Homework 2

Due Thursday, 02/04/2016, by 11:59 p.m.

Your programs should be in a single file that can be loaded into sml 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. (5 pts) Suppose we write the time of day as a triple of the hour, minutes after the hour, and either "AM" or "PM", e.g. class starts at (10, 0, "AM") and my office hours start at (2, 0, "PM"). Please write a function `comesBefore` that takes two times of day and returns whether the first comes before the second on a given day. E.g.,

   ```
   comesBefore  (10, 0, "AM") (2, 0, "PM")
   ```

   should return `True`.

   ```
   comesBefore :: (Int, Int, [Char]) -> (Int, Int, [Char]) -> Bool
   ```

   This can be a bit tricky as recall that 12:15 a.m. comes before 1:10 a.m., but pre-processing the hours to make them easier to compare can make your task easier. For one point extra credit write your solution using only boolean operators (not, &&, and ||) and comparison operators. I.e., no if-then-else or similar control structures.

2. (5 points) Write a function `nth` which returns the nth element of a list. E.g., `nth n lst` should return element n of `lst`, where as usual we start counting at 0. Thus `nth 0 [5,4,3] = 5`. (I know that Haskell has a built-in operator `!!` that does exactly this, but I want you to write it from scratch without using that operator.)

```
nth :: (Eq a, Num a) => a -> [t] -> t
```

3. (5 points) Suppose we have a list of pairs of elements. Write a function `keys` that returns the list of all first elements of the pairs (do not remove repetitions). Thus `keys [("hello",2), ("bye",3), ("on",4), ("hello",1)]` should return `["hello", "bye", "on", "hello"]`. *Hint: Consider using the `map` function on lists.*

   ```
   keys :: [(b, b1)] -> [b]
   ```

4. Haskell contains a function `elem` that takes an element and a list and returns whether the element is in the list. E.g. `elem 3 [1,3,5]` returns True.

   Of course we could also write our own version, `elem'`, recursively:

   ```
   elem' e [] = False
   elem' e (hd:tail) = (e == hd) || (elem' e tail)
   ```

   Please write the following variations on this function:

   (a) (5 points) Write a function `indexOf` that returns the index of the first occurrence of an element in a list. E.g., `indexOf elt lst` returns `i` iff `elt` first occurs in `lst` in position `i`, where as usual we start counting from 0. If `elt` does not occur in `lst` return -1. (Be careful to get the right answer for the case where the element doesn't appear. It takes more work than the previous problems.) Please don't use any of the built-in list operators like `findIndex` in writing your solution.

   ```
   indexOf :: (Eq a, Eq a1, Num a1) => a -> [a] -> a1
   ```

   (b) (5 points) Write a more general function `IndexOfList` that takes two lists of elements, `patterns` and `source` and returns the index of the first place in `source` where any of the elements in `patterns` occurs. Thus `indexOfList [1,3,5] [34,7,3,6]` returns 2, because the element 3 from the first list occurs in position 2 in the second list. As before the function should return -1 if no elements are found.

   ```
   indexOfList :: (Eq a, Eq a1, Num a1, Foldable t) =>
                                              t a -> [a] -> a1
   ```

5. The Haskell function `words` breaks a string up into a list of words, each of which was delimited by white space (e.g., spaces, tab, newline, etc.). For example `words "This is a test, isn't it?"` returns
`["This","is","a","test,","isn't","it?"]`.

(a) (5 points) Please write a function `pwords` that improves on this by separating punctuation from words. You may assume that the only punctuation marks are in the list `['.', ',', '?', '!',':', ';']` (which could also have been written as ".,?!:;". It should also break for white space (any of the characters in `" \n\t"`). You may find the function `indexOfList` from the previous exercise helpful here.

Evaluating `pwords "this !!  is a test,!"` should result in `["this"," ","!","!"," ","is"," ","a"," ","test",",","!"]`

```
pwords :: [Char] -> [[Char]]
```

(b) (5 points) Please write a function `glueWords` that puts a list of words back together, inserting a space between consecutive words. In particular, `glueWords (words "This is a test")` should return `"This is a test"` (with no extra spaces at the end),

```
    glueWords :: [[Char]] -> [Char]
```

Do not use the function `unlines` from `Data.String`

6. (5 points) Write a function `deleteAll` that deletes all occurrences of a given element from a list. E.g., `deleteAll elt lst` returns the list of all elements from `lst` except occurrences of `elt`. The order of elements in the list returned should be the same as in the `lst`.

```
deleteAll :: Eq t => t -> [t] -> [t]
```