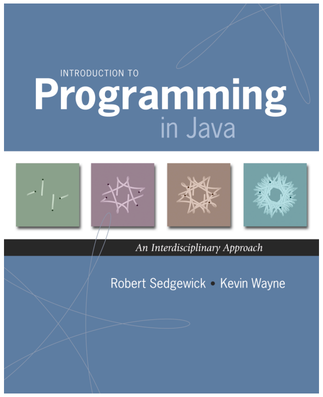
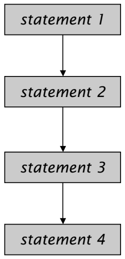
1.3 Conditionals and Loops



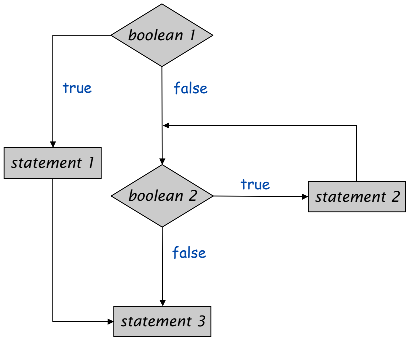
Control Flow

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

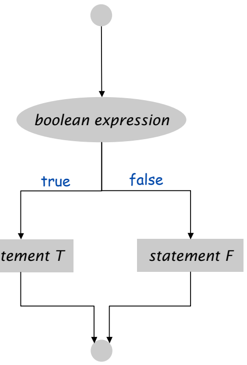
If Statement

The `if` statement. A common branching structure.

- Check `boolean` condition.
- If `true`, execute some statements.
- If `false`, execute other statements.

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

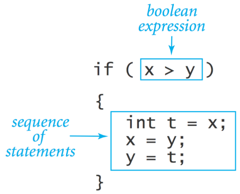
can be any sequence of statements



If Statement

The `if` statement. A common branching structure.

- Check `boolean` condition.
- If `true`, execute some statements.
- If `false`, execute other statements.

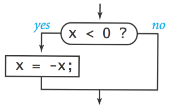


boolean expression

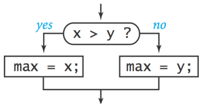
sequence of statements

```
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");
   }
}
```

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

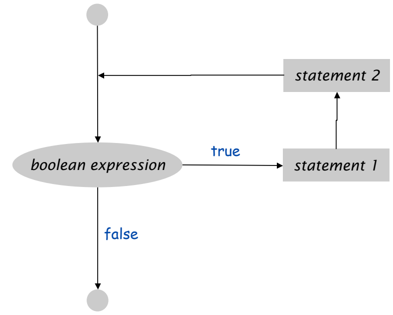
The `while` loop. A common repetition structure.

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

loop continuation condition

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

loop body



While Loops: Powers of Two

Ex. Print first `n` powers of 2.

- Increment `i` from `1` 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`

Click for demo

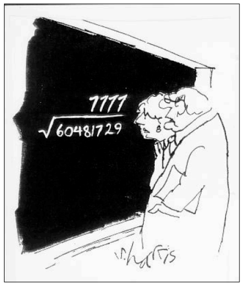
Powers of Two

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

print ith power of two

```
% java PowersOfTwo 4
1
2
4
8

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

A Wonderful Square Root



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

```
% 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.

t_0			=	2.0
t_1	=	$\frac{1}{2}(t_0 + \frac{2}{t_0})$	=	1.5
t_2	=	$\frac{1}{2}(t_1 + \frac{2}{t_1})$	=	1.4166666666666665
t_3	=	$\frac{1}{2}(t_2 + \frac{2}{t_2})$	=	1.4142156862745097
t_4	=	$\frac{1}{2}(t_3 + \frac{2}{t_3})$	=	1.4142135623746899
t_5	=	$\frac{1}{2}(t_4 + \frac{2}{t_4})$	=	1.414213562373095

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 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
```

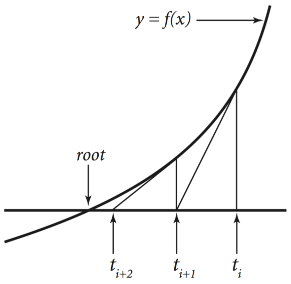
15 decimal digits of accuracy in 5 iterations

Newton-Raphson Method

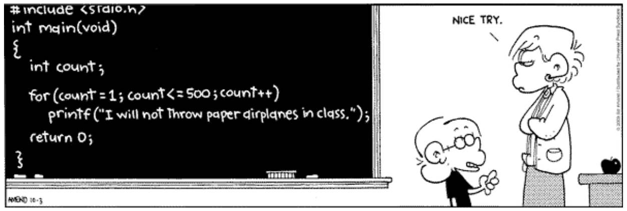
Square root method explained.

- 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



```
#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

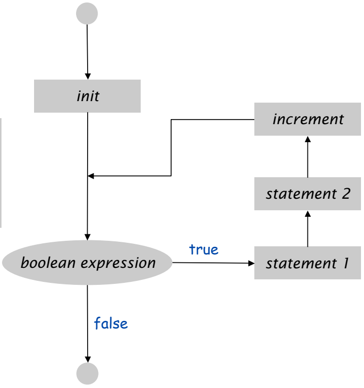
The `for` loop. Another common repetition structure.

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

loop continuation condition

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

body



For Loops: Subdivisions of a Ruler

Create subdivision of a ruler.

- Initialize `ruler` to empty string.
- 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 "

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
```

Observation. Loops can produce a huge amount of output!

Loop Examples

print powers of two	`int v = 1;`
	`for (int i = 0; i <= N; i++)`
	`{`
	`   System.out.println(i + “ “ + v);`
	`   v = 2*v;`
	`}`

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

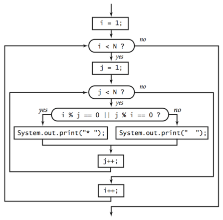
Loops enable you to do an operation n times using only 2 lines of code.

```
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

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%

```
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

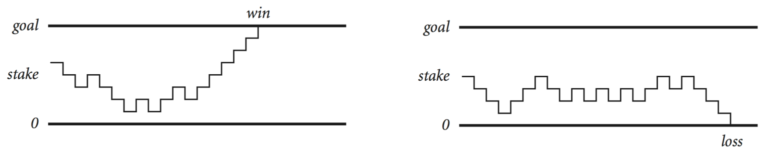
Gambler's Ruin

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 trials = Integer.parseInt(args[2]);
      int wins   = 0;
      // repeat experiment N times
      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);
   }
}
```

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 few hours of computing....

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

Remark. Both facts can be proved mathematically; for more complex scenarios, computer simulation is often the best plan of attack.

Debugging

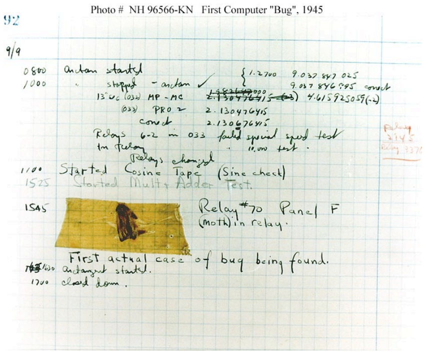


Photo # NH 96566-KN First Computer "Bug", 1945

92

9/9

0800 Anctan started {1.2700 9.037 847 025

1000 " stopped - anctan ✓ 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 " 11.000 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.

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

Admiral Grace Murray Hopper

Debugging a Program

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

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

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

Application. Break RSA cryptosystem.

Debugging a Program: Syntax Errors

Syntax error. Illegal Java program.

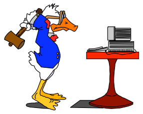
- Compiler error messages help locate problem.
- Eventually, a file named `Factors.class`.

Check if i is a factor.

```
public class Factors1 {
   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
      }
   }
}
```

As long as i is a factor, divide it out.

Compile-time error



Debugging a Program: Semantic Errors

Semantic error. Legal but wrong Java program.

- Use "`System.out.println`" method to identify problem.

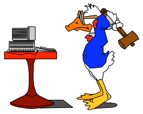
Check if i is a factor.

```
public class Factors2 {
   public static void main(String[] args)  {
      long N = Long.parseLong(args[0]);
      for (long i = 2; i < N; i++)  {
         while (N % i == 0)
            System.out.print(i + " ");
            N = N / i;
      }
   }
}
```

no output (17) or infinite loop (49)

As long as i is a factor, divide it out.

Run-time error



Debugging a Program: Performance Errors

Performance error. Correct program but too slow.

- Use profiling to discover bottleneck.
- Devise better algorithm.

```
public class Factors3 {
   public static void main(String[] args)  {
      long N = Long.parseLong(args[0]);
      for (long i = 2; i <= N; i++) {
         while (N % i == 0) {
            System.out.print(i + " ");
            N = N / i;
         }
      }
   }      too slow for large N  (999,999,937)
}
```

Check if i is a factor.

As long as i is a factor, divide it out.

Performance error

Debugging a Program: Success

Fact. If N has a factor, it has one less than or equal to its square root.
Impact. Many fewer iterations of `for` loop.

```
public class Factors {
   public static void main(String[] args)  {
      long N = Long.parseLong(args[0]);
      for (long i = 2; i*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();
   }
}
```

Check if i is a factor.

As long as i is a factor, divide it out.

Corner case: biggest factor occurs once.

Debugging a Program: Trace

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

3757208/8

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				

Debugging a Program: Analysis

Q. How large an integer can I factor?

```
% java Factors 168
2 2 2 3 7

% java Factors 3757208
2 2 2 7 13 13 397

% java Factors 9201111169755555703
9201111169755555703
```

after a few minutes of computing….

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

Programming in Java

Programming in Java. [a slightly more realistic view]

1. Create the program.

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

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

4. Enjoy the satisfaction of a working program!

Debugging a Program

Debugging. Cyclic process of editing, compiling, and fixing errors.

- Always a logical explanation.
- What would the machine do?
- Explain it to the teddy bear.

You will make many mistakes as you write programs. It's normal.

> As soon as we started programming, we found out to our surprise that it wasn't as easy to get programs right as we had thought. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs. - Maurice Wilkes

> If I had 8 hours to chop down a tree, I would spend 6 hours sharpening an axe. - Abraham Lincoln

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`

U.S.S. Grace Murray Hopper