CS3: Introduction to Symbolic Programming

Lecture 5: Recursion
Midterm-like problems

Fall 2006 Nate Titterton
nate@berkeley.edu
<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Lecture: Data abstraction in DbD</th>
<th>Lab: Miniproject I</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Sept 18-22</td>
<td>Lecture: Introduction to Recursion</td>
<td>Lab: Recursion</td>
</tr>
<tr>
<td>5</td>
<td>Sept 25-29</td>
<td>Lecture: Advanced Recursion</td>
<td>Lab: Recursion II</td>
</tr>
<tr>
<td>6</td>
<td>Oct 2-6</td>
<td>Lecture: Midterm 1</td>
<td>Lab: Recursion II</td>
</tr>
<tr>
<td>7</td>
<td>Oct 9-13</td>
<td>Lecture: Advanced Recursion</td>
<td>Lab: Recursion III</td>
</tr>
<tr>
<td>8</td>
<td>Oct 16-20</td>
<td>Lecture: finishing recursion</td>
<td>Lab: Miniproject #2</td>
</tr>
</tbody>
</table>
Announcements

• Nate's office hours:
  - Wednesday, 1:30-3:30, in 329 Soda
  - Special: Monday Oct 2nd, 1-3pm, in 329 Soda

• Midterm next week!
  - (More on this in a bit)

• Reading for this week
  - Simply Scheme, chapter 11
  - Difference between Dates, Recursive version
  - (These will be on the midterm)

• Still having trouble working at home?
  - Go to 333 Soda hall!
Drop day

• The last day to drop is Sept 29
  - I think
Recursion

• Everyone thinks it's hard!
  - (well, it is... aha!-hard, not complicated-hard)

• The first technique (in this class) to handle arbitrary length inputs.
  - There are other techniques, easier for some problems.

• What is it?
An algorithmic technique where a function, in order to accomplish a task, calls itself with some part of the task.
All recursion procedures need…

1. Base Case (s)
   • Where the problem is simple enough to be solved directly

2. Recursive Cases (s)
   1. Divide the Problem
      • into one or more smaller problems
   2. Invoke the function
      • Have it call itself recursively on each smaller part
   3. Combine the solutions
      • Combine each subpart into a solution for the whole
Problem: find the first even number in a sentence of numbers

(define (find-first-even sent)
  (if <test? (first sent))
    (do the base case>) base case: return
    ; that even number
    (do the recursive case>)
    ; recurse on the
    ; rest of sent
  ))
(define (count sent)

  (if (empty? (bf sent))

    1 ;base case: return 1

    (+ 1
        (count (bf sent)) ;recurse on the ; rest of sent

    ))
)
Base cases can be tricky

- By checking whether the (\textbf{bf \textit{sent}}) is empty, rather than \textit{sent}, we won't choose the recursive case correctly on that last element!
  - Or, we need two base cases, one each for the last element being odd or even.

- Better: let the recursive cases handle \textit{all} the elements

\textit{Your book describes this well}
Count the number of words in a sentence

(define (count sent)
  (if (empty? (bf sent))
      0 ;base case: return 0
      (+ 1
        (count (bf sent)) ;recurse on the ; rest of sent))
)
> (count '(a b c))

\[\text{sent} = (a b c)\]

\[\Rightarrow (\text{+ 1} (\text{+ 1} (\text{+ 1 0})))\]

\[\Rightarrow 3\]
Problem: *find all the even numbers in a sentence of numbers*

```
(define (find-evens sent)
  (cond ((empty? sent) ; base case
        '())
        ((odd? (first sent)) ; rec case 1: odd
         (find-evens (bf sent)))
        (else ; rec case 2: even
         (se (first sent)
             (find-evens (bf sent)))
        )))
```
> (find-evens '(2 3 4 5 6))

\[ (\text{se} \ 2) \rightarrow (\text{se} \ 4) \rightarrow (\text{se} \ 6) \]

\[ \text{sent} = ( 2 3 4 5 6 ) \]

\[ \text{sent} = ( 3 4 5 6 ) \]

\[ \text{sent} = ( 4 5 6 ) \]

\[ \text{sent} = ( 5 6 ) \]

\[ \text{sent} = ( 6 ) \]

\[ \text{sent} = () \]

\[ \rightarrow (\text{se} \ 2 \ (\text{se} \ 4 \ (\text{se} \ 6 \ (()))) \)) \]

\[ \rightarrow (2 \ 4 \ 6) \]
Why is recursion hard?

• ONE function:
  - replicates itself,
  - knows how to stop,
  - knows how to combine the “replications”

• There are many ways to think about recursion: you absolutely do not need to understand all of them.
  - Knowing recursion WILL help with all sorts of ways while programming, even if you don’t often use it.
Midterm 1: Oct 2\textsuperscript{nd} (next week)

- Location: 50 Birge
- Time: In the lecture slot, plus 20 minutes
  - (4:10-5:30)

- Open book, open notes.
  - Nothing that can compute, though

- Everything we’ve covered, including the coming week on recursion.

- Review session this Saturday, Sep 30th, 2-4pm.
  - 430 Soda (Wozniak lounge).

- Practice exam in reader (do this all at once)
- Check Announcements for more practice items, solutions
Special midterm issues

- Several of you have come to me with midterm conflicts

  - You need to email me (and get a response) or talk to me so I can get a count!
Consider a procedure named `double` that, given a word as argument, returns a two-word sentence. The first word is two. The second word is the result of adding an "s" to the end of the argument.

<table>
<thead>
<tr>
<th>expression</th>
<th>intended result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(double 'apple)</td>
<td>(two apples)</td>
</tr>
<tr>
<td>(double 'bus)</td>
<td>(two buss)</td>
</tr>
<tr>
<td>(double 'box)</td>
<td>(two boxes)</td>
</tr>
</tbody>
</table>
Now consider some \textit{incorrect} implementations of \texttt{double}. For each one, indicate what the call \texttt{(double 'apple)} will return. If no value is returned because the procedure crashes, give the error message that results.

\begin{verbatim}
(define (double wd)
  (sentence 'two '(word wd s)))

(define (double wd)
  (sentence 'two (sentence wd s))

(define (double wd)
  (sentence 'two (wd 's)))
\end{verbatim}
between?

Write a procedure called between? which takes three numbers as arguments, and returns true if and only if the second argument is between and not equal to the first and the third:

(between? 5 6 7) -> #t
(between? 7 6 5) -> #t

Part A: Write between? without using if or cond.

Part B: Write between? without using and or or.

Part C: Write a suite of test cases for between?. Make sure you test the possible sets of parameters exhaustively as possible, in order to test different ways the code could be written.

Also, make sure you describe what the result of the call should be!