1. 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> :t numbers
:t numbers
numbers :: [Int] -> Int
*Main> numbers [1..4]
1234
```

Write `numbers` using a tail-recursive helper function which should be defined inside of a `let` expression or using `where`.

2. The function `combinations3` takes a list as its argument and returns a list of length three lists representing all possible subsets of size three. For example,

```
*Main> :t combinations3
:t combinations3
combinations3 :: (Ord a) => [a] -> [[a]]
*Main> combinations3 "ABCDE"
["ABC","ABD","ABE","ACD","ACE","ADE","BCD","BCE","BDE","CDE"]
```

Write `combinations3` using a list-comprehension.

3. The function `runLengthEncode` takes a list of as its argument and returns a list of pairs of values and run lengths. See

http://en.wikipedia.org/wiki/Run_length_encoding

For example,

```
*Main> runLengthEncode [4,2,2,1,1,1,4,4,4,4]
[(4,1),(2,2),(1,4),(4,4)]
*Main> runLengthEncode "foo"
[('f',1),('o',2)]
```

Write `runLengthEncode`. Hint: Divide and conquer. Ask yourself: What helper functions would make this problem trivial and then write those. Make use of higher-order functions when appropriate. This is the key to modular design and you will complete your homework faster as a bonus.
4. The function \textit{runLengthDecode} takes a list of pairs of values and run lengths and returns a list of values. For example,

\begin{verbatim}
*Main> runLengthDecode [(4,1),(2,2),(1,4),(4,4)]
[4,2,2,1,1,1,1,4,4,4,4]
\end{verbatim}

Write \textit{runLengthDecode}.

5. The function \textit{church} takes an integer \textit{n} as its argument and returns a function which composes any unary function \textit{n} times. For example,

\begin{verbatim}
*Main> :t church
church :: (Num t) => t -> (c -> c) -> c -> c
*Main> (church 4) tail "ABCDEFGH"
"EFGH"
\end{verbatim}

Write \textit{church}.

6. The function \textit{rotate} takes an integer \textit{n} as its argument and returns a function which does a periodic left shift of a list by \textit{n}. For example,

\begin{verbatim}
*Main> :t rotate
rotate :: (Num t) => t -> [t1] -> [t1]
*Main> rotate 4 "ABCDEF"
"EFABCD"
\end{verbatim}

Hint: Use \textit{church}.

7. The Goldbach conjecture states that any even number greater than two can be written as the sum of two prime numbers. Using list comprehensions, write a function \textit{goldbach}, which when given an even number \textit{n}, returns a list of all pairs of primes which sum to \textit{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,

\begin{verbatim}
*Main> goldbach 6
[(3,3)]
\end{verbatim}

8. The function \textit{increasing} takes a list of enumerable elements as its argument and returns \textit{True} if the list is sorted in increasing order and \textit{False} otherwise.
*Main> increasing "ABCD"
True
*Main> increasing [100,99..1]
False

Write increasing.

9. The function `combinations` takes an integer $k$ and a list of elements of typeclass `Ord` as its arguments and returns a list of length $k$ lists representing all possible subsets of size $k$. For example,

*Main> :t combinations
`combinations :: (Ord a) => Int -> [a] -> [[a]]`
*Main> combinations 3 "ABCDE"
["ABC","ABD","ABE","ACD","ACE","ADE","BCD","BCE","BDE","CDE"]

Write `combinations`. Hint: Don’t use list-comprehensions. Do use `increasing`. Write `combinations1`. Use `combinations1` and `map` to write `combinations2`. Now use `combinations2` and `map` to write `combinations3`. Abstract the pattern.