Department of Computer Science
CSC 326H1F – Fall 2008

Test 2

Aids allowed: one 2-sided sheet

Time: 50 minutes

Total marks: 44

Name: _____

Student number: _____

1	/14
2	/10
3	/10
4	/10
Total	**/44**

1. [14 marks]

In each part of this question, function definitions are provided, followed by one or more expressions. For each expression, give the value that is returned. If no value is returned because an error occurs, clearly explain the error. When explaining an error, you are not required to provide the actual error message.

(a) [3 marks]

```
(define (f x) (if (number? x) x (f (car x))))
```

i) `(f '(9 (8 2) 7))`

9 [1 mark]

ii) `(f '(((5 6)) 10 11 (12) 13))`

5 [2 marks]

(b) [3 marks]

```
(define (g x y)
   (cond ((null? x) y)
         ((null? y) x)
         (else (cons (car x) (cons (car y) (g (cdr x) (cdr y)))))
         ))
```

i) `(g '(1 2 3) '(a b c))`

(1 a 2 b 3 c) [1 mark]

ii) `(g '(6 7 8 9 0) '(x y z))`

(6 x 7 y 8 z 9 0) [2 marks]

(c) [3 marks]

```
(define (h e y)
    (map (lambda(x) (cons e x)) y))
```

i) `(h 'a '())`

`()` [1 mark]

ii) `(h 2 '((a) (c) (e)))`

`((2 a) (2 c) (2 e))` [2 marks]

(d) [3 marks]

```
(define (j x y)
    (eval (cons x y)))
```

i) `(j + '(2 3 4))`

`9` [1 mark]

ii) `(j append '((b) (c)))`

This causes an error. [1 mark]
`eval` is called on `(append (b) (c))`, causing the evaluation of `b`. [1 mark]

(e) [2 marks]

```
(define (k x y)
    (apply cons (list x y)))
```

i) `(k 'car '(1 2 3))`

`(car 1 2 3)` [2 marks]

2. [10 marks]

(a) [4 marks]

Write a function **everyOther** that takes a list **L**, and returns a new list containing every other element of **L** starting with the first — that is, returns a list containing the 1st, 3rd, 5th, … elements of **L**. Example:

```
> (everyOther '(d e f g h i))
(d f h)
```

```
(define (everyOther L)
    (cond ((null? L) L)
          ((null? (cdr L)) L)
          (else (cons (car L)
                      (everyOther (cddr L))))
          ))
```

(b) [6 marks]

Write a function **addTwo** that takes a nested list of numbers, and returns a new list that is the same as the given list except that each number, no matter how deeply it's nested, has been increased by 2. Examples:

```
> (addTwo '(4 7 9))
(6 9 11)
> (addTwo '((5) (6 (7)) 8 (((10))) 1))
((7) (8 (9)) 10 (((12))) 3)
```

```
(define (addTwo L)
    (cond ((null? L) L)
          ((list? (car L)) (cons (addTwo (car L))
                                 (addTwo (cdr L))))
          (else (cons (+ 2 (car L))
                      (addTwo (cdr L))))
          ))
```

3. [10 marks]

(a) [6 marks]

Write a function **dotProduct** that takes two lists of numbers as parameters, and returns the number that is the sum of the products of the corresponding pairs of numbers in the two lists — that is, the mathematical dot product of two vectors. For example, the dot product of (3 4 5) and (6 2 –1) is 3*6 + 4*2 + 5*(–1) = 21. You may assume the lists are of equal length. **You must *not* use recursion and you must *not* define any helper functions.** Example:

```
> (dotProduct '(2 4 4) '(6 2 -1))
16
```

```
(define (dotProduct L M)
    (apply + (map * L M)))
```

(b) [4 marks]

Write a function **composeN** that takes a unary function **F** and a non-negative integer **N**, and returns a function formed by composing **N** copies of **F**. When **N** is 0, return the identity function. Examples:

```
> ((composeN cdr 0) '(1 2 3 4))
(1 2 3 4)
> ((composeN cdr 1) '(1 2 3 4))
(2 3 4)
> ((composeN cdr 2) '(1 2 3 4))
(3 4)
```

```
(define (composeN F N)
    (cond ((= N 0) (lambda (X) X))
          (else (let ((G (composeN F (- N 1))))
                    (lambda (X) (F (G X)))))
          ))
```

4. [10 marks]

Suppose we represent a BST (binary search tree) as described in class, so that each non-null tree node is a list of three items: the data value stored in the node, the left child node, and the right child node. Assume that all data values are numbers. You may ***not*** use the BST helper functions defined in class (unless you first define them yourself).

(a) [4 marks]

Write a function **`largest`** that takes a BST and returns the largest data value stored in the tree. You may assume that the given BST is not empty. (Hint: Recall that in a BST, the largest data value is found in the "right-most" node.) Example:

```
> (largest '(4 () (5 () ())) )
5
```

```
(define (largest L)
    (cond ((null? (caddr L)) (car L))
          (else (largest (caddr L)))
          ))
```

(b) [6 marks]

An *internal node* is a node with at least one non-null child. Write a function **`numInternals`** that takes a BST and returns the number of internal nodes in the tree. You may ***not*** assume that the tree is non-empty. Example:

```
> (numInternals '(4 () (5 () ())) )
1
```

```
(define (numInternals L)
    (cond ((null? L) 0)
          ((and (null? (cadr L)) (null? (caddr L))) 0)
          (else (+ 1
                   (numInternals (cadr L))
                   (numInternals (caddr L))))
          ))
```