COMP 212 Fall 2024
Lab 8

The code you download for this lab is in a bundle called a TAR file. To unzip it, save the TAR file to your `comp212` directory and then

```
% cd comp212
% tar -xf <tarfile>
```

Unpacking the sequence library should create a directory named `src`, and unpacking the lab code should create a directory named `lab08-handout`. You should unpack them so that two directories should be next to each other in your `comp212` directory. `cd` into `lab08-handout` and then start smlnj from there.

Note: on some platforms (e.g. MacOS), opening/double-clicking a TAR file will automatically unpackage it, in which case you can move the directories that opening the two downloads creates into your `comp212` folder.

1 Sequences Cheat-Sheet

For your convenience a brief description of some of the functions on sequences is given here. See the lecture notes for more details.

- `Seq.map : ('a -> 'b) * 'a Seq.seq -> 'b Seq.seq`, which takes a function and a sequence and returns a sequence whose elements are the result of applying the given function to the corresponding element in the given sequence.
- `Seq.reduce : (('a * 'a) -> 'a) * 'a * 'a Seq.seq -> 'a`, which combines all the elements of a sequence using a particular function and base case.
- `Seq.filter : ('a -> bool) * 'a Seq.seq -> 'a Seq.seq`, which computes the sequence that contains only those elements satisfying the given predicate.
- `Seq.length : 'a Seq.seq -> int`, which returns the number of elements in the sequence.
- `Seq.nth : int * 'a Seq.seq -> 'a`, which returns the element of the given sequence at the indicated index, assuming it is in bounds.

- `Seq.tabulate : (int -> 'a) * int -> 'a Seq.seq`, which computes a sequence of the given length such that the value of each element of the sequence is the result of applying the function to its index.
- `Seq.empty : unit -> 'a Seq.seq`, which forms an empty sequence.
- `Seq.cons : 'a * 'a Seq.seq -> 'a Seq.seq`, which inserts the given element at the beginning of the sequence.
- `Seq.append : 'a Seq.seq * 'a Seq.seq -> 'a Seq.seq`, which combines two sequences by inserting the elements of the second sequence after the elements of the first sequence.
- `Seq.zip : 'a Seq.seq * 'b Seq.seq -> ('a * 'b) Seq.seq`, which combines two sequences into a sequence of pairs, dropping any extra elements in the longer sequence if the two have different lengths.
- `Seq.drop : int * 'a Seq.seq -> 'a Seq.seq`, where `Seq.drop k s` removes the first `k` elements from `s`, or raises `Range` if there are not enough elements to drop
- `Seq.take : int * 'a Seq.seq -> 'a Seq.seq`, where `Seq.take k s` returns the sequence consisting of the first `k` elements from `s`, or raises `Range` if there are not enough elements to take.

Task 1.1 Rewrite your solution to the "eligible for signup" problem from last lab so that it works for sequences instead of lists.

```
fun eligible (l : (string * int) Seq.seq) : (string * int) Seq.seq =
    ...
```

Have us check your work before proceeding!

2 Exists

Recall from last week's lecture the function `exists : ('a -> bool) * 'a list -> bool`, which determines whether an element of the list satisfies the given predicate. You will write an analogous function for sequences:

Task 2.1 Write the function

```
seqExists : ('a -> bool) * 'a Seq.seq -> bool
```

to determine if the sequence has an element that satisfies the given predicate.

You can use

```
Seq.fromlist : 'a list -> 'a Seq.seq
Seq.tolist : 'a Seq.seq -> 'a list
```

to write tests, but you should never use these in homework problems (except to test), because they will usually ruin the span, defeating the point of writing code for sequences instead of lists.

Have us check your work before proceeding!

3 Tabulate Puzzles

The following functions ask you to become familiar with `Seq.tabulate`, `Seq.length`, and `Seq.nth`.

`Seq.tabulate (f,n)` behaves like `Seq.map(f,`$\langle 0, 1, 2, 3, \ldots, n-1 \rangle$`)`. I.e. it computes the sequence $\langle$ `f 0, f 1, f 2, ..., f(n-1)` $\rangle$. It has the same work and span as that use of `Seq.map`.

3.1 Increasing

Using `tabulate`, write a function

```
fun increasing (n : int) : 'a Seq.seq = ...
```

that returns the sequence `<0,1,2,...,n-1>`

3.2 Reverse

Write a function

```
fun reverse (s1 : 'a Seq.seq) : 'a Seq.seq = ...
```

that reverses the order of elements in its input sequence.
On a sequences of length n, your solution should have $O(n)$ work and $O(1)$ span.

3.3 Append

There is a function `Seq.append` that appends two sequences. Suppose there wasn't, and write

```
fun myAppend (s1 : 'a Seq.seq, s2 : 'a Seq.seq) : 'a Seq.seq = ...
```

On sequences of length n and m, your solution should have $O(n + m)$ work and $O(1)$ span.

3.4 Transpose

Write a function `tranpose` that transposes a sequence of sequences. For example,

```
  transpose (< <1,2,3>,
               <4,5,6>>)
  =
<<1,4>,
 <2,5>,
 <3,6>>
```

Write

```
fun transpose (s : 'a Seq.seq Seq.seq) : 'a Seq.seq Seq.seq = ...
```

that transposes a sequence of sequences. You may assume that s is rectangular, with dimensions $m \times n$, where $m, n > 0$. Your solution should have $O(m \times n)$ work and $O(1)$ span.

Have us check your code before proceeding!

4 Stocks

Task 4.1 Translate the `bestGain` function (and all necessary helper functions) from the Higher-order Functions II lecture from lists to sequences. Analyze the work and span.