2. Conditionals and loops

Conditionals and Loops

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

Context: basic building blocks for programming

Any program you might want to write

- Objects
- Functions and modules
- Graphics, sound, and image I/O
- Arrays
- Conditionals and loops
- Math
- Text I/O
- Primitive data types
- Assignment statements

This lecture: to infinity and beyond!

Control flow

- The sequence of statements that are actually executed in a program.
- Conditionals and loops enable us to choreograph control flow.
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 \( x < 0 \) \( x = -x; \)

Replaces \( x \) with the absolute value of \( x \)

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

Computes the maximum of \( x \) and \( y \)

Example of *if* statement use: simulate a coin flip

```java
class Flip {
    public static void main(String[] args) {
        if (Math.random() < 0.5)
            System.out.println("Heads");
        else
            System.out.println("Tails");
    }
}
```

Example of *if* statement use: 2-sort

**Q.** What does this program do?

```java
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;
        }
        System.out.println(a);
        System.out.println(b);
    }
}
```

**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.

```java
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]);
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}
```
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]);
        if (b < a) {
            int t = a; a = b; b = t;  // makes a smaller than b
        }
        if (c < a) {
            int t = a; a = c; c = t;  // makes a smaller than both b and c
        }
        if (c < b) {
            int t = b; b = c; c = t;  // makes b smaller than c
        }
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}
```

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);
    }
}
```

Good programming practice. Use conditionals to check for and avoid runtime errors.
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:
```java
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 $2^i$ to $2^n$.

[stay tuned for a trace]

Example of while loop use: print powers of two

A trace is a table of variable values after each statement.

<table>
<thead>
<tr>
<th>i</th>
<th>v</th>
<th>i &lt;= n</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>true</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>true</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>true</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>true</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>true</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>true</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>true</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>false</td>
</tr>
</tbody>
</table>

Prints the powers of two from $2^i$ to $2^n$.

Pop quiz on while loops

Q. Anything wrong with the following code?

```java
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;
        }
    }
}
```

A. Yes! Needs braces.

Pop quiz on while loops

Q. Anything wrong with the following code?

```java
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;
        }
    }
}
```

Q. What does it do (without the braces)?

A. Goes into an infinite loop.

```
% java PQwhile 6
4
2
8
16
32
64
```
Example of while loop use: implement Math.sqrt()

**Goal. Implement square root function.**

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 $c / t_i$.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$t_i$</th>
<th>$2 / t_i$</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1.5</td>
<td>1.333333</td>
<td>1.416667</td>
</tr>
<tr>
<td>2</td>
<td>1.4166667</td>
<td>1.4117647</td>
<td>1.4142157</td>
</tr>
<tr>
<td>3</td>
<td>1.4142157</td>
<td>1.4142114</td>
<td>1.4142136</td>
</tr>
<tr>
<td>4</td>
<td>1.4142136</td>
<td>1.4142136</td>
<td></td>
</tr>
</tbody>
</table>

computing the square root of 2 to seven places

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

Newton-Raphson method

**Explanation (some math omitted)**

- **Goal:** find root of function $f(x)$ (value of $x$ for which $f(x) = 0$).
- **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.**
2. Conditionals & Loops

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

The for loop

An alternative repetition structure.
- Evaluate an initialization statement.
- Evaluate a boolean expression.
- If true, execute a sequence of statements, then execute an increment statement.
- Repeat.

Example:
```java
int v = 1;
for (int i = 0; i <= n; i++)
{  
    System.out.println(i + " + v");  
    v = 2*v;
    i++;  
}
```
Prints the powers of two from $2^0$ to $2^n$

Example of for loop use

Create subdivisions of a ruler to 1/N inches.
- Initialize ruler to one space.
- For each value $i$ from 1 to N/2:
  - sandwich $i$ between two copies of ruler.

```
public class Ruler
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        String ruler = "\n        for (int i = 1; i <= N; i++)
        {  
            ruler = ruler + i + ruler;
            System.out.println(ruler);
        }
    }
}
```

Note: Small program can produce huge amount of output.
Q. What does the following program print?

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

Beginning of loop trace:

```
<table>
<thead>
<tr>
<th>i</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>21</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>34</td>
<td>21</td>
</tr>
<tr>
<td>10</td>
<td>55</td>
<td>34</td>
</tr>
</tbody>
</table>
```

Q. What does the following program print?

```java
public class M{
    public static void main(String[] args) {
        int f = 0, g = 1;
        for (int i = 0; i <= 10; i++)
            System.out.println(f);
        g = f - g;
    }
}
```
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 t = 0; t < trials; t++)
{
    int cash = stake;
    while (cash > 0 & cash < goal)
        if (Math.random() < 0.5) cash++;
    else
        cash--;
    if (cash == goal) wins++;
}
[Stay tuned for an explanation of this code.]

Example of nesting conditionals: Tax rate calculation

Goal. Given income, calculate proper tax rate.

<table>
<thead>
<tr>
<th>income</th>
<th>rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 47450</td>
<td>22%</td>
</tr>
<tr>
<td>$47,450 - $114,649</td>
<td>25%</td>
</tr>
<tr>
<td>$114,650 - $174,699</td>
<td>28%</td>
</tr>
<tr>
<td>$174,700 - $311,949</td>
<td>33%</td>
</tr>
<tr>
<td>$311,950+</td>
<td>35%</td>
</tr>
</tbody>
</table>

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 < 570250) rate = 0.35;
else rate = 0.35;

Pop quiz on nested if statements

Q. Anything wrong with the following code?

```java
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;
        else if (income < 114650) rate = 0.25;
        else if (income < 174700) rate = 0.28;
        else if (income < 311950) rate = 0.33;
        System.out.println(rate);
    }
}
```

A. Yes! Need else clauses. Without them, code is equivalent to:

```java
public class PQif {
    public static void main(String[] args) {
        double income = Double.parseDouble(args[0]);
        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;
        System.out.println(rate);
    }
}
```

Note. Braces are not needed in this case, but BE CAREFUL when nesting if-else statements because of potential ambiguity (see Q&A p. 75).
Gambler's ruin problem

A gambler starts with $\text{stake}$ and places $1$ fair bets. 
- Outcome 1 (loss): Gambler goes broke with $\text{stake}$.
- Outcome 2 (win): Gambler reaches $\text{goal}$.

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.

Digression: simulation and analysis

Facts (known via mathematical analysis for centuries) 
- Probability of winning = $\text{stake} / \text{goal}$.
- Expected number of bets = $\text{stake} \times$ desired gain.

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

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

Gamblers ruin simulation

```
public class Gambler {
    public static void main(String[] args) {
        // Get command-line arguments.
        int stake = Integer.parseInt(args[0]);
        int goal = Integer.parseInt(args[1]);
        int trials = Integer.parseInt(args[2]);

        // Run all the experiments.
        for (int t = 0; t < trials; t++) {
            int cash = stake;
            for (int x = 0; x < trials; x++) {
                int cash = cash * (Math.random() > 0.5) ? cash++ : cash--;
            }
            if (x >= goal) wins++;
        }
        System.out.println(wins + " wins of "+ trials);
    }
}
```

Example

- 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.

Early scientists were fascinated by the study of games of chance.
2. Conditionals & Loops

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

“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 program.” 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.

<table>
<thead>
<tr>
<th>program structure</th>
<th>no loops</th>
<th>n conditionals</th>
<th>1 loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of possible execution sequences</td>
<td>1</td>
<td>$2^n$</td>
<td>no limit</td>
</tr>
</tbody>
</table>

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 Edgar 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.”

− Edgar Dijkstra

Debugging a program: a running example

Problem: Factor a large integer $n$.
Application: Cryptography.

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

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;
            }
        }
    }
}

This program has bugs!
Debugging a program: syntax errors

Is your program a legal Java program?
• Java compiler can help you find out.
• Find the first compiler error (if any).
• Repeat.
• Result: An executable Factors.class file

```java
public class Factors {
    public static void main(String[] args) {
        long n = Long.parseLong(args[0]);
        n = n / 1;
        System.out.println(n);
    }
}
```

```java
% java Factors.java
% java Factors
long n = Long.parseLong(args[0]);
```

```java
% java Factors.java:5: cannot find symbol
symbol: variable i
location: class Factors
for (  
    ^
% java Factors.java:4: cannot find symbol
symbol: variable n
location: class Factors
for (
    ^
% java Factors.java:2: cannot find symbol
symbol: variable m
location: class Factors
for (  
    ^
```

Always do what you want it to do.

You need to start at 2 since 0 and 1 are not factors.

```java
% java Factors 98
98 = 2 x 7
```

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.
• Find the first runtime error (if any).
• Fix and repeat.

```java
public class Factor {
    public static void main(String[] args) {
        long n = Long.parseLong(args[0]);
        for (int i = 2; i < n; i++)
            if (n % i == 0)
                System.out.print(i + " ");
        n = n / 1;
    }
}
```

```java
% java Factors 98
2 7 1
```

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 to find out.
• Add trace code to find the first error.
• Fix the error.
• Repeat.

```java
public class Factors {
    public static void main(String[] args) {
        long n = Long.parseLong(args[0]);
        for (int i = 2; i < n; i++)
            if (n % i == 0)
                System.out.print(i + " ");
        n = n / 1;
    }
}
```

```java
% java Factors:java
% java Factors
long n = Long.parseLong(args[0]);
```

```java
% java Factors:java:5: expected
long n = Long.parseLong(args[0]);
```

```java
% java Factors 98
2 7 1
```

```java
% java Factors 98
98 = 2 x 7
```
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.

Method
• Consider each integer \( i \leq n/\sqrt{n} \) since all smaller factors already checked.
• While \( i \) divides \( n \) evenly, print \( i \) (it is a factor of \( n \)).
• Replace \( n \) with \( n/i \).

public class Factors
{
  public static void main(String[] args)
  {
    long n = Long.parseLong(args[0]);
    for (int 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();
  }
}

Lesson. Performance matters!

Q. How large an integer can \( i \) factor?

\[ \text{digits in largest factor} \]
\[ \begin{array}{|c|c|c|}
\hline
\text{digit} & 1 & N/1 \\
\hline
1 & \text{instant} & \text{instant} \\
6 & \text{instant} & \text{instant} \\
9 & 77 \text{ seconds} & \text{instant} \\
12 & 21 \text{ hours} & \text{instant} \\
15 & 2.4 \text{ years} & 2.7 \text{ seconds} \\
18 & 2.4 \text{ millennia} & 92 \text{ seconds} \\
\hline
\end{array} \]

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.

EDIT your program.

COMPILE your program to create an executable file.

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

TEST your program on realistic and real input data.

SUBMIT your program for independent testing and approval.

2. Conditionals & Loops