Assignment:  6
Due:  Tuesday, November 6 2018, 9:00pm
Language level:  Beginning Student with List Abbreviations
Allowed recursion:  Pure Structural, unless otherwise specified
Files to submit:  binary-trees.rkt, politics.rkt, paint-by-numbers.rkt, 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 pure 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.
- This is a long assignment. If you start too late you will have problems finishing in time.
- It will be very helpful to design examples and tests before writing your function implementations, especially for trees.
1. A “paint by numbers”-puzzle (known as a “nonogram” in mathNEWS) is a puzzle played on a rectangular grid. Each cell of the grid can be coloured black or white. The rows and columns of the grid are labelled with lists of numbers that indicate the sequence of black cells in that row/column.

For example, a row labelled with a sequence of numbers “2 3 2” indicates that this row contains zero or more white squares, followed by 2 adjacent black squares, followed by one or more white squares, followed by exactly 3 black squares, followed by one or more white squares, followed by exactly two black squares, followed by zero or more white squares. The goal of the puzzle is to fill in black squares so that all of the row and column labels are satisfied.

For example, here is a puzzle, unsolved and solved ¹:

There are many websites on the internet that let you solve paint by numbers puzzles online. One good one is http://webpbn.com.

Surprise! In this question you will not be writing a program to solve paint by numbers puzzles. (We’ll tackle that—or something similar—in A09.) Instead, this question will solve a different task: given a grid labelled with black and white squares, you will generate the row and column labels.

We can represent a rectangular grid as a nested list, where white squares are represented by the symbol ‘-’ and black squares by ‘B’:

```scheme
;; A Grid is a (listof (listof (anyof ‘B ’-)))
;; requires: all sublists have the same length
```

For example, the following grid has 5 rows and 12 columns.

```scheme
(define fish-grid
 ’((B B - - B B B B - - - B )
  ( 7 3 2 2))
```

¹These screenshots are from: https://www.chiark.greenend.org.uk/~sgtatham/puzzles/js/pattern.html
(a) Write a function `column` which consumes a natural number `col` and a `Grid`, and produces the `col`th column of the `Grid`, indexed from 0. If the `col`th column does not exist, the function produces `empty`. For example:

(check-expect (column 2 fish-grid) '(- B B - - ))
(check-expect (column 32 fish-grid) empty)

(b) Write a function `cells->tallies` which consumes a `(listof (anyof 'B '-))` and produces a list of tallies of adjacent black cells (think about it as either a row or a column), where each tally is represented by a Nat.

(check-expect (cells->tallies (first fish-grid))
(list 2 4 1))

You may use accumulative recursion for this question. You may also use the `reverse` function.

**Hint**: Accumulate a list of natural numbers for the tallies.

**Hint**: for each recursive case, consider the first symbol. Adjacent to that symbol will be either another symbol, or the empty list. What should be done for each case?

(c) Write a function `puzzle-labels` which consumes a `Grid`, and produces labels for the puzzle. Each label is represented by a list of natural numbers, corresponding to the adjacent black squares in that row or column. The function produces a list of length 2, where each element is itself a list of labels. The first element is a list of row labels (from top to bottom in the puzzle) and the second is a list of column labels (from left to right). For example:

(check-expect (puzzle-labels fish-grid)
(list (list (list 2 4 1)
(list 3 1)
(list 3 2)
(list 1 5)
(list 2 1))
(list (list 1 1)
(list 5)
(list 2)
(list 1 1)
(list 1 1)
(list 1 2)
(list 1 1)
(list 3)
(list 1))

---

CS 135 — Fall 2018 Assignment 6
empty
(list 1 ))

Place your solutions to this problem in paint-by-numbers.rkt.

2. 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 Nat) of nodes in the BST. This will become important for Question 3.

Example:

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

(b) Write a function bst-add that consumes a key (as Nat) and a BST and produces a new BST with a new node added, where the new node contains the key. As before, this will become important for Question 3.

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 distance between the tree’s root and its leaves. For example, my-bst has a height of 2. Write a function bst-height that consumes a BST and produces the height (as Nat) of that BST.
Because there is neither a root nor leaves in an empty tree, we don’t often consider its height. It’s not unreasonable, however, to define the height of an empty tree to be 0 (the same as the height of a tree with only 1 node).

Example:

\[
\text{(define another-bst (make-node 5 (make-node 3 (make-node 1
\text{ (make-node 0 empty empty) (make-node 2 empty empty))
\text{ (make-node 4 empty empty)) (make-node 9 empty empty)}))}
\]

\[
\text{(check-expect (bst-height another-bst) 3)}
\]

(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 \text{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:

\[
\text{(check-expect (bst-balanced? my-bst) true)}
\]
\[
\text{(check-expect (bst-balanced? another-bst) false)}
\]

Place your solutions to this problem in binary-trees.rkt.

3. You recently decided to take your multi-billion dollar tech-company to the next level and start “influencing” important politicians. To compel politicians to act in your company’s favour, you start creating a list of perks that you are willing to give out. Each of them is recorded in its own Perk:

```scheme
;; A Perk is a (list Nat Str)
;; requires: Nat > 0
```

The first element is a \textbf{compliance score} required to receive the perk. The second element is a description of the perk. For example:

\[
\text{(list 100 "100,000 dollars in campaign donations"}]
\]

To increase the incentive you create an individualized list of perks for each important politician. You maintain the list of your perks in an association list:
A Perk-list is either
* empty
* (cons (list Str (listof Perk)) Perk-list)
requires: Perks in (listof Perk) sorted by compliancy score (non-increasing)

The keys (Str) are the politician’s names (which we will assume is unique), and (listof Perk) are the values. The overall Perk-list is not ordered by name, but within each value, the (Perk)s are sorted such that the compliancy scores are in non-increasing order. For example, here is a short Perk-list:

(define short-perklist
  (list
    (list "Peter Smith" (list
      (list 83 "50,000 dollars campaign funding")
      (list 32 "Public support by your company")
      (list 13 "Opera tickets"))
    (list "Jennifer O'Brien" (list
      (list 137 "Position on the Board of Directors")
      (list 22 "Arranging photo-op with CEO"))
    (list "Steve Li" (list
      (list 91 "Sponsored TV ads")
      (list 56 "Invitation as keynote-speaker")
      (list 9 "Positive press release in his favour")
      (list 5 "Business dinner with CTO"))))

(a) Write a function add-perks which consumes a politician’s name, a list of Perk items (sorted in non-increasing order by compliancy score), and a Perk-list. If the politician does not exist in the Perk-list, a new Perk-list is produced consisting of the politician and their perks added to the end of the Perk-list. If the politician does exist, then the resulting Perk-list consists of the politician’s new Perks merged into their existing list of Perks. If a Perk with the same compliancy scores as an existing perk is added to the Perk-list, the new Perks take precedence, i.e., it occurs before the existing one in the new list.

For example:

(check-expect
  (add-perks "Jennifer O'Brien" (list
    (list 30 "Two free flights in company jet")
    (list 3 "Guided company tour"))
  short-perklist)
  (list
    (list "Peter Smith" (list
      (list 83 "50,000 dollars campaign funding")
      (list 32 "Public support by your company")
      (list 13 "Opera tickets"))
    (list "Jennifer O'Brien" (list
      (list 137 "Position on the Board of Directors")
      (list 30 "Two free flights in company jet")))

(list 22 "Arranging photo-op with CEO")
(list 3 "Guided company tour"))
(list "Steve Li" (list
 (list 91 "Sponsored TV ads")
 (list 56 "Invitation as keynote-speaker")
 (list 9 "Positive press release in his favour")
 (list 5 "Business dinner with CTO")))

(b) Write a function `perk-received` which consumes a politician’s name, compliancy score, and `Perk-list`. The function produces either a symbol or a string according to the following rules:

- If politicians have a high enough compliancy score to qualify for a perk in their `Perk-list`, they receive the best perk they qualify for, where “best” means “highest compliancy score requirement”. If there are two or more perks with the same compliancy score requirement, politicians receive the perk listed first.
- If politicians have a non-negative compliancy score but do not qualify for any of their `Perk` items, they receive a wristwatch, indicated by `wristwatch`. This is also the case if a politician has no `Perk` items or is not in the perk-list, but has a non-negative compliancy score.
- If politicians have been refusing your advances and have a negative compliancy score, they will “coincidentally” become the target of a smear-campaign, indicated by the function producing `smear-campaign`.

Here are some examples:

(check-expect (perk-received "Jennifer O’Brien" 25 short-perklist) "Arranging photo-op with CEO")
(check-expect (perk-received "Noton Thelist" 43 short-perklist) ‘wristwatch)
(check-expect (perk-received "Steve Li" 12 short-perklist) "Positive press release in his favour")
(check-expect (perk-received "Peter Smith" -25 short-perklist) ‘smear-campaign)

As rumours of the existence of this list started spreading, politicians started acting according to the interests of your company, hoping to eventually receive one of the rumoured perks. Subsequently, you started maintaining a list of actions that politicians have done in favour or against your company’s interest, so that in return you can return favours accordingly and secure your investments. Each of these actions is recorded in its own `Action`:

;; An Action is a (list Str Int Str)

The first element is a politician’s name (which we will assume is unique). The second element is a compliancy score. A positive score means the action was in favour and a negative score mean the action was against the interests of your company. Every score is either positive
or negative; Action items with a zero-score are not recorded. The third element is a textual description of the action. For example:

\[(\text{list } "\text{Jennifer O'Brien}" \ 10 \ "\text{Voted against a Net Neutrality bill}\")\]

means that Jennifer O’Brien voted against a Net Neutrality bill, which is worth a compliancy score of 10. Note that all strings are case-sensitive, so "Jennifer O'Brien" and "Jennifer O'brien" represent two different people.

A politician’s compliancy score is computed by adding all the compliancy points for all Actions for that politician. For example, consider the following list of Actions:

\[(\text{list } (\text{list } "\text{Peter Smith}" \ -28 \ "\text{Sponsored bill for higher corporate taxes}\")
(\text{list } "\text{Jennifer O'Brien}" \ 30 \ "\text{Pushed major contract for your company}\")
(\text{list } "\text{Jennifer O'Brien}" \ 5 \ "\text{Mentioned your company on morning TV}\")
(\text{list } "\text{Steve Li}" \ 12 \ "\text{Plays golf with your second cousin}\")
(\text{list } "\text{Jennifer O'Brien}" \ -10 \ "\text{Questioned your leadership in public}\")\]

Jennifer O’Brien would have a compliancy score of 25, and Peter Smith would have a compliancy score of −28.

With so many politicians hooked, you decide it’s time to give out some rewards!

(c) You find the number of Actions to be overwhelming and would like to store them in a Binary Search Tree indexed by politicians’ names.

The Binary Search Tree is made of ActionNodes, defined as follows:

\[(\text{define-struct actionnode } \ (\text{name score actions left right}))\]

An ActionNode is a \(\text{(make-actionnode Str Int (listof Action) ActionSearchTree ActionSearchTree)}\)

\(\text{ requires:}\)

\(\text{(string<? x name) is true for every (actionnode-name x) in the left subtree}\)

\(\text{(string>? x name) is true for every (actionnode-name x) in the right subtree}\)

Name is the politician’s name. Score is the compliancy score of this politician, calculated based upon the Actions for that politician in the tree. These Actions are stored in the (unordered) Actions list. New Actions are added to the front of the Actions list. Ultimately, each politician will be represented by no more than one ActionNode.

An ActionSearchTree is defined as follows:

\(\text{requires:\}
\text{An ActionSearchTree is one of:\}
\text{* empty\}
\text{* an ActionNode}\)

Write a function \text{add-action} which consumes an Action and an ActionSearchTree. It produces an ActionSearchTree consisting of the original tree with the new Action
added, either creating an ActionNode or updating an existing ActionNode’s score and action list.

For example:

```
(define short-ast (make-actionnode "Amanda Byers" -5 (list
  (list "Amanda Byers" -5 "Met with competitor"))
empty empty));
```

```
(check-expect (add-action (list "Amanda Byers" -5 "Met with competitor") empty) short-ast)
```

```
(check-expect (add-action (list "Amanda Byers" 7 "Argued on talk radio against raising minimum wage") short-ast)
  (make-actionnode "Amanda Byers" 2 (list
    (list "Amanda Byers" 7 "Argued on talk radio against raising minimum wage")
    (list "Amanda Byers" -5 "Met with competitor"))
  empty empty))
```

(d) You now need to generate a list of politicians and perks. Write a function perk-list which consumes an ActionSearchTree and a Perk-list, and produces a list of pairs. The first element of each pair is the politician’s name, and the second is the perk the politician receives (which might be a string representing the received gift, or one of the symbols 'wristwatch or 'smear-campaign). The result should be in increasing alphabetical order of politician’s name, with all names in the ActionSearchTree represented. (If a name is in the Perk-list but not in the ActionSearchTree it is ignored.)

Here is an example:

```
(check-expect (perk-list
  (make-actionnode "Amanda Byers" -5
    (list
      (list "Amanda Byers" -5 "Met with competitor"))
  empty
  (make-actionnode "Steve Li" 12
    (list
      (list "Steve Li" 12 "Plays golf with your second cousin"))
  empty
  (make-actionnode "Jennifer O'Brien" 25
    (list
      (list "Jennifer O'Brien" 30 "Pushed major contract for your company")
      (list "Jennifer O'Brien" 5 "Mentioned your company on morning TV")
      (list "Jennifer O'Brien" -10 "Questioned your leadership in public"))
  empty
  empty))
```
(list
  (list "Jennifer O'Brien" (list
    (list 137 "Position on the Board of Directors")
    (list 30 "Two free flights in company jet")
    (list 22 "Arranging photo-op with CEO")
    (list 3 "Guided company tour")))
  (list "Steve Li" (list
    (list 91 "Sponsored TV ads")
    (list 56 "Invitation as keynote-speaker")
    (list 9 "Positive press release in his favour")
    (list 5 "Business dinner with CTO")))
  (list "Amanda Byers" (list
    (list "Jennifer O'Brien" "Arranging photo-op with CEO")
    (list "Steve Li" "Positive press release in his favour"))))

You may use the built-in function append in your solution. You may also use accumulative recursion. Place your solutions in politics.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%)**: Reimplement cells->tallies with two restrictions: you must use pure structural recursion with no accumulators, and you must avoid exponential blowups (such as the exponential blowup for max-list on slide 07-04).

Submit your solution in bonus.rkt. As usual, if you use auxiliary sources for this question (including looking up algorithms) you must cite your sources. Not doing so is an academic integrity violation.

**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

\[ 0.11001001_2 \times 2^{10_2} \]
where $10_2$ 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.