COMP 212 Fall 2024
Lab 3

The goal for the third lab is to make you more comfortable writing functions in SML that operate on lists, and doing proofs. Remember to follow the methodology for writing functions—specifications and tests are part of your code!

1 Evens

Task 1.1 Write a function

```
evens : int list -> int list
```

that filters out all odd elements of a list without changing the order. For example,

$$\texttt{evens}[0,0,4] \cong [0,0,4]$$
$$\texttt{evens}[] \cong []$$
$$\texttt{evens}[0,0,4,9,3,2] \cong [0,0,4,2]$$

You should use the function `evenP` that we provided from last lab to determine if a number is even.

Task 1.2 Recall the function `length` from class:

```
fun length (l : int list) : int =
    case l of
        [] => 0
      | x :: xs => 1 + length xs
```

Prove the following theorem by structural induction on `l`:

Theorem 1. *For all lists* `l`, `length (evens l)` $\leq$ `length l`.

Task 1.3 Give expressions `e` and `es` such that both sides are well-typed and the following equivalence is **true**.

$$\texttt{length(e :: es)} \cong \texttt{1 + length es}$$

Task 1.4 Give expressions `e` and `es` such that both sides are well-typed and the following equivalence is **false**.

$$\texttt{length(e :: es)} \cong \texttt{1 + length es}$$

Task 1.5 At which point in your proof of Theorem 1 did you use an equivalence like this? Why is that use OK?

Have the course staff check your work before proceeding.

2 Append and Reverse

The "cons" function `::` adds one new element to a list. What if you want to *append* a whole list onto the front of another? Appending a list `l1` to another list `l2` evaluates to a list that contains all of the elements of `l1` in the same order, followed by all of the elements of `l2`, also in the same order. For example,

```
append([1,2,3], [5,13,5]) ==> [1,2,3,5,13,5]
```

A simple implementation of `append` takes elements off of `l1` one at a time, consing them onto the result of appending the rest of the list to `l2`.

Task 2.1 Write the function

```
append : int list * int list -> int list
```

that behaves according to the specification given above.

Task 2.2 Write a function

```
reverse : int list -> int list
```

such that `reverse l` has the same elements as `l` but in the opposite order.

Task 2.3 If the input list `l` has n elements, about how many steps does `reverse l` take?