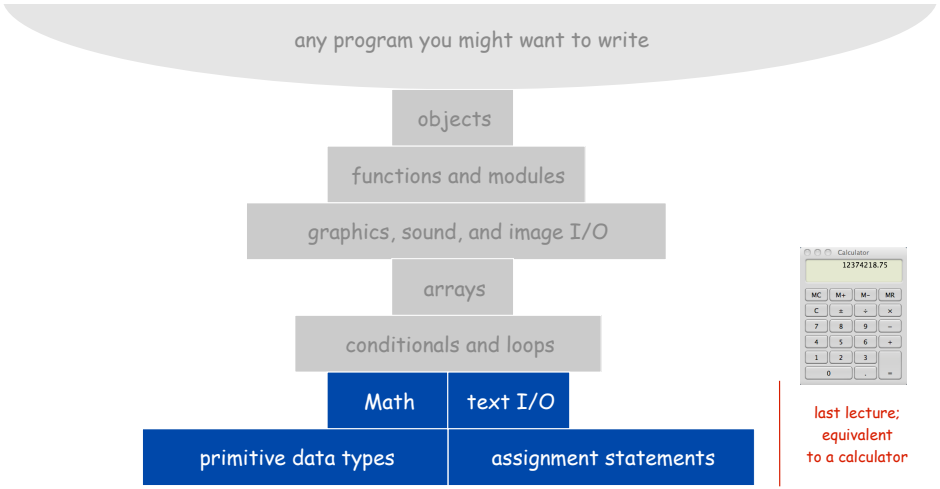
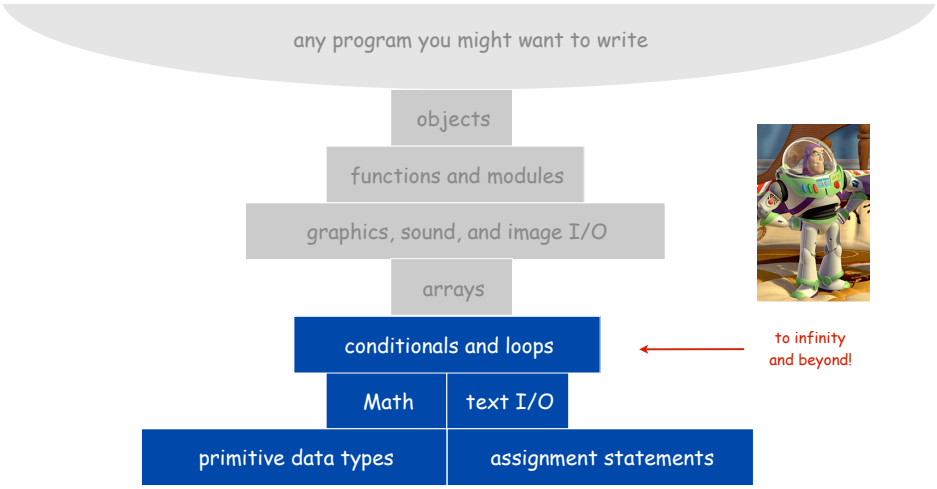
1.3 Conditionals and Loops



last lecture; equivalent to a calculator

1.3 Conditionals and Loops

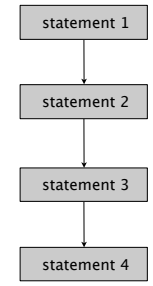


to infinity and beyond!

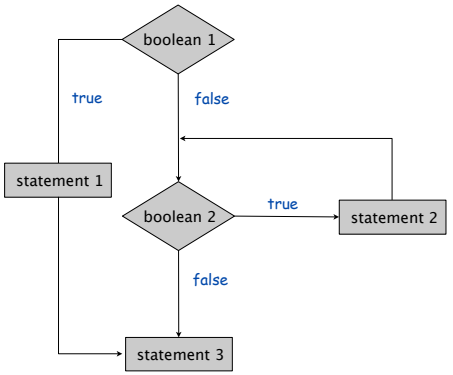
Conditionals and Loops

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.

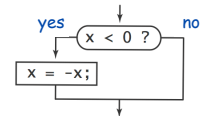
- Evaluate a `boolean` expression.
- If `true`, execute some statements.
- `else` option: If `false`, execute other statements.

boolean expression

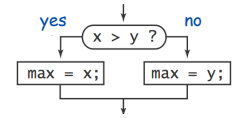
```
if ( x > y )
{
   int t = x;
   x = y;
   y = t;
}
```

sequence of statements

```
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

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

absolute value

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

maximum

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

2-sort

x > y before

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

x < y after

```
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
{
   System.out.println((-b + Math.sqrt(discriminant))/2.0);
   System.out.println((-b - Math.sqrt(discriminant))/2.0);
}
```

error check for quadratic formula

Loops

While Loop

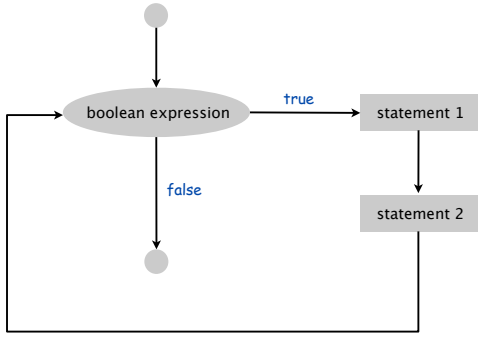
The `while` loop. A common repetition structure.

- Check a boolean expression.
- Execute a sequence of statements.
- Repeat.

loop continuation condition

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

loop body



While Loop Example: 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(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

```
1
2
4
8
16
32
64
```

`n = 6`

Powers of Two (full program)

```
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;
      }
   }
}
```

print ith power of two

```
% java PowersOfTwo 3
1
2
4
8

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


While Loop Challenge

Anything wrong with the following code?

```
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;
   }
}
```

While Loop Example: Square Root

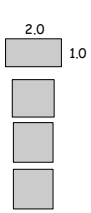
Goal. Implement `Math.sqrt()`.

```
% java Sqrt 60481729
7777.0
```

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.

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

"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.

```
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);
   }
}
```

error tolerance

```
% java Sqrt 2.0
1.414213562373095
```

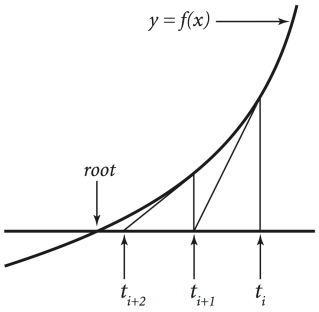
15 decimal digits of accuracy in 5 iterations

Newton-Raphson Method

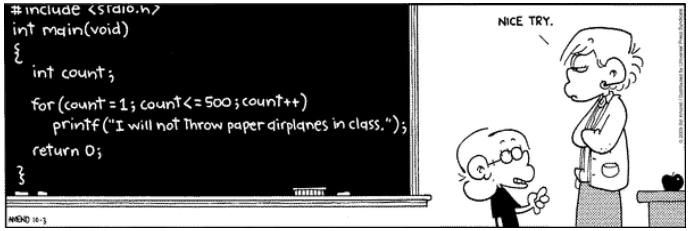
Square root method explained (some math omitted).

- 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 √c



The For Loop



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

The For Loop

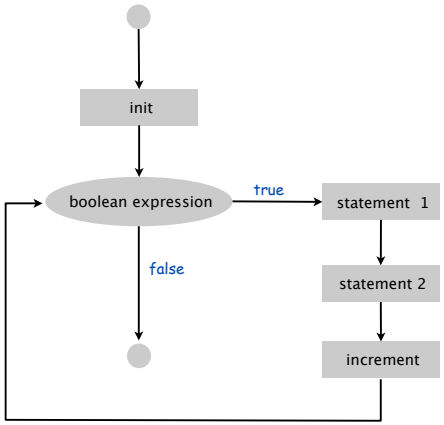
The `for` loop. Another common repetition structure.

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

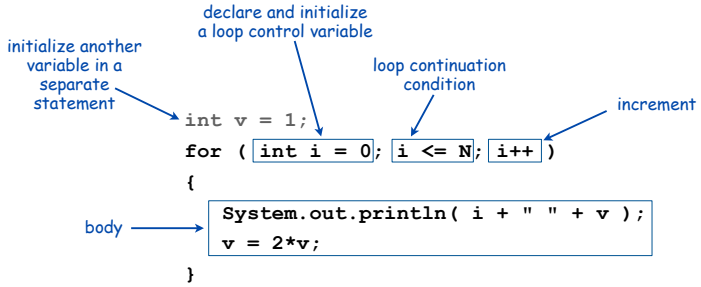
loop continuation condition

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

body



Anatomy of a `for` Loop



prints table of powers of two

Anatomy of a `for` Loop

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

v	i	output
1		
1	0	
1	0	0 1
2	0	
2	1	
2	1	1 2
4	1	
4	2	
4	2	2 4
8	2	
8	3	
8	3	3 8

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++;
}
```

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`.

```
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);
   }
}
```

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

end-of-loop trace

For Loops: Subdivisions of a Ruler

```
% java Ruler 1
 1

% java Ruler 2
 1 2 1

% java Ruler 3
 1 2 1 3 1 2 1

% java Ruler 4
 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

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

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

Observation. Loops can produce a huge amount of output!

Loop Examples

```
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

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

print a table of function values

N = 4

```
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

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

```
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

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

Nested If-Else Statements

Need all those braces? Not always:

```
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

```
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).

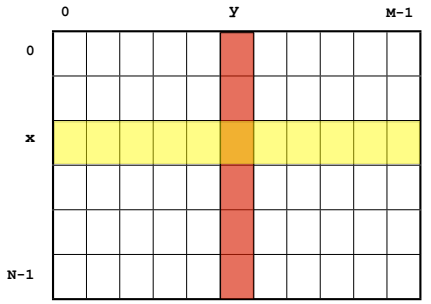
Nested If Statement Challenge

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;
```


Nested `for` loops

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




```
for (x = 0; x < N; x++)
     for (y = 0; y < M; y++)
            Do something at entry (x,y);
```

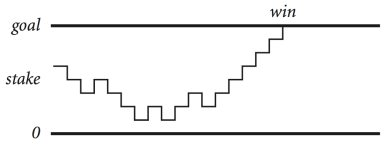
Nesting Example: Gambler's Ruin

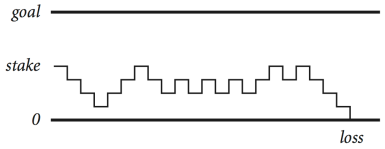
Gambler's ruin. Gambler starts with $stake and places $1 fair bets until going broke or reaching $goal.

- 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.





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]);

     // Count wins among args[2] trials.
     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);
   }
}
```

if statement within a while loop within a for loop

Digression: Simulation and Analysis

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
```

after a substantial wait....

Fact. Probability of winning = stake ÷ goal.

Fact. Expected number of bets = stake × desired gain.

Ex. 20% chance of turning $500 into $2500, 500/2500 = 20%

but expect to make one million $1 bets. 500*(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

Debugging Example

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

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

$98 = 2 \times 7^2$

$17 = 17$

Note: 1 is not prime. (else it would have to be in every factorization)

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

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

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.

```
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
      }
   }
}
```

Check whether i is a factor.

if i is a factor print it and divide it out

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`.

```
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
      }
   }
}
```

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, need argument

you will see this message!

Debugging: Semantic Errors

Semantic error. Legal but wrong Java program.

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

```
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)
```

Debugging: Semantic Errors

Semantic error. Legal but wrong Java program.

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

```
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

Debugging: Semantic Errors

Semantic error. Legal but wrong Java program.

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

```
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 2 2 2
```

??? infinite loop

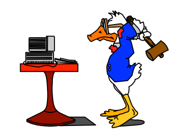
Debugging: Semantic Errors

Semantic error. Legal but wrong Java program.

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

```
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 (run-time) error:
indents do not imply braces



Debugging: The Beat Goes On

Success? Program factors 98 = 2 7 7.

- Time to try it for other 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++)
      {  // 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
%                ← ??? no output
% java Factors 6
2 %              ← ??? where's the 3?
```

Debugging: The Beat Goes On

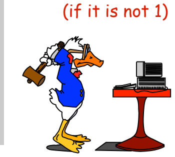
Success? Program factors 98 = 2 7 7.

- Time to try it for other 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);
      }
   }
}
```

```
% 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)



Debugging: Success?

Success? Program seems to work

- Add code for corner case, add comments.
- Remove trace to try larger inputs

```
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).

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

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

Corner case: print largest factor (and new line)

```
% 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
```


Debugging: Performance Errors

Performance error. 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();
   }
}
```

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

very long wait

Debugging: Performance Errors

Performance error. 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 * 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*i > N since no larger factors left

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


Debugging: Back to Semantic Errors!

Fresh semantic error. Fast program (now), but new error.

- Was performance fix exactly right?
- Again, consider (possibly new) corner cases.

```
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 error.

- Was performance fix exactly right?
- Again, consider (possibly new) corner cases.

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

```
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();
   }
}
```

Execute loop body if i * i == N

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....

digits in largest factor	(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, using analytic number theory

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

Debugging Your Program

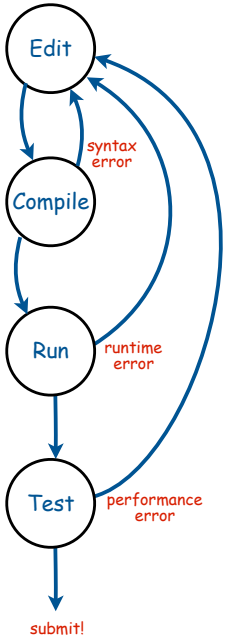
Debugging Your Program. [summary]

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 runtime (semantic) 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

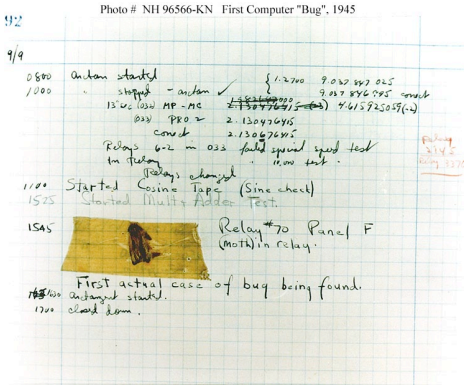
Good news: Can use computer to test program.

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