CS3:

Introduction to Symbolic Programming

Lecture 5:
Recursion
Midterm-like problems

Fall 2006

Nate Titterton
nate@berkeley.edu

Schedule

4	Sept 18-22	Lecture: Data abstraction in DbD Lab: Miniproject I
5	Sept 25-29	Lecture: Introduction to Recursion Lab: Recursion
6	Oct 2-6	Lecture: *Midterm 1* Lab: Recursion II
7	Oct 9-13	Lecture: Advanced Recursion Lab: Recursion III
8	Oct 16-20	Lecture: finishing recursion Lab: Miniproject #2

Announcements

- **Nate's office hours:**
  - Wednesday, 1:30-3:30, in 329 Soda
  - Special: Monday Oct 2nd, 1-3pm, in 329 Soda
- **Midterm next week!**
  - (More on this in a bit)
- **Reading for this week**
  - Simply Scheme, chapter 11
  - Difference between Dates, Recursive version
  - (These *will* be on the midterm)
- **Still having trouble working at home?**
  - Go to 333 Soda hall !

Drop day

- The last day to drop is Sept 29
  - I think

Recursion

- Everyone thinks it's hard!
  - (well, it is… aha!-hard, not complicated-hard)

- The first technique (in this class) to handle arbitrary length inputs.
  - There are other techniques, easier for some problems.

- What is it?

An algorithmic technique where a function, in order to accomplish a task, calls itself with some part of the task.

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

Problem: *find the first even number in a sentence of numbers*

```
(define (find-first-even sent)
  (if (even? (first sent))

     (first sent)              ;base case: return
                               ; that even number
     (find-first-even (bf sent)))
                               ;recurse on the
                               ; rest of sent
     ))
```

```
  (if <test>

     (<do the base case>)

     (<do the recursive case>)
```

Count the number of words in a sentence

```
(define (count sent)

  (if (empty? (bf sent))

     1                  ;base case: return 1

     (+ 1
       (count (bf sent)) ;recurse on the
                         ; rest of sent

))
```

Base cases can be tricky

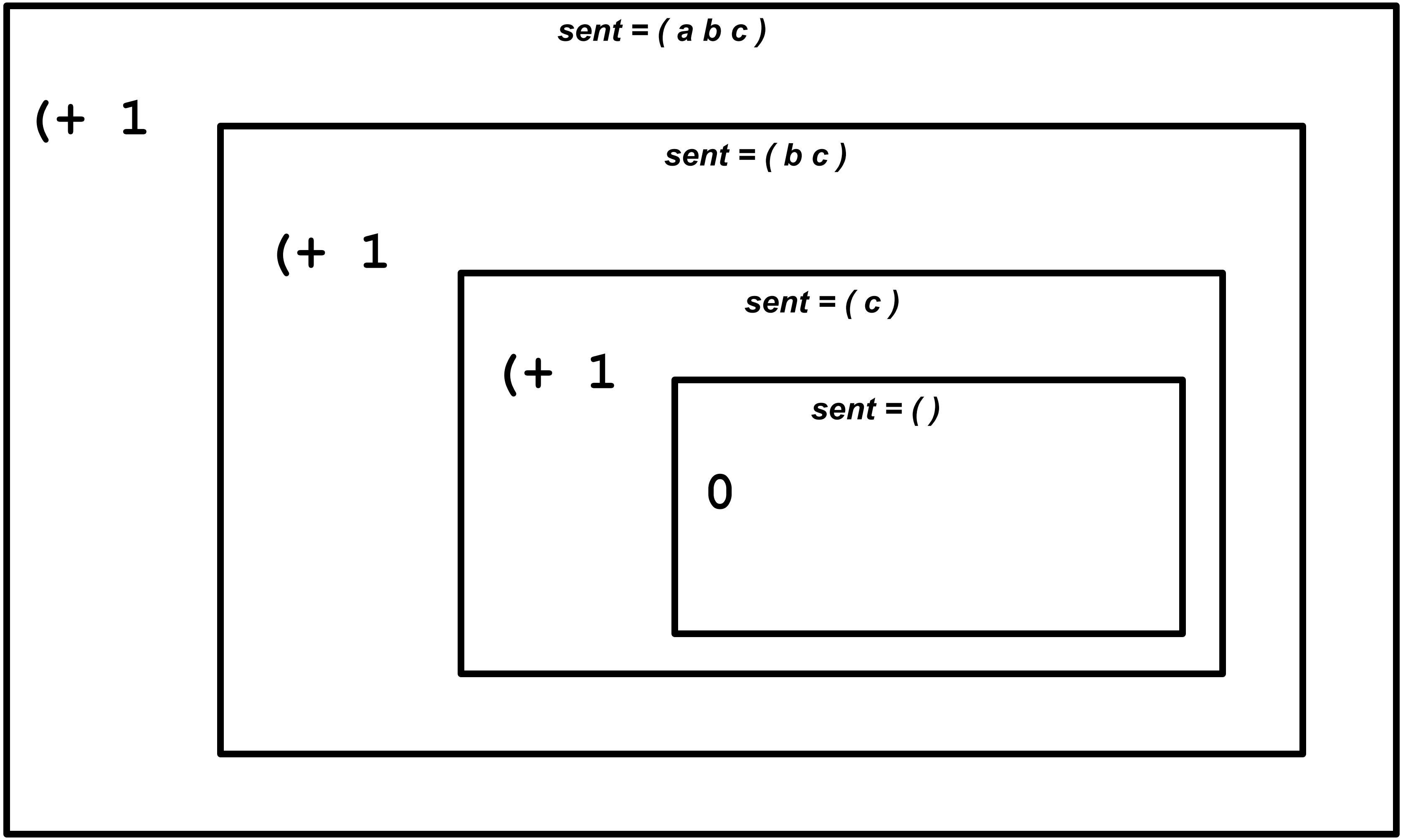
- **By checking whether the `(bf sent)` is empty, rather than `sent`, we won't choose the recursive case correctly on that last element!**
  - **Or, we need two base cases, one each for the last element being odd or even.**
- **Better: let the recursive cases handle *all* the elements**

Your book describes this well

Count the number of words in a sentence

```
(define (count sent)

   (if (empty? (bf sent)) )

       0                        ;base case: return 0

       (+ 1
         (count (bf sent))  ;recurse on the
                            ; rest of sent

))
```

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



sent = (a b c)

(+ 1

sent = (b c)

(+ 1

sent = (c)

(+ 1

sent = ( )

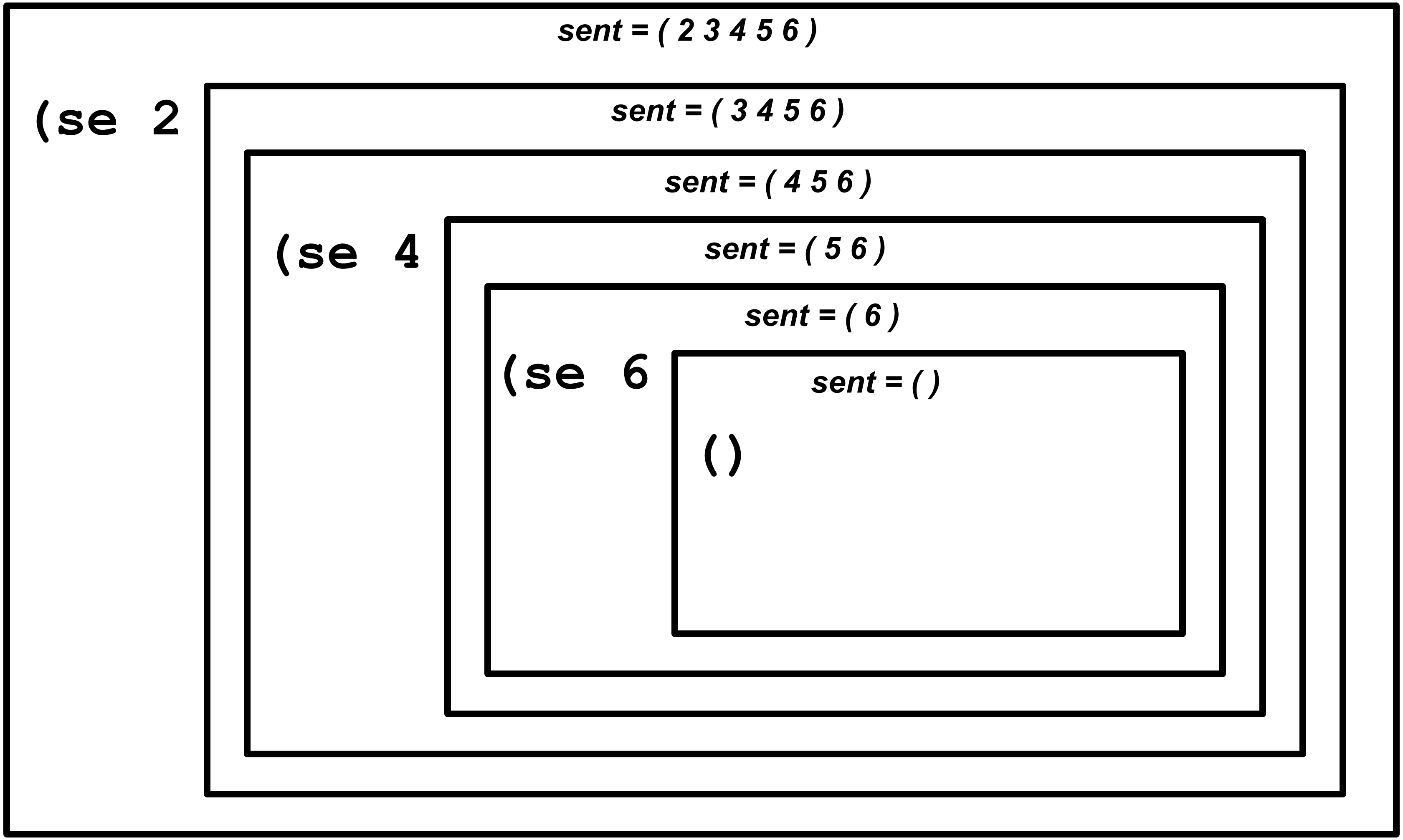
0

```
➔ (+ 1 (+ 1 (+ 1 0)))
➔ 3
```

Problem: *find all the even numbers in a sentence of numbers*

```
(define (find-evens sent)
  (cond ((empty? sent)        ;base case
         '()       )
        ((odd? (first sent))  ;rec case 1: odd
         (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)

sent = (4 5 6)

`(se 4`

sent = (5 6)

sent = (6)

`(se 6`

sent = ( )

`()`

```
➔ (se 2 (se 4 (se 6 ())))
➔ (2 4 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.**
  - Knowing recursion WILL help with all sorts of ways while programming, even if you don’t often use it.

Midterm 1: Oct 2nd (next week)

- Location: 50 Birge
- Time: In the lecture slot, plus 20 minutes
  - (4:10-5:30)

- Open book, open notes.
  - Nothing that can compute, though

- Everything we’ve covered, including the coming week on recursion.

- Review session this Saturday, Sep 30th, 2-4pm.
  - 430 Soda (Wozniak lounge).

- Practice exam in reader (do this all at once)
- Check Announcements for more practice items, solutions

Special midterm issues

- Several of you have come to me with midterm conflicts
  - You need to email me (and get a response) or talk to me so I can get a count!

Sample problem for midterm 1

Consider a procedure named `double` that, given a word as argument, returns a two-word sentence. The first word is two. The second word is the result of adding an "s" to the end of the argument.

expression	*intended result*
`(double 'apple)`	`(two apples)`
`(double 'bus)`	`(two buss)`
`(double 'box)`	`(two boxs)`

Now consider some *incorrect* implementations of double. For each one, indicate what the call

`(double 'apple)`

will return. If no value is returned because the procedure crashes, give the error message that results.

```
(define (double wd)
    (sentence 'two '(word wd s)))
```

```
(define (double wd)
    (sentence 'two (sentence wd s)) )
```

```
(define (double wd)
    (sentence 'two (wd 's)) )
```

between?

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`.

Part B: Write `between?` without using `and` or `or`.

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 describe what the result of the call should be!