Concepts of Programming Languages, CSCI 305, Fall 2021
Lab 7, Reading a csv file in Scheme, Nov. 5

1. Copy the following code into Scheme and figure out the scope of the variables: “inPort”, “outPort” and “line”. Modify the code, adding newlines and comments, to clarify the scope of these variables.

```
#lang racket
(define readWriteFiles
  (lambda ()
    (begin
      (display "Getting started")
      (newline)
      (let
          ((inPort (open-input-file "lab7_test.csv"))
           (outPort (open-output-file "lab7_output.txt" #:exists 'replace)))
          (begin
            (display "Ports are open.")
            (newline)
            (let
              ((line (read-line inPort)))
              (begin
                (display "Line of text to place into output file: ")
                (display line)
                (display line outPort)))
            (close-input-port inPort)
            (close-output-port outPort)
            (display "Files are released")
            (newline))))))

(readWriteFiles)
```

2. Place the file *lab7_test.csv* into the same directory as the Scheme code above, run the code and see that the new file was created.

3. Notice that the code only reads one line from *lab7_test.csv*. Modify the code so that it reads all the lines from *lab7_test.csv* and writes them to the output file. A good way to do this is to define a recursive function ***readAll***, which takes opened input and output ports, and reads from the input port and writes to the output port. This function could be called as follows:

```
#lang racket
(define readWriteFiles
  (lambda ()
    (begin
      (display "Getting started")
      (newline)
      (let
          ((inPort (open-input-file "lab7_test.csv"))
          (outPort (open-output-file "lab7_out.txt" #:exists 'replace))
         )
         (begin
           (display "Ports are open.")
           (newline)
           (readAll inPort outPort)   ; <------- helper function readAll
           (close-input-port inPort)
           (close-output-port outPort)
           (display "Files are released")
           (newline)
          )
       )  ;let
     )  ;begin
   )
)
```

Give the definition of the helper function readAll.

4. Install the package csv-reading as follows:
   Within DrRacket, under the File Menu (upper left corner)
   File\Package Manager
   Select the "Available from catalog" tab of the Package Manager

   Locate the package **csv-reading** written by neil@neilvandyke.org
   (can filter by neil@neilvandyke.org)
   Install csv-reading

   More information on this package:
   https://docs.racket-lang.org/csv-reading/index.html

5. Place the file *scanningTable.csv* into the same directory as the Scheme code and experiment with the following code. #lang racket

```
(require csv-reading)   ; Package that contains utilities for reading csv files.

; Build a reader that can read a csv file containing the scan table.
; Use the default parameters.
(define scanTable-csv-reader
  (make-csv-reader-maker '())
)

; Function that uses the csv reader to open a csv file once, and allow
; repeatedly reading the next row, returning the row contents as a list.
(define next-row
    (scanTable-csv-reader (open-input-file "scanningTable.csv"))
)

; Function that reads and displays each row of the csv file as a list.
(define display-contents
  (lambda ()
    (let
       ((line (next-row)))
       (unless (null? line)
         (begin
           (display "Next line: ")
           (display line)
           (newline)
           (display-contents)
           )))))

(display-contents)
```

6. Consider helper functions that will be useful for your lexical analyzer and begin writing code for those.