CS3 Spring 2006 Midterm #1 Review

Suggestions for studying: do as many problems as you can

1. Follow the link on the UCWise website to past exams.
2. The reader also contains past exams.
3. Lab material: Your wonderful lab assistant, Anita, has put up her notes on each lab at this link: http://inst.eecs.berkeley.edu/~cs3-lv
4. Practice chapter problems in the textbook.
5. Extra Problems online:
   http://hiroki.ucdev.org/cs3spring06
   http://inst.eecs.berkeley.edu/~cs3-td
6. If you haven't done the reading (book and case studies), you should (especially the case studies).

Problems

1. Quickies: Evaluate the following expressions.

(first (butfirst '(cs3)))   -> ERROR, can't do (first '( ))

(or 4 (/ 4 0) 'so-true 'super-true)   -> 4

(and + '+ 5 (= 3 4))   -> #f

(and < 'false (or #t)) -> #t

(word) -> ""

(sentence) -> ( )

(sentence "") -> ("")

(sentence 'butfirst 'of 'abc 'is (butfirst abc))) -> ERROR, undefined var abc

(if (and) (or) (and)) ->#f. (and) returns #t, (or) returns #f

(bf (bl (item (remainder 5 4) '(fu andrew hiroki bobak)))) -> ""

(count (day-span '(january 0) '(january 0))) -> 1

(+ 1 (first (quotient (word 3 4) 3))) 2

(starts-with-prefix? '(X I V)) -> #f

2. Remainder – Recursion, if/cond v.s. and/or/not

Scheme has a built-in procedure remainder. Here is a sample call: (remainder 8 3)→2. Now write your own remainder procedure: my-remainder1 using if and/or cond, and my-remainder2 using ands and/or ors.

```
(define (my-remainder1 num1 num2)
   (if (< num1 num2)
      num1
      (my-remainder (- num1 num2) num2) ) )

(define (my-remainder2 num1 num2)
  (or (and (< num1 num2) num1)
      (my-remainder2 (- num1 num2) num2) ) )
```

[Challenge: write remainder without using recursion]

```
(define (remainder3 num1 num2)
  (- num1  (* (quotient num1 num2) num2)  ) )
```

3. Largest (Recursion)

Define a procedure to find the largest number in two unsorted sentences. Do not use the built-in max procedure.
(largest '(3 1 8 4) '(9 2 5)) → 9

Solution using Accumulating recursion:

```
(define (largest sent1 sent2)
  (largest-h (first sent1) (se (bf sent1) sent2) ) )

(define (largest-h max-so-far sent)
  (cond ((empty? sent) max-so-far)
          ((< max-so-far (first sent) )
           (largest-h (first sent) (bf sent) ) )
          (else (largest-h max-so-far (bf sent) ) ) ) )
```

[Challenge: Define a procedure range that finds the smallest and largest number in two unsorted sentences. Ex: (range '(3 1 8 4) '(9 2 5)) → (1 9)]

```
(define (range sent1 sent2)
   (se (smallest sent1 sent2) (largest sent1 sent2) ) )

(define (largest sent1 sent2)
  SAME AS ABOVE )

(define (smallest sent1 sent2)
  (smallest-h (first sent1) (se (bf sent1) sent2) ) )

(define (smallest-h min-so-far sent)
  (cond ((empty? sent) min-so-far)
          ((> min-so-far (first sent) )
           (smallest-h (first sent) (bf sent) ) )
          (else (smallest-h min-so-far (bf sent) ) ) ) )
```

4. Remove-Card (Recursion + Data Abstraction)

A card is represented as a word: suit-rank. For example, c-3, h-k. Define a procedure to remove a specified card from a sentence of cards. Ex:
(remove-card 'c 3 '(c-3 h-k d-a c-3 s-q c-2))→ (h-k d-a s-q c-2)
Define accessors to get suit and rank of a card when doing comparisons.

```
(define (suit card)
  (first card) )

(define (rank card)
  (bf (bf card) ) )  ;; note that (last card) won't work for cards like h-10

(define (remove-card s r sent)
  (cond ((empty? sent) '( ) )
          ((and (equal? s (suit (first sent) ) )  (equal? r (rank (first sent) ) ) )
           (remove-card s r (bf sent) ) )
          (else (se (first card) (remove-card s r (bf sent) ) ) ) ) )
```

[Challenge: Define replace-card such that, the specified card in the sentence is replaced by the word joker. Ex: (replace-card 'c '3 '(c-3 h-k d-a c-3 s-q c-2)) → (joker h-k d-a joker s-q c-2)]

```
(define (replace-card s r sent)
  (cond ((empty? sent) '( ) )
          ((and (equal? s (suit (first sent) ) )  (equal? r (rank (first sent) ) ) )
           (se 'joker (replace-card s r (bf sent)  )  ) )
          (else (se (first card) (replace-card s r (bf sent) ) ) ) ) )
```

5. Multiply - Recursion with multiple arguments

Consider the following multiply procedure. It takes two sentences of equal length as arguments, the first being a sentence of letters, and the second being a sentence of numbers (0 or greater). It returns a sentence with each letter in the first argument repeated n times, where n is the corresponding number in the second argument. Here are two sample calls:

```
(multiply '(a b c d) '(2 2 0 1)) → (aa bb d)
(multiply '(a b c d) '(0 0 0 0)) → ()
```

However, there is a bug in the given program.
1) Provide a test call that would return an incorrect result, and 2) fix the bug in the procedure.

```
(define (multiply sent1 sent2)
  (cond ((empty? sent1) ())
        ((= (first sent2) 0)
         (multiply (bf sent1) (bf sent2)))
        ((= (first sent2) 1)
         (se (first sent1) (multiply (bf sent1) (bf sent2))))
        (else
         (multiply (se (word (first sent1) (first  sent1)) (bf sent1))
                   (se (- (first sent2) 1) (bf sent2))))))
```

1. (multiply '(a) '(3)) would return (aaaa) , not (aaa)
2. the bug is in the else case. Fix it by changing the second (first sent) into (first (first sent1))

[Challenge: try writing multiply on your own]

Solution using copies from lab and accumulating recursion

```
(define (multiply sent1 sent2)
 (multiply-h sent1 sent2 '( ) ) )

(define (multiply-h sent1 sent2 sent-so-far)
    (if  (empty? sent1)
         sent-so-far
         (multiply-h (bf sent1)  (bf sent2)
                     (se sent-so-far (copies (first sent1) (first sent2) ) ) ) ) )

(define (copies wd num)
    (if (= num 0)
        '( )
        (se wd (copies wd (- num 1) ) ) ) )
```