CS3:
Introduction to Symbolic Programming

(Special) Lecture 6:
More Recursion
Midterm problems and review

Spring 2007

Nate Titterton
nate@berkeley.edu

Schedule

5	Feb 12-16	Lecture: Introduction to Recursion Lab: Recursion
6	Feb 19-23	Lecture: *Special Lecture and Review* Reading: Simply Scheme, ch. 12, 13 "Roman Numerals" case study Lab: Work with Roman Numerals Two-argument recursion Tracing procedures
7	Feb 26 - Mar 2	Lecture: *Midterm 1* Lab: Advanced recursion
8	Mar 5 – 9	Lecture: Recursion Lab: Miniproject #2

Announcements

- **Nate's office hours:**
  - **Wednesday, 2-4, in 329 Soda**

- **TA review session**
  - **Saturday, Feb 24, 2-4pm, in 430 Soda**
  - **Send questions to us before hand!**

Midterm 1: Feb 26th (next week)

- Location: 160 Kroeber (same place)
- Time: In the lecture slot, plus 20 minutes
  - (5:10-6:30)
- Open book, open notes.
  - Nothing that can compute, though
- Everything we've covered, including the coming TWO weeks on recursion.
  - (But not the "roman numerals" case study)
- TA-led review session
  - Sat, Feb 24, 2-4pm, 430 Soda (Wozniak Lounge)
- Practice exams in your reader, solutions to be announced on Course Portal

Lab materials (last week)

- "combining method" with
  - `downup`,
  - `reverse`,
  - `copies`,
  - `sum-in-interval`,
  - `appearances`
- Data abstraction with `celebrity`
- The replacement modeler
- Work with recursive `day-span`
- Write
  - `down-to-0`
  - `remove`
  - `all-odd?`
  - `dupls-removed`
  - `is-sorted?`

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

Locate the "parts"

```
(define (find-evens sent)
  (cond ((empty? sent)
         '() )
        ((odd? (first sent))
         (find-evens (bf sent)) )
        (else
         (se (first sent)
             (find-evens (bf sent))) )
        ))
```

Base Case

Invoke the function recursively

Divide the problem

Combine the solutions

fa05 Midterm #1, 3c (1/6)

The following are buggy versions of the recursive procedure day-sum, defined in the cases study *Difference between dates, part II*. (The code for the case study is included as an appendix). The bugs result from small changes which are underlined.

For each version, note whether the bug creates a problem in the

a) conditional tests,
b) the base case return value,
c) making the problem smaller,
d) calling the function recursively, or
e) combining the recursive calls.

Also briefly describe in English the effect of the bug on the operation of day-span as a whole (*not just on day-sum)*—this should take between 1 and 2 sentences for each case. You might include an example call to day-span illustrating the problem, although this isn't necessary with a sufficient explanation (and, might be wrong!).

fa05 Midterm #1, 3c (2/6)

The real code

```
(define (general-day-span earlier-date later-date)
  (+ (days-remaining earlier-date)
     (day-sum
       (next-month-number earlier-date)
       (prev-month-number later-date) )
     (date-in-month later-date) ) )


(define (day-sum first-month last-month)
  (if (> first-month last-month)
      0
      (+  (days-in-month (name-of first-month))
          (day-sum (+ first-month 1) last-month))
      ) )
```

fa05 Midterm #1, 3c (3/6)

version 1

```
(define (day-sum first-month last-month)
  (if ( >=  first-month last-month)
      0
      (+ (days-in-month (name-of first-month))
         (day-sum (+ first-month 1) last-month))
      ) )
```

fa05 Midterm #1, 3c (4/6)

version 2

```
(define (day-sum first-month last-month)
  (if ( > first-month last-month)
      0
      (+ (days-in-month (name-of first-month))
         (day-sum first-month (+ last-month 1)))
      ) )
```

fa05 Midterm #1, 3c (6/6)

version 3

```
(define (day-sum first-month last-month)
  (if ( <  first-month last-month)
      0
      (+ (days-in-month (name-of first-month))
         (day-sum (+ first-month 1) last-month))
      ) )
```

fa05 Midterm #1, 3c (6/6)

version 4

```
(define (day-sum first-month last-month)
  (if ( > first-month last-month)
      1
      (+ (days-in-month (name-of first-month))
         (day-sum (+ first-month 1) last-month))
      ) )
```

Multiple ways of "approaching" recursion

1. **The combining method**
   - Write versions for specific "sizes" of arguments, and then generalize the pattern
2. **Write many base cases**
   - Solve all the cases you can directly, then use a clone
3. **Use the substitution model**
   - Expand the recursive calls to see a single, large expression
4. **Leap of faith**
   - Assume the procedure already works while you are writing it, then come back to base cases
5. **Tracing**
   - A way to see each internal recursive call (i.e., arguments and return values).
6. **Patterns and Templates**
   - Recursions fall into patterns … after the midterm!

Midterm Problem: what comes between? (1/3)

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`.**

Midterm Problem: what comes between? (2/3)

***Part B*:**

Write `between?` without using `and` or `or`.

Midterm Problem: what comes between? (3/3)

***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 descive what the result of the call should be!

Midterm Problem: sub-cursion?

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)) ➔ (b c d)
(sub-sentence 3 2 '(a b))          ➔ ()
(sub-sentence 3 0 '(a b c d e)     ➔ ()
(sub-sentence 3 9 '(a b c d e)     ➔ (c d e)
```