CS3: Introduction to Symbolic Programming

Lecture 6: Finishing up basic recursion

Spring 2006

Nate Titterton
nate@berkeley.edu
<table>
<thead>
<tr>
<th>Week</th>
<th>Dates</th>
<th>Lecture: <strong>Midterm 1</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Feb 27-Mar 3</td>
<td>Lab: Recursion III</td>
</tr>
<tr>
<td>8</td>
<td>Mar 6-10</td>
<td>Lecture: Finishing recursion</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lab: Miniproject #2: Number names</td>
</tr>
<tr>
<td>9</td>
<td>Mar 13-17</td>
<td>Introduction to Higher Order Procedures</td>
</tr>
<tr>
<td>10</td>
<td>Mar 20-24</td>
<td>More HOF</td>
</tr>
<tr>
<td>11</td>
<td>Mar 27-31</td>
<td><em>(Spring Break)</em></td>
</tr>
</tbody>
</table>
• You did quite well (IMO)
• If you need a re-grade, talk to your TA first, and then see the TA that graded that question
• Solutions will be available soon on the portal (check announcements).
Question 1: fill in the blanks

M1_1

Std. Dev = 1.61
Mean = 5.0
N = 67.00
Q2: Roman numerals, and recursion

• You did quite well on the recursion question
  - (The TAs and I were worried about this one)

• Take home point (which you've got): A recursive solution will have a series of conditional cases, some recursive, some non-recursive

![Histogram with labels: Standard Deviation = 3.00, Mean = 13.2, N = 67.00]
Q3: between?, and test cases

- Writing test cases is an "art", especially when there are no clear cut 'cond' clauses
  - Test ALL possible outputs
  - Test all conditions that lead to a particular decision
• This wasn't a great question in how it was "asked"
Q5: sub-sentence using recursion

- Many of you ran out of time
- Some nice solutions here (that I hadn't thought of)
A solution for sub-sentence

(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
        '()))
)
Number Spelling Miniproject

• Read *Simply Scheme*, page 233, which has hints

• Another hint (principle): don't force "everything" into the recursion.
  - Special/border cases may be easier to handle before you send yourself into a recursion
"Tail" recursions

• Accumulating recursions are sometimes called "tail" recursions (by TAs, me, etc).
  - But, not all recursions that keep track of a number are "tail" recursions.

• A tail recursion has no combiner, so it can end as soon as a base case is reached
  - Compilers can do this efficiently

• An embedded recursion needs to combine up all the recursive steps to form the answer
  - The poor compiler has to remember everything
(define (length sent)
  (if (empty? sent)
      0
      (+ 1 (length (bf sent)))))

Tail or embedded? (1/3)
(length '(a b c d)) ➞
(+ 1 (length '(b c d)))
(+ 1 (+ 1 (length '(c d))))
(+ 1 (+ 1 (+ 1 (length '(d))))))
(+ 1 (+ 1 (+ 1 (+ 1 (length '())))))
(+ 1 (+ 1 (+ 1 (+ 1 0))))
(+ 1 (+ 1 (+ 1 1)))
(+ 1 (+ 1 2))
(+ 1 3)
4
(define (sent-max sent)
  (if (empty? sent)
    ()
    (sent-max-helper (bf sent) (first sent))))

(define (sent-max-helper sent max-so-far)
  (if (empty? sent)
    max-so-far
    (sent-max-helper (bf sent)
      (if (> max-so-far (first sent))
        max-so-far
        (first sent))))))
(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))

\[
\begin{align*}
\text{sent} &= (2 3 4 5 6) \\
(\text{se} &\ 2) \\
(\text{se} &\ 4) \\
(\text{se} &\ 6) \\
(\text{se} &\ 2 (\text{se} &\ 4 (\text{se} \ 6 ())) \\
\Rightarrow &\ (2 \ 4 \ 6) \\
\end{align*}
\]
(define (sub-sentence start len sent)
  (cond ((empty? sent)
           '())
        ((> start 1)
          (sub-sentence (- start 1) len (bf sent)))
        ((> len (count sent))
          (sub-sentence start (- len 1) (bf sent)))
        (else
          '())))
"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)`]
• Recursions that have an *inner* and an *outer* recursion

(no-vowels '(I like to type)) \(\rightarrow\) ("" lk t typ)

(133t '(I like to type)) \(\rightarrow\) (i 1i/<3 +0 +yP3)

(strip-most-popular-letter '(cs3 is the best class)) \(\rightarrow\)
  (c3 i the bet cla)

(occurs-in? 'abc 'abxcde) \(\rightarrow\) #f
Advanced recursions (3/3)

• Tree recursion: multiple recursive calls in a single recursive step

• There are many, many others
Tree recursion: fibonacci

• The fibonacci sequence:
  1   1   2   3   5   8   13   21   34   55

(define (fib n)
  (if (<= n 2)
      1                     ;; base case
      (+ (fib (- n 1))     ;; recursive case
          (fib (- n 2)))))
pair-all

- Write *pair-all*, which takes a sentence of *prefixes* and a sentence of *suffixes* and returns a sentence pairing all *prefixes* to all *suffixes*.

- `(pair-all '(a b c) '(1 2 3))` ➞ `(a1 b1 c1 a2 b2 c2 a3 b3 c3)`

- `(pair-all '(spr s k) '(ite at ing ong))` ➞ `(sprite sprat spring sprong site sat sing song kite kat king kong)`
Write binary, a procedure to generate the possible binary numbers given \( n \) bits.

(binary 1) \( \rightarrow \) (0 1)

(binary 2) \( \rightarrow \) (00 01 10 11)

(binary 3) \( \rightarrow \) (000 001 010 011 100 101 110 111)