MP 1 - Basic Scheme Cont.

CS421 Agha - Spring 2015

rev. 1.0

1 Introduction

Assigned: January 30, 2015

Due: 11:59pm, February 6, 2015

Outline: Now that you have a bit of experience with Scheme we will work on some more complex scheme procedures. Content will be from lectures up to and including 1/29. After this assignment you should be fairly comfortable with scheme and future MPs will be more involved. This assignment will account for 6% of your grade.

2 Submission

First check that you have access to the following svn repository.

`https://subversion.ews.illinois.edu/svn/sp15-cs421/NETID`

Replace "NETID" with your actual netid. If you are registered for the course you should have access.
Checkout your svn directory with `svn co` and add your file with `svn add`.
Make sure your file is named `mp1.scm` and commit with the following command:

```
svn commit -m "your commit message" mp1.scm
```

Please check the following link if you need a refresher on how to use svn.

`https://wiki.cites.illinois.edu/wiki/display/cs242fa14/Subversion+Tutorial`

3 Problems

Please have your solutions in the same order as listed here.
Some of these problems may require you to write additional procedures not explicitly listed here.
Include the following at the top of your script.

```
#lang eopl
```

3.1 Recursion

1. Write a procedure `insert` which takes two arguments, a list and an integer `n` and inserts `n` at the end of the list.

```
> (insert '(1 2 3) 4)
(1 2 3 4)
```

2. Write a procedure `insertAtN` which takes 3 arguments, a list `lst`, an integer `n`, and another integer `val`. This procedure will insert `val` at the nth position in `lst` using 0-indexing.

```
> (insertAtN '(1 2 3 4) 2 9)
(1 2 9 3 4)
> (insertAtN '(1 2 3 4) 5 9)
(1 2 3 4 9)
> (insertAtN '(1 2) -2 3)
(3 1 2)
> (insertAtN '() 2 9)
(9)
```

3. Write a procedure `pairSums` which takes in a list of list of integers and returns a list of the sums of each of the lists of integers in the same order. Write this with a helper procedure.

```
> (pairSums '((1 2 3) (2 3 4) (3 4 5)))
(6 9 12)
```

3.2 Higher Order Functions

4. Write a procedure `pairSums2` which does the same thing as `pairSums` however without the use of a helper procedure.

5. Write a procedure `funcMap` which takes a list of procedures and a list of data and returns a list. The result list is each procedure applied to the list of data.

```
>(funcMap (list + - /) (list 1 2 3 4))
((1 2 3 4) (-1 -2 -3 -4) (1 1/2 1/3 1/4))
```

3.3 Data Abstraction

6. Write a binary search tree in terms of its interface. First write `makeSTree` which takes a number and returns a binary tree.

```
>(makeSTree 5)
(5 () ())
```

7. Write a procedure `insertLeaf` which takes a number and a binary search tree and inserts it at the correct leaf.

```
>(insertLeaf 2 (makeSTree 5)
(5 (2 () ()) ())
>(insertLeaf 8 (insertLeaf 2 (makeSTree 5)))
(5 (2 () ()) (8 () ()))
```

8. Write a procedure `existBST` which takes a number and a binary search tree and checks if the tree contains the number. Returns #t if it does, #f otherwise.

```
>(existBST 2 (insertLeaf 2 (insertLeaf 8 (makeSTree 5))))
#t
>(existBST 9 (insertLeaf 2 (insertLeaf 8 (makeSTree 5))))
#f
```

9. Using `define-datatype` write a data type named `stack` which follows the following grammar:

   ```
   Stack ::= empty-stack | (int Stack)
   ```

   define the following procedures for `Stack`:

   `emptystack` which has no arguments and returns an empty-stack.
   `push` which takes 2 arguments, an `int` and a `Stack`.
   `pop` which takes 1 argument, a `Stack` and returns null or the stack without the top element.
   `top` which takes 1 argument, a `Stack` and returns either null or the top element of the stack.