Readings
This Week: No new readings. Consolidate!
(Reminder: Readings are absolutely vital for learning this stuff!)

Labs and Tutorials
This Week: Lab #10
Labs are due at end of lab time! (Lab #10 is fairly short, but make sure to finish by the end of lab.)

Final Exam
- Wednesday, April 15, 7pm in SRC A
- This wasn’t a good room last year, but we’re stuck with the date, time, and room UBC assigns. 😞
  - Note: You are allowed to do scratch work on the exam paper! (But write your answer where indicated if the problem tells you to.)

Programming Assignment 3
- Assignment 3 is up on WebCT!
  - Click on the "Assignments" icon.
  - Assigned Sunday evening – sorry for delay!
  - Due at NOON, April 6 (Monday), via electronic hand in.
- Tips:
  - There is some Eclipse setup. Set-up ASAP!
  - Work in pairs. Some conceptual stuff.
  - Think carefully before coding. If concepts right, the coding is much much easier.

Survey #3
- Ben Yu's 3rd survey will be up on WebCT soon.
- This is an end-of-term survey, but we won’t have lectures for me to remind you.
- Keep an eye out on WebCT…
Learning Goals
By the end of class today you will be able to...

- Apply some basic techniques to help get started when facing a new problem.

Problem Solving

- The ability to solve problems is one of the most important skills to acquire. (Not just in CPSC 111!)
- But it’s rarely taught explicitly:
  - Because it requires creative insight, there’s no cookbook recipe that always works.
  - Usual approach is to practice solving problems, until you get better at it.

Problem Solving

- Again, there’s no recipe that always works
- Some basic techniques to get started:
  - Make sure you know what’s given and what kind of result you’re supposed to provide.
  - Try to solve the problem by hand, before programming
    - Work lots of small examples.
    - Think closely about what you did to solve the problem.
  - Think about what kinds of data to store, what kinds of operations need to be done to the data
    - This suggests objects and methods
  - Break things down into steps:
    - A sequence of steps becomes a sequence of statements
    - Look for repetition (loops), decisions (if)
    - Use helper methods to break out common sub-problems.

Example: Sudoku Solution Checker

- Sudoku is a popular logic puzzle:
  - The board is a 9x9 array of squares, with some squares filled in with digits from 1…9
  - The task is to fill in the rest of the squares with digits 1…9, so that:
    - Each row contains the numbers 1…9
    - Each column contains the numbers 1…9
    - Each 3x3 “zone” contains the numbers 1…9
  - Our challenge is to write a program that checks whether a solution is valid.

Example: Sudoku Solution Checker

- Write a method checkSudoku() that takes a 9x9 array of int, and returns true if and only if the array is a legal Sudoku solution.

Example: A correct solution...

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td>7</td>
<td>8</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>8</td>
<td>2</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>4</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>7</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>9</td>
<td>8</td>
<td>2</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>9</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>2</td>
<td>7</td>
<td>9</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>
### Problem Solving

- Again, there's no recipe that always works
- Some basic techniques to get started:
  - Make sure you know what's given and what kind of result you're supposed to provide.
  - Try to solve the problem by hand, before programming
  - Work lots of small examples.
  - Think closely about what you did to solve the problem.
  - Think about what kinds of data to store, what kinds of operations need to be done to the data
  - This suggests objects and methods
  - Break things down into steps:
    - A sequence of steps becomes a sequence of statements
    - Look for repetition (loops), decisions (if)
    - Use helper methods to break out common sub-problems.

---

<table>
<thead>
<tr>
<th>2</th>
<th>7</th>
<th>1</th>
<th>8</th>
<th>9</th>
<th>4</th>
<th>3</th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>8</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>3</td>
<td>9</td>
<td>8</td>
<td>1</td>
<td>7</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>9</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>1</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>7</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>9</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>4</td>
<td>8</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

### Problem Solving

- Again, there's no recipe that always works
- Some basic techniques to get started:
  - Make sure you know what's given and what kind of result you're supposed to provide.
  - Try to solve the problem by hand, before programming
  - Work lots of small examples.
  - Think closely about what you did to solve the problem.
  - Think about what kinds of data to store, what kinds of operations need to be done to the data
  - This suggests objects and methods
  - Break things down into steps:
    - A sequence of steps becomes a sequence of statements
    - Look for repetition (loops), decisions (if)
    - Use helper methods to break out common sub-problems.
Problem Solving

- Again, there's no recipe that always works
- Some basic techniques to get started:
  - Make sure you know what's given and what kind of result you're supposed to provide.
  - Try to solve the problem by hand, before programming
    - Work lots of small examples.
    - Think closely about what you did to solve the problem.
  - Think about what kinds of data to store, what kinds of operations need to be done to the data
    - This suggests objects and methods
  - Break things down into steps:
    - A sequence of steps becomes a sequence of statements
    - Look for repetition (loops), decisions (if)
    - Use helper methods to break out common sub-problems.

A Harder Example: Sudoku Solver

- This is a much harder problem. It's worth thinking about if you want more of a challenge…
- It will be useful to declare a SudokuBoard class:
  - You may want bookkeeping info along with the main grid of numbers.
  - You can then have methods like:
    - applyMove(int row, int col, int digit)
    - getLegalDigits(int row, int col)
    - copyBoard()