CS3:
Introduction to Symbolic Programming

Lecture 5:
More Recursion
Midterm 1 review

Fall 2007

Nate Titterton
nate@berkeley.edu

Schedule

4	Sep 17-21	Lecture: Data abstraction in DbD; Introduction to recursion Lab: Miniproject I; begin recursion
5	Sep 24-28	Lecture: Recursion Lab: More complex recursion Reading: "Dbd, recursive solution" case study (for Tue/Wed) "Roman Numerals" case study (for Thur/Fri)
6	Oct 1-4	Lecture: *Midterm 1* Lab: Recursion with multiple arguments Reading: Simply Scheme ch. 14
7	Oct 8-12	Lecture: Advanced Recursion Lab: Advanced Recursion Miniproject 2: Number spelling

Announcements

- **Nate's office hours (permanently):**
  - Wed, 2-4, 329 Soda
- **Reading for this week**
  - "Difference between dates recursive version", case study in the reader. For lab Tue/Wed
  - "Roman Numerals" case study in reader. For lab Thur/Fri
- **The last day to drop is Sept 28th**

- **Midterm is 90 minutes (4:10 – 5:40).**
  - Room TBA
  - TA-led review session: Saturday, 3pm-5pm

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

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

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

Divide the problem

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

Divide the problem

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

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

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

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

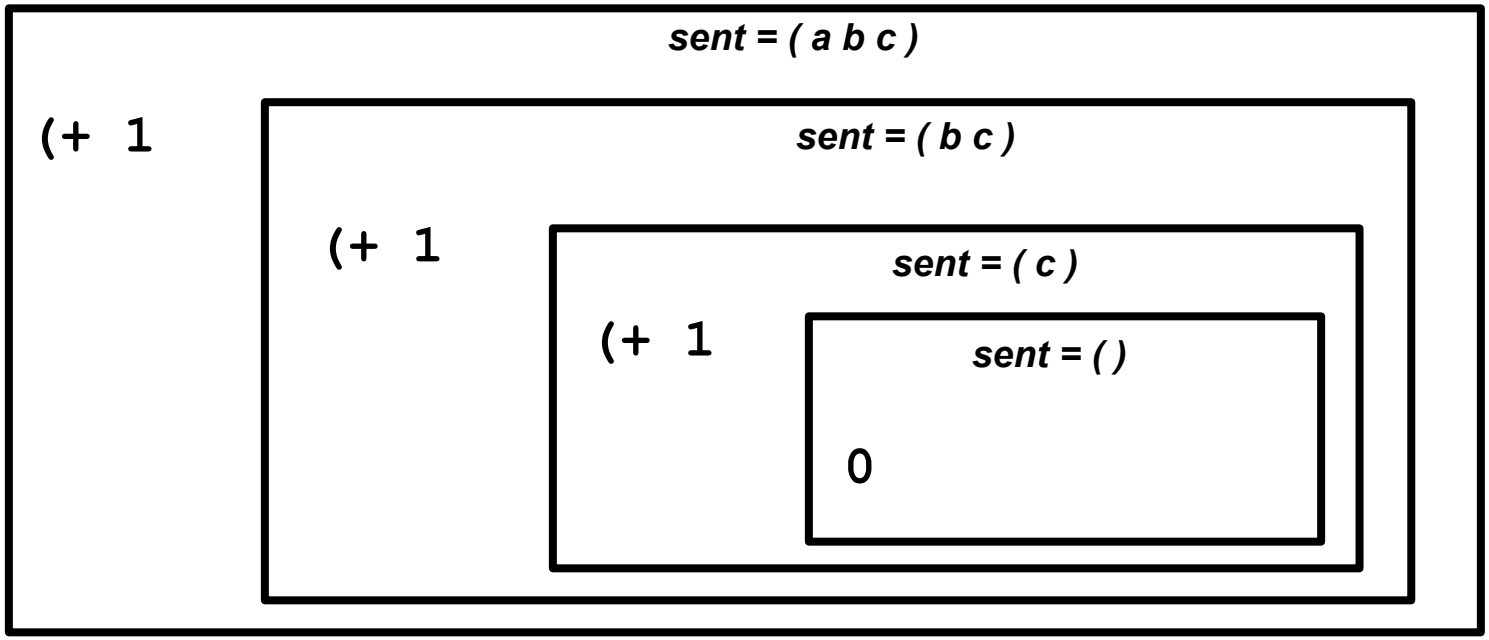
Combine the solutions

Another way to represent recursion

```
(define (count sent)
  (if (empty? sent)
      0
      (+ 1 (count (bf sent)))
      ))
```

```
(define (count sent)
  (if (empty? sent)
      0
      (+ 1 (count (bf sent)))
      ))
```

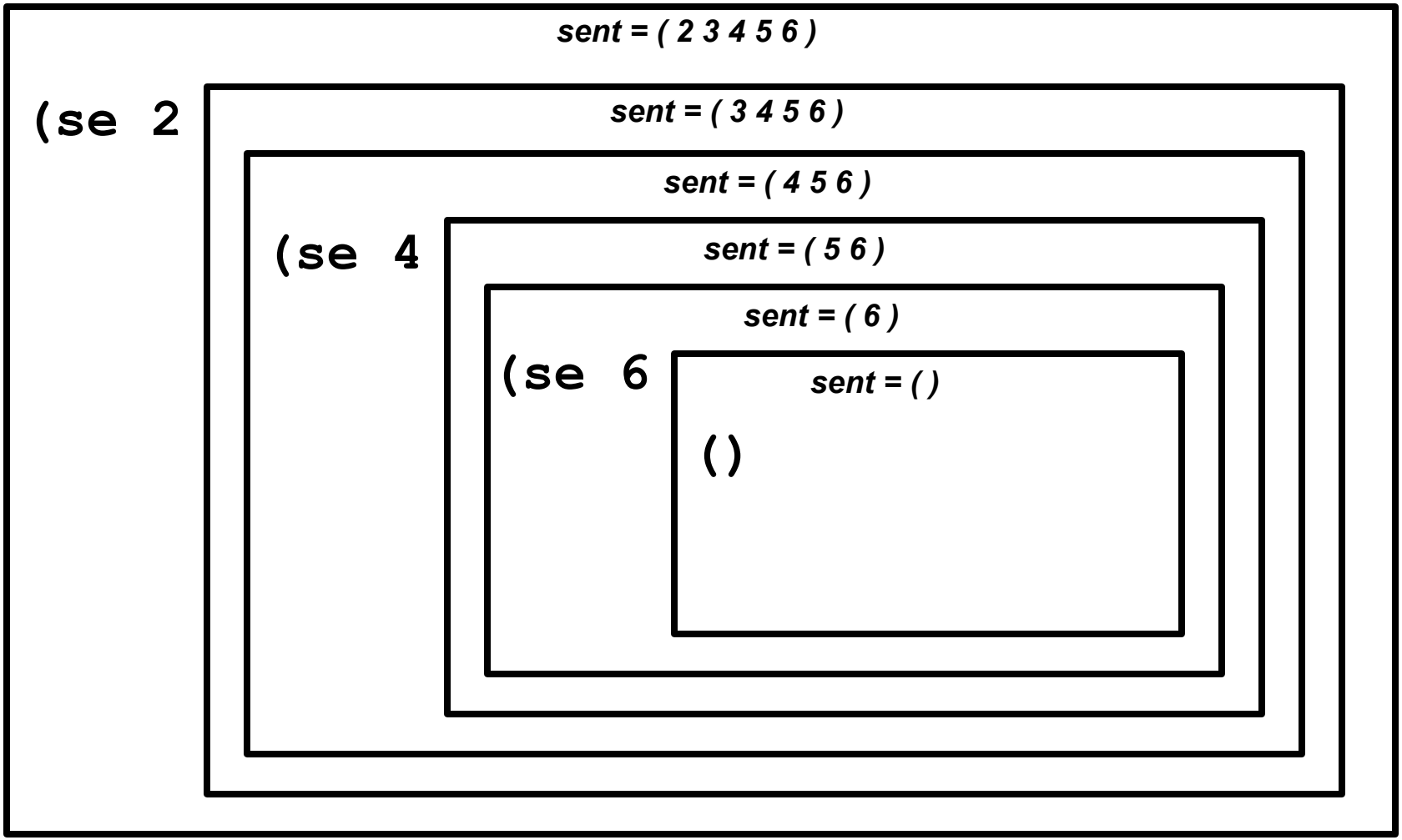
```
> (my-count '(a b c))
```



➔ `(+ 1 (+ 1 (+ 1 0)))`

➔ `3`

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



➔ (se 2 (se 4 (se 6 ())))

➔ (2 4 6)

Lab materials (last week)

- "combining method" with
  - `downup`,
  - `reverse`,
  - `copies`,
  - `sum-in-interval`,
  - `appearances`

Lab material (this week)

- Data abstraction and recursion
- The replacement modeler
- Work with recursive `day-span`
- Write
  - `down-to-0`
  - `remove`
  - `all-odd?`
  - `dupls-removed`
  - `is-sorted?`
- Work with “roman numerals”
  - `grouped`

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)
```

Midterm 1: Oct 1st (next week)

- Location: TBA
- Time: In the lecture slot, plus 40 minutes
  - (4:10-5:40)
  - If you have a conflict, you need to TALK TO ME.
- Open book, open notes. Really.
  - Nothing that can compute, though
- Everything we've covered, including this coming week on recursion.
  - Yes, this include "roman numerals" you will look at on Thur/Fri. At a general level, not in detail.
- TA-led review session
  - Sat, Sept 29, 3-5 pm, 306 Soda
- Practice exams in your reader
  - Do these all at once (to simulate an exam)
  - Solutions to be announced on Course Portal

Some midterm like problems

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.

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)
```