CS 314, Fall 2017, Steinberg sec. 5, 6

Project 1 – patterns (Scheme)

Please note:

- This assignment is due at 2 **AM**, Friday, October 20. Note that the due date is very early Friday morning. (Some people may consider it late Thursday night.) Late assignments will not be accepted.
- You can submit as many times as you want, but only one version will be graded: the last version that is not late.
- This project is graded on a 100 point scale based on how much of your code works correctly.
- As with all assignments, the CS Department intellectual honesty rules apply.
- Your code must work in the Racket version of R5RS scheme. If you use any other implementation, you must talk to Prof. Steinberg first.

See the attached file, patterns-handout.scm. It defines functions for

- defining and
- printing

pretty patterns of characters on the screen. A pattern is represented by a list as described in patterns-handout.scm. 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 pattern. E.g. if the pattern looks like

```
ab
cd
```

and you call the function with arguments 0 and 1 it would return the character #\b. (Row 0, column 1. Row and column numbers start at 0.)

Note how a function (actually, a closure) is serving as part of a data structure representing a pattern.

Your assignment is to fill in the code as indicated by the comments. You may also add additional functions, 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 on existing functions. You must fill in at each place indicated in such a way that the functions all work as specified.

Turn in your version of the file patterns-handout.scm with everything filled in.

Here are some examples. Note that defining a pattern does not cause anything to be printed.

```
> (define pa (charpat #\a))

> (define pb (charpat #\b))

> (define ab (append-cols pa pb))

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

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

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

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

> (display-window 0 1 0 1 pa)
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 ((p1 (append-rows ab (flip-cols ab))))
   (display-window 0 2 0 4 (append-cols p1 (flip-rows p1))))
abba.
baab.
.....
```