Assignment: 6  
Due: Tuesday, March 12 2019, 9:00pm  
Language level: Beginning Student with List Abbreviations  
Allowed recursion: Pure Structural, or Structural with Accumulators  
Files to submit: recursion.rkt, matrix.rkt, binary-tree.rkt, and bonus.rkt.  
Warmup exercises: HtDP 13.0.3, 13.0.4, 13.0.7, 13.0.8, 14.2.1, 14.2.2, 17.2.1, 17.3.1, 17.6.4, 17.8.3  
Practice exercises: HtDP 12.4.1, 12.4.2, 13.0.5, 13.0.6, 14.2.3, 14.2.4, 14.2.6, 17.1.2, 17.2.2, 17.6.2, 17.8.4, 17.8.8  

- Make sure you read the OFFICIAL A06 post on Piazza for the answers to frequently asked questions.  
- Unless stated otherwise, all policies from Assignment 05 carry forward.  
- Unless otherwise stated, if $X$ is a known type then you may assume that (listof $X$) is also a known type.  
- You may only use the functions that have been discussed in the lecture slides, unless explicitly allowed or disallowed in the question. In particular, you may NOT use any of the following Racket functions: reverse, make-string, replicate, member, member? or list-ref (except for when they are explicitly allowed). You may define your own versions of these functions as long as they are written using structural recursion.  
- If your function is a wrapper function, it will require all design recipe components, but the helper function which it calls will require only the purpose, contract, and definition: no examples or tests are needed for the helper function.  
- It will be very helpful to design examples and tests before writing your function implementations, especially for trees.  
- Solutions will be marked for both correctness [80%] and good style [20%]. Follow the guidelines in the Style Guide.
1. [25%] This question involves recursion.

(a) Write a function, \(\text{fibonacci}\), that consumes a natural number \(n\) and produces the \(n\)-th Fibonacci number using the well-known formula \(F_n = F_{n-1} + F_{n-2}\) with initial values \(F_0 = 0\) and \(F_1 = 1\).

For example:

\[
\begin{align*}
(fibonacci\ 0) & \Rightarrow 0 \\
(fibonacci\ 1) & \Rightarrow 1 \\
(fibonacci\ 2) & \Rightarrow 1 \\
(fibonacci\ 3) & \Rightarrow 2 \\
(fibonacci\ 7) & \Rightarrow 13
\end{align*}
\]

A direct translation yields code that is very slow!

\[
\text{(define (fibonacci-slow n)}
\begin{align*}
\text{(cond} & \text{[(<= n 1) n]} \\
\text{[else (+ (fibonacci-slow (- n 1))} \\
\text{(fibonacci-slow (- n 2)))])}
\end{align*}
\text{))}
\]

Write a function, \(\text{fibonacci}\), that computes the Fibonacci sequence in a much faster way, using accumulative recursion. \(\text{fibonacci}\) cannot be recursive and must use a helper function \(\text{fibonacci-acc}\) that uses accumulative recursion. The body of your \(\text{fibonacci-acc}\) function must have only one recursive application.

As a hint, we have provided a sample function header for \(\text{fibonacci-acc}\) that has four parameters, in addition to a base case:

\[
\text{(define (fibonacci-acc m n fm-1 fm-2)}
\begin{align*}
\text{(cond} & \text{... [(= m n) (+ fm-1 fm-2)]} \\
\text{...})
\end{align*}
\text{))}
\]

(b) Write a function, \(\text{my-list-ref}\) that behaves the same as the built-in function \(\text{list-ref}\). In other words, \(\text{my-list-ref}\) consumes a list and a Natural number \(n\) (where \(n\) is less than the length of the list) and produces the \(n\)-th element of the list (starting at 0). For example,

\[
\text{(check-expect (my-list-ref '(a b c) 1) 'b)}
\]

(c) Write a function, \(\text{string-of-char}\), that consumes a number \(n\) and a character \(c\) and produces a string of \(n\) \(c\)’s. For example:

\[
\text{(check-expect (string-of-char 5 #\z) "zzzzz")}
\]

(d) Write a function, \(\text{replace-vowels}\), that consumes a string and replaces each vowel with a number of x’s: the first vowel is replaced with 1 x, the second with 2 x’s and the \(n\)-th with \(n\) x’s. You may use \(\text{append}\) for this question. A vowel is one of \((a,e,i,o,u)\). Recursion should occur on a list of characters, not on the string.
For example:

(check-expect (replace-vowels "hello world") "hxllxx wxxrld")

Place your solutions in the file recursion.rkt

2. [30%] Matrices are useful tools in mathematics and computer sciences. For the sake of simplicity, consider a matrix to be a 2D grid of elements/numbers where each number is indexed by row and column.

For example, the following 3x2 matrix \( A \) has 3 rows and 2 columns with element \( a_{ij} \) at row \( i \) and column \( j \):

\[
\begin{pmatrix}
 a_{00} & a_{01} \\
 a_{10} & a_{11} \\
 a_{20} & a_{21}
\end{pmatrix}
\]

Using this notation, row 0 is

\[
( a_{00} \ a_{01} )
\]

and column 1 is

\[
\begin{pmatrix}
 a_{01} \\
 a_{11} \\
 a_{21}
\end{pmatrix}
\]

In Racket, we can model a Matrix as a list of rows, i.e., a list of lists, and each row list has the same length (it contains the same quantity of numbers).

For example, the 3x3 matrix:

\[
M = \begin{pmatrix}
 1 & 2 & 3 \\
 4 & 5 & 6 \\
 7 & 8 & 9
\end{pmatrix}
\]

can be represented in Racket as (define M (list (list 1 2 3) (list 4 5 6) (list 7 8 9))).

Similarly, we can create a data definition for a Matrix

```racket
;; A Matrix is a (listof (listof Num))
;; requires: each list of numbers has the same length.
;; all lists are non-empty
```

Note: Although the data definition requires non-empty lists, it is OK if you use empty as the base case in any recursion. (Normally it is not OK if your recursive function violates its own contract, but you’re being given explicit permission to not worry about it).

For each of the following questions, row and column parameters must be valid (that is, if a Matrix has 2 rows, then a row number parameter must be either 0 or 1). Additionally, you may use the list-ref function (you do not need to create your own like you did in Question 1).
(a) Write a function, `matrix-row`, that consumes a matrix (list of lists) and a row number and produces that row of the matrix. For example, `(matrix-row M 0)` produces `(list 1 2 3)`.

(b) Write a function, `matrix-col`, that consumes a matrix and a column number and produces that column of the matrix. For example, `(matrix-col M 1)` produces `(list 2 5 8)`.

(c) Write a function, `get-element`, that consumes a matrix, a row number and a column number, and produces the element at that row and column position. For example, `(get-element M 1 2)` produces 6.

(d) Matrices $A$ and $B$ can be added by the element-wise addition of their elements. For example,

$$
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix} +
\begin{pmatrix}
1 & 2 \\
2 & 1
\end{pmatrix} =
\begin{pmatrix}
2 & 2 \\
2 & 2
\end{pmatrix}
$$

In the matrix $C = A + B$ the elements of $C$ will be defined as $c_{ij} = a_{ij} + b_{ij}$ for all indices $i, j$.

Write a function, `matrix-add`, that consumes two matrices with identical dimensions and produces the result of adding them.

(e) Matrices $A$ and $B$ can also by multiplied together, although the process is a little more involved. For example,

$$
\begin{pmatrix}
1 & 0 \\
1 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 2 & 3 \\
2 & 1 & 1
\end{pmatrix} =
\begin{pmatrix}
1 & 2 & 3 \\
3 & 3 & 4
\end{pmatrix}
$$

Note that unlike with addition, the dimensions of the two matrices do not need to be equal. Instead, the number of columns in $A$ must match the number of rows in $B$. The resulting matrix $C$ will have the same number of rows as $A$, and the same number of columns as $B$.

Put differently (and using more letters), if $A$ is a $p \times n$ matrix, and $B$ is a $n \times q$ matrix, then $C$ will be a $p \times q$ matrix.

Using element notation, the formula for the multiplication $C = AB$ is given by

$$
c_{uv} = \sum_{i=0}^{n-1} a_{ui}b_{iv}
$$

For $0 \leq u < p$ and $0 \leq v < q$.

Write a function, `matrix-multiply`, that consumes two matrices and produces the result of multiplying them.

For more examples and additional explanation, you can read about matrix multiplication here: [https://en.wikipedia.org/wiki/Matrix_multiplication](https://en.wikipedia.org/wiki/Matrix_multiplication)

**Implementation Hint:** You may use the implementation of `dot-product` found in Module 6. (How is this a hint? Look at the sum given for element $c_{uv}$ above. Doesn’t that remind you of a dot-product?)
3. [25%] Binary search trees (BSTs) are a commonly used structure to store data in an ordered fashion. In this assignment question, you will implement some functions that are often required on BSTs (and trees in general). In the context of this question, imagine a BST populated with Nats.

;; A Binary Search Tree (BST) is one of:
;; * empty
;; * a Node

(define-struct node (key left right))
;; A Node is a (make-node Nat BST BST)
;; requires: key > every key in left BST
;; key < every key in right BST

An example of a BST would be:

(define my-bst
  (make-node 5
    (make-node 3 empty empty)
    (make-node 9 (make-node 7 empty empty) empty)))

(a) Write a function bst-count that consumes a BST and produces the total number (as a Nat) of nodes in the BST.

Example:

(check-expect
  (bst-count my-bst)
  4)

(b) Write a function bst-add that consumes a key (as a Nat) and a BST and produces a new BST with a new node added, where the new node contains the consumed key. If the key was already in the tree, then the new BST is identical to the consumed one (there is nowhere it could be added without breaking the data definition).

Example:

(check-expect
  (bst-add 4 my-bst)
  (make-node 5
    (make-node 3 empty (make-node 4 empty empty))
    (make-node 9 (make-node 7 empty empty) empty)))

(c) The height of a tree is defined as the maximum number of nodes between the tree’s root and its leaves. For example, my-bst has a height of 3. Write a function bst-height that consumes a BST and produces the height (as a Nat) of that BST.
Because there is neither a root nor leaves in an empty tree, the empty tree is a special case. We can define the height of an empty tree to be 0.

Example:

```
(define another-bst
  (make-node 5
    (make-node 3
      (make-node 1
        (make-node 0 empty empty)
        (make-node 2 empty empty))
      (make-node 4 empty empty))
    (make-node 9 empty empty)))
```

```
(check-expect
  (bst-height another-bst)
  4)
```

(d) In many algorithms, trees are particularly useful when they are “balanced”. A tree is considered balanced if the height difference between the left and right sub-tree of all nodes is no more than 1. Write a predicate `bst-balanced?` that consumes a BST and produces `true` if it is balanced and `false` otherwise. For the purpose of this assignment, an empty tree is considered to be balanced.

Example:

```
(check-expect
  (bst-balanced? my-bst)
  true)

(check-expect
  (bst-balanced? another-bst)
  false)
```

Place your solutions to this problem in `binary-tree.rkt`.

This concludes the list of questions for which you need to submit solutions. As always, check your email for the basic test results after making a submission.

**Bonus [5%]:** A square matrix is a matrix with the same number of rows and columns.

An identity matrix is a square matrix with ones on the main diagonal and zeros elsewhere. For example I is an identity matrix,

\[
I = \begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}
\]
The transpose of a matrix $M$ is the result of switching $M$'s rows and columns, and is denoted as $M^T$. For example:

$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$

$M^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$

An orthogonal matrix is a square matrix where if you multiply it with its transpose, the result is the identity matrix. Write a function, `orthogonal?`, that consumes a square matrix and produces true if the matrix is orthogonal, otherwise it produces false.

Place your solution in `bonus.rkt`

**Enhancements:** Reminder—enhancements are for your interest and are not to be handed in.

The IEEE-754 standard, most recently updated in 2008, outlines how computers store floating-point numbers. A floating-point number is a number of the form $0.d_1d_2\ldots d_t \times \beta^e$ where $\beta$ is the base, $e$ is the integer exponent, and the mantissa is specified by the $t$ digits $d_i \in \{0,\ldots,\beta - 1\}$. For example, the number 3.14 can be written

$0.314 \times 10^1$

In this example, the base is 10, the exponent is 1, and the digits are 3, 1 and 4. A computer uses the binary number system; the binary equivalent to the above example is approximately

$0.1100100110_2 \times 2^{102}$

where $102$ is the binary integer representing the decimal number 2. Thus, computers represent such numbers by storing the binary digits of the mantissa (“11001001” in the example), and the exponent (“10” in the example). Putting those together, a ten-bit floating-point representation for 3.14 would be “1100100110”.

However, computers use more binary digits for each number, typically 32 bits, or 64 bits. The IEEE-754 standard outlines how these bits are used to specify the mantissa and exponent. The specification includes special bit-patterns that represent Inf, –Inf, and NaN.