

## Midterm #1

Professor Grillmeyer

**Problem #1:**

What do the following Scheme statements evaluate to. Write your answers below each expression in the space provided. **If an expression results in an error, indicate what the error is.** Assume that the following **define** has been made:

```
(define lst '(i (do not) want to be a ((list))))
```

```
> (list 'lst)                >(member (list (list 'list)) lst)
```

```
> (truncate (length lst))    > (cadadr lst)
```

```
> (remainder (/ 5 2))        > (subseq lst 5)
```

```
>(max '(2 -7 3 4))          > (subseq lst 1 0)
```

```
> (expt 10 1)                > (reverse lst)
```

```
> (/)                         > (cons '(1 (2)) '((3)))
```

```
> (second lst)                > (list '(1 (2)) '((3)))
```

```
> (position '(want) lst)      > (append '(1 (2)) '((3)))
```

```
> (rest '(1 2))               > (if (> 3 4) (+3 4) (* 3 4))
```

```
> (middle lst)                > (and (> 3 4) (+ 3 4) (* 3 4))
```

> (list-ref lst 5)                      > (or (> 3 4) (+ 3 4) (\* 3 4))

> (remove '(want to) lst)            > (cond (> 3 4) (+ 3 4) (\* 3 4))

> (count 'a '(a (a) a))              > (let\* ((lst 'word) (word 'lst)) (list lst word))

### Problem #2:

Answer true or false to the following questions. **Write out true and false (not T and F) in your answers.**

\_\_\_\_\_ The function **display** is the same as **quote**, since both return their arguments unevaluated.

\_\_\_\_\_ Special forms use the normal rules of evaluation but have special rules indicating what they should return.

\_\_\_\_\_ A call to the special form **if** can occur anywhere where an expression can be placed.

\_\_\_\_\_ A function's body can consist of many expressions which are all evaluated but only the final one is returned.

\_\_\_\_\_ The function **null?** is used to check if something is false.

### Problem #3:

Fill in the blanks in the following questions. You can put zero or more functions **and/or arguments per blank. Do not use the return value (e.g., lisp) in any part of your solution.**

> (\_\_\_\_\_ '(tis better (to scheme) than (to (lisp)))) \_\_\_\_\_)

lisp

> (\_\_\_\_\_ '(tis better (to scheme) than (to (lisp)))) \_\_\_\_\_)

(to)

> (\_\_\_\_\_ '(tis better (to scheme) than (to (lisp)))) \_\_\_\_\_)

0

```
> (_____ '(windoze 98 (yawn (yawn) yawn)) _____)
```

```
(yawn yawn)
```

```
> (_____ '(windoze 98 (yawn (yawn) yawn)) _____)
```

```
97
```

#### Problem #4:

Complete the function `worth-more` below that takes a list of two sublists representing items you can buy and a list of two numbers (the prices of the items) and forms a sentence of the form shown below.

```
> (worth-more '((bud light 6-pack) (dom perignon champagne)) '(3.45 85.00))
```

```
(dom perignon champagne/is worth/81.55/more than/bud light 6-pack)
```

```
(define (worth-more items prices)
```

```
  (let*
```

```
    ((low-price _____)
```

```
     (high-price _____)
```

```
     (low-item _____)
```

```
     (price-difference _____))
```

```
    ; final return value
```

```
    _____))
```

#### Problem #5:

Write a function `extract` that takes a list `a-list`, an atom or a list item, and a number `extra` and returns a list starting at the first location of `item` in `a-list` and including `extra` additional elements beyond that. You may assume that `item` is a top-level element in `a-list` and that there are at least `extra` elements afterwards. Here is an example.

```
> (extract '(give me a list or give me death) 'me 2)
```

```
(me a list)
```

Complete the function `extract` below.

```
(define (extract a-list item extra)
```

#### Problem #6:

Use the function definitions below to answer the following questions.

```

(define (part1 lst elt)
  (subseq lst (position elt lst)))
(define (part2 lst elt)
  (subseq lst 0 (position elt lst)))
(define (part3 lst elt)
  (subseq lst (+ (position elt lst) 1)))
(define (part4 lst elt)
  (subseq lst 0 (+ (position elt lst) 1)))
(define (part5 lst elt)
  (subseq lst (- (position elt lst) 1)))
(define (part6 lst elt)
  (subseq lst 0 (- (position elt lst) 1)))

```

Complete the function `change-section` that takes a list `a-list`, two items that occur in the list `elt1` and `elt2`, and a list to insert `new-stuff`. The function `change-section` returns a new list with the elements from `elt1` to `elt2` inclusive replaced with the elements in `new-stuff`. If `elt2` occurs before `elt1` in `a-list`, the elements from `elt2` to `elt1` inclusive should be replaced. For example,

```
> (change-section '(i could use coffee and donuts now) 'coffee 'donuts '(some sleep))
```

```
(i could use some sleep now)
```

```
> (change-section '(i could use coffee and donuts now) 'now 'i '(good night))
```

```
(good night)
```

Complete the function `change-section` below. Use some version of `part` (defined above) in the `let` variables. `elts` is the list of `elt1` and `elt2` in the order they occur in `a-list`. `left-part` is the items in `a-list` before the first element of `elts`. `right-part` is the items in `a-list` after the second element of `elts`.

```

(define (change-section a-list elt1 elt2 new-stuff)
  ; get elt1 and elt2 in order and then get the left and right parts of a-list

  (let* ((elts (if (< (length (part_ a-list elt1))
                    (length (part_ a-list elt2)))
                (list elt1 elt2)
                (list elt2 elt1)))
         (left-part (part_ a-list (first elts)))
         (right-part (part_ a-list (second elts))))
    ; put together the three pieces to form a new list
    _____))

```

CS 3

**Posted by HKN (Electrical Engineering and Computer Science Honor Society)  
University of California at Berkeley  
If you have any questions about these online exams  
please contact [examfile@hkn.eecs.berkeley.edu](mailto:examfile@hkn.eecs.berkeley.edu).**