CMSC 331

Homework 5

Due Tuesday 4/24, 11:59PM, through GitHub

This homework will give you practice with the Scheme programming language and thinking about functional languages. All work is to be done on your own and should run on the GL servers. We will be using GitHub classroom to handle submission this semester. To get started, accept this homework at https://classroom.github.com/a/lNGddDMH. This will give you all the datafiles and directory structure you need.

Important information about git on GL: Git on GL is outdated, so whenever you see an instruction like "git clone https://github.com/REPO_NAME", you need to modifiy it by adding your GitHub username, so it reads "git clone https://GITHUB_USERNAME@github.com /REPO_NAME".

In addition, this will by default try to pop up a window to ask for your GitHub password. If you are on terminal, this will obviously not work, so you need to enter one of the following commands so you are prompted for your password on the terminal

```
unset SSH_ASKPASS
unsetenv SSH_ASKPASS
```

Important: please edit the README.md file to include your name and section number in it.

Constructing S-Expressions (20 Points)

One thing you have to master in learning Scheme or Lisp is how to construct s-expressions. In the following table, the first column lists some s-expressions we would like to construct. Complete the rest of the table by giving the simplest possible expression to build the s-expression.

permitted to include a quoted list (like '(b c)) in giving you answer, including a quoted empty list -- use null for that. You are permitted, of course to use quoted atoms.

We've done the first row for you as an example. You should, of course, verify your work using the Scheme interpreter.

	Expression	Using cons	Using list
1.ex	(A)	(cons 'A null)	(list 'A)
1.a	(A B)		
1.b	((A B))		
1.c	(A (B))		
1.d	(((A)) (B))		
1.e	(A (B) ((C)))		

De-construction S-Expressions (20 Points)

Another thing you have to master in learning Scheme and Lisp is how to access elements in an s-expression. In the following table, the first column lists some s-expressions containing a single instance of the atom X. Assume that the variable S is bound to the s-expression in the first column. The second column should be an expression that when evaluated returns the atom X. You are only allowed to use the functions car and cdr and a single instance of the variable s. We've done the first row for you as an example. You should, of course, verify your work using the Scheme interpreter. In doing so, you can assign s a value using the define function, e.g., do `(define s '(x))` when checking your answer for 2.ex.

	S	Expression to return the atom x
2.ex	(x)	(car S)
2.a	((1) x 2)	
2.b	(1 (x) 2)	
2.c	(1 2 (x))	

2.d	(1 (2 x))
2.e	((1) (2) (x) (3))

Basic Functions

3. Volume and Suraface Area of a Cone (10 Points)

Write a function `cone-facts` that takes exactly two parameters, the radius and height of a cone. This function should return the volume and surface area of the cone, as a list. They must be returned in that order.

For reference, the volume of a cone is $\frac{1}{3}\pi r^2 h$ and the surface area can be found using the formula $\pi r^2 + \pi r\sqrt{r^2 + h^2}$

NOTE: Use 3.14 as the value of PI so everyone will get the same results

4. Rotate and Swap (10 points)

Write a function `double-rotate` that takes exactly one argument, a list, and returns the same list with the first two items in the list moved to the back, and their positions swapped. An example is shown below.

```
> (double-rotate `(1 2 3 4 5))
(3 4 5 2 1)
> (double-rotate `(1 2))
(2 1)
> (double-rotate `(40 30 20 10 9 8 7))
(20 10 9 8 7 30 40)
```

Recursive Functions

5. Slice (10 Points)

Write a function named `slice` that takes in exactly 3 arguments, a list, and a start and end index. This function should return the elements of the list starting with the start index is shown below.

```
> (slice `(1 2 3 4) 0 2)
(1 2)
> (slice `(1 2 3 4) 0 1)
(1)
> (slice '(1 2 3 4) 0 4)
(1 2 3 4)
> (slice `(1 2 3 4) 3 4)
(4)
```

6. Remove First(10 Points)

Write a function named `remove-first` that takes exactly two arguments, a list and an object to remove from that list. Your function should return the original list with the **first** instance of the object removed from the list. If the object isn't in the list, the list should be returned as is. You must use recursion to solve this problem, but are permitted to use a helper function like was done in class. An example is shown below.

```
> (remove-first '(10 20 30 40 50 40 30 20 10) 10)
(20 30 40 50 40 30 20 10)
> (remove-first '(10 20 30 40 50 40 30 20 10) 40)
(10 20 30 50 40 30 20 10)
> (remove-first '("A" "B" "C" "d" "e" "F") "d")
("A" "B" "C" "e" "F")
> (remove-first '(10 20 30 40 50 40 30 20 10) 60)
(10 20 30 40 50 40 30 20 10)
> (remove-first (remove-first '(10 20 30 40 50 40 30 20 10) 40) 40)
(10 20 30 50 30 20 10)
```

Map/Apply/Filter

7. Perfect Squares (10 Points)

only the numbers that are perfect squares, which we will define as having square roots which are whole numbers. You may not use recursion to solve this problem, and must use some combination of the three functions mentioned above, although not all need to be used.

Hint: To test if something is a whole number, use the built in Scheme function `integer?`

.
Example output is shown below

```
> (perfect-squares `(1 2 3 4 5 6 7 8 9))
(1 4 9)
> (perfect-squares '(15 16 17 24 25 26 25))
(16 25 25)
> (perfect-squares '(2 3 5 6))
()
```

8. Standard Deviation (10 Points)

Using a combination of `map,` `apply` and/or `filter` write a function named `sd` that takes exactly one argument, a list of numbers. This function will return the standard deviation of the numbers in the list. You may not use recursion to solve this problem, and must use some combination of the three functions mentioned above, although not all need to be used.

As a reminder, the formula for standard devaition is:

$$\sqrt{\frac{\sum_{i=1}^{N}(x_i - \overline{x})^2}{N-1}}$$

You may wish to implement a seperate function to find the average of the list

Example Output:

```
> (sd `(1 2 3 4))
1.2909944487358056
```

```
> (sd `(1 2 3 4 5 6 7))
2.160246899469287
```

Running your Code

To load a file of Scheme code the function is **load**. After calling this with your file, you can test your functions. For example to test your answer to question 4:

```
$mzscheme
Welcome to MzScheme v4.1.1 [3m], Copyright (c) 2004-2008 PLT Scheme Inc.
> (load "hw5.scm")
> (double-rotate `(1 2 3 4 5))
(3 4 5 2 1)
```

How You Will be Graded

For the first problem, each answer will be worth 2 points, with 1 point being given for solutions with one or two small mistakes.

For the second problem, each answer will be worth 4 points, with 2 points being given for solutions with one or two small mistakes

For problems 3 - 8, the functions will be tested with a variety of input, and incorrect output will result in deductions between 0.5 and 1 points. It is very important that your functions return lists or numbers as instructed, rather than printing them to the screen. The exception to this gradings scheme are question 3, in which credit may be awarded separately for volume and surface area, and question 4, where partial credit is available for just rotating the lists but not swapping them.

Your code will be run and graded using automated scripts for the most part. It is very importat that you do not name your file anything other than hw4.lua, and that all function names stay as they were given. If you have some incomplete code in your file, it is reccomended that you comment it out, so that it does not prevent successful testing of the working functions in your file.

Your homework should be done on the GL system, as that is where it will be tested. The answers to the first two questions should be in a file named hw5.txt. All your functions should be defined in a single file, named hw5.scm. Please make sure your code runs without any syntax error when loaded into scheme using `(load "hw5.scm")` .

Commiting your code to your local repository and pushing it up to GitHub is your submission for this project. Your last submission before the due date will be taken as your submission.

Please test out your ability to commit and push your code to GitHub before the due date. If you have any problems submitting, email me or stop by office hours.