CS 112 Introduction to Programming
(Spring 2012)

Lecture #5: Conditionals and Loops

Zhong Shao

Department of Computer Science
Yale University
Office: 314 Watson

http://flint.cs.yale.edu/cs112

Acknowledgements: some slides used in this class are taken directly or adapted from those accompanying the textbook: Introduction to Programming in Java: An Interdisciplinary Approach by Robert Sedgewick and Kevin Wayne (Copyright 2002-2010)
A Foundation 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

Last lecture: equivalent to a calculator
A Foundation 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

to infinity and beyond!
Control Flow

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

straight-line control flow

control flow with conditionals and loops
Conditionals
The if statement. A common branching structure.
- Evaluate a boolean expression.
- If true, execute some statements.
- If false, execute other statements.

```java
if (boolean expression) {
    statement T;
} else {
    statement F;
}
```
The if statement. A common branching structure.

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

Ex. Take different action depending on value of variable.

```java
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

<table>
<thead>
<tr>
<th>Description</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>absolute value</strong></td>
<td><code>if (x &lt; 0) x = -x;</code></td>
</tr>
</tbody>
</table>
| **put x and y into sorted order**          | `if (x > y)`
|                                            |   
|                                            |   `{ int t = x; x = y; y = 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
The while loop. A common repetition structure.

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

```java
while (boolean expression) {
    statement 1;
    statement 2;
}
```
While Loop: Powers of Two

**Ex.** Print powers of 2 that are \( \leq 2^N \).
- Increment \( i \) from 0 to \( N \).
- Double \( v \) each time.

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

<table>
<thead>
<tr>
<th>( i )</th>
<th>( v )</th>
<th>( i \leq 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>

\( N = 6 \)

**Click for demo**
Powers of Two

```java
public class PowersOfTwo {
    public static void main(String[] args) {
        // last power of two to print
        int N = Integer.parseInt(args[0]);

        int i = 0;  // loop control counter
        int v = 1;  // current power of two
        while (i <= N) {
            System.out.println(i + " " + v);
            i = i + 1;
            v = 2 * v;
        }
    }
}
```

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

% java PowersOfTwo 6
0 1
1 2
2 4
3 8
4 16
5 32
6 64
While Loop Challenge

Q. Anything wrong with the following code for printing powers of 2?

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

Q. Anything wrong with the following code for printing powers of 2?

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

A. Need curly braces around statements in while loop; otherwise it enters an infinite loop, printing "0 1".

Moment of panic. How to stop infinite loop?
While Loops: 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 \( c / t_i \).

\[
\begin{align*}
t_0 &= 2.0 \\
t_1 &= \frac{1}{2}(t_0 + \frac{2}{t_0}) = 1.5 \\
t_2 &= \frac{1}{2}(t_1 + \frac{2}{t_1}) = 1.416666666666665 \\
t_3 &= \frac{1}{2}(t_2 + \frac{2}{t_2}) = 1.4142156862745097 \\
t_4 &= \frac{1}{2}(t_3 + \frac{2}{t_3}) = 1.4142135623746899 \\
t_5 &= \frac{1}{2}(t_4 + \frac{2}{t_4}) = 1.414213562373095
\end{align*}
\]

computing the square root of 2

\[
\sqrt{60481729} \\
7777
\]

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

Copyright 2004, Sidney Harris
www.sciencecartoonsplus.com
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 \( c / t_i \).

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

% java Sqrt 2.0
1.414213562373095

15 decimal digits of accuracy in 5 iterations
Newton-Raphson Method

Square root method explained.
- **Goal:** find root of any function \( f(x) \).
- **Start with estimate** \( t_0 \).
- **Draw line tangent to curve at** \( x = t_i \).
- **Set** \( t_{i+1} \) **to be** \( x \)-coordinate where line hits \( x \)-axis.
- **Repeat until desired precision.**

![Diagram of Newton-Raphson Method]

Technical conditions. \( f(x) \) is smooth; \( t_0 \) is good estimate.
```c
#include <stdio.h>
int main(void)
{
    int count;
    for (count = 1; count <= 500; count++)
        printf("I will not throw paper airplanes in class.\n");
    return 0;
}
```
The **for** loop. Another common repetition structure.

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

```java
for (init; boolean expression; increment) {
    statement 1;
    statement 2;
}
```
Anatomy of a For Loop

Q. What does it print?
A. 

```java
int v = 1;
for (int i = 0; i <= N; i++)
{
    System.out.println(i + " " + v);
    v = 2*v;
}
```
For Loops: Subdivisions of a Ruler

Create subdivision of a ruler.

- Initialize ruler to " ".
- For each value i from 1 to N:
  sandwich two copies of ruler on either side of i.

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

<table>
<thead>
<tr>
<th>i</th>
<th>ruler</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>1</td>
<td>&quot; 1 &quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot; 1 2 1 &quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot; 1 2 1 3 1 2 1 &quot;</td>
</tr>
</tbody>
</table>
For Loops: Subdivisions of a Ruler

```
% java RulerN 1
1

% java RulerN 2
1 2 1

% java RulerN 3
1 2 1 3 1 2 1

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

% java RulerN 5
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 5 1 2 1 3 1 2 1

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

Observation. Loops can produce a huge amount of output!
## Loop Examples

<table>
<thead>
<tr>
<th>Description</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>print largest power of two less than or equal to N</strong></td>
<td>int v = 1;</td>
</tr>
<tr>
<td></td>
<td>while (v &lt;= N/2)</td>
</tr>
<tr>
<td></td>
<td>v = 2*v;</td>
</tr>
<tr>
<td></td>
<td>System.out.println(v);</td>
</tr>
<tr>
<td><strong>compute a finite sum</strong></td>
<td>int sum = 0;</td>
</tr>
<tr>
<td><strong>(1 + 2 + ... + N)</strong></td>
<td>for (int i = 1; i &lt;= N; i++)</td>
</tr>
<tr>
<td></td>
<td>sum += i;</td>
</tr>
<tr>
<td></td>
<td>System.out.println(sum);</td>
</tr>
<tr>
<td><strong>compute a finite product</strong></td>
<td>int product = 1;</td>
</tr>
<tr>
<td><strong>(N! = 1 \times 2 \times ... \times N)</strong></td>
<td>for (int i = 1; i &lt;= N; i++)</td>
</tr>
<tr>
<td></td>
<td>product *= i;</td>
</tr>
<tr>
<td></td>
<td>System.out.println(product);</td>
</tr>
<tr>
<td><strong>print a table of function values</strong></td>
<td>for (int i = 0; i &lt;= N; i++)</td>
</tr>
<tr>
<td></td>
<td>System.out.println(i + &quot; &quot; + 2<em>Math.PI</em>i/N);</td>
</tr>
</tbody>
</table>
Nesting
Nested If Statements

**Ex.** Pay a certain tax rate depending on income level.

<table>
<thead>
<tr>
<th>Income</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 47,450</td>
<td>22%</td>
</tr>
<tr>
<td>47,450 - 114,650</td>
<td>25%</td>
</tr>
<tr>
<td>114,650 - 174,700</td>
<td>28%</td>
</tr>
<tr>
<td>174,700 - 311,950</td>
<td>33%</td>
</tr>
<tr>
<td>311,950 -</td>
<td>35%</td>
</tr>
</tbody>
</table>

graduated income tax calculation

```java
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;
```
Nested If Statements

Use nested if statements to handle multiple alternatives.

```java
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;
        }
    }
}
```
Nested If 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 on p. 75.]
Nested If Statement Challenge

Q. What's wrong with the following for income tax calculation?

<table>
<thead>
<tr>
<th>Income</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 47,450</td>
<td>22%</td>
</tr>
<tr>
<td>47,450 - 114,650</td>
<td>25%</td>
</tr>
<tr>
<td>114,650 - 174,700</td>
<td>28%</td>
</tr>
<tr>
<td>174,700 - 311,950</td>
<td>33%</td>
</tr>
<tr>
<td>311,950 -</td>
<td>35%</td>
</tr>
</tbody>
</table>

```java
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.
public class Gambler {
    public static void main(String[] args) {
        int stake = Integer.parseInt(args[0]);
        int goal = Integer.parseInt(args[1]);
        int T = Integer.parseInt(args[2]);
        int wins = 0;

        // repeat experiment T times
        for (int t = 0; t < T; t++) {
            // do one gambler's ruin experiment
            int cash = stake;
            while (cash > 0 && cash < goal) {
                // flip coin and update
                if (Math.random() < 0.5) cash++;
                else cash--;
            }
            if (cash == goal) wins++;
        }
        System.out.println(wins + " wins of " + T);
    }
}
Fact. Probability of winning = $\frac{\text{stake}}{\text{goal}}$.

Fact. Expected number of bets = $\text{stake} \times \text{desired gain}$.

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 (only) plan of attack.
Control Flow Summary

Control flow.
- Sequence of statements that are actually executed in a program.
- Conditionals and loops: enable us to choreograph the control flow.

<table>
<thead>
<tr>
<th>Control Flow</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>straight-line programs</td>
<td>all statements are executed in the order given</td>
<td></td>
</tr>
<tr>
<td>conditionals</td>
<td>certain statements are executed depending on the values of certain variables</td>
<td>if, if-else</td>
</tr>
<tr>
<td>loops</td>
<td>certain statements are executed repeatedly until certain conditions are met</td>
<td>while, for do-while</td>
</tr>
</tbody>
</table>