Assignment:  8
Due:   Tuesday, March 22, 2022 9:00pm
Language level:  Intermediate Student
Allowed recursion:  Simple recursion and Accumulative recursion
Files to submit:  chart.rkt, funabst.rkt, bonus.rkt, acadinteg-a08.txt
Warmup exercises:  HtDP 17.6.4, 19.1.5, 20.1.1, 20.1.2, 24.0.7, 24.0.8
Practice exercises:  HtDP 17.6.5, 17.6.6, 19.1.6, 20.1.3, 21.2.3, 24.0.9

• Unless stated otherwise, all policies from Assignment 07 carry forward.
• This assignment covers material up to the end of Module 15. Check Module 14 – Slide 35 and the style guide regarding the design recipe components for local helper functions.
• All helper functions must be **local** definitions unless otherwise specified. You may use earlier question parts as helpers without redefining them locally.
• You are not required to provide examples or tests for **local** function definitions. You are still encouraged to do informal testing of your helper functions outside of your main function to ensure they are working properly. Functions defined at the “top level” must include the complete design recipe, except for helper functions, which do not need tests.
• You may define global constants if they are used for more than one part of a question. This includes defining constants for examples and tests. Constants that are only used by one top level function should be included in the **local** definitions.
• You should include local definitions to avoid repetition of common subexpressions, to improve readability of expressions and to improve efficiency of code.
• You may only use the list functions that have been discussed in the notes up to the end of Module 14, unless explicitly allowed in the question.
• Download the file acadinteg-a08.txt from the assignment page. Fill in your Quest userid in the provided place to indicate that you have read and understood the policy. We take this to be equivalent to signing the document, so save a copy for your own records.
• Solutions will be marked for both correctness **[85%]** and style **[15%]**. Follow the guidelines in the Style Guide.
Here are the assignment questions you need to submit.

1. **[5% Correctness]** In this question you will perform step-by-step evaluations of Racket programs, as you did in assignment one. Please review the instructions on stepping in A01. To begin, visit this web page:

   https://www.student.cs.uwaterloo.ca/~cs135/stepping

   **Note:** the use of https is important; that is, the system will not work if you omit the s. This link is also under the Assignments menu on the course web page.
   
   When you are ready, complete the required questions under "Module 14: Locals" (5 questions) using the semantics given in class.

2. **[10+10+10+10+10 = 50% Correctness]** Tabular data is a common way to represent individual data that has common characteristics. For example, all students in this course have a variety of marks (for each assignment, for each midterm, etc.), but all students have the same number of marks. We can store information for one individual in one row, where each column represents the particular piece of data we wish to store. For example, the table

   \[
   \begin{pmatrix}
   8 & 3 & 4 & 9 \\
   3 & 7 & 5 & 6 \\
   -1 & 1 & -3 & 0
   \end{pmatrix}
   \]

   can be one way to represent four pieces of information for three different people. We can reference particular elements by their row and column position: for example, the number 1 is in row 2, column 1. That is, rows and columns are indexed starting from 0, as was discussed in the mult-table example in Module 08.
   
   We will represent this tabular data in Racket by way of a list of lists of numbers. For example, the above tabular data can be represented as:

   (list (list 8 3 4 9) (list 3 7 5 6) (list -1 1 -3 0)).

   The data definition below will be helpful in your contracts and when writing your functions:

   ```racket
   ;; A Table is a (listof (listof Num))
   ;; requires: each sub-list has the same length
   ```

   Your solutions to the following functions should not use higher order functions.

   **NOTE:** You **may not** use the built-in reverse function in this question.

   (a) Write the function **mult-by** which consumes a number and a table, and produces the table resulting from each number in the original table being multiplied by the consumed number.
(b) Write the function get-elem which consumes a row (as a natural number), a column (as a natural number) and a table, and produces the number which is in that row and column in the table. If the table is not large enough in either dimension, this function should produce false. There is a built-in function, called list-ref, which you are not permitted to use. You may write your own version of list-ref if you wish.

(c) Write a function mirror which consumes a table and reverses the elements of each row. Note that, you may not use reverse for this function, but you are allowed to implement your own version as an encapsulated helper function.

For example, \( \text{mirror } \left(\begin{array}{c} -3.2 & 4.5 & 7 \\ 13 & 3 & -3 \end{array}\right) \) produces \( \left(\begin{array}{c} 7 & 4.5 & -3.2 \\ 3 & 3 & 13 \end{array}\right) \).

(d) Write a function element-apply-many which consumes a list of functions (each with contract \( \text{Num} \rightarrow \text{Num} \), \( \text{Num} \rightarrow \text{Int} \), or \( \text{Num} \rightarrow \text{Nat} \)) and a table, and produces a list of tables. The first table should be the results of applying the first function to each table element, the second table should be the results of applying the second function to each table element, and so on. For example:

\[
(\text{define } (add3 x) (+ x 3))
\]

\[
(\text{element-apply-many } (\text{list abs floor add3})
  \left(\begin{array}{c} 7 & 4.5 & -3.2 \\ 3 & 3 & 13 \end{array}\right))
\]

produces

\[
(\text{list } (\text{list } 7 4.5 3.2)
  (\text{list } 3 3 13))
\]

\[
(\text{list } (\text{list } 7 4 -4)
  (\text{list } -3 3 13))
\]

\[
(\text{list } (\text{list } 10 7.5 -0.2)
  (\text{list } 0 6 16))
\]

(e) Write a function scale-smallest which consumes a non-empty table (with at least one column and one row) and a real number (the offset). This function produces a second function that consumes a number, multiplies that number by the smallest element of the table, and adds the offset.

To test this function, you may define an additional helper function in the global scope as follows:

\[
;; (\text{apply-function } f \text{ arg}) \text{ produces the result of } f \text{ with the given argument arg.}
;; \text{apply-function: } (X \rightarrow Y) X \rightarrow Y
(\text{define } (\text{apply-function } f \text{ arg})
  (f \text{ arg}))
\]
Then \((\text{apply-function } (\text{scale-smallest} \quad '((7\ 4.5\ 3.2)(-3\ 3\ 13))\ 2.4)\ 7))\) produces -18.6 because \(-3 \cdot 7 + 2.4 = -18.6\). Similarly, \((\text{apply-function } (\text{scale-smallest} \quad '((7\ 4.5\ 3.2)(-3\ 3\ 13))\ 2.4)\ -2.7)\) produces 10.5.

Be careful to avoid exponential blowups in your function implementation.

Place your solutions in \text{chart.rkt}.

3. [10+10+10= 30% Correctness] This question deals with functional abstraction (except part c). Place your solution in the file \text{funabst.rkt}.

(a) Write a function \(\text{and-pred}\) that consumes a predicate (boolean function which takes one argument) and a list, and produces \textit{false} if the application of the consumed predicate on any element of the consumed list produces \textit{false}, otherwise the function produces \textit{true}. For \textit{empty} consumed list the function should produce \textit{true}. For example:

\[
\begin{align*}
(\text{and-pred even? empty}) &= \text{true} \\
(\text{and-pred odd? (list 5 9 3)}) &= \text{true} \\
(\text{and-pred string? (list 5 "wow"}) &= \text{false}
\end{align*}
\]

Use \text{my-filter} from M15 as inspiration.

(b) In class, we have seen that we are now able to put functions into lists. What can we do with lists of functions? One thing is to apply each function in the list to a common set of inputs. Write a function \(\text{map-2-arg-fn}\) which consumes a list of functions (each of which takes two numbers as arguments) and a list containing two numbers. It should produce the list of the results of applying each function in turn to the given two numbers. For example,

\[
(\text{map-2-arg-fn} \quad (\text{list} + - * / \text{list}) \quad '(3\ 2)) \Rightarrow ' (5\ 1\ 6\ 1.5\ (3\ 2))
\]

Note that the first list being passed to \(\text{map-2-arg-fn}\) has five elements, each of which is a function that can take two numbers as input. The resulting list is also of length five.

Pay close attention to the contract for your function.

You cannot use the built-in \text{apply} function.

You will probably discover you need to use \text{local} to give a function a name at one point. However, \textbf{do not use either global or local helper functions} in this question.

(c) Write a predicate function \(\text{arranged}\) that consumes a (list predicate-function binary-relational-operator) pair and a list of values (operands). The predicate function in the first list is to determine the data type of the values in the second list and the binary relational operator consumes the same data type (see contract below). Note that the first list is of length 2 and the second list may be empty.

Here is a contract for \(\text{arranged}\) which you may type into your solutions.
The predicate arranged?

- produces true if the list of operands is empty or has one value and applying the type-checking predicate on it produces true.

  \[(\text{arranged?} (\text{list integer?} <) (\text{list})) \Rightarrow \text{true}\]
  \[(\text{arranged?} (\text{list integer?} >) (\text{list} 1)) \Rightarrow \text{true}\]
  \[(\text{arranged?} (\text{list integer?} >)(\text{list 'red})) \Rightarrow \text{false}\]

- produces false if applying the predicate on any of the operands produces false

  \[(\text{arranged?} (\text{list string? string>?}) (\text{list "wow" 'red})) \Rightarrow \text{false}\]

- produces true if applying the predicate on every operand produces true and applying the binary relational operator on all consecutive elements of the list of operands produces true, otherwise the function produces false.

  \[(\text{arranged?} (\text{list string? string>?})(\text{list "wow" "cs135" "amazing"})) \Rightarrow \text{true}\]

This concludes the list of questions for which you need to submit solutions. Don’t forget to always check your email for the public test results after making a submission.

4. **Bonus:** Write a function keyword-match that consumes a criterion and a keyword, and produces a predicate that consumes a movie and produces true if the movie matches the keyword in that criterion and false otherwise.

\[
\text{keyword-match: (anyof 'name 'director 'year 'genre 'imdb-score 'my-score)}
\]
\[
\quad (\text{anyof Str Sym Nat Num}) \to \text{Bool}
\]

Then write the function search from A06 again, this time using a filter and keyword-match.

**Do not** encapsulate keyword-match, i.e., do not define it as a local helper function, otherwise we cannot test it and you won’t get the marks for it.

You don’t need to report this: Compare your solution to your solution for A06. How much time did you save on typing, finding and fixing errors, and writing design recipes and tests this time, just by using function abstraction?

Place your solutions in **bonus.rkt**. You can use data definition and tests from **movie-list.rkt** in your solution from A06, paste them into **bonus.rkt**.
Note: You need to get both these functions correct in order to get your bonus marks. As usual, bonus marks are all or nothing.