Homework 3

Due Thursday, 02/11/2016, by 11:55 p.m.

Your programs should be in a single file that can be loaded into ghci and tested. Problem numbers and any discussion of the programs should be set off as comments (using –) to make the Haskell compiler ignore them. Your program file should be turned in on Sakai like last week.

I should be able to load your file into Haskell (ghci) and have it compile without error. If there are programs with compilation errors, please comment them out so they don't get in the way of your other programs.

Please include good comments and use good variable names with your programs. Otherwise it is very difficult to read your code because Haskell is so concise.

Important: Your programs should have the exact names and types specified in the problem statement as your code will be automatically tested. Code that uses the incorrect name or expects different types (i.e., won't correctly execute my tests) will be counted as incorrect. Notice that all the functions to be written are in curried form.

Use comments to explain what each function does, especially helper functions.

1. (a) (5 points) Write a function `freq` that takes a list as an argument and returns a list of pairs such that for each element that occurs in the list is paired with the number of times that element occurs in the list. E.g., `freq [5,3,7,7,3,2,7]` should return `[(5,1),(3,2),(7,3),(2,1)]`.

    ```
    freq :: (Eq t, Num t1) => [t] -> [(t, t1)]
    ```

    The text on page 50 presents a function to determine the number of times an element appears in a list. It is a slight variant on the following function:

    ```
    count :: Eq a => a -> [a] -> Int
    count e [] = 0
    count e (hd:tail) = if (e == hd) then 1 + (count e tail)
                                     else (count e tail)
    ```

    Feel free to use this (or the code in the text) in your code.

    You may also find it helpful to use your program `deleteAll` from last week.

   (b) (5 points) Write a function `wordFreq` that takes a string and returns a list of pairs of words in the string and their frequencies. E.g., `wordFreq "This is a test of this program to see if this gives a correct answer"` should return `[("this",3), ("is",1), ("a",2), ("test",1), ("of",1), ("program",1), ("to",1), ("see",1), ("if",1), ("gives",1), ("correct",1), ("answer",1)]`.

Notice that this function should ignore differences in capitalization in words. You can take care of that by writing a function `lowercase` that converts strings to lowercase. It is most easily written using the standard library function `toLower: Char -> Char`. To use this, your program must `import Data.Char`.

```
wordFreq :: Num t => [Char] -> [(String, t)]
```

2. (5 points) Write a function `removeDups` that takes a list and returns the list with all duplicates removed. Thus `removeDups [1,2,3,2,4,2]` should return `[1,2,3,4]`. *Hint: You might find it helpful to write a helper function that takes an element and a list and returns a new list which is obtained by removing all occurrences of that element from the list.* Use a let clause for the helper function. Please do NOT use the function `nub` from Data.List in your solution.

3. (5 points) Write a function `permutes n` that returns a list of all lists of length n composed of booleans. E.g., `permutes 2` returns `[[False,False], [False,True], [True,False], [True,True]]`

4. (10 points) In class we wrote a program to convert English to Pig Latin (see the on-line lecture notes from lecture 8). For this assignment please write a translator from English to Pirate-speak. This translation is accomplished by simple word substitution using the following set of replacement pairs:

```
("hello", "ahoy");
("hi", "yo-ho-ho");
("pardon", "avast");
("my", "me");
("friend", "bucko");
("sir", "matey");
("madam", "proud beauty");
("miss", "comely wench");
("stranger", "scurvy dog");
("officer", "foul blaggart");
("where", "whar");
("is", "be");
("the", "th'");
("you", "ye");
("tell", "be tellin'");
("know", "be knowin'");
("miles", "leagues");
("old", "barnacle-covered");
("attractive", "comely");
("happy", "grog-filled");
("nearby", "broadside");
```

```
("restroom", "head");
("restaurant", "galley");
("hotel", "fleabag inn");
("pub", "Skull and Scuppers");
("bank", "buried treasure");
```

That is, the translation is accomplished by replacing every word on the left by the corresponding word on the right. Thus the sentence “My friend saw a happy stranger in the hotel restaurant” would get translated to “Me bucko saw a grog-filled scurvy dog in the fleabag inn galley.”

Feel free to use the built in functions `words` and `unwords` and remember that `map` is your friend!!

For five points extra-credit your program should preserve capitalization and all punctuation in the sentence.

5. (9 points)Formulas of propositional logic are defined as follows:

   (a) propositional letters p, q, and r, with and without any number of primes (’) are formulas of propositional logic.

   (b) If F and G are formulas of propositional logic then so are $\neg F$ (“not F”), $F \vee G$ (read “F or G”), and $F \wedge G$ (“F and G”).

   We also use the abbreviation $G \rightarrow G$ (read “F implies G”) as an abbreviation for $\neg F \vee G$.

   Please write the following statements as formulas of propositional logic. Be sure to define propositional variables to stand for the simple statements contained in the more complicated ones (e.g., let p stand for “I will go to school” and let q stand for “I get a cookie now.” In particular, each of the below sentences should be written using one or more logical operations.

   (a) I will only go to school if I get a cookie now.

   (b) John and Mary are running.

   (c) A foreign national is entitled to social security if he has legal employment or if he has had such less than three years ago, unless he is currently also employed abroad.

6. (8 points) Suppose instead of writing the definition of terms of propositional logic as we did in class, we could use the following definition which omits parentheses:

   - atom := p | q | r | atom’
   - F ::= atom | ¬F | F∨F | F∧F

Please show that there are multiple parse trees for the formula $\neg p \vee q$. Use a truth table to show these parses result in different truth values for some assignments of truth values to p and q.

Find multiple parse trees for $p \wedge q \wedge r$. Do you get different truth values this time?

7. (3 points) The following data type definition is given in line 64 of the text's file, `FSynF.hs`, to represent formulas of propositional logic:

```
data Form =  P String | Ng Form | Cnj [Form] | Dsj [Form]
            deriving Eq
```

Thus the proposition letter "q" can be represented as `P ‘‘q’’`, while "$p \vee q$" can be represented as `Dsj[P ‘‘p’’,P ‘‘q’’]`. In fact, notice that you can represent conjunctions and disjunctions of arbitrarily many terms.

Please write a Haskell term corresponding to the formula $(\neg p) \vee ((\neg q) \wedge r)$.