CMPSC 461: Programming Language Concepts
Assignment 2. Due: Sep. 25, 11:59PM

For this assignment, you need to submit your solution as one single file to Canvas. You may NOT use any of Scheme's imperative features (assignment/loops) or anything else not covered in class. Define auxiliary functions where appropriate. While you may use whatever implementation of Scheme you like, we highly recommend using Petite Scheme (www.scheme.com) or repl.it (www.repl.it/languages/scheme). We will be testing your code on Petite scheme. For all problems, you can assume all inputs obey the types as specified in a problem. We have provided a test file "hw2-test.scm" on Canvas for your testing.

We will be running your programs against a script. So it is important to check the following requirements before you submit: 1) the function names in your submission must match EXACTLY as specified in this assignment; 2) make all of your function definitions global (i.e., use "define"); 3) name your submission as psuid.scm (e.g., xyz123.scm); 4) make sure the file you submit can be directly loaded into Scheme (to test, use command load "file_name" in Scheme, or copy-and-paste your code into repl.it and make sure that there is no error). Failing to follow these requirements may result in NO CREDIT for your submission.

Problem 1 [6pt] In assignment 1, we have encoded a pair and operations on pair as follows:

$$\texttt{PAIR} \triangleq \lambda a\ b\ p.\ (p\ a\ b)$$

$$\texttt{LEFT} \triangleq \lambda p.\ (p\ (\lambda t\ f\ .\ t))$$

$$\texttt{RIGHT} \triangleq \lambda p.\ (p\ (\lambda t\ f\ .\ f))$$

Implement functions `PAIR`, `LEFT` and `RIGHT` in Scheme. For example, (`LEFT` (`PAIR` 1 2)) should return 1. Hint: you need to curry the definition of `PAIR` in an appropriate way.

Problem 2 [6pt] In lecture, we use $(f^n\ z)$ to represent $f(f(\ldots(f\ z)))$ where f is repeated for n times. In general, a composition of functions is written as $(f_1 \circ f_2 \cdots \circ f_n)$. A composition of functions is yet another function, which takes inputs from the domain of f_n, and continuously apply f_n, f_{n-1}, and so on to the result of the previous function application.

Define a function `funcompose` that takes a list of functions and return their composition. For this question, assume that all functions take one input, and returns one output. Here are a couple of examples:

```
((funcompose '(sqrt floor)) 9) ; 3
((funcompose '((lambda (x) (+ 1 x)) floor)) 3.2) ; 4.0
((funcompose '((lambda (x) (> 1 x)) sqrt)) 4) ; #f
```

Problem 3 [6pt] Define a function `merge`, which takes two sorted lists of numbers, and returns one merged list where all numbers are sorted. Assume elements are sorted in the increasing order. For example,

```
merge '(2 7) '() ; returns (2 7)
merge '(1 1 2) '(1 3 5) ; returns (1 1 1 2 3 5)
merge '(1 1 6 8) '(2 7) ; returns (1 1 2 6 7 8)
```

Problem 4 [6pt] Define a recursive function `findMax` in Scheme to get the largest element from a list of non-negative numbers (the list may contain nested lists). If the list is empty, your implementation should return 0. Here are a couple of examples:

```
(findMax  '(4 5 1))       ; returns 5
(findMax  '(5 1 (3 (4 8)))) ; returns 8
(findMax  '(1 3 (3 3) () 6 (1))) ; returns 6
```

Problem 5 [6pt] We define the *depth* of a value as follows: the depth of a non-list value is 0; the depth of a list value is 1 plus the maximum depth of its elements.

Implement a recursive function `depthOfList` that takes a list l and returns the depth of l. Here are some examples:

```
(depthOfList '()) ; returns 1
(depthOfList '(1 2 (1 (2 3)))) ; returns 3
(depthOfList '(0 (0 ()) ())) ; returns 3
```

Problem 6 [20pt] Implement the following functions in Scheme using `fold-left` and `map`. DO NOT use recursive definition for this problem.

a) (5pt) Define a function `newlength` which implements the `length` function in Scheme (directly using `length` will receive no credits).

b) (5pt) Define a function `addOne`, which takes a list of numbers and returns a list where each number in the input list is increased by one. For example, (`addOne` '(1 2 3 4)) should return $(2\ 3\ 4\ 5)$, and (`addOne` '(2 4 6 8)) should return $(3\ 5\ 7\ 9)$.

c) (5pt) Define a function `lstOR`, which takes a list of Booleans and returns #`f` if and only if all of the Booleans are false. For example, (`lstOR` '(#`t` #`f`)) should return #`t`, and (`lstOR` '(#`f` #`f`)) should return #`f`. For your convenience, (`lstOR` '()) is defined as #`f` (functions `and`, `or` implement logical $\wedge$, $\vee$ in Scheme).

d) (5pt) Define a function `removeDup`, which takes a list of ordered numbers, and returns a list that is identical to the input except that duplicated numbers are removed. For example, (`removeDup` '(1 1 2)) should return (1 2); (`removeDup` '(3 5 5 7 7 7 9)) should return (3 5 7 9).