CS 357: Declarative Programming
Extra Practice (Spring '19)

1. Define a function *takeWhile* which takes a predicate and a list as arguments and returns the prefix of the list satisfying the predicate. For example,

   ```
   *Main> takeWhile (/= ' ') "This is practice."
   "This"
   ```

2. Define a function *span* which takes a predicate and a list as arguments and returns a pair of lists where the first element of the pair is the portion of the list which the function *takeWhile* would return and the second element is the remainder of the list. For example,

   ```
   *Main> span (/= ' ') "This is practice."
   ("This"," is practice.")
   ```

3. Consider the following data type declaration:

   ```
   data Maybe a = Nothing | Just a
   ```

   Define the function *findKey* (described in class) with type signature

   ```
   findKey :: (Eq a) => a -> [(a,b)] -> Maybe b
   ```

   using a list comprehension.

4. Define a function *splitText* which takes a string of text as its argument and returns a list of words with spaces removed. For example,

   ```
   *Main> splitText (/= ' ') "This is practice."
   ("This","is","practice.")
   ```

5. Without using explicit recursion, define a function *encipher* which takes two lists of equal length and a third list. It uses the first two lists to define a substitution cipher which it uses to encipher the third list. For example,

   ```
   *Main> encipher ['A'..'Z'] ['a'..'z'] "THIS"
   "this"
   ```

6. Consider the following data type declaration:

   ```
   data Maybe a = Nothing | Just a
   ```

   Define the function *findKey* (described in class) with type signature

```
findKey :: (Eq a) => a -> [(a,b)] -> Maybe b
```

using *foldr*.

7. The variance of a list of numbers of length n is the average squared difference between each number and the numbers' mean: $\sum_{i=1}^{n}(x_i - \bar{x})^2/n$ where $\bar{x}$ is the numbers' mean: $\sum_{i=1}^{n} x_i/n$. Without using explicit recursion, give a definition of a function, *variance*, which works as follows:

```
*Main> variance [1..10]
8.25
```

8. Give definitions for the following functions using *foldr*: *product*, *sum*, *or*, *and*, `++`, *!!*, *map*, *filter*, and *concat*.

9. An $n \times n$ matrix can be represented as a length n list of length n lists of numbers. An $n \times n$ identity matrix is zero everywhere except on its diagonal where it is one. Define a function *matrixId* which takes an integer n as its argument and returns an $n \times n$ identity matrix. For example,

```
*Main> matrixId 3
[[1,0,0],[0,1,0],[0,0,1]]
```

10. An $n \times m$ matrix can be represented as a length n list of length m lists of numbers. The function *rho* takes a list of length $n \times m$ and returns a length n list of length m lists. For example,

```
*Main> rho 2 3 [1..6]
[[1,2,3],[4,5,6]]
```

11. The transpose of a $n \times m$ matrix $\mathbf{A}$ is an $m \times n$ matrix $\mathbf{A}^{\mathrm{T}}$ where the (i, j)-th element of $\mathbf{A}^{\mathrm{T}}$ is the (j,i)-th element of $\mathbf{A}$. Define a function *transpose* which takes a matrix represented as a length n list of length m lists of numbers as its argument and returns the transpose represented as a length m list of length n list of numbers. For example,

```
*Main> transpose (rho 2 3 [1..6])
[[1,4],[2,5],[3,6]]
```

12. Define a function *matrixElement* which takes a matrix represented as a list of lists of numbers and two integers i and j as arguments and returns the (i, j)-th element of the matrix. For example,

```
*Main> matrixElement (rho 6 7 [1..]) 5 4
40
```

13. The dot product of two vectors $\vec{u}$ and $\vec{v}$ of length n (written $\vec{u} \cdot \vec{v}$) is defined to be $\sum_{i=1}^{n} u_i v_i$. Without using explicit recursion, define a function *dot* which takes two lists of numbers of equal length and returns their dot product.

```
*Main> [0,0,1] `dot` [0,1,0]
0
```

14. Two matrices **A** and **B** can be multiplied if the number of columns of **A** equals the number of rows of **B**. The (i,j)-th element of the product matrix $\mathbf{C} = \mathbf{AB}$ is defined as follows: $c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$ where m is the number of columns of **A**. Define a function *matrixProduct* which takes two matrices represented as lists of lists of numbers and returns the matrix product represented in the same manner. Hint: c_{ij} is the dot product of the i-th row of **A** and the j-th row of $\mathbf{B}^{\mathrm{T}}$.

```
*Main> matrixProduct (rho 2 3 [1..6]) (rho 3 2 [1..6])
[[22,28],[49,64]]
```

15. Define a function *prefixSum* which takes a list of numbers as its argument and returns a list of sums of all prefixes of the list. For example,

```
*Main> prefixSum [1..10]
[1,3,6,10,15,21,28,36,45,55]
```

16. The function *select* takes a predicate and two lists as arguments and returns a list composed of elements from the second list in those positions where the predicate, when applied to the element in the corresponding positions of the first list, returns *True*.

```
*Main> :t select
select :: (t -> Bool) -> [t] -> [a] -> [a]
*Main> select even [1..26]  "abcdefghijklmnopqrstuvwxyz"
"bdfhjlnprtvxz"
*Main> select  (<= 'g') "abcdefghijklmnopqrstuvwxyz" [1..26]
[1,2,3,4,5,6,7]
```

17. The Goldbach conjecture states that any even number greater than two can be written as the sum of two primes. Using list comprehensions, write a function *Goldbach*, which given an even number n returns a list of pairs of prime numbers which sum to n. Note: You will have to write a function which tests an integer for primality and this should be written as a list comprehension also. For example,

```
*Main> goldbach 6
[(3,3)]
*Main> :t goldbach
:t goldbach
Int -> [(Int,Int)]
*Main>
```

18. To 'decimate' literally means to kill every tenth man (it was a punishment in the Roman legions). Define a function *decimate* which removes every tenth element from a list. for example,

```
*Main> decimate [1..21]
[1,2,3,4,5,6,7,8,9,11,12,13,14,15,16,17,18,19,21]
```

19. Write a function called *smallest* which returns the k smallest numbers from a list of numbers.