1) **[4 marks]** On a particular system, a password is considered acceptable if it meets the following constraints:
    - It is at least 8 characters in length
    - It contains at least one uppercase letter ('A' to 'Z')
    - It contains at least one lowercase letter ('a' to 'z')
    - It contains at least one digit ('0'to '9')

    Write a Haskell function named okPassword that takes a string as its only parameter and returns a Boolean result. The result should be true if the password meets all of the constraints outlined previously. Otherwise it should be false. You are welcome to use any combination of recursion, list comprehensions, higher order functions and other Haskell features when solving this problem.

    Test cases to consider:

    okPassword "" should return False.
    okPassword "ABCDwxyz" should return False.
    okPassword "EFGH1234" should return False.
    okPassword "abcd7890" should return False.
    okPassword "Qwerty10" should return True.
    okPassword "cPsC 449" should return True.
    okPassword "AAAAA11111aaaaa" should return True.

2) **[4 marks]** Using a fold (and without using a list comprehension, recursion, or other higher order functions), write a Haskell function named pzn that takes a list of integers as its only parameter. Your function will return a tuple of integers as its only result. The first element in the tuple will be a count of the number of positive integers in the list, the second element in the tuple will be a count of the number of zeros in the list, and the third element in the list will be a count of the number of negative integers in the list.

    Test cases to consider:

    pzn [] should return (0, 0, 0)
    pzn [1, 2, 3] should return (3, 0, 0)
    pzn [-10, 0, 10] should return (1, 1, 1)
    pzn [-100..0] should return (0, 1, 100)
    pzn [0,0,0] should return (0, 3, 0)

    Hints: I used a left fold and wrote a named helper function when I created my solution.

3) **[6 marks]** Using Haskell, write a polymorphic function named closest which takes a numeric value, n, and a sorted list of numeric values as its only parameters. Your function should return the number in the list of sorted values which is closest to n (minimum distance, either positive or negative – if n is equidistant to two different numbers then return one or the other, it doesn't matter which one). Ensure that your function behaves correctly for both integers and floating point numbers. Also ensure that your function works correctly when n is smaller than the first number in values, and when n is larger than the last number in values. Report an error if the list of values provided as a parameter is empty.

   Test cases to consider:

   closest (-6) [-3, -2, -1, 0, 1, 2] should return -3
   closest 0 [-3.0] should return -3.0
   closest 0.2 [1.7, 5.1, 10.3] should return 1.7
   closest pi [2, 5, 10] should return 2.0
   closest (ceiling pi) [2, 5, 10] should return 5
   closest 14.0 [1, 4, 9] should return 9.0
   closest 1 [] should generate an exception with a message indicating that the list was empty

4) **[5 marks]** Consider a Haskell datatype that represents a ternary tree – a tree where each node has 3 children. Every node in the tree (both internal and Leaf) also stores a string. Such a tree can be defined by the following data type:

```
data TernaryTree = Internal String TernaryTree TernaryTree TernaryTree |
                   Leaf String deriving Show
```

   Write a function named countNodes that takes a TernaryTree as its only parameter and returns a pair containing two integers. The first integer in the pair should be the number of internal nodes in the tree. The second integer in the pair should be the number of leaf nodes in the tree.

   Test cases to consider:

   countNodes (Leaf "a") should return (0, 1).
   countNodes (Internal "1" (Leaf "a") (Leaf "b") (Leaf "c")) should return (1, 3).
   countNodes (Internal "1" (Leaf "a") (Internal "2" (Leaf "b1") (Leaf "b2") (Leaf "b3")) (Leaf "c")) should return (2, 5).

   You are strongly encouraged to test your function with more than just these three test cases.

5) **[4 marks]** Write a function named random_list. Your function will take an integer, seed, as its only parameter. It will return an infinite list of pseudo-random integers between 1 and 10 as its only result. Each value in the infinite list will be computed in the following manner:
   - Compute next_seed as 7 * seed `mod` 101
   - Compute the current random number as (next_seed - 1) `mod` 10 + 1
   - Compute the remaining values in the list by recursively calling your function with next_seed as its only parameter

   Test cases to consider:

   take 10 (random_list 1) should return [7,9,10,8,1,5,10,4,7,5]
   take 10 (random_list 2) should return [4,8,10,5,2,9,9,8,3,9]
   take 10 (random_list (-3)) should return [10,5,2,9,9,8,3,9,1,7]
   (random_list 1001) !! 1001 should return 4

6) **[5 marks]** A Prolog knowledgebase that encodes the following facts is available on the course website:

   Bob, Charles, Evan and Gordon are male
   Alice, Diane, Fiona and Harriot are female
   Bob and Evan like pizza.
   Charles likes Alice.
   Evan and Gordon both like Fiona.
   Fiona likes Charles and Evan.
   Charles, Evan and Gordon like golf.
   Alice and Diane like shopping.
   Diane likes Bob.
   Fiona and Harriot like golf.
   Diane and Fiona like steak.

   Without changing the facts that are already in the provided knowledgebase, create a Prolog rule that determines whether or not two people can be friends. For the purposes of this exercise, two people can be friends if:

   - They are both the same gender, and they have at least one like in common that isn't a person of the opposite gender.
   - They are opposite genders, and they have at least one like in common, and one doesn't like the other (and they both don't like each other).

   Note that a person cannot be their own friend.

Test cases to consider:

```
| ?- canBeFriends(charles, harriot).
yes
| ?- canBeFriends(charles, bob).
no
| ?- canBeFriends(bob, bob).
no
| ?- canBeFriends(X, bob).
X = evan ? ;
no
| ?- canBeFriends(evan, X).
X = bob ? ;
X = charles ? ;
X = gordon ? ;
X = harriot
yes
```

Hint: My solution is approximately 20 lines of Prolog code, organized with 1 clause on each line. I wrote three different versions of the rule – one that handled two males being friends, one that handled two females being friends, and one that handled two people of opposite gender being friends.

7) **[5 marks]** In Python, (and perhaps in some other languages too), individual elements in a list can be accessed using both positive and negative integers. When a positive index is used, the index represents an offset from the beginning of the list, with index 0 representing the first element in the list, index 1 representing the second element in the list etc. When a negative integer is used, indexing is performed from the end of the list. Specifically, -1 represents the last element in the list, -2 represents the second last element in the list, etc.

   Create a gProlog rule named elementAt. It's first parameter will be a list. It's second parameter will be an integer (positive or negative). It's third parameter will be an uninstantiated variable. When the rule executes, it should populate the uninstantiated variable with the appropriate element from the list, using positive or negative indexing as described in the previous paragraph. Your rule should fail (reporting no, **not** with an uncaught exception) if the index is too small or too large. Note that gProlog includes a built-in predicate named nth. You are **not** permitted to make use of that predicate when completing this problem.

Test cases to consider:

elementAt([4, 8, 16], -4, X) should report no.
elementAt([4, 8, 16], -2, X) should instantiate X to 8.
elementAt([4, 8, 16], 2, X) should instantiate X to 16.
elementAt([72,101,108,108,111,44,32,87,111,114,108,100,33], -1, X) should instantiate X to 33.
elementAt([72,101,108,108,111,44,32,87,111,114,108,100,33], 0, X) should instantiate X to 72.
elementAt([72,101,108,108,111,44,32,87,111,114,108,100,33], 6, X) should instantiate X to 32.
elementAt([72,101,108,108,111,44,32,87,111,114,108,100,33], 99, X) should report no.

Hint: My solution is less than 15 lines of Prolog code, organized with 1 clause on each line.