CS3:
Introduction to Symbolic Programming

Lecture 5:

Spring 2006

Nate Titterton
nate@berkeley.edu

Announcements

- **Nate's office hours *this week only*:**
  - Thursday, 2-4, in 329 Soda
  - (Usually, they are Wed 2-4)

Schedule

4	Feb 6-10	Lecture: Data abstraction in DbD Lab: Miniproject I
5	Feb 13-17	Lecture: Introduction to Recursion Lab: Recursion
6	Feb 20-24	Lecture: *HOLIDAY* Lab: Recursion II
7	Feb 27-Mar 3	Lecture: *Midterm 1* Lab: Recursion III
8	Mar 6-10	Lecture: finishing recursion Lab: Miniproject #2
9	Mar 13-17	Introduction to Higher Order Proceedures

Announcements

- **Reading for this week**
  - Simply Scheme, chapter 11
  - Difference between Dates, Part II
- **Questions for the Jon (the reader), regarding homework grades?**
  - email Jon at cs3-ra@imail.eecs.berkeley.edu

Announcements

- **Recursion in Lab this week.**

  Read Chapter 11 **in the textbook before lab.**

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

))
```

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

```
(define (find-evens sent)
  (cond ((empty? (bf sent))   ;base case (??)
         (first sent)  )
        ((odd? (first sent)) ;rec case 1: odd
         (find-evens (bf sent)) )
        (else                 ;rec case 2: even
         (se (first sent)
             (find-evens (bf 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

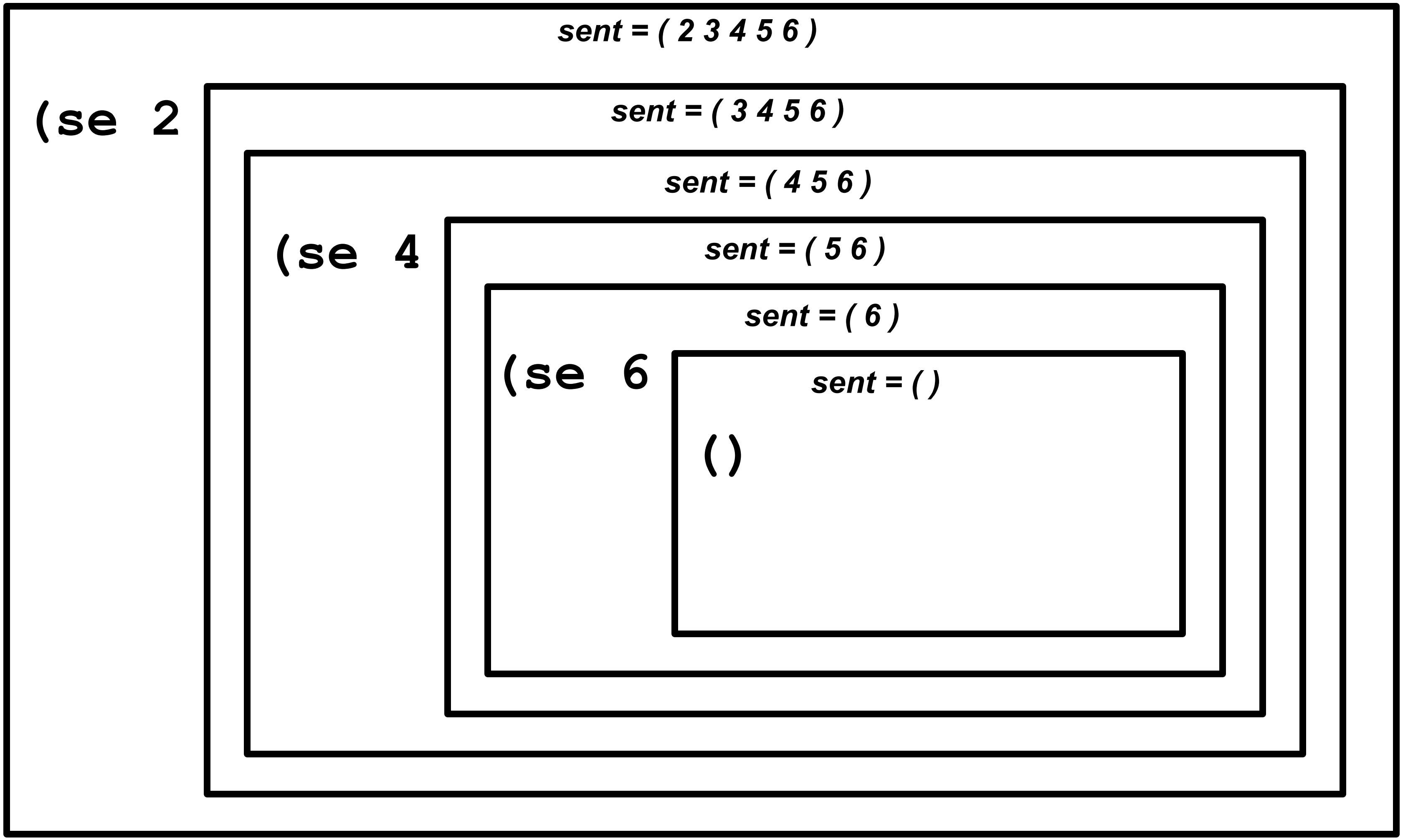
Count the number of words in a sentence

```
(define (count sent)

  (if (empty? (bf sent)) )

     0                          ;base case: return 0

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

))
```

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

```
(define (find-evens sent)
  (cond ((empty? (bf sent))    ;base case
         (first sent)  )
        ((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))
```




```
➔ (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: Feb 27th (in two weeks).

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

- Everything we've covered, including the coming two weeks on recursion.

- Review session Saturday, Feb 25th, 2-4pm.
  - 430 Soda (Wozniak lounge).
- Practice exam in reader (do this all at once)
- Check Announcements for more practice items, solutions

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)) )
```