CS3: Introduction to Symbolic Programming

Lecture 3: Review
Case Studies

Spring 2006
Nate Titterton
nate@berkeley.edu
Announcements

• Nate's office hours:
  - Wednesday, 1:30-3:30
  - 329 Soda

• Tue/Wed is a Catch-up day.
  - Use this day to catch up! That is, go back over the last two weeks and fill in places you missed
  - You will all be ready to go on Thur/Fri, right?

• We are still waiting on readers for homework grading...
<table>
<thead>
<tr>
<th>Week</th>
<th>Dates</th>
<th>Lecture: &lt;holiday&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Lab: Conditionals, Booleans, Testing</td>
</tr>
<tr>
<td>2</td>
<td>Sep 4-8</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Sep 11-15</td>
<td>Lecture: Case Studies</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reading: Difference between Dates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(just the first version in the reader)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lab: Work with Difference between Dates</td>
</tr>
<tr>
<td>4</td>
<td>Sep 18-22</td>
<td>Lecture: Data abstraction in DbD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lab: Miniproject I</td>
</tr>
<tr>
<td>5</td>
<td>Sep 25-29</td>
<td>Lecture: Introduction to Recursion</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lab: Recursion</td>
</tr>
<tr>
<td>6</td>
<td>Oct 2-6</td>
<td>Lecture: Midterm 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lab: Recursion II</td>
</tr>
</tbody>
</table>
Review

What is Scheme?

- A easy yet powerful programming language
- The "Listener" makes testing easy
- Unique features like "quoting"

• Words and sentences
  - Not usually part of scheme, but makes our early work more accessible

• Quoting something means treating it literally:
  - you are interested in the name that follows, rather than what is named
  - Quoting is a shortcut to putting literal things right in your code. As your programs get bigger, you will do this less and less.
Some terminology

- **Conditional**
  - `cond` and `if`

- **Booleans**
  - `#t` and `#f`
  - In practice, everything is true except `#f`

- **Predicates**
  - Procedures that return `#t` or `#f`
  - By convention, their names end with a "?"

  (odd? 5) ➞ #t

  (member? 'x '(a e i o u)) ➞ #f
• There is much more to programming than writing code
  - *Testing* is crucial, and an emphasis of this course
  - Analysis
  - Debugging
  - Maintenance.
  - "Design"
Some nice comments

• "In English, when something is in quotes we think about it differently. Same in scheme"

• "In order to remember how to parenthesize a cond statement... think of each statement as an if without the 'if' "

(actually, in lecture I mentioned that these quotes came from you guys, but I was wrong: these came from an earlier semester. Still, your quotes were just as good, I just used the wrong slide...)
A video resource

- http://wla.berkeley.edu
  Weiner lecture archives

- The "course" is an earlier CS3
  - Different emphasis; early lectures may work better than later ones
  - Very different lab experience
  - Same book
Write an answer procedure.

Write a procedure named answer that, given a sentence that represents a question, returns a simple answer to that question. (A question's last word ends with a question mark.) If the argument sentence is not a question, answer should merely return the argument unchanged.

- Given (am i ...? ), answer should return (you are ...).
- Given (are you ...? ), answer should return (i am ...).
- Given (some-other-word i ... ?), answer should return (you some-other-word ...).
- Given (some-other-word you ... ?), answer should return (i some-other-word ...).
- Given any other question, answer should return the result of replacing the question mark by a period.
You are writing big programs now. But, what can’t you do yet?
What does “understand a program” mean?
A big idea

• Data abstraction

  - **Constructors**: procedures to make a piece of data
    - word and sentence

  - **Selectors**: procedures to return parts of that data piece
    - first, butfirst, etc.
Case Studies

• Reading!?

• A case study:
  - starts with a problem statement
  - ends with a solution
  - in between, a …story… (narrative)
  - How a program comes to be

• You will write “day-span”, which calculates the number of days between two dates in a year
You need to read this

- The lab will cover the case study through a variety of activities.
  - This will culminate in the first “mini-project”

- We just may base exam questions on it

- It will make you a better programmer! 4 out of 5 educational researchers say so.
Some important points

• There is a large "dead-end" in this text
  - Like occur in many programming projects
  - Good "style" helps minimize the impacts of these

• There is (often) a difference between good algorithms and between human thinking
Extra Materials
(define (walk light city cops-present)
  (cond ((equal? city 'berkeley) 'strut)
    ((equal? light 'green) 'go)
    ((equal? light 'not-working) 'go-if-clear)
    ((and (equal? light 'flashing-red) cops-present) 'wait)
    ((equal? light 'flashing-red) 'hurry)
    (else 'just-stand-there)))