University of Waterloo
CS115 Midterm Examination

Term: Spring **Year:** 2010

Date: Monday, June 21, 2010
Time: 7:00 – 9:00 pm
Instructor: Lori Case
Lecture Sections: 001
Exam Type: Closed book
Additional Materials Allowed: Provided Reference Sheet

Last Name: ______________________

First Name: ______________________

ID: _ _ _ _ _ _ _ _ _

Signature: ________________

Instructions: (Read carefully before the exam begins):

1. Before you begin, make certain that you have one Exam Booklet with 10 pages and a separate Reference Sheet.
2. The Scheme language level is Beginner Student Scheme with List Abbreviations.
3. Supply exactly the parts of the design recipe requested in each question. Unless otherwise told, "complete" a function means to provide just the definition (body).
4. You may use a helper function where you feel it is needed. For each helper function, you are only required to write the function body (definition).
5. You may use any function defined in the exam as a helper function for any other function.
6. All solutions must be placed in this booklet.
7. If you need more space to complete an answer, you are likely writing too much. However, if you need more space, use the last page, and indicate that you have done so in the original question.
8. Relax! Read this instruction as often as needed.

Question	Marks Given	Out Of	Marker's Initials
1		10	
2		5	
3		3	
4		8	
5		10	
6		8	
7		6	
8		8	
9		4	
Total		60	

1. [10 marks] For each row in the table below, determine what would happen if you opened DrScheme and tried to evaluate the given code. If there is an error, briefly explain what is wrong (be more specific than saying “invalid Scheme expression”). If there is no error, give the value produced by DrScheme. The first two rows have been completed for you as examples.

Scheme Code	Answer
`(+ 1 2)`	3
`(+ 1 2))`	Error. There is an extra)
`(7 + 12)`	
`(+ 4 (* 1 (+ 4 2)))`	

`(+ 1 ((remainder 27 9)`
`      (sqr 5)))`	

`(/ 10 (- 3 (- 1 4)))`	
`(= ‘apple ‘banana)`	
`(and (< 3 4) (< 19 10))`	
`(- 12 (string-length “hiya”))`	

`(rest (rest (rest (rest`
`   (cons ‘a`
`     (cons 3`
`       (cons 5 empty)))))))`	

`(cond [(> 3 8) ‘happy]`
`      [(- 3 4) ‘sad]`
`      [(even? 9) ‘oops])`	

`(posn-x`
`  (first`
`    (rest`
`      (cons (make-posn 3 4)`
`        (cons (make-posn 5 6)`
`              empty)))))`	

2. [5 marks] Given the following definitions.

```
(define k 4)
(define (g x) (+ x k))
```

Trace the Scheme expression below. Show all steps, and put one step on each line. You may not need all the lines.

`(+ (/ k 2) (- (g 2) (g k)))`

3. [3 marks] Complete the body of a Scheme function named `f` to calculate
$f(x) = 1 + 1/2\ x + 1/3\ x^2$.

4. [8 marks] Consider the following Scheme code:

```
(define (eval v1 v2)
  (cond
    [(and (> v1 0) (> v2 0)) (/ (+ v1 v2) 2)]
    [(or (> v1 0) (> v2 0)) (max v1 v2)]
    [else 'negative-values]))
```

a) [2 marks] Write the contract for `eval`.

b) [2 marks] Write a purpose for `eval`.

c) [4 marks] Write 4 separate, distinct tests for `eval`.

(check-expect
(check-expect
(check-expect
(check-expect

5. [10 marks] Consider the card structure definition.

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

a) [2 marks] Define a constant called `four-of-hearts` corresponding to the card with suit `'hearts` and value 4.

Consider the Scheme function `merge-cards` which consumes two card structures (`c1` and `c2`) and produces a new card, according to the following rules:

- If the suits of the two cards are the same, the new card has that suit and its value is the sum of the values of c1 and c2, up to a maximum value of 10.
- Otherwise, the new card has the value of c1 and the suit of c2.

b) [2 marks] Write the contract of `merge-cards` (the function header is given below).

c) [2 marks] Write a single example for `merge-cards`.

d) [6 marks] Complete the body of the function `merge-cards`.

```
(define (merge-cards c1 c2)
```

6. [8 marks] Consider the Scheme function `drop-all-first` which consumes `lst`, a list of non-empty strings, and produces a new list containing the same strings as `lst`, except that the first character from each is removed. For example, `(drop-all-first (cons "cat" (cons "dog" (cons "sheep" empty))))` produces `(cons "at" (cons "og" (cons "heep" empty)))`.

   a) [2 marks] Write the contract for `drop-all-first` (the function header is given below).

```
;; drop-all-first:
```

   b) [6 marks] Complete the body of `drop-all-first`.

```
(define (drop-all-first)
```

7. [6 marks] Complete the Scheme function `within` which consumes a number `n`, a non-negative number `base`, and a non-negative number `range`, and produces `true` if `n` satisfies `(base - range) <= n <= (base + range)`, and `false` if not. For example, `(within 10 5 6) => true, and (within 10 5 2) => false.`

```
(define (within n base range)
```

8. [8 marks] Consider the Scheme function `all-within` which consumes a list of numbers and two non-negative numbers `base` and `range`, and produces the list of just those values that are between `(base-range)` and `(base+range)`, inclusive. For example, `(all-within (cons 4 (cons 10 (cons -1 (cons 5 (cons 8 (cons 1 empty)))))) 5 3)` produces `(cons 4 (cons 5 (cons 8 empty)))`.

   a) [6 marks] Complete the definition (the header and body) of `all-within`.

   b) [2 marks] Use `check-expect` to write one test for `all-within`. Do not use the given example.

9. [4 marks] Consider the following Scheme functions.

```
(define (delta x)
  (cond [(even? x) 2]
        [else 1]))

(define (multiply-some numbers)
  (cond [(empty? numbers) 1]
        [else (* (delta (first numbers))
                          (multiply-some (rest numbers)))]))
```

Write a condensed trace, showing the recursive calls of `multiply-some` and calls to `delta`, for the following expression. Put one step on each line. You may not need all the lines.

`(multiply-some (cons 21 (cons 4 (cons 62 empty))))`