CS3:
Introduction to Symbolic Programming

(Special) Lecture 5:
More Recursion
Midterm problems and review

Spring 2008

Nate Titterton
nate@berkeley.edu

Schedule

5	Feb 18-22	Lecture: *Holiday, no lecture* Reading: "DbD" case study, recursive version Simply Scheme, Chapter 12 (for Tue/Wed) "Roman Numerals" case study (for Thur/Fri) Lab: More complex recursion
6	Feb 25-29	Lecture: *Midterm #1* Lab: Recursion with multiple arguments Reading: Simply Scheme ch. 14 Homework: The "big" homeworks…
7	Mar 3-7	Lecture: Advanced Recursion Lab: Advanced Recursion Miniproject #2: Number Spelling

Midterm #1

- **Midterm 1**
  - 90 minutes long (4:10-5:40), 2050 VLSB
    - (Some of you will be taking it at 3pm in 606 Soda, IF you have emailed me beforehand).
  - Open book, open notes.
    - Nothing that can compute, though.
  - Everything we've done in class
    - Including the coming lab on the "Roman Numerals" case study.
  - Practice exams in your reader
    - Do these all at once (to simulate an exam)
    - Solutions will be announced
  - TA review session
    - Saturday, Feb 23, 2-4pm, in 306 soda
    - Send questions to us before hand!

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))) )
        ))
```

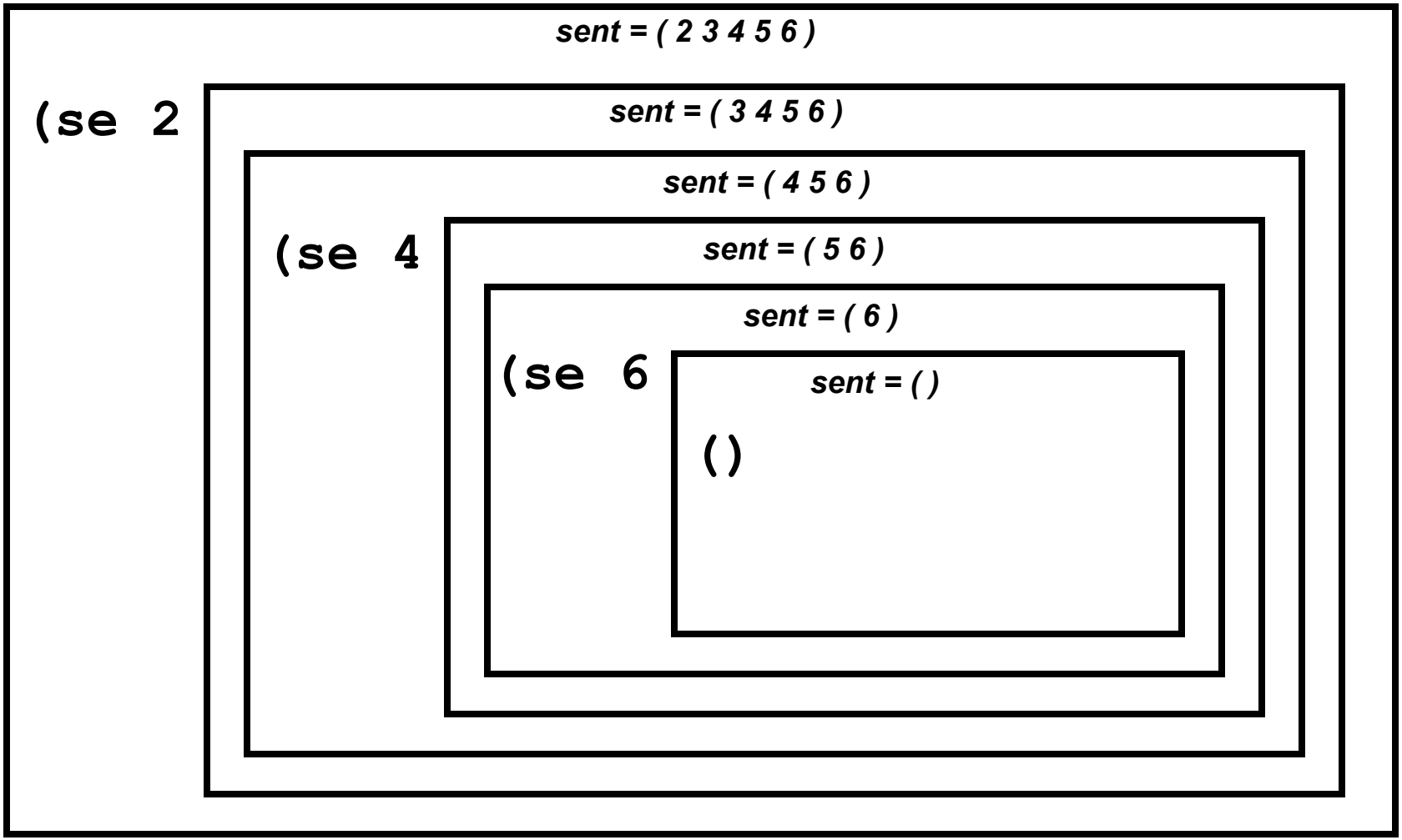
Base Case

Invoke the function recursively

Divide the problem

Combine the solutions

```
> (find-evens '(2 3 4 5 6))
```



➔ `(se 2 (se 4 (se 6 ())))`

➔ `(2 4 6)`

Recursion Lab materials

- "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?`

Some midterm like problems

Write sevens

- **Write a procedure `sevens` that takes a sentence of numbers, and replaces any pairs of numbers that sum to seven with the number 7.**

```
> (sevens '(2 3 4 5 6)) ➔ (2 7 5 6)

> (sevens '(3 4 3 2 5)) ➔ (7 3 7)

> (sevens '(6 1 0 2 7 0 4)) ➔
     (7 0 2 7 4)
```

Whatever floats your boat (sp07 mt1) (1/3)

This problem involves a procedure `can-order?`, which takes two ranks in the United States navy and returns `#t` if and only if the first rank is "above" the second and can, therefore, order the other one around. The following table lists the ranks:

Rank	Explanation
5	***5 star admiral***
3	***3 star admiral***
1	***1 star admiral***
cpn	***captain***
cmd	***commander***
ltn	***lieutenant***
en	***ensign***

Whatever floats your boat (sp07 mt1): part A

Write `can-order?` in the form of the "better solution" in the *Difference Between Dates* case study (the second attempt that successfully wrote day-span, after the dead end was reached in the first attempt). You can assume that the ranks passed to `can-order?` are valid.

<u>Chooose good names for your parameters and helper procedures, and add relevant comments above every procedure</u>.

- **Partial credit will be awarded for solutions that don't follow the form of the better solution in *Difference Between Dates*.**

Whatever floats your boat (sp07 mt1): part B

There are many possible valid calls to `can-order?`. For this problem, you will write test cases for the procedure.

- **We don't want a large list. Instead, we want you to describe what the general classes of tests cases are.**
- **That is, think about how the test cases can be grouped, such that the cases in a group are similar in how they check for errors or otherwise test the program.**
- **There are not many groups.**

For each group, briefly describe what the similarity is and provide a single test case. Be sure to include the correct result of the test.

`day-by-day` (1/2)

- **Fill in the blanks on `day-by-day` below, a recursive function which should provide the same responses as `day-span`.**
- **Use all of the framework code provided below (don't simply cross some out and ignore it).**
- **Be sure to use helper procedures where appropriate, and good procedure and parameter names – `cond` statements with 12 cases will be frowned upon. (Feel free to use any procedure defined in the recursive version of `day-span`).**

day-by-day (2/2)

```
;; works as day-span
(define (day-by-day date1 date2)
  (if (equal? date1 date2)
      ____________________
      ( + 1 (day-by-day (next-day date1) date2))) ))


(define (next-day date)
   (if (last-day-of-month? date)
       ____________________________________
       ____________________________________  ))


(define (last-day-of-month? date)
```

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)
```