CS3: Introduction to Symbolic Programming

Lecture 7: Advanced Recursion

Fall 2006 Nate Titterton
nate@berkeley.edu
<table>
<thead>
<tr>
<th>Week</th>
<th>Dates</th>
<th>Topics</th>
</tr>
</thead>
</table>
| 6    | Oct 2-6     | Lecture: *Midterm 1*  
Lab: Recursion II |
| 7    | Oct 9-13    | Advanced recursion                         |
| 8    | Oct 16-20   | Finishing recursion  
Miniproject #2: Number names |
| 9    | Oct 23-27   | Introduction to Higher Order Procedures    |
| 10   | Oct 30 -Nov 3 | More HOF                                   |
| 11   | Nov 6-10    | Finish HOF  
Miniproject #3: Election processing
Midterm 1

- You did quite well (IMO)
- Solutions will be available soon on the portal (check announcements)
Question 1: fill in the blanks

Std. Dev = 1.84
Mean = 7.1
N = 133.00
Q2: Writing stressed?, within-10?

P2A

Std. Dev = 1.31
Mean = 3.9
N = 133.00

P2B

Std. Dev = 1.04
Mean = 3.4
N = 133.00
Q3: tuesday-span

- While we were generous, most of you got the basic idea
Q4: translating a sentence

Std. Dev = 2.85
Mean = 7.2
N = 133.00
Q5: Data abstraction with tutors

- **Std. Dev = 2.90**
- **Mean = 5.7**
- **N = 133.00**

- **Std. Dev = 1.35**
- **Mean = 2.9**
- **N = 133.00**
Problem: *find all the even numbers in sentence of numbers*

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

(sent = (2 3 4 5 6))

(se 2

(sent = (3 4 5 6))

(se 4

(sent = (4 5 6))

(se 6

(sent = (6))

() )

(2 4 6)

⇒ (se 2 (se 4 (se 6 ())))

⇒ (se 2 (se 4 (se 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.
  - "down-up": recursion as an extension of writing many specific functions
  - "many base cases": recursion as using a clone, once you have many base cases
Patterns in basic recursion

• Mapping
  - does something to every part of the input sentence
  - E.g., square-all

• Counting
  - Counts the number of elements that satisfy a predicate
  - E.g., count-vowels, count-evens

• Finding
  - Return the first element that satisfies predicate (or, return rest of sentence)
  - E.g., member, member-even
• Filtering
  - Keep or discard elements of input sentence
  - E.g., keep-evens

• Testing
  - A predicate that checks that every or any element of input satisfies a test
  - E.g., all-even?

• Combining
  - Combines the elements in some way…
  - E.g., sentence-sum
What recursions aren’t covered by these patterns?

• **Weird ones like reverse, or downup**
  - ... bowling ...

• "Advanced" recursions:
  - *when it does more than one thing at a time*
  - Ones that don’t traverse a single sentence
    - E.g., mad-libs takes a sentence of replacement words [e.g., `
      (fat Henry three)`] and a sentence to mutate [e.g.,
      `(I saw a * horse named * with * legs)`]
  - Tree recursion: multiple recursive calls in a single
    recursive step
"when it does more than one thing at a time"

- Ones that traverse multiple sentences
  - E.g., mad-libs takes a sentence of replacement words [e.g., `(fat Henry three)`] and a sentence to mutate [e.g.,
    `'(I saw a * horse named * with * legs)`]
Advanced recursions (2/3)

- Recursions that have an *inner* and an *outer* recursion

\[
\text{(no-vowels '}(I\ \text{like\ to\ type})\Rightarrow\text{"'' lk\ t\ typ)}
\]

\[
\text{(l33t '}(I\ \text{like\ to\ type})\Rightarrow\text{(i\ 1i/<3\ +0\ +yP3)}
\]

\[
\text{(strip-most-popular-letter '}(cs3\ \text{is\ the\ best\ class})\Rightarrow\text{(c3\ i\ the\ bet\ cla))}
\]

\[
\text{(occurs-in? 'abc\ 'abxcde)}\Rightarrow\text{#f}
\]
• Tree recursion: multiple recursive calls in a single recursive step

• There are many, many others
The fibonacci sequence:

\[1 \ 1 \ 2 \ 3 \ 5 \ 8 \ 13 \ 21 \ 34 \ 55\]

(\text{define} (\text{fib} \ n)
  (if (\leq n 2)
    1 ;; base case
    (+ (\text{fib} (- n 1)) ;; recursive case
       (\text{fib} (- n 2))))))}
Write the procedure sub-sentence, which returns a middle section of a sentence. It takes three parameters; the first identifies the index to start the middle section, and will be 1 or greater; the second identifies the length of the middle section, and will be 0 or greater; and the last is the sentence to work with.

Do not use any helper procedures.
Do not use the item procedure in your solution.

(sub-sentence 2 3 '(a b c d e f g)) \rightarrow (b c d)
(sub-sentence 3 2 '(a b)) \rightarrow ()
(sub-sentence 3 0 '(a b c d e)) \rightarrow ()
(sub-sentence 3 9 '(a b c d e)) \rightarrow (c d e)
(define (sub-sentence start len sent)
  (cond ((empty? sent)
    '())
  ((> start 1)
    (sub-sentence (- start 1) len (bf sent)))
  ((> len 0)
    (se (first sent)
      (sub-sentence start (- len 1)(bf sent))))
  (else
    '()))
))