COMP 212 Fall 2022
Lab 7

The goal for this lab is to make you more familiar with higher-order functions in SML. Recall `map` from lecture:

```
map : ('a -> 'b) * 'a list -> 'b list
```

`map (f, L)` applies `f` to each element of `L`, returning a list of the results; that is, `map` $(\texttt{f}, [\texttt{v}_1, ..., \texttt{v}_n])$ computes $[\texttt{f v}_1, ..., \texttt{f v}_n]$

1 Filter

Consider the following two functions:

```
fun evens (l : int list) : int list =
    case l of
        [] => []
      | x :: xs => ( case evenP x of
                       true => x :: evens xs
                     | false => evens xs )

fun keepUpper (l : char list) : char list =
    case l of
        [] => []
      | x :: xs => (case Char.isUpper x of
                       true => x :: keepUpper xs
                     | false => keepUpper xs)
val [#"B"] = keepUpper [#"a", #"B"]
```

For the second function: characters are represented by the SML type `char`. Character literals are written `#"a"`, `#"A"`, etc. (like a string, but with a `#` in front). The function `Char.isUpper` determines whether a character is an upper-case letter.

The pattern here is "keep all the elements of the list that satisfy some predicate."

Task 1.1 Define a function

```
fun filter (p : 'a -> bool, l : 'a list) : 'a list = ...
```

that abstracts over this pattern. The function `p` represents the predicate.

Task 1.2 Re-define `evens` and `keepUpper` by calling `filter` with the appropriate predicate.

Task 1.3 For this lab, you do not need to write tests in your sml file, but should test directly in smlnj. For example, evaluate `evens [1,2,3,4]` and see that the result is correct.

Task 1.4 On Homework 4, we hadn't introduced higher-order functions yet, so for `quicksort_l` (quicksorting lists) we had you define a first-order but less-general variant of filter. Rewrite `quicksort_l` to use the `filter` function you defined above.

Have us check your answer before proceeding!

2 Map and filter

You are writing an eligibility test for your new social media app Parallelogram. Due to FTC restrictions, only people 13 years of age or older are eligible to register for an account. The registration form asks people for their date of birth, from which their age can be calculated. Write a function

```
eligible : (string * int) list -> (string * int) list
```

that is given a list of pairs `(person, birth year)`, and returns a list of pairs `(person, age)`, where `age` is the age—in years, as of 12:00am on January 1, 2022—of each `person` in the original list who was 13 years or older on that date. For example:

```
eligible [("Sri",1992),("Dan",1982),("CB",2004),("SJ",2019)]
== [("Sri",29),("Dan",39),("CB",17)]
```

You may not define this function recursively. Write it using `map` and `filter`.

Have us check your answer before proceeding!

3 All

Consider the following two functions:

```
fun allPos (l : int list) : bool =
    case l of
        [] => true
      | x :: xs => (x > 0) andalso allPos xs

fun allOfLength (len : int, l : 'a list list) : bool =
     case l of
           [] => true
         | x :: xs => ((List.length x = len) andalso allOfLength(len, xs))
```

Task 3.1 Write a higher-order function `all` that can be used to define `allPos` and `allOfLength`, and then define these two functions in terms of it.

Task 3.2 Using the above, write a function

```
square : 'a list list -> bool
```

that returns true iff the input list of lists is square. For example,

```
square [[1,2],[3,4]] == true
square [[1,2],[3]] == false
square [[1,2],[3,4],[5,6]] == false
```

Have us check your answer before proceeding!

4 Reduce

Consider the following two functions:

```
fun sum (l : int list) : int =
   case l of
        [] => 0
      | x :: xs => x + (sum xs)
fun join (l : string list) : string =
    case l of
        [] => ""
      | x :: xs => x ^ join xs
```

The pattern is "give some answer for the empty list, and for a cons, somehow combine the first element with the recursive call on the rest of the list."

Task 4.1 Write a higher-order function

```
fun reduce(c : 'a * 'a -> 'a, n : 'a, l : 'a list) : 'a = ...
```

where the function `c` describes how to combine the first element with the recursive call, and `n` is the answer for the empty list.

Task 4.2 Define `sum` and `join` as instances of `reduce`.

Have us check your answer before proceeding!

5 Map and reduce

We have provided

```
lines : string -> string list
words : string -> string list
wordcount : string -> int
```

lines divides a string into lines (delimited by the newline character). **words** divides a string into words (delimited by spaces or newlines). **wordcount** divides a string into words and counts how many there are.

Task 5.1 Define functions

```
(* computes the number of words in the longest line in a document *)
fun longestline (s : string) : int = ...
```

These functions should not be defined recursively, and you should not use the `length` function (practice using `map` and `reduce` instead).

For example, given the string

```
for life's not a paragraph
And death i think is no parenthesis
```

`wordcount` should return 12, and `longestline` should return `7`. Note that you can type in this document using `\n` for newlines:

```
"for life's not a paragraph\nAnd death i think is no parenthesis\n"
```

Have us check your answer!