0 to 2^n .

Pop quiz on `while` loops

Q. Anything wrong with the following code?

```
public class PQwhile
{
   public static void main(String[] args)
   {
      int n = Integer.parseInt(args[0]);
      int i = 0;
      int v = 1;
      while (i <= n)
         System.out.println(v);
         i = i + 1;
         v = 2 * v;
   }
}
```

Example of `while` loop use: implement `Math.sqrt()`

Goal. Implement square root function.

```
% java Sqrt 60481729
7777.0
% java Sqrt 2
1.4142136
```

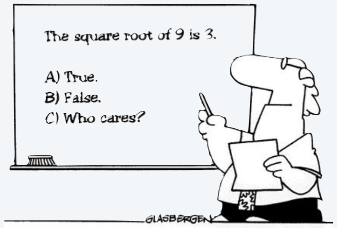
Newton-Raphson method to compute $\sqrt{c}$

- Initialize $t_0 = c$.
- Repeat until $t_i = c/t_i$ (up to desired precision): ← if $t = c/t$ then $t^2 = c$

  Set t_{i+1} to be the average of t_i and c / t_i.

i	t_i	$2/t_i$	*average*
0	2.0	1.0	1.5
1	1.5	1.3333333	1.4166667
2	1.4166667	1.4117647	1.4142157
3	1.4142157	1.4142114	1.4142136
4	1.4142136	1.4142136	

computing the square root of 2 to seven places



Many students actually look forward to Mr. Atwadder's math tests.

Example of `while` loop use: implement `Math.sqrt()`

Newton-Raphson method to compute $\sqrt{c}$

- 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.

Isaac Newton
1642-1727

Scientists studied *computation* well before the onset of the *computer.*

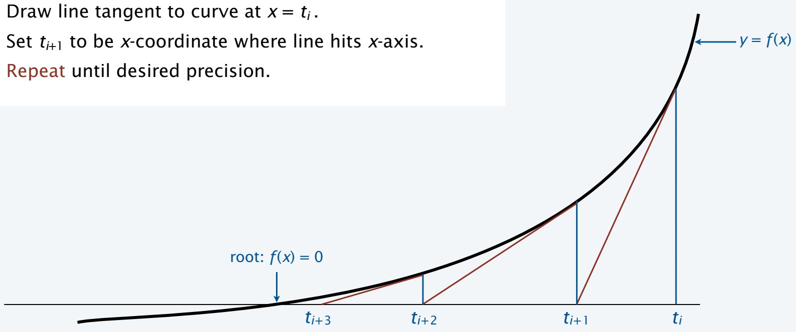
```
public class Sqrt
{
   public static void main(String[] args)
   {
      double EPS = 1E-15;   ← error tolerance (15 places)
      double c = Double.parseDouble(args[0]);
      double t = c;
      while (Math.abs(t - c/t) > t*EPS)
      {  t = (c/t + t) / 2.0;  }
      System.out.println(t);
   }
}
```

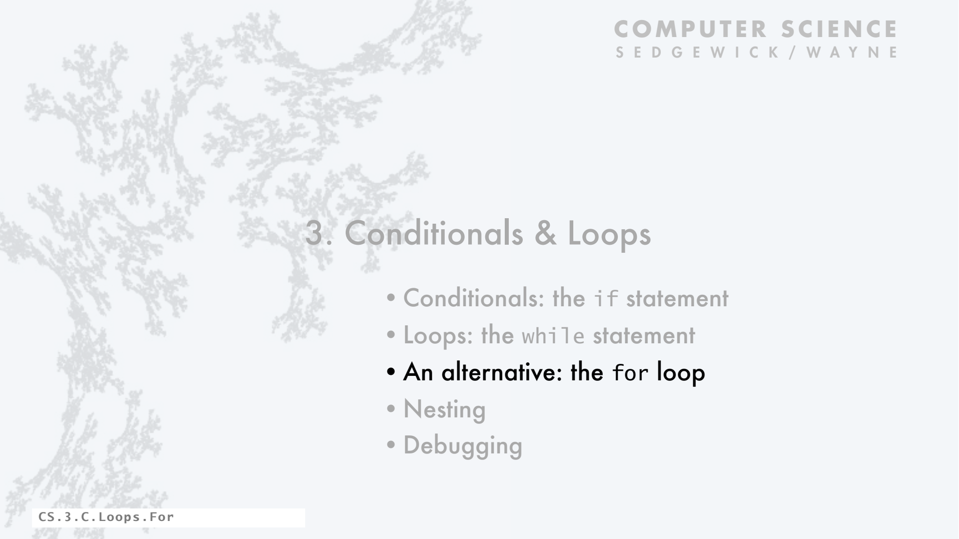
```
% java Sqrt 60481729
7777.0

% java Sqrt 2.0
1.414213562373095
```

Newton-Raphson method

Explanation (some math omitted)

- Goal: find root of function $f(x)$. ← use $f(x) = x^2 - c$ for $\sqrt{c}$
- 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.



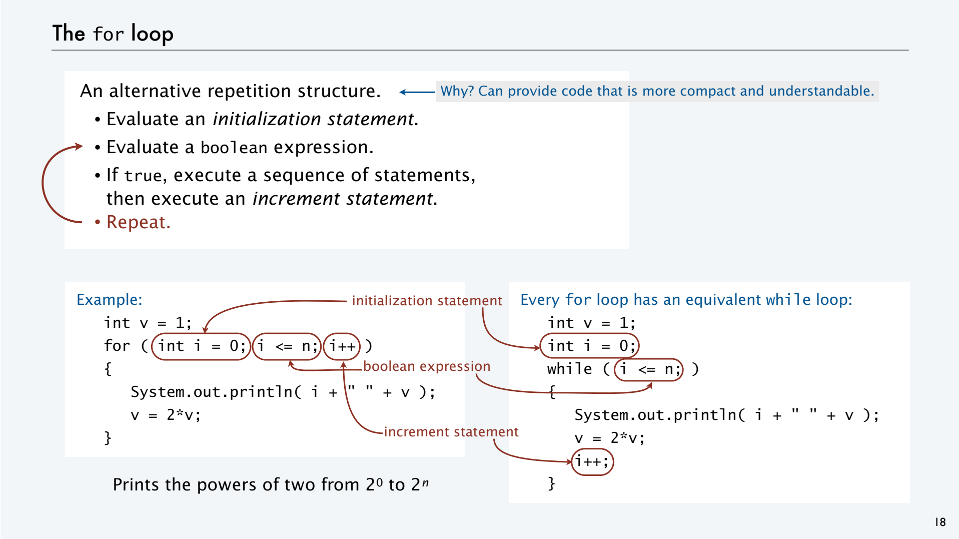


COMPUTER SCIENCE
SEDGEWICK / WAYNE

3. Conditionals & Loops

- Conditionals: the `if` statement
- Loops: the `while` statement
- **An alternative: the `for` loop**
- Nesting
- Debugging

CS.3.C.Loops.For



The `for` loop

An alternative repetition structure. ← Why? Can provide code that is more compact and understandable.

- Evaluate an *initialization statement.*
- Evaluate a `boolean` expression.
- If `true`, execute a sequence of statements, then execute an *increment statement.*
- Repeat.

Example:

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

initialization statement

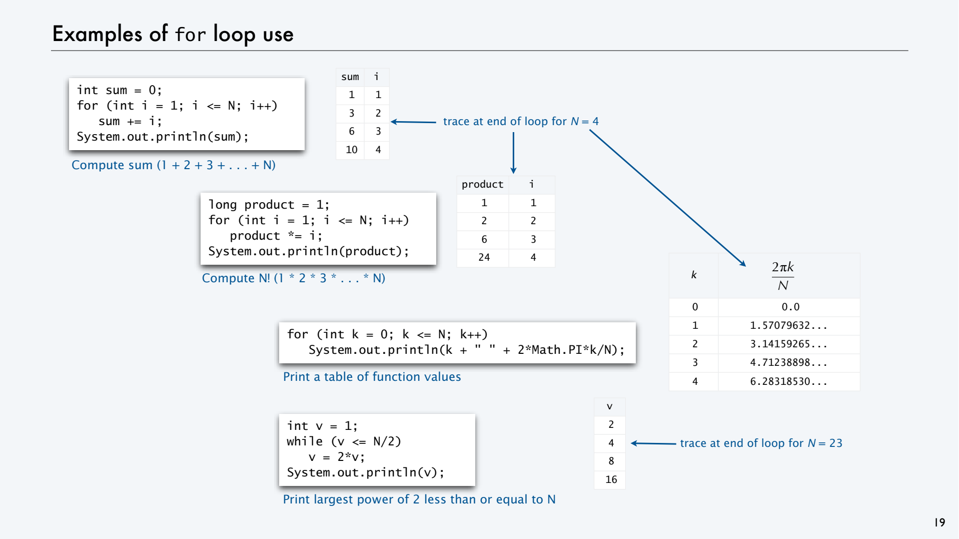
boolean expression

increment statement

Prints the powers of two from 2^0 to 2^n

Every `for` loop has an equivalent `while` loop:

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



Examples of `for` loop use

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

Compute sum (1 + 2 + 3 + . . . + N)

sum	i
1	1
3	2
6	3
10	4

trace at end of loop for $N = 4$

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

Compute N! (1 * 2 * 3 * . . . * N)

product	i
1	1
2	2
6	3
24	4

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

Print a table of function values

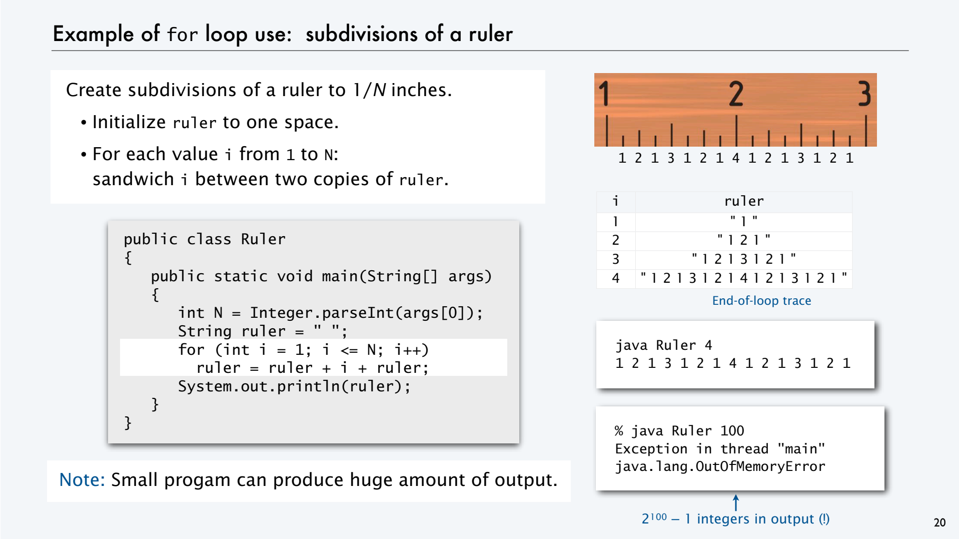
k	$\frac{2\pi k}{N}$
0	0.0
1	1.57079632...
2	3.14159265...
3	4.71238898...
4	6.28318530...

```
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

v
2
4
8
16

trace at end of loop for $N = 23$



Example of `for` loop use: subdivisions of a ruler

Create subdivisions of a ruler to $1/N$ inches.

- Initialize `ruler` to one space.
- For each value `i` from `1` to `N`: sandwich `i` between two copies of `ruler`.

```
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);
   }
}
```

Note: Small progam can produce huge amount of output.

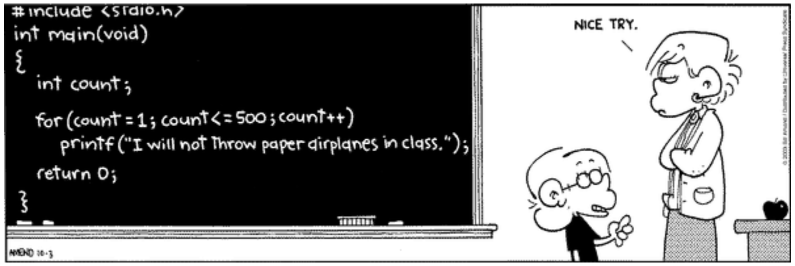
i	ruler
1	" 1 "
2	" 1 2 1 "
3	" 1 2 1 3 1 2 1 "
4	" 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 "

End-of-loop trace

```
java Ruler 4
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
```

```
% java Ruler 100
Exception in thread "main"
java.lang.OutOfMemoryError
```

$2^{100} - 1$ integers in output (!)


#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;
}
NICE TRY.
AMEND 10-3

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

Pop quiz on for loops (easy if you read exercise 1.3.13)

Q. What does the following program print?

```
public class PQfor
{
   public static void main(String[] args)
   {
      int f = 0, g = 1;
      for (int i = 0; i <= 10; i++)
      {
         System.out.println(f);
         f = f + g;
         g = f - g;
      }
   }
}
```

COMPUTER SCIENCE
SEDGEWICK / WAYNE

3. Conditionals & Loops

- Conditionals: the if statement
- Loops: the while statement
- An alternative: the for loop
- **Nesting**
- Debugging

Nesting conditionals and loops

Nesting

- Any "statement" within a conditional or loop may itself be a conditional or a loop statement.
- Enables complex control flows.
- Adds to challenge of debugging.

Example:

```
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++;
}
```

if-else statement
within a while loop
within a for loop

[Stay tuned for an explanation of this code.]

Example of nesting conditionals: Tax rate calculation

Goal. Given income, calculate proper tax rate.

income	rate
0 – $47,450	22%
$47,450 – $114,649	25%
$114,650 – $174,699	28%
$174,700 – $311,949	33%
$311,950 +	35%

```
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;
             }
        }
   }
```

if statement within an if statement

if statement within an if statement within an if statement

if statement within an if statement within an if statement within an if statement

Pop quiz on nested if statements

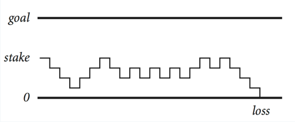
Q. Anything wrong with the following code?

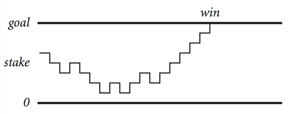
```
public class PQif
{
   public static void main(String[] args)
   {
      double income = Double.parseDouble(args[0]);
      double rate = 0.35;
      if (income <  47450) rate = 0.22;
      if (income < 114650) rate = 0.25;
      if (income < 174700) rate = 0.28;
      if (income < 311950) rate = 0.33;
      System.out.println(rate);
   }
}
```

Gambler's ruin problem

A gambler starts with $*stake* and places $1 fair bets.

- Outcome 1 (loss): Gambler goes broke with $0.
- Outcome 2 (win): Gambler reaches $*goal.*

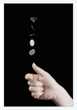




Q. What are the chances of winning?
Q. How many bets until win or loss?

One approach: Monte Carlo simulation.

- Use a *simulated coin flip.*
- Repeat and compute statistics.



Example of nesting conditionals and loops: Simulate gamber's ruin

Gambler's ruin simulation

- Get command-line parms.
- Run all the experiments.
  - Run one experiment.
    - Make one bet.
  - If goal met, count the win.
- Print #wins and # trials.

```
public class Gambler
{
    public static void main(String[] args)
    {
        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++)
        {
           int t = stake;
           while (t > 0 && t < goal)
           {
              if (Math.random() < 0.5) t++;
              else                     t--;
           }
           if (t == goal) wins++;
        }
        StdOut.println(wins + " wins of " + trials);
    }
}
```

for loop

while loop within a for loop

if statement within a while loop within a for loop

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

Digression: simulation and analysis

Facts (known via mathematical analysis for centuries)

- Probability of winning = stake ÷ goal.
- Expected number of bets = stake × desired gain.

Christiaan Huygens
1629-1695

Early scientists were fascinated by the study of games of chance.

Example

- 20% chance of turning $500 into $2500.
- Expect to make 1 *million* $1 bets.

500/2500 = 20%

500*(2500 - 500) = 1,000,000

stake goal trials

```
% 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
```

uses about 1 *billion* coin flips

Remarks

- Computer simulation can help validate mathematical analysis.
- For this problem, mathematical analysis is simpler (if you know the math).
- For more complicated variants, computer simulation may be the *best* plan of attack.

COMPUTER SCIENCE
SEDGEWICK / WAYNE

3. Conditionals & Loops

- Conditionals: the `if` statement
- Loops: the `while` statement
- An alternative: the `for` loop
- Nesting
- **Debugging**

CS.3.E.Loops.Debugging

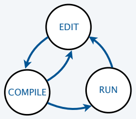
Debugging

is 99% of program development in any programming language, *even for experts*.

Bug: A mistake in a program.

Debugging: The process of eliminating bugs.

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*

Impossible ideal: "Please compile, execute, and debug my progam." ← Why is this impossible? Stay tuned.

Bottom line: Programming is primarily a *process* of finding and fixing mistakes.

Debugging

is challenging because conditionals and loops *dramatically increase* the number of possible outcomes.

program structure	*no loops*	*N conditionals*	*1 loop*
number of possible execution sequences	1	2^N	no limit

Most programs contain *numerous* conditionals and loops, with nesting.

Good news. Conditionals and loops provide structure that helps us understand our programs.

Old and low-level languages have a *goto* statement that provides arbitrary structure. Eliminating *gotos* was controversial until Edsgar Dijkstra published the famous note "*Goto considered harmful*" in 1968.

> *" The quality of programmers is a decreasing function of the number of goto statements in the programs they produce. "*
>
> *– Edsgar Dijkstra*

Debugging a program: a running example

Problem: Factor a large integer *N*.

Application: Cryptography.

Suprising fact: Security of internet commerce depends on difficulty of factoring large integers.

3,757,208 = 2 × 2 × 2 × 7 × 13 × 13 × 397

98 = 2 × 7× 7

17 = 17

11,111,111,111,111,111 = 2,071,723 × 5,363,222,357

Method

- Consider each integer *i* less than *N*
- While *i* divides N evenly
  Print *i* (it is a factor of *N*).
  Replace *N* with *N*/*i* .

Rationale:
1. Any factor of *N*/*i* is a factor of *N*.
2. *i* may be a factor of *N*/*i*.

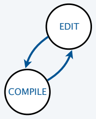
```
public class Factors
{
   public static void main(String[] args)
   {
      long N = Long.parseLong(args[0])
      for (i = 0; i < N; i++)
      {
         while (N % i == 0)
            System.out.print(i + " ")
            N = N / i
      }
   }
}
```

This program has bugs!

Debugging a program: syntax errors

Is your program a legal Java program?

- Java compiler can help you find out.
- Use javac to find the first error.
- Repeat.
- Result: An executable Factors.class file



Trying to tell a computer what to do

```
% javac Factors.java
Factors.java:5: ';' expected
       long N = Long.parseLong(args[0])
                                       ^
...
```

```
% javac Factors.java
Factors.java:6: cannot find symbol
symbol  : variable i
location: class FactorsX
          for (      i = 0; i < N; i++)
                     ^
...
```

```
% javac Factors.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 ;
      }
   }
}
```

need to declare type of i

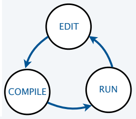
need terminating semicolons

This legal program still has bugs!

Debugging a program: runtime and semantic errors

Does your legal Java program do what you want it to do?

- You need to run it to find out.
- Use java runtime to find the first error.
- Fix and repeat.



```
% javac Factors.java
% java Factors      ← oops, need argument
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 0
       at Factors.main(Factors.java:5)
```

```
% java Factors 98
Exception in thread "main"
java.lang.ArithmeticException: / by zero
       at Factors.main(Factors.java:8)
```

```
% 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
```

```
% java Factors 98
2 7 7%
```

98 = 2 × 7× 7 ✓

```
public class Factor
{
   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 are not factors

need braces

This working program still has bugs!

Debugging a program: testing

Does your legal Java program *always* do what you want it to do?

- You need to test on many types of inputs it to find out.
- Add trace code to find the first error.
- Fix the error.
- Repeat.

```
% java Factors 98
2 7 7%     ← need newline
```

```
% java Factors 5
  ←          ??? no output
```

```
% java Factors 6
2  ←      ??? where's the 3?
```

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

AHA! Need to print out *N* (if it is not 1).

```
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 a program: testing

Does your legal Java program *always* do what you want it to do?

- You need to test on many types of inputs it to find out.
- Add trace code to find the first error.
- Fix the error.
- Repeat.

???
%$%@$#!!
forgot to recompile

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

```
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();
   }
}
```

Note: This working program still has a bug (stay tuned).

Debugging a program: performance

Is your working Java program fast enough to solve your problem?

- You need to test it on increasing problem sizes to find out.
- May need to change the algorithm to fix it.
- Repeat.

change the *algorithm*: no need to check when $i \cdot i > N$ since all smaller factors already checked

Method

- Consider each integer $i \leq N/i$
- While *i* divides N evenly
  print *i* (it is a factor of *N*)
  replace *N* with *N/i* .

might work,
but way too slow

```
% java Factors 11111111
11 73 101 137
% java Factors 11111111111
21649 513239
% java Factors 11111111111111
11 239 4649 909091
% java Factors 11111111111111111
2071723 5363222357
```

immediate

```
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();
   }
}
```

implement
the change

Debugging a program: performance analysis

Q. How large an integer can I factor?

```
% java Factors 9201111169755555703
9201111169755555703
```

digits in largest factor	`i < N`	`i <= N/i`
3	instant	instant
6	instant	instant
9	77 seconds	instant
12	21 hours†	instant
15	2.4 years†	2.7 seconds
18	2.4 millenia†	92 seconds

† estimated, using analytic number theory

```
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();
   }
}
```

Lesson. Performance matters!

experts are still trying to develop better algorithms for this problem

Note. Internet commerce is still secure: it depends on the difficulty of factoring 200-digit integers.

Debugging your program: summary

Program development is a *four*-step process, with feedback.

Telling a computer what to do
when you know what you're doing

EDIT your program.

syntax error

COMPILE your program to create an executable file.

runtime error

semantic error

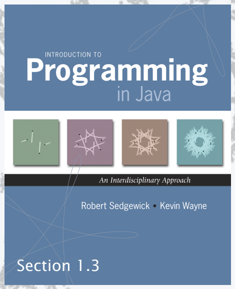
RUN your program to test that it works as you imagined.

performance error

TEST your program on realistic and real input data.

SUBMIT your program for independent testing and approval.

COMPUTER SCIENCE
SEDGEWICK / WAYNE



Section 1.3

http://introcs.cs.princeton.edu

3. Conditionals & Loops