

Assignment 1A

Due Friday January 30 at the start of tutorial.

No late assignments will be accepted.

What to do

The questions below require you to write Scheme functions. These are warm-up exercises to familiarize you with Scheme, with recursion, with list processing, and with higher-order functions. Your Scheme functions should be well commented, and should be written in good functional programming style. In particular, do *not* use any functions or constructs that change the values of variables or have other side effects or that use a sequential processing of commands. For example, you may not use `do`, `begin`, or any function ending in `!`, such as `set!`, `set-car!`, `vector-set!`, etc. Feel free to use helper functions wherever appropriate. Hand in a source listing of your Scheme functions and a sample terminal session with the Scheme interpreter. The terminal session should be short, and should demonstrate that your functions work correctly. In general, simple solutions are preferred and will receive the most marks. Each question is worth 5 points.

1. Define a Scheme function (`firstNum L`) that returns the first number in list `L`. For example, (`firstNum '(a b 2 c 3)`) => 2. You may assume that `L` contains at least one number.
2. Define a Scheme function (`numbers L`) that returns a list of all the numbers in list `L` in order. For example, (`numbers '(a b 2 c 3)`) => (2 3).
3. Define a Scheme function (`allNumbers E`) that returns a list of all the numbers in `s-expression E` in order. For example,

```
(allNumbers '(a b 2 c 3)) => (2 3)
```

```
(allNumbers '(a (b (2 (c 3)))))) => (2 3)
```

```
(allNumbers '(((1 (a b) 3)))) => (1 3)
```

```
(allNumbers 13) => (13)
```

```
(allNumbers 'b) => ()
```

4. Define a Scheme function (`pSum2 E`) that sums all the positive numbers in s-expression E. For example,

```
(pSum2 '(1 -2 b 4)) => 5
```

```
(pSum2 '(((1 -2) (b (4 -1 5)))) => 10
```

```
(pSum2 'b) => 0
```

```
(pSum2 7) => 7
```

5. Using mutual recursion, define two Scheme functions called `remEven` and `remOdd`, where (`remEven L`) removes from list L all those elements in even positions, and (`remOdd L`) removes those elements in odd positions. For example, in the list (a b c d), the element a is in position 1, b is in position 2, etc. Thus,

```
(remEven '(a b c d)) => (a c)
```

```
(remOdd '(a b c d)) => (b d)
```

For full marks, you *must* use mutual recursion (which gives a simple and elegant solution).

6. Suppose F is a binary function. Define a higher-order function (`replace-pairs F L`) that replaces X and Y by (F X Y), for each pair of adjacent elements, X and Y, in list L. You may assume that L is a list of even length. For example,

```
(replace-pairs list '(1 2 3 4 5 6)) => ((1 2) (3 4) (5 6))
```

```
(replace-pairs + '(1 2 3 4 5 6)) => (3 7 11)
```

```
(replace-pairs (lambda (X Y) (* X (+ Y 1))) '(1 2 3 4 5 6)) => (3 15 35)
```

7. Suppose that LF is a list of binary functions, and L1 and L2 are lists of the same length as LF. Define a function (`apply-all LF L1 L2`) that applies each function in LF to the corresponding elements in L1 and L2, and constructs a list of the results. For example,

```
(apply-all (list + * -) '(1 2 3) '(3 4 5)) => (4 8 -2)
```

```
(apply-all (list cons cons cons) '(a m u) '((b c) (n o) (v w)))  
=> ((a b c) (m n o) (u v w))
```

```
(apply-all (list cons list (lambda (X Y) (cons (car X) (cdr Y))))  
  '(a m (u v w))  
  '((b c) n (x y z)))  
=> ((a b c) (m n) (u y z))
```

Cover sheet for Assignment 1A

Complete this page and attach it to the front of your assignment.

Name: _____
(Underline your last name)

Student number: _____

I declare that this assignment is solely my own work, and is in accordance with the University of Toronto Code of Behavior on Academic Matters.

Signature: _____