Question 1 [12 marks]

Consider the program

```
tw c [] = []
tw c (h:t)
    | c h = h : tw c t
    | otherwise = []
```

(a) [3 marks] What is the inferred type of \( \text{tw} \)?

(b) [4 marks] What is the value of \( \text{tw} (>4) \ [8,9,5,3,2,7,11] \)?

(c) [4 marks] Consider what would happen if the first line in the definition of \( \text{tw} \) was missing. Does it give a type error, a runtime error (in some circumstances), or is there no problem? If it gives a type error, explain what the error is. If it sometimes gives a runtime error, give an example that would induce that error.

Question 2 [10 marks]

Consider the function

```
insertAt n e lst
```

that returns a list with \( e \) inserted at position \( n \) of list \( lst \). The positions of the other elements is shifted to include \( e \). If \( n \) is not a legal position, \( e \) is put at the end of the list.

It should have the following behaviour:

```
*Main> insertAt 1 'X' "abcd"
"aXbcd"
*Main> insertAt 9 8 [0,1,2,3]
[0,1,2,3,8]
*Main> insertAt 0 8 [0,1,2,3]
[8,0,1,2,3]
*Main> insertAt 2.3 8 [0,1,2,3]
[0,1,2,3,8]
*Main> insertAt (-2) 'd' "Foo"
"Food"
```
(a) [4 marks] Suppose `insertAt` is declared to have the type

\[ \text{insertAt :: Int -> t -> [t] -> [t]} \]

Explain in English (suitable for a CPSC 312 student who has just been introduced to Haskell) what this means.

(b) [6 marks] Fill in the missing values in the following recursive definition of `insertAt`. This should use pattern matching as much as possible. You can use : and [] and the arithmetic functions +, -, * but no other Haskell functions.

\[
\begin{align*}
\text{insertAt 0 ch lst} &= \underline{\quad} \\
\text{insertAt n ch (h:t)} &= \underline{\quad} \text{insertAt} \quad \underline{\quad} \text{ch} \quad \underline{\quad} \\
\text{insertAt n ch} \quad \underline{\quad} &= \underline{\quad}
\end{align*}
\]

**Question 3 [10 marks]**

In this question you can use:

- list comprehensions [\( f \ x \mid x \leftarrow \ \text{list, cond} \ x \) ]
- \( \text{foldr} \ \oplus \ v \ [a_1, a_2, \ldots, a_n] = a_1 \oplus (a_2 \oplus (\ldots \oplus (a_n \oplus v))) \)
- \( \text{foldl} \ \oplus \ v \ [a_1, a_2, \ldots, a_n] = (((v \oplus a_1) \oplus a_2) \oplus \ldots) \oplus a_n \)
- \( (\ \lambda \ x \rightarrow \ \text{exp}) \) is an anonymous function, where \( \lambda \) is pronounced lambda
- if \( c \) then \( e_1 \) else \( e_2 \) returns the value of \( e_1 \) if \( c \) is true or the value of \( e_2 \) if \( c \) is false

(a) [6 marks] Implement \( nw \) (as defined earlier) in terms of either list comprehensions, foldr or foldl. You may use if-then-else, lambda (\( \lambda \)) and : but no other built-in functions.

(b) [4 marks] Given the definition:

\[
\begin{align*}
\text{dcr c lst} &= \text{foldl} (\ \lambda \ r \ e \rightarrow \ c \ e : e : r) \ [\] \ \text{lst} \\
\text{toUpper :: Char -> Char} \\
\text{toUpper c} &= \text{toEnum} (\text{fromEnum} c - \text{fromEnum} \ 'a' + \text{fromEnum} \ 'A')
\end{align*}
\]

(Note that `toUpper` converts a lower case letter to upper case.) What is the value of \( \text{dcr toUpper "abc"} \)?

[Note that this does not give an error. You do not need to show your reasoning if you get the correct answer, but if you want partial marks, you need to explain your answer.]

**Question 4 [10 marks]**

(a) [3 marks] How does lazy evaluation differ from call-by-name? (We don’t want to know what they are, just how they differ.)
(b) [4 marks] The function max has the type declaration:

```
max :: Ord a => a -> a -> a
```

Explain what `Ord a` means.

(c) [4 marks] Haskell append (`++`) has type

```
(++) :: [a] -> [a] -> [a]
```

What is the result of typing into ghci:

```
[1,2,3] ++ [[4,5,6],[7,8,9]]
```

Explain why this is the result.