Homework 1: Scheming

COEN166/266, Spring 2016

Description

For this assignment, implement the following functions in scheme, using only constructs described in the "Scheme Quick Reference" document. These exercises are intended to get you comfortable with the Scheme development environment, the language constructs, list-based data types, and the use of recursion as an iteration construct.

For this assignment, no error checking is necessary: you can assume that all inputs are valid.

`find-biggest`

Takes a list comprised of numerical values and returns the largest one. For example:

```
guile> (find-biggest '(1 2 3 4))
4
guile> (find-biggest '(4 3 2 1))
4
guile> (find-biggest '(1 9 14 9 12 11))
14
```

`count-from`

Takes two integer parameters: a starting and an ending value. Prints all integers between those two number (inclusive). For example:

```
guile> (count-from 1 2)
1
2
guile> (count-from 14 21)
14
15
16
17
18
19
20
21
guile> (count-from 101010 101020)
101010
101011
101012
101013
101014
101015
101016
101017
101018
101019
101020
```

nth-item
Takes two parameters: an index (position) and a list. Returns the item in the list corresponding to the index specified. For example:

```
guile> (nth-item 4 '(1 2 3 4))
4
guile> (nth-item 1 '((1 2) (3 4)))
(1 2)
guile> (nth-item 2 '(a b c d e f))
b
```

replace-nth-item
Takes three parameters: an index (position), a list, and a value to substitute into the list. Returns the list, with the item at the specified index replaced with the new value. For example:

```
guile> (replace-nth-item 4 '(1 2 3 4) 14)
(1 2 3 14)
guile> (replace-nth-item 2 '(a b c d e) '(1 2 3))
(a (1 2 3) c d e)
guile> (replace-nth-item 1 '(((1 2 3) (4 5 6)) b c d) 'a)
(a b c d)
```

sorted?
Takes a single parameter: a list. Returns a boolean value indicating if that list is in a (numerically) sorted order. For example:

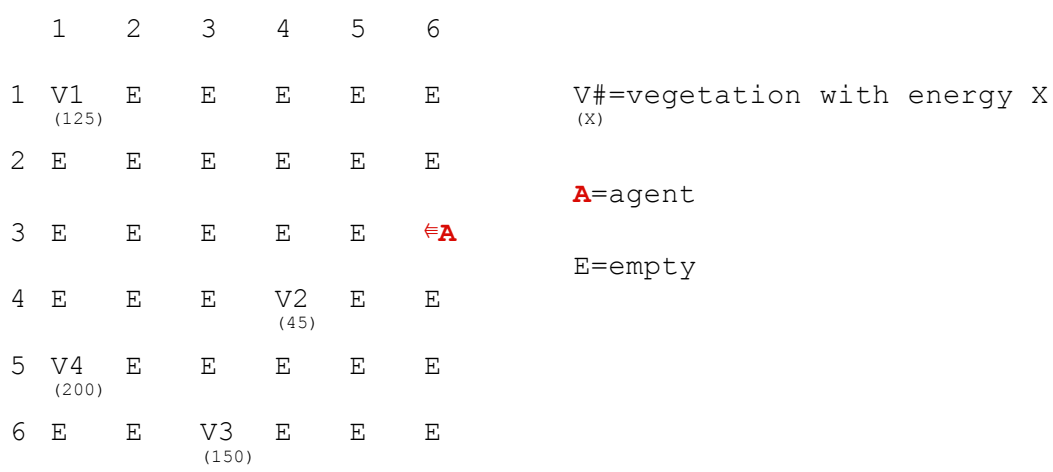
```
guile> (sorted? '(1 2 3 4))
#t
guile> (sorted? '(1 64 128))
#t
guile> (sorted? '(1 15 12 19 3))
#f
```

apply-action
Takes two parameters: an action and a state. Returns the updated state that will result by following that action. A state is a list of three elements: an X position, a Y position, and a direction (N, S, E, or W). An action is one of the following strings: **STAY**, **MOVE-1**, **MOVE-2**, **MOVE-3**, **TURN-LEFT**, **TURN-RIGHT**, or **TURN-AROUND**. Assume that all moves are forward, relative to the current direction. For example:

```
guile> (apply-action '(5 2 N) "MOVE-2")
(5 4 N)
guile> (apply-action '(0 1 W) "MOVE-1")
(-1 1 W)
guile> (apply-action '(14 -4 E) "STAY")
(14 -4 E)
guile> (apply-action '(0 0 S) "TURN-RIGHT")
(0 0 W)
guile (apply-action '(1 2 E) "TURN-AROUND")
(1 2 W)
```

get-location

Takes three parameters: a percept (a la AgentWorld), an X coordinate, and a Y coordinate. Assuming that the agent is in location 0,0 and is facing north, returns the element in the specified location. For example, consider an agent in a small environment like this:



For this map, the agent's percept will be:

Which corresponds to:

```
((empty empty empty)
 (empty (vegetation 2 45) empty empty empty)
 ((vegetation 3 150) empty empty empty empty empty barrier)
 (barrier empty empty empty empty empty empty barrier barrier)
 (barrier barrier empty (vegetation 4 200) empty empty empty
  (vegetation 1 125) barrier barrier barrier))
```

So, assuming that "percept" represents the value above, we would expect to see:

```
guile> (get-location percept 0 1)
empty
guile> (get-location percept -1 2)
(vegetation 2 45)
guile> (get-location percept 3 5)
barrier
```

Extra Credit

For additional credit implement the following functions:

`merge-ordered-lists`

Takes two lists of integers, each assumed to be in ascending order. Returns a single list comprised of the values from both lists in ascending order. For exaample:

```
guile> (merge-ordered-lists '(1 6 14 23) '(5 5 6))
(1 5 5 6 6 14 23)
```

`merge-sort`

Takes one parameter, a list of integer values. Returns a list with the same values, sorted in increasing numerical order. Merge sort will operate by splitting the list in half (+/- 1 element), sorting each of those lists (recursively, using merge-sort), and then merging the two sorted lists. For example:

```
guile> (merge-sort '(93 24 13 55 63 13))
(13 13 24 55 63 93)
```

Due Date

Monday, April 18th

Grading

If you want to get full credit on this assignment, be sure to:

- Follow the API provided exactly.
- Do not use any constructs which do not appear on the Scheme Quick Reference.
- Distinguish between “return” and “print”. A return value is the result of calling the function, whereas something that is printed or displayed is only shown as a side effect.
- Try the examples provided to make sure they work.
- Test your code with other examples that you write (these don’t need to be turned in).
- Make sure your code works on the design center workstations.
- Comment your code.
- Indent your code for readability.