CS 357: Declarative Programming
Homework 4

Give definitions for the following functions in Haskell.

1. The function *stutter* takes a list of elements and returns a list where every element has been duplicated. For example,

```
*Main> stutter "Hello World"
"HHeellllloo  WWoorrlldd"
*Main> stutter [1,2,3]
[1,1,2,2,3,3]
```

2. The function *compress* eliminates consecutive duplicate elements of a list. For example,

```
*Main> compress "HHeellllloo  WWoorrlldd"
"Helo World"
*Main> compress [1,2,2,3,3,3]
[1,2,3]
```

3. The function *findIndices* takes a predicate and a list as arguments and returns a list of numbers indicating the positions of elements in the list which satisfy the predicate. For example,

```
*Main> findIndices (< 'a') "AbCdef"
[0,2]
*Main> findIndices (== 0) [1,2,0,3,0]
[2,4]
```

   The function *intersect* takes two lists as arguments and returns a list of elements common to both lists. For example

```
*Main> intersect "abc" "cat"
"ac"
*Main> intersect [1,2,3] [8]
[]
*Main> intersect [3,2,1] [1,2,3]
[3,2,1]
```

4. The function *isPrefixOf* takes two lists as argument and returns *True* iff the first list is a prefix of the second list. For example,

```
*Main> "foo" `isPrefixOf` "foobar"
True
*Main> isPrefixOf [1,2,3] [4,5,6]
False
```

5. The function *isSuffixOf* takes two lists as argument and returns *True* iff the first list is a suffix of the second list. For example,

```
*Main> "bar" `isSuffixOf` "foobar"
True
*Main> isSuffixOf [1,2,3] [4,5,6]
False
```

6. The dot product of two vectors $\vec{u}$ and $\vec{v}$ of length n (written $\vec{u} \cdot \vec{v}$) is defined to be $\sum_{i=1}^{n} u_i v_i$. 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
```

7. The function *increasing* takes a list of enumerable elements as its argument and returns *True* if the list is sorted in increasing order and *False* otherwise.

```
*Main> increasing "ABCD"
True
*Main> increasing [100,99..1]
False
```

   Write *increasing*.

8. 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]
```

9. 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"
```

10. 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]
*Main> prefixSum [2, 5]
[2, 7]
```

11. 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]
```

12. The function *numbers* which takes a list of integers as its argument and returns the integer which has those numbers as digits. For example,

```
*Main> numbers [1..4]
1234
```

Write *numbers* using a tail-recursive helper function defined inside of a *let* expression or using *where*.