Midterm Questions (CS115 Fall 2009)

Last year our midterm was broken into two sections (module 1-2, and module 3 to the present): Here are some of the key questions from the two midterms

1. Determine what would happen if you opened DrScheme and tried to evaluate the given code. If there is an error, say what went wrong.

For everything after letter h: use the following code that has been defined for you

```
(define my-list (cons 2
                      (cons 'red
                            (cons (make-posn -1 4)
                                  (cons 'blue empty)))))
```

a) (* 2 (min 5 3)) => 6

b) ((+ 2 2)(- 2 2)) => error. You cant evaluate (4 0)

c) (3 – 4) => error. Should be (- 3 4)

d) (+ 1 (3)) => error. Should be (+ 1 3)

e) (* (+ 1 3) (max 5 (- 4 3))) => 20

f) (sqr (string-length "at")) => 4

g) (/ 3 (* 5 0)) => syntax is okay, but error, you cant evaluate (/3 0)

h) (length my-list) => 4

g) (member 4 my-list) => error. 4 is not a member of the list

h) (rest (rest (rest my-list))) => (cons 'blue empty)

i) (cons 5 (first my-list)) => error: remember, the definition of a list is that a list a (first my-list) and (rest my-list). (rest_mylist) must be a list.

j) (first (rest my-list)) => 'red

h) (empty? (rest my-list)) => false

2) Trace (* (+ (min 2 3)(- 6 5))5)

=> (* (+ 2 (- 6 5))5)

=> (* (+ 2 1)5)

=> (* 3 5)

=> 15

3) A company wishes to represent information about vehicles that have been towed: model, colour, license plate, and a fine to be paid. For example, one vehicle might be a silver Mercedes with licence plate TRYO 456 for which $500.50 is owed.

a) give a structure definition for the *vehicle*. Choose appropriate field names

(define-struct vehicle (model colour plate fine))

b) give a data definition for the structure vehicle. Choose appropriate types for each field

```
;; a vehicle is a structure (make-vehicle m c p f) where
;; m is a string
;; c is symbol
;;p is a string and
;;f is a num
```

c) define a constant my-vehicle of type vehicle corresponding to the example given above

(define my-vehicle (make-vehicle “Mercedes” ‘silver “TRYO 456” 500.5))

4) Using
(define-struct employee (name ID salary))

a) List the functions that are automatically created
make-employee
employee?
employee-name
employee-ID
employee-salary

b) give a template for a function that consumes an employee
```
(define (new-employee employee)
        ...(employee-name employee)...
        ...(employee-ID employee)...
        ...(employee-salary employee)...)
```

c) Give a recursive data definition for employee-list, which is a list of employee.
```
;;an employee-list is either
;; empty or
;; (cons f r) where f is an employee and r is a list of employee
```

d) create a function salary-sum that consumes an employee-list and produces the sum of the salaries of the employees in the list

```
(define (salary-sum employee-list)
        (cond
                [(empty? employee-list)0]
                [else (+ (employee-salary (first employee-list))(salary-sum (rest employee-list)))]))
```

4. The function my-fun consumes two integers and produces values as indicated below

```
(define (my-fun arg1 arg2)
        (cond
                [(or (zero? arg1)(zero? arg2))'done]
                [(> arg1 arg2)(/ arg2 arg1)]
                [else (/ arg1 arg2)]))
```

a) provide a contract

```
;;my-fun: int int -> (union symbol num)
```

b) produce a thorough set of tests for the function

```
(check-expect(my-fun 0 4)'done)
(check-expect(my-fun 3 0)'done)
(check-expect(my-fun 4 2)0.5)
(check-expect(my-fun 3 12)0.25)
```

c) trace the function application (my-fun 0 1)

```
=> (cond
        [(or (zero? 0)(zero? 1))'done]
        [(> 0 1)(/ 1 0)]
        [else (/ 0 1)]))
=> (cond
        [(or (true)(zero? 1))'done]
        [(> 0 1)(/ 1 0)]
        [else (/ 0 1)]))
=> (cond
        [(true)'done]
        [(> 0 1)(/ 1 0)]
        [else (/ 0 1)]))
=> 'done
```

5.

```
(define-struct card (value suit))
;;A card is a structure (make-card v s), where
;; v is an integer from 1 to 10
;; s is a symbol from the set ‘hearts, ‘diamonds, ‘spades, ‘clubs
```

a)Create a scheme function even-card? That consumes a card and produces true if the value is even and false otherwise

```
(define (even-card? acard)
        (cond
                [(even? (card-value acard))true]
                [else false]))
```

b) Create a scheme function evens that consumes a list, cardlist, of cards and produces a list with each card in cardlist that has an even number

```
(define (evens cardlist)
        (cond
                [(empty? cardlist) empty]
                [(even? (card-value (first cardlist)))(cons (first cardlist)(evens (rest cardlist)))]
                [else (evens (rest cardlist))]))
```

c) create a function change-all that consumes a list of cards and produces a list with each card replaced by one with the same suit but one higher value (or 1, if the original value was 10)

```
(define (change-all cardlist)
        (cond
                [(empty? cardlist) empty]
                [(and (<= (card-value (first cardlist))9)(>= (card-value (first cardlist))1)) (cons
(make-card ((+ (card-value (first cardlist))1)(card-suit (first cardlist))))(change-all (rest cardlist))))]
                [(equal? (card-value (first cardlist))10)(cond(make-card (1 (card-suit (first
cardlist))))(change-all (rest cardlist))]))
```

6. Create a scheme function odd-symbols that consumes a non-empty list of any values, produces true if the number of symbols is odd and produces false otherwise. You may want to use the built in function symbol?

;;helper function that eliminates any value that is not a symbol

```
(define (list-of-symbols alov)
        (cond
                [(empty? alov)empty]
                [(symbol? (first alov))(cons (first alov)(list-of-symbols (rest alov)))]
                [else (list-of-symbols (rest alov))]))

(define (odd-symbols alov)
                (even? (length (list-of-symbols alov))))
```