CS 2500, Spring 2013
Problem Set 11, Final Homework

Due date: Tuesday, April 9 @ 11:59pm

Programming Language: Intermediate Student Language with Lambda

You must work on and submit this homework with your assigned partner. You will receive no credit for ignoring your partner and working on the homework alone or not working on the homework at all.

Using Blackboard, submit a single Racket file containing all of the code and documentation for this assignment. Place your names and husky email addresses in a comment at the beginning of your file.

Name your file: hw11-lastname1-lastname2.rkt

You must follow the design recipe in your solutions: graders will look for data definitions, contracts, purpose statements, examples/tests, and properly organized function definitions. For the latter, you must follow templates. You do not need to include the templates with your homework, however, unless the question asks for it.

Problem 1:
A number, *n>1*, is prime if it is not divisible by any numbers besides 1 and itself, such as 2, 3, 5 and 7.

a)
Design the function `prime?` which consumes a Natural Number and returns true if it is prime and false otherwise. Use the fact that if *n* is not prime, one if its factors must be less than or equal to the square root of *n*.

b)
Design the function `list-primes` which consumes a Natural Number, *n*, and produces the list of prime numbers up to *n*.

Problem 2:
A palindrome is a word, number or phrase that reads the same forward and backward.

a)
Design the function `make-palindrome`, which consumes a non-empty String and constructs a palindrome by mirroring the String around the last letter. For example, `(make-palindrome "fundies")` should produce "fundieseidnuf".

b)
Design the function `is-palindrome?,` which consumes a non-empty String and determines whether the String is a palindrome or not.

Problem 3:
The Fibonacci numbers, which are the numbers in the following sequence:

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...
```

are defined by the recurrence relation: $F_n = F_{n-1} + F_{n-2}$, with seed values $F_0 = 0$ and $F_1 = 1$.

a)
Design the function `fibonacci`, (without accumulators) which consumes a Natural Number and produces its Fibonacci number. For example, `(fibonacci 11)` should produce 89.

b)
Use accumulators to design a more efficient version of `fibonacci`.

c)
Design a function, `list-fibonacci`, that consumes a Natural Number and produces the list of Fibonacci numbers from F_0 to F_n.

Problem 4:
Here is a data definition for binary trees that contain Strings at every node of the tree, not just the leaves.

```
;; A WordTree is one of:
;; -String
;; -(make-node WordTree String WordTree)

(define-struct node (left word right))
```

Here are two examples of `WordTrees`:

```
(define holidays
   (make-node
      (make-node "Christmas" "Labor Day" "MLK Day")
       "Patriots Day"
      (make-node "Presidents Day" Thanksgiving" "Veterans Day")))

(define movies
   (make-node
       (make-node "Argo" "Flight" "Life of Pi")
        "Lincoln"
        "Skyfall"))
```

Notice that the two example trees are ordered alphabetically. A `WordTree` is ordered if:

- all of the strings in its left child alphabetically precede the node's word,
- all of the strings in its right child alphabetically succeed the node's word,
- and the left and right subtrees are both ordered.

a)
Use structural recursion to design the function `word-in-tree?,` which consumes an ordered `WordTree` and a String and produces true if the String is in the tree and false otherwise.

b)
Use binary search to design the more efficient function `word-in-tree?.bin`, which consumes an ordered `WordTree` and a String and produces true if the String is in the tree and false otherwise.