Project 0

0. Abstract:

Scheme is a functional langauage that can be applied to a number of tasks, but the breadth and capability of the language is difficult to gauge without some demonstrative example. In order to see how compelling it is to manipulate functions as first-class elements of a language, you'll complete a Scheme text-based drawing library. Given Scheme's somewhat alien nature it can be useful to get direct visual feedback to demonstrate how what you write is behaving.

1. Introduction:

The drawing library skeleton you have attached prints characters to the interaction window using “figure”s. A “figure” is a Scheme list in the following format:

(function maxrows maxcolumns)

function - a function on two parameters, a row and column coordinate

- the function resolves to '*' if the coordinates are in the figure's area

- the function resolves to ' ' if the coordiantes are not in the figure's area

maxrows - maximal number of rows needed to draw all of this figure.

maxcolumns - maximal number of columens needed to draw all of this figure.

You have a few basic functions that allow you to manipulate figures:

make-figure:

takes a figure function, row and column value and after a bounds check, either will return a default 'out of bounds' character, or it will evaluate the figure function at the row and col values given.

range-check:

takes a row, maxrow, column and maxcolumn value and results in a boolean that indicates if the row and column values given lie within the max values given.

display-window:

takes a figure, an initial and max row and an initial and max column, and evaluates the figure given over the range stipulated.

repeat-cols:

takes a figure and a scalar and constucts a new figure that will consist of the figure given with its columns repeated the given number of times.

Besides a few other internal functions that are used to support those above, these are all the functions implemented in the library. Your task is to implement the functions defined but not coded.

2. Methodology:

See the attached file, drawing_stars.scm. It contains the functions detailed above and a number of functions declared that you need to fill in. A figure is represented by a list (as described above). The first element of that list is a function. If you call that function with 2 arguments; a row number and a column number, it will return the character at that row and column of the figure. If the figure looks like:

*
**

.. and you call the function with arguments 0 and 1 it would return the character #*, the character at row 0, column 1. Note that a function is serving as part of a data structure representing a figure.

Your assignment is to fill in the code as indicated by the comments. You may write additional functions if you like, but you may not change the existing code other than by filling in as indicated and defining additional functions. Additional functions must have comments similar in style to the comments in the existing functions. You must fill in at each place indicated in such a way that the functions all work as specified.

3. Results:

Here are some examples of what will be printed with a correct implementation. These are to aid your understanding – they are not sufficient test cases for your code. Note that only defining a figure does not cause anything to be printed.

```
> (define ca (charfig #\a))

> (define cb (charfig #\b))

> (define ab (append-cols ca cb))

> (define cde (append-cols (charfig #\c)
                           (append-cols (charfig #\d)
                                        (charfig #\e))))

> (define abcd (append-rows  ab cde))

> (display-window 0 0 0 0 ca)
a

> (display-window 0 0 0 1 ca)
a.

> (display-window 0 1 0 1 ca)
a.
..

> (display-window 0 1 0 1 ab)
ab
..

> (display-window 0 2 0 2 abcd)
; note that cde gets truncated to 2 columns
; because ab has only 2 columns
ab.
cd.
...
```

```
> (display-window 0 1 0 2 cde)
cde
...


> (display-window 0 3 0 3 (sw-corner 4))
*
**
***
****

> (display-window 0 3 0 3 (flip-rows(sw-corner 4)))
****
***
**
*

> (display-window 0 3 0 3 (flip-cols(sw-corner 4)))
   *
  **
 ***
****

> (let ((f1 (append-rows ab (flip-cols ab))))
   (display-window 0 2 0 4 (append-cols f1 (flip-rows f1))))
abba.
baab.
.....
```

4. Evaluation:

Your code must operate in DrRacket in the R5RS Scheme language.

Keep in mind you are allowed unlimited resubmissions. We will grade your latest one only. Feel free to and please do use Sakai as a one-way 'git'. If you do use 'git', make sure your repository is private (yes, free student repositories on github can be made private).

Do not change the file name from drawing_stars.scm on submission.

Your code will be evaluated based on:

how well it performs on an exhaustive set of tests

to a lesser degree, your code quality and commenting